Application Design, Microservices, Continuous Development

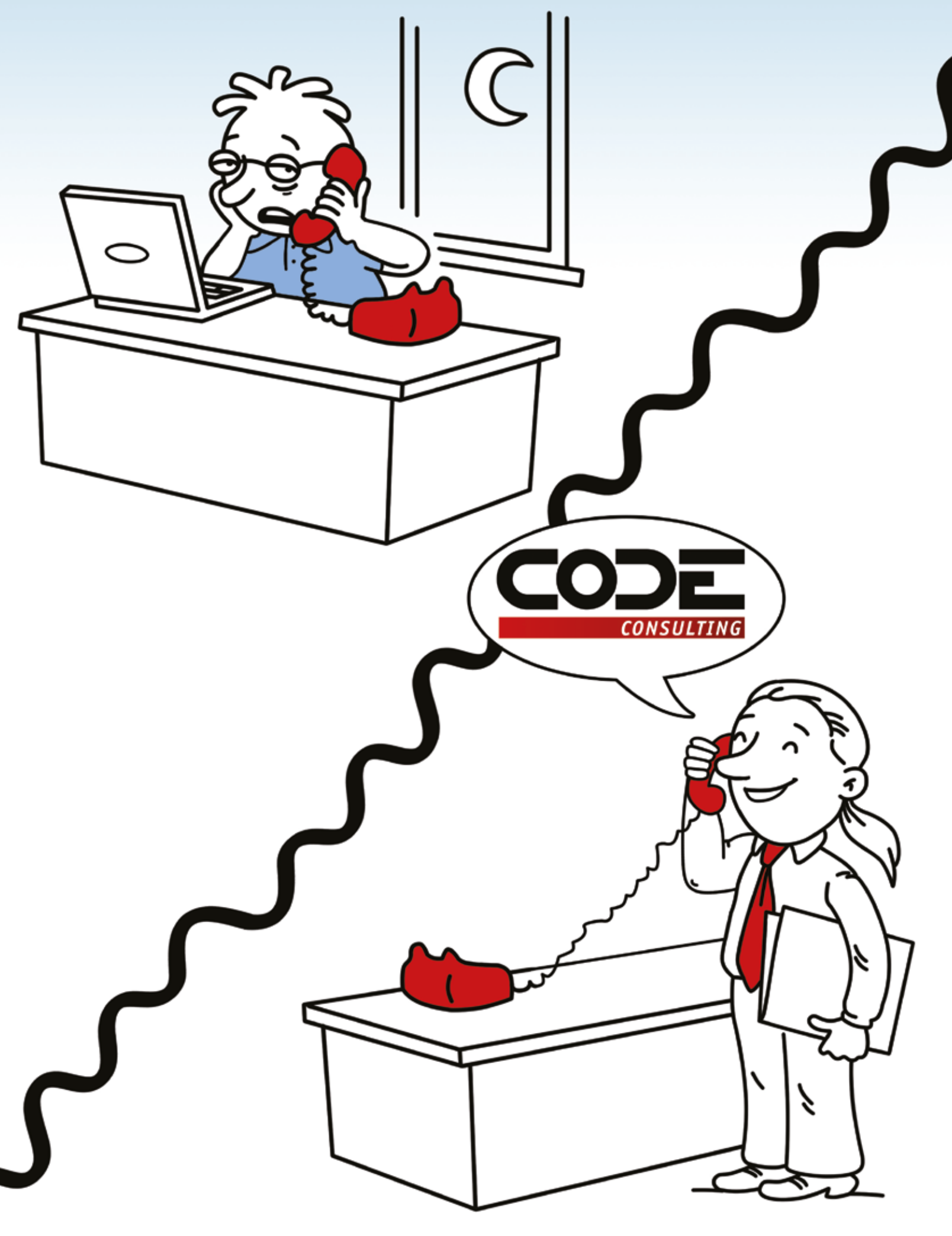
CODE

JAN
FEB
2018

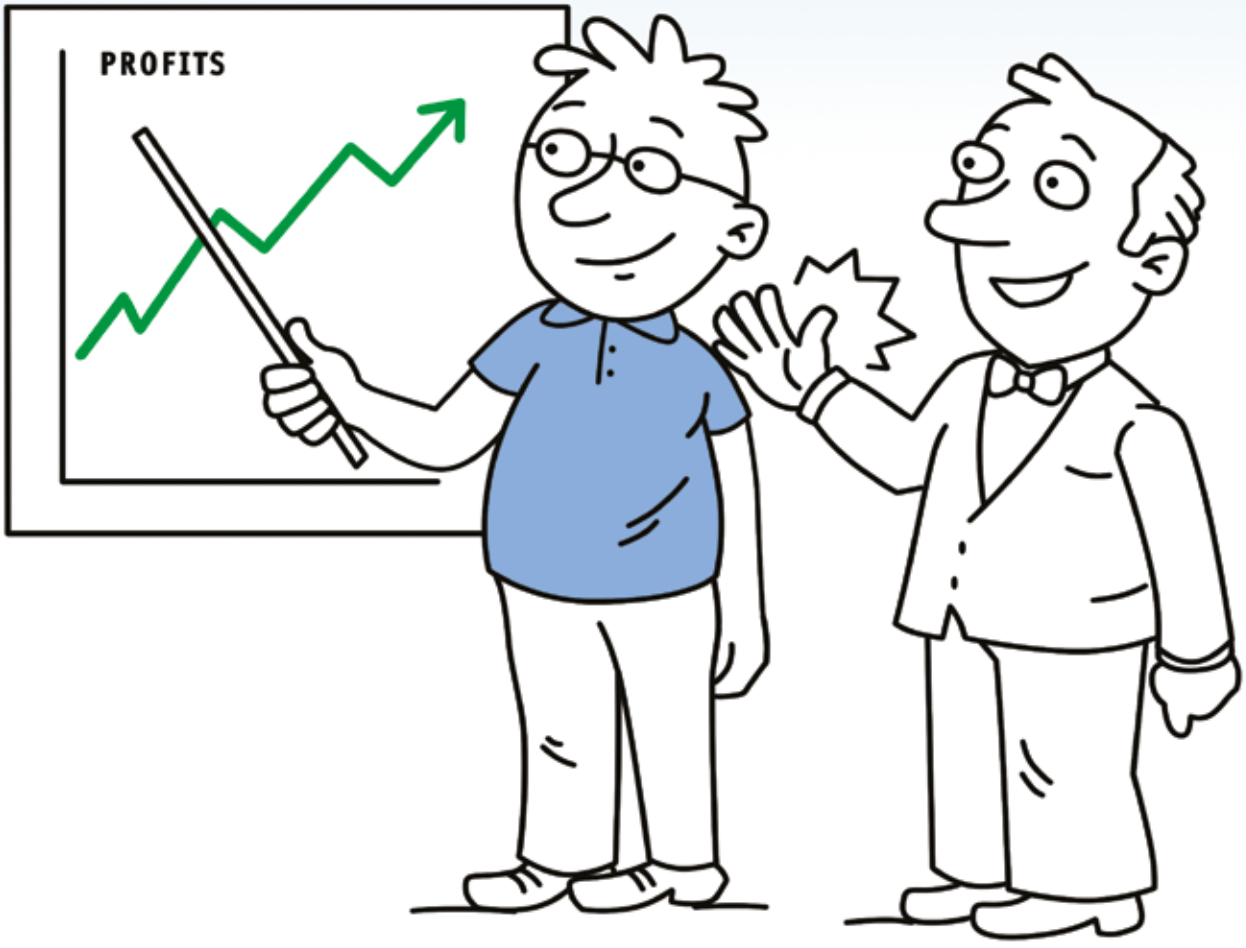
codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 5.95 Can $ 8.95

A Good Idea is Just the Start

02>
0 74470 56130 9


CODE
CONSULTING

CODE Consulting Will Make You a Hero!



Lacking technical knowledge or the resources to keep your development project moving forward? Pulling too many all-nighters to meet impending deadlines? CODE Consulting has top-tier programmers available to fill in the technical and manpower gaps to make your team successful! With in-depth experience in .NET, Web, Azure, Mobile and more, CODE Consulting can get your software project back on track.

Contact us today for a free 1-hour consultation to see how we can help you succeed.

Helping Companies Build Better Software Since 1993

www.codemag.com/techhelp
832-717-4445 ext. 9 ● info@codemag.com

CODE CONSULTING

Features

Columns

Departments

US subscriptions are US $29.99 for one year. Subscriptions outside the US pay US $44.99. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Bill me option is available only for US subscriptions. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com.

Subscribe online at codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 300, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 300, Spring, TX 77379 U.S.A.

Canadian Subscriptions: Canada Post Agreement Number 7178957. Send change address information and blocks of undeliverable copies to IBC, 7485 Bath Road, Mississauga, ON L4T 4C1, Canada.

Focusing on What Matters

A few months ago, I picked up a book at good old Barnes and Noble with the title, "The Subtle Art of Not Giving a F***" Before I made this purchase, I spent a few minutes perusing its pages to make sure it wasn't some type of nihilistic manifesto. Lucky for me, it wasn't. The book is meant to help the reader understand how to focus on what's important and how to not give a frack about things that don't really matter.

It's a rare event when I can apply ideas from a book into my life directly. This book was one of those times. My silly story-of-the-month is how I used this book elevate my driving experience. And no, I am not talking about the "BMW Ultimate Driving Experience;" I'm talking about reducing my tendency to road rage when I drive. Every time I feel myself getting frustrated, I remember the title of the book and its lessons about caring what matters. Which leads me to the subject of this editorial. As developers, we need to pay attention to the things that matter.

Work-Life Balance

There's a beauty and a curse to what we do for a living. The beauty of what we do for a living is that we can do our work anywhere and any time we want. It's the ability to work around-the-clock that can cause problems. When I started my consulting company, I spent nearly every waking hour writing code. This, along with writing articles and books, and speaking at conferences, took a toll on my personal life, ultimately resulting in a divorce. I can tell you right now, this pattern has drastically changed. I try to work a "normal" forty-hour week, make sure I'm home on weekends when I travel, and have pretty much curtailed speaking engagements. Trying to keep a normal schedule has allowed me to focus on priorities of true importance, like family, friends, and hobbies. I also notice that I have more time to scratch that learning itch we developers all seem to have.

Continuous Improvement

It seems that Continuous is the big word these days: Continuous Deployment, Continuous Integration, etc. Let's add a new one to the mix: Continuous Improvement. We all need to focus on improving our skills as developers. For this item, let's focus on the quality of our work. As developers, we need to be constantly trying to improve the quality of our code.

There are numerous ways to make our code better. The first that comes to mind is to do code reviews with other developers of equal or better skill. Walk them through sections of your code and see if they have any ideas about how to improve. On a recent project, I spent time reviewing some critical code with my friend (and frequent CODE Magazine contributor) Rick Strahl. It was refreshing to have him confirm some of the better ideas and recommend a simple set of changes that improved the quality and stability our application.

Another technique for improving code is to review other people's work. The beauty of Open Source Software is the ability to examine first-hand the work of other skilled developers. Just yesterday, I noticed on my Twitter feed that there was a new Postgres module for nServiceBus. I'm currently adapting some of my code to Postgres and loved having the opportunity to review someone else's work on this platform while trying to improve my code. There are other ways to improve your work: read articles, watch videos, attend conferences. Find what works for you and improve!

Always Be Learning

There is only one constant in this business and that's change. As you have probably surmised from reading my other editorials, it's important to at least have a minimal understanding of the new ideas and technologies currently in use or on the distant horizon.

How can you possibly stay current? Staying current with today's everchanging landscape is much like eating an elephant. How do you eat an elephant? One bite at a time! (For the record, I don't condone the eating of elephants. <g>) Learning new tech can be done one bite or technology at a time.

I read a funny Tweet the other day. It went something like this. **Q: How are you going to learn the newest C# language changes? A: I'm gonna wait for the next version of ReSharper to point out changes in my code and follow the squiggly green underline.**

ReSharper is a tool that many developers to improve their coding process. One of its features is to make code recommendations. These recommendations generally align with new language features, and presto: the green underline tutorial.

Another learning technique is sitting in your hands right now. One of the many goals of CODE Magazine is to help you learn. In this issue, we spend time exploring Microservices and Continuous integration. Future issues will have articles on Docker and using "R" for data analysis.

One of my learning techniques is to buy and read the latest "<insert subject here> For Dummies" books. These books do a decent job of covering the salient points for many technologies. I'm not going to kid you, keeping up on tech is difficult and gets more difficult every day. But you've gotta keep eating away at the elephant.

Focusing on what matters is a skill that all human beings need to focus on. Spending time on what matters versus what doesn't is a sure way to improve your life. I'd love to hear from you on what matters in your life as a developer. Maybe it can be the subject of an article in CODE Magazine.

Rod Paddock
CODE

Data Quality Made Easy.
Your Data, Your Way.

Melissa provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email, phone, and more.

Our **data quality** solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools, integrations and plugins for the **Microsoft** and **Oracle Product Ecosystems.**

Microsoft SQL Server Integration Services · Microsoft Dynamics CRM · Microsoft .NET · ORACLE Forms · Java · ORACLE PL/SQL

Start Your Free Trial

www.melissa.com/code

Melissa Data is now Melissa.
See What's New at www.Melissa.com

1-800-MELISSA

melissa
GLOBAL INTELLIGENCE

Configuration Settings for Angular Applications

Just like in .NET applications, you might want to have global configuration settings in your Angular applications that you can access from any component or service class. There are many approaches you can take for retrieving these global settings; I'm going to use a service that can be injected into any class. I think the flexibility of using a service is an ideal method for providing application-wide settings to any class that needs them.

Paul D. Sheriff
www.fairwaytech.com

Paul D. Sheriff is a Business Solutions Architect with Fairway Technologies, Inc. Fairway Technologies is a premier provider of expert technology consulting and software development services, helping leading firms convert requirements into top-quality results. Paul is also a Pluralsight author. Check out his videos at http://www.pluralsight.com/author/paul-sheriff.

This article illustrates how to create an Angular service to read and modify configuration settings. I'm going to use Visual Studio Code and Visual Studio 2017, along with C# and Angular 4. You're going to learn to create a class to hold configuration settings. These settings can be retrieved from a JSON file or from a Web API call. Once you've retrieved the settings, you can store them into local storage. Placing them into local storage allows your user to modify those settings.

Create a New Angular Project

If you're new to Angular, you are going to need the following tools installed on your computer.

- Node.js
- TypeScript
- Angular
- Angular CLI
- An editor, such as Visual Studio or Visual Studio Code

In this article, I'm going to use Visual Studio Code and the Angular CLI to create the sample project. I'll assume that you have the tools listed above installed on your computer. If you haven't already installed them, please do so now, so that you can follow along with this article.

Open a command prompt on your computer and navigate to a folder where you typically create your development projects. For example, on my computer, I usually go to my D drive and go to a folder named \Samples. So, from the command prompt I enter the following commands.

```
D:
cd \Samples
```

Using the Angular CLI, I create a new project under the Samples folder using the following command.

```
ng new ConfigSample
```

Executing the **ng new** command creates a new folder named \LogSample. The **new** command creates the files needed to develop an Angular application. While you are still in the command prompt, navigate to the new folder using the following:

```
cd ConfigSample
```

Start Visual Studio Code by typing **code** followed by a space and a period. This tells Code to start and open the current folder.

```
code .
```

When you are in Visual Studio Code, click on the View menu and choose the Integrated Terminal... menu option. A small window opens at the bottom of Visual Studio Code. Into this command window, type the following command to start up the **lite-server**.

```
npm start
```

The lite-server is a small server that runs on your local computer on port 4200 by default. Start your browser of choice and navigate to the http://localhost:4200 and you should see a Web page that looks like **Figure 1**.

Create the Application Settings Service Classes

You're going to create a class to be used across many other component and service classes in your Angular

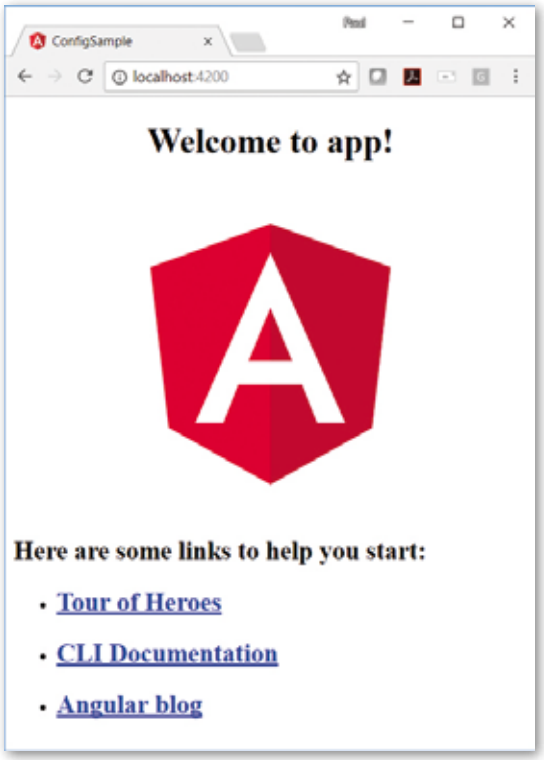


Figure 1: The start page created using Angular CLI

application. Place this new class into a \shared folder located under \src\app. Create the \shared folder now. Right mouse-click on the \shared folder and add a new file named appsettings.ts. Create a class in this new file with properties to hold the values you wish to use in your Angular application. For example, if you have a Product form that's used to add new products to a database, you might want to provide some default values for a couple of the fields. The following code includes a class named AppSettings with two properties; defaultUrl and defaultPrice.

```
export class AppSettings {
  defaultUrl: string =
        "http://www.fairwaytech.com"
  defaultPrice: number = 1
}
```

An AppSettingsService Class

It's now time to create an Angular service class to return an instance of the AppSettings class. Add a new TypeScript file under the \shared folder named appsettings.service.ts. Add the code shown in the following code snippet:

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';
import { AppSettings } from "./appsettings";

@Injectable()
export class AppSettingsService {
  getSettings(): Observable<AppSettings> {
    let settings = new AppSettings();

    return Observable.of<AppSettings>(settings);
  }
}
```

The preceding code is a standard representation of an Angular service class. In the getSettings() method, create a new instance of the AppSettings class and return that object from this service. The main reason you create a service here is to provide the flexibility to change the implementation of how you retrieve the settings later. For example, you might choose to read the settings from a JSON file, or make a Web API call to get the settings. Any method that calls this service always makes the same call regardless of where those settings are stored. The calling methods don't know if the implementation changes; they still receive the same settings class.

The Consumer of the Settings Service

Now that you have a simple application settings service class created that can return an instance of the AppSettings class, test it by creating a consumer. You can build a unit test to call the service, but let's use something a little more real-world. In this section of the article, build a Product class, a Product HTML page, and a Product component class to add a new product. You're going to set default values from the AppSettings into the appropriate properties of a Product object, then display those properties on the HTML page.

Add a new folder named \product under the \src\app folder. Add a new file in this folder named product.ts. Add a few properties to this class to represent the typical properties associated with a product, as shown in the following code snippet.

```
export class Product {
  productId: number;
  productName: string;
  introductionDate: Date;
  price: number;
  url: string;
}
```

Create an HTML page to which you can bind the properties of the Product class. Add another new file within the \product folder and name it product-detail.component.html. Add the code shown in **Listing 1**.

Create a component class to work with the HTML page you just created. Add a new file within the \product folder named product-detail.component.ts. Add the appropriate import statements to create the component class. Import the Component class and the OnInit interface. You also need the Product class you just created. Finally, import the AppSettings and AppSettingsService classes in this new component class.

Listing 1: Add an HTML page to display the Product properties.

```
<div *ngIf="product">
  <table>
    <tbody>
      <tr>
        <td>Product Name:</td>
        <td>
          <input [(ngModel)]="product.productName" />
        </td>
      </tr>
      <tr>
        <td>Introduction Date:</td>
        <td>
          <input [(ngModel)]=
                   "product.introductionDate" />
        </td>
      </tr>
      <tr>
        <td>Price:</td>
        <td>
          <input [(ngModel)]="product.price" />
        </td>
      </tr>
      <tr>
        <td>URL:</td>
        <td>
          <input [(ngModel)]="product.url" />
        </td>
      </tr>
    </tbody>
  </table>
  <div>
    <button (click)="saveProduct()">
      Save
    </button>
  </div>
</div>
```

Sample Code

You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select PDSA Articles, and then select "CODE Magazine: Configuration Settings for Angular Applications" from the drop-down list. It's also available on the CODE Magazine page associated with this article.

```
import { Component, OnInit }
  from "@angular/core";
import { Product }
  from './product';
import { AppSettings }
  from '../shared/appsettings';
import { AppSettingsService }
  from '../shared/appsettings.service';
```

Each component class you create must be decorated with the @Component() decorator function. Pass in an object to this function to supply some meta-data about how this component is used. The object passed in creates a selector property set to the value **product-detail**. This value is used as a new element on an HTML page to invoke this component class. You also specify the location of the HTML page you just created by using the templateUrl property.

```
@Component({
  selector: "product-detail",
  templateUrl:
    "./product-detail.component.html"
})
```

After the @Component decorator, export a class named ProductDetailComponent and make sure that it implements the OnInit interface. The ngOnInit() function is called after the ProductDetailComponent class is instantiated. Just write an empty ngOnInit() method for now, and a saveProduct() method too. The saveProduct() method is called from the button in the HTML you created previously.

```
export class ProductDetailComponent
             implements OnInit {
  ngOnInit(): void {
  }

  saveProduct(): void {
  }
}
```

Next, add a constructor to the ProductDetailComponent class. Into this constructor, add a private variable named appSettingsService that's of the type AppSettingsService. Adding this code in the constructor tells Angular to inject an instance of the service into this class when it creates an instance of this component.

```
constructor(private appSettingsService:
             AppSettingsService) {
}
```

In the HTML created earlier, you referenced a **product** object. Create that property named **product** now. Also, create a property to assign to the global settings retrieved from the AppSettingsService.

```
product: Product;
settings: AppSettings;
```

Write the code within the ngOnInit() method to call the getSettings() method on the appSettingsService object. The subscribe method calls getSettings() and receives the settings back from the service. Subscribe has three parameters: a success function, an error function, and a completed function. In the success function, set the result returned to the settings property in this class. Because all you're doing is returning a new instance of a class, you can safely ignore the error function. In the completed function, create a new instance of the Product object and assign the price and URL properties to the defaults returned from the settings object.

```
ngOnInit(): void {
  this.appSettingsService.getSettings()
    .subscribe(settings =>
          this.settings = settings,
      () => null,
      () => {
        this.product = new Product();
        this.product.price =
          this.settings.defaultPrice;
        this.product.url =
          this.settings.defaultUrl;
      });
}
```

Update AppModule

For your components and services to work within your application, you need to inform Angular of the service and the component. Open the app.module.ts file and add the following import statements.

```
import { FormsModule }
  from '@angular/forms';
import { ProductDetailComponent }
  from "./product/product-detail.component";
import { AppSettingsService }
  from "./shared/appsettings.service";
```

After you have the appropriate classes imported, modify the metadata properties of the @NgModule decorator function. Add the FormsModule class to the imports property. Add the ProductDetailComponent to the declarations property. Modify the providers property and add the AppSettingsService to the array. The @NgModule decorator function should now look like the following code snippet.

```
@NgModule({
  declarations: [AppComponent,
                 ProductDetailComponent],
  imports: [BrowserModule, FormsModule],
  providers: [AppSettingsService],
  bootstrap: [AppComponent]
})
```

Update AppComponent

Open the app.component.html file and delete all the code in this HTML file. Replace the code with the following.

```
<h1>Configuration Settings Sample</h1>
<div>
  <product-detail></product-detail>
</div>
```

Save all the changes to all your open files. Switch your open browser on the URL http://localhost:4200 and you should see your product page appear. Because the **product** property is bound to the different fields on the HTML

Listing 2: Retrieve application settings from a JSON file.

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/observable/of;
import 'rxjs/add/observable/catch;

import { AppSettings }
  from "./appsettings";

const SETTINGS_LOCATION =
         "assets/appsettings.json";

@Injectable()
export class AppSettingsService {
  constructor(private http: Http) {
  }

  getSettings(): Observable<AppSettings> {
    return this.http.get(SETTINGS_LOCATION)
      .map(response => response.json() || {})
      .catch(this.handleErrors);
  }

  private handleErrors(error: any):
   Observable<AppSettings> {

    // Return default configuration values
    return Observable.of<AppSettings>(
      new AppSettings());
  }

  private handleErrors(error: any):
    Observable<AppSettings> {
  }
}
```

page you created, the values are displayed automatically on the page, as shown in **Figure 2**.

Get Settings from a JSON File

In the previous sample, you hard-coded values into the AppSettings class. Instead of hard-coding the settings values, let's place those settings into a JSON file. If you don't already have one, create a folder called **\assets** under the \src folder. Add a JSON file named **appsettings.json** in the \assets folder. Add the following code into this file:

```
{
  "defaultUrl": "http://angular.io",
  "defaultPrice": 2
}
```

Modify AppSettingsService Class

Open the appsettings.service.ts file and add a few more import statements.

```
import { Http } from '@angular/http';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
```

Import the Http class from @angular/http. Import the ReactiveJS operators map and catch. Modify the getSettings() method to call the http.get() method, passing in the path to the JSON file you created. Extract the response in the map() function and either return the JSON object from the response or an empty object. In the catch method, call a method named handleErrors(). The complete listing of the code that should be contained in the appsettings.service.ts file is shown in **Listing 2**.

Handling Exceptions

If the JSON settings file cannot be found, or if some other exception happens, the handleErrors() method is called. In addition to returning an AppSettings object, you might want to record the error information somewhere. In my last article, I created a logging system for Angular; use that if you want. For this article, just log the error to the console window. Regardless, the handleErrors() method returns an instance of an AppSettings class. You don't want your application to fail just because you can't get some specific

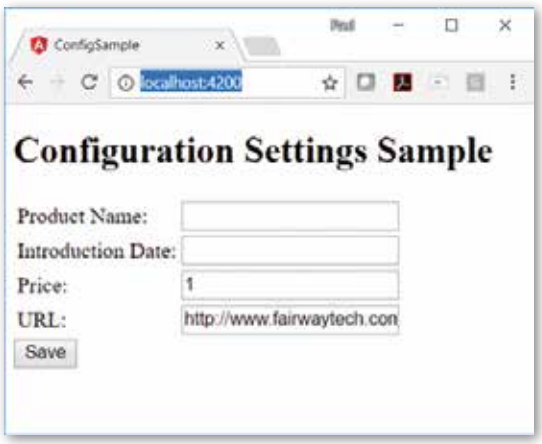


Figure 2: The Price and URL fields are filled in with defaults from the AppSettings class.

global settings. So, return an instance of AppSettings with the appropriate defaults set. Make your handleErrors() method look like the following code snippet.

```
private handleErrors(error: any):
    Observable<AppSettings> {
  // Log the error to the console
  switch (error.status) {
    case 404:
      console.error("Can't find file: " +
          SETTINGS_LOCATION);
      break;
    default:
      console.error(error);
      break;
  }

  // Return default configuration values
  return Observable.of<AppSettings>
    (new AppSettings());
}
```

Update AppModule

If you don't already have it in your application, import the HttpModule from @angular/http in your app.mod-

Listing 3: Modify the getSettings() method to retrieve values from local storage

```
getSettings(): Observable<AppSettings> {
  let settings = localStorage.getItem(SETTINGS_KEY);

  if (settings) {
    return Observable.of(JSON.parse(settings));
  }
  else {
    return this.http.get(SETTINGS_LOCATION)
      .map(response => {
        let settings = response.json() || {};
        if (settings) {
          this.saveSettings(settings);
        }

        return settings;
      })
      .catch(this.handleErrors);
  }
}
```

ule.ts file. Ensure that the HttpModule is listed in the imports property in the @NgModule() decorator function as well.

Save all the changes to the files you made. Go to the browser and you should now see the values from the JSON file appear in the price and the URL fields.

Using Local Storage

Retrieving the settings from the AppSettings class is great, but the user can't change those settings. To do that, you must store those settings somewhere. All modern browsers allow you to store key/pair values into a local storage area that persists between browser sessions. This storage is ideal for the global setting values presented in this article.

Saving Data into Local Storage

Open the appsettings.service.ts file and add a constant at the top of this file called SETTINGS_KEY. This constant contains the key value used for retrieving, storing, or deleting values in local storage.

```
const SETTINGS_KEY = "configuration";
```

Create a saveSettings() method that accepts an instance of an AppSettings class. Call the setItem() method on the localStorage object, passing in the key value contained in the SETTINGS_KEY constant as the first parameter and the AppSettings object as the second parameter. You need to stringify the AppSettings object in order to convert the JSON object to its appropriate string representation.

```
saveSettings(settings: AppSettings) {
  localStorage.setItem(SETTINGS_KEY,
                       JSON.stringify(settings));
}
```

To test the new saveSettings() method, add a new button to the product-detail.component.html page. Call a method named saveSettings() from the click event of this button by writing the following code:

```
<button (click)="saveSettings()">
  Save Settings
</button>
```

Open the product-detail.component.ts file and add the saveSettings() method to be called from the click event on the button.

```
saveSettings(): void {
  this.settings.defaultPrice =
     this.product.price;
  this.settings.defaultUrl = this.product.url;

  this.appSettingsService
   .saveSettings(this.settings);
}
```

In the saveDefaults() method, take the bound product properties and move them into the appropriate properties in the settings property of the ProductDetailComponent class. Call the saveSettings() method of the appSettingsService class that was injected by Angular into your ProductDetailComponent class.

Retrieve Settings and Store into Local Storage

Now that you have the ability to store values into local storage, you need to modify the getSettings() method in the appsettings.service.ts file to attempt to retrieve those values from local storage. If the values are found in local storage, return those values; otherwise return the values from the JSON file. Modify the getSettings() method to look like **Listing 3**.

The getSettings() method attempts to get the settings object from local storage by passing the SETTINGS_KEY value to the getItem() method. If the variable named settings contains a value, create an AppSettings object using JSON.parse() and returning an Observable of the AppSettings object.

If nothing is found in local storage, retrieve the values from the file using the http.get() method. If the values are found in the file, save them into local storage by calling the saveSettings() method. After the first time calling the getSettings() method, succeeding calls to this method retrieve values from local storage. You only get the default values from the JSON file the first time.

Delete Settings

In many applications, the user can reset back to factory defaults. To accomplish the same thing in your Angular application, delete the values stored in local storage. If you delete all of the values, then the next time the getSettings() method is called, the original values from the JSON file are read. Add a deleteSettings() method to the AppSettingsService class.

```
deleteSettings(): void {
  localStorage.removeItem(SETTINGS_KEY);
}
```

Because the complete AppSettings object is stored within the one key in local storage, call the removeItem() method and pass in the SETTINGS_KEY constant to erase all the settings. To test this method, add a new button to the product-detail.component.html page in the sample.

```
<button (click)="deleteSettings()">
  Delete Settings
</button>
```

Open the product-detail.component.ts file and add the deleteSettings() method. In this method, call the deleteSettings() method on the appSettingsService object that was injected into this component.

```
deleteSettings(): void {
  this.appSettingsService.deleteSettings();
}
```

Save all of your changes, switch back to your browser, and you should see your two new buttons appear on the screen. Try modifying the price and URL fields and click the Save Settings button. Refresh the page to ensure that the new values are retrieved from local storage. Next, try clicking on the Delete Settings button to delete the values from local storage. Refresh the page again to ensure that the values are retrieved from the JSON file.

Create Web API for Configuration Settings

In the last two examples in this article, you hard-coded global settings and retrieved them from a JSON file. Now you're going to retrieve settings by calling a Web API method and get the settings from a SQL Server table. To use a Web API call, you need to perform a few steps.

- Create a new Web API project in Visual Studio
- Create a SQL Server table named AppSettings
- Create Entity Framework classes to retrieve data from the SQL table
- Add a C# ConfigController class
- Enable Cross-Origin Resource Sharing (CORS)
- Convert C# PascalCase property names to JSON camelCase property names

Create a Visual Studio Web API Project

Open Visual Studio 2017 and select File > New > Project... from the main menu. From the New Project dialog, select Visual C# > Web > ASP.NET Web Application (.NET Framework). Set the name to ConfigWebApi and click the OK button. In the New ASP.NET Web Application dialog, select the Web API template and click the OK button. After a minute or two, you'll have a new Web API project created.

Create SQL Server Table

Create a table named AppSettings on an available SQL Server. This table should have column names that match the Angular AppSettings class. As with any proper database design, add a primary key for the table named AppSettingsId. Set this primary key as non-clustered, and add the IDENTITY attribute to start with the value of 1 and increment by 1 for each new record added. Below is the T-SQL statement to create this table in SQL Server.

```
CREATE TABLE dbo.AppSettings(
       AppSettingsId int NOT NULL
   CONSTRAINT PK_AppSettings PRIMARY KEY
     NONCLUSTERED IDENTITY(1,1),
       DefaultUrl nvarchar(255) NULL,
       DefaultPrice money NULL
    CONSTRAINT DF_AppSettings_DefaultPrice
```

```
    DEFAULT ((1))
)
```

After creating your table, add a new record to the table with some default values such as the following:

```
INSERT INTO AppSettings(DefaultUrl, DefaultPrice)
VALUES ('http://www.google.com', 42);
```

Create Entity Framework Classes

Right mouse-click on the Models folder in your Visual Studio project and choose Add > ADO.NET Entity Data Model... from the context-sensitive menu. Set the name of the data model to ConfigSample. Select Code First from Database from the first page of the Entity Data Model Wizard dialog and click the Next button. Create a new connection to the SQL Server and select the database where you created the AppSettings table and click the Next button. Drill down to the AppSettings table you created and click the Finish button. After about a second, two Entity Framework classes have been generated and added to your Models folder.

ConfigController Class

You're now ready to write your Web API controller and a method to retrieve the AppSettings values in your SQL Server table. Right mouse-click on the Controllers folder and select Add > Web API Controller Class (v2.1) from the menu. Set the name of this new controller to **ConfigController** and click the OK button. Add a using statement at the top of this file.

```
using ConfigWebApi.Models;
```

Delete all the methods within this class, as you don't need them. Add a Get() method, shown in **Listing 4**, to retrieve the settings from the SQL Server table. When writing a Web API method, return an object that implements the IHttpActionResult interface. The ApiController class has a few methods you call to create a return object. The Get() method creates an instance of the ConfigSample DbContext. Use the FirstOrDefault() method on the AppSettings collection to retrieve the first record in the AppSettings table. If a null is returned from this method, set the return variable to NotFound(). If a record is found, call the Ok() method and pass the settings variable into this method so that the values are returned from this method. If an error occurs, return an InternalServerError object.

Enable CORS

Because you created a new Visual Studio application to run your Web API method, it's running in a different domain from your Angular application. For your Angular application to call this Web API you must tell the Web API that you're allowing Cross-Origin Resource Sharing (CORS). Right-mouse click on your Web API project and select Manage NuGet Packages... Click on the Browse tab and search for **cors** as shown in **Figure 3**. Install this package into your project.

After CORS is installed into your project, open the \App_Start\WebApiConfig.cs file and add the following line of code in the Register() method:

```
public static void Register(HttpConfiguration config)
{
  config.EnableCors();

  ...
}
```

Go back to your ConfigController class and add the following using statement:

```
using System.Web.Http.Cors;
```

Add the EnableCors() attribute just above your ConfigController class. You can get specific on the origins, headers, and methods properties to restrict access to only your Angular application if you want. For the purposes of this article, just set them to accept all requests by specifying any asterisk for each, as shown in the following code snippet:

```
[EnableCors(origins: "*",
            headers: "*",
            methods: "*")]
public class ConfigController : ApiController
{
   ...
}
```

Convert C# Class to JSON

C# property names are generally created using Pascal casing. Pascal casing means each letter of each word in

Listing 4: Add a Get() method to your Web API controller to retrieve settings from a SQL table.

```
[HttpGet]
public IHttpActionResult Get()
{
  IHttpActionResult ret;
  ConfigSample db = null;
  AppSetting settings = new AppSetting();

  // Retrieve settings from SQL table
  try {
    db = new ConfigSample();

    settings = db.AppSettings.FirstOrDefault();
    if (settings == null) {
      ret = NotFound();
    }
    else {
      ret = Ok(settings);
    }
  }
  catch (Exception) {
    ret = InternalServerError(
        new ApplicationException(
          "Error retrieving settings
           from AppSettings table."));
  }

  return ret;
}
```

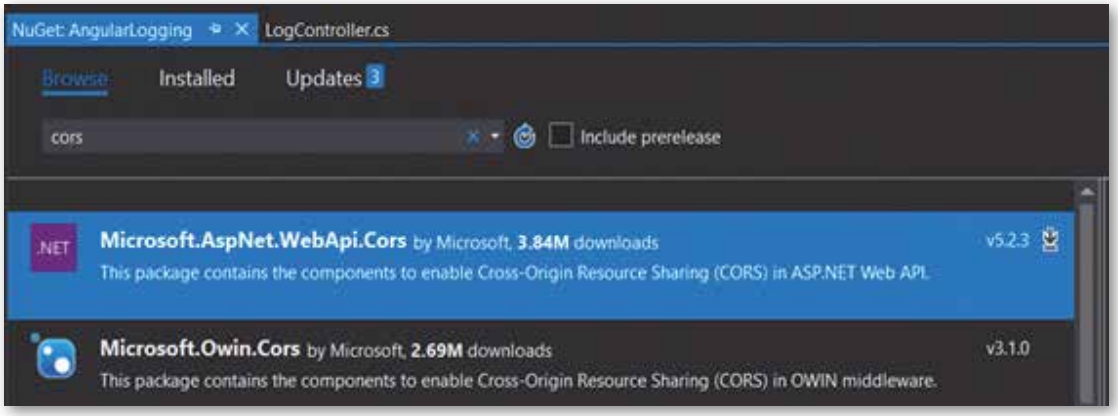


Figure 3: Search for Microsoft CORS

the property name is initial-capitalized. The convention for naming properties in JavaScript and TypeScript is to use camel casing. Camel casing is where each word in the property name is initial capitalized except for the first word. So, the DefaultUrl property name in C# is expressed as defaultUrl in TypeScript. When the Web API serializes the AppSettings class to return it to your Angular application, it takes the property names of C# and translates them directly into JSON. You can change the serializer so that the Web API converts the Pascal-cased C# property names into camel-cased TypeScript property names.

Open the WebApiConfig.cs file and add a couple of using statements at the top of this file.

```
using System.Net.Http.Formatting;
using Newtonsoft.Json.Serialization;
```

Locate the Register() method and just below the first comment, add the lines shown in the following code snippet. Leave the rest of the code the same in the Register() method.

```
public static void Register(
  HttpConfiguration config)
{
  // Convert to camelCase
  var jsonFormatter = config.Formatters
    .OfType<JsonMediaTypeFormatter>()
      .FirstOrDefault();

  jsonFormatter.SerializerSettings
   .ContractResolver = new
      CamelCasePropertyNamesContractResolver();

  // Rest of the code below here
}
```

Run your Web API project and when it's running in the browser, copy the URL from the address line to the clipboard. You're going to need this address for your Angular service.

Modify AppSettingsService

Switch back to your Angular project in Visual Studio Code. Open the appsettings.service.ts file and locate the constant SETTINGS_LOCATION. Replace the contents of the value with what's in your clipboard. Then, add on "api/config" to the end. Your constant should now look like the following (with a different port number, of course).

```
const SETTINGS_LOCATION =
    "http://localhost:8314/api/config";
```

Switch to the browser that's running your Angular project. Click on the Delete Defaults button to delete the previous settings. Refresh your browser, and you should now see the values returned from your Web API call in the appropriate fields on the Web page.

Summary

In this article, you learned an approach for handling application-wide settings for Angular applications. A service approach is the most flexible approach for providing settings to any other class in your application. You can choose to store your settings in a class or in an external JSON file, or make a call to a Web API to retrieve the values. Store the settings retrieved into local storage to allow your users to modify those settings, if desired. If you delete the values from local storage, you allow your user to revert to the original default settings.

Paul D. Sheriff

CODE

SharePoint Framework Extensions

SharePoint Framework, or SPFx for short, is growing up. At the Ignite conference, Microsoft released a number of improvements to SPFx, along with promises for the very near future. Now you can use SPFx on-premises with SharePoint 2016 Feature Pack 2. Not everything that works in the cloud also works on-premises. For instance, the topic here, extensions,

Sahil Malik
www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets. You can find more about his training at http://www.winsmarts.com/training.aspx.

His areas of expertise are cross-platform mobile app development, Microsoft anything, and security and identity.

is currently a SharePoint online-only thing. This is because SharePoint 2016 on-premises doesn't support Modern UI yet. But there is a good chance that with SharePoint 2019, it will. Classic sites could get this feature working down the road as well. Most certainly, you shouldn't be investing in techniques such as JSLink given that you have SharePoint framework extensions available now.

Other things, such as the very welcome improvement of Tenant-level deployment, are also not available on SharePoint on-premises yet. This may change in the future, but as a high-level guidance, on-premises will always be a subset of SharePoint online. Whatever you write for on-premises will work in the cloud. The other way around is not true. Also, bit by bit, features from SharePoint online will make it to on-premises.

One of the very interesting improvements that Microsoft has recently released is SharePoint Framework Extensions. This brings SPFx to more than just Web Parts, and truly expands the applicability of SPFx and establishes it as a solid dev story going forward.

What Are SPFx Extensions?

SharePoint Framework extensions can be used to extend the SharePoint user experience. Specifically, you can customize more than just Web Parts. You can change how lists render. You can add content to specific parts of the page or add new actions at specific areas within SharePoint.

Think of this new development experience as a "development experience on rails." Microsoft wants to enable all the facilities you may need to solve a business need. At the same time, they're providing your train with rails, so you stay on track, while enjoying the beautiful scenery outside! Can you abuse SPFx to completely take over a SharePoint page? Absolutely! But you shouldn't. Microsoft is making it very clear what's acceptable and what's not. It's best to keep your train on the rails, even though riding a train in the jungle sounds like fun.

There are three kinds of extensions currently:

- **Application customizers:** Allow you to add scripts to a page, or access well-known HTML element placeholders and put custom rendering logic there
- **Field Customizers:** Allow you to modify how a list renders. For instance, a "progress" field might be a lot more interesting as a progress bar than a number. With field customizers, you can change that number to look like a progress bar.
- **Command Sets**: Allow you to add new actions and provide client-side code for implementing behaviors.

Let's examine each one of these one by one. For the purposes of this article, I'll assume that you're familiar with the basics of SPFx and have a working SPFx dev environment ready. I'm working with version 1.3.2 of the yeoman generator for SharePoint.

Application Customizers

Application Customizers allow you to add scripts on a page, or to access well known HTML element place holders and put custom rendering logic there. If you're a seasoned SharePoint developer, you may remember something called "delegate controls." You can think of Application Customizers as delegate controls for the SPFx world. Microsoft wants to make it possible for you to change certain elements, such as the header and footer among others, in a supported manner.

This is quite useful because now you can do things, such as examine a site URL and provide users with a disclaimer. Or embed functionality that lets users change the view of a site that affects all your other SPFx customizations. Best of all, a certain very valuable scenario opens up. When you do tenant-wide deployment of SPFx solutions, they "skip feature deployment;" in other words, your feature.xml is ignored. This means that if your solutions depend on, say, having a list or a content type present, tenant-wide solutions can't really support that at the moment.

This problem may get solved down the road if we get a "hook," a place where you can run code when a solution is deployed or removed. But right now, you can use two possible workarounds:

- You can use PnP PowerShell to enable those features in the sites you want to have that feature in
- You can use an Application Customizer that runs a one-time script to enable those features

Also note that currently, Application Customizers are supported at site, Web, and list scopes. Let's see how to write a simple Application Customizer.

As always, create a new SPFx project using this command:

```
yo @microsoft/sharepoint
```

This leads you through a wizard, where you'll:

- Target "SharePoint Online only."
- Choose "N" when asked to make it a tenant-wide solution.
- Chose "Extension" as the kind of component type.
- Choose "Application Customizer."

Once the project is created, type **code .** to open it in VS Code.

Listing 1: The Application Customizer

```
import { override } from '@microsoft/decorators';
import { Log } from '@microsoft/sp-core-library';
import {
  BaseApplicationCustomizer
} from '@microsoft/sp-application-base';
import { Dialog } from '@microsoft/sp-dialog';

import * as strings from 'AppcustomizerApplicationCustomizerStrings';

const LOG_SOURCE: string = 'AppcustomizerApplicationCustomizer';

export interface IAppcustomizerApplicationCustomizerProperties {
  testMessage: string;
}

export default class AppcustomizerApplicationCustomizer
extends BaseApplicationCustomizer<
IAppcustomizerApplicationCustomizerProperties> {

  @override
  public onInit(): Promise<void> {
    Log.info(LOG_SOURCE, `Initialized ${strings.Title}`);

    let message: string = this.properties.testMessage;
    if (!message) {
      message = '(No properties were provided.)';
    }

    Dialog.alert(`Hello from ${strings.Title}:\n\n${message}`);

    return Promise.resolve();
  }
}
```

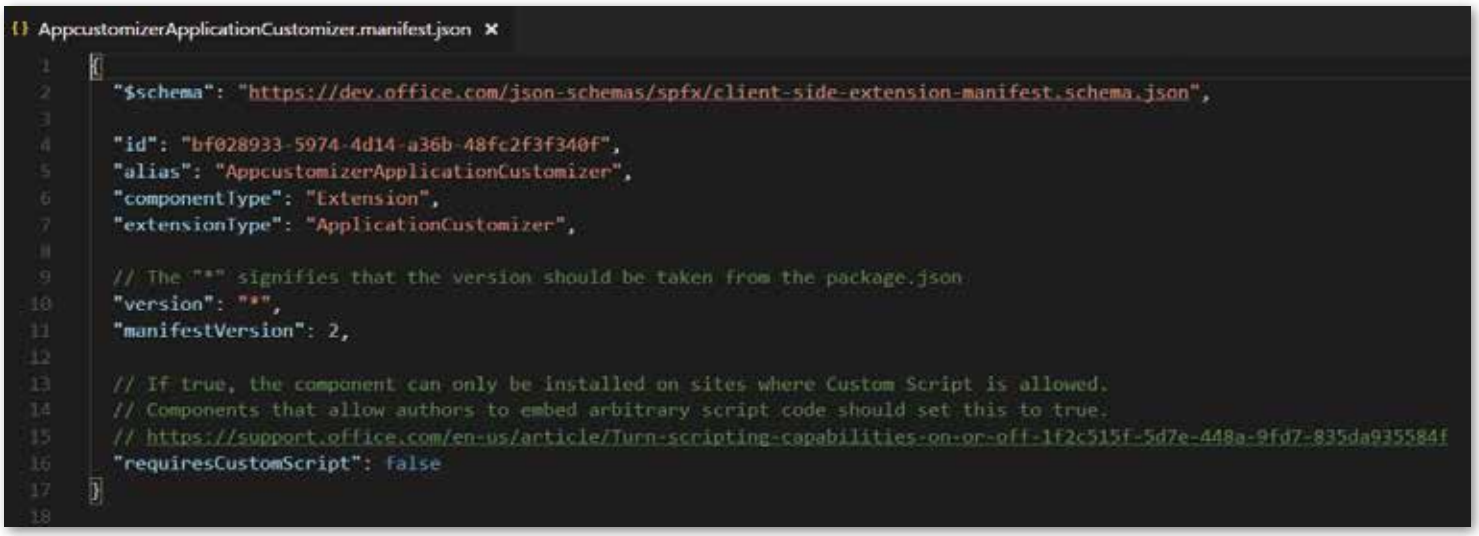


Figure 1: The Application Customizer manifest.json file.

Once your project opens, go to the **src\extensions** folder and locate your customizer. There should be only one, and it should match the name you provided when creating the solution. Inside there, you should see a **manifest.json** file which describes to SharePoint the details of your customizer. It should look like **Figure 1**.

Note that there is a GUID there, the "id" field. That's the unique identifier for your customizer; take a note of that because you'll need it soon.

Next, open the .ts file for your customizer. You should find it in the same folder as the manifest.json file. My .ts file can be seen in **Listing 1**.

As can be seen in **Listing 1**, an Application Customizer is simply a class that inherits from BaseApplicationCustomizer. Just like Web Parts, it can be strongly typed to reflect the properties it supports. Also note that our customizations go in the "OnInit" method and not the constructor. This is because the state of the customizer isn't available in the constructor.

In order to run the customizer, run **gulp serve --nobrowser**, and access the following URL:

```
https://yoursitecollectionurl/
loadSPFX=true&
debugManifestsFile=
https://localhost:4321/temp/manifests.js&
customActions={
"theGUIDOfYourCustomizer":{
"location":
  "ClientSideExtension.ApplicationCustomizer",
"properties":
  {"testMessage":"Hello as property!"}}}
```

Note that I've broken down the URL with return characters to make it easier to read. But it's a URL, so write it in a single line. Also, the GUID mentioned above is the GUID from your manifest.json file.

Your browser should show you a dialog box, as shown in **Figure 2**.

This disclaimer in **Figure 2** is a good idea, because by allowing debug scripts to load, you're effectively running them within the context of the page. It serves as a good warning to end-users not to load any random stuff from the Internet, while giving developers an easy way to in-

Listing 2: AppcustomizerApplicationCustomizer.module.scss

```
.app {
  .top {
    height:60px;
    text-align:center;
    line-height:2.5;
    font-weight:bold;
    display: flex;
    align-items: center;
    justify-content: center;
  }
  .bottom {
    height:40px;
    text-align:center;
    line-height:2.5;
    font-weight:bold;
    display: flex;
    align-items: center;
    justify-content: center;
  }
}
```

Figure 2: Load the debug scripts.

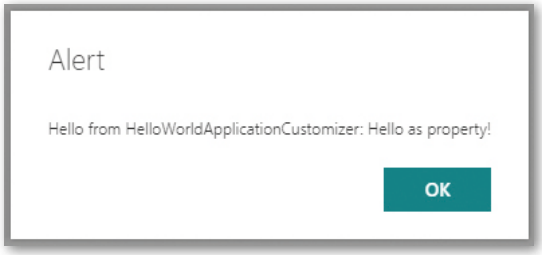


Figure 3: The Application Customizer is running.

ject and test their code. Go ahead and load debug scripts and verify that you can see the customization, as shown in **Figure 3**.

Using this technique, you can run any arbitrary JavaScript on the page. Is this ripe for abuse? Absolutely, but what isn't!? You should make an effort to stay within the "rails" Microsoft has provided for your train to run in. One acceptable use of Application Customizers is to use them to access certain predefined areas of your page. Let's extend the Application Customizer to support that. In your .ts file for the customizer, edit the import statement as follows:

```
import {
  BaseApplicationCustomizer,
  PlaceholderContent,
  PlaceholderName
} from '@microsoft/sp-application-base';
```

The intention here is that you wish to customize the top and bottom place holders, so also go ahead and edit the interface specifying the properties to support the Top and Bottom placeholder messages, as shown here:

```
export interface IAppcustomizerApplicationCustomizerPr
operties {
  testMessage: string;
  Top: string;
  Bottom: string;
}
```

You wish to show the top and bottom place holders in a particular style, which you need to define in a new scss file. Go ahead and import the file in the .ts file as shown next:

```
import styles from
'./AppcustomizerApplicationCustomizer.~
~module.scss'
```

Note that for the import to work, the file must be named ***.module.scss**. This is how the SPFx templates are wired to use Web Pack. The **import** statement in TypeScript can access the styles mentioned in a .module.scss file.

Inside this new scss file, add the styles as shown in **Listing 2**.

Next, you need to make appropriate changes to take advantage of these new place holders and properties. The new code for the Application Customizer can be seen in **Listing 3**.

As can be noted from **Listing 3:**

- Not every placeholder is guaranteed to be present on every page. You can iterate through the list of available place holders using this.context.placeHolderProvider.placeHolderNames property.
- You need to check to see whether or not a certain placeholder is present.
- If the placeholder is present, render it as you wish.

In order to run this, use a similar URL as before, but just make sure that you now also pass in the **Top** and **Bottom** properties. The query string portion of the URL you can use to test this is shown in **Listing 4**. Note that I have broken the URL apart into new lines to make it easier to read.

Verify that this customizer now produces an output, as shown in **Figure 4**.

Deploying the Customizer

So far, you've been running the customizer in debug mode. You can't expect users to remember that long URL. In order to deploy the Application Customizer, you need to do two main things:

- Set the ClientSideComponentId and ClientSideComponentProperties values.
- Perform the usual packaging, CDN, and deployment that's typical to any SPFx package.

The idea is that using a feature.xml file, you specify to SPFx that a certain customizer with a given GUID, as specified in the ClientSideComponentId property, is going to customize using properties specified as escaped JSON in ClientSideComponentProperties.

You may be scratching your head here, wondering if you really have to hard-code these properties into an escaped JSON inside an XML file. Yes, you do! But these properties are just an initial starter. You can use these properties to point to a Web service that can give you dynamic values, or simply calculate dynamic values based on context. So, this isn't really a show stopper. Also remember that there's a concept called **tenant properties**, so this initial

Listing 3: Application Customizer rendering top and bottom place holders

```
import { override } from '@microsoft/decorators';
import { Log } from '@microsoft/sp-core-library';

import styles from './AppcustomizerApplicationCustomizer.module.scss';
import { escape } from '@microsoft/sp-lodash-subset';

import {
  BaseApplicationCustomizer,
  PlaceholderContent,
  PlaceholderName
} from '@microsoft/sp-application-base';
import { Dialog } from '@microsoft/sp-dialog';

import * as strings from 'AppcustomizerApplicationCustomizerStrings';

const LOG_SOURCE: string = 'AppcustomizerApplicationCustomizer';

export interface IAppcustomizerApplicationCustomizerProperties {
  testMessage: string;
  Top: string;
  Bottom: string;
}

export default class AppcustomizerApplicationCustomizer
  extends BaseApplicationCustomizer
<IAppcustomizerApplicationCustomizerProperties> {

  private _topPlaceholder: PlaceholderContent | undefined;
  private _bottomPlaceholder: PlaceholderContent | undefined;

  @override
  public onInit(): Promise<void> {
    Log.info(LOG_SOURCE, `Initialized ${strings.Title}`);
    this.context.placeholderProvider.changedEvent.add(this,
 this._renderPlaceHolders);

    // Call render method for generating the HTML elements.
    this._renderPlaceHolders();
    return Promise.resolve<void>();
  }

  private _renderPlaceHolders(): void {
    // Handling the top placeholder
    if (!this._topPlaceholder) {
      this._topPlaceholder =
        this.context.placeholderProvider.tryCreateContent(
          PlaceholderName.Top,
          { onDispose: this._onDispose });
      if (!this._topPlaceholder) {
        return;
      }

      if (this.properties) {
        let topString: string = this.properties.Top;
        if (!topString) {
          topString = '(Top property was not defined.)';
        }

        if (this._topPlaceholder.domElement) {
          this._topPlaceholder.domElement.innerHTML = `
            <div class="${styles.app}">
<div class="ms-bgColor-themeDark ms-fontColor-white ${styles.top}">
<i class="ms-Icon ms-Icon--Info" aria-hidden="true"></i> ${escape(topString)}
              </div>
            </div>`;
        }
      }
    }

    // Handling the bottom placeholder
    if (!this._bottomPlaceholder) {
      this._bottomPlaceholder =
        this.context.placeholderProvider.tryCreateContent(
          PlaceholderName.Bottom,
          { onDispose: this._onDispose });

      if (!this._bottomPlaceholder) {
        return;
      }

      if (this.properties) {
        let bottomString: string = this.properties.Bottom;
        if (!bottomString) {
          bottomString = '(Bottom property was not defined.)';
        }

        if (this._bottomPlaceholder.domElement) {
          this._bottomPlaceholder.domElement.innerHTML = `
            <div class="${styles.app}">
<div class="ms-bgColor-themeDark ms-fontColor-white ${styles.bottom}">
<i class="ms-Icon ms-Icon--Info" aria-hidden="true"></i>
${escape(bottomString)}
              </div>
            </div>`;
        }
      }
    }
  }
  private _onDispose(): void {}
}
```

starter value could point to a tenant property, the value of which can be set by an administrator. And that tenant property can point to a Web service, etc., for a further set of run-time configurable parameters for your solution to run.

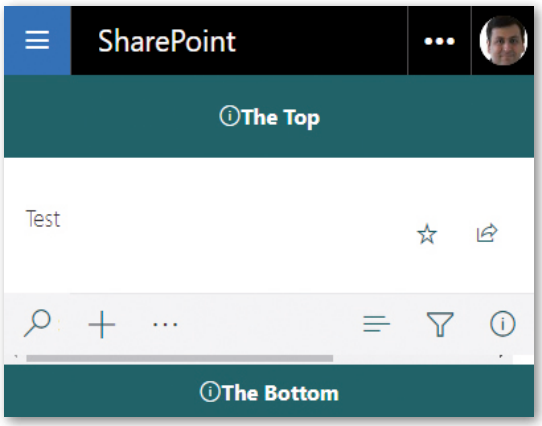


Figure 4: The Application Customizer modifying the top and bottom place holders

Figure 5: The Field Customizer in action

Listing 4: The URL you can use to test the Application Customizer.

```
https://yoursiteurl?
loadSPFX=true&
debugManifestsFile=https://localhost:4321/temp/manifests.js&
customActions={
"theGUIDofYourCustomizer":{
"location":"ClientSideExtension.ApplicationCustomizer","
properties":{
"Top":"The Top", "Bottom":"The Bottom"
}}}
```

Listing 5: The default implementation of a Field Customizer

```
public onRenderCell(
  event: IFieldCustomizerCellEventParameters): void {
  // Use this method to perform your custom cell rendering.
  const text: string =
  `${this.properties.sampleText}: ${event.fieldValue}`;
  event.domElement.innerText = text;
  event.domElement.classList.add(styles.cell);
}
```

At this point, edit the sharepoint\assets\elements.xml file, and ensure that:

- The ClientSideComponentId property matches the GUID of your manifest.json
- The ClientSideComponentProperties include an escaped JSON string indicating the properties you'd wish to use. An example of such a string could be, **{"Top":"The top","Bottom":"The bottom"}**

Next, for this elements.xml file to get activated during deployment, ensure that in the config\package-solution.json file, it's included under features\assets\elementManifests node. This should already be in place; I just wanted to point out how things are wired up.

Finally, do the usual sppkg packaging and deployment, activate the app, and deploy files to a CDN. Verify that now the application customizer is available by default on the site on which you have activated the customizer.

There is, however, an important thing to consider about tenant-wide deployment of SPFx solutions that contain extensions. Tenant-wide solutions don't activate your feature XML files. This means that although you can deploy these solutions, they don't take effect. However, using PnP PowerShell, you can manually activate the features that you require to get around this requirement. This way, you can use tenant-wide deployment with customizers and easily enable your customizations across a number of sites.

Field Customizers

Next, let's turn our attention to Field Customizers. The deployment steps are exactly the same as an Application Customizer, so I'll keep the discussion pertinent only to the differences.

Put simply, Field Customizers allow you to change how list views for a particular field get rendered. This gives you immense flexibility in how the user views the data. An important consideration here is that the list view doesn't finish rendering until each field renderer method has done its job. So, keep these either short and quick, or leave heavy-duty rendering for asynchronous operations.

Beyond the obvious usage of custom rendering fields, field renderer methods can use be used to present master-detail information or even information from other systems.

Creating a Field Customizer is quite simple! Just run the SharePoint yeoman generator, as shown under the Application Customizer, but instead of picking to create an Application Customizer, choose to create a Field Customizer instead.

When the application is finished scaffolding, visit the .ts file for your FieldCustomizer. Note that it's just a class that inherits from BaseFieldCustomizer, and it's strongly typed to an interface that represents Field Customizer properties. There seems to be a pattern here! Yes, all SPFx artifacts follow this obvious pattern.

Also note that your custom rendering goes in a simple method called **onRenderCell**. Here, you can access event.domElement to access the domElement where the field is being rendered. Once you have hold of the DOM element, you can pretty much render it however you wish. This can be seen in **Listing 5**.

> All SPFx artifacts follow an obvious pattern.

This Field Customizer is written to look for a field called **Percentage**, so in a modern UX site, go ahead and create a list, and add a column called "Percentage". Go ahead and add some data into this list. In order to run this customizer, visit a URL, as shown in **Listing 6**.

Verify that it changes the UX of the Percentage field, as shown in **Figure 5**.

Command Sets

Command Sets allow you to add your own custom actions within the UX of SharePoint. Remember **custom action**? Think of this as the modern custom action.

As always, generate a new SharePoint project, choose to target SharePoint online, create an extension, and this time create a command set kind of an extension. Now, visit the manifest.json for your command set, and you'll note that the various commands you'd like to add are defined in this file. The relevant snippet from this JSON file can be seen in **Listing 7**.

And as usual, the actual definition of the command set extension can be found in the associated .ts file. Note that, as usual, it's a class that inherits from a strongly typed instance of BaseListViewCommandSet, which is strongly typed to the properties your command set supports. Inside this class, you'll see two methods overridden.

The onListViewUpdated method is an event. It's called whenever the ListView state changes. This is where you can hide/show it, change its title, etc. This is called for each command for every list view change.

The onExecute method is where you put the handling logic when the command is executed by the user.

Additionally, you can use tryGetCommand to find out whether a certain command exists, or use the ID property to get the ID of the current command.

As usual, go ahead and run this command-set extension. Verify that your command shows up in the command bar of any list where this command-set extension is activated. Also verify that when you select an item, both commands show up. These can be seen in **Figure 6**.

You can imagine that this would be incredibly useful, for instance, in a document that you can implement and "send to another system" or a proposal where you can enable "discuss in a meeting" functionality etc.

Listing 6: Running your Field Customizer

```
yoursiteurl?
loadSPFX=true&
debugManifestsFile=https://localhost:4321/temp/manifests.js&
fieldCustomizers=
{"Percentage":{
"id":"yourFieldCustomizeGUID",
"properties":{"sampleText":"Hello!"}}}
```

Listing 7: Commands defined in the manifest.json file

```
"items": {
  "COMMAND_1": {
    "title": { "default": "Command One" },
    "iconImageUrl": "icons/request.png",
    "type": "command"
  },
  "COMMAND_2": {
    "title": { "default": "Command Two" },
    "iconImageUrl": "icons/cancel.png",
    "type": "command"
  }
```

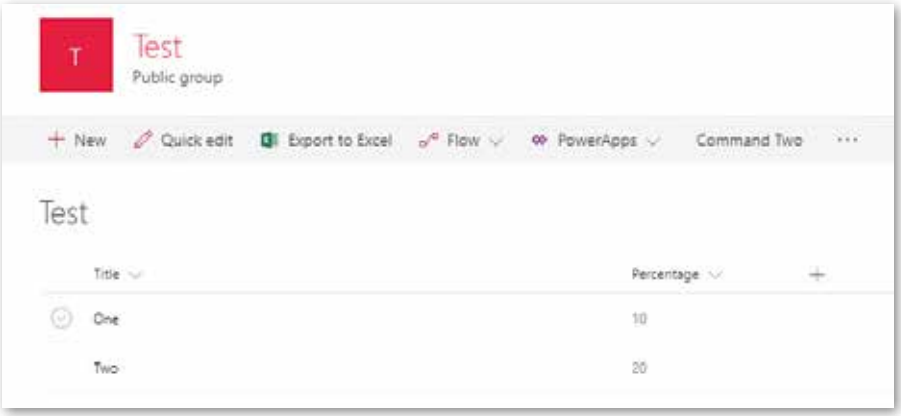


Figure 6: The commands show up in a list.

Summary

With extensions, the SharePoint framework is boldly charging into different areas of customizing SharePoint. These are all very valid requirements, and also give a good dev story to modern sites. Not only that, with SharePoint framework's applicability to on-premises, we finally have a programming model that's consistent between on premises and cloud.

SharePoint Framework isn't "done" yet. There are still critical features and functionalities that are still being built. But it's getting close, and it's forming well into a very good development mechanism for SharePoint.

Speaking of things that aren't done yet, here's a short list of what's missing: an API to manage deployment of SharePoint packages, the ability to call graph and custom APIs easily, site collection-level app catalogs, and maybe even the ability to use Angular with SPFx. All of these are coming, and I hope to talk about them in the very near future.

Sahil Malik

A SQL Programming Puzzle: You Never Stop Learning

As I continue to work on my next major article on Data Warehousing, I want to take a break and do a little post-mortem on a database programming challenge I recently faced. It was a quiet Saturday afternoon, and I honestly thought it would take no longer than an hour to write a SQL query to aggregate some data for specific ranges of time. Unfortunately, I was a little too confident, and it wound up taking me the better part of a day (with interruptions). I finally came up with a solution, then reviewed a pattern I hadn't considered, and finally wound up simplifying my solution. So, what started as a seemingly mundane task became an unexpected learning experience.

Kevin S. Goff
kgoff@kevinsgoff.net
http://www.KevinSGoff.net
@KevinSGoff

Kevin S. Goff is a Microsoft Data Platform/SQL Server MVP. He is a database architect/developer/speaker/author, and has been writing for CODE Magazine since 2004. He's a frequent speaker at community events in the Mid-Atlantic region and also spoke regularly for the VS Live/Live 360 Conference brand from 2012 to 2015. He creates custom webcasts on SQL/BI topics on his website.

This is a Humbling Industry

I started in this industry in 1987. Along the way, I've had significant flashpoints, both positive and negative. I've also heard some profound quotes. In 2003, I overheard a debate between Developer A and Developer B. Developer B felt that Developer A was excessively overconfident about a situation, and Developer B said something I'll never forget: "This is a humbling industry. You really need to be careful, that's all I'm saying." As it turns out, Developer B was right, as Developer A indeed experienced a crisis for which the term "hubris" aptly applied.

I have 30 years of experience and have worked in four decades. I'm fortunate that I've worked on many systems and have had many successes (and failures to learn from). When you approach a certain level of experience, it's a never-ending "Catch-22." Clients often seek those with meaningful experience, but experience has a way of backfiring if you become too cocky. Most of us have seen developers whose arrogance wound up backfiring, but even honest and sincere veterans can fall victim to it, even in isolated situations. That happened to me a few weekends ago, and it reinforced that no matter how much I might want to pride myself on being able to solve database programming puzzles, there will always be humbling experiences and **always** someone who knows more!

Step 1: Defining the Problem and the Desired Outcome

How many times have you heard this: "What is it you're trying to accomplish?" At the risk of stating the obvious, defining the problem or goal in the simplest terms possible is essential to building a solution.

> At the risk of stating the obvious, defining the problem or goal in the simplest terms possible is essential to building a solution.

The siuation I faced is too intricate to cover in its entirety in an article, so I'll strip it down to the bare essentials. I can best simplfiy the challenge I faced in an image. Suppose you have an order that goes through the processes shown in in **Figure 1**.

I have an order that spent three days in Process A, then four days in Process B, and one day in Process C. Because of a problem, the work reverted back to Process A for three days, then moved on to Process D for two days, then back (again) to Process A for two days and finally to Process D again for one day.

Here's the goal: Produce the result set shown in **Figure 2** that reflects each stage:

Initially, I thought this would be incredibly easy. Well, once in a while, our prior experiences can haunt us, if we mistakenly conclude that the new challenge follows the pattern of past successes. After my first query failed miserably (so miserably that I won't humiliate myself here), I stepped back and realized that I'd need more than just a simple MIN/MAX/GROUP BY.

Moving On, My Next Step: Grab for a Branch, Any Branch!

I realized that I needed to break this down into a few steps. I started with something simple. When you feel like you're falling from a tree, you grab the first branch that you can!

I needed to detect, for any given day, the process for the following day. That seemed to be the only way to establish the "breaks" in process that would ultimately lead to the final result. So, I did what you can see in the next code snippet, and directed the results to a table (TempResult1_GetNextDay). Note that I used the **LEAD** function, which Microsoft added in SQL Server 2012. (For those using older versions of SQL Server, or another database, you could perform a SELF-OUTER-JOIN to the same table, varying the date by one day, to get the Process for the next day.)

```
SELECT *, LEAD(ProcessName, 1) OVER
    (ORDER BY ProcessDate) AS ProcessNextDay
     INTO TempResult1_GetNextDay
     FROM ProcessRows
```

If you're not familiar with the LEAD function, let's go back to the data in **Figure 1**. For the row with the ProcessDate of 2017-10-03, I also want to store a value for the process on the next day (Process B). Alternatively, for the row with the ProcessDate of 2017-10-04, I could store a value for the process on the prior day (Process A.) Either way, having that value for the next/prior day allows me to easily see when the process changes by day. The LEAD function allows me to "skip forward" one row,

based on the order of Process Date, to grab the Process Name for the next row:

```
LEAD(ProcessName, 1) OVER
   (ORDER BY ProcessDate) AS ProcessNextDay
```

That produces the results in **Figure 3**. Now that I've identified the "breaks," take a look at 2017-10-03, which is the last day of Process A, because the following day it changed to Process B. Yes, I could have used the **LAG** function instead to create "ProcessPreviousDay" and identified the breaks that way. Either way, there are seven instances where the ProcessName differs from ProcessNextDay, and that's the key to creating the seven groups for the final result set.

> I could have used the LAG function to create "ProcessPreviousDay" and identified the breaks that way.

Now You Can Identify the Groups

From **Figure 3**, where I see the days where the process is about to change, I can query the rows from TempResult1_GetNextDay, but only for those rows where the ProcessName and ProcessNextDay are different (or the ProcessNextDay is null).

Now You See an Opportunity to Line the Data Up!

I can read from Temp1Result_GetNextDay and manipulate the dates to produce a result set that I truly covet right now in **Figure 5**: lining up the start/end date for each process phase. I can use the LAG function to skip "back" one row for a new phase, to get the process start date for that phase.

Here's the code for **Figure 5**.

```
SELECT DATEADD(d,1,LAG(ProcessDate,1)
    OVER (ORDER BY ProcessDate))
       as ProcessStartedDate ,
 ProcessDate AS ProcessEndedDate ,
  RANK() OVER (ORDER BY ProcessDate)
   as GroupNumber ,
   ProcessName
   INTO TempResult2_GetGroups
  FROM TempResult1_GetNextDay
     WHERE ProcessName <> ProcessNextDay OR
          ProcessNextDay IS NULL
```

Note that I used the RANK function to produce a GroupNumber column. This isn't required for the final result set: It was merely for reference purposes while testing. Now I have a second intermediate result set: TempResult2_GetGroups. Now that I have the start/end dates lined up, I can calculate the elapsed days for each phase. Almost!

Note that back in **Figure 5**, there's a NULL value for the Process Started date for the first phase. That's because

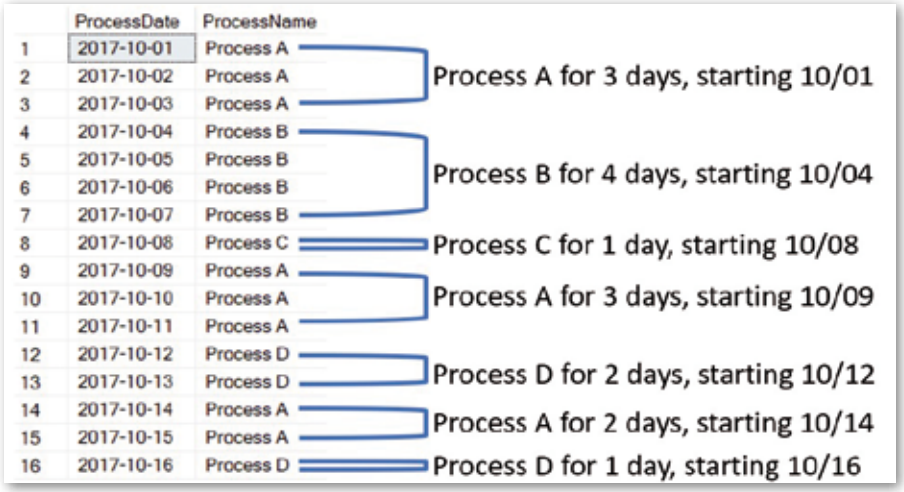


Figure 1: Raw data of daily processes

	ProcessStartedDate	ProcessEndedDate	ProcessName	NumDays
1	2017-10-01	2017-10-03	Process A	3
2	2017-10-04	2017-10-07	Process B	4
3	2017-10-08	2017-10-08	Process C	1
4	2017-10-09	2017-10-11	Process A	3
5	2017-10-12	2017-10-13	Process D	2
6	2017-10-14	2017-10-15	Process A	2
7	2017-10-16	2017-10-16	Process D	1

Figure 2: The desired end result

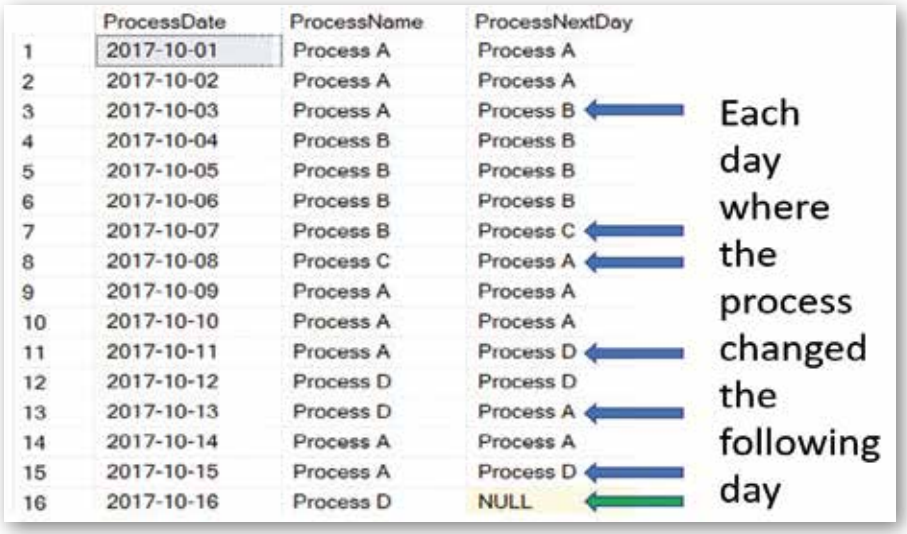


Figure 3: Identifying where the processes break out

I used the LAG function to go backward before the first row. You can generate another intermediate result set (TempResult3_GetGroups) and grab the first process date from the original table for that process using a correlated subquery. That gives the results in **Figure 6**.

```
select ISNULL(ProcessStartedDate,
   (SELECT MIN(ProcessDate) FROM ProcessRows Inside
   WHERE Inside.ProcessName = Outside.ProcessName))
        AS ProcessStartedDate,
   ProcessEndedDate, GroupNumber, ProcessName
INTO TempResult3_GetGroups
FROM TempResult2_GetGroups Outside
```

Not All Google Searches Are Created Equal

When researching a topic on Google, focus on the keywords. You can give five developers the same challenge on a puzzle that they've never seen before. All five might search Google differently. Try to identify patterns. Keywords are critical. That's why they are called KEY words.

Finally, the end of the journey! I can use the DATEDIFF function to get the elapsed days for each phase, which gives the final result shown in **Figure 7**.

```
SELECT *,
    DateDiff(d,ProcessStartedDate,
             ProcessEndedDate) + 1 as NumDays
FROM TempResult3_GetGroups
```

	ProcessDate	ProcessName	ProcessNextDay
1	2017-10-03	Process A	Process B
2	2017-10-07	Process B	Process C
3	2017-10-08	Process C	Process A
4	2017-10-11	Process A	Process D
5	2017-10-13	Process D	Process A
6	2017-10-15	Process A	Process D
7	2017-10-16	Process D	NULL

Figure 4: Rows where processes are about to break (e.g., 2017-10-03 is the last day before A turns to B)

Listing 1 contains the full query, without the use of the temporary result sets.

At that point, I thought I was done. I gave myself an "A" for effort but a "D" for not correctly recognizing the actual pattern of data at the beginning. The more you can recognize, define, and apply patterns, the more effective you'll be.

> The more you can recognize, define, and apply patterns, the more effective you'll be.

Another Approach:

After I came up with a solution, someone alerted me to a pattern that I freely admit I completely overlooked: a pattern known as "gaps and islands." The great SQL author Itzik Ben-Gan (http://tsql.solidq.com/) has covered this pattern

	ProcessStartedDate	ProcessEndedDate	GroupNumber	ProcessName
1	NULL	2017-10-03	1	Process A
2	2017-10-04	2017-10-07	2	Process B
3	2017-10-08	2017-10-08	3	Process C
4	2017-10-09	2017-10-11	4	Process A
5	2017-10-12	2017-10-13	5	Process D
6	2017-10-14	2017-10-15	6	Process A
7	2017-10-16	2017-10-16	7	Process D

Figure 5: A key intermediary result set: lining up the start/end date for each process phase

Solving It Yourself versus Research

Sometimes developers are criticized because they'll try for hours to find a solution that they might otherwise (possibly) find much sooner on a technical blog. This is always a difficult call: yes, you might get the answer quickly, though you might miss out on some "elbow grease." Experienced developers are called experienced in part because they've accumulated a large amount of that very "elbow grease."

	ProcessStartedDate	ProcessEndedDate	GroupNumber	ProcessName
1	2017-10-01	2017-10-03	1	Process A
2	2017-10-04	2017-10-07	2	Process B
3	2017-10-08	2017-10-08	3	Process C
4	2017-10-09	2017-10-11	4	Process A
5	2017-10-12	2017-10-13	5	Process D
6	2017-10-14	2017-10-15	6	Process A
7	2017-10-16	2017-10-16	7	Process D

Figure 6: The dates are lined up and I can add a column for the elapsed days.

Listing 1: My first attempt

```
;with TempGroupingsCTE as
    (SELECT DATEADD(d,1,LAG(ProcessDate,1) OVER
       (ORDER BY ProcessDate)) as ProcessStartedDate ,
      ProcessDate AS ProcessEndedDate ,
      ProcessName, ProcessNextDay
   FROM (SELECT *, LEAD(ProcessName, 1) OVER
         (ORDER BY ProcessDate) AS ProcessNextDay
                    FROM ProcessRows ) Temp
     WHERE ProcessName <> ProcessNextDay
    OR ProcessNextDay IS NULL),

IntermediateResultCTE AS
    (SELECT ISNULL(ProcessStartedDate,
              (SELECT MIN(ProcessDate) FROM ProcessRows Inside
                 WHERE Inside.ProcessName = Outside.ProcessName))
      AS ProcessStartedDate,
      ProcessEndedDate, ProcessName
  FROM TempGroupingsCTE Outside )

SELECT ProcessStartedDate, ProcessEndedDate, ProcessName,
   DateDiff(d,ProcessStartedDate, ProcessEndedDate) + 1
  as NumDays
FROM IntermediateResultCTE
```

in his books. If you search his name and the keywords "gaps and islands," you can find many online references where he (and others) have covered this pattern. Here is one such link: http://www.itprotoday.com/microsoft-sql-server/solving-gaps-and-islands-enhanced-window-functions.

The "gaps and islands" pattern covers real-life business process scenarios with a range of sequence values, breaks in sequences, and gaps/missing values. Islands are essentially unbroken sequences, delimited by gaps. My example is one that falls under the gaps and islands pattern, where a traditional MIN/MAX/COUNT/GROUP BY won't suffice. The pattern uses different combinations of the LAG/LEAD and ROW_NUMBER functions across different groups/partitions. Sometimes it can be difficult to truly absorb the application of the pattern, until you face an actual example with your company's data.

Let's go back to the original result set, which I show again in **Figure 8**. In Section 1, I show each day and process and the SQL Server ROW_NUMBER function to generate a unique row number by day. Then in Section 2, I re-rank each row by Process and Day, so I've temporarily ordered and ranked the rows to show days one through eight for Process A, even though one set occurred from 2017-10-01 to 2017-10-03, another set occurred from 2017-10-09 to 2017-10-11, and then a third set from 2017-10-14 to 2017-10-15. Each Process and Day now has a second ranking value.

In Section 3, I list the rows back in the original order of day, but then I subtract each row's value from Section 2 from each row's value in Section 1. The resulting number itself is meaningless, but it represents a value that I can

Listing 2: An adaptation of the "Gaps and Islands" pattern

```
;with TempCTE as
     (SELECT *, ROW_NUMBER() OVER (ORDER BY ProcessDate) -
                ROW_NUMBER() OVER (PARTITION BY ProcessName
                                   ORDER BY ProcessDate)
      as GroupNumber
     FROM  ProcessRows)

SELECT MIN(ProcessDate) as StartDate,
       MAX(ProcessDate) as EndDate,
COUNT(*) as NumRows, ProcessName
FROM TempCTE
Group by ProcessName, GroupNumber
order by StartDate;
```

	ProcessStartedDate	ProcessEndedDate	GroupNumber	ProcessName	NumDays
1	2017-10-01	2017-10-03	1	Process A	3
2	2017-10-04	2017-10-07	2	Process B	4
3	2017-10-08	2017-10-08	3	Process C	1
4	2017-10-09	2017-10-11	4	Process A	3
5	2017-10-12	2017-10-13	5	Process D	2
6	2017-10-14	2017-10-15	6	Process A	2
7	2017-10-16	2017-10-16	7	Process D	1

Figure 7: The final result set!

Section 1: straight row numbers			Section 2: Partitioned Row numbers by Process, Date			Section 3: re-order by date, but show the Partitioned Rows			Section 1 minus Section 3
2017-10-01	Process A	1	2017-10-01	Process A	1	2017-10-01	Process A	1	0
2017-10-02	Process A	2	2017-10-02	Process A	2	2017-10-02	Process A	2	0
2017-10-03	Process A	3	2017-10-03	Process A	3	2017-10-03	Process A	3	0
2017-10-04	Process B	4	2017-10-09	Process A	4	2017-10-04	Process B	1	3
2017-10-05	Process B	5	2017-10-10	Process A	5	2017-10-05	Process B	2	3
2017-10-06	Process B	6	2017-10-11	Process A	6	2017-10-06	Process B	3	3
2017-10-07	Process B	7	2017-10-14	Process A	7	2017-10-07	Process B	4	3
2017-10-08	Process C	8	2017-10-15	Process A	8	2017-10-08	Process C	1	7
2017-10-09	Process A	9	2017-10-04	Process B	1	2017-10-09	Process A	4	5
2017-10-10	Process A	10	2017-10-05	Process B	2	2017-10-10	Process A	5	5
2017-10-11	Process A	11	2017-10-06	Process B	3	2017-10-11	Process A	6	5
2017-10-12	Process D	12	2017-10-07	Process B	4	2017-10-12	Process D	1	11
2017-10-13	Process D	13	2017-10-08	Process C	1	2017-10-13	Process D	2	11
2017-10-14	Process A	14	2017-10-12	Process D	1	2017-10-14	Process A	7	7
2017-10-15	Process A	15	2017-10-13	Process D	2	2017-10-15	Process A	8	7
2017-10-16	Process D	16	2017-10-16	Process D	3	2017-10-16	Process D	3	13

Figure 8: Rank each day by date, then by process date, then subtract one ranking from the other

ProcessDate	ProcessName	RowNumOverAll	RowNumberByGroup	Offset_GroupNumber
2017-10-01	Process A	1	1	0
2017-10-02	Process A	2	2	0
2017-10-03	Process A	3	3	0
2017-10-04	Process B	4	1	3
2017-10-05	Process B	5	2	3
2017-10-06	Process B	6	3	3
2017-10-07	Process B	7	4	3
2017-10-08	Process C	8	1	7
2017-10-09	Process A	9	4	5
2017-10-10	Process A	10	5	5
2017-10-11	Process A	11	6	5
2017-10-12	Process D	12	1	11
2017-10-13	Process D	13	2	11
2017-10-14	Process A	14	7	7
2017-10-15	Process A	15	8	7
2017-10-16	Process D	16	3	13

Figure 9: Calculating the overall row number and the row number within a group

Always Keep at Least One Foot in the SQL Coding Pool

If you're a musician, you know the value of practicing and playing. If you're a writer, you want to write regularly. The same thing applies with being a successful database developer/architect: Keep at least one foot in the SQL coding pool!

use to determine each consecutive set of days on which a specific process occurred. If you remember from my first solution, this is the key to solving the problem. In this case, NOW I can grab the minimum and maximum values for each group, and count the number of days in the group!

As you saw from my first solution, one of the keys was identifying the groups/breaks. The gaps/islands approach identifies groups differently, by using the ROW_NUMBER function first for the overall set of rows, and then within a specific process (**Figure 9**):

```
;with TempCTE as
SELECT *,
     ROW_NUMBER() OVER (ORDER BY ProcessDate)
         as RowNumOverAll ,
     ROW_NUMBER() OVER (PARTITION BY ProcessName
         ORDER BY ProcessDate) as RowNumberByGroup,
     ROW_NUMBER() OVER (ORDER BY ProcessDate) -
       ROW_NUMBER() OVER (PARTITION BY ProcessName
              ORDER BY ProcessDate)
      as Offset_GroupNumber
FROM  ProcessRows
```

Once I've built a temporary result set, THEN I can grab the MIN/MAX values and group by each Process Name and Group Number. Note that the math of the group number isn't relevant: The value of 5 or 11, in itself, has no meaning. It's just that EVERY ROW in that sequence has an overall row number and a group row number whose difference is the same. That's the key to this particular pattern!

```
SELECT MIN(ProcessDate) as StartDate,
       MAX(ProcessDate) as EndDate,
       COUNT(*) as NumRows, ProcessName
FROM TempCTE
Group by ProcessName, GroupNumber
order by StartDate;
```

Which Approach is Better/Faster?

After I saw the gaps and islands pattern, I was a little embarrassed that my solution, although functional and reliable in this situation, was more verbose and arguably less elegant than the gaps and islands approach with the rownumber offset calculation. But is it faster?

As it turns out, not necessarily. The SQL Server ranking functions come at a bit of a cost, as the cost of the second solution is higher than the first, and a review of the time and IO statistics show that the second solution is a few milliseconds faster. However, the elegance and shorter code of solution 2 might still be worth it.

A Final Word: The "Row-by-Row" Approach

Database veterans will tell you (and rightly so) that set-based approaches are the best, and that row-by-row approaches usually won't scale. There's a classic argument in SQL Server about the use of cursors. As a general rule, the practice of using cursors is discouraged, for good reason: They can be very resource-intensive, slow, and can generate resource errors if you abuse them. It can be very tempting to revert to a cursor and handle the logic here "row-by-row." Given that my situation dealt with a huge number of rows, cursors were never a viable approach. There's usually going to be a set-based pattern to apply to your situation; the more you use them and research them, the more you can leverage them in the future.

Final Thoughts:

I hope I've provided some information to help you in building SSRS reports. In future articles, I'll continue to show different SSRS power tips. As a SQL Server/Business Intelligence contractor/consultant, I've always found that working near the report layer helps me to understand the breadth and depth of the client's business application. Users always want functionality that the product doesn't provide out of the box, so the value of SSRS depends on whether the tool provides enough hooks or open architecture for developers to extend it. Overall, SSRS does a very good job here.

Kevin S. Goff

GrapeCity | Developer Solutions

ComponentOne
Studio for Xamarin

FlexGrid for Xamarin: Get the Industry's Best Native Mobile Data Grid

Deliver great enterprise apps with ComponentOne's FlexGrid, a full-featured, lightweight, high-performance data grid for Xamarin.Forms, Xamarin.iOS, Xamarin.Android, and Xamarin.UWP

Customize your grid with cell factories, including conditional formatting and embedding other controls

Design responsive apps faster with built-in star-sizing, auto-sizing, and adaptive styles for all devices

Improve performance with CollectionView's incremental, on-demand loading and pull-to-refresh

Deliver intuitive user experiences with built-in animation and gestures

Studio for Xamarin also includes 70+ charts, input controls, gauges, and a calendar

Download your free trial at **GrapeCity.com**

A Good Idea is Just the Start

When the dust settles, it's the execution that matters. At Rocksauce Studios, the Digital Design and Development Agency I founded in 2010, we have these words spelled out big and bold on our conference-room doors. We want every new entrepreneur to read them, learn them, and live them. Who is this article for? Any developer who has an idea for his or her own software, but needs some guidance on tackling all of the steps that happen before the development stage. That may not be you today, but it may describe you tomorrow. At the very least, every developer can gain some insight into the why and how of things that occur during the concept, planning, design, and redlining stages of a project.

Q Manning
q.manning@rocksaucestudios.com
www.rocksaucestudios.com
www.twitter.com/qmanning

Q is a father, husband, designer, writer, film director, and builder of products. He's the co-founder and CEO of Rocksauce Studios, and co-founder ProductionCliq.

Q has been building products since the dotBomb tech boom. After a brief excursion into filmmaking, Q returned to the tech world and started building apps and platforms for over a hundred clients, including Bechtel, Dosh, IBM, and others.

These stages help you determine if you have a good idea, plan what you're building and how, then define what the look, feel and branding are that appeals to your target audience. This way, once development starts, the engineers can focus on building the best product possible.

So, let's get the hard part out of the way first: Your idea isn't worth much to anyone but you. In fact, you can't even copyright an idea; you can only copyright certain aspects of the implementation of that idea. Unless you want someone else building a better widget, you need to put a lot of your effort into how the idea will be implemented in a way that will knock the socks off of your intended users.

> You need to put a lot of your effort into implementing your idea in a way that will knock the socks off of your intended users.

Every day, someone thinks that they have the Next Big Thing gestating and growing in the back of their brains. Most don't stop to consider whether or not anyone else cares about their concept. Even fewer have thought anything more profound than, "It would be great to have a piece of software that did X!"

These dreamers surely aren't thinking about log-in screens, resetting passwords, or what happens if a user loses data connection for some reason.

"But," you croak. "I'm a developer! I know how to make software!"

To that, I say, "You know how to code software, but you may not know how to design things so that people will care enough to download it in the first place."

Lately I've realized that I'm an old man in the technology industry. The year 2017 marked 20 years that I've been building software, from design to code and back again. I taught myself PaintShopPro as the dotCom bubble began its initial ascension, and I learned HTML and CSS to make my ideas come to life. My aging mind has learned (and forgotten) ASP, ColdFusion, ActionScript, PHP, JavaScript, jQuery, Sass, and too many others to count.

My point is that when you read this, you aren't just listening to some hoity-toity graphic designer telling you how they think things should work. The advice I give comes from a guy who's helped create hundreds of different pieces of software, for all types of platforms. I've rolled with the punches that the industry and users have thrown me.

I'm battle-hardened from the wars of designing everything to be pixelated to match Windows95. My battles were fought on the shores of transitioning to rounded, bubbly fun when XP and OS X grabbed hold of the world. And I light candles in a vigil to respect the hundreds of hours designers spent creating skeuomorphic interface elements, only to be usurped by the trend of flat color and nifty-fonts.

One thing, however, has remained the same: User delight defines successful software. A good idea is essential, sure, but the software world is littered with the corpses of great ideas about which users didn't care. Before Facebook, there was MySpace, and before that, there was Friendster. Why is Facebook king? It focused on great features, delightful interactions, and it never stopped iterating. If new technology needed to be created to keep up with users, so be it.

Don't fall into thinking that money is what makes the difference. Back in 2012, Tinder came into a crowded dating market with literally hundreds of competitors and did so without tens of millions of dollars.

What made Tinder a hit? It focused on a pure prospect: both participants needed to opt-in, and Tinder created a ridiculously gratifying interface of swiping left if you found a person unattractive, and swiping right if you saw that person as desirable, as you can see illustrated in **Figure 1**. Today, the term "swipe left" is ubiquitous for finding someone unappealing.

Tinder didn't need Facebook's money to completely change the dating or user interface landscape. Focus on a simple idea, with a distinctive user-tested interface, made them an overnight success.

Engineers solve problems based on functional requirements, but legendary software solves problems based on user behavior and creating audience satisfaction. Great software needs great development, but development alone and $2.00 will buy you a cup of coffee, and that's about it. Everyone reading this article has felt the pain of using a solution created by engineers who only tapped into other engineers as their test audience.

So, do engineers just give up? Just tip their hats to the designers or user experience folks of the world and wait for their marching orders? No, engineers do what engineers do—break down the problem into specific parts, approach the situation systematically, tackle each one, then move to the next, until a solution exists.

Defining a Sample Project

Explaining this whole process works much better when I refer to something specific. Let's define a fake project to fuel the examples. **Table 1** breaks down the details for the sample product.

Why This Project?

Everyone has familiarity with social media of some sort, so chatting about specifics on this project will make sense to most people. As for the name, SocialFeedReader is straight-forward and says what the product does. SocialFeedReader fits the personality of the type of users I see using this project. Plus, I like it.

Are You a Good Fit?

Just because you had a bright idea about something doesn't mean you want to live and breathe it for the next five years. Do you fall asleep thinking of this solution? Does your mind drift to it while you're in meetings or conversations? What makes you the right person to create this solution?

If yes, then you're in a good position. If no, you have to ask yourself, "Why am I trying to build this product?" Your lack of passion will show up in the software, sabotaging its success potential from the start. Instead, why not find an idea you actually want to suck up every minute of the next few years?

Creating your own NBT (next big thing) isn't the same as working for someone else. You've had a satisfying career in building software for other people. Building tech has driven you in the past, even if you didn't buy into the vision or love the end-product. Now it's time to build your own.

Creating a piece of popular software is no different than creating a restaurant, a bar, a nursery, or any other business. Long hours are needed. Sacrifices are made. Failure comes way before success, and you may want to quit along the way at least once, if not for long stretches of time. Perseverance and a good solution are what gets you through the insanity that comes with running a software startup.

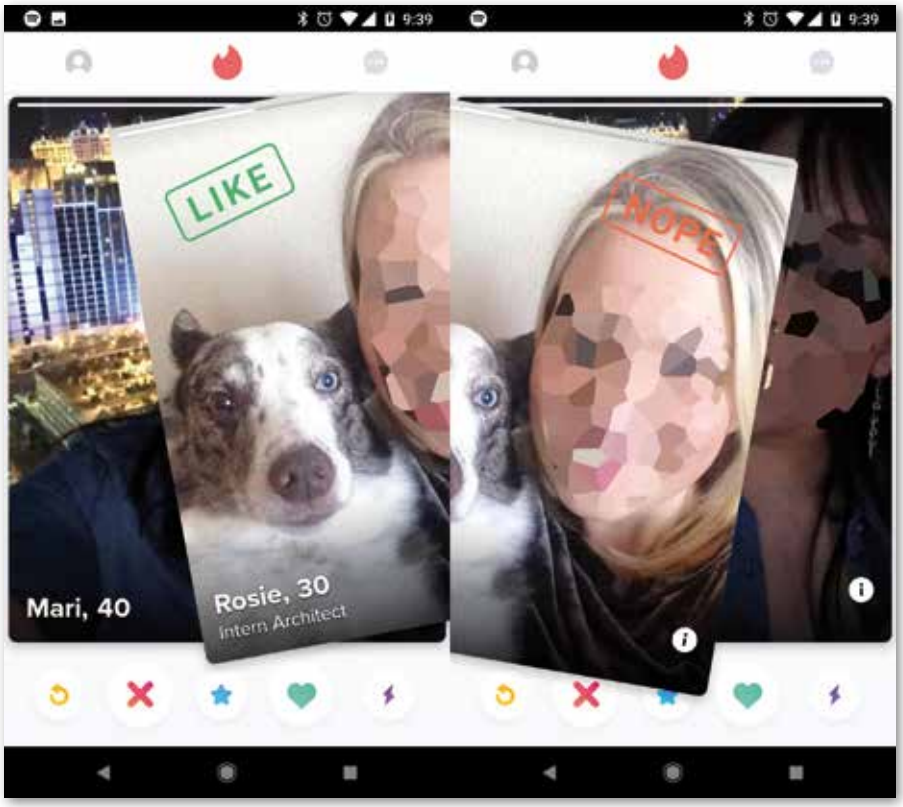


Figure 1: Tinder's ingenious interface changed dating and pop culture

This sample social media app falls right into my wheelhouse as an avid social media user, early-adopter of software platforms, and general tech enthusiast. Running daily visions for the NBT in social media would suit me fine.

Strategy and UX

Once you have a plan, all you have to do is work out the details, right?

Your goal is to build an MDP (minimum delightful product) here. An MDP is required to make a project useful and compelling to users. Anything more than this creates feature bloat, scope creep, and pushes success further out. An MVP (minimum viable product) is the bare minimum that a project needs to be useful. It's important to know the difference.

The Homework Phase of Every Successful Project

If you start developing now, with only the idea and none of the facts, you'll be the only user defining what the

Detail	Answer
Project Name	SocialFeedReader
Project Type	Unnamed Social Media Text-to-Voice and Reply
Platform(s)	iOS, Android, with eventual Web
Audience	People who want to use social media while doing other things
Logline/Summary	SocialFeedReader is a service that lets people connect their social media accounts (Twitter, Facebook, LinkedIn, etc.), and have text-to-voice read-aloud social posts in real-time while using a voice-driven interface to allow for voice responses in return

Table 1: Define the sample project, called SocialFeedReader

project should be. No great piece of software was built in a tunnel, no matter how magical a confident CEO or head of development may appear. Memorable, long-lasting, popular solutions do one thing: They solve a problem that needs solving on a large enough scale.

Three questions serve as the foundation for your application. Here they are:

- What's the problem that this software will solve?
- How does my solution solve the problem better than other apps?
- Does the problem need solving often enough to drive continued usage?

Do these seem obvious? You'd think so. However, millions (billions?) of investor dollars could have been saved over the last 20 years had the innovators behind failed software taken the time to answer these questions. Every day, this is what I ask of burgeoning entrepreneurs. Rarely do they have answers. Mostly, they thought up a neat idea, then never thought to try to pick it apart.

Questions like these don't just keep bad projects from starting, they help great projects get created. Facing reality on the validity of what you're attempting to build might kill off bad ideas, but it saves heartache, time and money.

Be honest with yourself. Sometimes there's a problem that can be solved, but solving it won't help enough people enough to generate a decent ROI (return on investment). An integrated EchoSign or Adobe Sign solution that only helps rural veterinarians who focus on rodents smaller than a Guinea Pig may be a valid solution for someone to design. But the problem and the audience are too small to be useful to enough people to validate the investment needed to build the software. That's great for a lazy Sunday afternoon's coding endeavor, but it's not the type of thing that gets Mark Cuban to invest.

Okay, so now you've identified a problem that needs solving, you have an exciting way to solve it, and you think the problem can be solved frequently enough to be a viable piece of software.

What if someone already beat you to the punch, and you just don't know about it?

Applying the Questions to SocialFeedReader

Let's look at some sample answers for the SocialFeedReader project.

- **What is the problem this software will solve?** Social media currently requires active ingestion, making it non-viable for multi-tasking.
- **How my solution solves the problem better?** Currently, social media ingestion is active, requiring site or app visits, and active reading, active scrolling. Moving this intake to passive means the act becomes a multi-tasking item, similar to listening to the radio or a podcast, rather than something requiring full attention.
- **Does it need solving often enough to drive continued usage?** There are potential strong use cases for athletes, office workers, drivers, or others who want to experience social media updates while performing other activities

SocialFeedReader answers two out of three of these, with a high probability to be three out of three. Not bad. If an entrepreneur came to me with this idea, I would give an enthusiastic thumbs-up to the concept.

You Can't Beat 'Em If You Don't Know 'Em

Entrepreneurs occasionally have ideas that already exist, but they haven't done the research. Maybe they relied on Apple's terrible search algorithm or they searched on only the most obvious keywords, unaware that a vernacular and jargon is already defined for the industry they hoped to tackle. Whatever the reason, in 2018, most problems already have a solution in some way or another.

Your job is to prove the solution. Other software is out there at this moment, solving the problem you thought was being ignored. Maybe hundreds, thousands, or millions of people use it each day. Perhaps they love it, and already have that software's logo emblazoned on the back windshields of their cars, next to those little proxy figures of their children and animals. Brace yourself, but there might even be annual conferences held each year for users to meet up, learn tips and tricks, and socialize about how this software changed their lives.

Some problems may have dozens of existing software competitors of which you aren't aware. How many CRM competitors like Salesforce are out there? Or Project Management solutions, like Basecamp or Asana? Even something as specific as source code versioning has multiple competitors, from GitHub to Bitbucket. Each has a fanbase, and each swears that this solution is the end-all, be-all way the problem should be solved. Knocking these competitors out of their top spot is the goal. Another approach is having a completely new idea.

To prove the solution, you have to become intimately familiar with the existing solutions. Be a spy to learn everything you can about how each answers the issue.

- Go to every screen and take screenshots. Understand them.
- Try to accomplish the goals you want to fix with your own solution. Note whether or not you can.
- Keep a spreadsheet of features so you know who's doing what.
- Look at their pricing models because they are who you'll need to go up against.
- Define who the market leader is and how much marketing/advertising they're doing on a regular basis.
- Ask yourself, "Is my solution so kludgy that it would be a better plug-in for existing software, rather than a competitor?"

Data is gold. Intimate knowledge of what's out there is vital in making sure you're creating something the market won't only support, but will decide to use instead of what's already out there. Competitors have at least one advantage: they're already on the market. Concerns about validity, UX direction, user personas, branding, UI paradigms, and other questions that you need to ask

have already been answered to your competition's satisfaction.

Get to know what they're doing well and what they're doing wrong. Then ask yourself, "Does my solution solve the problem better?"

If "No" Is the Answer...

It happens. Here's what you do.

- **Take a moment to mourn:** Be glad you chose to investigate the market before spending six to twelve months creating a solution that would be inert once it launched.
- **See if a new solution comes to mind:** Mediocre or partial solutions sometimes inspire better answers. You're now aware of what the competition is doing. Your old idea may be inadequate, but regular usage may encourage a new solution that doesn't already exist. With that, you can move on to the next steps.
- **Take your solution to a different problem:** Great software is sometimes just taking what worked for one industry, and applying it to another. Your answer may work for an entirely different industry or a different concept. Think of how frequently you see the "swipe left" paradigm in software that has nothing to do with dating.
- **Go back to the beginning:** Release yourself, because some ideas just don't get out of the starting gate. Life is full of hundreds of problems, however, so now that you know the formula, find a different problem/solution pair to tackle.

If "Yes" Is the Answer

How exciting!

- **Take a moment to celebrate:** Whether you were aware of the market or not, you came up with a great way to solve a problem that others seem to share.
- **See what else needs solving:** Use the expertise you've gained in this particular software market, and see what additional problems need addressing. Keep track of them, and see if they pass the same tests as your initial solution and mesh nicely with other features. If so, then you've now grown your software's potential further.

Awareness of your competition is tackled. Validation of your approach is defined. Your next step is to understand who your users are, and what they want.

SocialFeedReader Competition

Competition in the text-to-voice arena is nearly non-existent. A handful of homegrown apps or plug-ins attempt to make the concept work for Twitter, although Facebook is a different story entirely.

As **Figure 2** shows, competition for this product was tough to find, and what existed wasn't pretty. Reader for Twitter is poorly designed, uses old Android OS user experiences, and functions poorly.

No competition can be a bad thing. Typically, this is a critical indicator that the technology may not be feasible, creating what I call a "Magic Unicorn" situation. If this were a real project, the next step is creating a "Proof of Concept" to verify that the technology (text-to-voice engines, Social Media APIs, etc.) supports the app's goals.

Perhaps the reason there's no competition is simply because no one wants the solution. This is very likely, and the reason why user research is so vital to creating a great product.

For this article, I'll continue forward, ignoring the Magic Unicorn technology required to bring the app to life.

No Software Works for Everyone

User interfaces for games don't work for CRM solutions. Dating apps are bright, colorful, and full of images of different people, but that approach fails for a project management solution in the food industry.

You have to know your audience. Smart approaches to a problem don't mean much to users when they can't figure out how to operate the software. Pinpointing the types of people who'll be using the application gives you a reference point to keep features in check and goals in mind.

User personas shouldn't be based on imaginary users. Base user personas on real data, culled from the types of people who'll be using the product. Eventually, you'll decode all of the user research and use that to define a few vital types of users who will use the software.

Typically, you want between two and five user personas. More features and capabilities mean a more extensive selection of user types. Fewer features mean defining fewer ideal users.

Figure 2: Reader for Twitter, a potential SocialFeedReader competitor, is an outdated Android app that uses text-to-voice to read various feeds.

ID	Questions	Follow-up
1	Name	None
2	Occupation	None
3	Gender	None
4	Income	None
5	Operating System/Type of mobile device commonly used	Any reason why?
6	Do they have the problem you are looking to solve?	If no, define why not?
7	Are they currently solving this problem with software?	If yes, move to question 9
8	If no, why not? What prevents them from solving the problem?	Do they wish they could solve?
9	What software are they currently using to solve the problem?	What made them choose it?
10	What do they like/dislike about their current solution?	What could it do better?
11	Have they tried an alternative solution?	If yes, what? Why didn't they stick with it?

Table 2: Sample User Interview Questions

Audience Interviews

Identify who has the problem. Is it nurses? Construction professionals? Filmmakers? Stay-at-home dads? Teenage girls? Who's the audience that needs your software? Identify as many as you can, but be realistic: Facebook may work for 87-year-old grandmothers and 17-year-old teenage boys at the same time, but it's taken a decade and billions of dollars to get to that point. Don't try to tackle everyone, everywhere, right out of the gate. Take a glance at **Table 2** for a list of sample questions.

With this set as a basis, you have a good foundation to break down the need for the solution, to see what people are currently using, what works/doesn't work about it, and why they may not have changed to a different solution if they are unhappy with their current choice.

Don't ask about your specific solution to the problem. People want to be nice and will gladly tell you they can't wait to use what you're coming up with, even when they have no intention of doing so. I've never had an entrepreneur tell me, "I asked people if they liked my idea, and most didn't, but I'm going to make it anyway."

The purpose of the questions is to see what problems need better solutions, not confirmation bias of what you've already planned to build.

Find People to Interview

Users come from everywhere. Dozens of services exist that can hook you up with people to toss your questions at. Some are better than others, but generally, these can be helpful. Industry-specific niche solutions may have a harder time getting quality information. Consumer apps that target broad groups will have more success with these groups.

If you want to discover and vet participants, here are a few places to get started:

- Trade Organizations tied into specific industries
- Local networking for the sector or group
- Twitter/Facebook groups for the industries
- Friends and Family
- CraigsList ads, offering a gift card or other carrot to participate
- Depending on your solution, "person on the street" interviews could work.

Find your participants, ask your questions, and get your information. Be affable, conversational, and, as engaging as possible. Granted, this may not be your strong suit, so bring in someone else to handle this piece for you, if you need to.

Investigate the Data

With user interviews done, you need to parse through the data and find the items that stick out. The goal of this is to see specific requests that users have, things they love or hate, and what type of people are currently using the solution. This group of people is your demographic, and your job is to create a solution that works for them more than anything else.

Distill feedback into two buckets:

- How is the problem being solved currently?
- What type of people are using the solution?

The first is necessary for creating your user stories and app features, to ensure you're on the right track. The second bucket is tapped for creating the key user personas who make up your audience.

Pick Your User Personas

Through the interviews, fundamental types of users should come into focus. Their specifics could be age, gender, work environment, daily routine, education, or demographics like that. From these similarities, you should be able to pinpoint different types of key user personas that have various needs from the software.

One user may have poor eyesight and care most about the ability to print from any page. Another user may be more interested in syncing data across multiple devices, because of their on-the-go lifestyle. Whatever it may be, these are the users, and their adoption of your software is vital to success.

Usability.Gov is a great resource, with details, information, and recommendations for defining an excellent User Persona. You can find them here: https://www.usability.gov/how-to-and-tools/methods/personas.html

In **Figure 3,** you can find an example of Steven, who serves as a potential SocialFeedReader user, age 34. Ste-

ven is a composite user, built from details pulled from the user interviews.

You can see why User Personas are helpful to the UX process. Knowing who the software is for and how and why they'll use it drives the interaction decisions on the project. Without personas, engineers (or designers) wind up as the end-user, and that usually leads to mediocre software that misses the mark on creating a good user experience.

User Experience Design

Whether you're building your project using Waterfall or straight Agile, taking the time needed to put in the hard work of creating user stories, tap-throughs, and A/B testing your initial concepts is vital. Development is, after all, the longest part of the process. The goal with UX is to figure out as many problems as you can in advance, before you start figuring out which fonts work, and way before you start writing code for lists, views, or data connections.

Create a Strong Foundation

Making changes during UX can take minutes. In development, those same changes could take days, if not weeks.

User Stories: What Your Software Does

Every project has a particular set of user stories, driven by the user personas. These are the ways people use a piece of software, and here's where you start getting granular. User stories are when you see how large or small your project is. For the first time, you'll be able to break down the project into digestible tasks that can be estimated.

You've taken the time to create the user personas, and from that, individual user stories, needs, and features have come into focus. Hopefully, some ideas have also moved out of focus or into a future iteration of the project, to keep you closer to the MDP. If not, don't worry. Breaking down each feature is a sobering way to scale your project into something manageable.

User interview questions give you a leg-up on features that you'll need to have. User personas help ensure that you're solving problems for multiple audiences that may be using the solution. **Table 3** below breaks down some important elements for defining a good user story. In **Figure 5**, you can see an example of what a set of SocialFeedReader user stories would look like.

Every Single User Story You Can Think Of

You're not going to figure out every feature of your project up-front. Developers, better than anyone, know that it's the unknown that bites you later in a project. The goal with this step is to mitigate as many of those unknowns as possible. Reducing it by 25%, 50%, or even 80% is a massive gift down the line.

This process also gets rid of mediocre ideas or items that don't fit into the first version of the MDP. User stories, particularly those that are estimated out in hours or story points, help you see just how big a project is, and what may be "fluff," or un-needed features. Use this to your advantage, and start moving elements into Version 2 if they aren't vital to creating your lean-mean-MDP.

USER INTERVIEW

STEVEN | 34 | MALE

Sales Professional, $150k income

SUMMARY

Steven is an avid user of social media, frequenting Facebook, Twitter, LinkedIn, and Instagram. Using it for both personal and business, Steven has multiple accounts to help keep his different personas separate to insure his comments or views have less impact on clients. Steven has a 30-minute commute to the office each day. He frequents the gym at least three times a week, with sessions lasting at least an hour. A solution like this would help him get keep up to date, without interfering with his regularly activities.

What OS do you use?

iPhone for my mobile, Mac for my computer.

Do you have trouble using Social Media while multi-tasking?

All the time. In sales, I have to keep up with what's going on with my potential clients, in my industry, and everything else. Plus, I'm a big political junkie, so I try to keep up with what's going on in the world. I've gotten a ticket for surfing while driving before, so yeah, this is a problem. I go to the gym a few days a week, and it's tough to keep focused when scrolling Facebook. At work, it's even tougher for me.

Have you solved this yet?

Not really. When I'm at a stoplight, I can check my feed and I still scroll through stuff when I'm on the elliptical or whatever. But there's no easy solution I know of.

So you haven't found a solution?

No, there's nothing out there that I know about. I'd figure with Siri or Alexa being so smart, I could get them to read me my Twitter feed, but everytime I ask them, they give me some jokey answer. If your app can solve this problem, I'd pay for it.

Would you use an app that read your social feeds back to you?

Yeah, man! That's what I've been looking for! I mean, as long as it didn't sound too much like a robot, though. It would need to sound like a real person. When I have Alexa read me something, it sounds nice, so as long as it was that type of thing, I'd go for it.

Would responding back to things be important to you?

No, I don't think so. It would be a nice to have type of thing, but as long as I could start it and just forget about it, that would be pretty great.

What if it was a new social network? And all audio? Would you switch over?

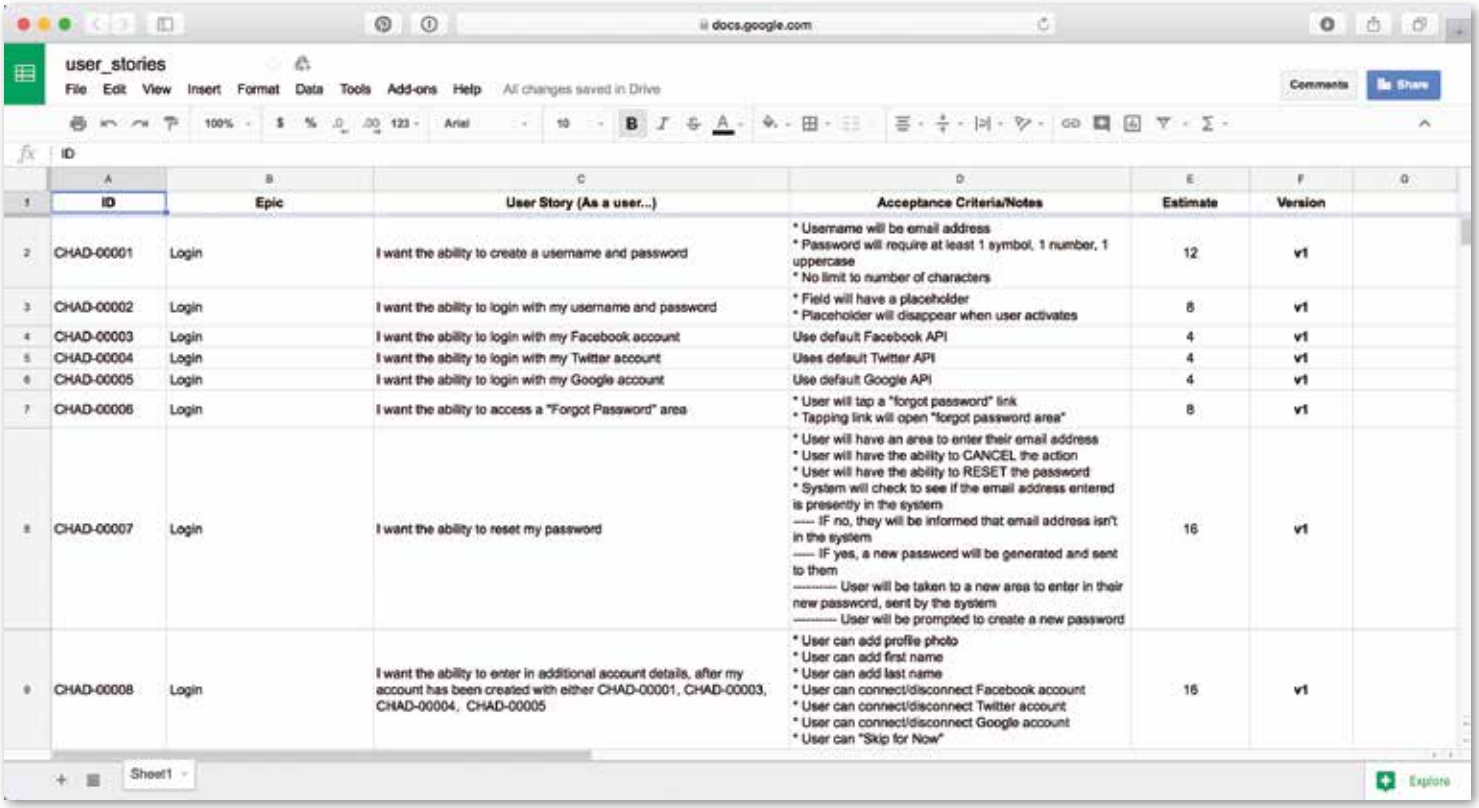
That doesn't sound very good to me. I use Twitter and Facebook because my friends are there. LinkedIn is important for me because that's where business is done, and then Instagram is where I put photos of my family. Social media doesn't work if it's just you talking to an empty room, y'know? So if it's my existing stuff, yeah, I'm interested, but I don't want to start over from scratch.

Figure 3: A Sample of a User Persona done in a faux-interview style.

You'll return to the user stories document throughout the rest of the project, to add new stories that come up. **Figure 4** shows how these user story details can work in a real-world scenario, and helps you see how granular each item can get.

Wireframing and Testing

Every screen, feature, detail, and flow of your project needs to be created first in the UX process. The end result is a digital prototype that you put in front of potential users to see what works and what doesn't. You do this during UX, before you start designing and way before development begins, to help make sure the right thing is being built.



user_stories

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

Comments Share

docs.google.com

	A	B	C	D	E	F	G
1	ID	Epic	User Story (As a user...)	Acceptance Criteria/Notes	Estimate	Version	
2	CHAD-00001	Login	I want the ability to create a username and password	* Username will be email address * Password will require at least 1 symbol, 1 number, 1 uppercase * No limit to number of characters	12	v1	
3	CHAD-00002	Login	I want the ability to login with my username and password	* Field will have a placeholder * Placeholder will disappear when user activates	8	v1	
4	CHAD-00003	Login	I want the ability to login with my Facebook account	Use default Facebook API	4	v1	
5	CHAD-00004	Login	I want the ability to login with my Twitter account	Uses default Twitter API	4	v1	
6	CHAD-00005	Login	I want the ability to login with my Google account	Use default Google API	4	v1	
7	CHAD-00006	Login	I want the ability to access a "Forgot Password" area	* User will tap a "forgot password" link * Tapping link will open "forgot password area"	8	v1	
8	CHAD-00007	Login	I want the ability to reset my password	* User will have an area to enter their email address * User will have the ability to CANCEL the action * User will have the ability to RESET the password * System will check to see if the email address entered is presently in the system ----- IF no, they will be informed that email address isn't in the system ----- IF yes, a new password will be generated and sent to them --------- User will be taken to a new area to enter in their new password, sent by the system --------- User will be prompted to create a new password	16	v1	
9	CHAD-00008	Login	I want the ability to enter in additional account details, after my account has been created with either CHAD-00001, CHAD-00003, CHAD-00004, CHAD-00005	* User can add profile photo * User can add first name * User can add last name * User can connect/disconnect Facebook account * User can connect/disconnect Twitter account * User can connect/disconnect Google account * User can "Skip for Now"	16	v1	

Sheet1

Explore

Figure 4: Example of user stories

Column Name	Description	Example
Unique ID	Seems silly to tell you this, but you need an identifier. It's much easier to cut, discuss, or modify a user story by its identifier, rather than trying to describe it each time.	CHAD-0001
Epic	Epics are collections of user stories that accomplish a goal. Think of it as a folder or container if that helps you out.	User Login
User Story	A user story is a specific task that needs to be accomplished by a user in the application. These should always come from the user's perspective, which is driven by the User Personas.	"As a user, I want the ability to use my email address as my username."
Acceptance Criteria	What defines success or failure for the user story is your acceptance criteria. If there are any specific interactions, animations, or other concepts that need to make it into development.	Email address must be validated to contain both "@" and "." characters
Dev Estimate	Estimate how many hours, or points, you think each particular epic, user story, and acceptance criteria will take to code.	5hrs
Version	In what version of the project is this feature expected? Most features start out as Version 1, but by the end of the UX process, many of these get moved into Version 2 or later as the genuine MDP comes into focus.	v1

Table 3: Simple user story template

Ideas become user stories. User stories become blueprints or wireframes. Wireframes become software with a purpose. Just as houses aren't built without blueprints, no piece of software should be either. It can't be repeated enough: Figuring out problems in the UX phase is far faster and more cost-effective than reworking development over and over.

Wireframes come in different levels of fidelity. As with audio, you have low-fidelity and high-fidelity options. The level of fidelity is determined by how closely the wireframe resembles the final product. Sketches on a whiteboard are low-fidelity, whereas designs built in Illustrator or Sketch may be high-fidelity. The closer a final wireframe looks like the end-product, the more time the UX process can save.

Typically, a project starts with a series of sticky notes with features/user stories written on them, as seen in **Figure 5**. Notes like this make it a cinch to move around the order of a project, helping you define a user path through the application. Or, if there are multiple persona types, the multiple different paths, based on user needs. Sticky notes are also great ways to A/B test these flows among potential users.

> Just as houses aren't built without blueprints, no piece of software should be either.

Not interested in A/B testing the concepts this early? Use a whiteboard or a pad of paper to get initial concepts and flows nailed down.

All of this leads to high-fidelity wireframes. These define which specific items sit on distinct pages, what buttons lead to different areas, and how much content can fit into a typical screen or view. User stories drive everything that needs to be in the high-fidelity wireframes, and the wireframes can further add to user stories.

Missing functionality or features will appear, so add those back into the user stories document. Higher fidelity mockups begin to define the exact acceptance criteria expected on various screens. Most of the time, the initial version of user stories won't have much acceptance criteria outlined. These will continue to grow through UX, and especially, during design.

Figure 5: Sticky note wireframing for SocialFeedReader

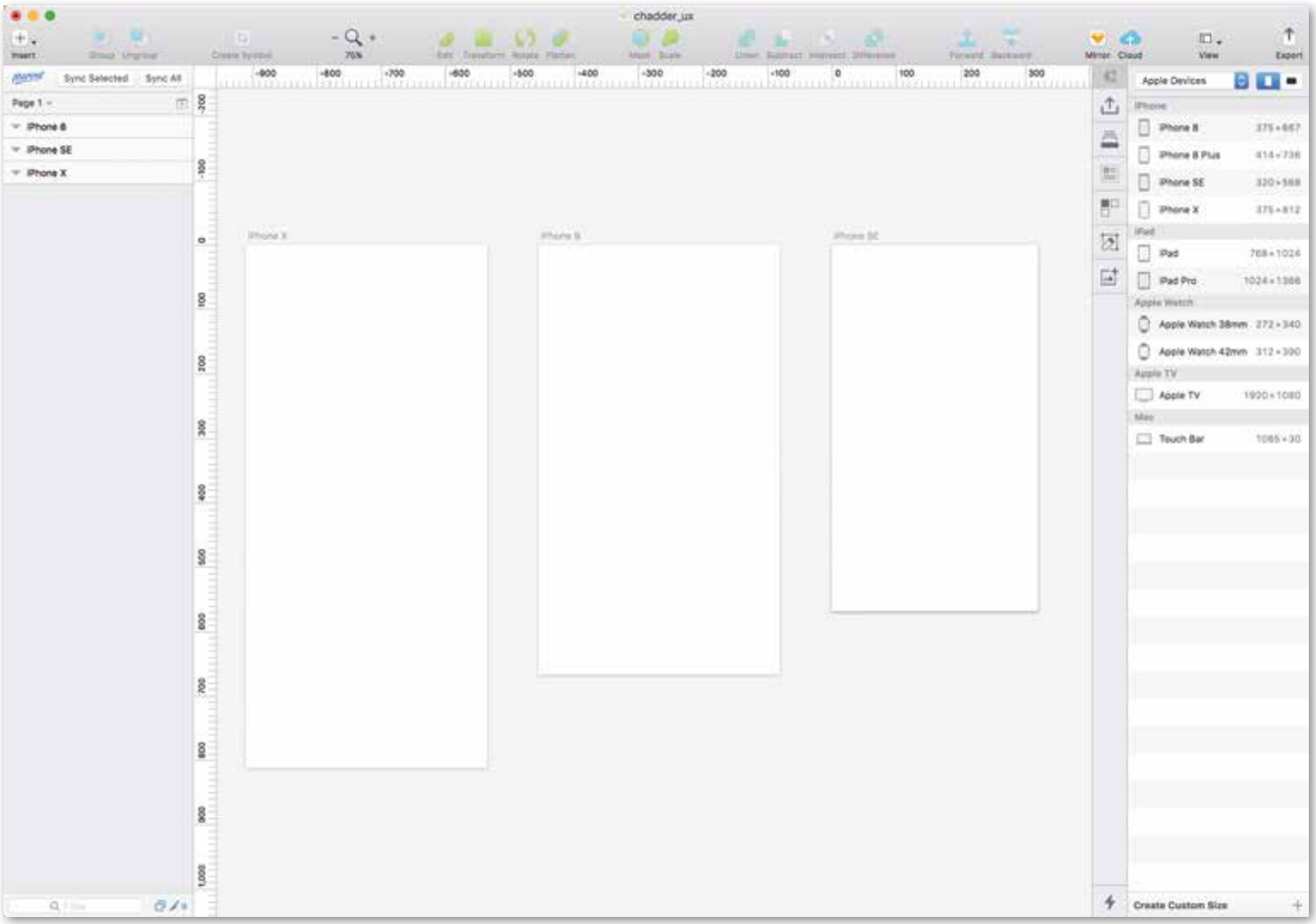


Figure 6: Sketch for the Mac has become the go-to wire framing and design software.

Figure 7: Example UX for the main feed

Keep your user interviews in mind as you're working through these first versions of your software. What did users love or hate about the competing solutions? That data is priceless when you're attempting to build innovative, usable solutions.

Interactive Tap-Through Process

A useful experience is to put sticky-notes in front of users, who then pretend to use the app. This is especially good for higher-level flows or functionality. As higher-fidelity wireframes are created, the goal is to develop interactive prototypes that can pinpoint more detailed actions.

If an interactive prototype is for a mobile device, it's called a tap-through. (When the prototype is for the Web or a desktop piece of software, it's called a click-through.)

To create these high-fidelity wireframes, I recommend Sketch if you're on a Mac, or Adobe XD if you're on a PC (although XD is on Mac as well). Both of these pieces of software have a small learning curve for design and are created specifically to design interface elements. **Figure 6** shows what the wireframe looks like in Sketch on a Mac.

With both Sketch and Adobe XD, you can create a new "page" and define which type of device you're creating for (iPhone 8, Android, Responsive Web, etc.), and immediately have the right aspect ratio in front of you. All of the tools are easy to figure out, and dozens of tutorials exist online to help. Check out **Figure 6** to see how Sketch handles setting up different screen sizes and artboards.

Here's where intuitive interface concepts come into the picture. What those are depends on the details you gleaned from the user interviews. Spots where your demographics felt frustration or confusion are ripe for an interaction shake-up. Play around with different concepts, making sure to steer clear of things that feel too gimmicky.

Magic really happens when you take those pages and push them into Marvelapp.com, Invision.com, or, if you're using Adobe XD, publish to Adobe's own Web server. Marvel, Invision, and Adobe all do the same thing: They let you connect flat images to one another as if they were real pieces of software. With these tools, you can have a "login" button that takes the user to the "logged-in" version of the application or show a user what a scrolling news feed looks like (see **Figure 7**).

Interactive tap-throughs not only make A/B testing more useful but also help creators figure out if their flows make sense in the bigger picture.

For high-fidelity designs make sure that the right screen size is used. For the Web, this can be forgiving, but for mobile apps, screen size matters. Popular software, like Sketch, lets you define the type of screen you're hoping to design for, so picking iPhone SE or Android or similar gets you the 1x version of that screen. For SocialFeedReader, handling the UX of main user feed was vital.

The 1x screen is where assets start, and during the exporting process at the end, the assets are cropped out for 2x, 3x, MDPI, HDPI or whatever sizes are needed. All of this starts with 1x sizing as a baseline, so make sure your team is getting this right.

Don't just look up a device's resolution on the Web and think designing to that will work. You'll be kicking yourself later when all of the artwork has to be resized.

A/B Testing

A/B Testing is a fancy name for the process of seeing which item/flow people like better. For best results, a large group of people are shown either A or B, give their feedback on what they used, and then the data is tabulated. Most of the time, getting enough folks available for a truly blind A/B test is tough. Instead, many test subjects are shown both A/B options and asked to provide feedback on which they prefer. (For a look at how Marvel App does this, see **Figure 8**.)

How ever you do it, A/B testing in any way is going to put you in a better spot than had you merely built the software based on your own biased uses. Making sure your

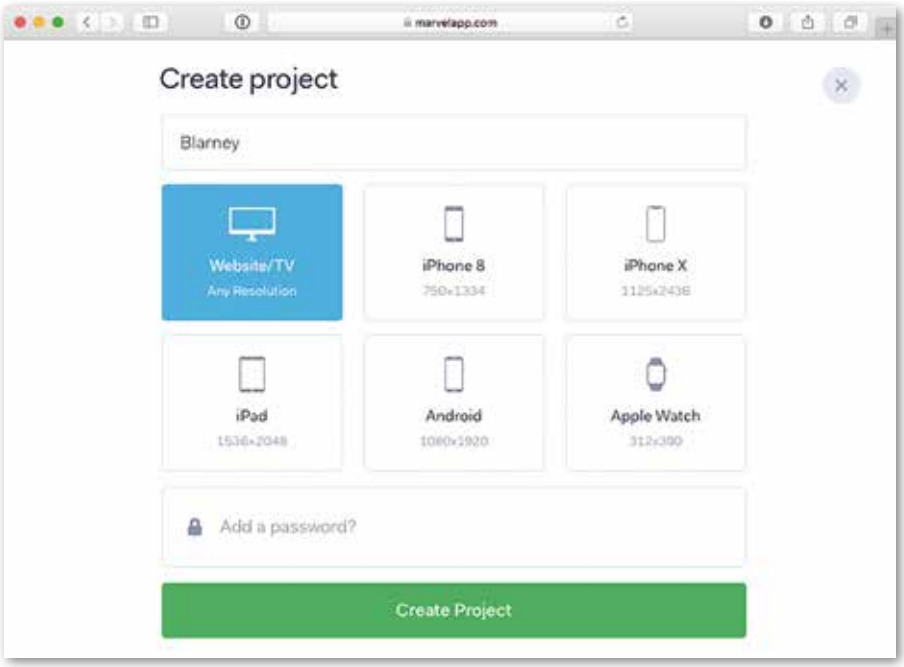


Figure 8: Setting up a project in Marvel App

project works for a wide audience can only help user-adoption of the software. That's the goal.

A/B testing for UX should focus most on making sure that users understand how things flow from screen to screen, general placement of objects, and whether or not the solution is clear to the user. To help ensure this, keep high-fidelity wireframes monochromatic in order to keep conversations away from things like color palette choices. Additionally, use the same font throughout, and only use variations of bold and font-size to differentiate items when it's useful to the UX.

Whether your monochromatic palette is shades of blue, shades of gray, or shades of green, keep it consistent. You don't want prejudice against colors or combinations to get in the way of conversations about flow or functionality.

Take a Deep Breath: The UX is done

Enjoy a moment of relief because a lot of the hard problems on the project have now been figured out. Sadly, much remains to be sussed out. Happily, the base of the software is defined in a systematic, structured way that means the project will be able to grow and pivot more easily. Far more is now known than is unknown.

The next phases of the project are defining the brand, personality, distinctive user interface visuals, and then getting everything ready for pixel-perfect art implementation during development.

Generic Software Is a Bore to Use

"It just...feels better." Most likely, you've heard this phrase when someone tried to describe why they prefer one piece of software over another. Although it may have seemed like an unquantifiable subjective viewpoint, the truth is that aesthetics, fit, finish, and polish are what drive most winners in the software world.

The software we tend to use has a strong brand, and typically, a pretty interesting personality. Usually, our favorite apps provide little moments of gratification for performing various tasks, and they have easy to read text, and, most of all, consistent visual elements to create a design language.

Developer-created software is like store-bought cereal. Kellogg's Frosted Flakes isn't any tastier than Great Value Sugar Frosted Flakes, but you know which one you'd rather eat. Maybe it's Tony the Tiger or something else, but you're drawn to the brand.

Engineer-driven applications rely on a set of checked-off features to attract users. Substantial software brands, however, have a memorable brand, a distinctive visual identity, and either a design system or brand guidelines that drive the aesthetics of the project.

Maybe you don't think it matters if your fonts all match or if the margin on rows is always the same. Think that way, and you're throwing away every hard-worked development hour. People may not be able to point out those specific problems, but they can tell you "something about this feels…cheap."

> Whether it's Facebook, Salesforce, Basecamp, or Twitter, we all want software that does things in a beautiful way.

Apple became the world's largest software company by focusing on putting user experience first, and then letting the engineering team figure out how to make it happen.

Defining Brand and Personality

Amazon isn't named OnlineBookStore, and eBay isn't called OnlineAuctionWebsite. Naming, branding, and personality come from a lot of places, but typically, it's about resonating with a particular audience.

Jeff Bezos wanted a company that would sell everything from A to Z, and the name Amazon evoked a sense of grandness. It's a bold choice for branding, but it's a lesson that we can all follow. Think of the brands you love, and most of the time, it isn't always clear what they do. Often, the name of a product becomes synonymous with the action or product, such as Googling or Tweeting something. In the year 2000, these words would have sounded like nonsense. Heck, maybe they still do.

Of course, a catchy name isn't a requirement for, or guarantee of, success. Salesforce is pretty on the nose, as is IBM or Whole Foods. So how do you start to figure some of this stuff out?

Great branding can be built from mediocre names, such as Microsoft or Gap, but often, a unique name goes hand-in-hand with an unforgettable brand, as we see with Instagram or Twitter.

Re-visit User Interviews and Competitive Analysis

During the UX phase, you did a significant competitive analysis of the current software landscape. Was there a particular visual style that seemed consistent across the competition?

If so, you can either follow-suit or choose to shake things up by being creating a disruptive brand that's unexpected.

How do you pick? User interviews and user personas should guide the way. Who is your primary psychographic and demographic? Do you need to appeal to an audience that loves the outdoors? Or is it a more unisex product where legibility for those with bad eyesight is most important? In some cases, you may need to consider playing into a specific gender stereotype because that's your primary audience. Whether it's politically correct or not, there's a reason that the sports supplement industry feels hyper-masculine; it's because marketers have determined that this is what repeatedly draws in their buyers.

Design is about communication. You can't communicate if people won't engage.

A/B Test…to a Point

Design by committee is garbage. Every designer will tell you this, and eventually, you'll have to be the ultimate decider of the right choices. The benefit of A/B testing is to help inform your decisions by giving you real feedback from potential users.

Use the A/B groups you have previously tested with, as well as new individuals who haven't been exposed to your materials before. See what direction resonates with these groups and individuals. Your gut instincts matter here, and ultimately, you'll be responsible for interpreting their advice to make sure you head in a direction that feels good for your audience.

> If you want your work to stand out, you need to pay for the skills to help it get there.

A word of caution: Avoid analysis paralysis because you will never please everyone all of the time.

Don't be afraid to be bold. Meek requests never get fulfilled. Middle-of-the-road brands that look like the rest of the competition never get noticed.

Be bold. Be strong. Be distinct. Make powerful and memorable claims in your brand statements and back that up with equally delightful visuals.

Professionals can help. If you're not handling design duties yourself, make sure to stay clear of $99 logo farms or $5-gig websites. Places like those are full of people who have a collection of stock materials that they just regurgitate for clients. Your product could be indistinguishable from all the rest, or even worse, you may end up paying for someone else's logo.

Amazing artists don't give their work away for free, any more than skilled developers would. Professionals get

Technologies for Wireframing

Which technology you use to wireframe ultimately depends on your preference. Just make sure to use an appropriate technology for the stage of the process.

Low-Fidelity Wireframing

- Sticky Notes
- Whiteboards
- Markers

High-Fidelity Wireframing

- Sketch or Adobe XD for designing wireframes
- Marvel App (**see Figure 9**) or Invision for interactive prototypes

Figure 9: Blarney logo options in a couple of different font styles.

paid for what they do. If you want your work to stand out, you need to pay for the skills to help it get there.

A talented creative partner could be the catalyst to beating the competition. Don't be afraid to give them a big chunk of equity if you can't afford a solid hourly rate. Just stay clear of insulting them with requests for free work for exposure. Exposure doesn't feed their kids and a good artist has lots of opportunities to work on projects at a substantial rate.

Trademarks and Copyrights. Hire a lawyer for this, because you'll need to do a real search and file the right paperwork. Not seeing a product on Google or the AppStore doesn't mean much of anything. You want to make sure you can own the rights to use whatever great brand you create. The worst day ever is discovering that you've spent the last 12 months writing code, and not only are there three other products just like it already on the market, but that you're being taken to court for infringement by a massive company that funded one of them.

For names, do this search early to make sure you don't base your entire company on a brand you aren't legally allowed to use. Or that someone else is using for chewing gum.

Name the Sample Project

Most people I meet with already have a name picked out for their project, and almost always, they are pretty attached to the name. For the sample project, my initial thought was SocialFeedReader.

A quick search on Google reveals that the name SocialFeedReader is used for a poorly branded social media private messaging application. So at this point I have a choice: Either come up with a new name or move forward with SocialFeedReader, knowing that I will have to compete in Google searches, that I will have to fight for social media handles, website domain, and probably a dozen other points of contention.

If I choose to move forward, that means getting a lawyer involved sooner rather than later. The existing app may not have a copyright or trademark, but they do have pre-existing technology and have been on the market, all of which plays into a court's decision.

As far as my thinking went and the name suggests, SocialFeedReader was intended to evoke a clear and simple social brand, meant for all audiences, relying heavily on illustrations of animal icons as heavy brand elements.

To keep that direction, a new name is needed. I wanted something whimsical.

"Blarney" immediately came to mind. Both fun to say and memorable, blarney means "talk that aims to charm or flatter." So, that fits the goals of the software, too. Plus, a search of Google and various app stores shows me that "**Blarney**" isn't being used by itself. Just saying the word aloud immediately caused a logo concept to materialize in my head, as you can see in **Figure 9.**

Lawyers and research are still needed to lock the name in, but I'm far more confident in Blarney than I am with mov-

Figure 10: Final user interface design concept for Blarney on iPhone X

Figure 11: Blarney's color palette is vintage and modern all in one.

Figure 12: Blarney's brand is fun and playful, mixing the concept of a tape reel and a robot.

ing forward SocialFeedReader. And, now that I think on it, I like it even more. Blarney is more interesting than SocialFeedReader, which is a descriptive name, but nothing more.

Does this naming process seem arbitrary? Yeah, of course it does. Sometimes, you come up with a great name that really fits your product, but most of the time, you're seeking something distinctive and memorable. You want to own a certain portion of your audience's mind, and the more distinct a name is, the higher that likelihood is too.

Remember not to get too attached to the first name out of your head. I've helped rename over 50 project, and it's followed a methodology a lot like the one I just showed you. A thesaurus can be your best buddy during this process.

Great Design Can Be More Important Than Any Line of Code

First impressions matter, and your product's interface is the first impression people get of your solution to their problem.

Mobile apps typically have about 30 seconds to one minute after installation to compel a user to keep using the software. Websites have even less time to impress, because tapping "back" is so simple. In-depth SAAS solutions may get a few minutes or so to make their case, but users need to see the benefit right away if you want them to keep going deeper. Most of all, people need to enjoy their experience with the solution.

UX does the heavy lifting for defining the project features, as well as the flow from screen to screen. In some places, UX will pick the size or placement for a button and it will be perfect. In other places, though, the UX will benefit from some finesse that the design phase can offer. If the UX works well, don't change it, but always leave room for design to make tweaks or changes that create a better experience.

Design is the closest the project will get to its final state before development. A good tip here is to have the designer, be that you or someone else, do their design work literally on top of the UX work. With software like Sketch or anything in the Adobe Creative Suite, the whole file or an individual element can be duplicated. Then, a designer can just lock the UX elements, and put their new pieces on top.

This makes for a great before/after comparison, as you can check in by looking back at **Figure 9** and ahead at **Figure 10**.

Photos may get bigger, font sizing can change, button radii might get modified, but overall, the goal is to let the good work of the UX phase support the design work that's needed. After all, it went through A/B testing to find a great balance, so trust that good research even if someone else suggests tossing it away.

Fix the mistakes of the competition. Issues with interfaces drive a vast majority of user complaints. Take advantage of your research to improve in spots where the competition is failing. Users can be quick to change software if it solves a problem more intuitively, even if they have invested in the ecosystems (software, user accounts, training) of competing software.

Define a consistent visual language. Software isn't always linear, but the same functionality should have the same interface. Form fields should always look the same whenever a user encounters them. Buttons shouldn't change shape or color unless there's a potent reason behind a specific instance. User profile images should always have the same border radius and be of a consistent size. The margin from the left-side of the screen should be the same everywhere in the application any time text shows up.

This consistency creates a great user experience by defining a visual language. Consistent buttons mean a user immediately knows to tap that button. Similar header treatments throughout mean a user can quickly discern body copy from header copy. Defining a color palette that can be used throughout helps familiarity, as seen in **Figure 11**.

Muscle memory patterns are magic. Everyone runs the mouse cursor over words with underlines. This is muscle memory, when the body reactively performs a series of actions, without the user putting too much (or any) conscious thought into the action.

This kind of behavior is muscle memory that's been ingrained in us since the beginning of the Internet. Muscle memory patterns help users accomplish hundreds of software tasks daily, all without really thinking about them.

Consistent design leads to muscle memory. Your job is to create a few consistent muscle memory patterns for significant actions in your software, so when a visual cue appears, your users immediately know to react. Create new muscle memory points where needed, but don't rewrite what the operating system or trendy platforms are already doing.

Pull-down to refresh is second nature. Don't make it "swipe left to refresh" just to be different. Users will instinctively attempt to pull-down, so let them. Piggyback or borrow good ideas from prevalent software like Snapchat, Instagram, Facebook, or others. If you're doing the same thing they are, there's no reason to break the mold.

With all of this, be wary of copyright infringement. Any iconography, imagery or symbols you see are brand distinctive, and you shouldn't simply copy them. Want to use a head for a User Profile Icon like Facebook does? Great, but design your own that isn't exactly the same. For icons, a great resource is The Noun Project (http://thenounproject.com), which has a lot of options you can purchase. Even your logo and branding can be influenced by the competition, as seen in **Figure 12**, but still has these potential concerns.

Some software interactions are patented or are protected by copyright, though if it exists as part of the operating systems Human Interface Guidelines (HIG) or is part of their coding standards, it most likely falls under fair use. Always double-check to make sure.

A different OS means different interactions. Web apps can be platform-agnostic in their design. Mobile software needs to take heed of what each OS does differently.

> ASMR is deployed everywhere in modern-day applications because it adds to the user's sense of things processing as they should.

For instance, iOS has a Las Vegas-style picker wheel that pops up instead of a drop-down menu. Android has a small, scrollable action sheet instead. The iOS apps tend to place the navigation along the bottom of an app. Android software puts that navigation at the top or within what's known as a fab menu (the little circle you see at the bottom of Google software, the one you tap and a new menu pops open).

Operating systems change regularly, and with it, the HIGs do as well. Currently, Material Design is a big mover in the Android world, where an object (such as a fab) turns into an overlay when you tap a button. Note that iOS doesn't use this concept, so although there are pieces of software that use it, there's a learning curve for iPhone users.

Tapping into what an OS user expects will make your app "feel right." Making an Android owner use an iOS-style picker wheel feels "odd."

Animation Frameworks Are Pretty Cool

Hundreds of animation frameworks/libraries exist that are available to your project. Choosing one can save dozens to hundreds of hours of development and give your software some polish.

ASMR (autonomous sensory meridian response) is a bit of a buzzword in 2017, but tapping into the phenomenon sets some of the most popular software parts from the competition. Think of it as a sense of gratification or satisfaction; ASMR is that pleasant tingly sensation you get from believing you have accomplished something.

Animation sequences help create these moments of ASMR in software. For better or worse, this is some of the same psychology that Facebook and others have used to create "addiction" in users. It's ASMR that makes Tinder feel so satisfying, as you complete the task of swiping left or right.

Animations are in all your favorite apps. "Favorite "something on Twitter and the little heart explodes. Swipe away an email with Google Inbox and watch a series of cascading animations as the row disappears, the rows beneath it move up, and a Material Design Toast (small notification that pops up like, well, a piece of toast) appears to let you know the action has completed, and you have a short time to "undo" that action. Open your phone or go to your favorite SAAS website and you'll see this ASMR in action.

Even the simple act of a link transition fading from one color to another can fire-off this response. ASMR is deployed everywhere in modern-day applications because it adds to the user's sense of things processing as they should.

All of these items can be created from scratch, but using an animation framework (or multiple libraries) can help designers pick specific interactions for specific actions, and have those deployed verbatim by a development team without hours of back and forth tweaking.

Some frameworks, like those based on Google's Material Design ideology, serve as a design system, a brand guideline, and as an animation framework. If Android is your destination, using one of these structures is a no-brainer. Even if your output is on the Web, you get access to some well-thought out, delightful interaction points.

Here are a few great frameworks to look up and test out:

- **Material Design Lite:** https://getmdl.io/. Official Google framework, this is a lightweight Material Design framework for the Web
- **Polymer:** https://www.polymer-project.org. Also, an Official Google project, if your output is Ionic, this is for you.

Doing it All Yourself

Whether you're taking the advice of social media Svengalis like Gary Vaynerchuk to build it yourself, or just want to see if it's possible, the challenge can be accomplished.

Just recognize that the larger a project gets, the more difficult it is for one individual to handle all of the roles on a project.

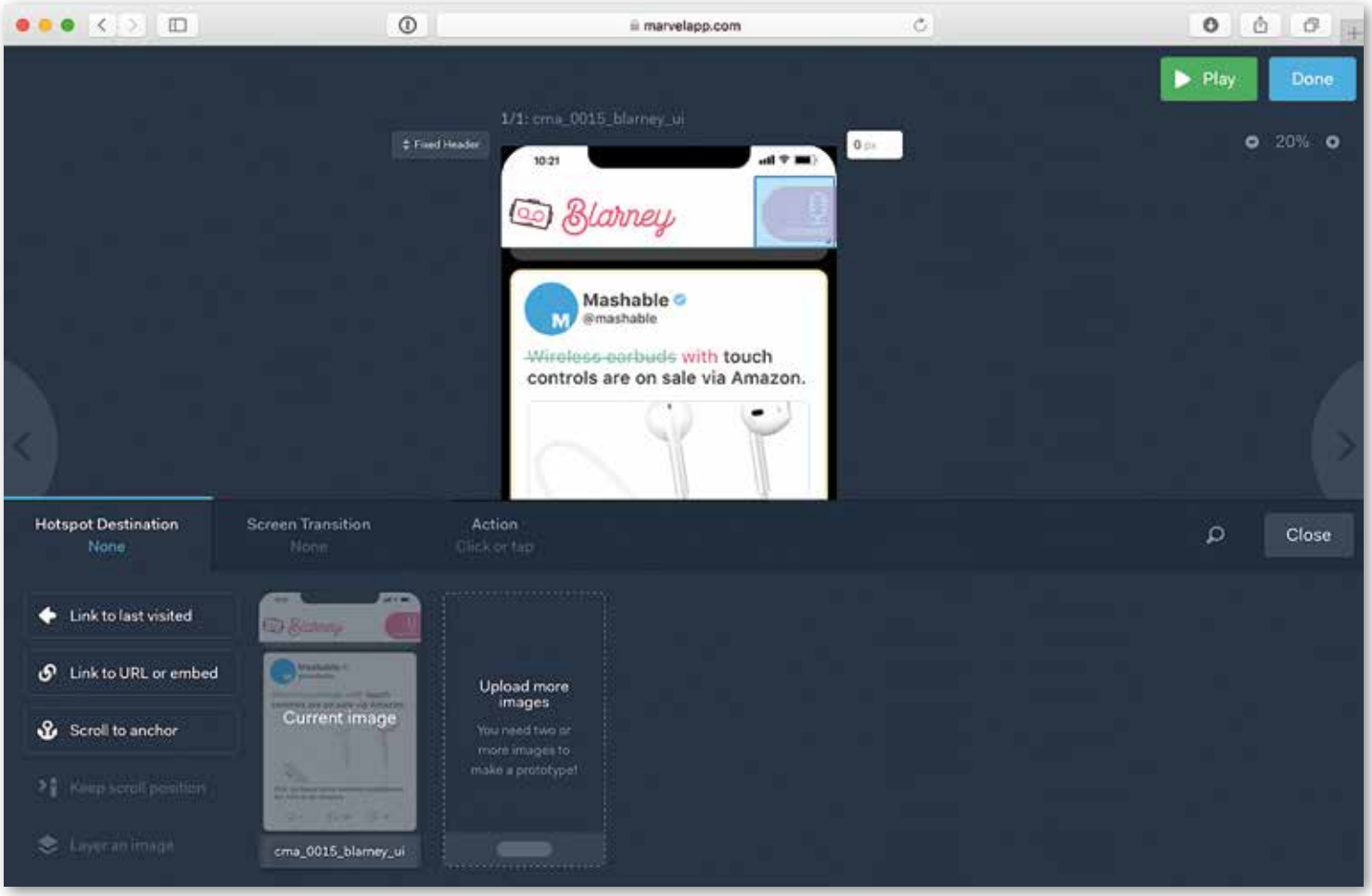


Figure 13: Creating an interactive tap-through with final art in MarvelApp.

- **Angular Material:** https://material.angularjs.org/. Maintained by a team in Google, this is great if you're using the Angular.js.
- **Animate.CSS:** https://daneden.github.io/animate.css/. A ridiculously lightweight set of CSS animations that can easily be deployed in any Web-based project.
- **Hover.css:** http://ianlunn.github.io/Hover/. A large collection of hover-effects for the Web
- **Vivus.js:** https://maxwellito.github.io/vivus/. One of my favorites, Vivus.js animates in SVG objects on the Web.
- **Lottie:** https://github.com/airbnb/lottie-ios. For the designers who want to design their own screen movements. Created by AirBNB, Lottie is a library that turns After Effects animations into animations for both iOS and Android.
- **Facebook Origami:** https://origami.design. A prototyping tool that helps you animate iOS and Android interactions and transitions.
- **Facebook Pop:** https://github.com/facebook/pop. An animation Framework that was used in the late great Facebook Paper.

All of this is just the start. Simple Google searches can find so many more examples that all you'll need to do is decide on one that can work as a shared language between the design and development team.

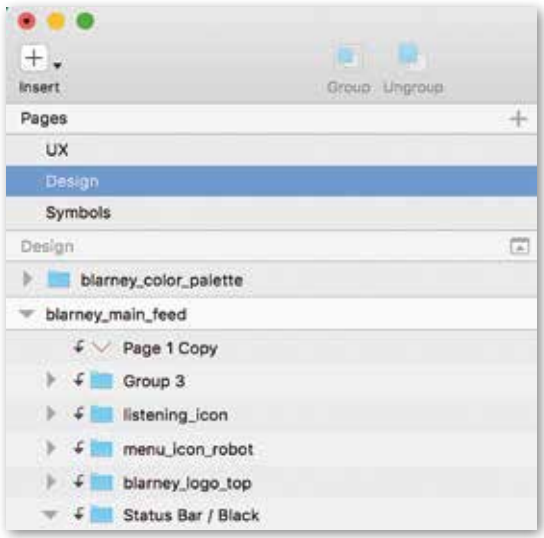


Figure 14: Asset naming conventions in Sketch.

Final Screens Get You Closer

Once the last designs begin to materialize, it's time to get another set of interactive tap-throughs created.

A good rule of thumb is to duplicate the existing UX tap-through, then begin replacing screenshots with the

final designs as they start flowing in. This way, no vital screens are forgotten during the art phase, and as a user taps or clicks through, they won't run into a dead end where something wasn't designed. UX screens can serve as a backup in case a final artwork file isn't there.

Design A/B testing should still focus on flow and placement, but with much of that figured out during UX testing, your real goal is to make sure that the choices made will resonate with the userbase. Or, hopefully, that you haven't broken any of the functionality validated during UX. **Figure 13** shows gives some insight into the Marvel App interface for creating a prototype for A/B testing.

For design A/B testing, you want to validate the changes you made that aren't necessarily UX. Anything can be A/B tested, either solo or as a series of new questions during user interviews. Some of these key items are:

- Logo
- Font
- Colors
- Tone of voice
- Margins
- Text treatments
- Overlay styles
- Tutorial Styles
- Animation implementation

Be wary, however. Design can be polarizing, and someone may detest what you show them simply based on a dislike for the color palette. Test with enough users that you can find commonality and clear and repeatable feedback.

If one user dislikes your iconography choices, that's subjective. If 75% of people take issue with your icons, you have a problem. So how do you avoid allowing this feedback turn into "design by committee?" You take in everything you learn, and be the final decision maker.

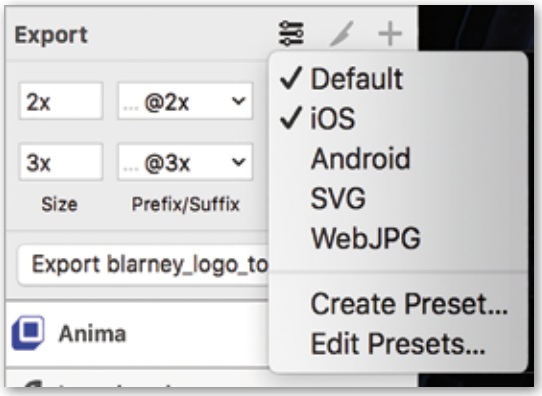


Figure 15: Sketch file exporting options

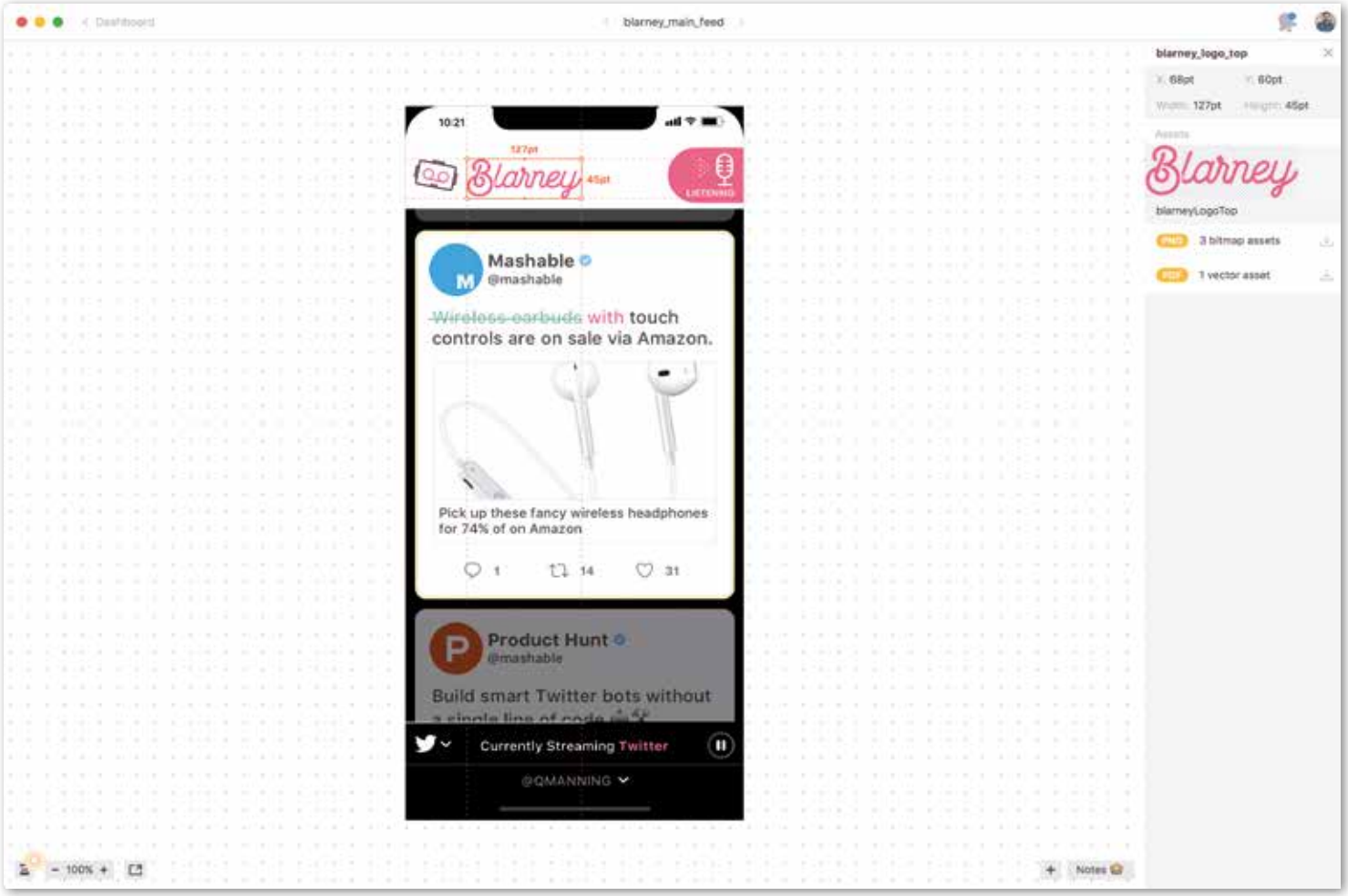


Figure 16: The Zeplin interface showing redline measurements and information for the Blarney logo

Picking the Best Design Tools

What design tools work best depends on the phase of the project, what you're hoping to accomplish, and what you're most familiar with.

Sketch and **Adobe XD** have the smallest learning curves, because they're the least feature heavy, and are tailor made for interface design. **Photoshop** and **Illustrator** do a lot more, and are more complicated accordingly.

Invision is now offering an early beta of their own design software, which ties into their online offering seamlessly.

Getting It to Development

Users have given you the thumbs up for your designs. Now all that's left is to make sure that the developers implement it correctly.

For many projects, this can be a desperate situation. UX artists and designers spend hours discerning what the right placement is for an element, or moving text back and forth, up and down, in order to choose just the right amount of spacing between it and other items on a page. Then, when it gets to development, it seems all bets are off.

Developers, by and large, aren't designers, and you shouldn't expect them to be. Looking at a screenshot, then picking out 5px of space versus 10px can be a significant pain. Now apply this to the dimensions of logos, images, font-sizes, and a hundred other items.

Exporting, also known as slicing or crop and compress, gives developers the assets and tools they need. Every single asset needs to be exported for use in development, meaning every logo, photo, icon, divider line, background or any other artistic element that the developer won't be creating in code.

File Set-Up

Using the software mentioned earlier, designers can group and name each of these assets on the screen, as you can see in **Figure 14**.

If it's going to be exported, it needs to be named. These are brought into the build of a project, so you need to make sure that the file names are correct.

Here are some quick tips for filenames that work for Android, Web, or iOS:

- No spaces
- Underscore (_) instead of dashes (-)
- Avoid using periods (.)
- Use all lowercase letters, not camelCase

For example, "header_background_image.png" is a good filename that should work for all platforms.

Exporting Files

Depending on the software, the way you export varies. Sketch, on Apple computers, automatically appends @2x or @3x to assets, as well as places files into MDPI, HDPI, or similar folders for Android usage. These things save time on projects, and get all of the right resolution files to the developer with the correct names. Adobe XD, Photoshop, and Illustrator also now provide exporting of assets in various resolutions, although they don't offer the automatic folder creation that Sketch does, as seen in **Figure 15**.

For anything with a transparent background, like icons or logos, use PNG-24 or SVG. For any photograph, you want to use a JPG. These are good rules of thumb for any platform.

Optimizing Files

Whether dealing with website download times or mobile app package size, you want to have your assets as small as possible.

For icons, logos or items with a transparent background, the use of a vector SVG is ideal due to their small byte size, their ability to be resized infinitely with no degradation, and how they don't require any @1x or multiple DPI options. However, mobile operating systems may have issues with implementing SVG files without individual plugins or libraries.

With the launch of the iPhoneX and iOS11, Apple is now recommending using PDF files when possible. Like SVG, a PDF is just a vector file format. PNG files and JPG files need to be optimized as much as possible; otherwise, you can quickly get into the hundreds of megabyte size range. Use a PDF in the same places you would use SVG.

Whether PNG or JPG, numerous optimization options exist out there; my go-to service is http://tinypng.com.

TinyPNG is one of the few places that creates a lower-sized, color-indexed PNG-8 and gives it a true alpha channel. True alpha channels allow the PNG to be placed anywhere, on any background design or color, without the presence of any aliasing. TinyPNG can also optimize JPG, though choosing the right level of compression is typically a subjective decision.

Redlining Is Vital

Redlining is taking an art file and annotating all of the measurements on the screen. Include things like the size of graphics, the distance from one object to another, the size of fonts, and so on. Measurements like these ensure that your project comes across as pixel perfect.

Numerous options exist out there for redlining, but I use http://zeplin.io. Zeplin is a desktop/SAAS combo that takes a Sketch or other layered file-type, and automatically creates all of the necessary redline measurements for you. You can see the Blarney design in Zeplin in **Figure 16**.

Using Zeplin or other automatic redline tools can shave dozens of hours off of a project, while also giving accurate results that developers can trust.

You're Set Up for Success

No plan or advice can guarantee you success in the marketplace. Nothing can guarantee you a smooth development cycle, when APIs may break or functional requirements for operating systems may change with a new release.

But, you can now roll with the punches a little more relaxed, and speak with more confidence about the decisions you make on your project. At the end of the day, what more could you want?

Q Manning

Quality Software Consulting for Over 20 Years

CODE Consulting engages the world's leading experts to successfully complete your software goals. We are big enough to handle large projects, yet small enough for every project to be important. Consulting services include mentoring, solving technical challenges, and writing turn-key software systems, based on your needs. Utilizing proven processes and full transparency, we can work with your development team or autonomously to complete any software project.

Contact us today for your free 1-hour consultation.

Helping Companies Build Better Software Since 1993

www.codemag.com/consulting
832-717-4445 ext. 9 • info@codemag.com

CODE CONSULTING

Securing IIS Web Sites with Let's Encrypt Certificates

HTTPS or HTTP over TLS (formerly SSL) is no longer an optional component when you build a Web site today: It's a requirement. Encrypted connections hide traffic on the wire and make it much more difficult to hijack HTTP connections or steal valuable cookie information to reuse in playback attacks. TLS can also prevent a host of drive-by and man-in-the-middle

Rick Strahl
www.west-wind.com
rstrahl@west-wind.com

Rick Strahl is president of West Wind Technologies in Maui, Hawaii. The company specializes in Web and distributed application development and tools, with focus on Windows Server Products, .NET, Visual Studio, and Visual FoxPro. Rick is the author of West Wind Web Connection, West Wind Web Store, and West Wind HTML Help Builder. He's also a C# MVP, a frequent contributor to magazines and books, a frequent speaker at international developer conferences, and the co-publisher of CODE Magazine.

attacks that are all too easy to instigate over non-secure connections in any public space. TLS keeps data secure while users are sending and receiving data, making it much harder to "listen in" on a connection on the Web. It's not a panacea for all security issues, but it's big fat low-hanging fruit to start with, and your site should proactively encourage this secure-by-default behavior.

As a response to the security concerns, we're now seeing many new Web features that **require** secure connections. The Geolocation APIs in Chrome now only work over secure connections. Most Google APIs that you can integrate with on your website today like maps, geolocation, translate, graphs and so on, all must originate from secure endpoints of your source site in order to work. Another important milestone is the newish high performance **HTTP2** protocol that only works over secure connections. Other new Web network protocols are following suit. The day is fast approaching when unsecure connections will be relegated to a shunned digital backwater.

> The day is fast approaching when unsecure connections will be relegated to a shunned digital backwater.

Google seems to be leading the charge, as they were the first to force some of their APIs, like Geolocation, to work only over TLS. Chrome now also flags websites that don't run HTTPS with a warning icon, as shown in **Figure 1. Figure 2** shows a site that has a secure certificate installed and the message shown there is much more confidence-inspiring.

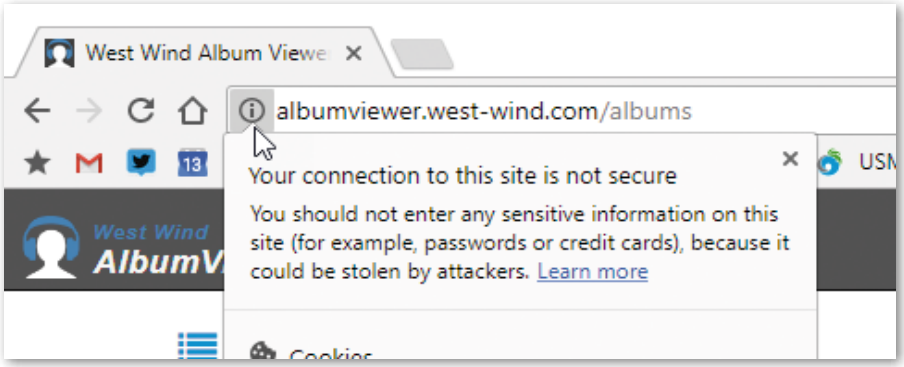


Figure 1: Non-HTTPS websites are flagged with a warning in the address bar.

But even more to the point for the Web at large, back in late 2014, Google's search engine algorithm started down-ranking non-secure sites in favor of secure ones. More recently Google started showing non-secure sites with a warning icon (**Figure 1**). This really hits website owners where it hurts, because traffic is the currency that drives investment and profit for most websites. And it seems to be working. HTTPS adoption has been on a rapid rise over the last couple of years.

> HTTPS is no longer an option; it's now a requirement for a website.

In short, TLS is no longer an option but a necessity for **any** website, large and small. And now, with new tooling and free certificates available from Let's Encrypt and a few other providers, there's no longer any excuse to not use encrypted HTTP, even on small hobby or sample sites. Anything public should just run on HTTPS. Setting up a new certificate, even on Windows and IIS can now literally be done in a few minutes and the renewal process can be fully automated. It's essentially fire and forget. If you haven't secured your sites yet, this article is for you.

In this article, I show you how to set up a certificate and get it installed in IIS via some of the automated tools. There's not much to show because it's so easy, but my goal with this article is exactly that: Show that it's a no brainer task and there's no reason to put off getting your site to run under TLS.

Let's get to it.

Free Certificates for All

A few years back, the Internet Security Research Group (ISRG), which is a consortium of sponsors that includes the Electronic Frontier Foundation, Mozilla, Cisco, Google, Facebook, and many others, started setting up a new Certificate Authority with the goal to provide free and fully automatable certificates to the public. Due to the benefits of TLS, they felt that it's important enough to tackle this space and challenge the up-to-then entrenched providers. A host of other big network names jumped onboard to provide support and funding for this project with the goal to provide free and easy-to-manage certificates to the Web community. Certificate issuance is big business and it's no surprise that along Let's Encrypt's way, there were some interesting attempts

to interfere with the progress by some of the big entrenched certificate providers. In the end, Let's Encrypt made it to production and has been a huge success in the last couple of years, since its initial release. Let's Encrypt has been a huge driving force in the adoption of HTTPS connections. At the end of June, Let's Encrypt had served over 100 million certificates in less than a two-year timespan.

Free is a big motivator and it's reflected in this adoption rate. Personally, I've been running a number of small personal sites. Although certificate prices for non-validated certificates had been dropping significantly even in years prior to Let's Encrypt's emergence, even $20 for 10 sites adds up to real money. If you add in the time it took to manage the upgrade process and installation, there were resources and some significant effort involved. For personal sites, it was often a matter of writing it off as "too much effort."

But with the cost factor gone, plus next to zero administration, everything changes.

Certificate Automation

If you've been around doing Web development or Web administration over the years, you probably know that installation and management of certificates has been a major hassle. Between the cost of certificates, applying and renewing certificates and managing the process of generating certificate requests and then installing final certificates into servers was always a pain, especially on Windows with IIS with its broken certificate renewal system that only seemed to work with Microsoft's own generated certificates. Updating a certificate always seemed like a dreadful task to look forward to.

Let's Encrypt provides **a completely open API** to manage the certificate request and retrieval process. In combination with high-level tools that automate the API, it's possible to compress the entire certificate generation, retrieval, and renewal process into a very short process that takes less than a minute. The client tools can then also install the generated certificate right into the Web Server, ready to serve requests immediately.

To me, this has been the biggest selling point of Let's Encrypt, even more so than the free factor. In the past, I've had about 15 certificates that I had to renew over the course of the year. So, pretty much once a month I'd have to renew one certificate or another and each time it takes a little time out of my day to do so. First, I had to apply for a certificate, send in a request, then wait a few hours get the request back, build the final certificate that I could install in the certificate store and then bind it to IIS. The process isn't hard, but it was spread out over the course of a few hours asynchronously. Rinse and repeat for a renewal 15 times a year. Blech.

Let's Encrypt changes that entire process through its open **ACME API,** which automates the entire process of certificate generation. Rather than dealing with a different company each time, you use a common ACME client implementation that directly talks to the automated API to create and issue a new certificate in about 30 seconds. Likewise, the API supports updating and renewing certificates through an automation API endpoint.

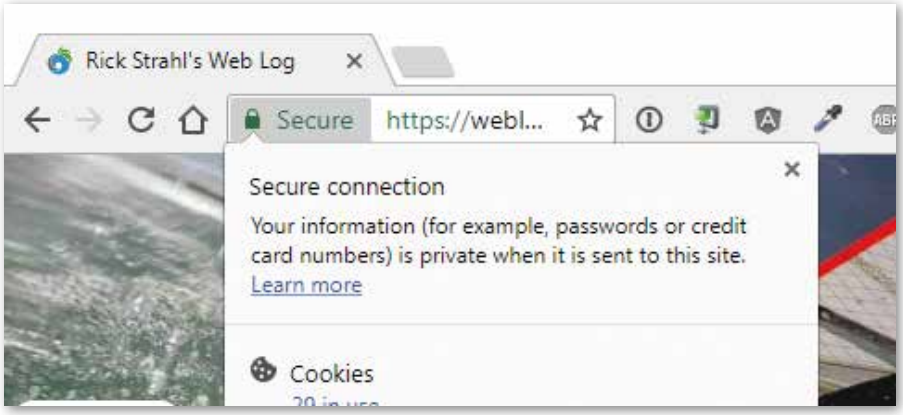


Figure 2: Sites with an HTTPS Certificate and secure content show a reassuring secure icon and message.

Native Let's Encrypt tooling (https://certbot.eff.org/) provided by the EFF and the Let's Encrypt foundation is supported only on Linux and Mac, but there are several ports of the ACME client protocol including a number of .NET-based libraries that runs on Windows. One such implementation, ACMESharp (https://github.com/ebekker/ACMESharp), has become the basis for most of the high-level tools that sit on top of it and provide an easy interface to managing certificates.

Limitations

Okay, Let's Encrypt sounds awesome, but it's not all rainbows and unicorns. Before I take a look at how to create a new certificate and install it in IIS, there are a few things you should understand about some limitations of Let's Encrypt Certificates:

- **Certificates are domain validated.** Because Let's Encrypt uses an automated process, there's no background check or other validation beyond a DNS domain validation. Basically, you run the certificate request from the domain that's to host the certificate, which validates that you control the domain you're generating the certificate from. Beyond that, there's no additional validation. If you need Organization Validated or Extended Validation certificates (the green label certs that show a company name), you still need to use a commercial provider and pay the big bucks. These certificates not only aren't free, but they're very expensive. These certificates aren't any more secure, but they're backed by background checks for the companies that use them, which provides an extra level of confidence for users of a mainstream site. The green bar isn't about security; it's mainly about trust and optics in the browser bar.
- **No wildcard certificates.** Currently, there's no support for wild card domain certificates available from Let's Encrypt. For small sites that have one or even a few websites on a single domain, this isn't much of a problem. But if you're building multi-tenant applications that might have many sites on various sub-domains, this can become a problem. Although you can easily create certificates on sub-domains using automation, there are some limits in how many you can create quickly without hitting Let's Encrypt rate limits. Let's Encrypt prom-

TLS and SSL: What's the Difference

TLS stands for Transport Layer Security, which is the successor to SSL, which stands for Secure Socket Layer. The principles are the same but the protocols have been updated to modern techniques and stronger encryption levels. The current minimum level of security recommended for certificates is TLS 1.2. SSL should no longer be used as it's dated and no longer considered to be secure enough. If you get a certificate today, it's likely to be a TLS 1.2 certificate.

What about Azure?

Unfortunately, using Let's Encrypt on Azure for Azure websites or Azure App Services is not quite as smooth as the process described here. Microsoft has no native support for Let's Encrypt—not surprisingly; apparently, they'd rather sell you a certificate. Although it's possible to create a Let's Encrypt certificate for an Azure website or App Service, the process involved is quite complicated, requires a third-party add-in, and is generally unpleasant. If you want more info, check out Troy Hunt's post (https://goo.gl/aT4KGy) from over a year ago. Not much has changed since then, so I wouldn't hold my breath for Microsoft to provide built-in support for Let's Encrypt.

Note that you can use Let's Encrypt with Azure-hosted Virtual Machines using the same process described here, as long as you have explicit domains mapped to the VMs IP addresses.

ises wild card certificates in 2018, but we'll have to wait and see how that implementation work for validation.

- **Rate limits.** Let's Encrypt has rate limits regarding how many new certificate requests you can make for a given domain within the course of a week. LE recently upped the rate from five to 20 requests a week. I frequently ran into the rate limits with five requests, especially when first experimenting with Let's Encrypt, which can be frustrating because you're effectively locked out for seven days once you hit the limit. You can check the rate limits here: https://letsencrypt.org/docs/rate-limits/.
- **Certificates expire after a maximum of 90 days.** Let's Encrypt certificates have to be renewed more frequently than traditional certificates, which are typically valid for a year. More frequent renewals ensure that if there ever is a breach (not that there has been one), it doesn't get exploited for long. On the other hand, if you're using Let's Encrypt with auto-renewals, it doesn't really impact you much because the renewals are fully automated, so this isn't as big an issue as you might think.

Creating a Certificate on Windows

There's no official Let's Encrypt client for Windows provided by Let's Encrypt, but there are a number of third-party providers. I'm going to look at two different tools here that address two slightly different use cases:

- LetsEncrypt-Win-Simple (https://github.com/Lone-Coder/letsencrypt-win-simple)
- Certify the Web (https://certify.webprofusion.com)

> Let's Encrypt currently has no support for wildcard domains which means that each and every domain has to have its own certificate.

The first is a very easy-to-use command-line tool that you essentially run once to configure an individual website. The second is an easy-to-use GUI tool that provides a lot

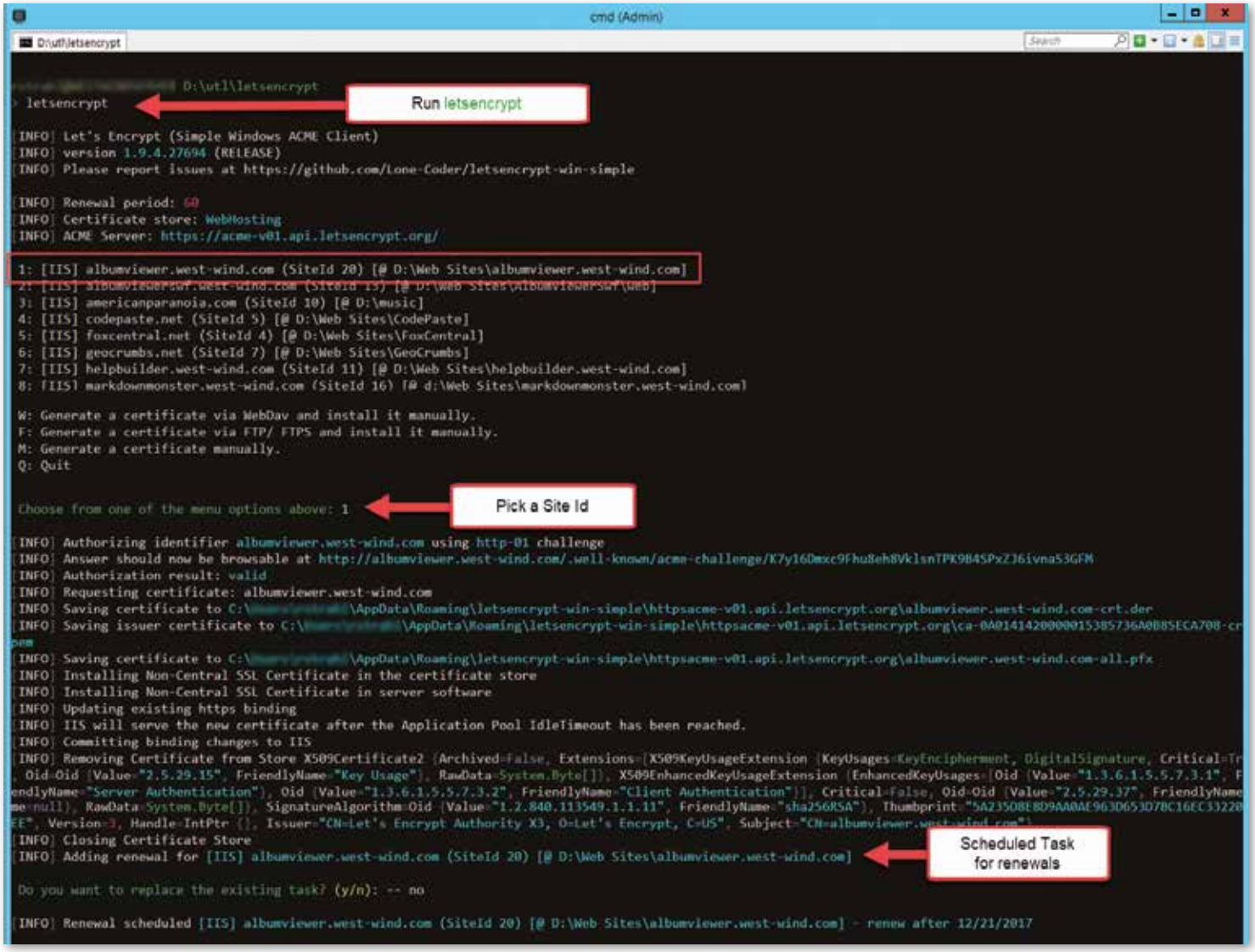


Figure 3: LetsEncrypt-Win-Simple creates a new TLS certificate in less than a minute.

of additional functionality in customizing and managing certificate requests along with a nice visual way to see certificates and their status. Both tools can have you up and running in minutes.

> LetsEncrypt-Win-Simple is a very simple command-line utility that can get your certificates created and installed in couple of minutes.

LetsEncrypt-Win-Simple

Lets-Encrypt-Win-Simple is a very easy to use command-line utility that lets you get Let's Encrypt going on an IIS-based Web Server in a couple of minutes. Literally. To get the utility, go to the Let's Encrypt-Win-Simple GitHub repository and the **Releases** page here: https://github.com/Lone-Coder/letsencrypt-win-simple/releases.

To get going, use these steps:

1. Download the latest **non-beta zip file** to your Web Server.
2. Unpack the resulting files into a folder of your choice.
3. Open an admin command prompt.
4. Type **Let's Encrypt.** First time execution asks you for a contact email for notifications/renewals. This brings up a list of IIS Web Sites of the server.
5. Pick a site with the number from the left. Press Enter.
6. Watch it go...The process should only take about 15 seconds.
7. You'll be prompted to add a scheduled task for renewals.
8. Put in account info for the account to run the scheduled task under.

Done!

Figure 3 shows what this process looks like from the command line.

If you think this is just too easy, you're not alone. When I first ran LetsEncrypt-Win-Simple and had it work effortlessly in one minute, I was blown away. Compared to my old manual process using Let's Encrypt's automated process, it's ridiculously simple.

And that's really the main point of Let's Encrypt: The automated process, even more so than the free pricing, makes Let's Encrypt a no-brainer for adding HTTP encryption to your site.

Figure 4 shows a happy HTTPS enabled site and certificate.

Certify the Web: A Richer GUI Client

It's hard to beat the simplicity of Let's Encrypt-Win-Simple, but if you want a little more control over the certificate management process, how certificates are created, trigger actions on creation, renewals and failures, or simply to get more useful error information if something goes wrong, take a look at Webprofusion's **Certify the Web** (https://certifytheweb.com), also known as Certify for short. Usually, you think of a GUI application as a dumbed-down version of a command line API, but here, the opposite is the case: The app piles on a bunch of useful features to customize the certificate generation and management process.

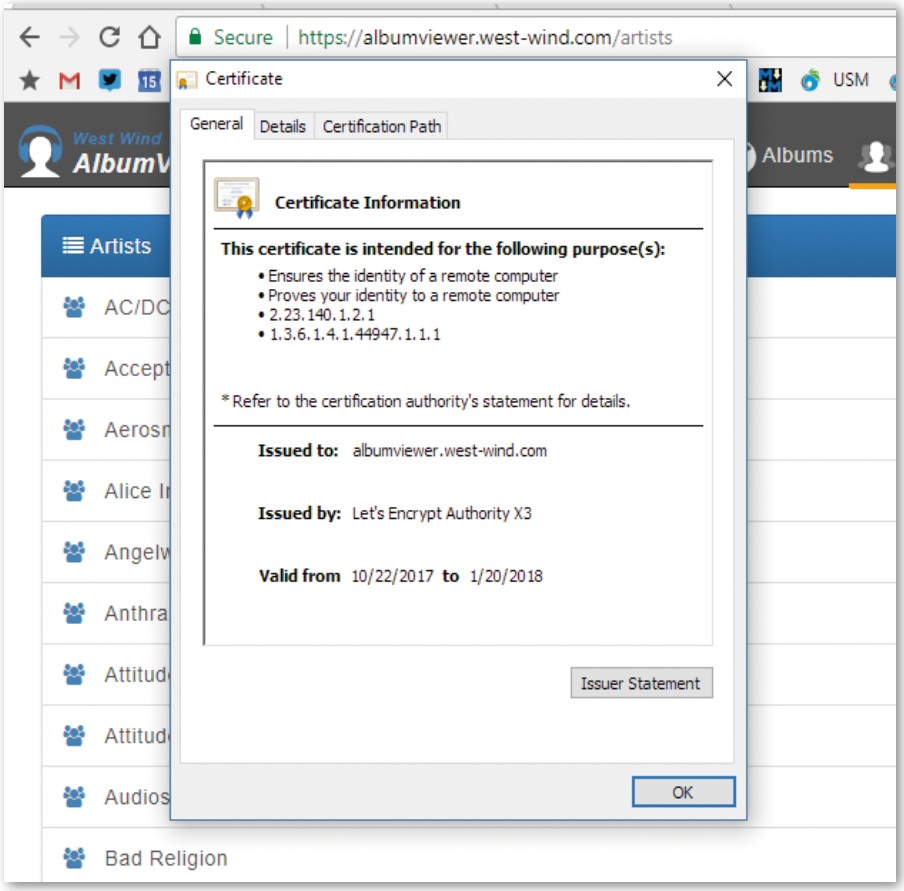


Figure 4: When it's all done, I have a valid LetsEncrypt Certificate and a green browser bar.

Figure 5 shows Certify the Web's advanced certificate configuration page and you can see that there are a host of options available. The options are just that, too: optional. If you're the "just get it done" type, the defaults get you a certificate in a few seconds. At its simplest, you just choose New Certificate, select your IIS site from the drop-down list, and hit Request Certificate. First-time users should also use the Configure Auto Renew option to set the credentials for the scheduled maintenance task.

Behind the scenes, Certify uses the same ACME .NET APIs that Let's Encrypt-Win-Simple uses, but Certify makes the process more interactive so you get to see what happens in more detail. Certify also can sense a number of configuration issues (e.g., IIS configuration and DNS settings such as CAA and DNSSEC) before making certificate requests, which helps to avoid API rate-limiting problems and takes the guess work out of troubleshooting.

Advanced users can choose to individually configure each certificate and have the option to specify validation protocols (such as validating a site via HTTP or TLS

ACME Protocol

Let's Encrypt uses the standards-based API for automating interactions between certificate authorities and clients. This is a well-known API with specific endpoints for platform-specific clients to implement. Clients exist for just about all platforms, and on Windows there's a .NET-based ACME client library that's used by most of the tools available, including LetsEncrypt-Win-Simple and Certify the Web, both discussed in this article.

Because this API is open-source and well-known, it's possible to integrate Let's Encrypt into many solutions directly. For example, Apache now has integrated support for Let's Encrypt in the core Web server and can handle the certificate management process directly. Hopefully, in the future, we'll see specific integrations into other Web servers including on Windows and, more importantly, on Azure.

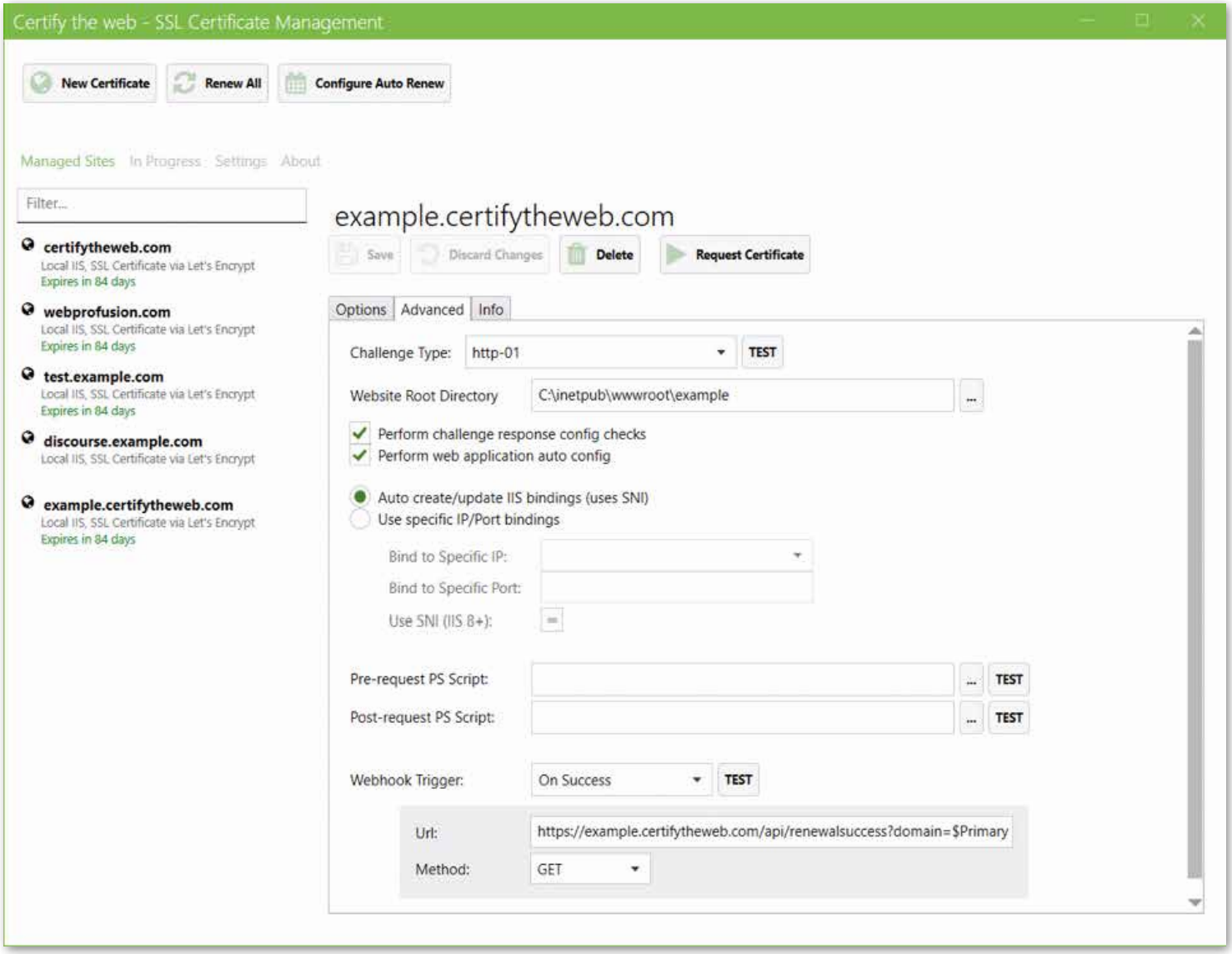


Figure 5: Certify the Web provides a rich GUI client to Let's Encrypt with many advanced admin features.

for certificate requests selectively). You can also group together several subdomains into a single group (or list of Subject Alternative Names, or SANs) that are submitted as a single certificate request to Let's Encrypt, which makes it easier to manage many subdomains of the same site. This is especially useful for multi-tenant applications and gets around the domain overload that can occur if you're running a lot of subdomains on a single IIS box.

Certify the Web also supports executing scripts for pre- and post-processing with PowerShell scripts or Web Hooks that allow a backend application to monitor renewals and domain registrations for each site. The PowerShell options are especially useful if you need to manipulate, export or otherwise re-use the renewed certificate for services such as Remote Desktop or Exchange). You can even choose to revoke your certificates as an added level of security.

Please note that Certify is not a free tool. You can manage up to five sites/certificates for free, but any additional sites require a paid license at a reasonable cost (https://certify.webprofusion.com/register). A professional license is available for unlimited certificates on up to three servers and there's an enterprise version that supports unlimited certificates for up to 100 servers. Free licenses are available for organizations who contribute code to the open source project (https://github.com/webprofusion/certify). It's worth bearing in mind that if you have many sites, you can mix and match completely free options like LetsEncrypt-WinSimple and still use Certify for other sites on the same server, depending on your requirements. It's worth trying out all the available tools to see which works best for you.

For administrative installations, Certify is an excellent choice for managing a lot of related certificates easily. The notification hooks are an especially nice touch that allows administrators to get properly notified if something goes wrong in the registration or renewal process, which is a major issue if you're dealing with hundreds of certificates on a server.

Certificate Creation Troubleshooting

Here are a couple of things you need to be aware of when creating certificates on your IIS Web Server. Although the process of creating certificates is fully automated for any of the tools you're likely to use, there are a few things that can throw things off if your website does some custom routing or URL Rewriting. Here are a few things that have tripped me up before.

The .well-known Virtual Folder Structure

Let's Encrypt uses a specific domain validation process (called an **http-01** challenge) that involves creating and then accessing a temporary **.well-known** virtual directory underneath the website you are creating the certificate for. The Let's Encrypt clients create the folder and then place several validation files there that the Let's Encrypt authority picks up for verification. The files are extension-less JSON files that are enabled by clearing out all IIS handler extensions for that folder and mapping all files to the **StaticFileModule** in a custom **web.config**file that's placed in the target folder.

It's important that this process can work, so if you're using IIS Rewrite rules that re-route every request, you may have to turn those off temporarily to get the initial certificate installed. If you're already using a certificate, it's also a good idea to allow Let's Encrypt to hit the site with a non-HTTPS request.

Note that if you're using an older version of Let's Encrypt-Win-Simple, the provided **web.config** didn't automatically handle the static file mapping, which caused a lot of problems for ASP.NET MVC sites that took over URL routing for extension-less URLs. This was addressed in 1.94 and later: If you use an older version, I recommend that you upgrade. You can read more about this issue and a few workarounds for scenarios where the custom web.config file isn't enough in a blog post of mine (https://weblog.west-wind.com/posts/2017/Sep/09/Configuring-LetsEncrypt-for-ASPNET-Core-and-IIS).

WWW and Root Domains

Let's Encrypt validates **specific** domains, which means that each and every DNS name you want encrypted HTTPS for needs to have a certificate. This means that if you have a root site (myDomain.com) and a www domain (www.myDomain.com) **both of them** have to get separate certificates. **Each and every domain and subdomain requires a certificate**.

According to the Let's Encrypt website, starting in early 2018, Let's Encrypt will offer wildcard certificates that can cover all domains under a given root domain, which will be a welcome addition. On my site, I run 15 separate subdomains for west-wind.com and having a single wildcard certificate for all subdomains would certainly reduce the clutter. As easy as Let's Encrypt makes the process of creating a new certificate and renewing it, it's even easier to not have to create one at all in the first place.

Other Platforms

As mentioned in the beginning, Windows is a special case for Let's Encrypt in that there's no official Let's Encrypt client. The two clients mentioned in this article are the most popular ones and they are certainly up to the task on Windows. On Mac or Linux, you can use the official Let's Encrypt clients that provide raw certificate creation services. These tools are generally low-level and don't explicitly handle tasks like installing into a specific Web server because on non-Windows platforms there tend to be a lot more Web Server choices available. However, there are also many utilities available that specifically layer on top of the base tooling and work with specific Web servers.

You might also wonder about using Let's Encrypt with Azure. Unfortunately, there's no direct support for Let's Encrypt on Azure, and running a Let's Encrypt client on Azure for anything but Virtual Machines is quite complicated. Although there are some third party add-ins available, even with those, the process is still quite involved.

Lock It Up!

If you're still running unsecured, non-HTTP websites today, you really don't have any excuse any longer. With Let's Encrypt, the process to create a new TLS certificate has become so easy that there is no reason not be using HTTPS anymore. With readily available tools even on Windows, the process of creating certificates takes only a few minutes and with an auto-configured renewal process, the entire operation becomes a one-time fire-and-forget process. No more unsecured Web sites –let's make it happen!

Rick Strahl

CODE

ACMESharp for Custom Windows Integrations

If you want to create custom integrations for Let's Encrypt using .NET or PowerShell tools, you can use the ACMESharp library. The two tools described in this article are both based on the ACMESharp .NET client library. You can use those same base libraries for your own integrations.

Note that both Lets-Encrypt-WinSimple and Certify the Web are open source and allow contributions, so if there are features you want beyond what's included, there's the option of extending and getting involved in these projects yourself.

Understanding Microservices and Microservice Architecture

Have you noticed what a cornucopia of buzz-words the technology world is? It seems that we're in a never-ending quest to put as many words before "-driven development" and "-oriented architecture" as we can. One such term that's been thrown around for the last few years is "microservices." The problem is that there are a lot of opinions about what they are and how to do them right.

Miguel Castro
miguelcastro67@gmail.com
@miguelcastro67

Whether playing on the local Radio Shack's TRS-80 or designing systems for clients around the globe, Miguel has been writing software since he was 12 years old. He insists on staying heavily involved and up-to-date on all aspects of software application design and development, and projects that diversify into the type of training and consulting he provides to his customers. He believes that it's never just about understanding the technologies but how technologies work together. In fact, it's on this concept that Miguel bases his Pluralsight courses. Miguel has been a Microsoft MVP since 2005 and when he's not consulting or training, he speaks at conferences around the world, practices combining on-stage tech and comedy, and never misses a Formula 1 race. But best of all, he's the proud father of a very tech-savvy 12-year-old girl.

A further problem is that many of these opinions and definitions have the correct intent while differing a bit from one another. So, let me further stir the pot by now offering you my opinion. I've always tried to be a realist and not fall into the hype of the new "shiny coin." I try to concentrate on what can truly bring value to my customers. I've found true value in approaching my designs using microservices to different extents in different scenarios, but I've approached it a little differently from what you may have read elsewhere. I've approached things in a holistic fashion.

Microservices are not really a particular type of service; they're a part of an architecture that makes the services, the act of calling the services, and the clients that call the services, behave a certain way and provide certain characteristics to an application. The more accurate definitions and explanations of microservices I have seen are those that approach it from an architectural perspective. I'm not implying that microservices require a tight coupling with the application that consumes them. But I am saying that microservices are as much about the architecture of a system as they are about the implementation technology and the tools used to create them. The goal of this article is to give you clarity in understanding microservices by way of an overview of their characteristics and that of an architecture that implements them. How the concepts I'm going to discuss are implemented remains a large and complex topic, and one of possibly several potential future articles.

> Microservices are part of an architecture that makes the services and clients behave a certain way and provide certain characteristics to an application.

If you're asking yourself, "do we really need another description of microservices," I'll tell you why I think the answer is yes. This is large topic with a number of different actors and ways to implement those actors. Learning about various points of view has always helped me take an eclectic stance and adopt ideas from more than one scenario. I hope, if you do the same, I'm contributing to that.

Looking at the Big Picture

If you've made the transition from developer to architect, whether in a full- or part-time capacity, the concept of the "big picture" should not be strange to you. As an architect, one of the tasks that falls under my responsibility is to talk to the developers across the entire stack and be mindful of what their needs are, both individually and collectively. How are service boundaries defined? Are services too coarse or too fine? How are services accessed? What parts of a client are dependent on another? How much can fail on the service-side before the entire client-side is rendered useless? These are just a few questions that architects ask when looking at building a new system. In fact, these are important questions when looking at an existing system that may require modification or even overhaul.

So, can microservices benefit the system you're designing? There are many who say absolutely. But you can't really know without having a good understanding of what they are, how they're accessed, and if your application can be decomposed to accommodate them. And then there is the matter of the technology implementation choices you face. This is looking at the big picture.

The "Micro" in Microservice

The biggest misunderstanding about microservices is the prefix the term uses: "micro." Linguistically, this implies small services. Really small services. That's not exactly the case. Microservices are about orchestrating and decomposing a system into services that are loosely coupled and that encapsulate areas of volatility. When done correctly, yes, this results in smaller services that expose operations in as fine-grained logical groupings as possible. This is because in microservices, the service boundaries are closer to the center, resulting in a more specific responsibility. This is probably where the term "micro" comes from.

> A microservice is designed to do a very specific thing and only that thing, and to operate as autonomously as possible.

The decomposition that leads to how many services to have and what they do differs greatly from system to system, as does the size of the service. If this sounds a lot like the good old **Service-Oriented Architecture** (SOA) that you've read about plenty in the past, that's because microservices have been described as SOA done right. SOA is about breaking up a monolithic system into a set of coarse-grained, somewhat autonomous components

and then distributing the hosting of these components, or services, for shared client access. Systems are now separated into areas of responsibility so they can evolve and be deployed independently and not compromise the entire application. This also has the advantage of eliminating quite a bit of responsibility from the client regarding things like database access and details of workflow steps that can now be housed, orchestrated, and exposed by a service layer. On the Microsoft stack, you'd typically use either **WCF** or **Web API** to write and expose services. A client can now be a lot "dumber" and need only to know where a service is and how to access it. With WCF, this is done with a client channel (usually crated by a proxy class); in Web API, it uses the **HttpClient** class. Of course, in the case of the client being pure JavaScript, services are limited to a REST implementation of some kind.

> The biggest misunderstanding about microservices is the prefix the term uses, "micro."

More than likely, I haven't told you anything you didn't already know, so how are microservices different? Well, the immediate answer is that in microservices, the granularity of services gets finer, making microservices an evolution of traditional SOA, but it's about much more. Let's go back to my original statement about microservices being as much about architecture as they are about the implementing technology, and how I approach things in a more holistic fashion. This means I try to think about the entire application and how it will be laid out.

In traditional SOA, you think about services on the server side, obviously. You spec out your service decomposition there, but traditionally the client doesn't take any of that into consideration. It tends to stay as far away from that design as it possibly can, with the expectation that it will be able to use the services later. When architecting a microservice-based system, this can change a little. I'm not saying that the services are designed to cater to a specific client. By no means should that be the case. The decomposition on the server side remains decoupled from whatever clients it services. What I mean is to give thought to how a client is broken out because that becomes important when you start introducing the technology that accesses services; as you'll see, that's a big part of a microservice architecture.

Client Composition

Did you think you'd be reading about client design when discussing services? Let me be clear. I'm not saying that microservices should be designed around their potential clients. I'm saying you should be thinking about them when architecting a complete system that will use microservices. Now, before some of you start yelling loud enough to reach me here in New Jersey, let me elaborate on that statement. If you're talking about the services exclusively, the clients are, for the most part, irrelevant. But I'm talking about a larger picture here, remember? I'm talking about organizing a composite system with regard to both the services and the clients. The relationship here is loose, but it's a relationship nonetheless.

Most definitions of microservice-based solutions discuss fault tolerance to some degree, as I'll also do later. But what about complete failure of a service? On the service side, this may very well affect other services, either crippling them partially or rendering them completely useless. This would obviously affect the client as well, but it certainly shouldn't render an entire client useless. Although there's service decomposition that takes place when the service layer is designed, there's also decomposition of a client so that failure of a service or groups of services manifests on the client in a controlled fashion, making only a part or parts of the client inoperable—hopefully temporarily of course. Note that the decomposition of a client is not particularly a new concept, nor one that is tied to the service-oriented world. This is one of the things that leads to what is sometimes referred to as a composite application.

A Practical Example

Let me put this into perspective using a real-world example in a business sector with which we are all familiar: eCommerce.

An eCommerce system, such as Amazon.com, can have an entire service layer powering it. The decomposition of all the needed functionality resulted in groups of services that handle areas such as **User Account**, **Shopping**, and **Checkout**. In fact, these areas can be further broken into sub-areas, each somewhat autonomous. The **User Account** area can consist of **Account Creation**, **Login**, **Profile Management**, and management of a user's definition, all specific to this system. In the case of Amazon, this could include the one-click setting, Kindle devices, etc. The **Shopping** area can be broken out into **Product Browsing** and **Building a Shopping Cart**. And even **Product Browsing** can be split into browsing various product categories. The **Checkout** area, which, on the surface, may seem like a self-contained area in its finest grained form already, can allow the as-is checkout versus the changing of checkout options, such as credit card or shipping address.

All of these areas, sub-areas, or sub-sub-areas I'm mentioning have their own service or small groups of services that handle the fulfillment of tasks that cover that area's responsibilities. The finer grain I can decompose service-side areas into, the more "micro" my services appear, right? Yes, but that fine-grain decomposition alone doesn't give this eCommerce system a microservice architecture. Some of these services may depend on the availability of others. Some of this dependency may even span areas of composition. An infrastructure is necessary to make these determinations. As I'll talk about a little later, this is one of the key characteristics of a microservice architecture and it's known as discoverability. Now let's shift to the client a little bit.

All these areas, as I've called them, that are driven by services have some kind of manifestation on the client. More than likely it's not a one-to-one relationship. For example, Product Browsing may be a part of the site that is serviced by several microservices. But the temporary failure of a service or a group of services that are needed for Product Browsing to operate shouldn't affect the user's ability to purchase what's currently in the user's shopping cart. And if the service that handles user login

A Reputable Source

When it comes to architectural principles in software, Martin Fowler is a dominating force and one whose reputation is secure and long-lived. A posting by him and James Lewis on microservices has become a go-to starting point for many architects and developers. Although this CODE Magazine article is based on my own opinions, experiences, and beliefs, you'll find some overlap on our opinions when reading their work. I've given you my own interpretation of how I approach a microservice architecture, one with some unique points, and the fact that I agree with a lot of things Fowler and Lewis point out isn't a coincidence. I used their writings as my starting point when I first got involved with microservices.

Fowler and Lewis' original post on microservices:
https://martinfowler.com/articles/microservices.html

The Microservices Resource Guide:
https://martinfowler.com/microservices/

is down, the site should be able to accommodate shopping under an anonymous identity. Of course, the Checkout section of the site would be off line until the Log-in service is back up. And then, it would also rely on its own services being available.

Each part of the client must accommodate a failure in accessing the services it calls upon. The discovery facility to which I alluded earlier forms a large part in facilitating this. Discoverability is one of many characteristics of a microservice architecture. This and all the other characteristics I'm going to talk about have more than one type of implementation possibility. There are products on the market, free and otherwise, that assist in providing your architecture with said characteristics. I'm not going to point you to any specific solution in this article. In fact, in many cases I've written custom implementations to accommodate my particular needs. My goal here is to make you aware of characteristics that your architecture and the actors within need to have if you want to design a clean, scalable, and robust microservice-based solution.

Microservice Architecture Characteristics

As you've probably figured out by now, fully embracing microservices is about more than just accessing a smaller service. A system properly architected to be microservice-based needs to look at having certain characteristics, some of which apply only to the services, but others to the system as a whole. Keep in mind that not only does the implementation of each of the characteristics I'll discuss vary, but also a microservice architecture isn't necessarily limited to only these characteristics. In fact, even the depth to which each of these carries can vary among implementations. This list represents items that you find in every well thought-out microservice architecture.

- Redundant hosting
- Isolation
- Dependency checking
- Service discovery
- Easy client access
- Failover and exception management

I'll limit each item to a conceptual explanation as anything beyond it both outside the scope of this article and beyond the room we have. You may find that these characteristics overlap a bit.

Redundant Hosting

Good services shouldn't be limited to one-location hosting. For systems to scale successfully, you don't want the same service in the same process in the same computer being the only point of availability to clients. But of course, hosting the same service from multiple addresses may introduce the question of which one to use. This is going to depend on how your services are hosted. If you're using virtual containers, such as Docker, your addressing may only differ by a host name. Whereas if you're simply hosting services on the same computer but different processes, a port name may differentiate the address. A load balancer may or may not exist in front of a group of the hosted services, or a discovery service may be the differentiator.

There's no exact right answer to this and there's no one solution for how to handle it. Note that with regard to how many services a host should handle, the ideal answer is one. But when looking at the business space to which the service provides value, one service may take a dependency on another. Whether this multitude of services is grouped into a single host or each is provided with a host of their own is going to depend on your environment, discovery facility, hardware, and several other factors.

Isolation

A service or group of services hosted together should be exposed in a fashion that doesn't put any other service or group of services at risk. This can only be guaranteed by isolating their hosting from one another, ranging from process isolation to computer isolation. If the desire is to have computer isolation in conjunction with deployment facilitation, this is where virtual containers can assist. But make no mistake, containers do introduce their own element of complexity.

> Containers introduce their own element of complexity.

Dependency Checking

You can't always guarantee that a service can or even should handle all its responsibility internally. This eventually leads to breaking that golden rule of reusability that you've been taught and have been practicing for years. A common task can easily be required by more than one service. Duplicating twice may be fine, three times may be acceptable, but twenty times is unfeasible and blatantly unmanageable. If one service is going to depend on another, the availability of the other needs to be checked. This can and should be handled by whatever discovery solution you choose but also, it shouldn't be limited to the service in need. Ideally, a client shouldn't be able to call a service that has unavailable dependencies. The extent of the dependency can vary. It can be assigned to an entire service or only a particular operation. Such information needs to be provided in some way. Again, you have choices. You can prevent the call all the way to the client level, or allow the call and adjust what gets returned to the client by the called service. How and where service dependencies are defined varies depending on the solution you choose.

Service Discovery

This key characteristic can be implemented in various ways but they all come down to a client not knowing specifically where a service is, and simply having a way to *ask* for it. This can be as broad as having a way to determine a hosting address, with the client still suppling the details of the call, including the resource (URL) and the payload. But it can also be more fine-grained where a client can ask for a service by a designated label, for example "CheckoutService". In either scenario, the discovery process handles looking for the appropriate service and returning it to the client. Checks for unavailable services and redundantly provided services should

Docker

Docker is about virtualization. Even better, it's about cross-platform virtualization. To paraphrase Fowler and Lewis' own definition, "Docker enables true independence between applications and infrastructure and developers and IT ops." Docker gives you portability for applications so that they can be developed on one technology stack with one platform in mind, yet deployed and hosted on a different platform. But not every application can be Docker-friendly. Docker isn't a trivial technology and if you need to perform any networking tasks, it becomes even less so.

Although it's not the only solution, and even not the preferred solution for many, Docker has established itself as a player in the world of microservices. This is largely due to deployment facilitation, and also to some discovery and load-balancing abilities built right into the product.

Find out more here:
https://www.docker.com

be built into this process and remain transparent to the client. As I mentioned in "Dependency Checking," a part of service discovery can also be to confirm that any of the dependencies are available as well.

Although many can disagree about what's important in a microservice architecture, it's pretty safe to say that discoverability is at the top of the list. A centralized discovery server is at the heart of all solutions and there are several off-the-shelf solutions for this. All have similarities, including the ability to keep a registry of what services you have and where they are. All other details, like constant heartbeat checking or automatic registering, may differ from solution to solution. How the act of discovery takes place should be exposed to the client in a clean and organized means. That task typically falls to the API Gateway.

Easy Client Access

In a microservice architecture, a client uses an **API Gateway** to perform a service call. How far from discoverability to the actual operation call an API Gateway can handle varies depending on your solution. An API Gateway is a key component in a microservice architecture because it's the first line of access for the client. In its simplest form, it can receive a request for a service and merely check a repository in order to return the address for that service.

How much it leaves for the client to do on its own or how much it handles for the client differs greatly. A robust and feature-filled API Gateway knows how to identify a service down to details about the operation desired and even other services on which it might depend. A common and key component of a well-written gateway is the ability to handle potential failover scenarios. It should be up to the API Gateway to handle a failure and try an alternative endpoint for a service that either the client or another service requested. It isn't until all avenues have been exhausted that the client should receive a failure notice from the API Gateway.

An API Gateway should also be the first line of security. If a call handled by an API Gateway successfully authenticates, that authentication should be held and passed through to the actual service call. The API Gateway is the part of a microservice solution that's very commonly developed in-house, as its intelligence, more often than not, needs to cater to and interact with just about every other characteristic in a microservice architecture.

Failover and Exception Management

Handling what happens when a service or group of services is down is vital to the continuous operation of an application. This is an area that again makes you look at the application as-a-whole. The down-state of a service should affect only the area of the application that needs that service and ideally not any other. Realistically, this depends on the application. A down-state of an area of a website may impede a particular workflow and prevent usage past a certain point. Knowing this information upon service request should be part of the responsibilities of a good API Gateway, and knowing how to manifest that failure gracefully to a user should be part of the application's overall architecture. A good microservice design includes redundant hosting for every service, ideally on both the same computer and across separate computers. An API Gateway should be able to take a request and simply find the service applicable to it. Selecting it from a series of locations based on availability and/or load balancing can be the responsibility of a really smart gateway or can be a shared responsibility of a gateway and a load balancer.

The API Gateway is the part of a microservice solution that's very commonly developed in house, as its intelligence will more often than not need to cater to and interact with just about every other characteristic in a microservice architecture.

Design for Failure

A lot of these characteristics are centered around a concept known as "design for failure." I give credit to Martin Fowler and James Lewis on possibly coining this as a software term. Put simply, this is the software architecture's version of Murphy's Law. Applications using a microservice architecture should have the layers above the service layer designed as autonomously as possible, following the similar concept already recommended in services. Again, this in no way means that that the client is coupled with the services, but it's certainly logically dependent on them; after all, services drive the clients. So, as I explained earlier with Amazon as an example, this means that sections of the client should be partitioned in such a way that they depend on each other as little as possible. Failures of any kind from the service layer and down should be gracefully handled by the client. The only real way to ensure this while you're developing an application is to always count on things failing.

When designing for failure, a service needs to know exactly what to do if any part of it fails or if a dependent call to another service fails. Similarly, a client needs to know what to do if a service call fails and restrict as little of itself as it can afford. A failure of a service can easily be because an error in a down-level object call was left unhandled. It can also mean that the database being accessed is offline. We've all seen all these scenarios in applications with all types of architectures, but the isolation you put into a microservice architecture should prohibit failures like this from toppling over an entire service layer or worse, an entire application. This doesn't mean to simply catch errors and tell a user that "the application is temporary offline." The point here is about failures only affecting a part of an application without prohibiting use of another.

Handling failure in an application properly should have nothing to do with microservices. This is just good architecture and design that every application should consider. A microservice architecture makes it so that failures on the service side don't prohibit usage of the entire application.

Depth of Encapsulation

This is perhaps one of the more controversial topics when researching definitions and descriptions of microservices. It goes to exactly how autonomous a microservice should be regarding its specific tasks and the resources on which it depends. The reason for the controversy is valid, because in a way, it can violate the reusability principles you've been following for a long time. To most people, the encapsulation of a microservice with its resources is one of its defining characteristics. This means that when you talk about a microservice, you're also talking about the objects it calls, and even the database it uses. All of it is considered part of the microservice—or is it?

This isn't an easy one to define, figure out, or even justify. Like many things in the wonderful world of software, it's very easy to define something and say, "this is what it is and this is how you do it," but you know that in the real world, absolute mindsets don't always work out as planned. Yes, ideally a microservice carries all its dependent resources along-side and as a group, all of it defining its autonomousity (is that even a word? It should be!).

> When you talk about a microservice, you're also talking about the objects it calls, and even the database it uses.

Remember those reusability principles I mentioned? Think about it. You have object libraries to reuse. You have databases that span an entire application. You reference utility objects all the time. Well, object references are easy enough to reuse. Each microservice gets its own reference and when deployed, doesn't clash with that of another microservice. The hard one is databases. And there's no easy way around this challenge because there's no easy solution. A dedicated database that services a single microservice (or set of jointly hosted microservices) is easy enough to implement, but what about the data in it that will then be used by another microservice? Again, there's no easy solution here. A common approach is a hybrid solution where an application doesn't have a single database, but it doesn't have one-per-microservice either.

Another alternative is to use database triggers that keep data in sync across databases. This can depend on how your particular workplace feels about database actors such as triggers and stored procedures. Although it may raise an eyebrow when I say it, there are indeed shops that refuse to use these database actors and instead rely on code-based DALs to handle all that. I've also seen dedicated "sync services" that watch and monitor multiple data updates and then syncs them across databases. Having a database professional on your team helps a lot here because a properly designed database is often crucial in choosing your particular answer to this problem. This is also a scenario where services calling other services often happens. Remember "dependencies?"

Microservices Product Choices

So, do all these things that I've discussed fall into the responsibility of the developers to write? If you need service discoverability, do you need to write a discovery hub yourself? What about hosting isolation and deployment? Well, they can be your responsibility. I've certainly written them and with very good reason and specific requirements in mind when doing so. But no, you don't necessarily have to develop every part of this infrastructure yourself. I'm a control-freak and I love to write infrastructure products and plumbing code, but that's just me (#gluttonforpunishment). There are products and services out there that provide you with a lot of this. A number of them are free, and some of them are not. I'm not here to push any particular product, as most of my microservice work has been home grown (I told you I was a control freak).

Discovery

There are discovery products out there, like **Microphone** and **Consul**, that allow you to create a registry of your services. I can't speak to their capabilities of self-registering of services that come up and auto-deregistering of services that go down. This is a feature I put into a home-grown discovery hub. In any case, these discovery solutions are designed to become the first point of contact for clients, through the API Gateway. The scope of information that you provide in a discovery service depends on the product and how it was written.

Hosting

There are many solutions for hosting services and for even running service code in various ways. Microsoft Azure for example, has the ability host services in a scalable fashion, allowing you to spin instances up and down as your needs require. Azure also offers **Azure Functions,** which lets you run code written in a variety of languages that responds to different events or is exposed RESTfully. How you turn a feature like this into part of your microservice solution depends on your architecture. Amazon's competition to Azure Functions is **AWS Lambda**. These are also event-driven code constructs.

API Gateway

I told you that the API Gateway is a crucial piece in a microservice architecture. Well, if you Google the term, you'll see that everyone has their own implementation idea about this. There are many people who've put their own versions of API Gateways on GitHub. I've even seen API Gateways that integrate the discovery piece within themselves. I don't consider this a good idea because it puts a lot of weight on the client by forcing it to keep the registry of all the services. I've always been fan of writing my own API Gateways; I've never really found any commercial products that cater to the specific tasks of a well-designed gateway, perhaps because it does have custom demands, especially in the area of interacting with a discovery mechanism. If the API Gateway is going to interact with a discovery mechanism, as it should, which discovery mechanism you use is going to steer how your API Gateway works. Some providers give you some help but you have to be using their product stack. If you're hosting services on Amazon, they have a developer's guide for an "Amazon API Gateway."

What's Next?

Make no mistake, microservices and a microservice architecture may not exactly simplify an application. All these things I've mentioned need to be thought out and discussed. Decomposing an application and distributing its responsibilities isn't a trivial task, but it's a crucial one; when looking at the whole architecture, you can't dismiss microservices in a client either. Designing for failure affects all parts of an application and in every layer, including the client. Seeing an application in a holistic way means understanding not only how the individual components work, but how they may or may not continue to work in the absence of some other component.

It's very common for systems to be built monolithically first, then refactored into a microservice architecture later. There are many who'll balk at this thought, and this isn't advice. I'm merely saying that not every team has the resources, budget, or time to get a first-version product out the door using a microservices approach. And speaking of teams, the breakdown is completely different. Whereas a monolithic architecture breaks out the teams by the layer of the application they control and know best, a microservice architecture's teams have a more vertical breakdown. Each team controls and owns a slice of the application that spans from the client to the database and everything in between.

This doesn't mean that the teams are kept sheltered within their own slice. Any dependencies, however loose, among services or among parts of the client need to be known and communicated on an ongoing basis. And there may be a team or two responsible for some horizontal slices of reusable object libraries that may be shared by the other teams. One particular advantage that's always discussed when you read about microservices is that teams may be developing their microservices using a different technology stack from another team. It's going to be up to you to decide if this is feasible or not. Yeah, on the surface it sounds like a great amount of freedom, but in reality, that approach may not be a value proposition for your environment. It's very easy for some teams to run off and jump on the "new shiny toy" without thinking about whether or not it actually brings value.

> One advantage in using microservices is that one team might develop their microservices using a different technology stack from another team.

When discussing the decision to go toward microservices, you need to ask yourself to what level are you going to take it. There's no microservice bible that says, "do it this way only." In fact, I suspect you're reading this because you've found that there isn't even a single definition of microservices or microservice architecture, and that got you even more confused. Yeah well, it's confused me as well, believe me.

Are all the characteristics I discussed necessary for every microservice architecture? Although it would be nice to have them all, you may decide that "failover handling" is just not important enough to you. I don't mean exception handling, I mean failover. Remember, this is one service going down and another hosted instance picking up the call in its place. Whether your reasons are time, budget, or resources, it's very possible to build an app in a microservice fashion and only have one host for each service. Of course, you certainly shouldn't ignore exception handling in this case, or any case. If a service goes down, you better be handling the problem gracefully all the way up to the client.

What about discovery? You may not have the time to write your own discovery server or the time to even look into and set up a commercial service that gives you one. Maybe you're refactoring parts of a monolithic app into microservices. This, in conjunction with not having failover support may mean that each service has only one address. Your API Gateway wouldn't be asking a discovery hub for a particular service but instead just addressing a service directly. The good thing about a microservice architecture is that some of these characteristics can be added later, so you can prioritize your needs. If you encapsulate all client access into at least a simple API Gateway, you give your clients one point of contact with the services and can modify and enhance the gateway's internals later.

The biggest priority hasn't changed since the days of conventional SOA, so you should start by concentrating on the decomposition of your system and how your services are going to be broken out. Lining up your microservice boundaries by business needs is crucial in your application's success. It affects everything else going forward.

I've given you my views on microservices and microservice architecture throughout this article, and I hope you weigh my information in conjunction with a lot of other good information out there on this topic. Earlier, I wondered if you were asking yourself, "do we really need another description of microservices," and I rhetorically answered that I thought we did. I sincerely hope that I've added to your arsenal of knowledge on this topic, and if you pulled some food for thought on developing your own ideas, then I guess your answer to my question is also, "yes."

Miguel Castro
CODE

Developer Update: iOS 11 and iPhone X

Continuing the established pattern of annual release cycles, Apple recently released iOS 11 to the public. The release included a number of updates and features for end users, but it also brought with it a suite of new developer tools and technology to take advantage of. With the introduction of tools like ARKit, MusicKit, Core ML, PDFKit, and others, developers have new ways to create amazing mobile experiences. Not only did Apple recently release this new software, but they also released new Hardware. This year, we saw the incremental release following up the 7/7+ line with an upgraded 8/8+, as expected, but Apple also released a new model called the iPhone X. The iPhone X, unlike its predecessors, has a Super Retina edge-to-edge screen and no home button. It's a new form factor and, as you might expect, these changes impact developers.

Jason Bender
Jason.bender@rocksaucestudios.com
www.controlappdelete.com
www.twitter.com/TheCodeBender

Jason Bender is the Director of Engineering/User Experience for Dosh (http://www.dosh.cash), an Austin, Texas-based fintech startup that gets you cash back when you eat out, go shopping, or book travel at over 100,000 brands and businesses. He's been coding since 2005 in multiple languages including Objective-C, Swift, Java, PHP, HTML, CSS, and JavaScript. With a primary focus on iOS, Jason has developed dozens of applications including a #1 ranking reference app in the US.

In this article, you'll take a look at the tools that iOS 11 introduces and how you can use them. You'll also get the rundown on everything you need to know about the iPhone X and how it might impact your development cycles.

iOS 11

The eleventh iteration of Apple's mobile operating system comes on the 10th anniversary of the original release and brings a host of new toolsets for developers including:

- **ARKit:** A framework for creating Augmented Reality experiences
- **Core ML:** A new machine-learning framework that supports the integration of a broad range of model types
- **MusicKit:** Lets users play their Apple Music and local music library from your applications
- **PDFKit:** Lets you display and manipulate PDF documents in your application
- **Drag and Drop:** A new feature allowing users quick ways to move text, images, and files among applications

Additionally, iOS 11 introduces a redesigned AppStore experience. This new AppStore provides developers further flexibility with app listing and deployments, giving you more control over how your application is presented to the end user.

ARKit

ARKit is packed with powerful tools that help merge digital objects with the physical world around you, letting developers take their apps beyond the screen and enable them to interact with the real world in new and exciting ways. It does this using Visual Inertial Odometry (VIO) to track the world around you. VIO takes in your camera sensor data and marries it to the outputs from Core Motion to allow your phone to accurately sense how it moves in a room.

ARKit is also able to find horizontal planes in a specified space that it's evaluating. By doing this, it's able to identify things like tables and floors and then use those planes to position virtual items in a life-like manner. Additionally, using the camera sensors to assess available light, it's able to approximate shadows and other variables for increased realism on any virtual object in the display.

> ARKit's powerful tools let developers take their apps beyond the screen and interact with the real world in new and exciting ways.

Although ARKit may look intimidating, getting started is easy. You can experiment with the basics by creating an ARKit sample template project through Xcode. Simply launch XCode and go to File > New > Project and then select **Augmented Reality App,** as shown in **Figure 1**.

After selecting the application template, name your application on the following screen and choose **SpriteKit** as the Content Technology. This generates a sample application for you to start playing with. SpriteKit is a 2D framework and provides a good starting point for experimenting with ARKit. With this sample, you can drop your 2D sprite into the physical world and watch it reposition and twist in the 3D space as you walk around with the phone camera.

Before running the sample application, let's jump into the source and make sense of what's happening. You'll notice that there's a view controller that has an ARSKView sub-view. This view is the bridge between ARKit and SpriteKit. You can use it to present an SKScene with special ARKit configurations like the ARWorldTrackingConfiguration. Apple defines that as "a configuration that uses the rear-facing camera, tracks a device's orientation and position, and detects real-world flat surfaces." This results in the scene asking for camera permissions and then showing a full screen camera view lens preview to the user.

If you take a look inside that SKScene you'll notice the following code in the touchesBegan method:

```
guard let sceneView =
self.view as? ARSKView else {
    return
}
```

```
// Create anchor using the camera's
// current position
if let currentFrame =
sceneView.session.currentFrame {

    // Create a transform with a translation
    // of 0.2 meters in front of the camera
    var translation = matrix_identity_float4x4
    translation.columns.3.z = -0.2
    let transform =
        simd_mul(currentFrame.camera.transform,
        translation)

    // Add a new anchor to the session
    let anchor = ARAnchor(transform: transform)
    sceneView.session.add(anchor: anchor)
}
```

This code creates an anchor point relative to the position on screen that the user touches. In SpriteKit, you might be used to creating nodes and adding them to the scene, but in ARKit you work with anchors. Anchors represent a placeholder within the scene and have an associated ID but no actual content. These anchors then get turned into nodes later using the nodeFor method in the view controller. Let's take a closer look at how the anchor is created. Consider the following lines:

```
// Create a transform with a translation
// of 0.2 meters in front of the camera
var translation = matrix_identity_float4x4
translation.columns.3.z = -0.2
let transform =
simd_mul(currentFrame.camera.transform,
translation)
```

The first variable "translation" represents a standard transform matrix. A transform matrix is a complicated unit that can represent device orientation, rotation, pitch, roll, yaw, and a number of other relevant bits of information when dealing with the 3D space. If you'd like to read more about how these matrices work, check the sidebar. The transform matrix also holds x, y, and z values. In the second line, you alter the z coordinate, which positions the anchor to a distance relative to the camera. Once you modify the z value, the translation is applied to the origin of the camera at the time the screen is pressed, and then that transform is used to create an anchor 0.2 meters from the camera at the location where the user taps the screen.

The anchor is only a placeholder, so this code alone isn't enough to show any visible feedback to the user. However, if you jump into the view controller you'll notice the following method:

```
func view(_ view: ARSKView,
nodeFor anchor: ARAnchor) -> SKNode? {
    // Create and configure a node for the
    // anchor added to the view's session.
    let labelNode = SKLabelNode(text: "Enemy")
    labelNode.horizontalAlignmentMode = .center
    labelNode.verticalAlignmentMode = .center
    return labelNode;
}
```

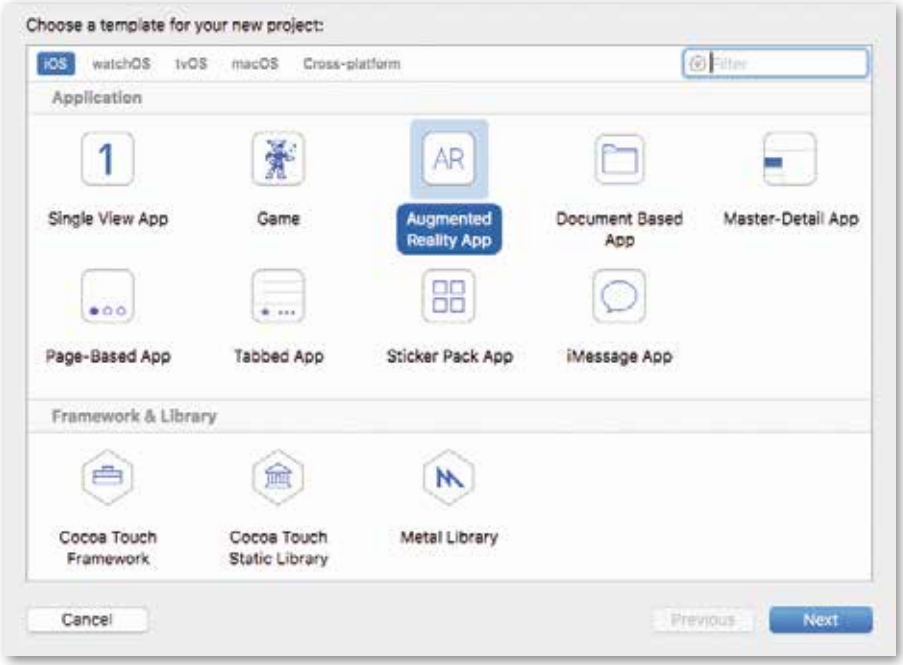


Figure 1: ARKit template selection when creating a new XCode project

For each anchor in the scene, this method is called to return a node for display relative to that anchor. In this case, a label is created and returned. In the sample, you'll notice that it's using an emoji character that resembles a small alien instead of standard text. The end result is pictured in **Figure 2**. Notice that the first screenshot represents a single touch on the screen. The second screenshot demonstrates that as the phone is moved to the left, the alien is anchored in the physical space.

As the sample project demonstrates, anchoring 2D objects in a 3D space is achievable without any heavy lifting using ARKit. In fact, the template project is less than 50 lines of code. For more on this kit and how you can apply it to your own projects, refer to the sidebar for additional links and reference material.

Core ML

The ML in Core ML stands for "machine learning." It's the same machine learning framework that Apple uses within its own services such as Siri, Camera, and QuickType, and now it's available to you as well. Machine learning is a catchy buzz phrase but what does it translate to in terms of useable technology that you can apply? To answer this, let's look at a specific component of Core ML called "Vision." Apple describes Vision as a framework whose features include face tracking, face detection, landmarks, text detection, rectangle detection, barcode detection, object tracking, and image registration.

Let's say, for instance, that you wanted to determine whether any particular picture is of a bedroom or not. If you had a single image of a bedroom and needed to code against that alone, it would be a tough problem to solve. However, if you built a trained model using a number of sample images, all featuring bedrooms, eventually you could train the application on what to look for. As the model analyzes more and more samples, it gets smarter over time.

ARKit

For more information on using the power of ARKit, see https://developer.apple.com/arkit/.

Transformation Matrix

To better understand how the transform matrices work when dealing with 3D space, check out https://en.wikipedia.org/wiki/Transformation_matrix.

Core ML Models

https://developer.apple.com/machine-learning/

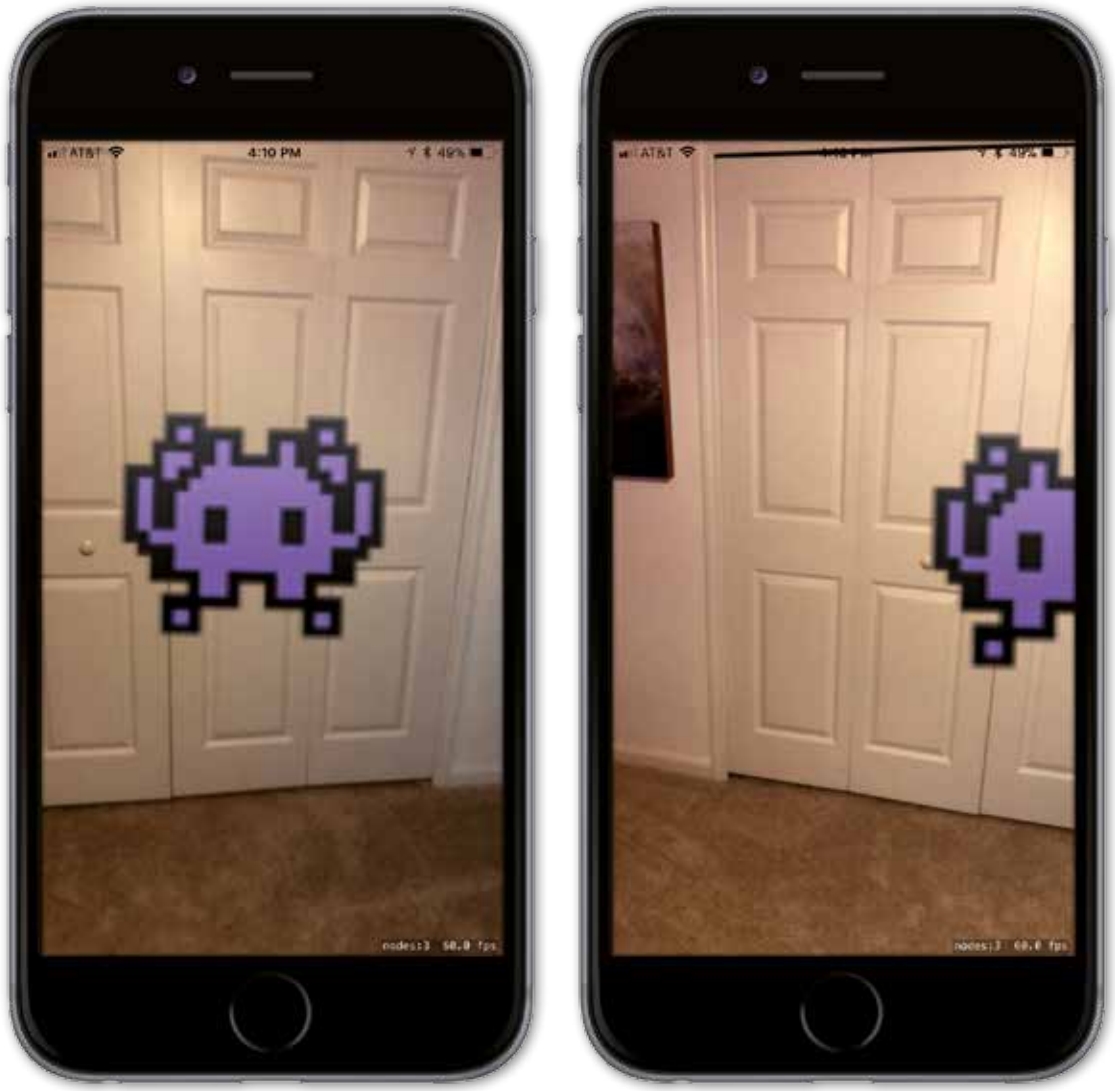


Figure 2: Sample ARKit application demonstrating how to anchor 2D sprites in a physical space

In fact, once you have a model to compare against, working with Core ML is almost trivial. However, creating the model, especially when dealing with images, isn't trivial at all. Luckily, Apple provides a number of models for you to take advantage of out of the box (see sidebar for links). One particular model, **Places205-GoogLeNet**, detects the scene of a specified image from 200+ categories, including "bedroom." Once you have the model loaded and your image converted over to the needed CVPixelBuffer format, analyzing it through an established model is relatively simple, as demonstrated in the following snippet:

```
let model = GoogLeNetPlaces()
guard let prediction = try? model.
prediction(sceneImage: pixelBuffer!)
else { return }
print(prediction.sceneLabel)
```

XCode auto generates a class that wraps an imported model so you can refer to it with ease, as shown in the declaration of the model variable. In this case, the print at the end of the prediction reads "bedroom." The processing and analysis of the image all takes place locally on the device without the need for network calls.

MusicKit

MusicKit lets you integrate Apple Music directly into your custom applications. Once the user approves access to an Apple Music account, you can create playlists, add songs to their library, and play any of the songs from Apple Music's offerings. If the user isn't currently an active Apple Music member, you can prompt them for a free trial without having to leave your application.

When working with MusicKit, most of the functionality is inaccessible until the user grants you permission to the account. Two different permission levels exist:

1. Media Library Authorization: grants access to the user's media library
2. Cloud Service Authorization: grants ability to play back music from the Apple Music catalog or add items to the user's iCloud Music Library

To request access to the user's media library, you use the MPMediaLibrary APIs like this:

```
// check authorization status
guard MPMediaLibrary.authorizationStatus() ==
```

```
.notDetermined else { return }

// request auth
MPMediaLibrary.requestAuthorization { (_) in
   // handle response
}
```

If you're interested in using the cloud services, you'll have to use the SKCloudServiceController APIs. Although similar in format, the authorization request differs from the MPMediaLibrary, as demonstrated in the following snippet:

```
// check authorization status
guard SKCloudServiceController.
authorizationStatus() == .notDetermined
else { return }

// request auth
SKCloudServiceController.requestAuthorization {
[weak self] (authorizationStatus) in
    switch authorizationStatus {
    case .authorized:
        // user granted auth
    default:
        break
    }
}
```

After obtaining a user's permission, if you plan to make a call to the Apple Music API, each request must be signed with a developer token. This is different than the dev certificates you use to sign your application. It's a specific token for use with MusicKit. To obtain that token, you first need to create a MusicKit signing key within your Apple Developer Program account. Specific instructions on how to do that can be found at http://developer.apple.com/go/?id=apple-music-keys-and-tokens. That link also demonstrates the not-so-straightforward process for how you then take your signing key and use it to construct a developer token. MusicKit uses a JWT spec (JSON Web Tokens) and requires you to create a header JSON payload, with specific information, in order to authenticate properly to the service.

Not only will you need a developer token and user permission, but if you want to make requests to the music API that targets user-specific data, you'll also need to obtain a user token. You can do this using the requestUserToken method on the SKCloudServiceController() object.

MusicKit has a series of challenges to overcome in terms of tokens, permission, and authorization, but once you hurdle those obstacles, the service is relatively easy to work with. Let's look at an example where your application wants to create a new playlist for a user. The following code snippet demonstrates how you would achieve that using MPMediaLibrary.

```
/*
Create an instance of `UUID` to identify the new
playlist.  If you wish to be able to retrieve
this playlist in the future, save this UUID
in your application for future use.
*/
let playlistUUID = UUID()

// this represents the metadata to associate
// with the new playlist.
var playlistCreationMetadata =
MPMediaPlaylistCreationMetadata(name:
"My Playlist")
playlistCreationMetadata.descriptionText =
"This playlist contains awesome items."

// Request the new or existing playlist
MPMediaLibrary.default().getPlaylist(with:
playlistUUID, creationMetadata:
playlistCreationMetadata) {
(playlist, error) in
    guard error == nil else {
        // Handle Error accordingly
    }

    self.mediaPlaylist = playlist
}
```

In this example, self.mediaPlaylist is an instance of MPMediaPlaylist. This object has a method called addItem that you can then use to add songs to the playlist you just created. Apple has a sample project that further demonstrates and expands on this functionality. You can find it linked in the side bar.

PDFKit

Originally an OSX framework, Apple has now made PDFKit's functionality fully available on iOS 11. As the title suggests, PDFKit is used to manipulate and display PDF documents to users within your applications. It also supports more advanced features, such as annotations. PDFKit comes stocked with two view options:

- **PDFView**: encapsulates the functionality of PDFKit into a single view widget that you can add to your app using Interface Builder. This view lets you display a PDF, select content, navigate via pages, set zoom level, copy textual content, and even tracks page history.
- **PDFThumbnailView**: view that contains a set of thumbnails, each representing a page from a PDF document. This view has several methods to track user selection of pages and other interactions.

PDFKit even supports advanced features, like annotations.

Loading a PDF in a PDFView takes very minimal effort. The following snippet of code demonstrates what that implementation looks like:

```
// add PDFView to the display
pdfView = PDFView()
view.addSubview(pdfView)

// .. set up constraints for subview ..

// load PDF and display it immediately
let url = Bundle.main.url(forResource:
"code-mag-demo", withExtension: "pdf")!
pdfView.document = PDFDocument(url: url)
```

MusicKit Sample Project

The sample project is available for download at https://developer.apple.com/musickit/.

PDFKit Sample Project

The sample project is available for download at https://github.com/tzshlyt/iOS11-PDFKit-Example.

Figure 3: Example of a PDF display and page thumbnail display using PDFKit

Although the PDFView class simplifies things for you, if you desire a more robust or custom solution, you can create a custom PDF viewer using the PDF Kit utility classes directly. **Figure 3** demonstrates a PDF running within PDFView and its corresponding page display using PDFThumbnailView. Check out the GitHub link in the sidebar for access to the source of this sample application.

Drag and Drop

The aim of iOS 11 is to improve multitasking with the introduction of Drag and Drop; adding a system-wide way for users to move text, images, and files among applications. Although this is mostly useful on iPad devices, the functionality does exist in a more limited capacity on phones as well. Drag and Drop only works within a single app on iPhone whereas iPad devices can drag and drop contents between different apps. Drag and Drop also supports multi-touch, letting you select and manipulate multiple items simultaneously.

UITableView and UICollectionView also got an upgrade with built in drag and drop support. Each of those classes now has a dragDelegate and dropDelegate property. Whichever class you assign as the drag and drop delegate must then adhere to the UITableViewDragDelegate and UITableViewDropDelegate protocols. Additionally, to support drag functionality, you'll want to enable the dragInteractionEnabled property on the respective table or collection view. The drag protocol introduces the following method:

```
func tableView(_ tableView: UITableView,
itemsForBeginning session: UIDragSession,
at indexPath: IndexPath) -> [UIDragItem] {
    // your code here
}
```

This method gets called whenever the user initiates a drag operation by holding down a finger on a table cell. The method's responsibility is to return an array of dragged items that the user is now manipulating. If this method returns an empty array it's the equivalent to declining the drag operation. Once the user has an item or series of items they're dragging, they need a place to drop them. UITableViewDropDelegate requires the following method:

```
func tableView(_ tableView: UITableView,
performDropWith coordinator:
UITableViewDropCoordinator) {
```

```
    // your code here
}
```

This method is a bit more involved as it must handle several different scenarios. For instance, the functionality to handle a dropped item might differ based on the number of items being dropped or by the location where the user dropped them. It's possible to get back a location of the drop but it's also plausible that the user released the items over some whitespace in the table and there's no corresponding destination path reported as a result. So, an example of how you might have to handle this could look like:

```
let dropDestinationPath: IndexPath

if let indexPath =
coordinator.destinationIndexPath {
  // if drop destination was captured
  dropDestinationPath = indexPath
} else {
  // if no given drop destination, use the
  // last row of the last section as the
  // path for inserting the dropped items
  let lastSection =
  tableView.numberOfSections - 1
  let lastRow = tableView.numberOfRows
  (inSection: section)
  dropDestinationPath = IndexPath(row:
  row, section: section)
}
```

Here, you calculate the index to insert the dropped items by first checking whether the drop location could be determined by the user drop event. If not, you simply grab the last row of the last section, which you can use to insert the items at the end of the table. This is merely an example of what you could do, but it's by no means the only way to handle dropped items. Once you've determined where you're going to put the items being dropped, you need to obtain the dropped items. The coordinator has a session property that you can use to call coordinator.session.loadObjects to retrieve those items.

There's an extended set of functionality within this feature set for you to take advantage of. It's definitely worth reading more about and you can do that here https://developer.apple.com/ios/drag-and-drop/.

Other Notable Features

The features mentioned previously only represent part of what iOS 11 offers. It also includes several other notable features and functionality worth mentioning.

- **Files:** iOS 11 introduces the new Files app, providing a centralized space to search, browse, and organize files. Your custom applications can read and write files to that application as well.
- **NFC Reading:** iPhone 7 introduced NFC hardware support and with iOS 11, developers can now use that support to detect NFC NDEF tags nearby.
- **Metal 2:** Offers significant increases to performance with near-direct access to the GPU. There are new APIs, such as imageblocks, tile shading, threadgroup sharing, and more.
- **Business Chat:** Lets you connect to your customers directly through iMessage. It's essentially a customer-service platform that allows your customers to reach you via a seamless native experience, regardless of whether or not you have a native application. For instance, if your business is registered in Business Chat, Safari automatically shows the chat button when a user searches your business in Google. The chat integrates directly with Safari, Siri, Spotlight, and Maps to make it easier than ever for customers to reach you. This feature is still in developer preview and hasn't launched to the public yet. Find out more here: https://developer.apple.com/business-chat/.

The iOS 11 App Store

The AppStore underwent major changes in iOS 11. Not only did the storefront and app listings get visual changes, but developers also got new deployment options. In terms of visual changes, the most noticeable is the introduction of the "Today" tab. The Today tab is your destination for original stories, premiers, new releases, app of the day, and game of the day. Think of it as your AppStore news feed. Many of these stories feature quotes from the developers and offer a unique perspective on some of the applications you use on a daily basis. **Figure 4** demonstrates what the Today tab looks like.

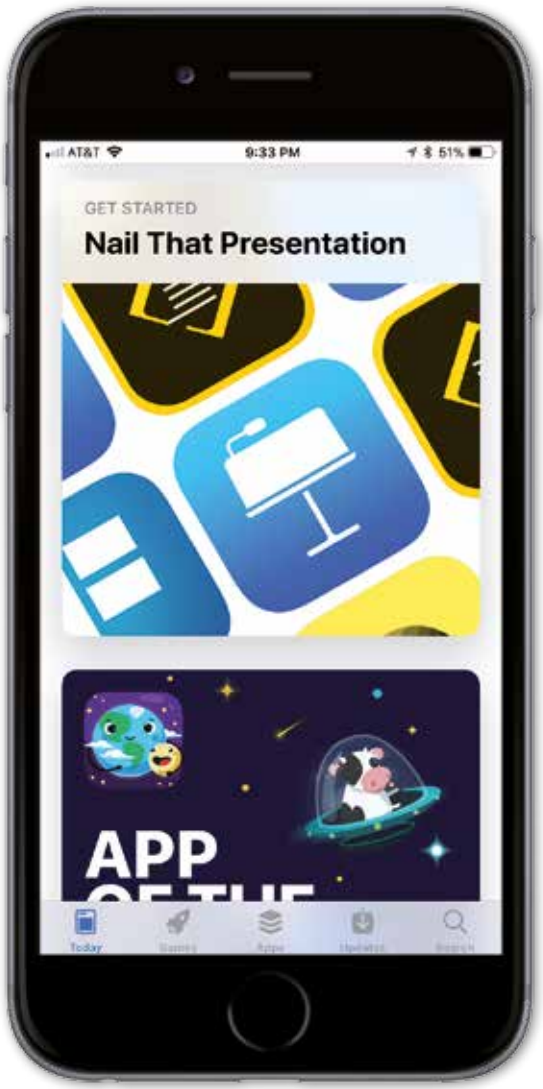


Figure 4: New Today tab in the iOS 11 AppStore

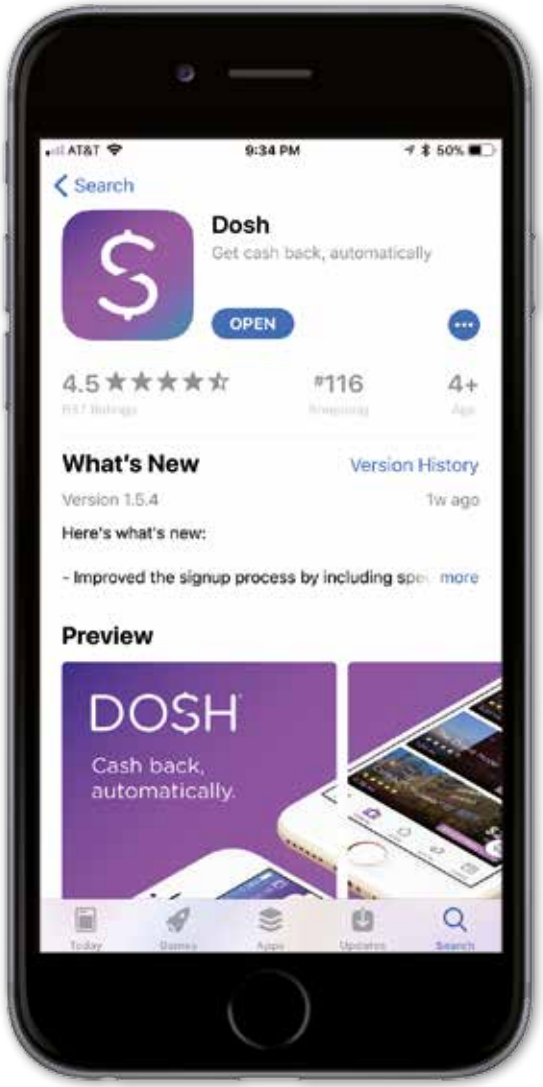


Figure 5: New iOS 11 AppStore application details view layout

Rather than simply being an app category, games now live as a separate entity within the AppStore. They have their own dedicated tab that's formatted into sections similar to what you'd expect from the normal apps landing page: recommendations, videos, top charts, categories, and hand-picked collections.

The meatiest change is the product details page when you tap into an application, as shown in **Figure 5**. A slew of new options exist to help you better showcase your work. You can show more of your user experience using up to three app previews that auto play, making it easier than ever for customers to see how your app works or how your game plays. You can also now localize all of your app previews and screenshots. Additionally, the new "subtitle" field allows you to add a tag line beneath your app title, giving you space to convey the intention or value proposition of your application. The listing also shows where your application ranks within its category. Ratings have also been moved to front and center, with a large display at the top of the listing.

Apple also made a significant change to how in-app purchases get showcased on your app details page. You can promote up to 20 in-app purchases, including subscriptions, from the details page, and even let users start a purchase before downloading your application. In-app purchases can also appear in search results and even get featured by the editorial team. **Figure 6** gives you an idea of how this looks.

In terms of deployment options; developers have a few new choices when they deploy an application update:

- **Phased Release:** Lets you gradually release your application update over a seven-day period to users who have automatic updates enabled. Users can override this via a manual update. You can pause a phased release for up to 30 days or choose at any point and time to release to all users.
- **Reset Summary Rating:** Lets you reset your application's summary rating for all territories when you release a new version. However, once released with this option, you won't be able to restore a previous rating summary. This doesn't erase customer reviews. It affects the star rating display at the top of your listing. This is especially helpful if you had

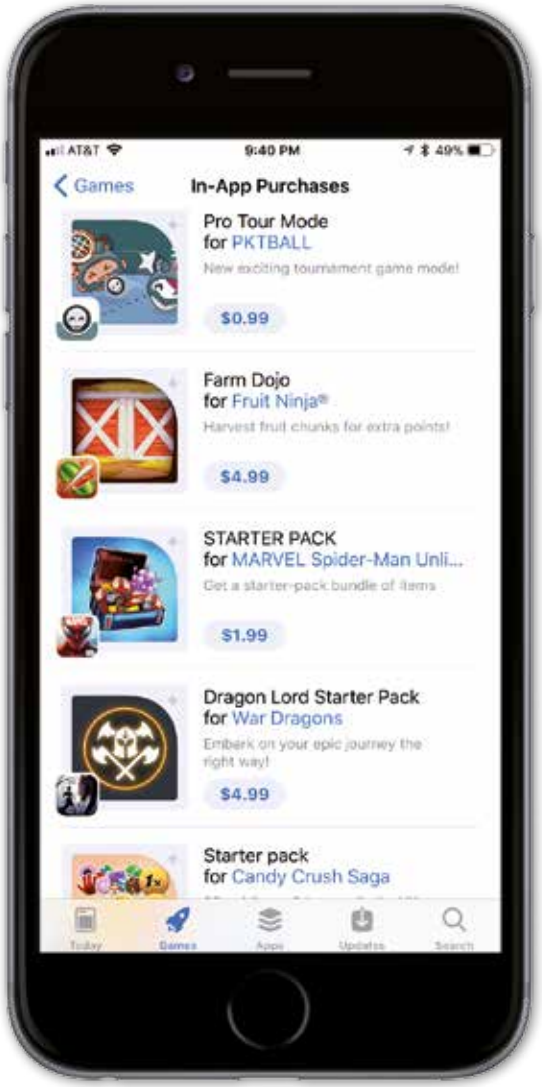


Figure 6: Example of how in-app purchases can exist as stand-alone items in the iOS 11 AppStore

Figure 7: Demonstration of the iPhone X form factor and the safe area offsets

a bad release and got an influx of negative reviews, follow it up with a fix, and want to clear out the resulting negativity. It's also nice not having your stars reset to zero with every minor update.

The iPhone X

Apple recently released the iPhone X that brings its own set of changes. Most notable is the display. It's an edge-to-edge Super Retina display that has an unconventional form factor. The top and bottom of the display are designated "safe areas" that are reserved for specific system UI or gestures. Although your applications still exist in that space, there is a new set of interface guidelines to adhere to in terms of what you can and cannot do in those spaces. Most notably, there shouldn't be actionable items in the safe area. Depending on how your application is currently laid out, you might end up with your old UI overlapping with these spaces and you'll likely need to make some changes to optimize. **Figure 7** demonstrates a traditional iPhone display in conjunction with the iPhone X. Notice that the navigation and tab bars are extended in height, leaving blank space where the safe areas are defined.

Introducing a new SafeAreaLayoutGuide, iOS 11 lets you base constraints off of the safe area bounds rather than positioning items relative to the superview. Basically, these smart guides position your content relative to the safe area when on a device that needs or requires it, and position elements relative to the superview bounds otherwise. **Figure 8** gives you an idea of where the designated safe areas are on the iPhone X. This is the primary means for updating your current applications to be iPhone X compatible. If you're using interface builder and auto-layout, it's as simple as selecting your constraint and changing it to be relative to the safe area rather than the superview.

The status bar is also a different height on the iPhone X. If your application is using a fixed status bar height, you'll likely need to rework that to support the change. It's also worth noting that the status bar on the iPhone X doesn't change height when background tasks like voice recording or location tracking are active. The interface guidelines also restrict you from masking or calling attention to the key display features such as the rounded corners, sensor housing, or home screen indicator. Basi-

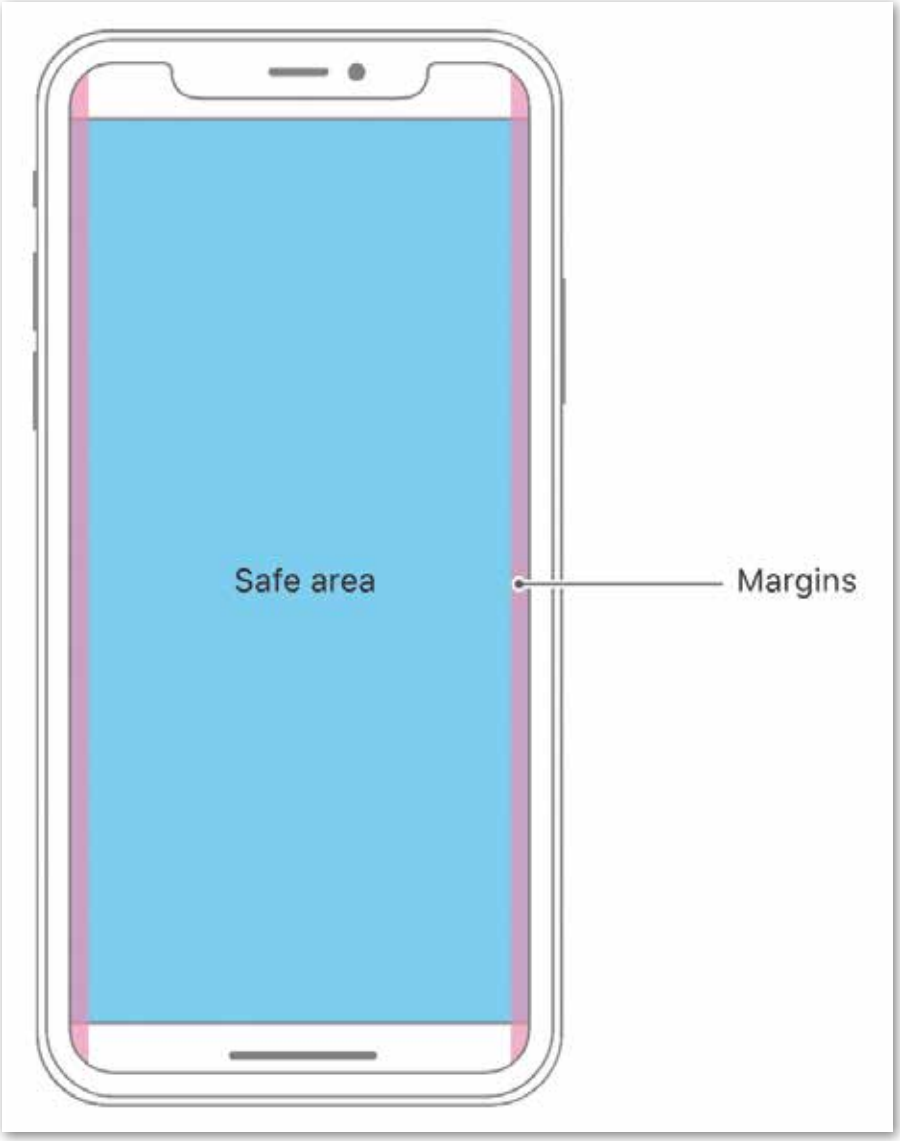


Figure 8: The iPhone X safe area diagram

cally, using black bars at the top and the bottom or creating visual adornments that interact with those spaces are considered grounds for rejection by the store.

XCode is now equipped with an iPhone X simulator so you can test your latest designs to make sure they are compatible. You'll want to pay extra attention if your application supports both portrait and landscape mode, as the safe areas shift as the device is rotated.

There's also an entirely separate set of considerations for optimizing your mobile Web experiences for the iPhone X. Apple has done a good job documenting those approaches over at https://webkit.org/blog/7929/designing-websites-for-iphone-x/.

The iPhone X brings with it more than a new form factor. It also comes equipped with face-tracking technology. You can take advantage of this tech via ARKit, allowing you to animate 3D content that follows the user's face and matches a user's facial expressions. Earlier, I talked about launching an ARKit session using a world-tracking configuration. ARKit also has a ARFaceTrackingConfiguration. Using this configuration, ARKit automatically attaches ARFaceAnchors to the scene that contain information about the face, including orientation and position, using the face anchors so that you can position 3D items relative to the face. This is great for interactive photo filters and other features where you want to manipulate the appearance of a user's face. It's worth noting that ARKit only tracks one face at a time. If more faces are present, it decides which is the most prevalent.

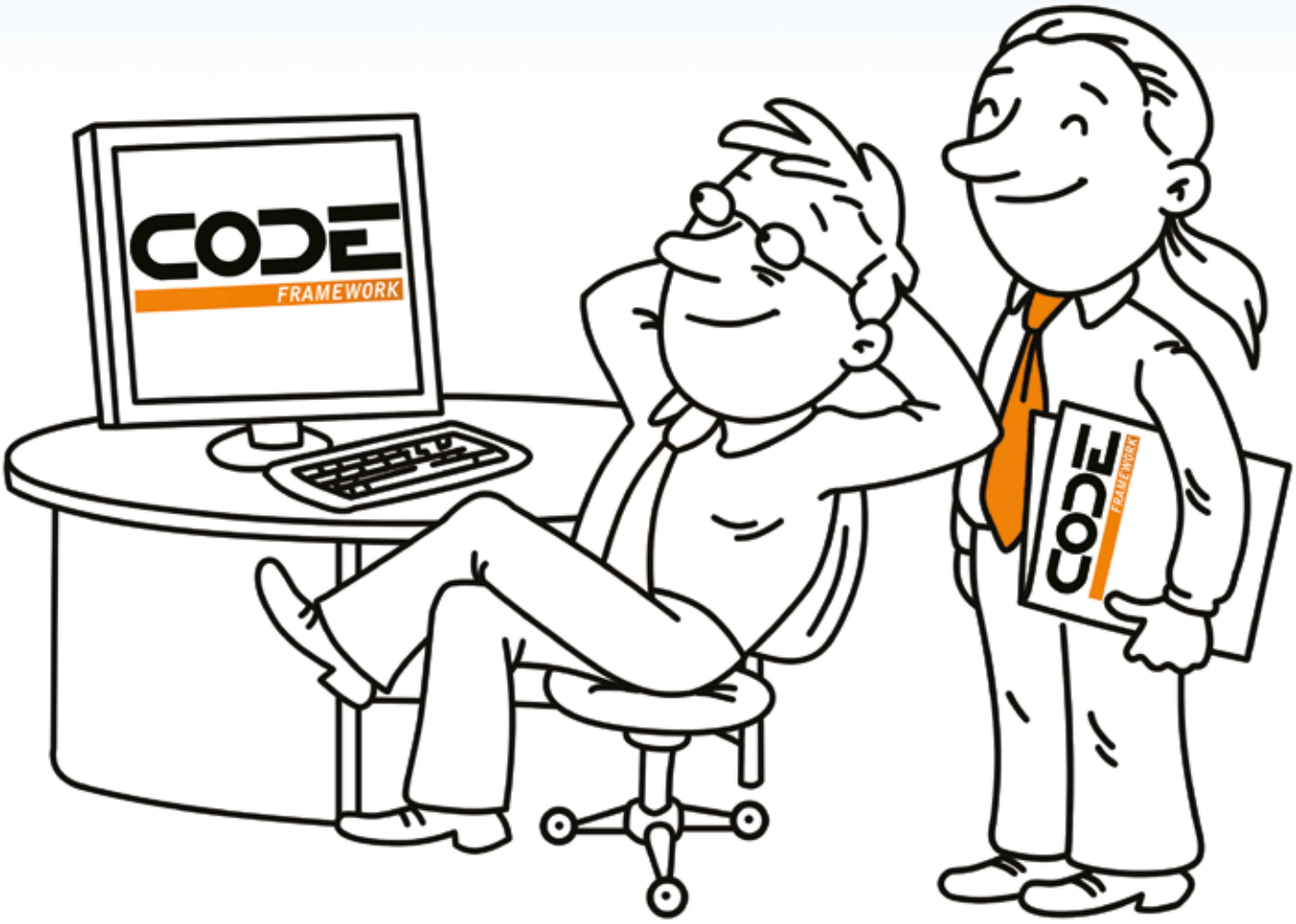
The iPhone X comes equipped with face-tracking technology, allowing you to animate 3D content that follows the user's face and matches a user's facial expressions.

Wrapping Up

Hopefully, this article has shown you the potential of iOS 11 and what it has to offer developers. As Apple continues to iterate and innovate, developers have an ever-increasing tool kit to create amazing experiences for end users. Make sure to check out the sidebars for additional links to reference material and code samples for new iOS 11 features and frameworks.

Jason Bender

CODE Framework:
Business Applications Made Easy



Architected by Markus Egger and the experts at CODE Magazine, CODE Framework is the world's most productive, maintainable, and reusable business application development framework for today's developers. CODE Framework supports existing application interoperability or can be used as a foundation for ground-up development. Best of all, it's free, open source, and professionally supported.

Download CODE Framework at www.codemag.com/framework

Helping Companies Build Better Software Since 1993

www.codemag.com/framework
832-717-4445 ext. 9 • info@codemag.com

CODE
FRAMEWORK

How I Learned to Stop Worrying and Love Continuous Integration

Imagine you're a cook in a restaurant kitchen. A customer orders a cheeseburger. Simple enough; you've made thousands of cheeseburgers and it's something you're known for. You get to work, craft the best burger you can, and send it out to the table. A minute later, the waiter brings it back. "They wanted Swiss cheese," he says. Sighing, you cook another cheeseburger,

Geoff Callaghan
gcallaghan@eps-software.com

Geoff Callaghan is the development team lead in the Denver, Colorado office of StarRez, the market leader in providing housing software solutions to universities and colleges around the world. He's worked on projects that run the gamut from debugging legacy code on integrated circuits in industrial applications to creating multi-tier software using WPF, WCF, and CODE Framework. Currently, his work is focused on .NET, MVC, and TypeScript. Having worked as a manager, a technical lead, and a business analyst, he's familiar with all stages of the software development process, and is as at home in front of customers as he is behind a keyboard.

Geoff lives in Colorado with his beautiful wife Laura. In his spare time, he's an avid reader, an amateur musician, and a bewildered gardener.

this time adding the correct cheese. Shortly thereafter, the burger is back on your table. Too much mayonnaise. Again and again, no matter how perfect the burger seems to you, it's never quite right. No onions. Too many onions. Too rare. Too well done. Is that bun gluten free? By the time you get it perfect, you've thrown away seventeen burgers (or let's face it, given them to a very happy kitchen staff), and lots of other orders went out late. You're frustrated, the customers are annoyed, and the restaurant is losing money.

If you're new to Continuous Integration, this scenario may feel familiar.

Continuous Integration (CI) is a process by which code is pulled from a central repository, and changes made are merged back into that repository on a frequent basis. The idea is to make small changes often, as opposed to large changes that won't be complete for weeks or months. The end result, ideally, is that integrating the changes into the main code base can be mostly automatic. Time is spent on development, rather than on integrating large code changes that don't merge well with other large code changes added to the code base by other developers.

I've been hearing about CI for some years and I've experienced it on a small scale. I recently took a position at a company called StarRez, which uses a fairly rigorous CI process. Specifically, we use Jira, BitBucket, and Bamboo from the Atlassian suite. I've managed a fair number of development projects and I was very excited at the power and control promised by these tools, especially considering that the bulk of the development team is on the other side of the planet in Australia. Before new code is merged into the product, multiple experienced developers review it for correctness and adherence to coding standards, while also taking the opportunity to point out the impact the new code might have on other aspects of the product. Aspects that, as the new guy on the team, I quite possibly didn't know anything about. What a great way to ensure code quality! What a great way for junior developers to learn the ropes and for experienced, but newly hired, developers to learn the product!

At least, that's what my brain said.

Unfortunately, developers aren't all brain. We're human, and a process like this doesn't just impact our intellect. It hits us where we live: directly in the ego. Those of us who have been around for a while are used to a lot of autonomy, especially if we've mostly worked alone or on smaller teams. We're proud of our code, and rightly so. Submitting it for approval, and then re-submitting and re-submitting, can be very humbling. It can also take a lot longer than we're used to. Delays can increase if the developers reviewing your code are in a different time zone. A simple change can take a whole day to go through, and each new change means another day. It's easy to tell yourself that if it weren't for all these hoops to jump through, you could release this fix to the customer in twenty minutes. It's just one line of code. Is it worth it? Can't we make an exception in this case?

Shortly after taking this position, I heard about a company that was looking for someone to help sell their developers on the benefits of CI. They were having a hard time understanding why it was necessary, and they were concerned that their productivity would decrease significantly. I could sympathize. I'm hopeful that this article will help developers see CI as an essential tool for code quality, and also see how using CI correctly can improve product quality, speed up big identification, and most importantly, teach us how to be better and more productive developers.

Git Basics

First, I'd like to go through some of the basics. If you've used any source control tool, such as Team Foundation Server or Git, you're familiar with the concept. Code is checked out of a repository, you make your changes, do some testing, and check it back in. This works well, as far as it goes, as long as everyone is careful. No one likes to wake up in the morning and discover a broken build. And no one likes to be the one who broke it.

CI takes us to another level of code sharing. I'll be using Visual Studio and GitHub as my tools, so that's where my screenshots come from. Of course, there are other toolsets and many Git clients. Feel free to use whatever tools you prefer. I won't be going into too much technical detail, so the differences between tools should be fairly minor.

Connect to a Repository

I'll assume here that you're going to be working on an existing project as opposed to starting something new. To connect to a repository, clone it. You'll need the URL for the Git repository. In Visual Studio, open the Team Explorer window and click on the Manage Connections icon. In **Figure 1**, it looks like a green plug.

Now click Clone. You'll be shown a screen like **Figure 2**, where you can enter the URL of the Git repository and the local directory where you want your code to live.

Click the Clone button, and the code is brought down to your computer. The repository in **Figure 2** shows the address of the repository for a new killer app. It has the miraculous ability to display a console window that spells out "Hello World". Pressing Enter makes the window go away. Who wouldn't pay 99 cents for that? Go ahead and clone the repository and open the HelloWorld solution.

Create a Branch

Here's the problem: StarRez, as mentioned above, is an Australian company. "Hello World" is great, but I'd like the wording to be a bit more Australian. To accomplish this, I create a branch in which to make the changes. On GitHub, this is done by going to the repository and clicking on the Branch button. You can either select an existing branch or create a new one by typing in a branch name, as shown in **Figure 3**.

Open Your Branch

Now that your branch has been created, you need to download it to your computer. Back in Team Explorer, click on the Home button, then click Sync, and then Fetch. Once the Fetch is complete, click on Home again, and then Branches. You should be presented with a screen that looks like **Figure 4**.

In the Branches window, as shown in **Figure 5**, you should now see the new Australianify branch. Double click on it to open the branch.

Make Your Changes

This is the familiar part. Just make the changes to your code and test it to make sure it's right.

Stage Your Changes

Staging the changes means that you're pushing your code into the repository, but not yet requesting that the changes become a permanent part of the code base. You can stage as many times as you like, and you can also revert any of those staged changes. If you're working in multiple branches, you'll have to stage before you switch among them.

Switch to the Changes tab (as shown in **Figure 6**) and click Commit All.

You'll be notified that a commit has been created locally. If you're confident that your changes are complete, you can now Sync them with the Git server.

Push Your Changes

This is a small change, so you're now ready to request that the changes become a permanent part of the application. Return to Team Explorer and click Push.

The Approval Process

Up to this point, you've been working in a vacuum. Now that you believe the code is complete, it's time to involve other members of the team.

Create a Pull Request

A Pull Request is how you submit your code for the approval of other team members. Depending on the tools and the process your organization uses, approval from certain reviewers may be required before you're permitted to merge your code with the main branch.

Wait...

This, emotionally speaking, is the hard part. You move on to a new task, putting this one aside. Tomorrow, or possibly after the weekend, you see that your code has been reviewed,

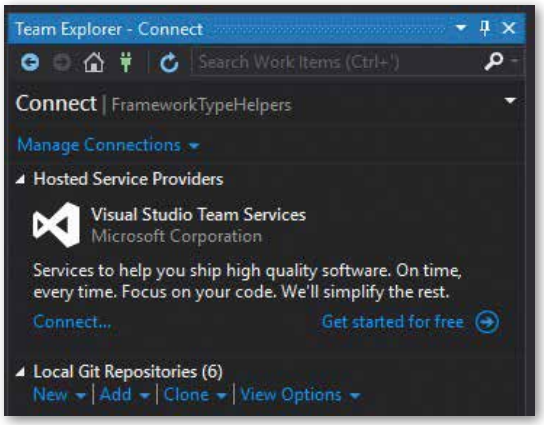


Figure 1: The Team Explorer in Visual Studio

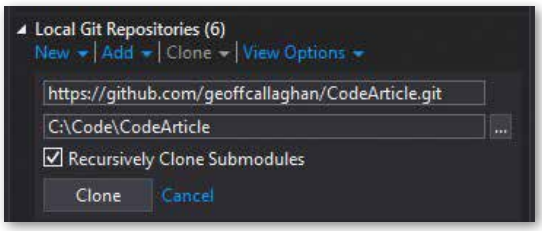


Figure 2: Cloning a repository

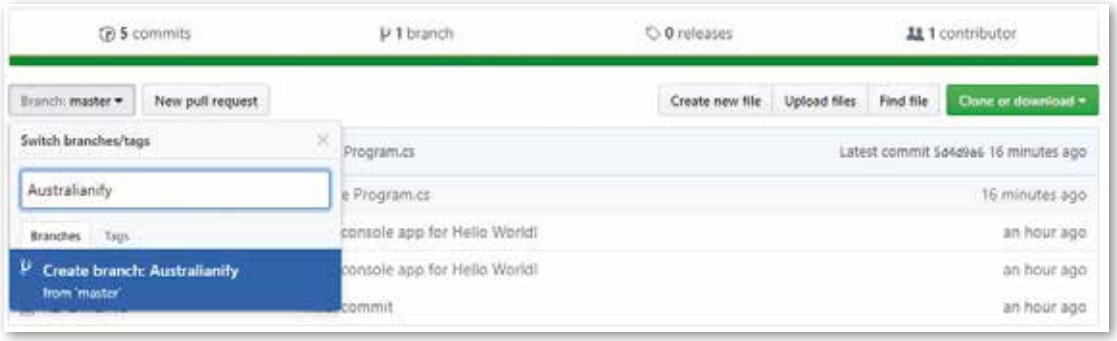


Figure 3: Creating a branch in a GitHub repository

and hopefully, that it passed muster. Now all you have to do is send the task to QA for testing, and if they don't find any issues, you should be ready to merge the code into the main branch. If, on the other hand, your code is found wanting, you'll need to go back to that branch and try again.

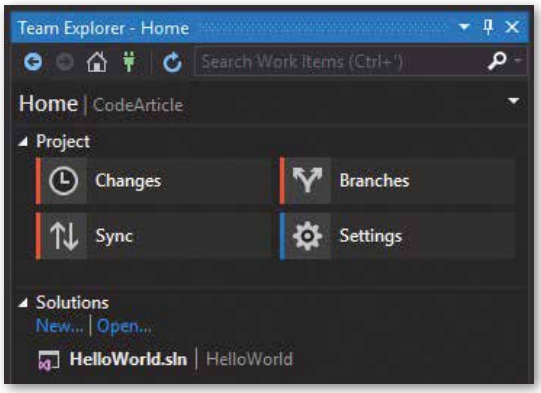


Figure 4: Syncing a repository from Visual Studio Team Explorer

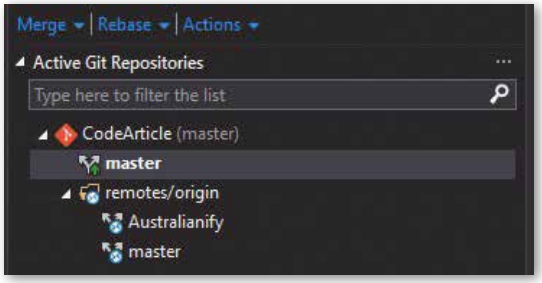


Figure 5: Opening a branch

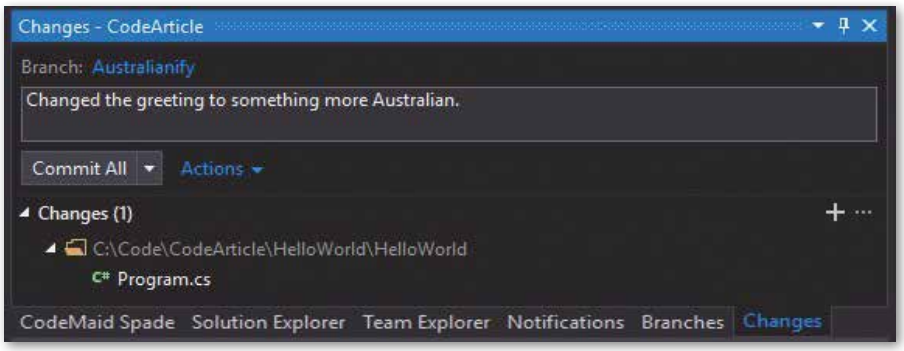


Figure 6: Staging code changes

Remember, the first change was to make the greeting a bit more Australia-friendly. **Figure 7** shows the code change:

Simple enough, right? This should fly through the approval process. You go home and sleep the sleep of the just. Morning arrives, you sit down with your coffee and pull up GitHub. Yawn. Stretch. Sip. Wait, what?!? Looks like it wasn't that simple **(Figure 8)**.

Well, that's what I get for learning about Australia through TV commercials. OK, no big deal. I do a little research, ask a few of the other developers, and decide to change the greeting to "How are you going?" After a quick test, I push the branch to again.

The next day, you're presented with **Figure 9**.

Hmm. Ok, fair enough. I'll find the new helper class and implement it.

Another day passes. See **Figure 10**.

On this one, my face is a little red. I was given a copy of the coding standards when I was hired. I got a little sloppy here and forgot to check. This time, I'll check my code against all the standards.

Finally, I get the stamp of approval. (See **Figure 11.**)

Phew! That task is complete. Now I can focus on the other tasks on my plate. I should probably go ahead and order that Tesla, too.

Automated Builds and Tests

Each branch is required to pass one or more automated builds prior to merging. This ensures that our code doesn't cause unintended consequences and that the merged code compiles properly.

Automated tests have also been added to the build process to further protect us from the possibility of releasing buggy code. In the example above, there may have been a test requiring that the output be "Hello World". The test would have to change to allow for the new language, and possibly expand to test for all supported languages.

Merge

Merging the code back into the main product is the final step. Code from the branch is integrated into the master branch, and becomes part of the product.

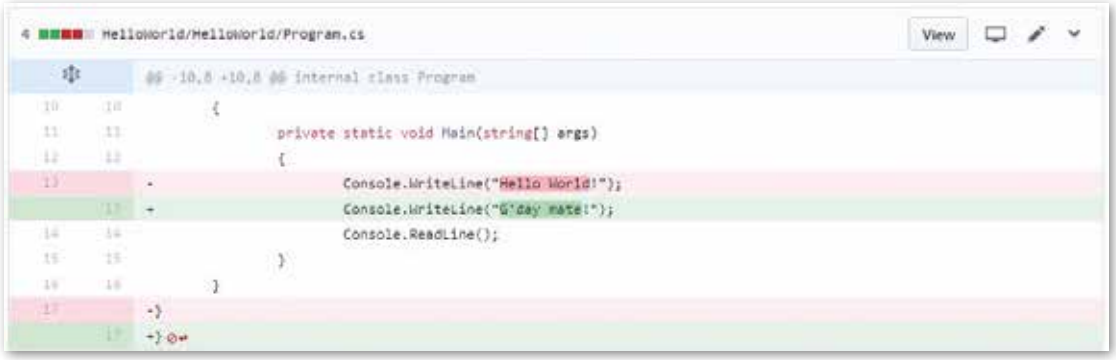


Figure 7: Code differences from a GitHub Pull Request.

Figure 8: A comment on a Pull Request.

Figure 9: Not quite there yet.

Figure 10: Oops.

How It Feels

The approval process of CI is the hardest part to deal with. Ideally, you check in your code, assign some kindhearted developers to review it, they click Approve, and there is much rejoicing. In reality, there's a decent chance you forgot something, or didn't know something, or just that your team lead would prefer you handle something differently. If you're sitting next to that person in an office, this isn't a big deal. You can run through the approval process several times on any given day. The code is still fresh in your mind, and if you've moved on to another task, you still shouldn't have too much trouble getting back into this one.

Remote work adds another wrinkle, especially if you're in another time zone. If the code was written on a Friday, it could be three days before you're notified of a problem with the code. You're probably deep into another task. Switching back to this one—again—can be very frustrating. It's bad enough to discover that you misunderstood a key part of the task, or that there's an unintended consequence. It's even worse to find out you've missed a sprint deadline because one of the reviewers pointed out that you used single quotes when you should have used double quotes. More on that later. Hi, Murray.

How to Make It Feel Better

There are a lot of things you can do to make sure your code gets through the approval process as soon as possible. There are also some helpful ways to look at the process to help you see it as a positive experience. Here are a few that work for me:

Coding Standards

Most software companies have a set of coding standards. Verifying that these standards are being adhered to is part of approving a Pull Request. It may seem like the reviewers are nit-picking, but those standards are there for a reason.

For example, StarRez has a standard of using double quotes instead of single quotes in their JavaScript and TypeScript files. I have a bad habit of forgetting to verify that I've used the correct quotes before I check in the code. Each time I forget, it's at least another day added to the task. A simple once-over before checking in the file can ensure that you don't lose time for failing to proofread your code for typos. Also, make sure that you named your variables according to standards, used the appropriate code cleanup tools (such as CodeMaid or ReSharper) and checked your spelling. Even in the comments. I know it doesn't impact the build, but misspelled words drive me crazy and I'm sure I'm not the only one.

Figure 11: Code is approved.

Double Check It Yourself

Once the code is committed, you have an opportunity to give the code a once-over to make sure it all looks good. Your Git client has a Diff tool of some kind, which allows you to compare your new code to the old code. Look at it as if you were the reviewer, not the coder. Keep a list of items you tend to forget, and verify that you haven't forgotten them this time. Until that list shrinks to nothing, use it every time.

> A simple once-over before checking in the file can ensure that you don't lose time for failing to proofread your code for typos.

Ask for Advice

It's always a good idea to collaborate on a task, especially if it's in an area where you don't have a lot of expertise. Try to find some time to work one-on-one with another developer who has that expertise, preferably a developer who'll be reviewing your code.

It's also a good idea to do some Googling to get tips from other sources. A lot of companies use various CI tools. Other developers run into the same problems you do. Their insight could prove very helpful. Perhaps you could even find a concise, witty, well-written article in a top-notch technical magazine and use it to make your life easier <wink wink>.

Insist on Clear Specs

Whether you've been assigned a task by a supervisor or just grabbed one from the backlog, it's important that you understand it well before you begin coding. Sometimes tasks aren't as simple as they seem at first glance. You should feel free to request more detail. It's always better to get the specifications as clear as possible to avoid re-work and delays. The sooner an issue is discovered, the cheaper it is to fix. Keep in mind that it's your responsibility to make sure you understand the task before you begin coding.

Take Advantage of the Reviewer Role

Keep in mind that as a member of the team, you'll be asked to review code written by other team members. This can be an opportunity to learn new aspects of the application, new techniques, new libraries. The comments from the other reviewers might point out issues you'll run into in your own code. Think of it as a meeting in a conference room. Ask questions, listen carefully, learn as much as you can.

Think of Pull Requests as Conversations

Sometimes when you're stuck on something, it can be very helpful to talk through the problem with other developers. Pull Requests can be used for this purpose. If you run into a roadblock, or can't decide which of multiple options is preferable, create a Pull Request and add reviewers you think might have some insight. Add some comments yourself to highlight the areas you'd like to discuss. Reviewers may even suggest other developers who could help. Once you have all the information you need, you can complete the code and add more reviewers.

Summary

Granted, the example I've used is not very realistic, but it's easy to extrapolate these steps to a task in your not-too-distant past. Take a moment to think about the implications of what can be achieved using CI. There may be some added hoops to jump through and it may take a bit longer to complete tasks. The payoff is that the product is far less likely to contain bugs. The code is guaranteed to be cleaner, because team leads have signed off on it before allowing it to be merged. Multiple developers have at least a passing familiarity with every chunk of code that's been added to the product. Automated builds have run to ensure that the code compiles. There may even be a suite of tests ensuring that no functionality was broken by this change. You've achieved a higher level of quality, reliability, and maintainability.

The thing to remember is that we all have something to teach and we all have something to learn. The review process is not only an opportunity to learn more about the software you've been hired to help build, it's also an opportunity to increase your skills as a developer, and to pass that knowledge on to other developers. Maybe there's a new class in C# you've never used. Maybe the product uses a third-party library you didn't know was there. Maybe it's just a new technique you haven't seen before. These things can be pointed out during the review process, and now that you know them, they become part of your toolbox.

Geoff Callaghan

(Continued from 74)

the analysis process around what the risks are, you begin to get a better feel for when the risks will be greatest. If asking your boss for a promotion is based on your skill, then making that request right after you've demonstrated your skill to the entire company is a good time to begin that conversation. "Hey, boss, thanks for the compliment on my presentation. I think it's a good example of the stuff I bring to the company every day, and I'd like to open the conversation with you about moving up in the company to a position where I can bring that stuff to a wider group and have a bigger benefit to the firm." Conversely, if your team has just taken a hit (app outage, for example), that's probably not a great time to start that discussion, even if the fault for that failure was nowhere near you; you may be seen as trying to dodge responsibility, or at best, trying to escape the team in a low moment, neither of which looks good.

Finally, recognize that failure—a significant source of fear—is not a terrible thing. Carol Dweck is a psychologist who's written a great deal about the "growth mindset," and a large part of that mindset is the realization that failure is not an indication that you are a terrible person, but an opportunity to learn a way that doesn't work. She talks about the "fixed mindset," which suggests that intelligence (among other qualities) is a reflection of what you are, as opposed to something that you can improve over time, the "growth mindset." Growth doesn't happen without failure, though. Watch babies trying to learn to walk, or talk, or eat, or.... A one-year-old toddles a few steps then falls right on his rump, a clear failure. Baby just gets back up, tries again. Eventually he walks, runs, and learns to chase down programmers in the halls who didn't show up for the status meeting. Again. Getting comfortable with fear means getting comfortable with failure.

And let's be honest—every time your code compiles, that's because of a previous failure. What programmer could possibly be any good at their craft if they gave up the first time code didn't compile?

Summary

Let me be very clear: I don't expect that anybody will be able to read these four points, nod, maybe print a copy of it to read on the train to work every day, and whammo! Fear mastered. This is a life-long commitment, and there will be good days, and bad days, and days when it's the furthest thing from your mind because dude, life. It's a journey, not a quick-fix.

But the next time you find yourself hesitating because you're afraid of whatever-happens-next, ask yourself, what are you really afraid of? Name it. What's the risk here? List the risks. When are the risks worth taking? Quantify and analyze them. What can you learn from the failure? Spell it out.

Then, I suspect, you'll find that either you've come to realize that this isn't a good time to take the chance, or, more likely, you're just not quite so afraid anymore. Nervous, maybe, anxious, probably, but that's a long way from fear. Go get 'em.

Ted Neward
CODE

CODE

Jan/Feb 2018
Volume 19 Issue 1

Group Publisher
Markus Egger

Associate Publisher
Rick Strahl

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers In This Issue
Jason Bender
Miguel Castro
Sahil Malik
Ted Neward
Rick Strahl
Geoff Callaghan
Kevin S. Goff
Q Manning
Paul D. Sheriff

Technical Reviewers
Markus Egger
Rod Paddock

Art & Layout
King Laurin GmbH
info@raffeiner.bz.it

Production
Franz Wimmer
King Laurin GmbH
39057 St. Michael/Eppan, Italy

Printing
Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

Circulation & Distribution
General Circulation: EPS Software Corp.
International Bonded Couriers (IBC)
Newsstand: Ingram Periodicals, Inc.
Media Solutions

Subscriptions
Subscription Manager
Colleen Cade
ccade@codemag.com

US subscriptions are US $29.99 for one year. Subscriptions outside the US are US $44.99. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards accepted. Bill me option is available only for US subscriptions. Back issues are available. For subscription information, e-mail subscriptions@codemag.com.

Subscribe online at
www.codemag.com

CODE Developer Magazine
6605 Cypresswood Drive, Ste 300, Spring, Texas 77379
Phone: 832-717-4445
Fax: 832-717-4460

On Fear

For an industry that prides itself on its analytical ability and abstract mental processing, we often don't do a great job applying that mental skill to the most important element of the programmer's tool chest—that is, ourselves. "Go ahead, take that risk. Seriously, what are you afraid of?"

It's a line that gets used over and over again, particularly in reference to the pursuit of dreams and/or potential life partners. Despite the fact that fear is a deeply human emotion—and one that often keeps us alive—it's just a matter of figuring how to just "shut it off" on demand, usually in the pursuit of some grander utopian vision. "If only you can get past your fear," so goes the meme, or "screw your courage to the sticking-place," "be brave," or whatever other euphemism for "put fear in its place," you too can achieve greatness by getting past the only real obstacle standing between you and legendary status.

At least, that's how it works in the movies.

Why Fear?

Fear is a contextual thing. Some people find it exciting to bungee-jump off bridges or to skydive. I cannot for the life of me imagine why anybody would want to flirt with the possibility—no matter how remote—of ending up a messy smear on the ground. For some, excitement; for me, fear.

Let's be clear about a few things: fear is a necessary component for human survival. It's the trigger for a surge of adrenaline from the body when the brain perceives itself to be in a life-or-death situation, and that adrenaline is often the difference between life and death. When you're in the middle of a fight-or-flight situation, the adrenaline provides the necessary energy boost to the rest of the body so it can carry out the necessary steps to survival, be that "lifting the car off your loved one" or "scampering up that tree before the bear gets hold of you."

Wilderness survival scenarios notwithstanding, fear often manifests in less-comfortable situations: When you're getting ready to give the demo to the biggest client your company will have this year, you're going to feel fear. It may not be a life-or-death situation, but it's going to feel like it, just the same (particularly if your boss is nervous about the demo). Fear is defined as "an unpleasant emotion caused by the belief that someone or something is dangerous, likely to cause pain, or a threat." Failing to land this client could lead to your company concluding you are not worth continuing to employ. Worse yet, your failure could mean the loss of employment for the rest of your team. Or the company might go under. And so on. The fear is real, and it deserves to be recognized as such, regardless of the actual threat to life and/or limb.

It is tempting to suggest that fear somehow is the problem—that fear is what's keeping us from our success. If only we weren't afraid, so the thinking goes, we could take those risks and reap those rewards that come with great risks. (Of course, if the risks weren't great, the rewards wouldn't be either. But that's another story for another day.)

Fear itself isn't the problem. Which is good, because it's not like it's something you can get rid of. It's a bodily response, and you could no more get rid of fear responses that you could get rid of the flinch when somebody throws a ball at your eyes at close range, or the jump that happens when you touch a hot plate unexpectedly. Fear is going to be a part of us so long as we inhabit these human bodies, so the next question is, how do we get past it in those non-life-threatening situations? How can we live with fear, when fear is getting in the way?

Getting Past Fear

Despite the fact that the emotion has probably kept us alive any number of times, it's still important to know how to get past it. Unfortunately, despite the suggestions otherwise, it's not a question of simple willpower—you can't just "man up" and face it down. Fear is trying to keep your fool human self alive, and to ignore it is to basically try to ignore hunger or pain. It's not going to be easy, particularly for the more important things in life. That said, it's important to try, because the best things in life are never had by sitting on the couch wishing they would just happen to you.

If you want to get past fear, there's a couple of things that can help.

First, name your fear. Fear, like mold, mushrooms, and killer clowns, grows in the dark. Fear thrives in the quiet corners of the brain where you don't normally visit, and it whispers into the back of your mind when you're not paying attention. Bringing it out into the light, by naming it out loud and talking about it—even if only to yourself—takes it out of its natural habitat, and allows you to start bringing your mind around them. "I'm afraid that if I go to my boss and suggest that he should promote me to VP, he will fire me" is a fear, but now having named it, you can start to examine it. In the example here, let's be clear, if your relationship with your boss is such that you're thinking about asking for a promotion, it's probably not a situation where you're going to be fired for asking for it. It's no guarantee that you'll get it, mind you, but the reasons you don't get it may have nothing to do with you or your performance—the HR department may have instituted a hiring freeze, the boss may actually have other plans for you, or the boss just "can't afford to lose you" (in which case you're probably never going to get that promotion, so it may be time to start thinking about alternatives). Even should the worst-case scenario come to pass—your boss looks at you, laughs, and fires you on the spot—it's arguable that the fear may actually be doing you a favor. (Seriously, would you want to keep working for a manager who fires people when they ask for a promotion?)

Second, embrace the idea that fear comes from risk. No risk, no fear. No risk, no reward, either. If you wanted to live your life like the proverbial mushroom, you certainly can, but you're not going to grow much, either. In the woods, the tallest trees are the ones that get the most sunlight—but you have to fight the other trees to get there. No, you don't have to physically confront your co-workers about getting that promotion, but you do have to be willing to take the chance that a promotion will change the dynamic between you and your co-workers, and you have to decide whether the reward (a promotion, better salary, better benefits, whatever) is worth the risk that comes with it. This will force you, by the way, to sit down and actively think about what the risks are, rather than just leaving them nebulous and unexamined, which will sometimes lead you to realize that the risks are not all that painful.

Third, actively judge when to take the risk, rather than just "leaving it to chance." By beginning

(Continued on page 73)

Qualified, Professional Staffing When You Need It

CODE Staffing provides professional software developers to augment your development team, using your technological requirements and goals. Whether on-site or remote, we match our extensive network of developers to your specific requirements. With CODE Staffing, you not only get the resources you need, you also gain a direct pipeline to our entire team of CODE experts for questions and advice. Trust our proven vetting process, and let us put our CODE Developer Network to work, for you!

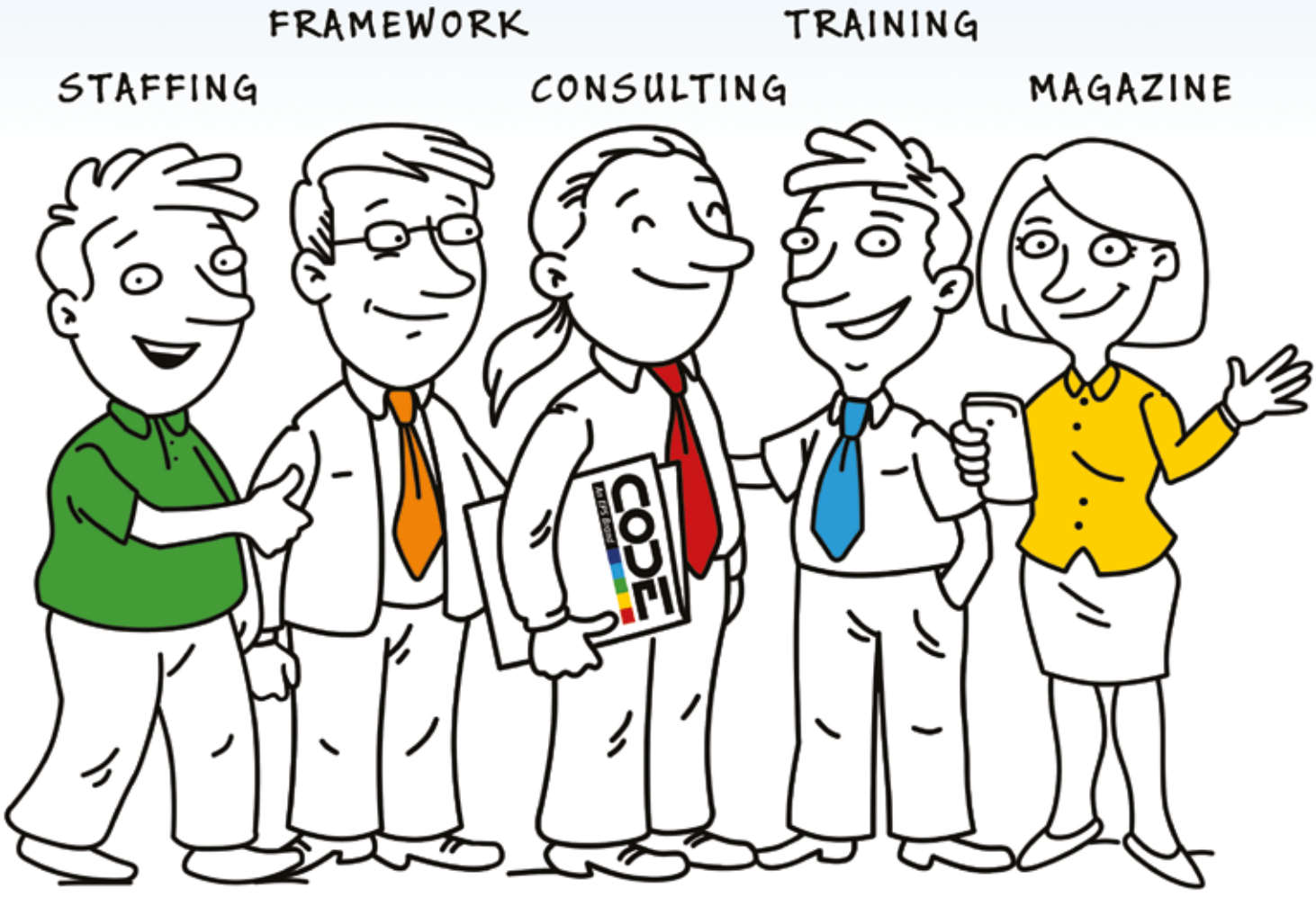
Contact CODE Staffing today for your free Needs Analysis.

Helping Companies Build Better Software Since 1993

www.codemag.com/staffing
832-717-4445 ext. 9 • info@codemag.com

CODE STAFFING

CODE - More Than Just CODE Magazine



The CODE brand is widely-recognized for our ability to use modern technologies to help companies build better software. CODE is comprised of five divisions - CODE Consulting, CODE Staffing, CODE Framework, CODE Training, and CODE Magazine. With expert developers, a repeatable process, and a solid infrastructure, we will exceed your expectations. But don't just take our word for it - ask around the community and check our references. We know you'll be impressed.

Contact us for your free 1-hour consultation.

Helping Companies Build Better Software Since 1993

www.codemag.com
832-717-4445 ext. 9 • info@codemag.com

CODE
An EPS Brand