```rust
use sha2::{Sha256, Digest};
use chrono::NaiveDate;
use rand::{Rng, SeedableRng};
use rand::rngs::StdRng;
use serde::Deserialize;
use std::fs;

// Investment struct
#[derive(Deserialize)]
struct Investment {
    investment_type: String,
    amount: f64,
}

// Loan struct
#[derive(Deserialize)]
struct Loan {
    loan_type: String,
    amount: f64,
}

// User data struct
#[derive(Deserialize)]
struct UserData {
    name: String,
    birth_date: String,          // "YYYY-MM-DD"
    salary: f64,
    credit_score: u32,
    monthly_expenses: Vec<f64>,
    investments: Vec<Investment>,
    loan_balances: Vec<Loan>,
    phone_number: String,
    addresses: Vec<String>,
    education_years: u8,
    marital_status: String,
    num_children: u8,
    vehicle_values: Vec<f64>,
    frequent_travel_expenses: Vec<f64>,
    subscription_costs: Vec<f64>
}

// Convert hash to a complex password
fn hash_to_password(hash: &str, length: usize) -> String {
    let mut password = String::new();
    let chars: Vec<char> = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()_+-=[

    // Use the first part of the hash as seed
    let seed: u64 = u64::from_str_radix(&hash[0..16], 16).unwrap_or(0);
    let mut rng = StdRng::seed_from_u64(seed);

    for _ in 0..length {
        let idx = rng.gen_range(0..chars.len());
        password.push(chars[idx]);
    }

    password
}

fn main() {
    // Read user data from JSON file
    let data = fs::read_to_string("user_data.json")
        .expect("Failed to read file");
```

```
let user: UserData = serde_json::from_str(&data)
    .expect("Failed to parse JSON");

// Convert birth date to NaiveDate
let birth_date = NaiveDate::parse_from_str(&user.birth_date, "%Y-%m-%d")
    .expect("Incorrect birth date format");

// Combine all user data into a single string
let mut data_string = format!("{}{}{}{}",
    user.name,
    birth_date.format("%Y%m%d"),
    user.salary,
    user.credit_score
);

// Add expenses and investments
for expense in &user.monthly_expenses {
    data_string.push_str(&expense.to_string());
}

for invest in &user.investments {
    data_string.push_str(&invest.investment_type);
    data_string.push_str(&invest.amount.to_string());
}

for loan in &user.loan_balances {
    data_string.push_str(&loan.loan_type);
    data_string.push_str(&loan.amount.to_string());
}

for address in &user.addresses {
    data_string.push_str(address);
}

data_string.push_str(&user.marital_status);
data_string.push_str(&user.education_years.to_string());
data_string.push_str(&user.num_children.to_string());
data_string.push_str(&user.phone_number);

for vehicle in &user.vehicle_values {
    data_string.push_str(&vehicle.to_string());
}

for travel in &user.frequent_travel_expenses {
    data_string.push_str(&travel.to_string());
}

for subscription in &user.subscription_costs {
    data_string.push_str(&subscription.to_string());
}

// Create SHA-256 hash
let mut hasher = Sha256::new();
hasher.update(data_string.as_bytes());
let result = hasher.finalize();
let hash_hex = format!("{:x}", result);

// Generate complex password
let password = hash_to_password(&hash_hex, 20); // 20-character password

println!("User-specific password: {}", password);

// Save password to file
fs::write("user_password.txt", &password).expect("Failed to write password to file");
```

```
}
```