Project Description

This document describes a simple banking website, this website consists of just typescript code and therefore does not have a GUI for the user to interact with. The website allows an individual to create a bank account (either checking or savings) provided they have sufficient funds to deposit into the account. After creating an account and successfully logging in, the user can withdraw/deposit funds into the account they have set up, and create an additional account (either checking/savings) if they do not already have that account type and view their available balance. Users will also be able to view their personal information and update their address, and/or login password. Authorization for these utilities is handled by two different authentication methods. The first is a standard form of authentication requiring the user to input their username and password, this will allow the user to log in to their account and view their personal information or update their password. The other authentication method will require a secondary password in addition to the username and password used to log in to their account. This acts as a form of two-factor authentication requesting more information that only the account owner should know in order to verify that they have the authorization to access the utilities this method provides. These include anything to do with finances (withdrawing/depositing funds, creation of a bank account), and changing the user's login password. The integrity of the program is maintained with a log that notes all significant activities that occur within the program: depositing/withdrawing funds, viewing available balance(s), creating a new account type, updating address/login password, and creation of a bank account. It is further maintained by using Sha256 to encrypt any highly sensitive user information including login password, secondary password, and the user's security question.

Purpose

The purpose of this project is to create a more secure banking website than what I have personally experienced. To my knowledge, a good portion of banking websites provide individuals full access to the user's account following the individual providing the account holder's login credentials. At the cost of convenience to the user, this program will require the user to reauthenticate themselves in order to use any of the utilities, with an increasing level of information required from the user depending on the perceived value that the utility provides (i.e., viewing/modifying personal information will require less information then all methods that deal with finances).

Packages that can be used

- Express - Easy to use web framework for javascript/typescript, equivalent packages include, (Django - Python, Rocket - Rust, gin - Go, etc.)
- email-validator - Easy-to-use module to verify emails input by the user for javascript/typescript, equivalent packages - (email-validator 2.0.0.post2 - python, validator - rust, Go standard library using mail.ParseAddress(), etc.)
- Sqlite3 - Database to hold all user/financial information, alternatives, (Oracle Database, Amazon Relational Database Service (RDS), MariaDB, Firebird, etc.)

Data Design

User/financial information should be stored in a database of some kind, since accessing personal information and financial information will require different authentication methods it would make the most sense to store these two kinds of information in different parts of the database. Personal information will include the user's full name, address, and email; log-in information will include the user's username, login password, secondary password, security question, and some kind of field to indicate whether the user is logged in or not. All information within the database should be connected to a specific user using some kind of randomly generated user ID, this can be stored in different parts of the database to reference which sections of information are connected to which user. Moving onto financial information it would be best to keep the information pertaining to checking/savings accounts in separate parts of the database as there will be different methods to move/view money within these accounts. Both checking/savings accounts will have the user's balance, bank account number, and user ID.

Website Utilities

The first utility that a user will need to use this website is a method to create a new account, for security reasons, the POST request used to create this account will only require an individual to enter the user name that they would like to use. The website should check if any other users have used the username entered before; if the username is unique the method will continue, if not the user will get a message informing them that something went wrong. Next, the user will be prompted to enter their password, again the website will check to see if any users have used the password entered before; if they have the user will once again be notified that something went wrong. Following this the user will be prompted for a security question and secondary password, at this point the username and password is enough to distinguish one user from another so the website will no longer check to see if the other users have used the same secondary password or security question. Both passwords and the security question will be encrypted immediately after entry; so that in the case that the website crashes unexpectedly this critical information will not be exposed. Next, the user will be prompted for their personal information (i.e. name, address, email), email will be validated to ensure that it is of valid form. The user will then be prompted to enter the type of bank account that they would like to set up (checking/savings), and the balance that they would like to deposit into their account. There will be a minimum of $200 to create a bank account. These will be the final prompts for the user, and all information that was entered if valid will be inserted into their respective portions of the database. In addition, a marker will be inserted into the login portion of the database, this will be used to indicate whether the user is actively logged in to their account or not.

Picking up where we left off with the mark to indicate login status, let's talk about the two endpoints that the user can use to log into or out of their account. Because the user will need to be logged in in order to use any of the utilities on the website. The **login** endpoint again for security reasons will only require the user to enter their username and further authentication will be handled on the website. The website will first check to see if the username exists in the

database, if it does not the user will be notified that something went wrong. Otherwise, the user will be asked to enter their password (login password) it will immediately become encrypted and the website will fetch the user's encrypted password from the database and compare the two to verify that they match. If they do not, the user will receive an error message. Also if the user is already logged into their account they will also receive an error message; otherwise, the user will be notified that they successfully logged into their account. Moving along the **logout** endpoint will also only require the user to send their username, the method will then check to see if the user is indeed logged in, if they are not the user will receive an error message, however, if they are the user will be notified that they have successfully logged out. Both of these methods will be PUT requests as they update the login status marker.

Now let's cover the two authentication methods that will be used to gain access to all utilities on the website. The first of these will function much like a **standard authentication** process on many other websites. Since most endpoints will require the user to send their username with the HTTP request, this method will prompt the user to enter their password. Immediately after entry, the password will be encrypted and compared to the password that is associated with the user that sent the HTTP request. If the passwords match the method will return true to indicate that authentication of the user was successful; otherwise, the user will receive an error message. For the remainder of this document, we'll refer to this authentication method as **authentication method 1**, it will be used for utilities that do not pose as much of a security risk compared to others in the scope of this website. Moving on we'll discuss what we'll call **authentication method 2**, this method will function much like authentication method 1. However, in addition, it will require the user to enter their secondary password, like the login password this password will be encrypted immediately after entry and compared to the secondary password for this user. Like authentication method 1 this method will return true if both passwords match indicating that authentication was successful; otherwise, the user will receive an error message. The method will be used for all utilities on the website that deal with the movement of money, or anything that is deemed critical. Both of these methods will be called internally by the website with no way for the users of the website to invoke outside of the HTTP requests.

Next, let's cover endpoints of the website that will allow the user to modify/view their personal information; as mentioned earlier in this section all utilities on the website will require the user to be logged in. To **view personal information** the user can send a GET request with their username, the method will verify that the user is logged in, if not they will receive an error message. Otherwise, authentication method 1 will be used to re-authenticate the user, if the authentication is successful the user will receive a message displaying their name, address, and email. At this point in the design process, there are two methods used to update/change the user's information, the first can be used to update the user address. To do so the user will send a PUT request with their username and their new address, for the sake of brevity all further methods will verify that the user is logged in. The method used to **update the address** will then use authentication method 1 to re-authenticate the user, if the attempt is successful the user will receive a message informing them that their address has been updated, otherwise they will receive an error message. To **update their login password** the user can send a PUT request with their username, the method will then use authentication method 2 and then prompt the user to enter their security question to authenticate the user. Following this, the website will prompt

the user for their new password, and inform the user that the update was successful. Like the previously mentioned functions, the new password should be encrypted immediately after the user inputs it.

Finally, we'll cover the functions that are used in order to access the user's checking/savings account. Since the functions used in each account type are almost identical they will only be covered once. Starting this section off, the user can submit a POST request to **create an account** of the type that they do not already have, either checking/savings. To do this they will send a POST request with their username, the function will then use authentication method 2, and check to see if the user already has this account type if they do the user will receive an error message. Otherwise, the user will be asked how much they would like to deposit with a minimum balance of $200 being required to create the new account. If sufficient funds are deposited the user function will generate an account number and inform the user that the new account type has successfully been created. Next, the **function to deposit funds** into their select bank account will use a POST request using the individual's username and the balance that they would like to deposit, it will also use authentication method 2 to verify access to the account. If authentication is successful the balance sent with the post request will be added to the user's current available balance and the user will be informed that the deposit was successful. The **function to deposit funds** will function much in the same way, the user will send a POST request with their username and the balance that they would like to withdraw. The function will then use authentication method 2 and if successful, check to see if the user's available balance is greater than the requested withdrawal amount, if the requested amount is greater the user will receive an error message. Otherwise, the amount requested will be subtracted from the user's available balance and the user will be notified that the withdrawal was successful. The last function in this section can be used by the user to check their available balance, this will be done by sending a GET request with their username, the function will then use authentication method 2 and if successful display the user's balance.

Log

The log will be written to a file in order to track any significant activity performed on the website. This includes the creation of the user, updating the address/password, and withdrawing/depositing funds into either the user's checking/savings account. Creation of a new bank account type, and checking the balance of either the checking/savings account.