Low-level Design Document

Break down component design information from HLD:

Front End: Mobile and Desktop Clients will be developed for Web. Mobile Clients will be developed as responsive web applications and then converted into native applications using web app frameworks like React Native. Main desktop client will be made with HTML, CSS, and JS or the combination JSX from React. The front end will be made using React. To make the UI more appealing we may possibly use Bootstrap and Material UI CSS frameworks which have their own components that interface well with React.

Back End: The back end will be hosted on Heroku and will be written in NodeJS. The backend will handle API calls for communicating with the PostgreSQL database that will store customer information and other app related information. The backend will also handle communication with the Google Maps API and pass that information to the Clients whenever they need to display Chase Bank ATM locations.

Database: We have decided to use PostgreSQL as the database for the backend. This database will be hosted using Heroku web hosting service. For mapping functionality, we will be using Google's Google Map API for websites. If time permits, we will using Android Studio and or Xcode to create a mobile app to access the program for clients with Android and Apple phones.

2. Logic can be represented in pseudo code

```
(called on login)
getAccountData(account number, account password, authorization code)
{
        if(authorization code == valid)
        {
                Retrieve account password
                if(account password == saved password)
                {
                        Retrieve data
                        Return data
                }
        }
```

```
}

createAccount(username, password, startingBalance, personal info)
{
	Verify username and password data
	Create database entry
	Populate with provided data
	Return successful account creation
}

removeAccount(accountNumber)
{
	if(!(account balance > 0))
		Delete account entry from database
	Else
		Throw error
	Return successful account deletion
}

(called on deposit/withdrawal)
updateBalance(account number, amount, authorization code, update_type)
{
	if(authorization code == valid)
	{
		Lock access to this account
		Retrieve current account balance
		if(update_type == withdrawal && amount > account balance)
		{
			Unlock access to account
			Throw error
		}
		add/subtract amount depending on update type
		Save new balance in database
		Unlock access to this account
	}

	Return successful update balance
}

generateReport(authorization code, criteria[])
```

```
{
        if(authorization code == valid)
        {
                For all criteria
                {
                        Get data that matches criteria
                }
                Return data
        }
}
```

High-level Test Plan

The strategy for our testing will be first to test the functionality of the website's various features. This includes from scratch a user entering the website, creating an account from scratch and performing transactions. Functional testing also includes the bank manager's interface where the manager will be able to generate reports of customers with specified attributes. After successful basic functionality testing, we will test for usability testing. For example if someone unfamiliar with our website design is able to successfully navigate on the site and complete basic functionality testing with basic instructions with our help. If there is something difficult to find on the website we will modify our design for better visibility and navigation. After this we will test the back end components to see if our various softwares are interfacing with each other correctly. This is basically seeing if our front end code is interfacing correctly with the back end code. If the web server is connecting the database to the website application interface incorrectly. Testing includes any logic errors or exceptions that may happen in regular use of the website. Next part of the testing includes compatibility testing where we rectify compatibility differences platforms. Since our target browsers are Chrome and Firefox depending on the framework being used we will have to different sets of code with the same functionality depending on the browser. If there is time we will make a mobile application for the website on android devices which also depending on the framework may require additional code to function. There's also performance testing where we will optimize web page loading, data retrieval, and transactions to maintain acceptable speeds. While not required for our project, we want to add a few security measures to our website to prevent malicious action i.e. limit the number of password guesses / entries.

Functional / Component Tests:

1. **Account Creation**
   Goal: Successfully create an account using website interface.
   Expectation: 1 account per email
2. **Account Login**
   Goal: Login into account
   Expectations
   - User uses password to access account.
   - If account does not exist website informs the user
   - If password is incorrect, inform the user
   - Account persists until user decides to delete account
3. **Transactions**
   Goal: transactions are successfully performed through website or ATM
   Expectations:
   - Transactions are atomic
   - Incorrect Transactions are given error messages i.e withdrawing more than the account balance
4. **Manager Reports**
   Goal: Manager can create reports with specified attributes through special manager only interface.
   Expectation:
   - Reports are generated in a reasonable time
   - Reports are generated according to the specified attributes selected
   - Report format is readable
5. **Find ATM**
   Goal: Display a map of nearby Chase ATMs through Google's Map APi
   Expectation:
   - Map is displayed within reasonable time
   - Map shows ATMs nearest to customers specified location
6. **Cheque deposit**
   Goal: Users are able to deposit checks by image through the website
   Expectation:
   - Provide smooth interface for customers to upload image of their checks
   - Actual image recognition software shouldn't be needed so pretend to deposit amount specified to customer account
7. **Deposit/Withdrawal via API**
   Goal: API for manager to credit or withdraw money from accounts
   Expectations:

- Able to remotely credit account without additional work
- Consistent

8. **Account Management**

Goal : Users have a dashboard when logging in to choose what actions they want to perform.

Expectations:

- Links that lead to the action the user wants to perform
- Separate pages for manager and regular user

Expanding on what was mentioned above overall priority for testing criteria are functional components. Besides blind user testing we'll be using automated web based testing using Selenium to see if the website application to speed up testing multiple different types of user input. Since our backend is to be written in Node.js in addition to writing our own test cases we will be using test frameworks such as Mocha to ensure that there aren't any special exceptions that may arise from testing. After the rough part of the project is completed and Blackbox testing is done we will update the website if there is any usability problems that arise.