
MIAMI

CSC 431

<Earmark>

System Architecture Specification (SAS)

Team Number: 1 (8)

Justine Mathurin	<COO/Leader>
Sofia Portillo	<CEO/Supervisor>
Shamir Cetoute	<CXO/ Head of Presentations / Developer>
Nyanti Eason	<CFO/ VP of HR/ Head of Support>
Kyle Mendelson	<CIO / CBO / Developer>
Beau Bridges	<CTO/Senior Developer>
Rose Gupta	<Vice-CFO>
Hoang Pham	<Front-end Developer / Researcher>

Version History

Version	Date	Author(s)	Change Comments
1.0	3/25/20	Backburners	First Draft
1.1	4/22/20	Backburners	Second Draft
2.0	5/2/20	Backburners	Final Draft: fixed comments

Table of Contents

Table of Tables

<Generate table here>

Table of Figures

<Generate table here>

41.System Analysis

41.1 System Overview

The System will comprise two main parts: a data manager and a REST controller. The data manager will control the connection to the database to handle database queries. The System will be handled on a NodeJS server. Utilizing npm packages to implement the data manager and database along with it. The REST controller will manage all connections to and from the app, processing end-to-end requests.

Per the database the System will use express as a server and mysql as a server. Express serves routes for endpoint requests from the apps to process and return data all in JSON format. Other npm packages will be used in assistance to effectively implement the System.

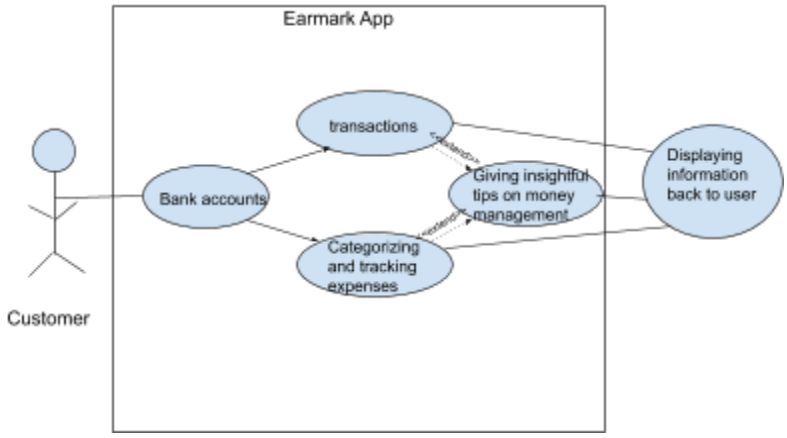
Data is sent from the app to the System to be processed and or stored with a response back to the client app.

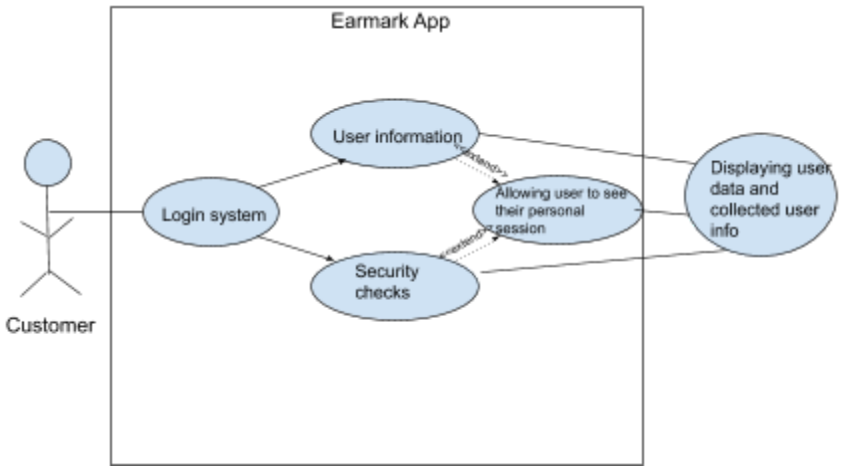
41.2 Actor Identification

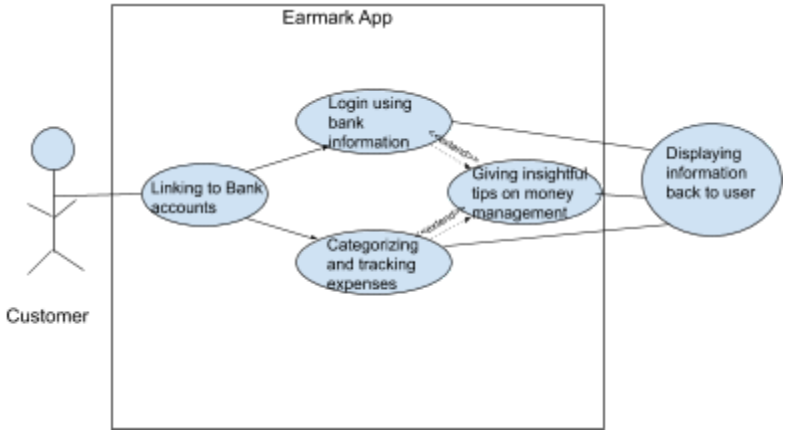
There are two types of human actors: administrators and users. Admins would be able to see everyone's login information ie. username/emails, given the permission of the user (through a pin/series of security questions). They will not be able to see anything else as the app will not be storing information. Users will be able to see their own information, such as banking info, transactions etc., after logging in. Only the user will have access to changing login information for their account. Users will be unable to see other users' information.

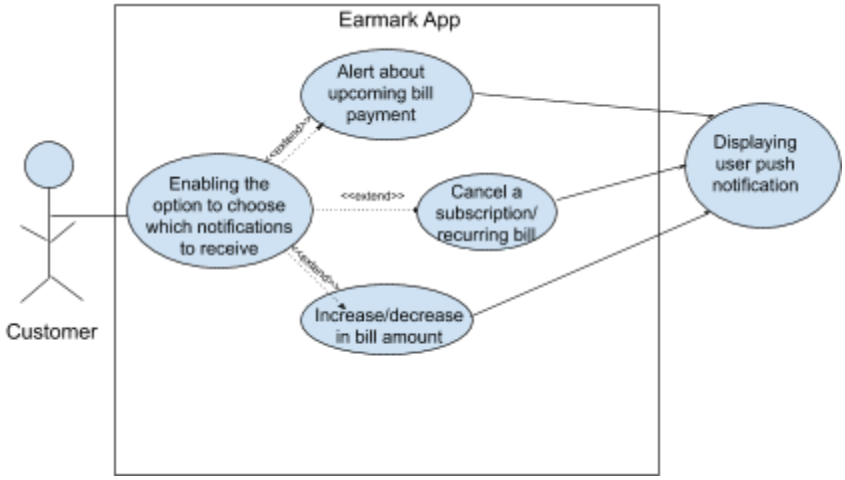
41.3 System Diagram

USE CASE DIAGRAM:




Earmark App
User information
Login system
Allowing user to see their personal session
Displaying user data and collected user info
Security checks
Customer


Earmark App
Login using bank information
Linking to Bank accounts
Giving insightful tips on money management
Displaying information back to user
Categorizing and tracking expenses
Customer


Earmark App
Alert about upcoming bill payment
Enabling the option to choose which notifications to receive
<<extend>>
Cancel a subscription/ recurring bill
Displaying user push notification
Increase/decrease in bill amount
Customer

41.4 Design Rationale

1.4.1 Architectural Style

//Client-server interaction
//The main is the backend while front-end displays backend data
//REST-API receives input from user and follows a series of functions to validate data and add data to our database
Do we use this? - Model html/C -View- css/ -Controller - app (MVC)
//Model is in JS manipulated to mysql controller is the backend of the APP that manipulates data into the model and View is the data displayed

This system exists with both the front-end and the back-end incorporated into each other to create a beautiful display using HTML/CSS and the back-end includes the REST API with a database maintained. In designing this, we are using 2 styles, the Model-View-Controller (MVC) and the client-server interaction. Our model is the JS manipulated to mysql which includes HTML with the view being the CSS and JS. The controller would be the app which is the UI for the app to be seen and also be the way to show and hide the data. For the client-server style, the REST controller receives user input, validates the input, and passes that on to be processed by the database and finally responding to the client's request.

1.4.2 Design Pattern(s)

Express is a flexible Node.js framework that provides a robust set of features. Along with react native a framework branching from reactjs to be mobile friendly. We predict that the following design patterns will be primarily used in the following ways:

- Singleton: To maintain a single, consistent reference, such as a database reference
- Middleware: To handle flows that have middleware functions in the system's request-response cycle,such as in the data conversion process. We are using react-redux to handle local storage in the app, handling updates received from the server.
- Stream: To process large amounts of flowing data, such as in the file writing/core download process

1.4.3 Framework

JavaScript accompanied with React Native will be used to ensure the UI can be compatible on both iOS and Android devices. This will make for a user friendly environment. Node.js will be used in coding the apps components, along with SQL to have a client-to-server interaction for data storage. We will be using Adobe XD for design and Adobe Illustrator for the logo and

icons. Plaid's feature to securely access the information by letting the user log into the bank account without us ever seeing their bank info.

Links:

- *https://nodejs.org*
- *https://reactnative.dev/*
- *https://plaid.com/*
- *https://www.adobe.com/*

42.Crosscutting Requirements

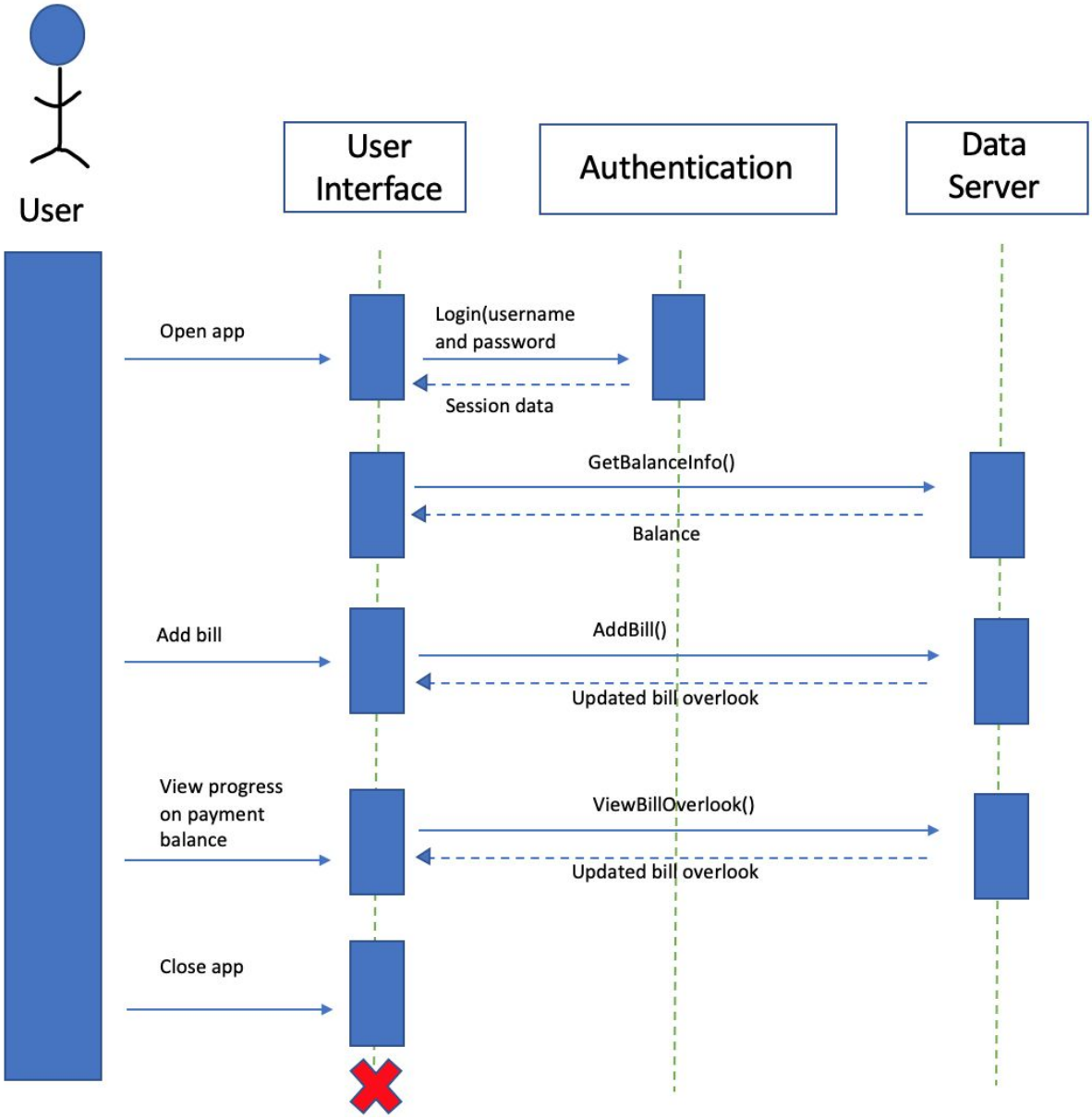
We have identified 2 cross-cutting requirements that are involved in the putting bills for remind process:

1. Adding a bill:
   - Through the use of our pop up screen, users will be prompted with the bill's name, amount, due date and if it is paid or not. If it isn't paid then the data will be shown on top of the front screen.
2. Save data to database:
   - Also, we need to save the data that was inputted on our database to show the bill on the front of the activity page to be reminded or to save it in the history. That means by that due date, the bills have to be paid.

Also, there are 3 cross-cutting requirements when it comes to setting up a budget:

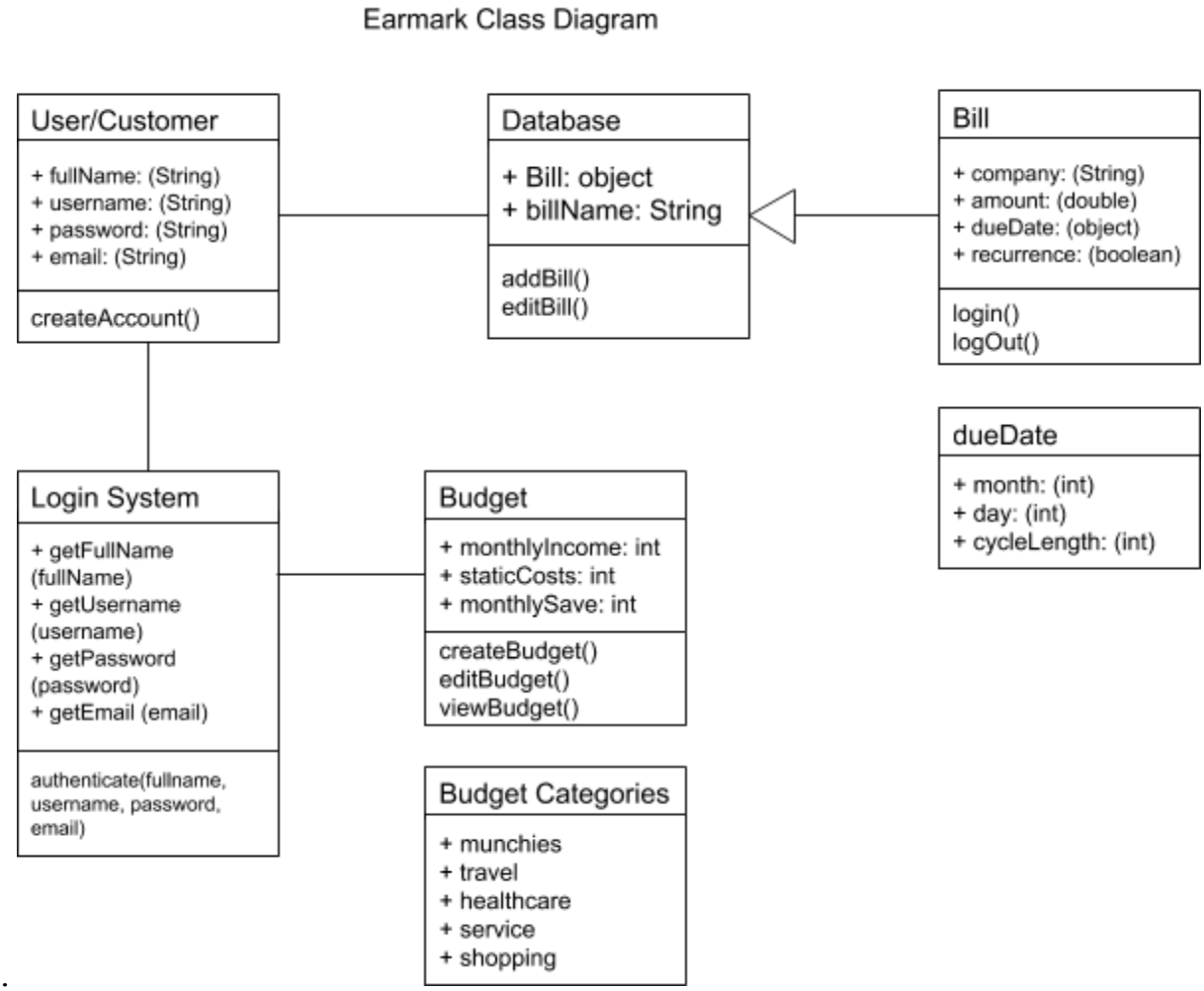
1. Linking to your bank account:
   - Linking to a bank account requires a link to Plaid. Plaid will prompt the users for their banking info. Our app will save the info and show in the front page how much they have.
2. Manually inputting a budget:
   - Our app will show multiple sliders to set budgets for different categories and it will be saved to our database.
3. Showing current budget:
   - After knowing how much money is in the bank account and what is the user's budget, a pie chart will be shown on the current budget screen and user can always edit the budget

42.1 Functional Design



43.Structural Design(in progress)

The following class diagram represents an overview of the component breakdown in our system. A primary focus for this design is modularity.



.