Investo
Group 2
Rick Dymond, Logan Howard, Benjamin Hackett, Brian Evans
CSI 4999 - Senior Capstone
4/9/23

Table Of Contents

Project Description

Financial and investing Android mobile application. Our goal as a team is to help users save money month to month by providing a quantitative budget report. In addition, the user will be able to monitor how money is moving in/out of their budget based on spending habits. The user will be able to see their recurring bills over the course of the month, helping them to pay their bills on time. The intended use of this application is for users to specify budgets derived from various sources, and how bills/daily transactions affect the money in their budget.

Target Audience

Anyone 15 years or older seeking financial tracking.

Similar Products

Intuit Mint - Automatically sync bank accounts to app
Nerd Wallet - Manage banking, investing, and loans/insurance from a single app
EveryDollar - Personal wealth and spending tracker
Honeydue - budgeting app designed for couples
Monefy - Journal style data entry expense management

Source of Revenue

The main source of revenue for this application is Google Ads. Each time the user logs into the application, small banner ads will display. Ads will advertise various products and brands. Our revenue will be directly proportional to the number of views the banner ads receive.

User Types

Stakeholders

Stakeholder Name	Position	Internal/External	Role
Google Ads	Sponsor	External	Provide AD revenue
Rick Dymond	System Engineer	Internal	Address problem(s) with the system and find solution(s)
Logan Howard	UX Development Engineer	Internal	Provide continual user experience research and development
Benjamin Hackett	UI Development Engineer	Internal	Provide continual user experience interface and development
Brian Evans	Database Engineer	Internal	Database design and development

Figure 1: Stakeholder table showing the roles and positions of the team

Software Architecture

Use Case Diagrams and Tables

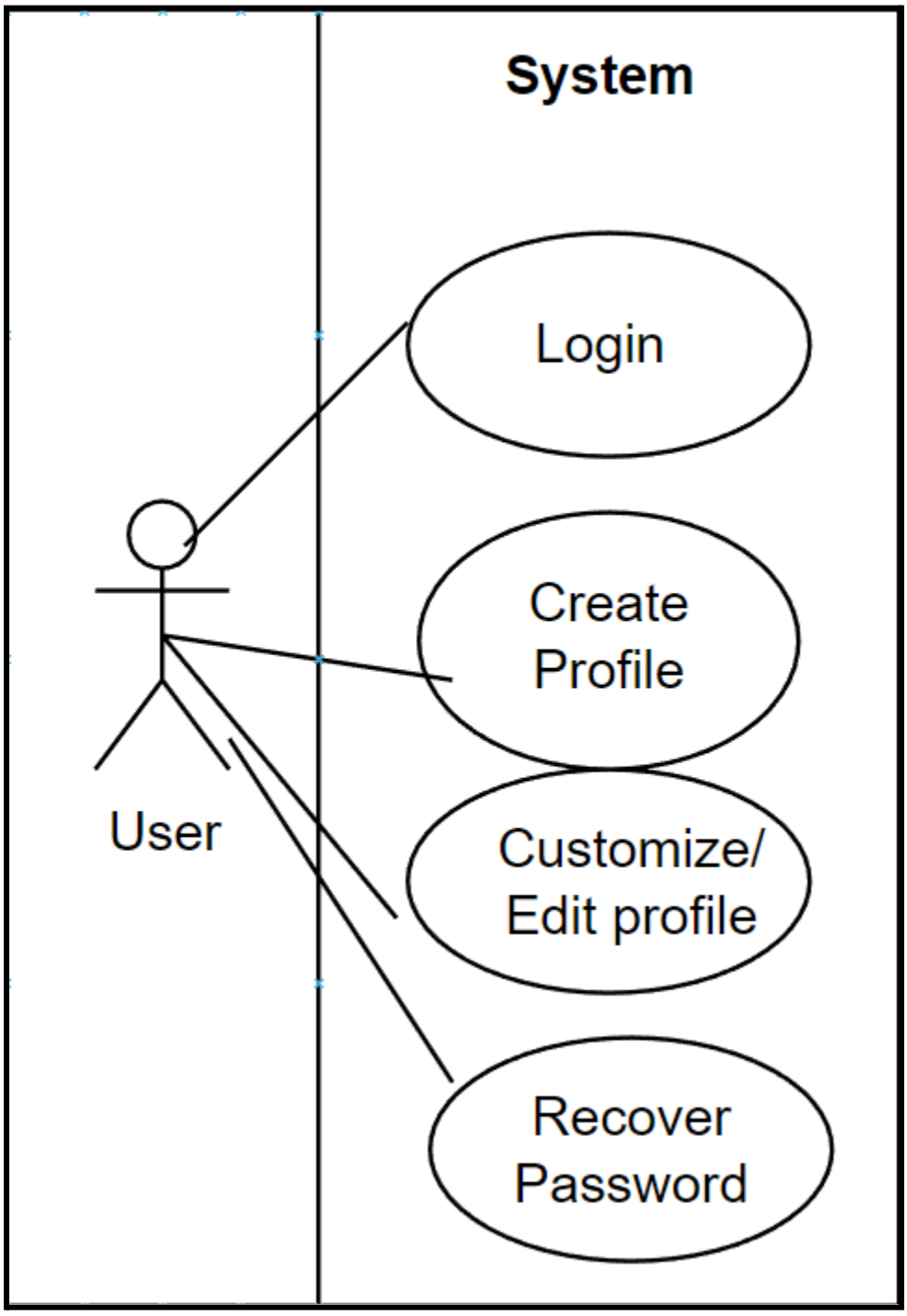


Figure 2: Use case diagram for the login subsystem

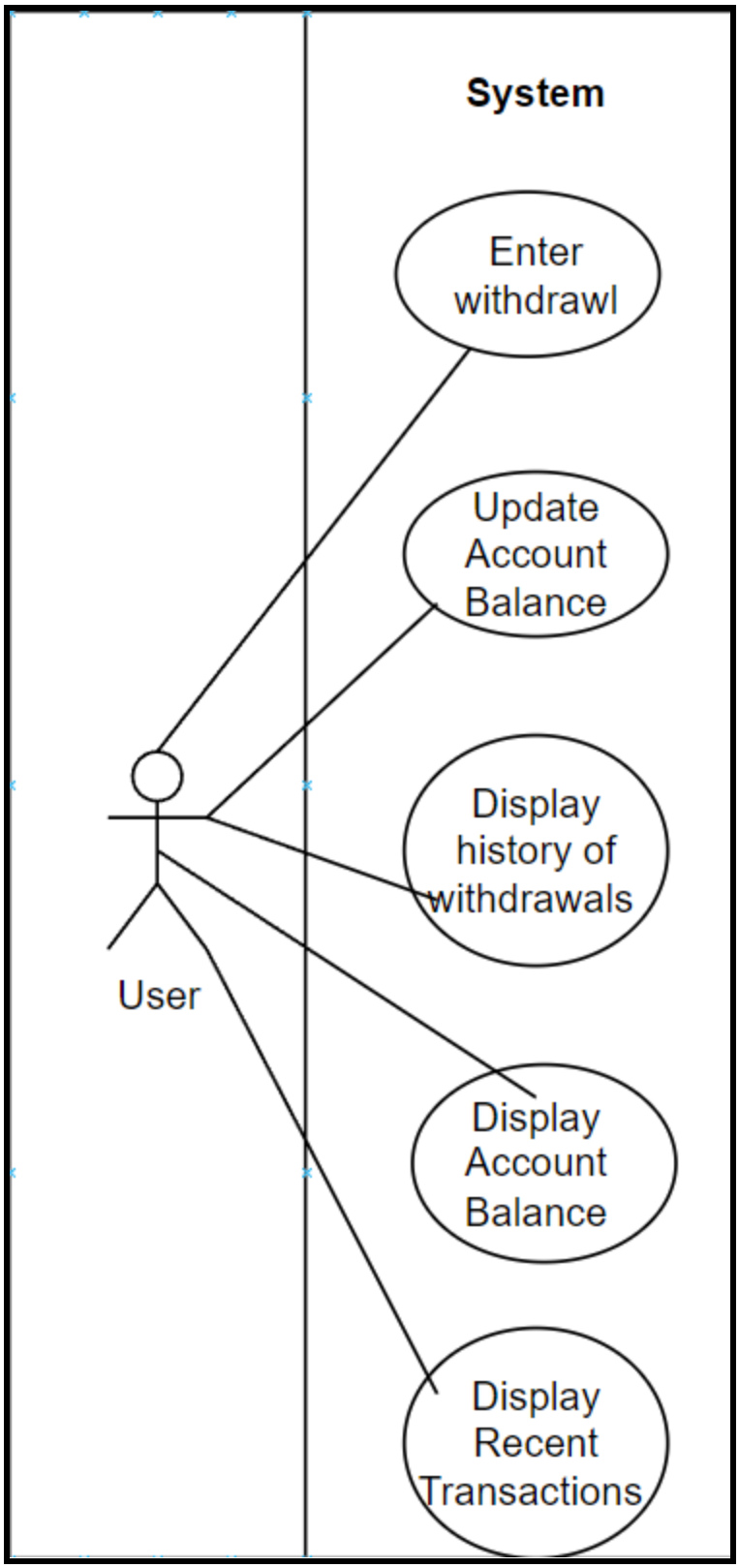


Figure 3: Use case diagram for the overview activity

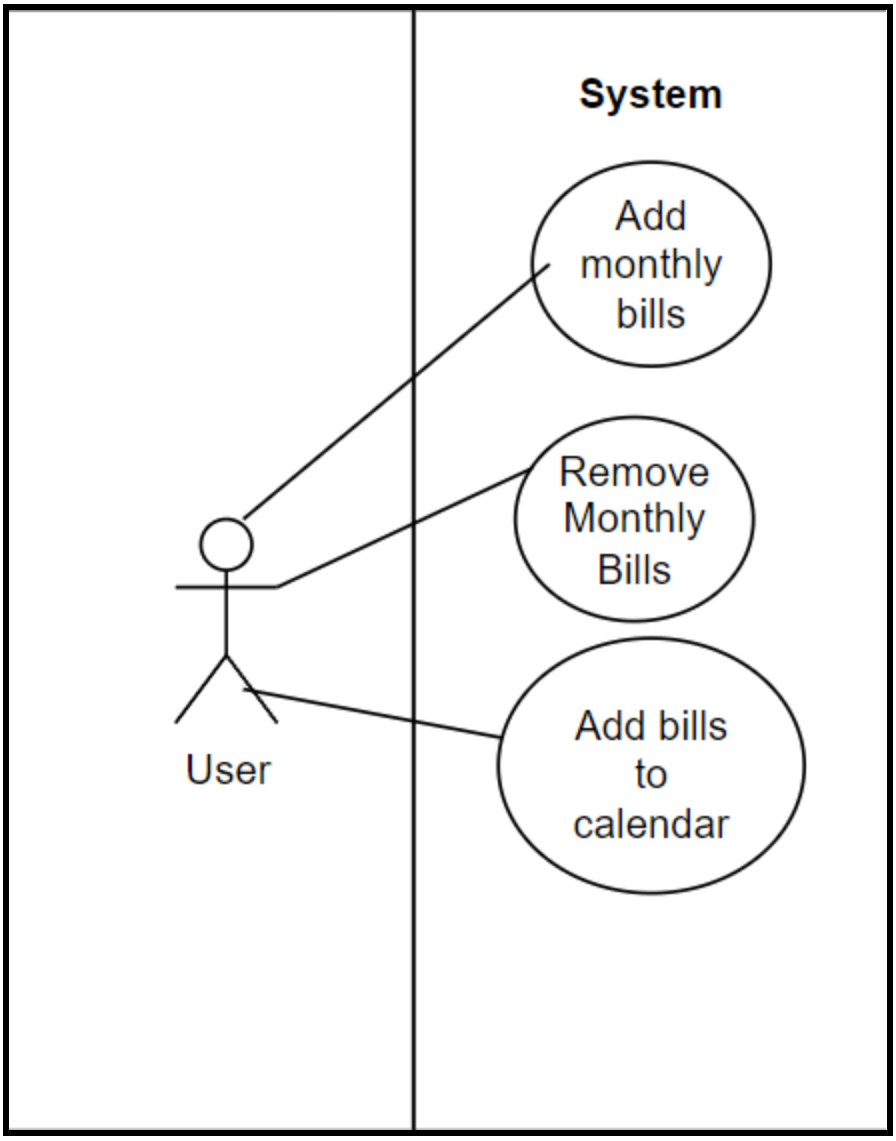


Figure 4: Use case diagram for the bills activity

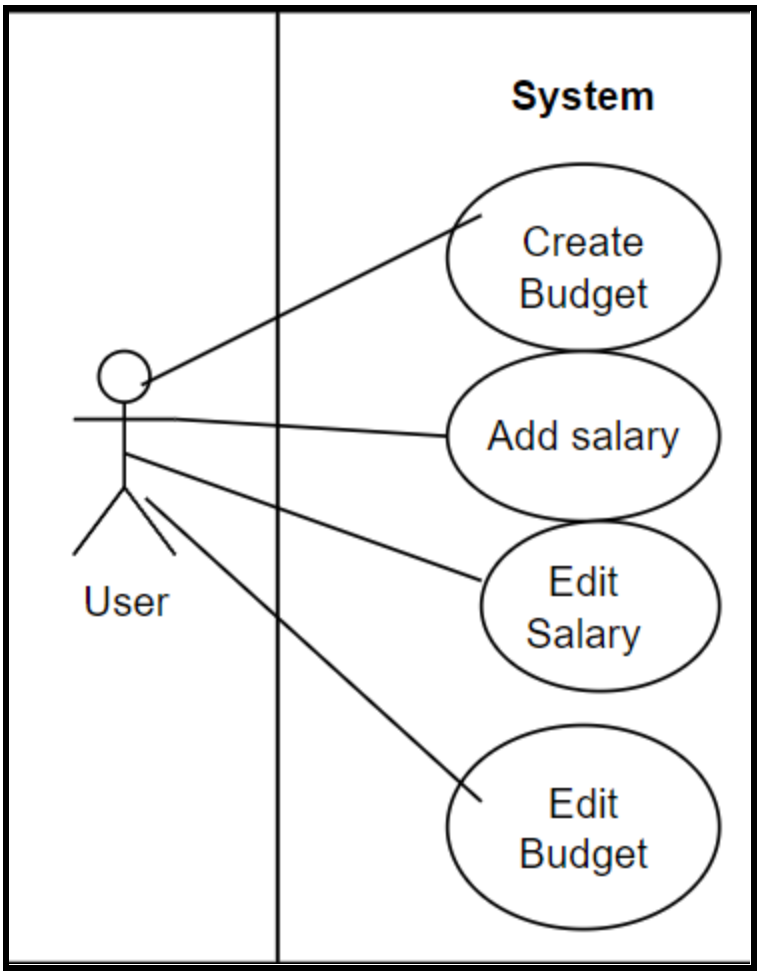


Figure 5: Use case diagram for the budget activity

User

Use Case	Description
Login	Input username and password
Create Profile	Pick Username, enter info (email etc)
Display Account balance	Total funds in account
Enter withdrawal	Submit withdrawal amounts
Display history of withdrawls	Show recent withdrawals from account
Update user's balance	Update account balance after withdrawal/ deposit bill

Add salary	Input user yearly salary
Edit Salary	Increase or decrease salary
Display Recent transactions	Show recent purchases from account
Add monthly bills	Add recurring monthly bills
Remove Bills	Remove no longer exist recurring subscriptions or bills
Add bills to calendar	Submit recurring bills to calendar view
Create budget	Set monthly budget goal
Edit Budget	Increase or decrease budget for month
Recover password	Forgot password function to get password back
Customize profile	Set profile picture and add info

Figure 6: Use case table for the standard user

Admin

Use Case	Description
Manually correct database	Fix errors in DB
Delete Users	Remove users

Figure 7: Use case table for the admin user

FURPS

Functional Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
FR001	The system shall allow the user to login	Must	
FR002	The system shall allow the user to create a profile	Must	
FR003	The system shall display the user's account balance	Must	
FR004	The system shall allow entry of withdrawal	Must	
FR005	The system shall display the history of withdrawls	Must	
FR006	The system shall update the user's balance	Must	
FR007	The system shall allow a user to add a salary	Must	
FR008	The system shall allow the user to edit a salary	Must	
FR009	The system shall display recent transactions	Must	
FR010	The system shall allow users to add monthly bills	Must	
FR011	The system shall allow a user to	Must	

	remove monthly bills		
FR012	The system shall allow a user to add bills to the calendar	Want	
FR013	The system shall allow a user to create a budget	Must	
FR014	The system shall allow a user to edit their budget	Must	
FR015	The system shall allow a user to recover password	Want	
FR016	The system shall allow the user to customize their profile picture	Want	

Figure 8: Functional requirements table

Usability Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
UR001	The system shall use the following color codes: #42f584, #545454, #FFFFFF as the main colors	Must	Keep these colors as the main colors for the application
UR002	The system shall use the following color codes: #FFA726, #66BB6A, #EF5350for the pie chart	Must	
UR003	The system must have scroll views to display more content per activity	Want	

Figure 9: Usability Requirements table

Reliability Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
RR001	The system shall have 100% reliability	Want	Offline local application
RR002	The system shall display a toast to verify login correctly	Want	

Figure 10: Reliability Requirements table

Performance Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
PR001	The system shall be able to run consistently across all Android devices	Want	
PR002	The system shall use low system resources	Want	

Figure 11: Performance Requirements Table

Security Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
SR001	The system shall use the users username and a password to secure the account	Must	
SR002	The system shall keep the database hidden to users	Must	

Figure 12: Security Requirements table

Design Constraints

Requirement ID	Requirement Statement	Must/Want	Comments
DR001	The system shall be Android exclusive	Must	(No Apple)
DR002	The system shall not exceed 1GB of memory usage	Must	

Figure 13: Design Constraints Table

Implementation Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
IMR001	The system shall use SQLite database	Must	
IMR002	The system shall be developed in Android Studio using Java with a min SDK 26	Must	

Figure 14: Implementation Requirements Table

Interface Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
INR001	The system shall use device photos for profile picture	Must	
INR002	The system shall be responsive for tablets	Must	

Figure 15: Interface requirements table

Physical Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
PHR001	The system shall use an Android device	Must	Has to be run on an Android device

PHR002	The system must be in mobile or tablet format	Must	

Figure 16: Physical Requirements table

Supportability Requirements

Requirement ID	Requirement Statement	Must/Want	Comments
SUPR001	The system shall support users with an Android device	Must	Has be able to support multiple SDK versions
SUPR002	The system shall be installed from the Google Play Store	Must	

Figure 17: Supportability Requirements table

Technology

This project was developed in Android Studio, written using Java and XML, and has a minimum SDK 26 for device supportability. The database uses SQLite that takes advantage of BaseColumns for static table/column identification.

System Sequence Diagrams (SSD)

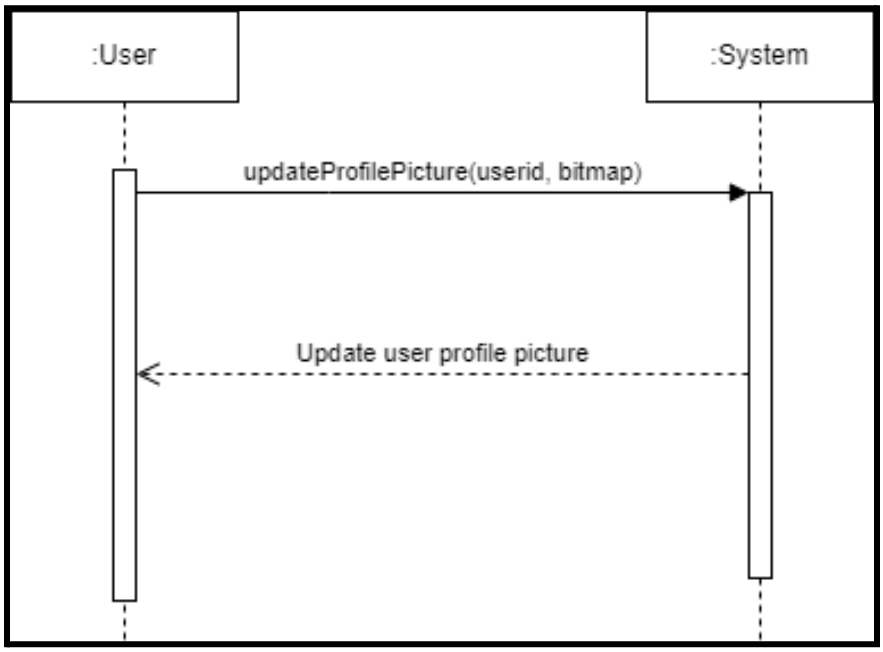


Figure 18: Update profile picture SSD

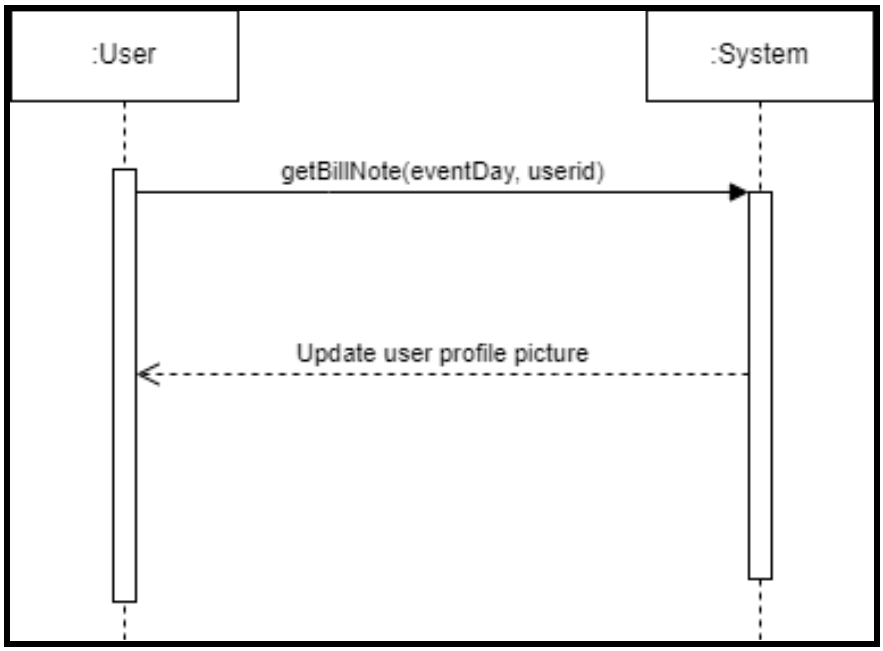


Figure 19: Get bill note SSD

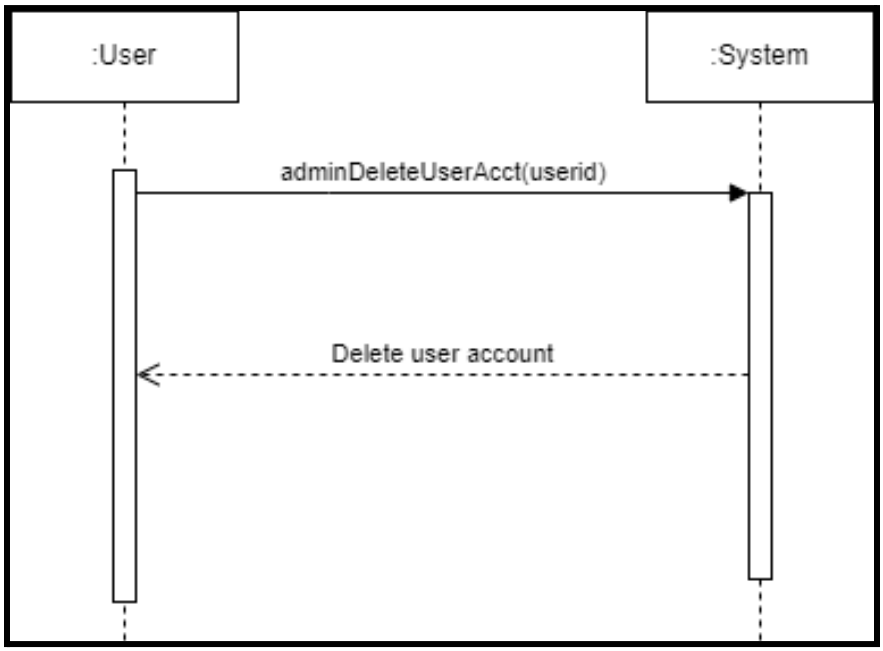


Figure 20: Admin delete SSD

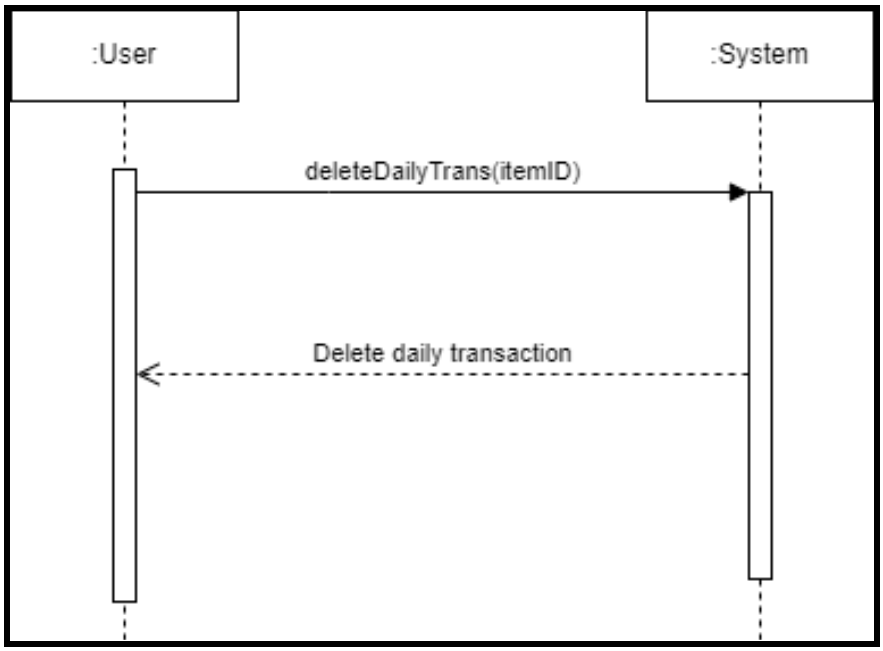


Figure 21: Delete user SSD

Activity Diagram

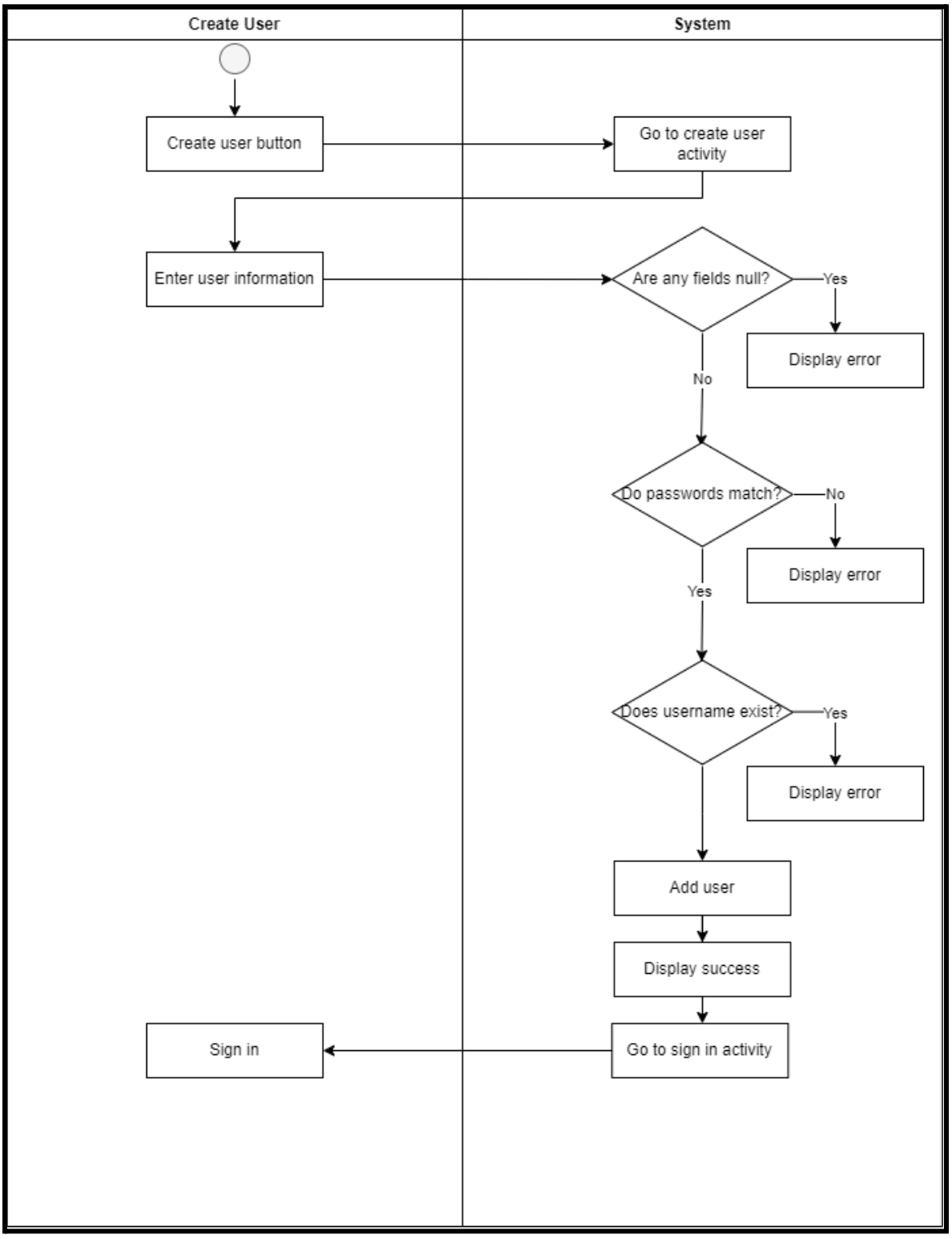
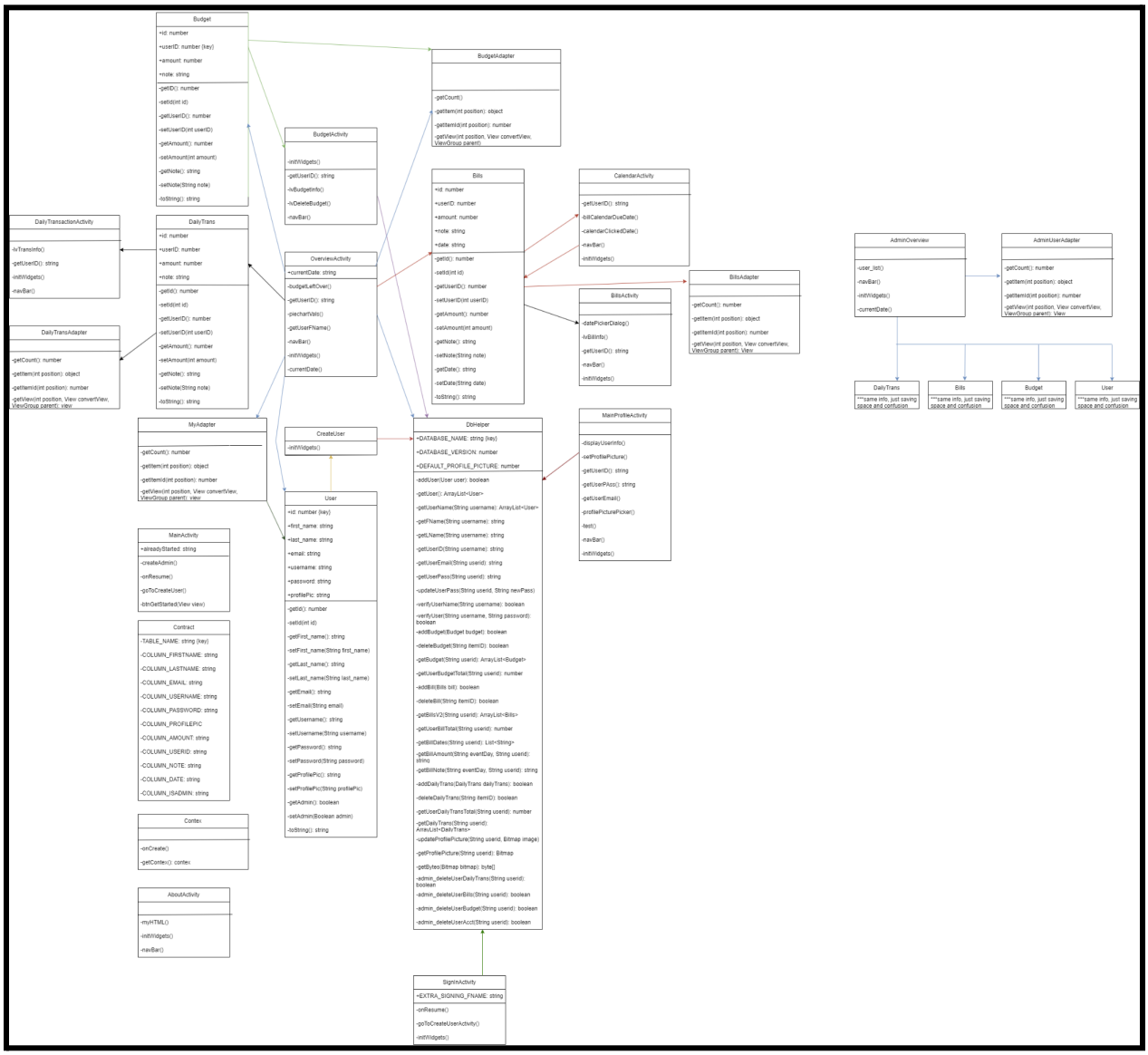


Figure 22: Activity diagram for user creation in Investo

Design Class Diagrams



*Note: Please zoom in for the best viewing of our Design Class Diagram. If it is still not clear enough, this URL should allow for better control and viewing: click here.

Also, the admin table was added at a later date than the rest. Additionally, to avoid confusion, it was separated from the rest of the classes. There would be too many lines crossing everywhere, so just assume that the tables that the Admin Class connects with have their full length, variables, and functions still!

Figure 23: Design Class Diagram

User Interface Design

Sketches

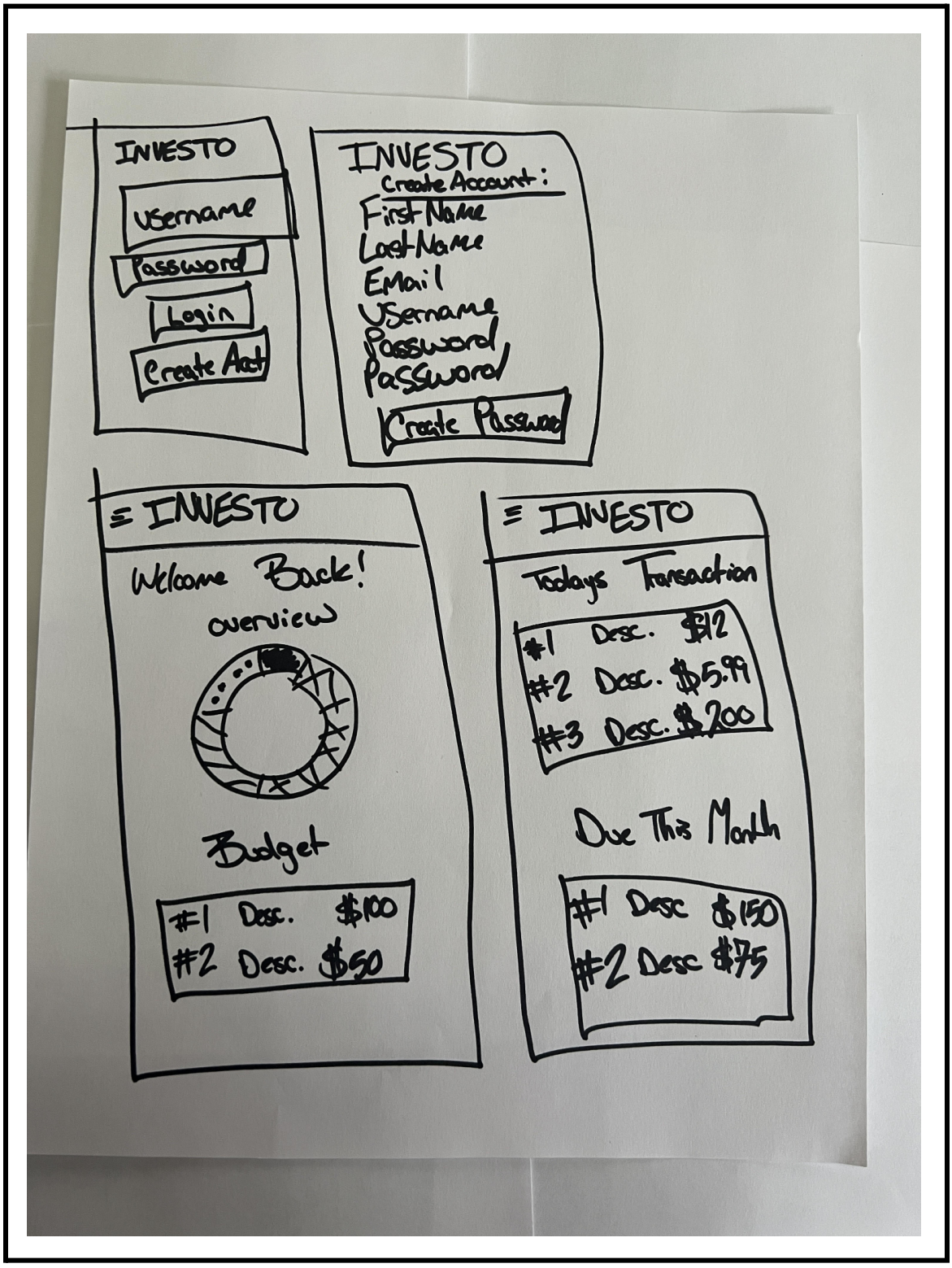


Figure 24: Sketch wireframe

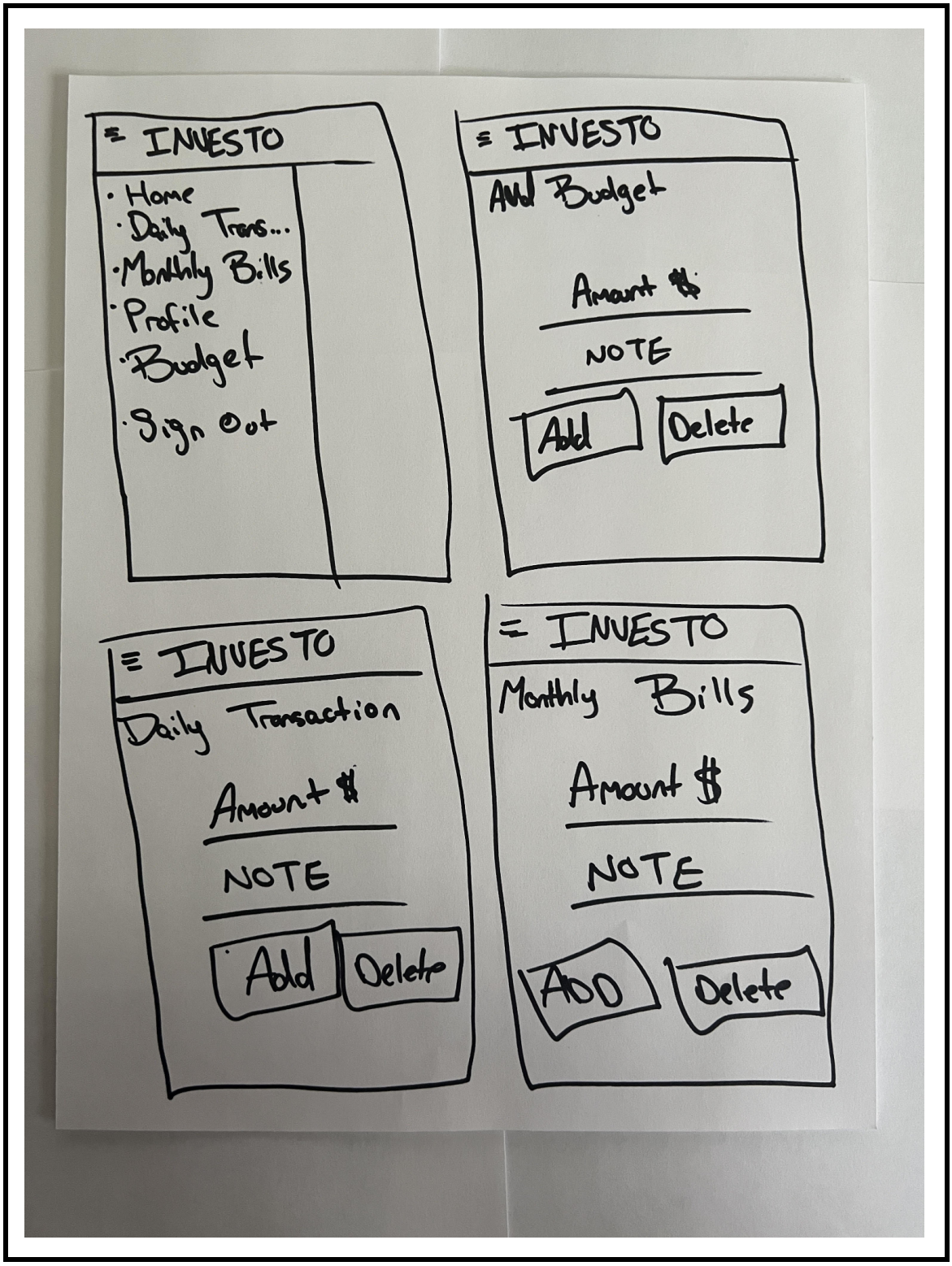


Figure 25: Sketch wireframe

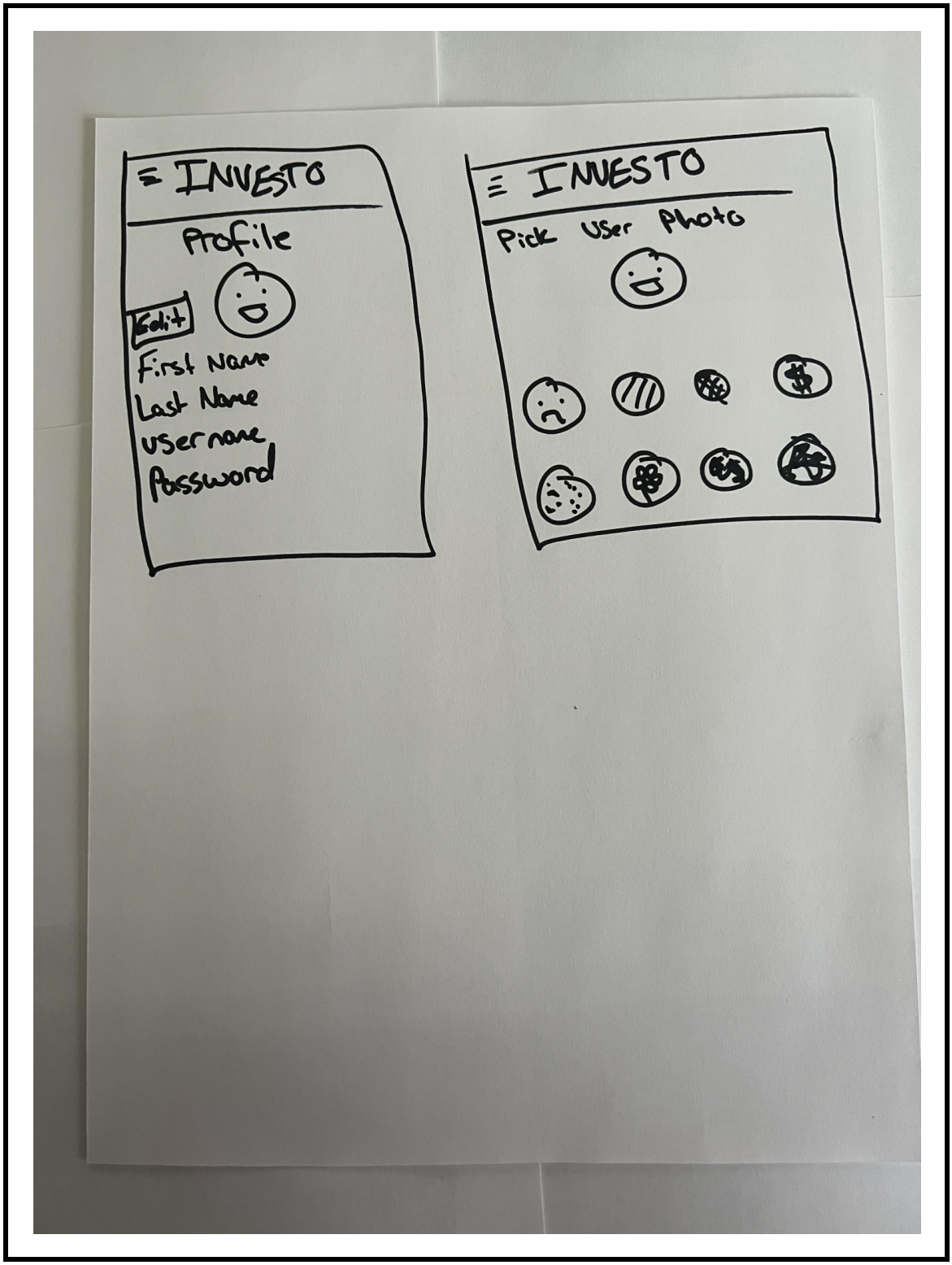


Figure 26: Sketch wireframe

Prototypes

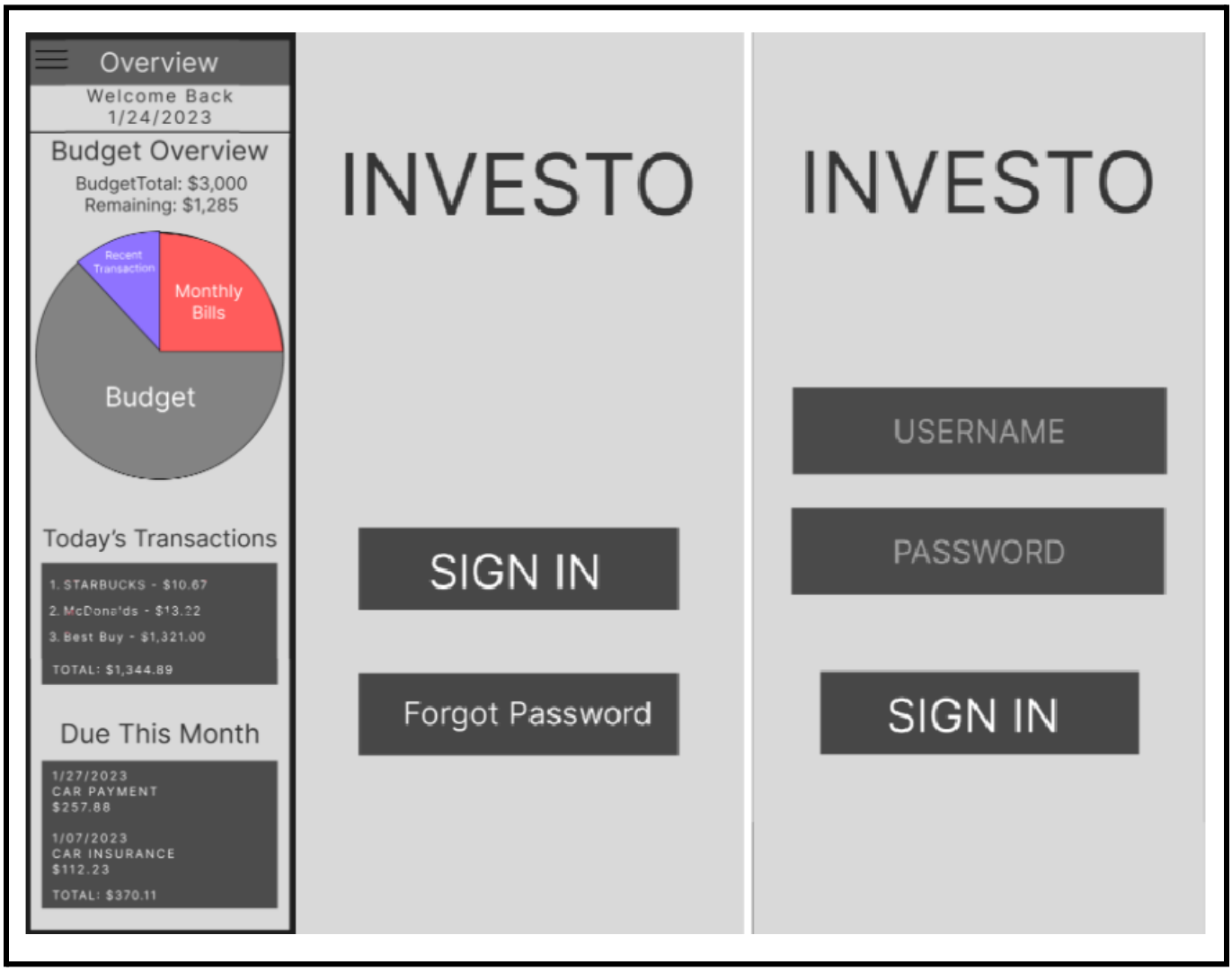


Figure 27: Figma wireframe for Investo

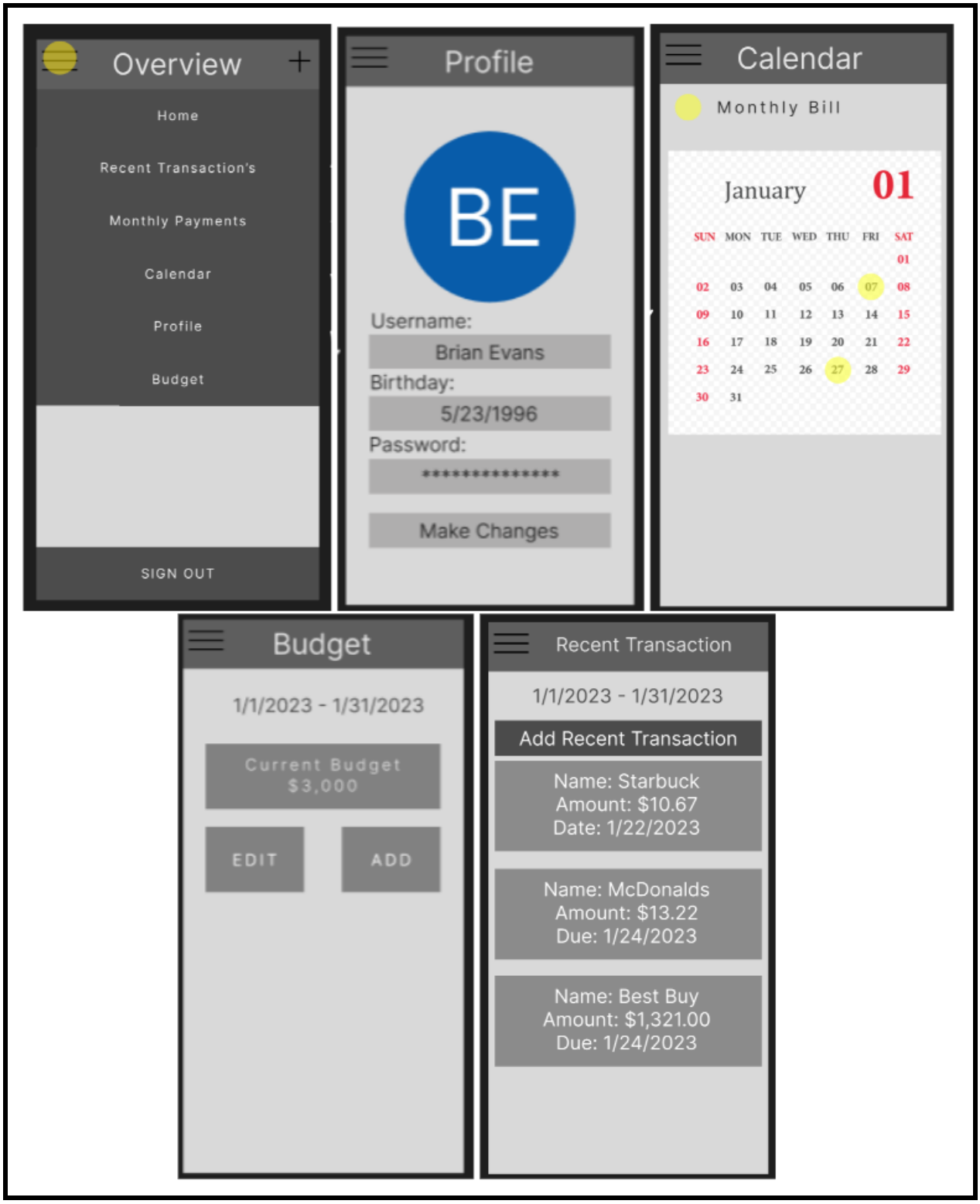


Figure 28: Figma wireframe for Investo

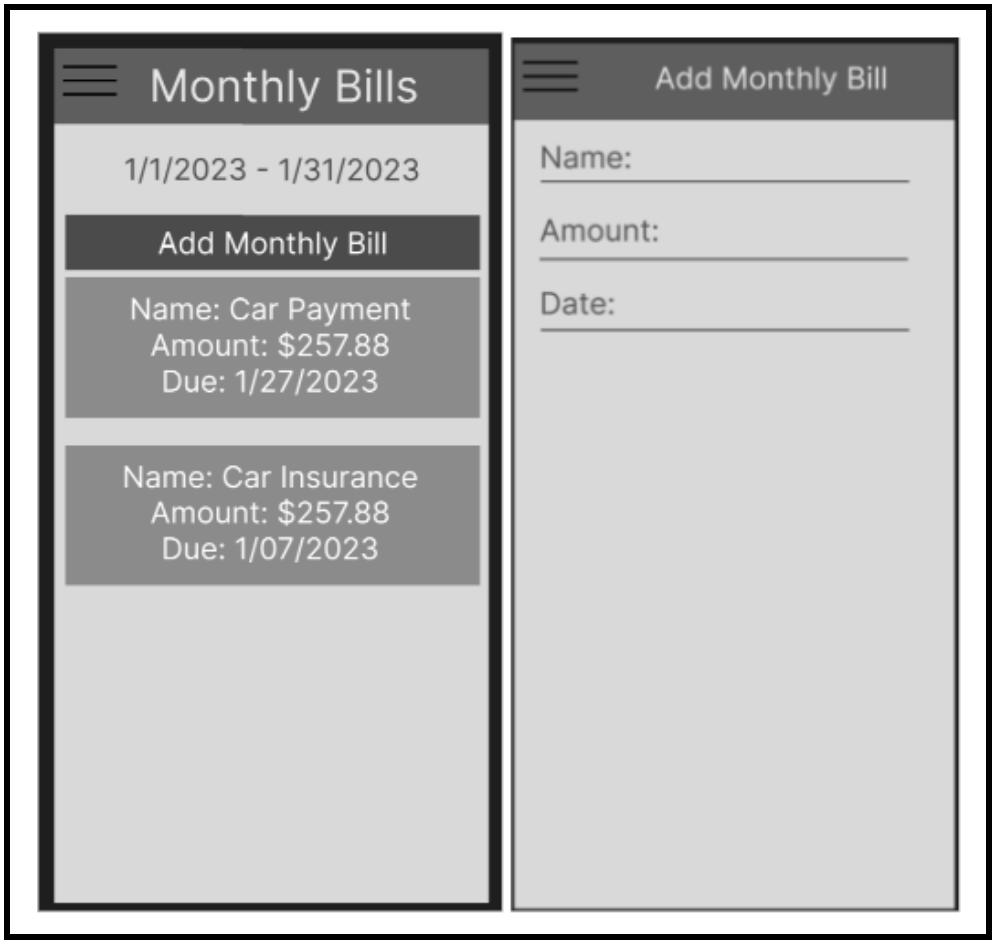


Figure 29: Figma wireframe for Investo

HCI Rules Discussion.

Strive for Consistency - Investo maintains a consistent sequence of actions throughout the process flow of our mobile app. We provide consistent colors and layouts throughout all possible views and maintain the same terminology within all activities. It's important to avoid increasing the cognitive load on the end-user and causing confusion. The user flow is consistent across all layouts and the icons/labels in the sidebar are always the same.

Keep Users in Control - Investo is a responsive mobile application that allows users to drive their experience to maintain a sense of control. Investo's layouts all consist of similar back-end logic that behaves in a consistent and predictable manner. Our top priority was making information easy to access for the user without too much difficulty. The sidebar provides an easy-to-navigate interface for accessing different layouts.

Offer Informative Feedback - Feedback is a critical component of Investo's design. If our application cannot provide the information requested by the user, then the user has no reason to keep using Investo. We designed the application so the user knows where they are in the process and can navigate freely. Every action has appropriate back-end functionality to respond when user input is made.

Final Screenshots

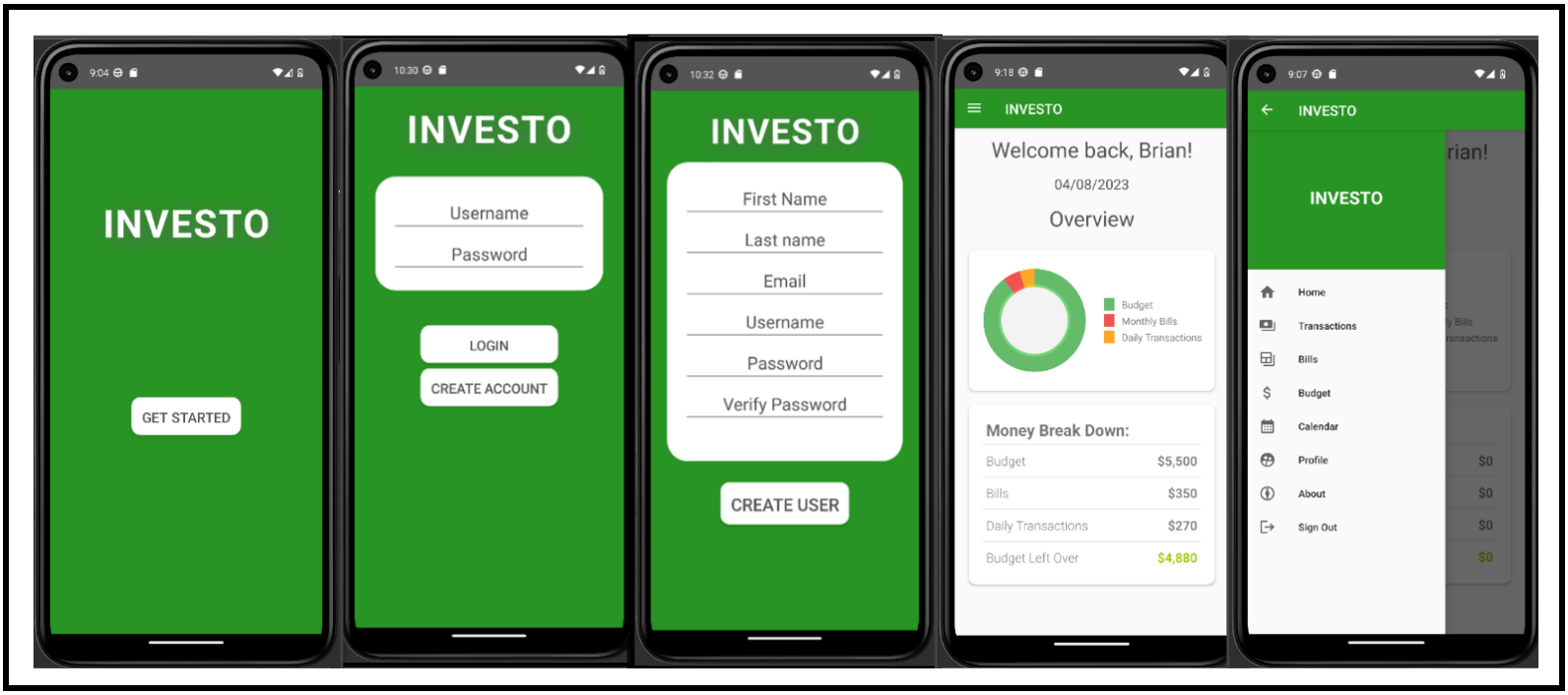


Figure 30: Final screenshots

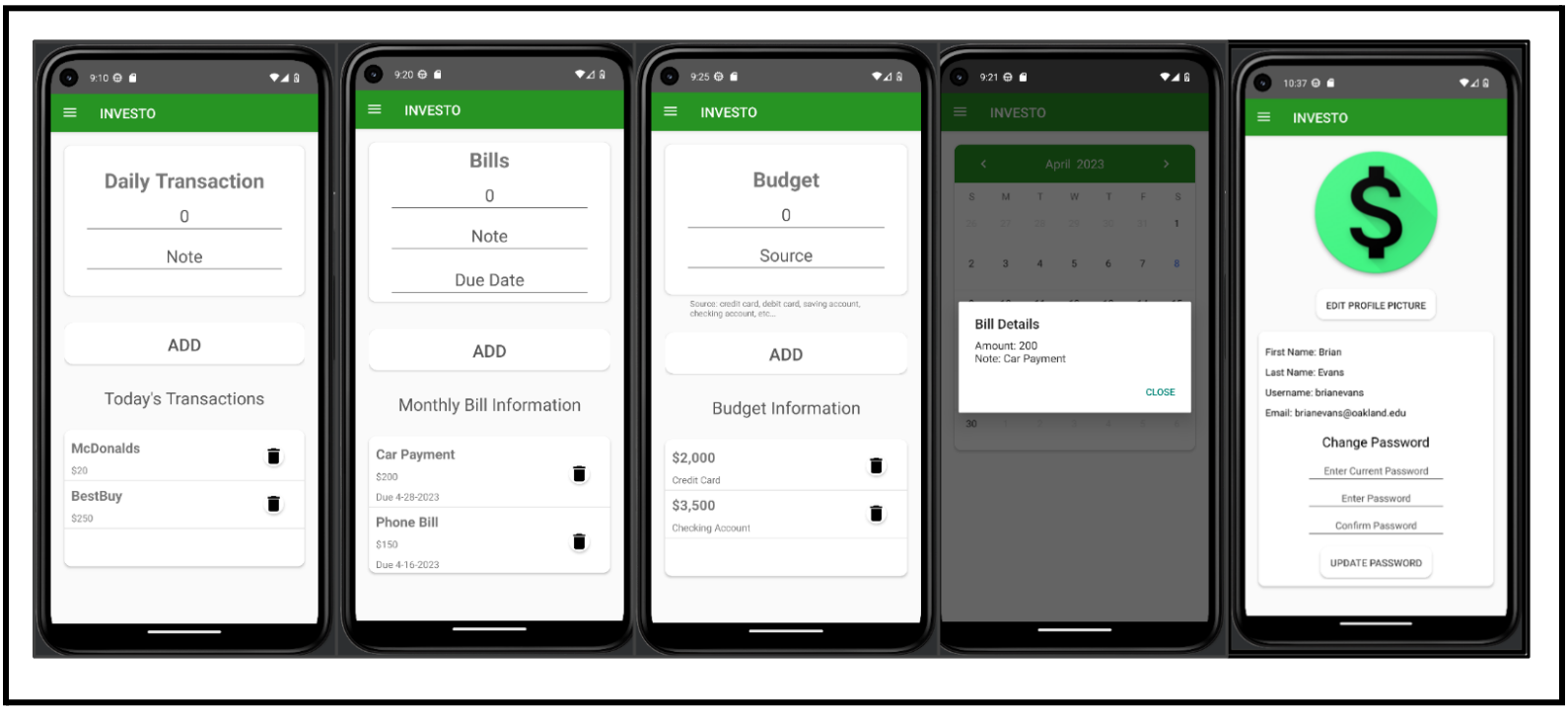


Figure 31: Final screenshots

Figure 32: Responsive HTML Page (desktop view)

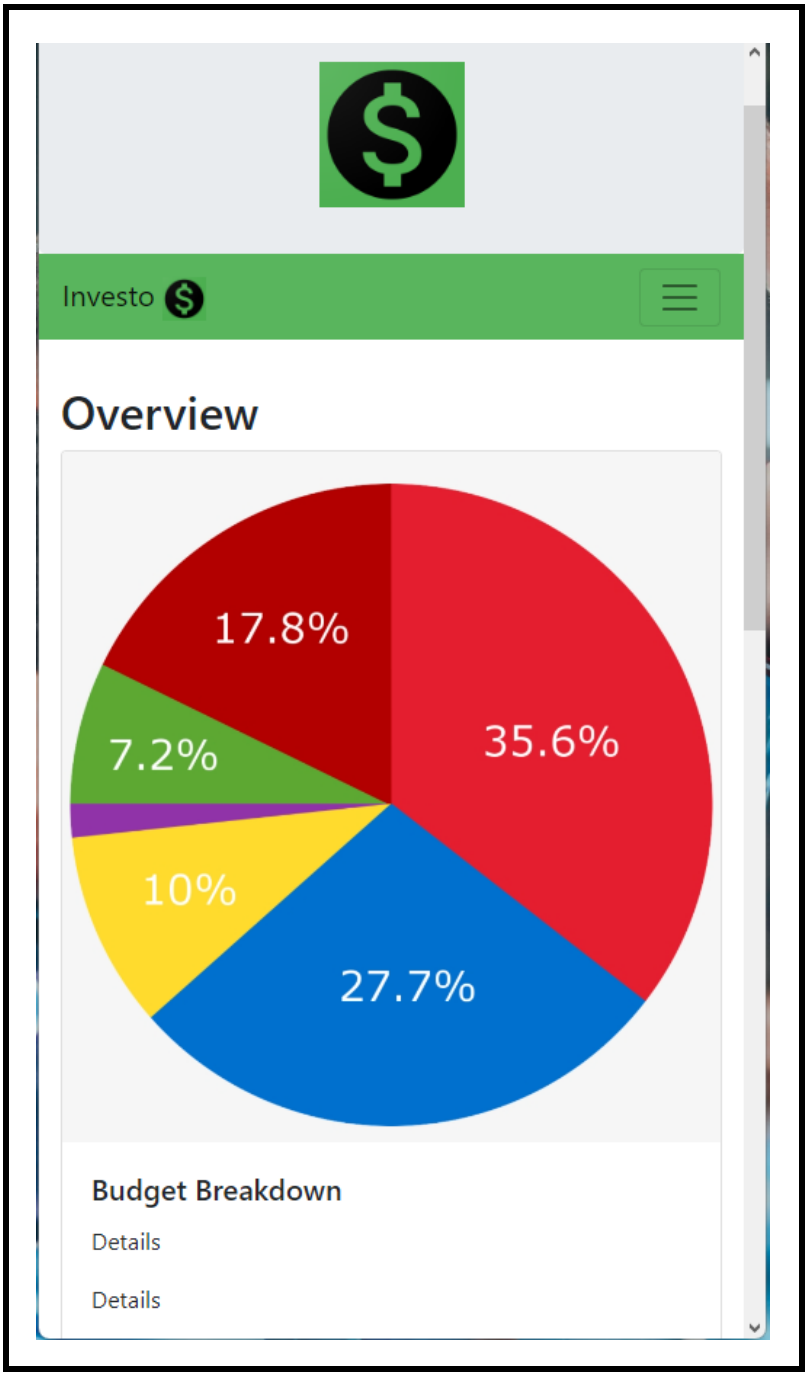


Figure 33: Responsive HTML Page (mobile view)

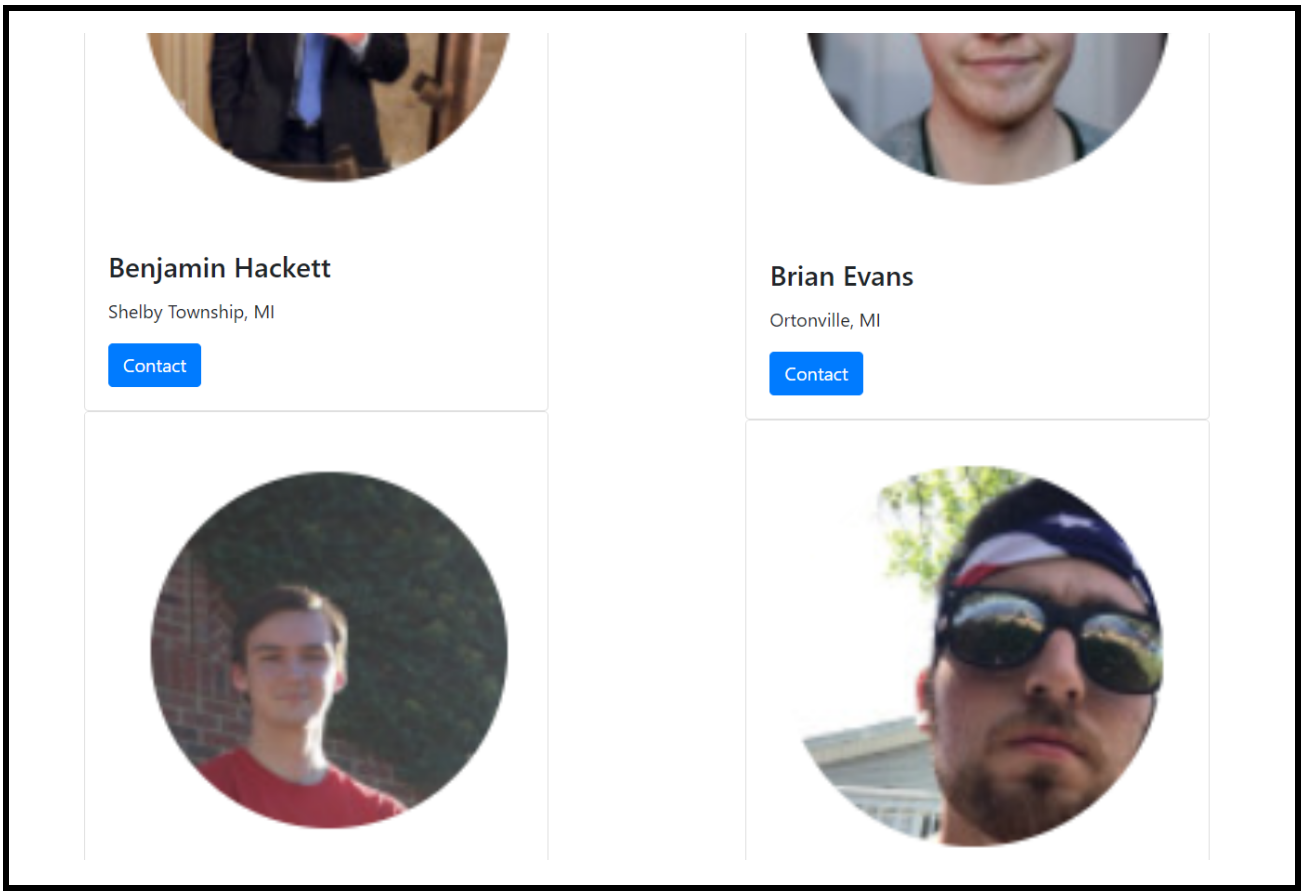


Figure 34: Responsive HTML Page

Database Design

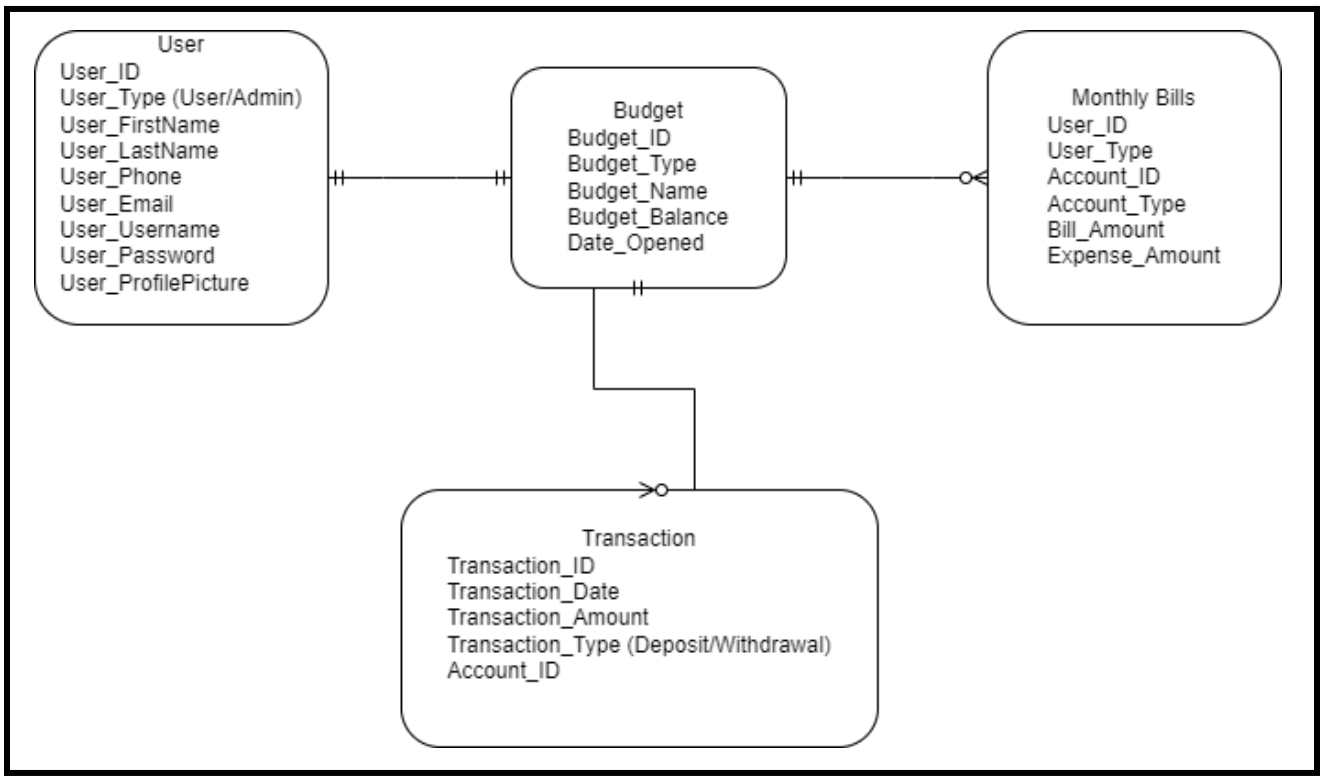


Figure 35: ERD Diagram for Database

Table 1: Bank Account
Account_ID
Account_Type (Checking/Saving)
Account_Name
Date_Opened
Account_Balance
Account_Type (Business/Personal)

Table 2: Transactions
Transaction_ID
Transaction_Date
Transaction_Amount
Transaction_Type (Deposit/Withdrawal)
Account_ID

Table 3: Users
User_ID
User_Type (Adult/Business)
User_Name
User_Phone
User_Email
User_Login
User_Password

Table 4: Monthly Bills/Expenses
User_ID
User_Type
Account_ID
Account_Type
Total_Amount

Sample Code

createUser Function

This code is a snippet from our CreateUser Class that runs when the user creates a new account and clicks “Create User". The first bit of code following the onClick(View v) is initializing string variables to the important fields that the user entered in; their name, username, and password. Next, we check whether the user's input is valid. First, we must check if the user put anything in the boxes, since the user cannot input nothing (aka empty string or “”).

Next, the second if statement checks whether the user wrote the same password for the “Password” and “Verify Password” fields. These fields need to be the same so we check whether they are equal. If they are not the same, we display to the user that the passwords do not match and make them fix it. Once the passwords match, we move on and then start the process of verifying the user’s information.

This is shown in Figure 36.

```
btn_createUser.setOnClickListener(new View.OnClickListener() {

    @Override

    public void onClick(View v) {

        String fname = et_firstName.getText().toString();

        String username = et_username.getText().toString();

        String password = et_password.getText().toString();

        String rePassword = et_repassword.getText().toString();


        if (!username.equals("") || !password.equals("") || !rePassword.equals("")) {

            if (!password.equals(rePassword)) {

                Toast.makeText(CreateUser.this, "PASSWORDS DOES NOT MATCH", Toast.LENGTH_SHORT).show();

            } else {

                //both password match, adding new user account

                //Toast.makeText(CreateUser.this, "PASSWORDS MATCH", Toast.LENGTH_SHORT).show();

                    Boolean verifyUser = dbHelper.verifyUserName(username);
```

Figure 36: This code was used to check the validity of the user information when creating an account.

calendarClickedDate Function

This function is from the CalendarActivity Class, and runs when the user is viewing the Calendar and clicks on any date. Our first line of code is creating a format for the date in order to cut off any leading 0's in cases like 03-03-2023 for March 3rd, 2023. The next line uses this format to format the date of the calendar day that the user clicks on and setting that to 'clickedDate'.

We then pull up the bill amount and note from the database using dbHelper and assigning it to local variables. The next line creates an Alert Dialog, which is essentially a pop-up rectangle displaying the information from that day. Inside the box, it tells you the amount and note values, plus a 'Close' button to exit the alert. If the chosen day does not have a value, 'null' is displayed for both values; otherwise the bill details that the user entered are displayed. This is shown in Figure 37.

```
public void calendarClickedDate() {
    bills_calendar.setOnDayClickListener(new OnDayClickListener() {
        @Override
        public void onDayClick(@NonNull EventDay eventDay) {
            SimpleDateFormat dateFormat = new SimpleDateFormat("M-d-yyyy");
            String clickedDate = dateFormat.format(eventDay.getCalendar().getTime());

            String amount = dbHelper.getBillAmount(clickedDate, getUserID());
            String note = dbHelper.getBillNote(clickedDate, getUserID());

            AlertDialog.Builder builder = new AlertDialog.Builder(CalendarActivity.this);
            builder.setTitle("Bill Details");
            builder.setMessage("Amount: " + amount + "\nNote: " + note);
            builder.setPositiveButton("CLOSE", null);
            builder.show();
        }
    });
}
```

Figure 37: This code was used to view the billing information of the clicked calendar day.

updateUserPass Function

Our final function comes from the DbHelper Class, but is accessed from the MainProfileActivity Class. This function runs when the user is in the Profile activity and is trying to update their password. Specifically, it only runs once the MainProfileActivity detects that both the password from the database is equal to the current password the user enters **and** the new password is equal to the re-inputted/confirm password input. Thus, once the new and old passwords are deemed valid, we update the database, removing the old password and adding the new one.

In the code above, we first get a writable instance of our SQLiteDatabase and a ContentValues object, which is a key-value store, to store the new password for the user. We then update the database by calling db.update with four arguments: the name of the table in the db, the ContentValues object cv, a selection string that will update the rows where the ID value is equal to the userid, and String[] {userid} to specify which values to substitute in the selection string. We then close our access to the database after the update is complete.
This is shown in Figure 38.

```
public void updateUserPass(String userid, String newPass) {
  SQLiteDatabase db = getWritableDatabase();

  ContentValues cv = new ContentValues();
  cv.put(Contract.User.COLUMN_PASSWORD, newPass);

  db.update(Contract.User.TABLE_NAME, cv, Contract.User._ID + " =?", new String[]{userid});
  db.close();
    }
```

Figure 38: This code is used to update the user's password.

ADDIE Model Discussion

This ADDIE model can be implemented into our application planning, designing, and development of our application.

In the analysis phase we can first determine the need for this application, the targeted audience, how long the development of this application should take, and what is required in order to create this service.

With the information gathered from the analysis phase, we can then move on to the design phase. In this phase, we are able to develop wireframes and prototypes which will be the foundation of the application.

With the analysis and design phase completed, we are able to begin development. In this phase we are able to create a working prototype, to test features and functionality to ensure all the advertised services are working accordingly, into a production-level application. Data such as pictures, text, and videos will also be created and collected.

The implementation phase allows us to launch the application onto an online platform, such as Google Play. This launch can be advertised on social media platforms to gain public interest and more foot traffic within the application. After users have begun using this service we are able to take end-user feedback to better enhance the application and or fix bugs that have occurred.

Finally, in the evaluation phase, we are able to take a step back and determine that the application as a whole is working as intended. Further software testing can be accomplished by having users test the application and all of its features included. This testing will be completed during the early version of the application following production-level release and continued user support. This allows us to determine design aspects that should be altered.

Appendix A: System Vision Document

Problem Description:

Many people struggle with budgeting and need a way to organize their finances. Investing has also become more popular and advice for new investors is in demand.

System Capabilities:

Displays bank balance
Review of transactions
Spending advice and goals
Notifications for reaching saving/ investing goals

Business Benefits:

Create a personalized experienced of financial help for users
Help users budget
Provide investing advice

Appendix B: Work Breakdown Structure

I. Research the problem to understand the related details

1. Find/ explore similar apps to compare what problems they solve - 2 hours
2. Define use cases -1 hour
3. Brainstorm features - 3 hours

II. Design the solution

1. Design wireframe/ layout for vision of app - 2 hours
2. Design database tables - 4 hours

III. Build the app

1. Code the UI - 4 hours
2. Code database implementation - 8 hours
3. Code logic for functionality - 20 hours

References

Meet android studio : Android developers. Android Developers. (n.d.). Retrieved January 1, 2023, from https://developer.android.com/studio/intro

Murach, J. (2015). *Murach's Android Programming, 2nd Edition* (2nd ed.). Mike Murach & Associates.

A computer science portal for geeks. GeeksforGeeks. (n.d.). Retrieved January 1, 2023, from https://www.geeksforgeeks.org/

VASAN, L. A. L. I. T. (2021, March 29). *Android tutorials and tips*. Android Tutorials Hub. Retrieved January 1, 2023, from http://www.androidtutorialshub.com/

Google Developers Training Team. (n.d.). *Introduction*. Introduction · GitBook. Retrieved February 1, 2023, from https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/