Pinchr

(expense manager web app)

CPSC 362–03

2/29/2024

Samuel Vo

Eduardo Gaxiola

Josh Reyes

Andrew Cesario

Robert Moberly

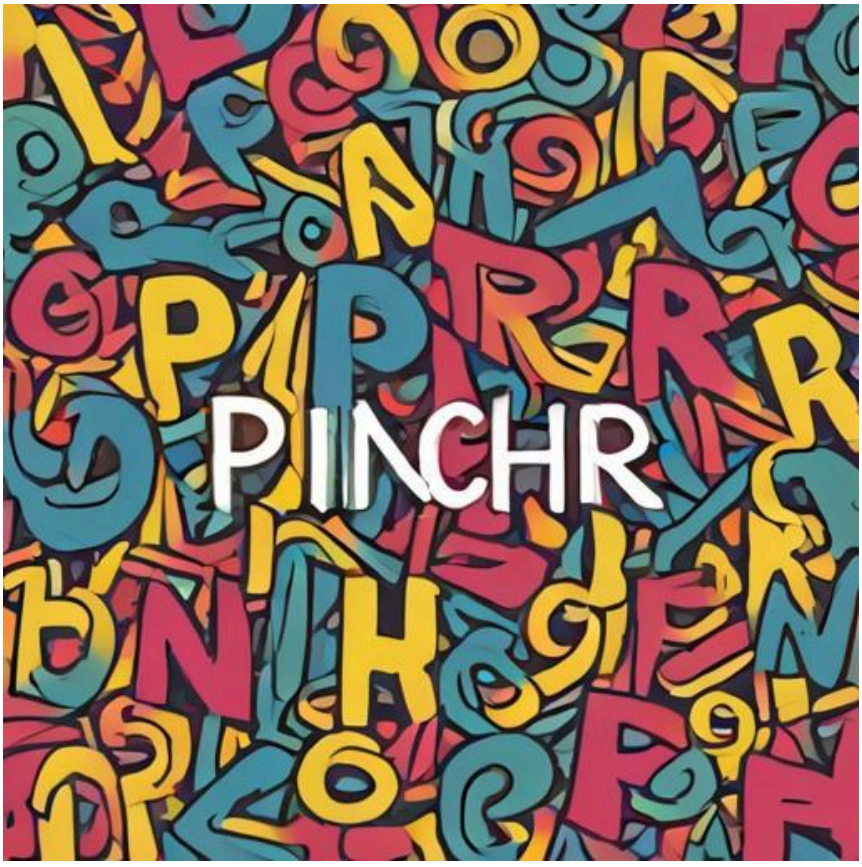


Table of Contents

Introduction

1.1 Executive Summary

Pinchr is an expense manager that is simple and lightweight. Central to the design of the service is control and simplicity. Pinchr does not require invasive bank APIs or other access to sensitive financial data in order to function. Users are put in control of their budgeting, monthly income, and expense statements with simple tools that create a clear overview of their financial journey.

Users can enter their expenses, give them tags and sort them into categories, and see how their money is spent. They are able to create budgets and savings goals, encouraging them to spend wisely and save thoughtfully. With Pinchr, users are reminded of their monthly recurring expenses, build smart spending habits, categorize their spending for easy monitoring, and stay on top of their financial health with ease.

1.2 Terms and Acronyms

Pinchr - the name of the software. It is a simple and lightweight web application to manage expenses.

UI - Stands for user interface. It is the visual elements the user sees on the website or app they are interacting with as they navigate through the pages and features.

DB - short for Database. This is where data is stored, including user credentials, expense data and any other information the user inputs.

HTML - short for Hypertext Markup Language. This programming language is used for creating and structuring the content for a web page.

Frontend Web Development - This is the term used when referring to creating the looks, visuals and design of the software.

CSS - Abbreviation for Cascading Style Sheets. This is the language used for styling a web page.

JavaScript - A core programming language for web-based applications. This language is used to create the behavior of websites and applications.

Node.js - is an environment for running JavaScript applications. Node.js is ideally suited for tasks that use a large amount of data.

Google Firebase - A collection of tools and cloud computing services for utilizing databases, hosting applications, and providing user authentication.

Backend Web Development - The term used when referring to the development of features, functionality, and services that the user don't directly interact with but make the app work.

GitHub - A Version System Control Hosting Platform for hosting software, giving release notes, and updating information regarding your software.

Chart.js - A charting library for JavaScript. It is used for creating data visualizations like bar, pie, line, scatter, or other types of charts.

D3.js - Another charting library for JavaScript. It focuses more on dynamic and interactive data visualizations for web browsers.

Django, Django ORM - Django is a dependency that runs on a web server and is a great backend web development framework to provide secure and reliable functionality on the server side of the web application.Django ORM is one of its features that is included called (Object Relational Mapping). This feature basically allows you to manipulate data in a relational database using object-oriented programming.

React - A JavaScript library that allows you to create dynamic and interactive UIs.

MySQL - Stands for (My Structured Query Language). "My" is named after the daughter of the co-founder, Michael Widenius. It is simply an open-source relational database management system used in small-scale to large-scale projects, applications, and software.

Microsoft Azure - A cloud computing service offered by Microsoft. It offers other services as well such as database, networking, storage solutions, and hosting as well for many types of web applications.

1.3 Requirements

Functional

- Allows users to create an account with the website using Google sign-in or by email. Users are allowed to use email as a username and create a unique password if they don't have a Google account.
- Allows users to set a budget amount. The user manages the budget in this phase by changing the amount when necessary.

- Users could edit, add or delete expenses. Users will have the option to create categories for different types of expenditure activities such as food and drinks, utilities, entertainment, hobbies, and more. The user manages the expenses according to their needs.
- Users will be able to generate, interact, and view a report of their expenses in different formats like a chart, graph, or some sort of statistical visual representation in order to track their spending. The user could also export their data report if they want.

Non-functional:

- Simple, intuitive, web responsive and user-friendly UI.
- Budget information will be displayed.
- Login forms will display for user input.
- Features for creating a category and buttons for users to input expense data will be displayed.
- Expense data will be displayed on graphs, charts, or any other visuals during report generation.

Design

2.1 Use Cases

Use Case 1	**Priority:** High
Use case name:	Create Account
Primary Actor:	New User
Goal:	To register an account to track personal expenses.
Preconditions:	The user must have access to a valid email address.
Trigger:	The user selects the "Create Account" option.

Scenario:	1. User enters the address to the web application. 2. User selects the “Sign Up” option. 3. User enters a chosen email address and password. 4. The system validates the email format and password strength. 5. The system sends a confirmation email to the user. 6. User confirms the email by clicking a link. 7. System logs the user into their new account. 8. User is directed to the homepage.
Exceptions:	Email already in use, invalid email format, password too weak, failure to confirm email.
Frequency:	Once per user.
Channel to actor:	Web Browser Interface with internet connectivity.
Secondary actors:	Email Verification System, Database
Open issues:	How to handle users who do not receive the confirmation email.

Use Case 2	**Priority:** High
Use case name:	Login to Account
Primary Actor:	User
Goal:	To allow the user to access their account by logging in.
Preconditions:	The user must already have an account.
Trigger:	The user selects the "Login" option.
Scenario:	1. User navigates to the Login web page. 2. User selects the “Login” option. 3. User enters their email address and password. 4. The system validates the credentials. 5. The system grants access to the user’s

	account. 6. User is directed to the homepage.
Exceptions:	Incorrect email/password, account not confirmed, account locked or suspended.
Frequency:	As needed by the user.
Channel to actor:	Web Browser Interface with internet connectivity.
Secondary actors:	Authentication System, Database
Open issues:	Account recovery options for forgotten passwords or locked accounts.

Use Case 3	**Priority:** Medium
Use case name:	Manage Budget
Primary Actor:	User
Goal:	To create and manage a budget plan.
Preconditions:	User is logged in.
Trigger:	User navigates to the budget management section.
Scenario:	1. User selects “Manage Budget” from the main menu. 2. User sets budget categories and limits. 3. System displays a budget plan. 4. User can modify or confirm budget settings. 5. System saves the budget plan.
Exceptions:	Invalid budget amounts, overlapping budget categories.
Frequency:	As needed by the user.
Channel to actor:	Web Browser Interface with internet connectivity.
Secondary actors:	Budget Analysis Tool, Database

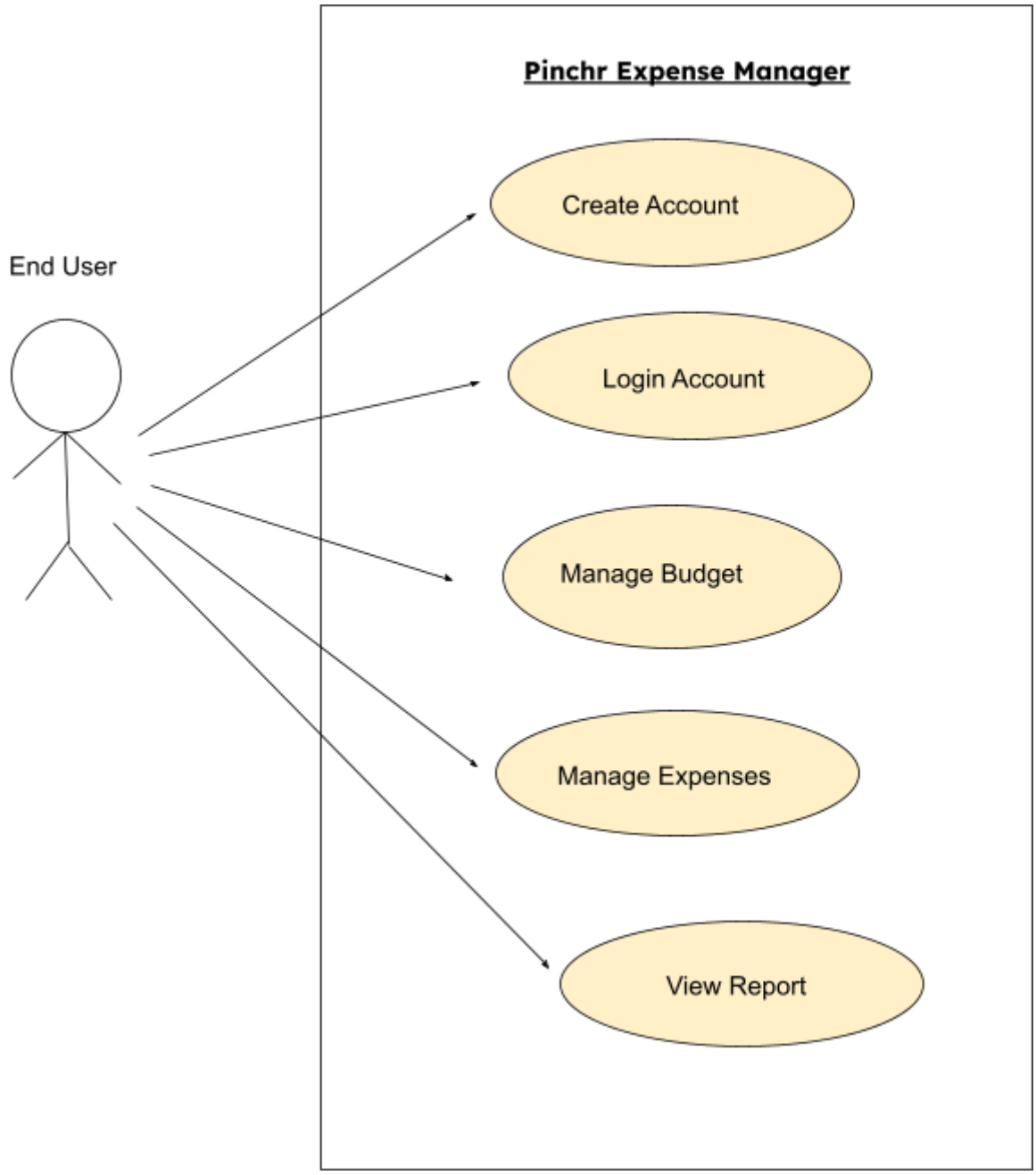
Open issues:	How to adjust budgets automatically based on spending trends.

Use Case 4	**Priority:** High
Use case name:	Manage Expenses
Primary Actor:	User
Goal:	To allow the user to add, edit, and delete expenses.
Preconditions:	User is logged in.
Trigger:	User selects the "Manage Expenses" option.
Scenario:	1. User selects “Manage Expenses” from the menu. 2. User chooses to add, edit, or delete an expense. 3. User enters or modifies expense details. 4. System updates the expense records. 5. System reflects changes in the budget and reports.
Exceptions:	Invalid expense entries, errors in expense categorization.
Frequency:	As many expenses a user has.
Channel to actor:	Web Browser Interface with internet connectivity.
Secondary actors:	Expense Categorization System.
Open issues:	Handling of expenses in foreign currencies.

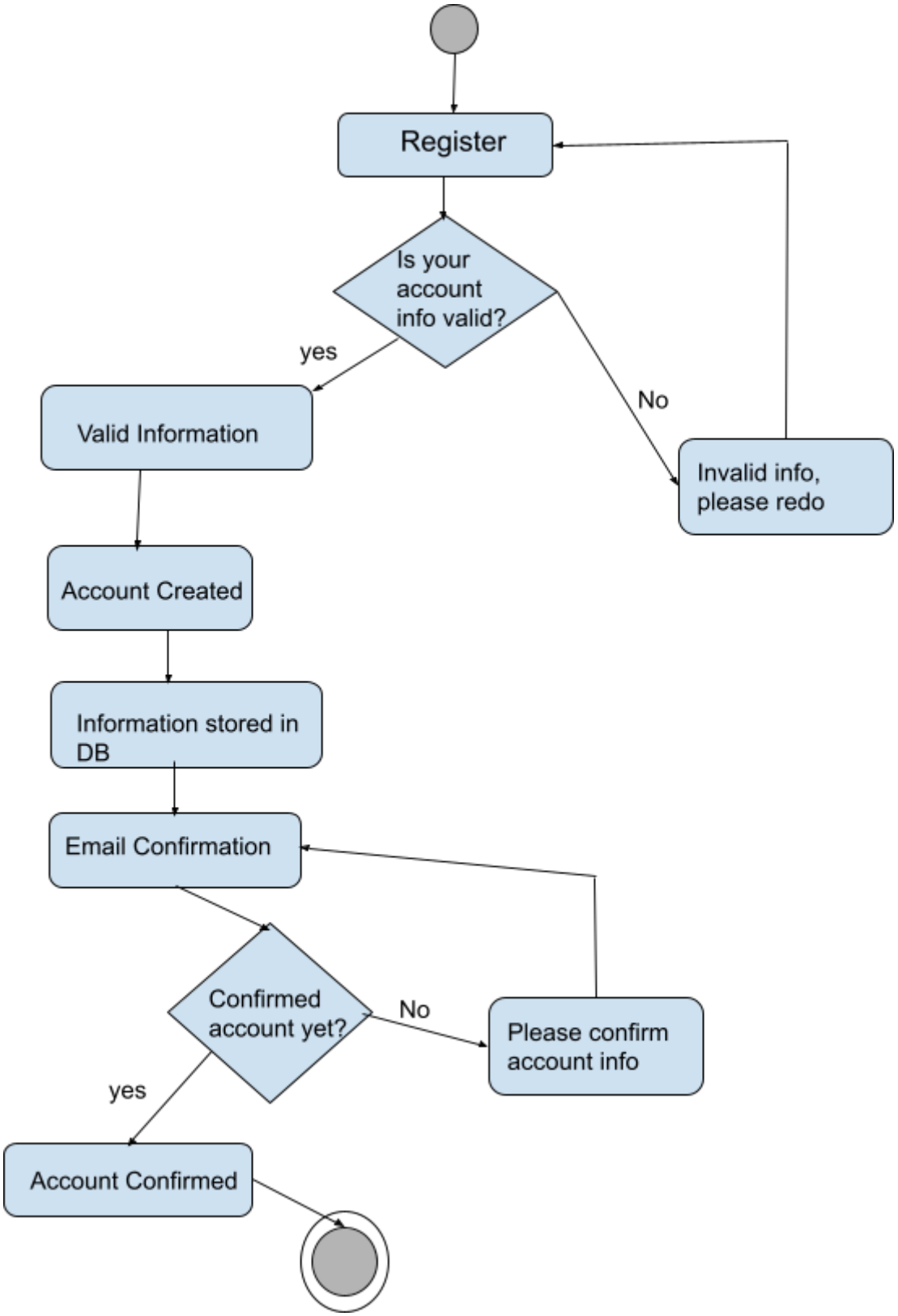
Use Case 5	**Priority:** Medium
Use case name:	View Report
Primary Actor:	User

Goal:	To provide the user with a graphical view of their spending.
Preconditions:	User is logged in and has entered some expenses.
Trigger:	User selects the "View Report" option.
Scenario:	1. User selects “View Report” from the main menu. 2. User chooses the type of report they wish to view (e.g., monthly expenses, category breakdown). 3. System generates the report based on the existing expense data. 4. System displays the report in the chosen format (graphs, pie charts, etc.). 5. User can interact with the report to view different details or export the data.
Exceptions:	No data available to generate reports.
Frequency:	As needed by the user.
Channel to actor:	Web Browser Interface with internet connectivity.
Secondary actors:	Data Visualization Tool, Database
Open issues:	Customization of reports based on user preferences.

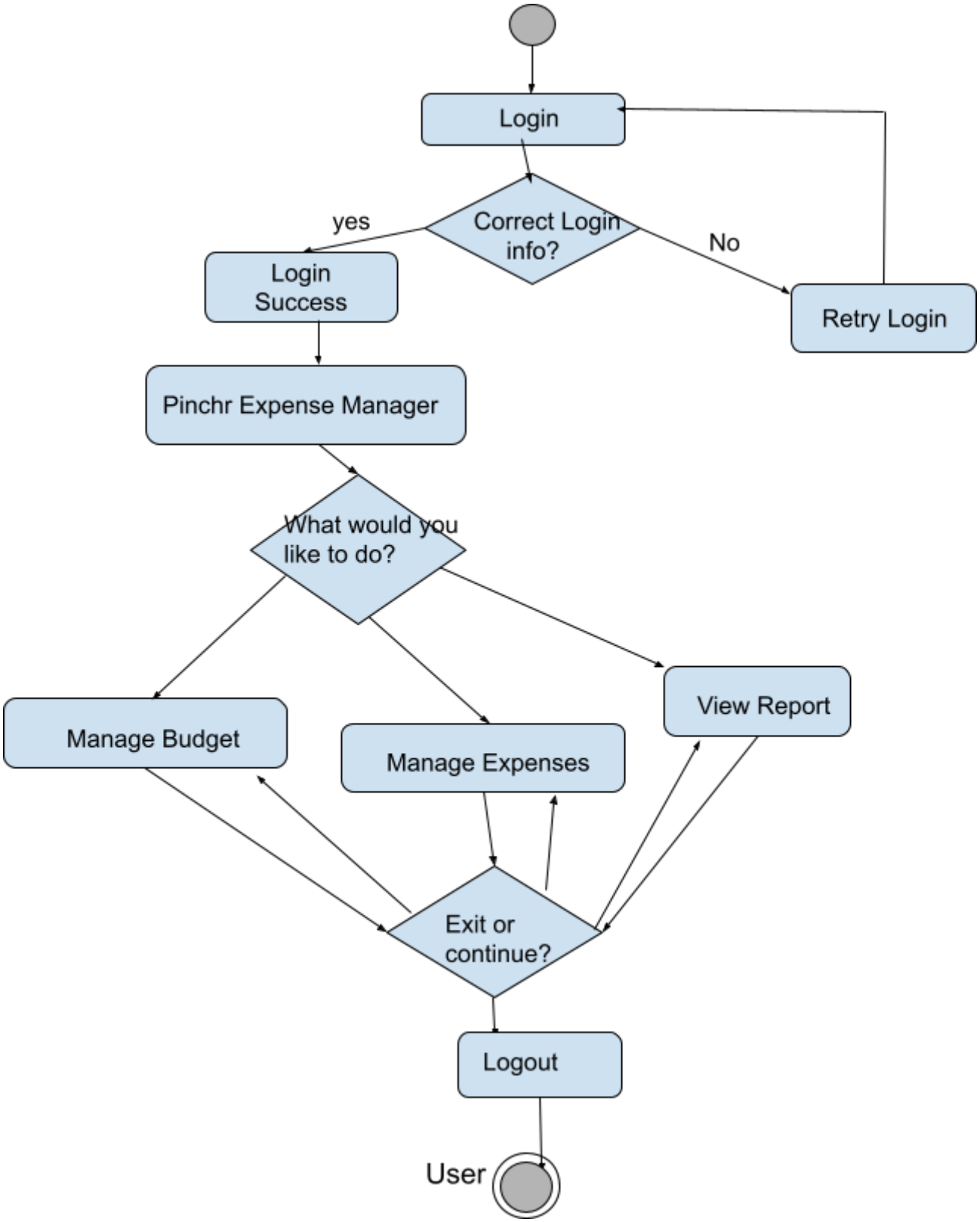
2.2 Use Case Diagram



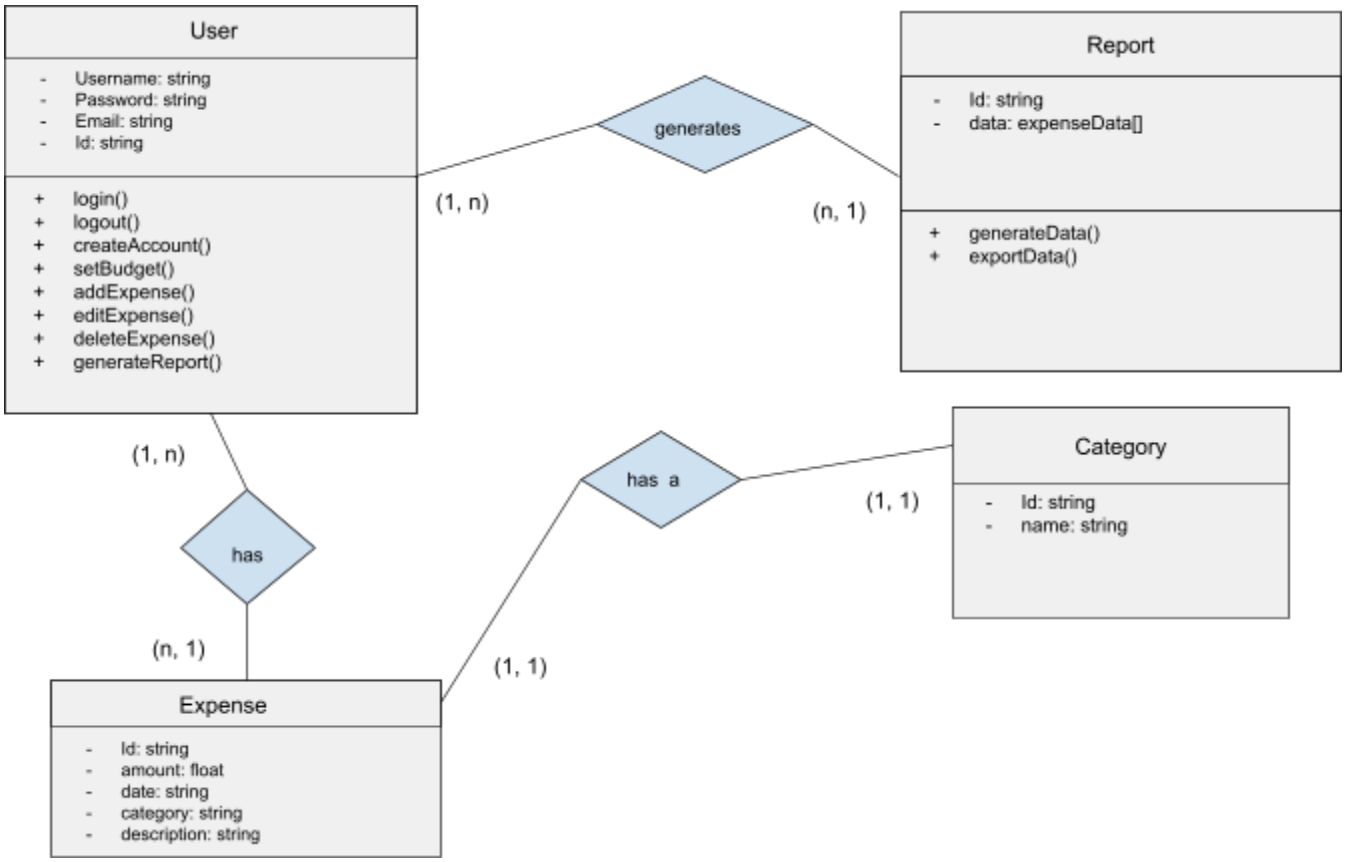
2.3 Registration Activity Diagram



2.3 Web App Activity Diagram



2.4 UML Diagram



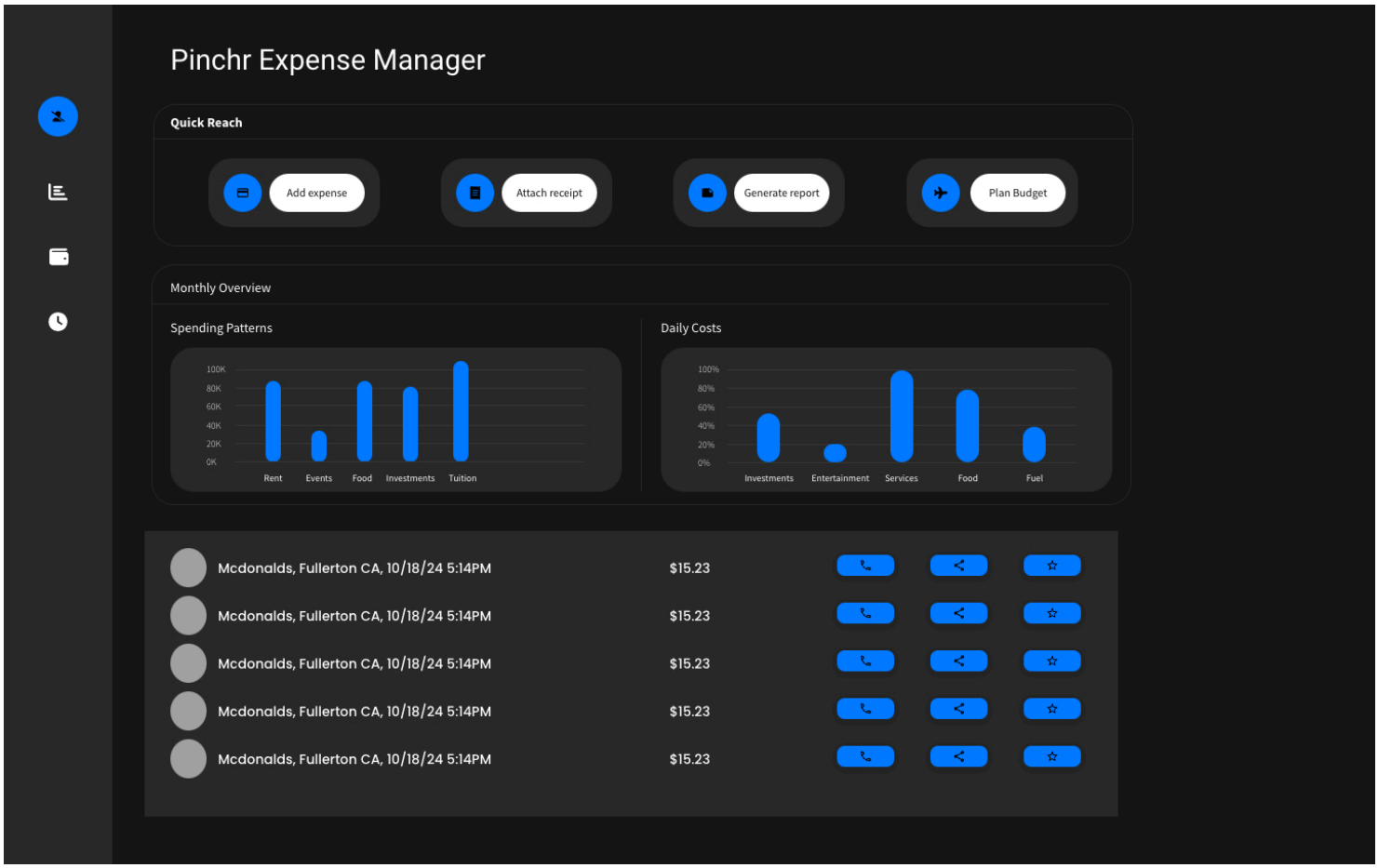
2.5 Software environment/dependencies

- Login Information and Authentication - Google Firebase
- Frontend Web Development - HTML, CSS, JavaScript
- Responsive Design Framework - Bootstrap
- JavaScript Library - React
- Charts/Graph library - Chart.js, D3.js
- Database - MySQL, Firebase
- Backend Development/Services - Firebase, Django
- Deployment - Microsoft Azure, Firebase
- Server environment - Node.js
- Version Control System Hosting Platform - Github

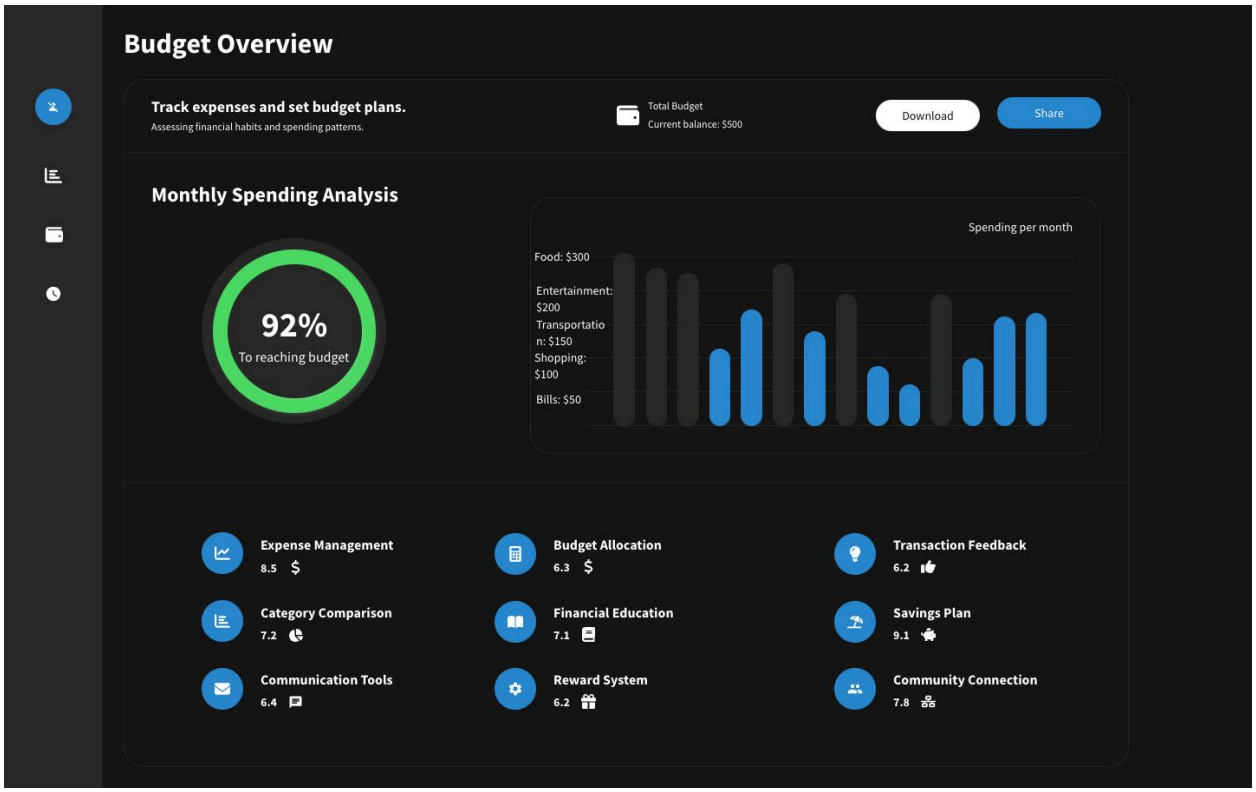
2.6 Mockup UI

Login Page

Home Page


Pinchr Expense Manager
Quick Reach
Add expense
Attach receipt
Generate report
Plan Budget
Monthly Overview
Spending Patterns
100K
80K
60K
40K
20K
0K
Rent
Events
Food
Investments
Tuition
Daily Costs
100%
80%
60%
40%
20%
0%
Investments
Entertainment
Services
Food
Fuel
Mcdonalds, Fullerton CA, 10/18/24 5:14PM
$15.23
Mcdonalds, Fullerton CA, 10/18/24 5:14PM
$15.23
Mcdonalds, Fullerton CA, 10/18/24 5:14PM
$15.23
Mcdonalds, Fullerton CA, 10/18/24 5:14PM
$15.23
Mcdonalds, Fullerton CA, 10/18/24 5:14PM
$15.23

Expense Analysis



Notes

3.1 Testing Strategies

Unit testing - This test will focus on code that tests if the individual parts of the program will match the requirements given. Usually, tests are created with methods in mind to create a value that hopefully matches the requirements. We will implement this testing strategy whenever applicable in our software to ensure the individual parts of code like functions work properly and align with the requirements.

Functional Testing - Functional testing is verification that a system does what it needs to do according to the application requirements and performs the intended task. This type of testing will involve software input and output, data, user interactions, and how the program responds to all of the different events happening.The whole point of functional testing is just to make sure that the program works as intended. We will definitely need this type of testing in our software to ensure that all data input, output, and user interactions works properly without any errors or bugs.

User Interface Testing (UI) - UI testing is the process that validates the functionality and visual appeal of the interface that a user will interact with. It checks that the visual elements of the program are functional according to requirements, and tests to make sure that all interactable buttons work bug-free. From data type testing where it is checked that only valid data can be entered into a certain prompt such as expenses and dates, to making sure that buttons in a menu bring you to the right screen. This testing will make sure all menus, features, and settings will navigate the end user to the correct page.

Regression Testing - this testing is the process of re-testing functional and non-functional requirements that have been previously tested by other testing strategies mentioned above to confirm that everything works properly after modifications have been made. It is used for maintaining software stability, quality, and performance in a situation where the software might have a feature that once worked but suddenly failed after a change has been made to fix another issue. This testing strategy is important in our project because we need to make sure users are consistently able to access their account, expense data, and reports without any issues.

3.2 Code Review and Version Control Procedures

Version Control Procedures

- Version Control System: Our project will be using GitHub as our version control system
- Branching Strategy: We'll be focusing on a Trunk-Based branching strategy. Developers will be making small, continuous commits to the main branch.
- Commit Messages: Each commit message should detail briefly what was implemented and changed. If needed, these messages should refer to an ongoing need or subject of development conversation.
- Merge Policy: Once code has been sufficiently verified through testing procedures, passes integration checks, and is cleared through ongoing discussions, it will be merged with the main branch.
- Release Management: Releases are distinguished between minor and major releases. Major releases update the version number, where minor releases update the revision number. These are separated by a decimal, so revision 3 to major version 1 is denoted as version 1.3.
- Merge Conflict Handling: Merge conflicts are handled with developer meetings if needed. Additional testing can be developed by the team to determine the cost in system resources and gather additional data to make an informed decision. Each conflicting version is kept on hand in the event the implementation can be used in the future.

Code Review Procedures

- Purpose of Code Review: The purpose for implementing code review procedures into our project is to ensure the quality, security, efficiency, and a set of standards for our program.
- Participants: Reviewer - a peer developer who examines the code to ensure standards. Author - the developer who wrote the code being reviewed.
- Process of Code Review:
    1. Submission: Authors submit a pull request for their feature to the main branch after properly testing
    2. Manual Review: Reviewer reviews the code using the code review checklist and also giving the author comments or suggestions.
    3. Approval: The reviewer then decides if the branch is safe to be merged.
    4. Follow-Up: If any issues arise after merging, a follow-up review is taken place to fix any issues.

- Code Review Checklist:
    1. Does the code address the issue/feature at hand without out-of-scope changes?
    2. Are there any potential security vulnerabilities?
    3. Is the code conforming to the coding standards?
    4. Is the code covered by new tests and are existing tests still passing?
    5. Are there any potential performance issues?
    6. Is the documentation(inline comments) updated accordingly?
- Handling Feedback: Author must address all feedback reviewer makes, either by making changes or providing clarifications. Reviewers and authors should collaborate to reach a consensus on any points of conflict.
- Tools: Github, inline comments, Discord(communication)

3.3 Risk Table

Risk Name	**Probability**	**Likelihood (1 - 4)** **1 - catastrophic** **2 - critical** **3 - marginal** **4 - negligible**	**RMMM (Risk, Mitigation, Monitoring, Management) Strategies**
Service Downtime	60%	2	Regular maintenance, monitoring and keeping track of any potential impacts that would cause service downtime. Regularly updating the software and finding vulnerabilities would also help mitigate this issue. Implementing redundancy
Data Breach	30%	1	Contain and mitigate services that have been compromised to keep unauthorized access from affecting other user data. Implement security monitoring tools to keep track of suspicious activities. Implement encryption for sensitive user data to protect it on transit and at rest.
Negative User	50%	3	Acknowledge user

Experience			feedback, investigate issues the user faces, prioritize fixing the errors and bugs that the user faced. Implement the solution once it has been created. Have a follow up with the user and continue to improve the web app.
Inadequate Scalability	60%	2	Regularly performance testing, continuously optimizing the software infrastructure and codebase to adjust to scalability would help mitigate this risk. Lastly, using scalable architectural patterns in the web app would be highly advised for scalability issues.
Errors in computation	10%	2	Regular maintenance check, debugging, and updating code to fix errors in computation will solve this issue.
SQL injection	15%	2	Properly sanitizing and validating user inputs, using parameterized queries or prepared statements

			provided by the database library are effective against SQL injection attacks. Configuring the database server would also help minimize the risk of SQL injection because unnecessary settings will be disabled and security patches will be updated regularly to keep it secured.
Insufficient Support and Maintenance	5%	3	Regular maintenance, updates, and focusing on fixing bugs in the program is enough to solve this problem.
Third-party dependencies failure	50%	1	Diversifying dependencies, keeping dependencies updated, monitoring for any changes to the dependencies, and keeping up to date with any release notes from the dependencies will help mitigate this issue.

3.4 Developer Documentation

- Some issues that came up while working on our expense manager web application are getting the frontend and backend to interact with each other properly.
- We spent some time figuring out how to use the dependencies because we never used them before, so it took time to learn and apply them to our web app.
- We were not sure if our web app would be secure enough.
- Designing an intuitive and user-friendly interface for entering and managing expenses.
    - Simple input fields accessed through an “Input expense” button which makes changes to the backend database. “Edit Expense” button along with “Delete Expense” button with respective functionalities. The buttons are easy to find on the main page.
- Handling concurrent access to the MySQL database to prevent data corruption or inconsistencies.
    - We prevent these inconsistencies by wrapping them in a transaction; BEGIN; … COMMIT;
- Implementing mechanisms to synchronize data between the client-side application and the MySQL database.
    - Use Object-Relational Mapping in python as Django ORM to synchronize data between the client-side application and the MySQL database.
- Optimizing database queries and indexing to improve the performance of data retrieval and manipulation operations.
    - Continuously make changes to prepared queries to make sure they touch one the exact function they are built to serve.
- Setting up regular backups of the MySQL database to prevent data loss in case of hardware failures, software errors, or other disasters.
    - Doing consistent backups to multiple devices to prevent data loss. Also look into automated backup mechanisms.
- Encrypting sensitive data such as user credentials and financial information both in transit and at rest.

    - Use MySQL’s built in encrypting and decrypting methods to store and retrieve sensitive data. The methods are AES_ENCRYPT and AES_DECRYPT.
- Protecting against SQL injection attacks by properly sanitizing user inputs and using prepared statements or parameterized queries.
    - MySQL has built in PREPARE statements that are easy to use. When a user wants to insert some data to a field, we use the EXECUTE statement to insert it to the database.