CSC 431

Budget Buddy

System Architecture Specification (SAS)

Team 8

Faisal Sayed	fxs479@miami.edu
Cameron Hackett	cjh200@miami.edu
Hannah Belton	heb74@miami.edu

Version History

Version	Date	Author(s)	Change Comments
0.1	3/20	Faisal Sayed	Initial start
0.3	3/29	Cameron Hackett	Adding info
0.5	4/10	Hannah Belton	Adding info
1.0	4/22	Faisal Sayed	Finished/working version

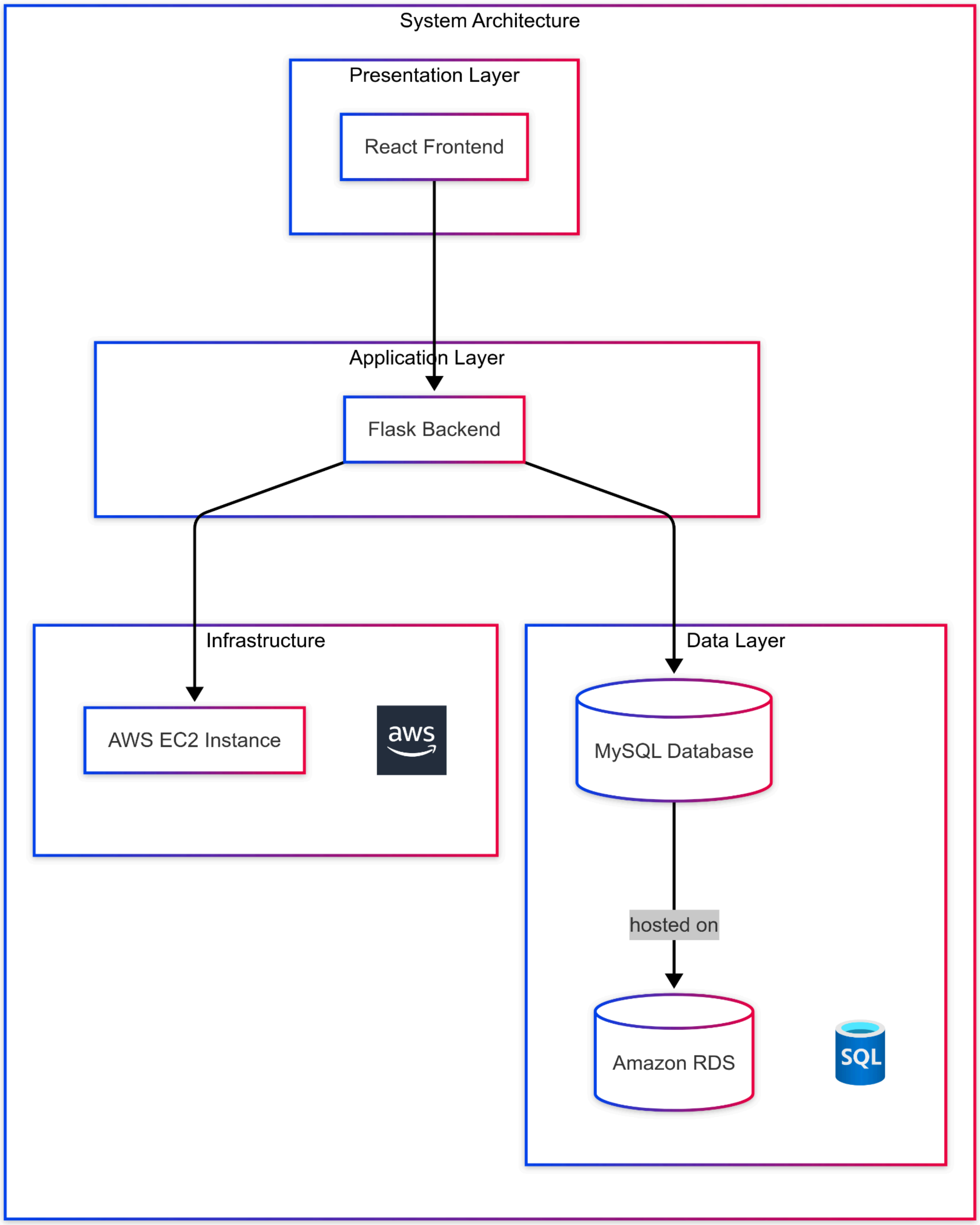
Table of Contents

1. System Analysis

1.1. System Overview

- **Budget Buddy** is a personal finance web application that allows users to track their income, categorize expenses, monitor savings, and set financial goals.
- The system provides an intuitive interface for entering and visualizing financial data, helping users make informed budgeting decisions.
- The application follows a **3-tier architecture** comprising the **presentation layer (React frontend)**, **application layer (Flask backend)**, and **data layer (MySQL database)**.
    - It is deployed on an **AWS EC2 instance**, providing scalability and reliable hosting. This separation of concerns improves modularity, maintainability, and ease of testing.

1.2 System Diagram



1.3 Actor Identification

User: Interacts with the UI to manage income, expenses, and budgets.

System Admin *(optional)*: Manages deployment and monitors server/database performance.

Database: Stores persistent financial data and user information.

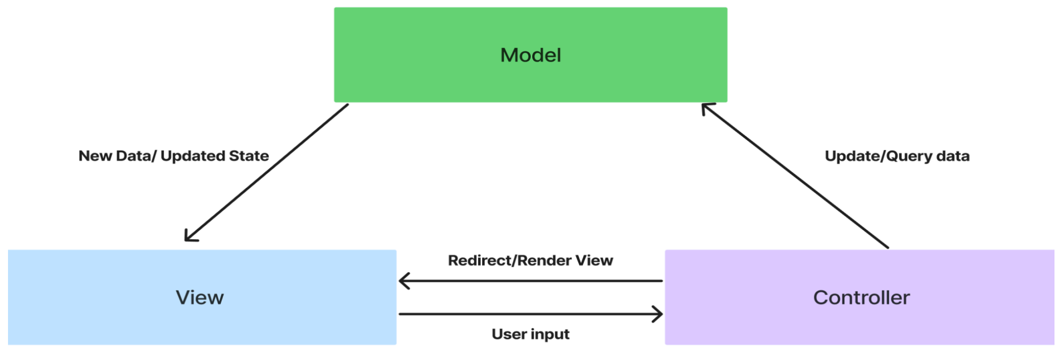
1.4 Design Rationale

1.4.1 Architectural Style

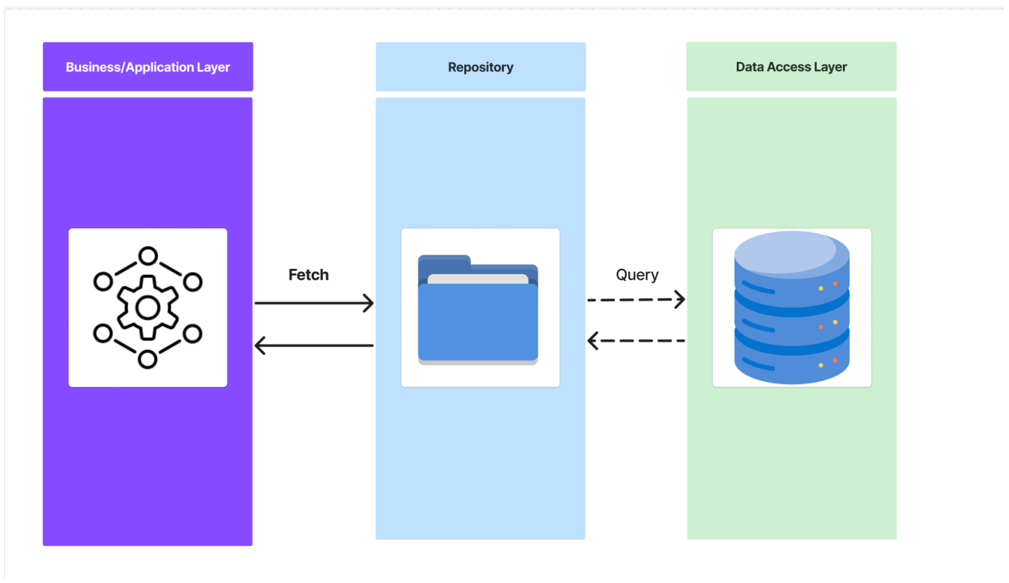
The system uses a **3-tier architecture**, separating concerns into presentation, business logic, and data management. This modular structure enhances scalability and allows independent development and maintenance of each layer.

1.4.2 Design Pattern(s)

Model-View-Controller (MVC): Organizes the backend logic by separating models (data), views (API responses), and controllers (logic/requests).



Repository: Provides a central place to access and manage data



1.4.3 Framework

Frontend: [React.js] for building a responsive, component-based user interface.

Backend: [Flask] for lightweight API development and routing.

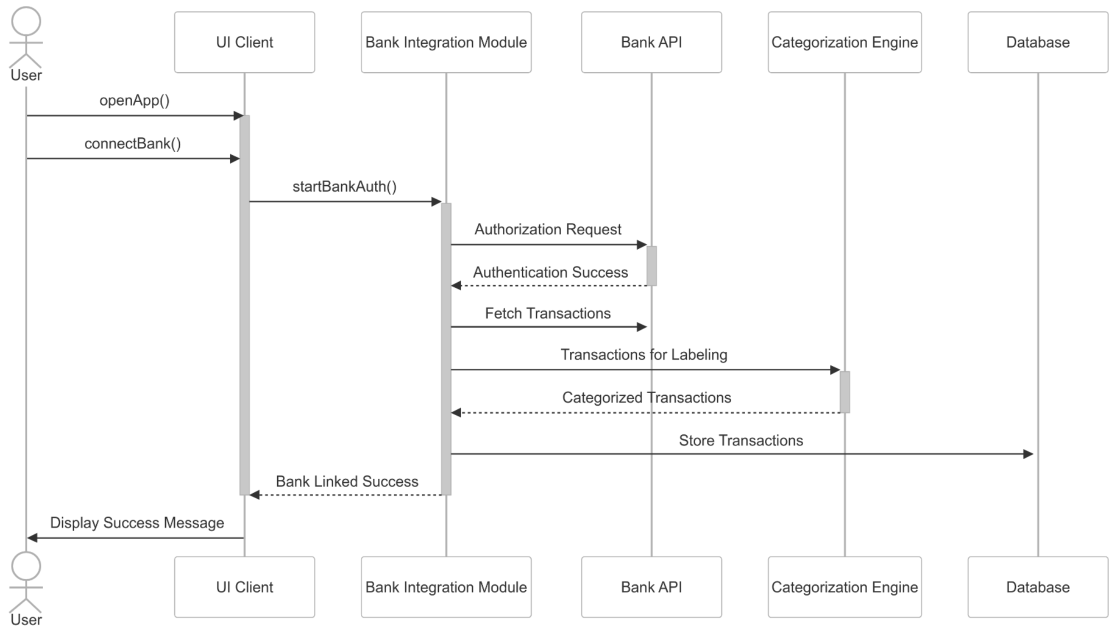
Database: [MySQL] for structured data persistence, Hosted on AWS RDS

Deployment: Backend Web Server Hosted on [AWS EC2]

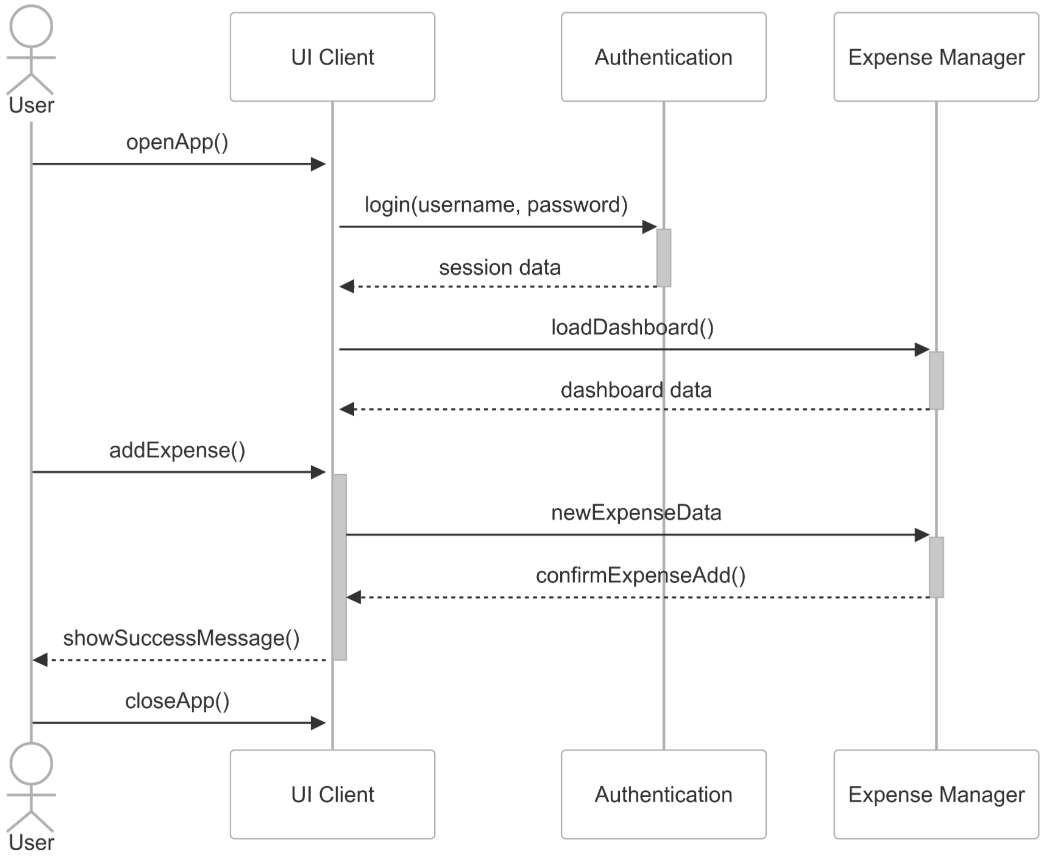
Frameworks were chosen for their simplicity, large community support, and compatibility with full-stack development using Python and JavaScript.

2. Functional Design

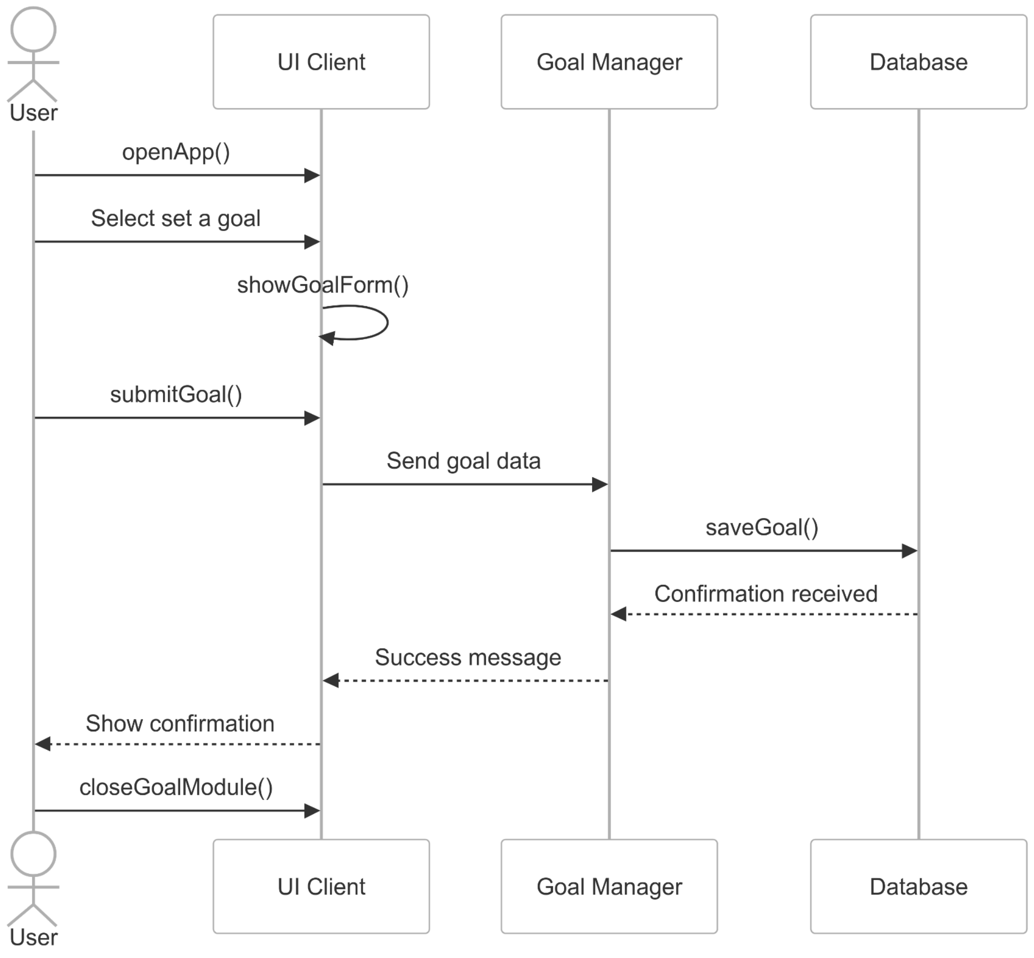
2.1 Link Bank Account



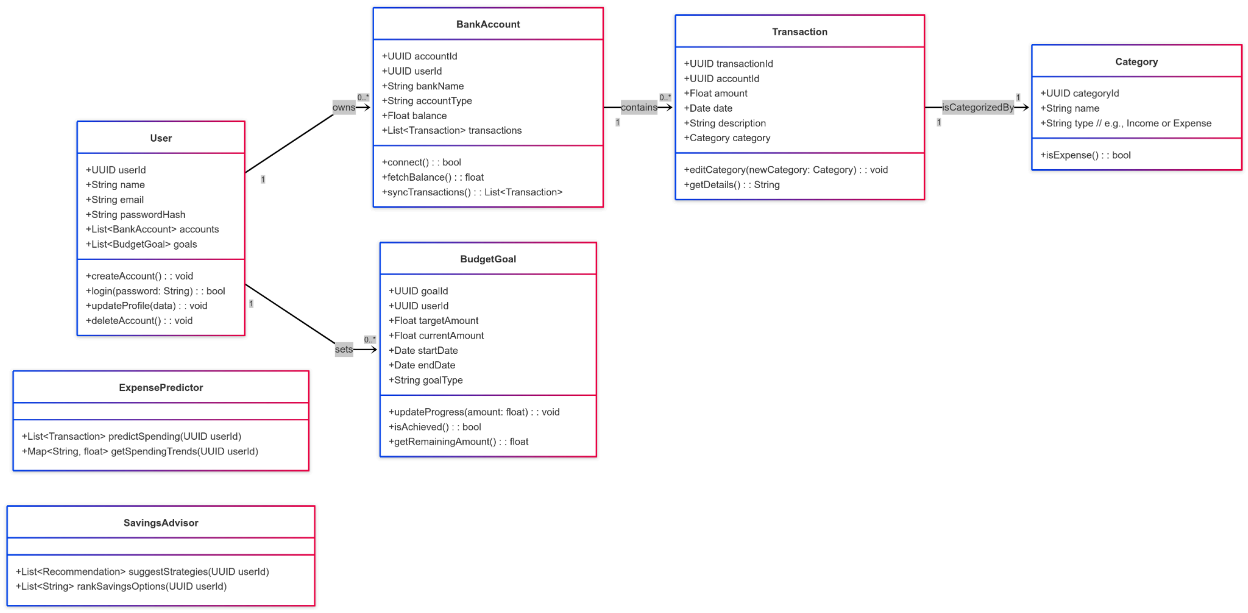
2.2 View Dashboard



2.3 Set Budget Goal



3.Structural Design


BankAccount
+UUID accountId
+UUID userId
+String bankName
+String accountType
+Float balance
+List<Transaction> transactions
+connect() : : bool
+fetchBalance() : : float
+syncTransactions() : : List<Transaction>
Transaction
+UUID transactionId
+UUID accountId
+Float amount
+Date date
+String description
+Category category
+editCategory(newCategory: Category) : : void
+getDetails() : : String
Category
+UUID categoryId
+String name
+String type // e.g., Income or Expense
+isExpense() : : bool
owns
0..*
1
contains
0..*
1
isCategorizedBy
1
1
User
+UUID userId
+String name
+String email
+String passwordHash
+List<BankAccount> accounts
+List<BudgetGoal> goals
+createAccount() : : void
+login(password: String) : : bool
+updateProfile(data) : : void
+deleteAccount() : : void
BudgetGoal
+UUID goalId
+UUID userId
+Float targetAmount
+Float currentAmount
+Date startDate
+Date endDate
+String goalType
+updateProgress(amount: float) : : void
+isAchieved() : : bool
+getRemainingAmount() : : float
sets
0..*
1
ExpensePredictor
+List<Transaction> predictSpending(UUID userId)
+Map<String, float> getSpendingTrends(UUID userId)
SavingsAdvisor
+List<Recommendation> suggestStrategies(UUID userId)
+List<String> rankSavingsOptions(UUID userId)