Fintrack

Amrit Randev
Anushka Chokshi
Karthik Manishankar

Dec 14, 2023

CS157A

1. Project Introduction/Objective

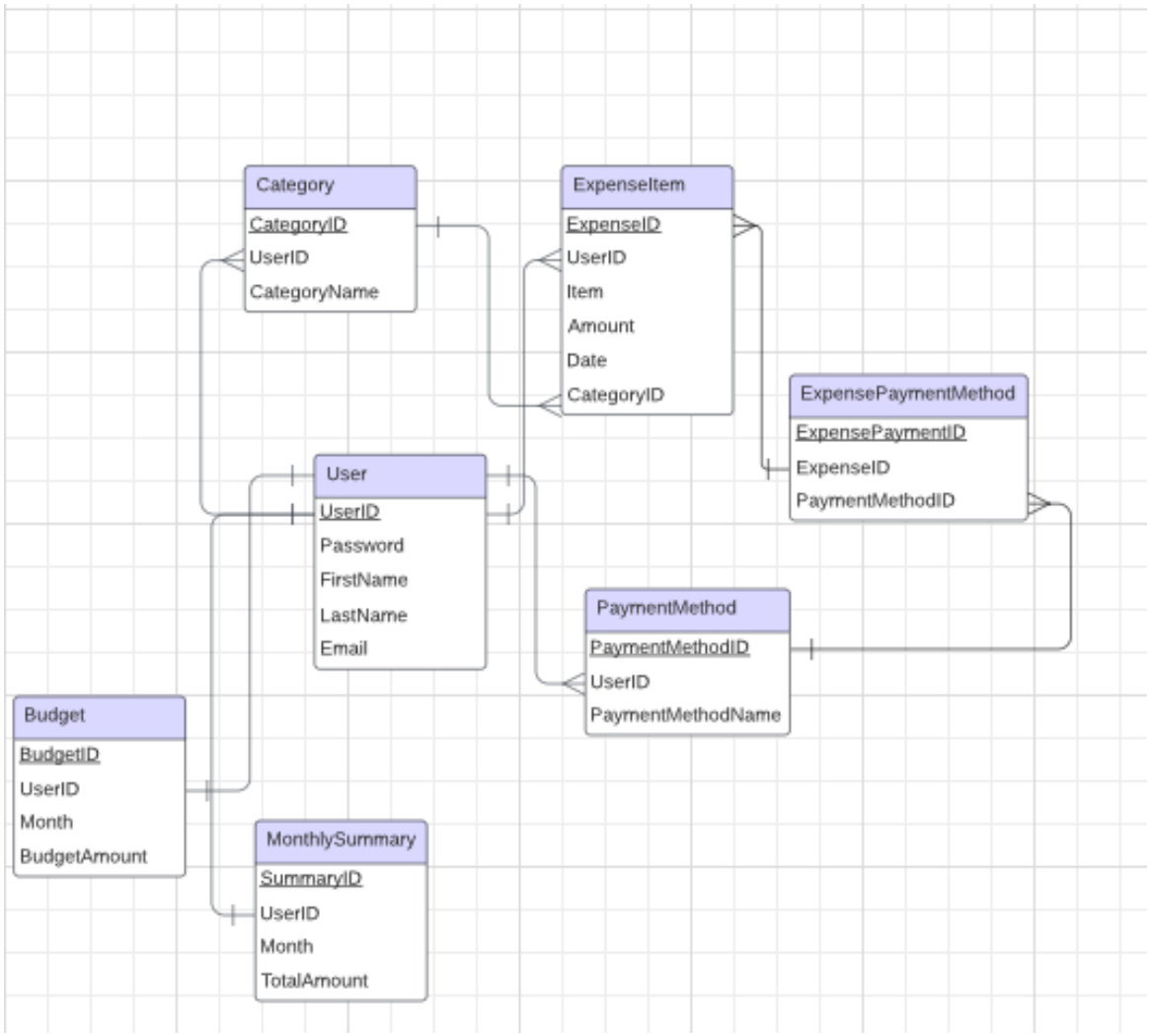
With consumerism at an all-time high, tracking and managing your finances is an essential life skill that individuals need. Seeing this issue, and how it affects our lives today, our group decided to create an application that tracks your expenses and allows you to see how much you have been spending and what categories your spending falls under. Our application includes multiple functionalities to achieve our objective of allowing users to track and manage their finances and spending. Users can log expenses including details such as the name of the item, the price when it was bought, and what category it falls under. Users are also provided with valuable insights into their spending patterns, allowing them to understand their expenses by category, time period, and more. This way they can have a sense of what they spend on, and what might be unnecessary moving forward. Users also can set budgets for different spending categories, and monitor their progress in each category. The system will be created in a way to notify the user when the budget is exceeded.

2. Project High-Level Design

The project was created using different technologies primarily utilized to develop rapid prototypes. As we faced difficulties in figuring out the best framework to choose for a finance tracking application, we realized that we needed to choose a lightweight framework that is both easily deployed and more easily accessible when paired with SQL. As a result, we chose Flask. Compared to Flask, our initial choice—Django—was a much more “heavy” framework. It offered plenty of toolings that we did not need at all for this application. Flask was just a much more lightweight application that offered exactly what we wanted! The frontend technologies we used for this project were HTML, CSS, JavaScript, and Bootstrap to both stylize our site and develop a fully functional user interface that is modern, intuitive, and aesthetically pleasing to

potential users. Our backend technology utilizes Flask as the framework and SQLite as our database. We chose SQLite as it is portable and easy to use/set up—why it is better to use in our case. In addition, we didn't need to utilize an external server instance to run our application and database at the same time. The primary drawback of SQLite was that there were some minor syntactical changes and that there were some limitations due to the overall structure of how SQLite was built. One of the libraries we used to develop the charts for users to better visualize their finances was Chart.js which gave us the ability to show overall spending per month for users and a user's spending on each category for the user.

3. Database Design (ER Diagram)



This ER diagram represents the relationships between the different tables. The relationships are as follows:

- User to ExpenseItem: A user can have multiple expense items (one user to many expense items)
- Category to Expense Item: A user can have multiple expense items in one category.
- ExpenseItem to ExpensePaymentMethod: There can be multiple expense items using the same payment method as an expense item can only be associated with one payment method.

- User to Budget: The user can create one budget (one user to one budget)
- User to MonthlySummary: Each user has a single monthly summary entry (one user to one monthly summary)
- User to Category: A user can create multiple categories (one user to multiple categories)
- User to PaymentMethod: A user can have multiple payment methods (one user to many payment methods)
- PaymentMethod to ExpensePaymentMethod: A payment method can be associated with multiple expense payments (one payment method to many expense payments)

Schema/Table Overview

Tables	Columns
User	UserID, Password, FirstName, LastName, Email
Category	CategoryID, UserID, CategoryName
ExpenseItem	ExpenseID, UserID, Item, Amount, Date, CategoryID
ExpensePaymentMethod	ExpensePaymentID, ExpenseID, PaymentMethodID
PaymentMethod	PaymentMethodID, UserID, PaymentMethodName
Budget	BudgetID, UserID, Month, BudgetAmount
MonthlySummary	SummaryID, UserID, Month, TotalAmount

The underlined represent primary keys and highlighted in blue are the foreign keys for the tables.

4. Normalization of tables

The schema adequately satisfies critical normalization requirements for efficient data storage. Analyzing the base elements, all tables suitably meet First Normal Form standards. The fundamental attributes within entities such as User and ExpenseItem hold atomic values without

composite duplicated data pieces. Plus each core construct has specialized primary keys uniquely identifying distinct rows.

Additionally, we can analyze the dependency for the Second Normal Form. The non-prime columns exclusively rely on complete primary key columns, rather than subsets of composite keys. For instance, the Category table has the single column CategoryID sufficiently determining the CategoryName values. This clear separation avoids partial dependencies which go against 2NF guidelines.

Finally, in all tables, non-key attributes only connect directly to primary identifiers. For example, MonthlySummary's TotalAmount ties directly to its primary key composite columns UserID and Month. The underlying data framework steers clear of transitive cascading dependencies – fully satisfying the third normal form's standards.

In summary, the defined schema for this system leverages the full normalization benefits of 1NF, 2NF, and 3NF.

1. **User**

   1NF - Contains atomic values for each attribute (UserID, FirstName etc.), has unique primary key UserID to identify rows.

   2NF - Non-prime attributes like FirstName, Email fully dependent on primary key UserID

   3NF - No transitive dependencies, all non-key attributes relate directly to primary key UserID

2. **Category**

   1NF - Atomic attributes CategoryID, CategoryName. CategoryID is primary key.

   2NF - Non-key CategoryName relies directly on primary key Category ID

3NF - No transitive dependencies as CategoryName ties directly to CategoryID

3. **ExpenseItem**

   1NF - Attributes are atomic values. Composite key of UserID and ExpenseID identifies rows.

   2NF - Non-key attributes like Date, Amount depend on full composite key

   3NF - No transitive dependencies, all attributes relate directly to entire primary key

4. **ExpensePaymentMethod**

   1NF - Atomic attributes for keys ExpensePaymentID, ExpenseID and PaymentMethodID

   2NF - Non-prime attributes directly linked to full primary key

   3NF - No indirect relationships, primary key determines values

5. **PaymentMethod**

   1NF - Simple atomic attributes PaymentMethodID, PaymentMethodName

   2NF - PaymentMethodName relies solely on full primary key ID

   3NF - Direct 1:1 mapping between all columns

6. **Budget**

   1NF - Atomic attributes BudgetID, Month, BudgetAmount. Unique rows.

   2NF - Non-keys depend only on full BudgetID primary key

   3NF - No transitive paths, direct relations between columns

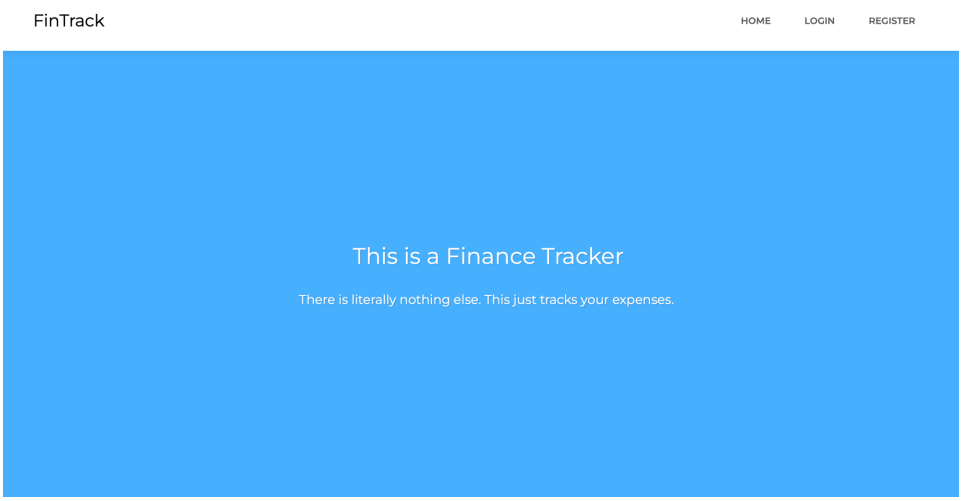
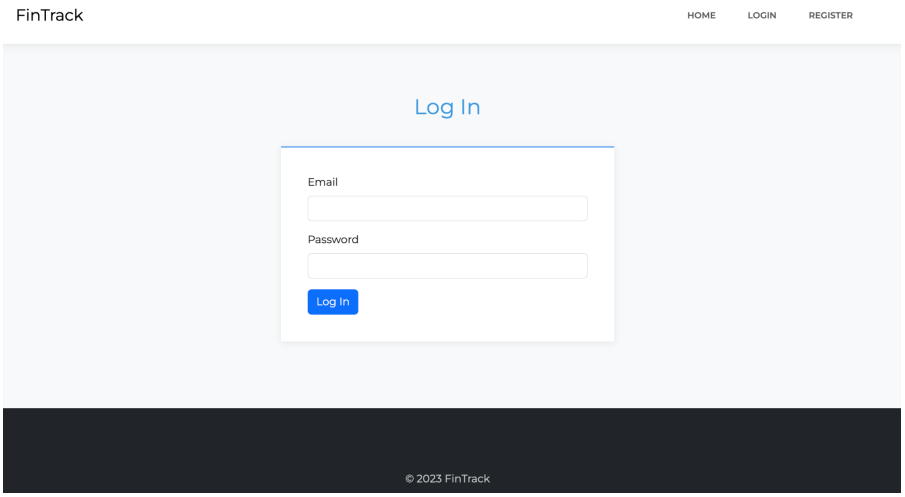
7. **MonthlySummary**

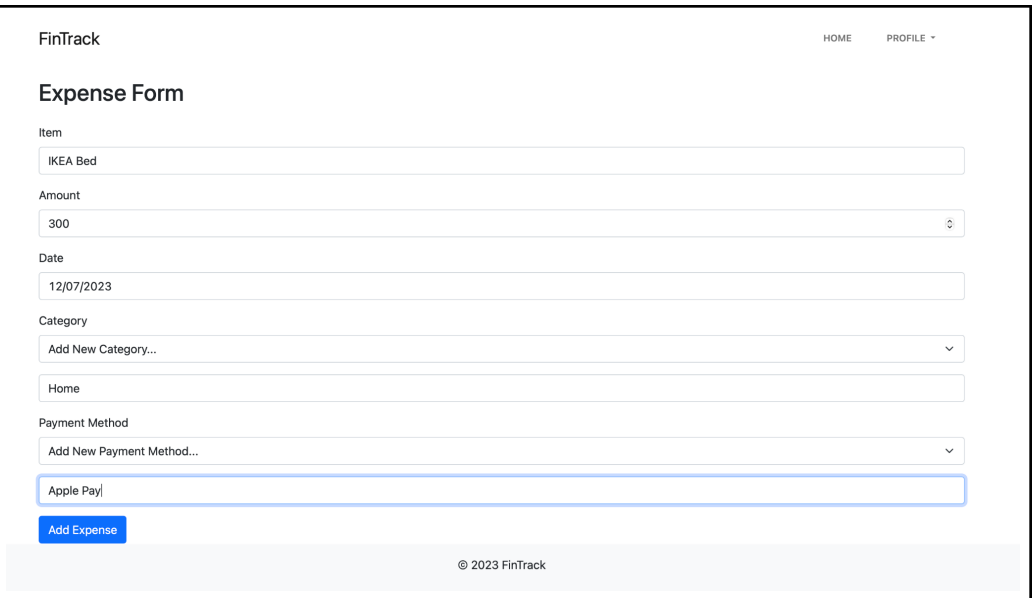
   1NF - Atomic UserID, Month, TotalAmount attributes identify each summary

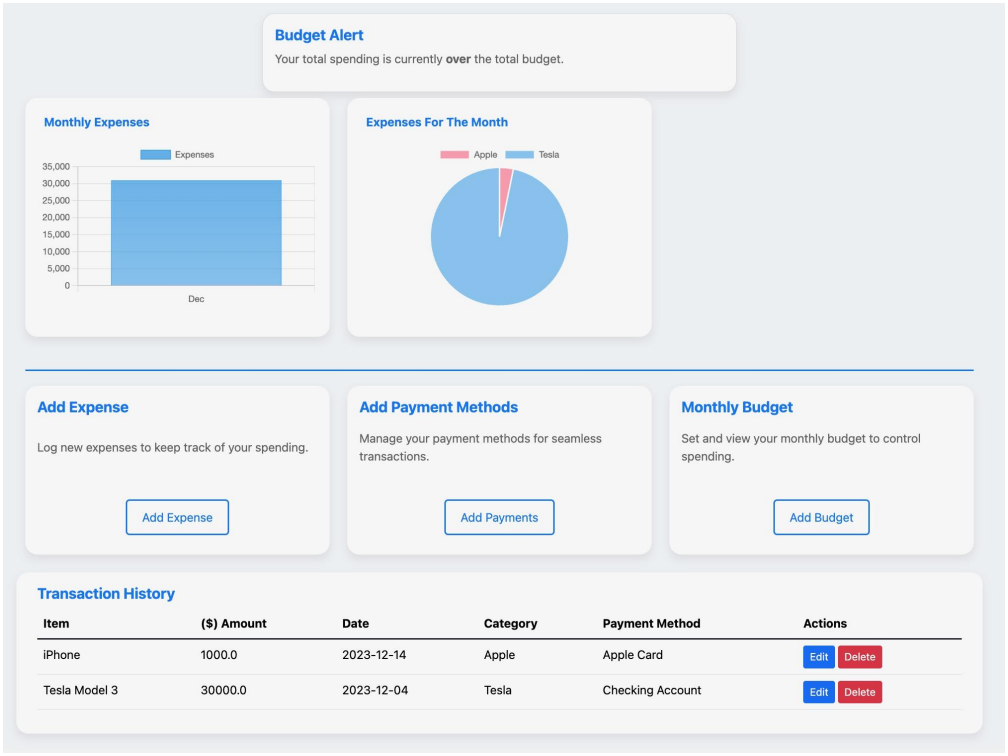
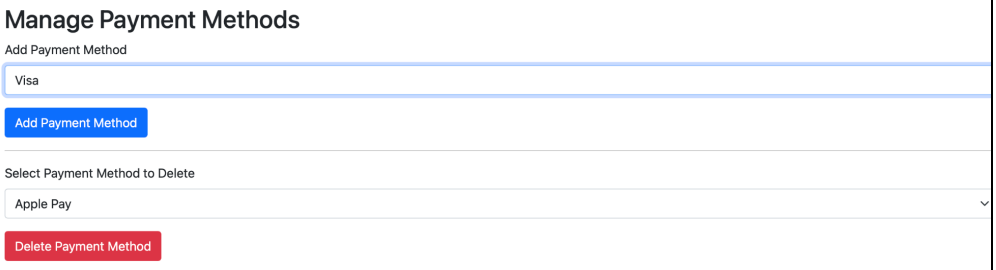
   2NF - TotalAmount ties directly to full composite primary key

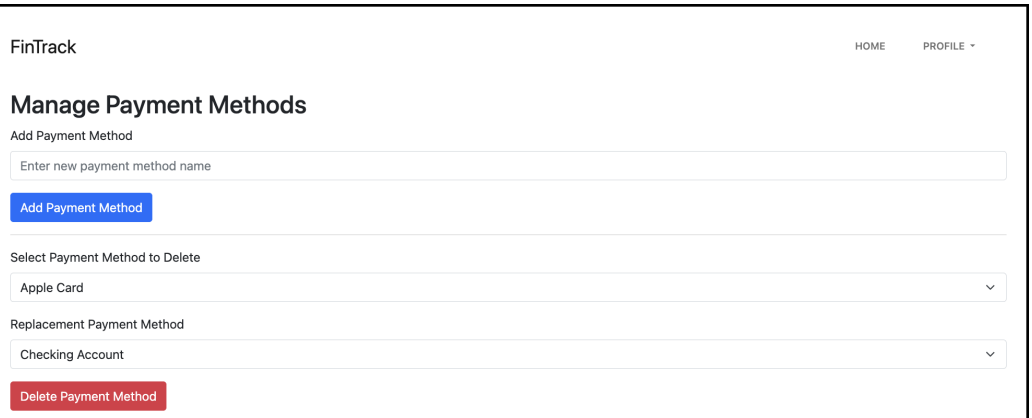
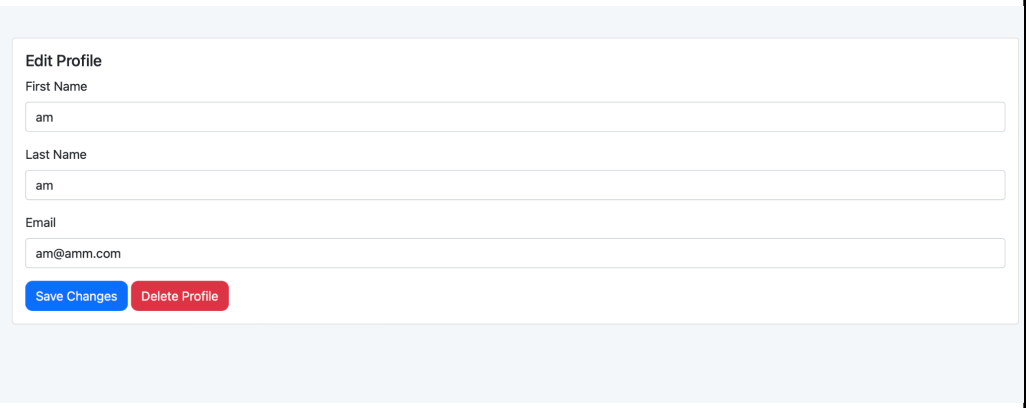
   3NF - Only direct column dependencies to the primary key

5. Application Screenshots

Home Page	
Login Page	

Expense Form	
Expense Modification Form	

Main Page (after adding some expenses)	
Manage Payment Methods Page where users can create and delete payment methods.	

Manage Payment Methods page where users can modify payment methods if another payment method already exists	
Monthly Budget Form where users can create and set a monthly budget.	
Edit Profile page where users can update their profile information, make changes as needed or delete their account.	

6. Individual Contributions

***Karthik**:* Set up project, designed + developed UI through HTML/JS/CSS/BS5 + modified to work with Jinja templating, implemented authentication for registration, login/logout with SHA256 encryption/decryption, implemented schema and integrated database functionality to allow for usage within application, landing pages, dashboard, expense addition, expense modification/deletion, transaction history list UI + buttons to delete and edit expenses, budget alert, payment method modification so that users can modify previously existing transactions and change the card they used, budget functionality, expense for the month chart, profile and expense deletion, session cookie implementation to ensure that users are logged in for certain pages, profile settings where users can change details such as first name, last name, and email, helped on presentation/report, and fixed bugs (Github: @Kardee12)

***Anushka**:* Payment method functionally implemented, monthly budget functionality implemented, transaction history table implemented at bottom of the main page to showcase expenses, created and updated the readme to allow outsiders to clone and test our project, budget functionality implemented, fixed bugs in features such as transaction history, helped on presentation/report, and budget alert added, worked on slides/report (GitHub handles: @anushkac1, @AnushkaChokshi)

***Amrit**:* Helped debug the application such as budget functionality, monthly budget functionality, and charts, and used charting libraries to create charts to visualize data. Created presentation and report material including diagrams and more. (GitHub @amrandev).

7. Result

As a result, we created FinTrack, our comprehensive finance tracking application aimed at improving our users' lives through smart financial tracking. Through our use of modeling, high-level design, schema implementation, and normalization, we created features to add payments, set and alter budgets, add expenses, and view the results of all the transactions. We made use of languages like Flask and SQLite to meet our goal of creating an efficient and accessible finance-tracking application.

8. References/Sources

- Db.py = https://flask.palletsprojects.com/en/1.1.x/tutorial/database/ - cited in code
- Class lectures