Bharati Vidyapeeth Deemed to be University
Institute of Management & Entrepreneurship
Development, Pune

A PROJECT REPORT ON

”PERSONAL FINANCE MANAGEMENT SYSTEM”

IN THE PARTIAL FULFILLMENT OF THE REQUIREMENT FOR DEGREE OF
BACHELOR OF COMPUTER APPLICATION

(SEMESTER VI)

SUBMITTED BY

SANJAY ASHOK ROUT (RN. 05)

SUBMITTED TO

Dr. Ramchandran Mahadik	**Dr. Swati Desai**	**Ms. Deepti Deshmukh**
VP- IMED & PD-BCA	**Class Coordinator**	**Project Guide**

Dr. Ajit More
I/C Director, IMED Pune

BHARATI VIDYAPEETH (DEEMED to be UNIVERSITY)

INSTITUTE OF MANAGEMENT AND ENTREPRENEURSHIP DEVELOPMENT, PUNE

Bharati Vidyapeeth Deemed to be University
Institute of Management & Entrepreneurship Development, Pune

Certificate of Completion

This is to certify that **Sanjay Ashok Rout** is bona-fide student studying in Semester 6th, Bachelor of Computer Application (BCA) degree, program of Bharati Vidyapeeth (Deemed to be University), Pune for the Year 2023-24. As a part of University curriculum**, Sanjay Ashok Rout** has successfully completed a Major software project title as

"PERSONAL FINANCE MANAGEMENT SYSTEM "

Under the guidance of **Ms. Deepti Deshmukh**
This fulfills the requirement of university course
Of BCA Semester 6th.

Ms. Deepti Deshmukh
Project Guide

Dr. Swati Desai
Class Coordinator

Dr. R.V. Mahadik
VP- IMED & PD-BCA

Dr. Ajit More
I/C Director, IMED Pune

INTERNAL EXAMINER

EXTERNAL EXAMINER

Bharati Vidyapeeth Deemed to be University
Institute of Management & Entrepreneurship Development, Pune

DECLARATION

I undersigned hereby declare that the project report entitled is developed and submitted by me for the partial fulfillment of the Bachelor of Computer Application **(BCA-VI)** semester under the guidance of **Ms. Deepti Deshmukh.**

All the material obtained for this project report is exclusively based on information collected by me during the system study and has not formed part of any other report previously.

Student Signature
Student name: Sanjay Ashok Rout
Roll No. 5
Date: April 15th, 2024

ACKNOWLEDGEMENT

Making the following acknowledgement is not a mere formality but a way to express my sincere thanks to the people who were instrumental in presenting the project report.

Firstly, I express my boundless sense of gratitude to **Ms. Deepti Deshmukh** for allowing me to work on this project.

I express my deep sense of gratitude to **Ms. Deepti Deshmukh** for her keen interest and efficient guidance in project work. She is the person conceived the idea of the project, and inspiring to complete this task, without which this could not have seen in the light of the day.

Cordial thanks and heartfelt obligations to **Ms. Deepti Deshmukh** guidance and valuable contribution during project work.

I would heartily extend my gratefulness to all who helped me in completing this project.

Student Name & Sign:

Sanjay Ashok Rout

Abstract

Personal finance management system is a single-user financial records keeping system. It encourages you to contain your overspending habits. First, you have to create an estimated spending amount called a budget for a definite term. After that, it assists you to keep track of how much you have earned and spent within that budget. You can get detailed reports via GUI charts powered by Chart JS.

The application features a robust dashboard that displays an overview of the user's financial health, including recent transactions, upcoming bills, and budget summaries. Users can add and categorize transactions, set financial goals, and create budget plans to ensure they remain within their spending limits. The system also includes reporting tools that generate detailed reports and insights, helping users understand their spending patterns and make informed financial decisions.

Developed using Laravel, a PHP framework known for its reliability and scalability, the Personal Finance System emphasizes security and user-friendly interfaces. It is designed to be responsive, ensuring compatibility with various devices, thereby enabling users to manage their finances on-the-go.

This project aims to empower users by simplifying personal finance management, promoting financial literacy, and assisting in achieving financial stability and growth through better budgeting and expenditure tracking.

INDEX

Introduction

Personal finance is a single-user financial records keeping system. It encourages you to maintain your overspending habits. First, you have to create an estimated spending amount called a budget for a definite term. After that, it assists you to keep track of how much you have earned and spent within that budget. You can get detailed reports via GUI charts powered by Chart JS.

Purpose
The core purpose of the Personal Finance System is to provide users with an accessible, easy-to-use platform to track their expenses, income, savings, and investments. The system helps users avoid common financial pitfalls such as overspending, under-saving, and inefficient financial planning. By integrating all aspects of personal finance into one interface, it ensures that users can see the complete picture of their financial health at any time.

Feature

a. Dashboard Overview
b. Transaction Management
c. Budget Planning
d. Financial Goals
e. Reports and Analytics

Target Audience
The Personal Finance System is ideal for tech-savvy individual looking to enhance their financial understanding and control. It is particularly valuable for those who are beginning to take serious steps towards financial independence and want to cultivate good financial habits early on.

Objective

1. **Single-User Financial Record Keeping**: To provide a secure and private platform for individual users to maintain and manage their financial records efficiently. This system serves as a personal financial assistant that simplifies the complexities involved in daily financial management.
2. **Budget Tracking**: To enable users to establish a budget on the basis of goal set by them and track their spending against this budget.
3. **Categorization**: To facilitate detailed tracking of income and expenditures by categorizing them into predefined or custom categories. This categorization helps users analyze their income and spending patterns across different areas such as groceries, entertainment, utilities, and more, making it easier to identify areas for cost savings.
4. **Investment Analysis**: To provide charts by the 'Chart JS' that analyze and report on the users' income and expenditure.
5. **Financial Goal Setting**: To assist users in setting and monitoring financial goals such as saving for a home, planning for retirement, or funding education. The system tracks progress towards these goals and provides visual feedback through charts and graphs, offering motivation and guidance to achieve financial aspirations.
6. **Interactive Reporting and Visualization**: Utilize Chart.js to offer interactive and intuitive graphical representations of financial data. This visual approach aids users in quickly understanding their financial status, analyzing trends, and adjusting their financial strategies based on real-time data.

Scope

1. **Responsive Web Application**: Deliver a user-friendly web application accessible on multiple devices, providing a seamless user experience for managing personal finances.
2. **Comprehensive Dashboard**: Implement a central dashboard featuring key financial indicators, recent transactions, and quick links to major actions, enabling easy navigation and immediate insights into financial health.
3. **Transaction Management**: Allow users to record, categorize, and manage their income and expenses with options to edit and delete entries as necessary.
4. **Interactive Reporting**: Utilize Chart.js to generate interactive and dynamic charts for financial data visualization, aiding in better understanding and decision-making.
5. **Maintenance and Updates**: Establish a framework for easy updates and maintenance of the system, allowing for new features and improvements without significant downtime.
6. **Third-Party Integration**: Plan for future integration with financial institutions and other relevant third-party services to expand functionality.

Problem Statement

Managing personal finances effectively remains a challenge for many due to the absence of user-friendly tools that consolidate all aspects of financial management. Individuals often struggle with maintaining accurate records of spending and income, leading to poor budget adherence and financial planning. Most available solutions are either too complex or lack comprehensive features such as real-time budget tracking, financial goal setting, and insightful analysis of spending habits. This gap highlights the need for an integrated system that simplifies personal finance through an intuitive interface, making it accessible to everyday users. The "Personal Finance Management System" is designed to fill this void by offering a simple yet powerful tool to track, manage, and analyze personal finances, promoting better financial decisions and healthier financial habits.

Keywords

Technical keywords

PHP, Laravel, MySQL, JavaScript, Chart.js, HTML5, CSS3, Responsive Design, Web Application, JSON, REST API, Secure Authentication, Git, Composer, npm, XAMPP, Apache Server, Linux Server, User Interface (UI), User Experience (UX), Front-end Development, Back-end Development

Non-technical keywords

Budget tracking, expense categorization, financial analysis, and financial goal setting

System Overview

Architecture:

- **Front-End**: The client-side of the system utilizes HTML5, CSS3, and JavaScript to deliver a responsive and interactive user interface. Chart.js is incorporated to provide dynamic, visually appealing charts for financial data visualization. Bootstrap framework ensures a consistent, responsive design across various devices and screen sizes.
- **Back-End**: The server-side is powered by the Laravel framework, known for its elegant syntax and robust features. PHP is used for server-side scripting, handling business logic, and managing database operations. Laravel's MVC architecture helps in organizing the codebase and enhances maintainability.
- **Database**: MySQL serves as the relational database management system, storing all user data securely. This includes user profiles, transaction records, budget details, and financial goals. The database schema is designed to support complex queries for reporting and analytics effectively.

Component:

a. Dashboard
    a. Home
    b. Balances
b. Budget
    a. Active Budget
    b. Create a new one
    c. View all budgets
c. Category
    a. Create a new one
    b. View all categories
d. Payment Options
    a. Create a new one
    b. View Payment options
e. Transactions
    a. Create a new one
    b. View transactions

Hardware and software used

Hardware used:

Processor:	Intel(R) Core (TM) i7-6700HQ CPU @ 2.60GHz 2.60 GHz
Speed:	2.60 GHz
RAM:	16 GB
Storage:	118 SSD and 1TB HDD
Network:	Home wireless network through router

Software used:

Operating System:	Windows 10
Front End:	HTML5, CSS3, Bootstrap, Chart.js
Back End:	PHP
Database:	MySQL
Programming Language:	PHP, JavaScript
Framework:	Laravel
Project Type:	Web Application
IDE:	Visual Studio Code (latest version)

- **Development Tools:** Composer (dependency management), npm (JavaScript package management), Git (version control), XAMPP (local server setup), drawio

Libraries and Frameworks

- Accounting js for formatting numbers/amounts _v0.4.2 public\vendor\js\accounting.min.js_
- Bootstrap CSS Framework _v4.6.0 CDN_
- Chart js _v3.5.1 CDN_
- Font-awesome _v4.7.0 CDN_
- jQuery JavaScript Library _v3.6.0 CDN_
- Laravel PHP Framework _v8.61.0_
- Particles js for snowfall effect in the login page _v2.0.0 public\vendor\js\particles.min.js with particles.settings.js and settings.json_
- Popperjs _v2.10.1 CDN_
- Sass to process scss _resources\sass\dev.scss_

Dependencies

- PHP Version: The project requires PHP version 7.3 or higher, including all versions of PHP 8.
- Laravel Packages:
  - laravel/framework: The core framework itself, version 8.54 or later.
  - laravel/sanctum: Provides a lightweight authentication system for SPAs and simple APIs.
  - laravel/tinker: Allows interaction with the entire Laravel application from the command line.
  - Other Dependencies:
  - guzzlehttp/guzzle: A PHP HTTP client that makes it easy to send HTTP requests and trivial to integrate with web services.
  - fruitcake/laravel-cors: Handles Cross-Origin Resource Sharing (CORS) capabilities in Laravel applications.

Font

- Poppins _google fonts_

Steps followed to create the software file:

Each step provided here corresponds closely to typical tasks involved in setting up a Laravel and Node.js environment. This combination is usually employed in full-stack web applications where Laravel handles backend logic and API responses, while Node.js processes frontend assets.

a. composer create-project --prefer-dist laravel/laravel Personal_Finance_Managemet_System
b. cd Personal_Finance_Managemet_System
c. Database setup
    - DB_CONNECTION=mysql
    - DB_HOST=127.0.0.1
    - DB_PORT=3306
    - DB_DATABASE=personal_finance
    - DB_USERNAME=your_username
    - DB_PASSWORD=your_password
d. Run the migrations to create default tables:
    - php artisan migrate
e. Build the Application Structure
    - Model and Migration Creation
    - Controllers
    - Routes
f. Frontend with Blade
g. php artisan migrate
h. php artisan db:seed
i. php artisan serve
j. npm install & npm run dev

Usage of software technology in the project

Laravel

Laravel serves as the foundational framework, providing a robust structure for the application and offering a wide array of features that facilitate rapid development. Here's how Laravel is used in the project:

- **MVC Architecture**

  Laravel follows the Model-View-Controller (MVC) architecture which organizes the application into three interconnected components. This separation enhances manageability and scalability:

  - Models handle data interactions, representing the application's data structure and business logic.
  - Views present content in a user-friendly format.
  - Controllers process incoming requests, interact with models, and render views.
  - Eloquent ORM
  -

- **Blade Templating Engine**

  Blade is Laravel's powerful templating engine and allows you to define templates with plain PHP code, which is then cached until modified, optimizing performance. It includes convenient shortcuts for common PHP functions and constructs.

- **Artisan CLI**

  Laravel's command line interface (CLI), Artisan, provides a number of helpful commands for development, including:

  - Database migration management
  - Queue and job handling
  - Code generation for controllers, models, and other components

- **Database Migrations and Seeding**

  Laravel's migration system provides version control for database schemas, allowing you to define tables and columns in PHP. Seeders allow you to populate your tables with sample or initial data.

- **Routing**

  Laravel's routing allows you to define routes for your application in a simple and expressive manner. Routes direct incoming requests to a controller or a closure, specifying what logic should be executed.

- **Middleware**

  Middleware offers a convenient mechanism for inspecting and filtering HTTP requests entering your application. This could be used for tasks like logging, authentication, or caching.

- **Laravel Mix**

  While not a PHP feature, Laravel Mix is a web asset compilation tool that provides a fluent API for defining basic webpack build steps for your application, such as compiling SCSS or JavaScript files.

- **Package Ecosystem**

  Laravel has a rich package ecosystem with tools like Cashier for subscription billing, Socialite for OAuth authentication, Scout for full-text search, and many more which can be leveraged to extend the functionalities of your finance system.

PHP

PHP plays a pivotal role as the server-side scripting language. Laravel, the framework used in this project, is built on PHP, so the usage of PHP is integral throughout the system. Here's how PHP is utilized:

- **Core Application Logic**

  PHP is used to create the core business logic of the application. This includes processing financial data, handling user requests, performing calculations, and managing interactions with the database.

- **Handling HTTP Requests**

  PHP scripts receive HTTP requests from the client-side, process them according to the business logic, and send back appropriate HTTP responses.

- **Database Interactions**
  PHP, through Laravel's Eloquent ORM, interacts with the MySQL database. It performs CRUD (Create, Read, Update, Delete) operations on financial records like transactions, budgets, and user profiles.

- **Data Processing**
  PHP processes and manipulates financial data for various functionalities like budget tracking, expense categorization, and financial reporting.

- **Artisan Commands**
  PHP is the language used to write Artisan commands, which can automate tasks and provide utilities for database migration, queue management, and more.

- **Middleware**
  PHP is used to write middleware which filters and processes HTTP requests, such as verifying authentication, performing logging, or applying CORS headers.

- **Error and Exception Handling**
  PHP's built-in error handling is extended by Laravel to manage exceptions and present user-friendly error pages.

SQL

MySQL in the "Personal Finance Management System" is the relational database management system (RDBMS) that stores and manages all the data crucial to the application. Its usage can be described in the following aspects:

- **Data Storage**
  MySQL serves as the primary data repository for the application. It stores various types of data, including:

  - **User Information**: Secure storage of user profiles, including authentication details.
  - **Financial Records**: Detailed transaction data, including dates, amounts, categories, and descriptions.
  - **Budget Plans**: User-defined budget information for various categories and time frames.

- **Data Retrieval**

The application queries MySQL to retrieve:

- **User Data**: For login verification and user profile display.
- **Transaction History**: To list and summarize past financial activities.
- **Budget Status**: To check current spending against budget limits.

- **Data Manipulation**

MySQL is used to perform CRUD (Create, Read, Update, Delete) operations such as:

- **Creating Records**: Adding new transactions, setting up budgets, or registering a new user.
- **Reading Records**: Fetching data to be displayed on the dashboard or in reports.
- **Updating Records**: Modifying existing transactions, adjusting budget settings, or updating user profiles.
- **Deleting Records**: Removing outdated or incorrect entries.

- **Transaction Management**

  MySQL manages transactions to ensure data integrity, particularly in operations that involve multiple steps or updates, which is crucial for financial applications to prevent data corruption or loss.

JavaScript

JavaScript plays a crucial role in the "Personal Finance Management System," especially on the client side to enhance user interaction and experience. Here's how JavaScript is utilized in the project:

- **Dynamic Content Updates**

  JavaScript manipulates the DOM (Document Object Model) to update the content displayed to the user without needing to reload the entire page. This is particularly useful for updating financial data in real-time as users enter transactions, modify budgets, or interact with the application.

- **Interactive UI Components**
  JavaScript powers interactive UI components such as modal windows, drop-down menus, and form validation feedback, which are integral to a seamless user interface and better user engagement.

- **Charting and Visualization**
  Using libraries like Chart.js, JavaScript renders complex financial data into charts and graphs, making it easier for users to understand their financial status, spending habits, and trends.

- **Event Handling**
  JavaScript responds to user actions like clicks, form submissions, and keyboard events to handle things like opening a detailed view of transactions, adding new records, or navigating through the application.

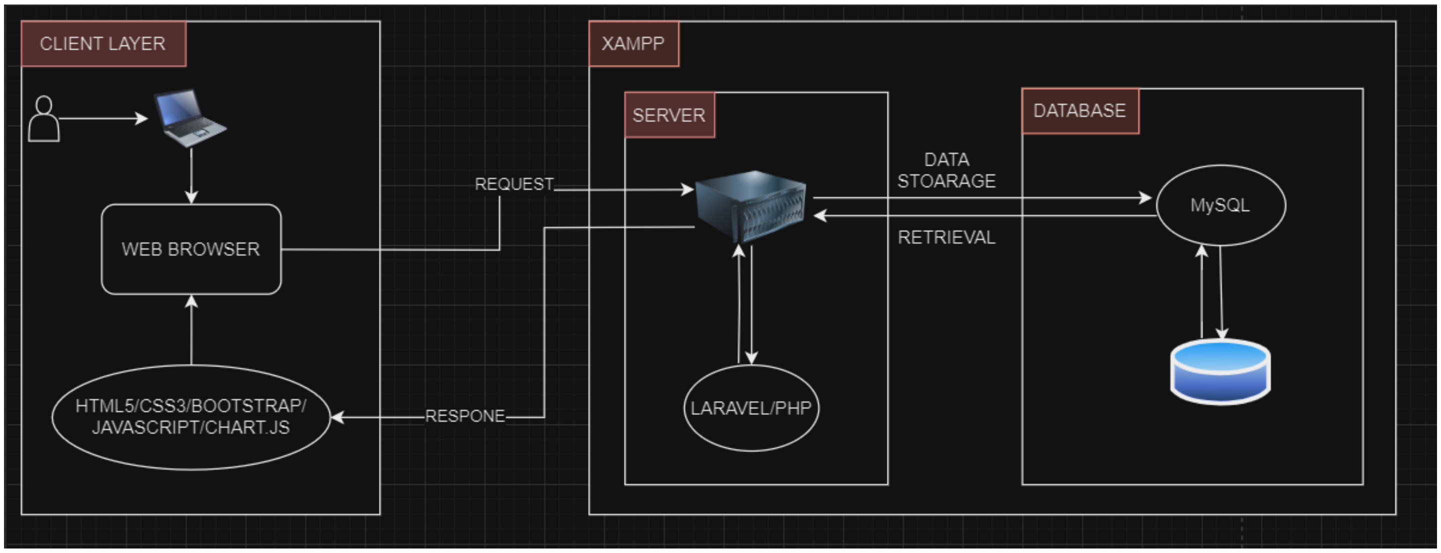
- **Responsive Design**
  In conjunction with CSS, JavaScript ensures the application's interface is responsive and functions well on different devices and screen sizes.

- **Single Page Application (SPA) Features**
  If parts of the application are designed as an SPA, JavaScript manages routing, content replacement, and state management without full page refreshes, leading to a smooth and app-like experience.

Diagram

System Architecture Diagram:

This diagram will illustrate the overall system architecture, showing how different components such as the client-side, server-side, and database interact with each other



[link]

Explanation:

- **Client Layer**
  Represents the point of interaction for the users of the Personal Finance Management System.

  - Web Browser: The medium through which the users access the system.
  - Technologies Used: The front-end technologies (HTML5/CSS3/Bootstrap/JavaScript/Chart.js) denote the stack used to create the user interface. Bootstrap is utilized for responsive design, JavaScript for dynamic interactions, and Chart.js for graphical data representation.

- **Server**
  - Encapsulated within the XAMPP environment, which is a popular PHP development platform that includes Apache server, MySQL database, and PHP.
  - Laravel/PHP: The server is running the Laravel framework with PHP as the server-side scripting language. Laravel handles the application logic, routing, and

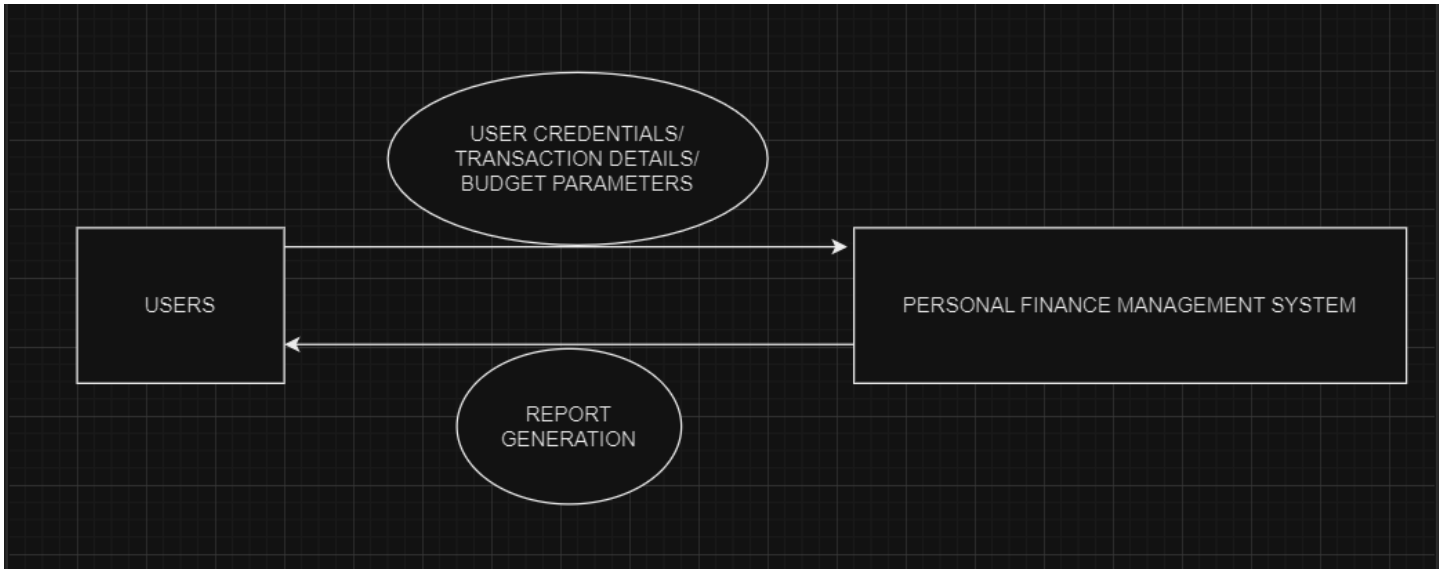
interaction with the database.

- **Database**
  - MySQL: The database component where all data related to the application is stored. It includes tables for user data, transactions, budgets, etc.
  - Data Storage: Refers to the action of storing data in the database, such as new transactions or updated budget details.
  - Retrieval: Refers to querying the database to fetch data for various operations, like generating reports or displaying transaction history.

- **Data Flow**
  - Request: When a user performs an action in the web browser, such as logging in or submitting a new transaction, a request is sent to the server.
  - Response: After the server processes the request (e.g., checks login credentials or adds a transaction to the database), it sends back a response to the web browser, which might be a new page, the results of a transaction entry, or an error message if something goes wrong.

DATA FLOW DIAGRAM

➢ **DFD LEVEL 0: CONTEXT DIAGRAM**



LINK

Explanation:

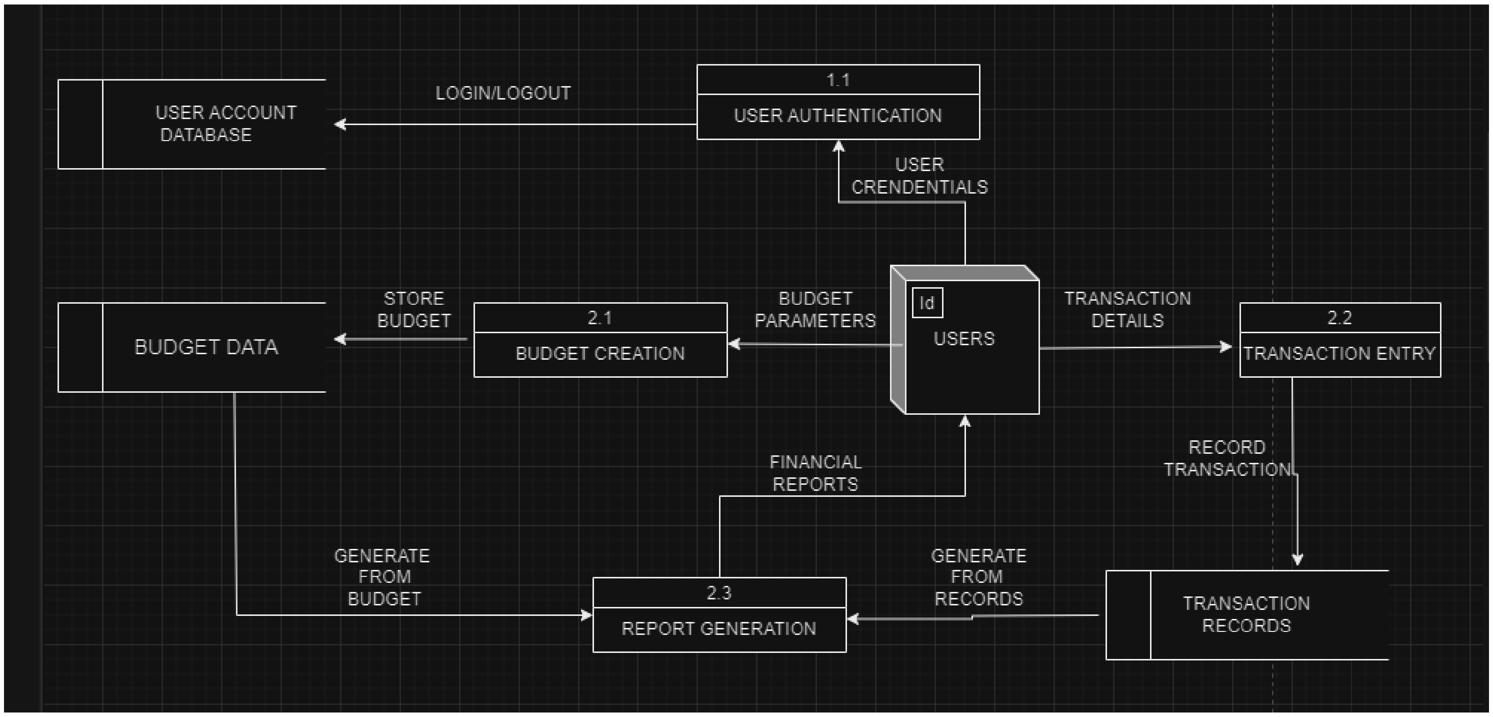
- **Users (External Entity)**:
  This represents the users of the Personal Finance Management System. They are the primary actors who interact with the system.

- **User Credentials / Transaction Details / Budget Parameters (Data Flow)**:
  These are the types of data that users input into the system. User credentials are used for logging in and authentication, transaction details are for recording financial transactions, and budget parameters are for setting up and managing budgets.

- **Personal Finance Management System (System Boundary)**:
  This is the main system where all the internal processes occur. It takes input from users, processes it, and outputs information back to the users. The system is responsible for managing all aspects of personal finance according to the user inputs it receives.

- **Report Generation (Data Flow)**:
  This represents the output of the system. Once the system processes the data input by the users, it can generate various types of reports. These reports might include financial summaries, budget reports, transaction histories, and investment analyses.

➢ DFD Level 1 Diagram



LINK

Explanation:

The final Data Flow Diagram (DFD) for your "Personal Finance Management System" outlines how data moves between processes, data stores, and external entities within the system. Here's an explanation based on the diagram:

External Entities

USERS: This represents the individuals who will be interacting with the system, providing input such as user credentials and transaction details, and receiving financial reports.

Processes

- **User Authentication:** Validates user credentials to grant access to the system.
- **Budget Creation:** Users input budget parameters to set up their budget.
- **Transaction Entry:** Records transaction details provided by the user.
- **Report Generation:** Compiles data into financial reports for the user.

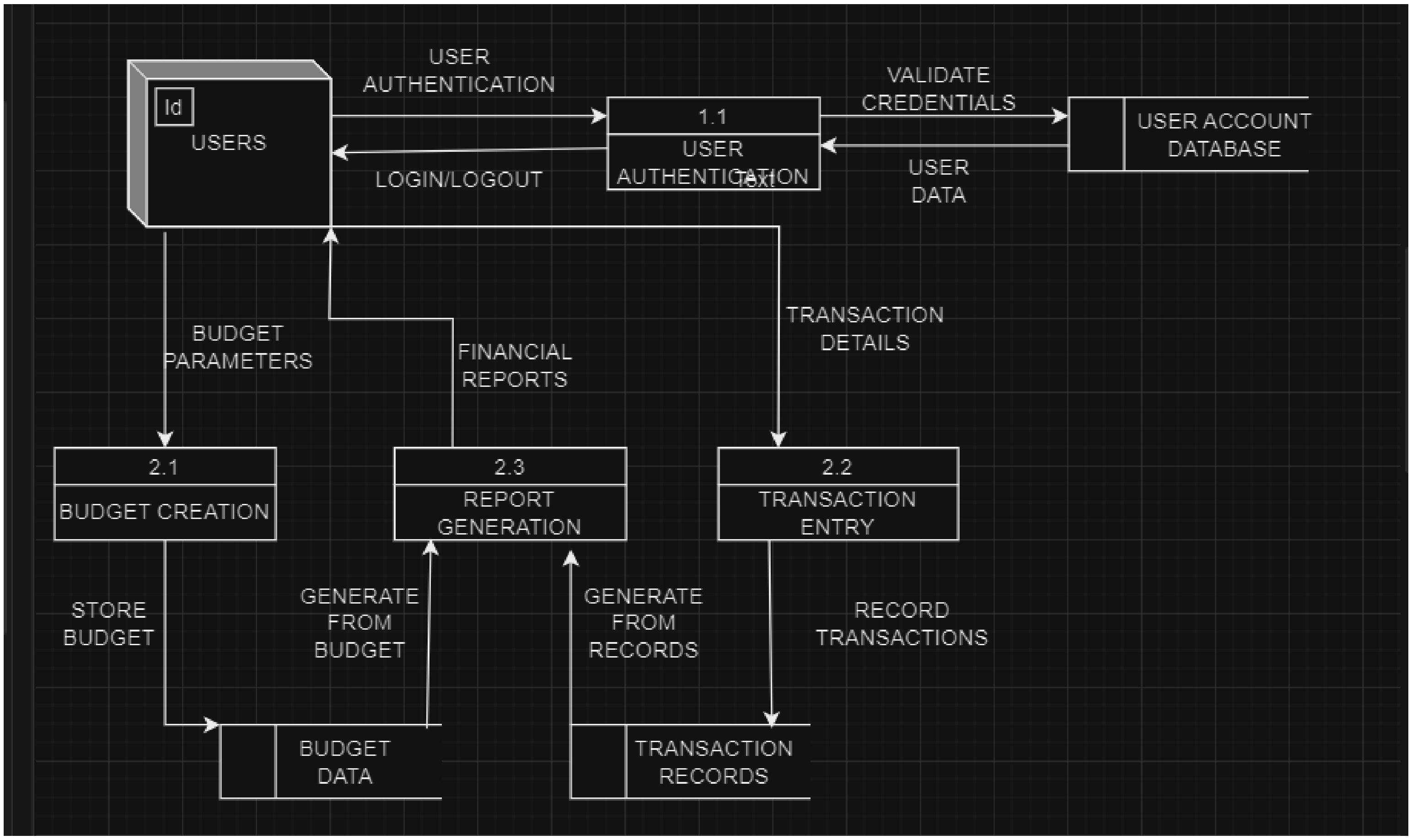
Data Stores

- **User Account Database:** Contains user credentials and profile information.
- **Budget Data:** Stores the details of the users' budgets.
- **Transaction Records:** Maintains a record of all user transactions.

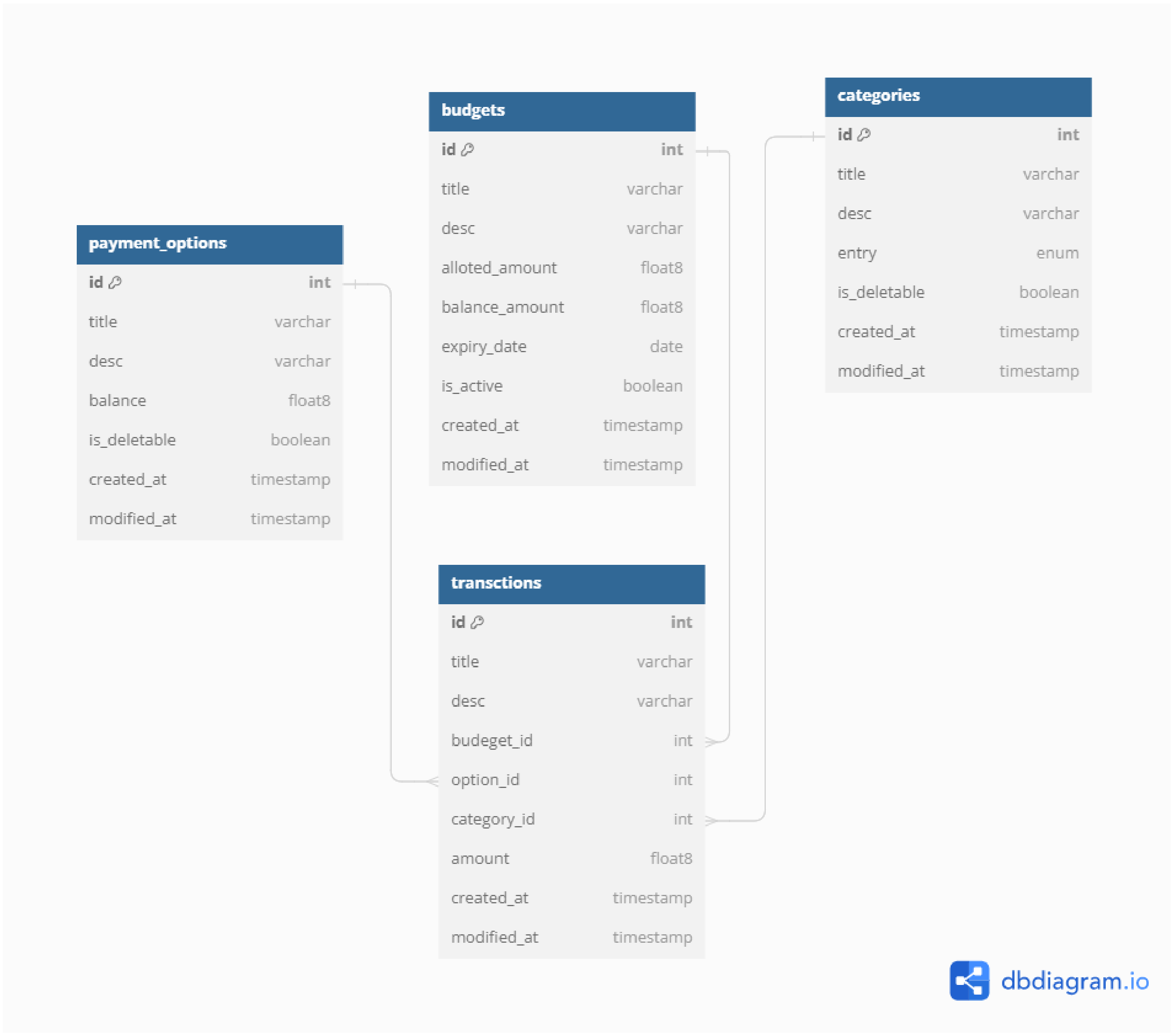
Data Flows

- **User Credentials:** Data input by users to log in or out of their accounts.
- **Budget Parameters:** Information users provide to create and define their budgets.
- **Transaction Details:** Specific details about each financial transaction a user enters.
- **Login/Logout:** Represents the action of a user signing in or out of the system.
- **Store Budget:** The action of saving the budget data to the Budget Data store.
- **Record Transaction:** The action of saving transaction details to the Transaction Records store.
- **Generate from Budget:** Utilizes stored budget data to contribute to the report.
- **Generate from Records:** Utilizes stored transaction records to contribute to the report.
- **Financial Reports:** The output data consisting of processed and compiled information provided to the users.

- **DFD Level 2 Diagram**



DATABASE SCHEMA



budgets	
id	int
title	varchar
desc	varchar
alloted_amount	float8
balance_amount	float8
expiry_date	date
is_active	boolean
created_at	timestamp
modified_at	timestamp

categories	
id	int
title	varchar
desc	varchar
entry	enum
is_deletable	boolean
created_at	timestamp
modified_at	timestamp

payment_options	
id	int
title	varchar
desc	varchar
balance	float8
is_deletable	boolean
created_at	timestamp
modified_at	timestamp

transctions	
id	int
title	varchar
desc	varchar
budeget_id	int
option_id	int
category_id	int
amount	float8
created_at	timestamp
modified_at	timestamp

Tables:

- **budgets**
  - Contains information about budget plans.
  - Fields include unique ID, title, description, allotted and balance amounts, expiry date, an active status indicator, and timestamps for creation and modification.

- **categories**
  - Holds different categories for transactions.
  - Fields include unique ID, title, description, entry type (could denote income or expense via an enum type), deletable status, and creation and modification timestamps.

- **transactions**
  - Stores individual transaction records.
  - Fields include unique ID, title, description, foreign keys linking to budgets (**budget_id**), payment options (**option_id**), and categories (**category_id**), transaction amount, and timestamps for creation and modification.

- **payment_options**
  - Describes different payment options or methods.
  - Fields include unique ID, title, description, balance (which could represent the current balance for this payment method), deletable status, and creation and modification timestamps.

Relationships:

- **budgets to transactions**: One-to-Many relationship (A single budget can be associated with many transactions, but each transaction is linked to only one budget).

- **categories to transactions**: One-to-Many relationship (A single category can apply to many transactions, but each transaction has one category).

- **payment_options to transactions**: One-to-Many relationship (A payment option can be used for many transactions, but each transaction is conducted with one payment method).

Explanation:

The schema organizes financial data into four main entities, each responsible for a distinct aspect of the personal finance management process:

- **Budgets**: This entity manages budget data, allowing users to create and track financial plans over time.

- **Categories**: This entity categorizes financial data, which helps in classifying transactions into various types such as 'Groceries', 'Salary', 'Investments', etc.

- **Transactions**: Central to the application, this entity captures all financial transactions, linking each to a budget, a category, and a payment option.

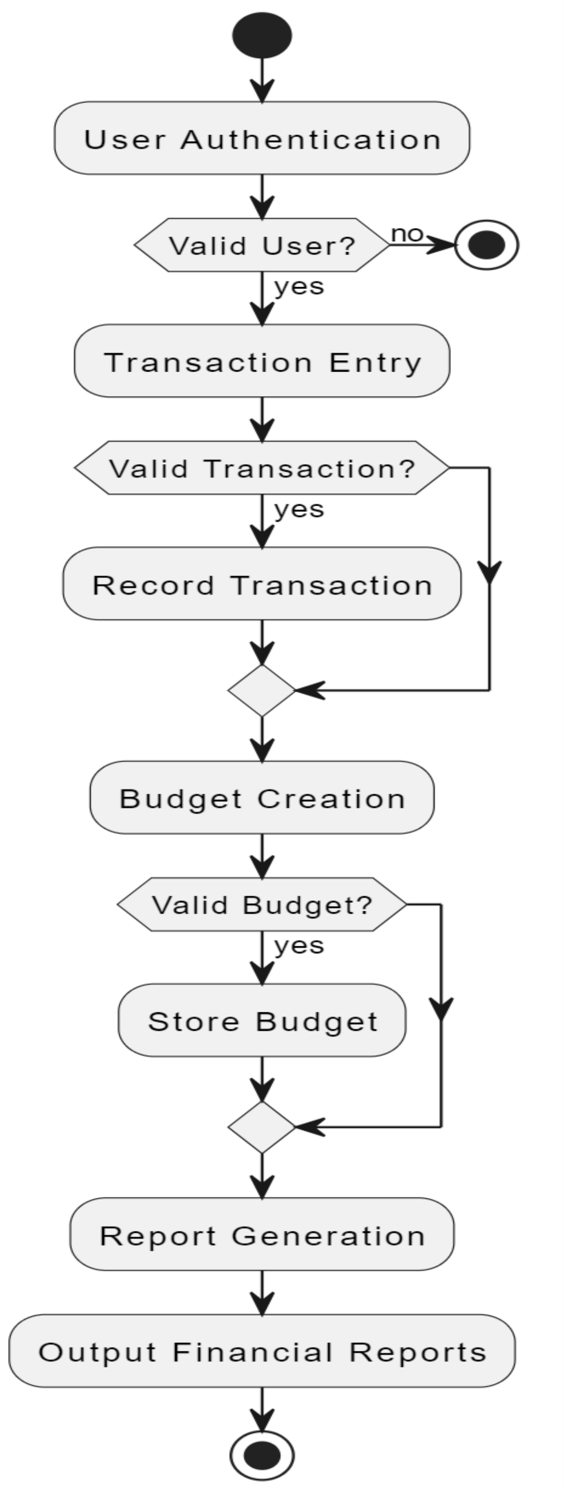
- **Payment Options**: This entity provides details on the various methods through which transactions can occur, such as 'Credit Card', 'Cash', 'Bank Transfer', etc.

UML

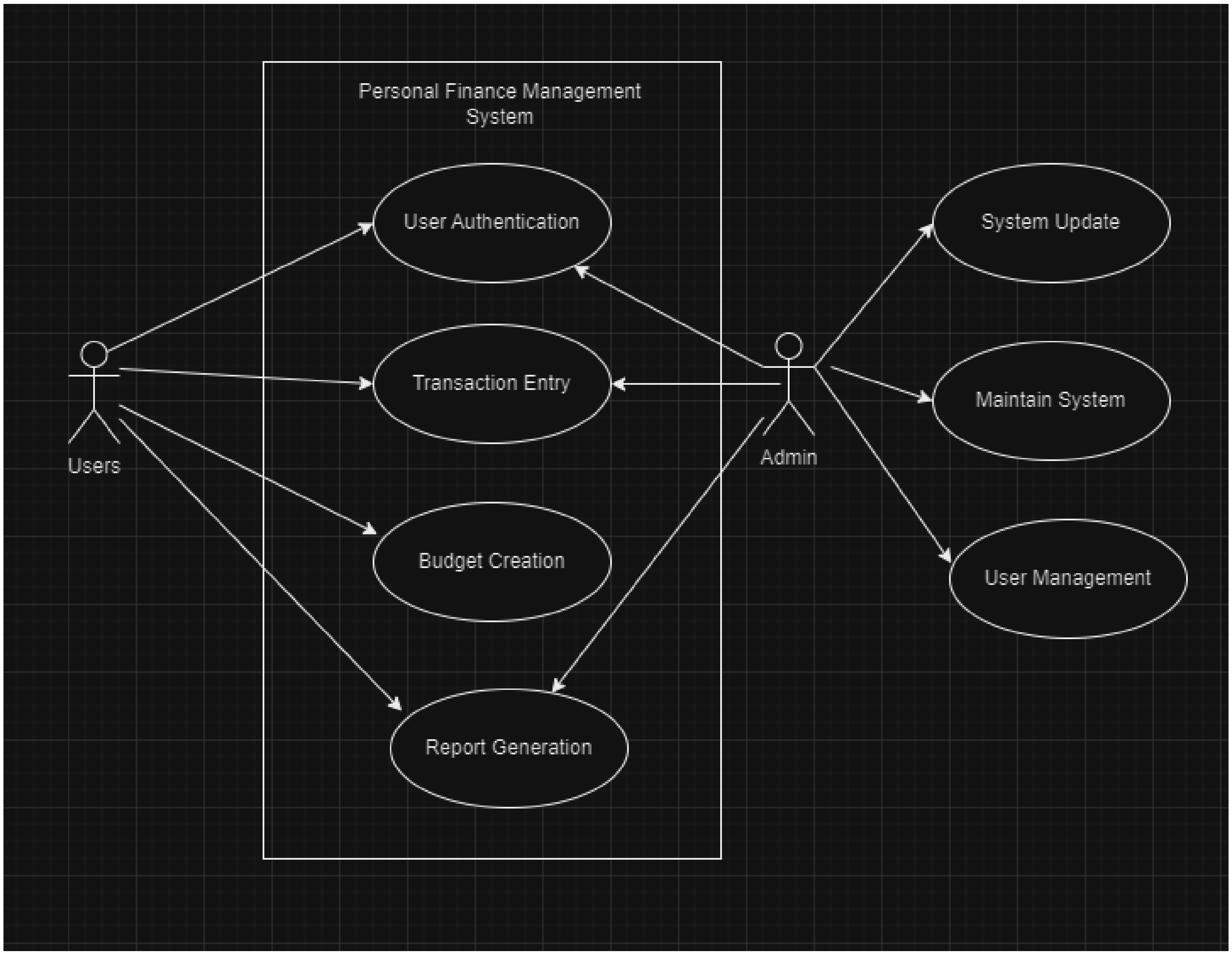
Explanation

- **Start:** The starting point of the user interaction flow.
- **User Authentication:** The system attempts to authenticate the user.
- **Valid User?:** A decision point where the system checks if the user credentials are valid.
  - If No, the process ends (depicted by the circle with a bullseye, which denotes an end point).
  - If Yes, the flow proceeds to the next step.
- **Transaction Entry:** The user enters a transaction into the system.
- **Valid Transaction?:** The system checks if the entered transaction is valid.
  - If Yes, it proceeds to record the transaction.
  - It's not shown what happens if the transaction is not valid, but typically, the user might be asked to re-enter the transaction details.
- **Record Transaction:** The valid transaction is recorded in the system.
- **Budget Creation:** Following the transaction entry, there is a step for creating a budget.
- **Valid Budget?:** The system validates the budget details.
  - If Yes, it proceeds to store the budget.
  - The flowchart does not specify the course of action if the budget is not valid, but like with transactions, the user may need to adjust the budget details.
- **Store Budget:** The validated budget is stored in the system.
- **Report Generation**: After storing the budget, the system can generate reports.
- **Output Financial Reports:** The system outputs the generated financial reports.

- **End:** The end of the process flow.



USE Case Diagram

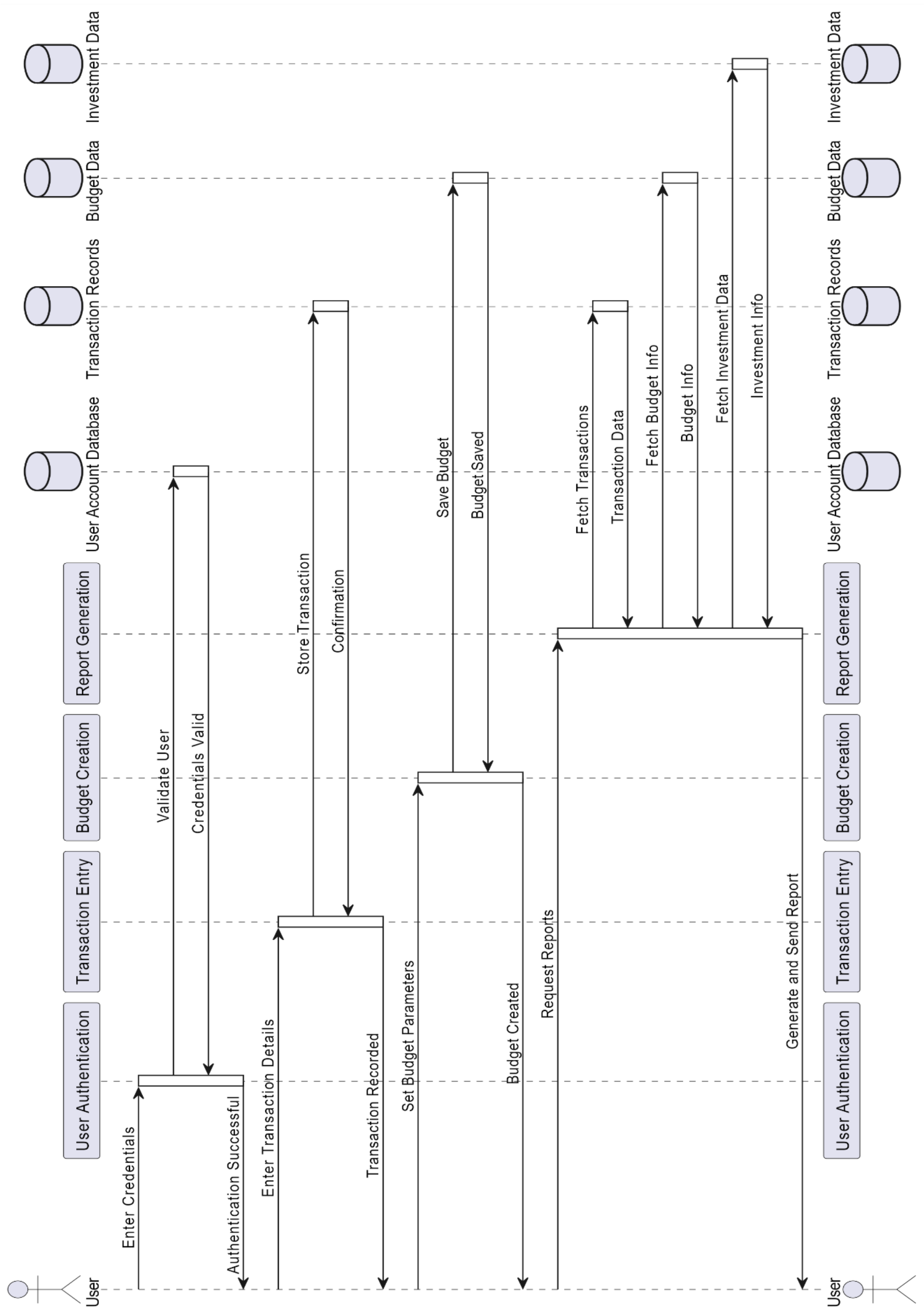


- **User Authentication (UC1):** Represents the functionality where users can log in and log out of the system. It is the gateway for accessing other functionalities.
.
- **Transaction Entry (UC2):** Allows users to enter details of their financial transactions. This use case is essential for keeping financial records updated.

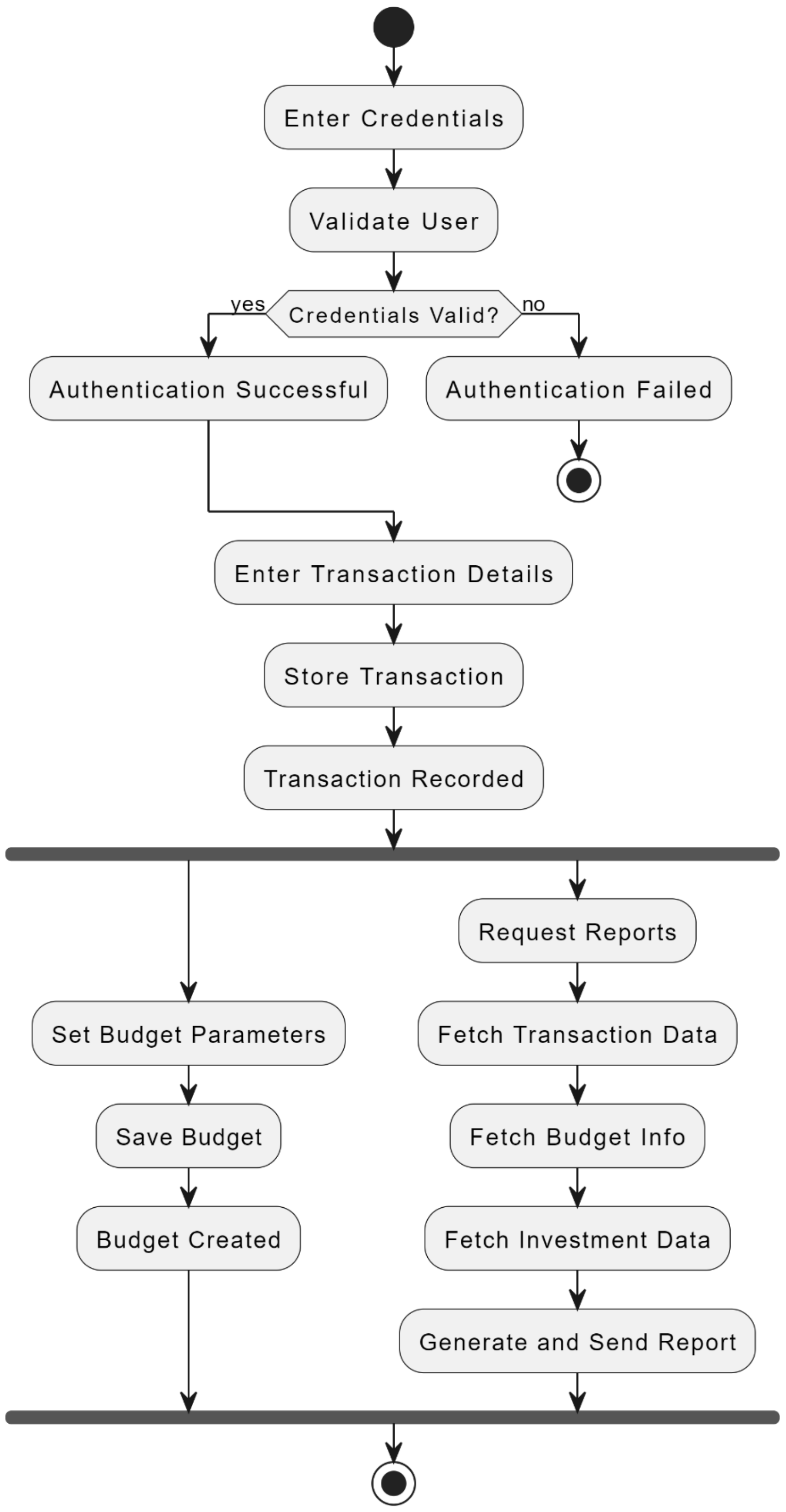
- **Budget Creation (UC3):** Users can create budgets by setting parameters for spending or saving, which helps in financial planning.

- **Report Generation (UC4):** The system generates various financial reports based on user transactions, budget setups, and possibly other data, providing insights into financial status.

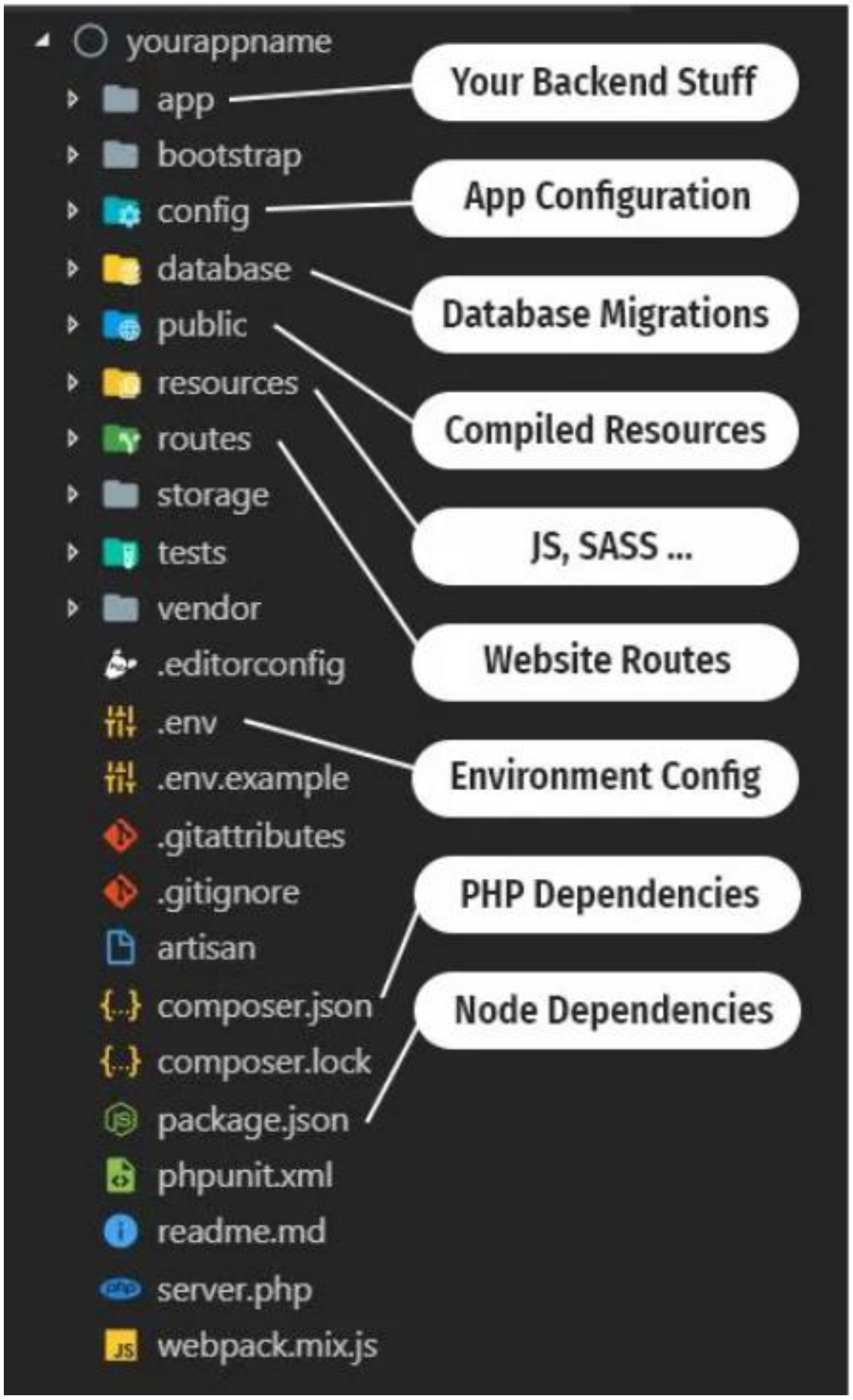
Sequence Diagram



Activity Diagram



Folder and file Structure



Folders

- app/Console : Terminal commands
- app/Exceptions: Error handling stuff

- app/Http: Controllers and Middleware (C in MVC architecture)
- app/Modes: For example, User.php represents the user table in database

- app/Provider: Used when creating providers
- app/bootstrap: Cashe files for bootstrap (no need to touch)

- app/config: All the configuration files
- app/database : database tables and migrations, and also seeders

- app/node_modules: modules installed during running command ‘npm install’
- app/public: All users can access css, js in public folder

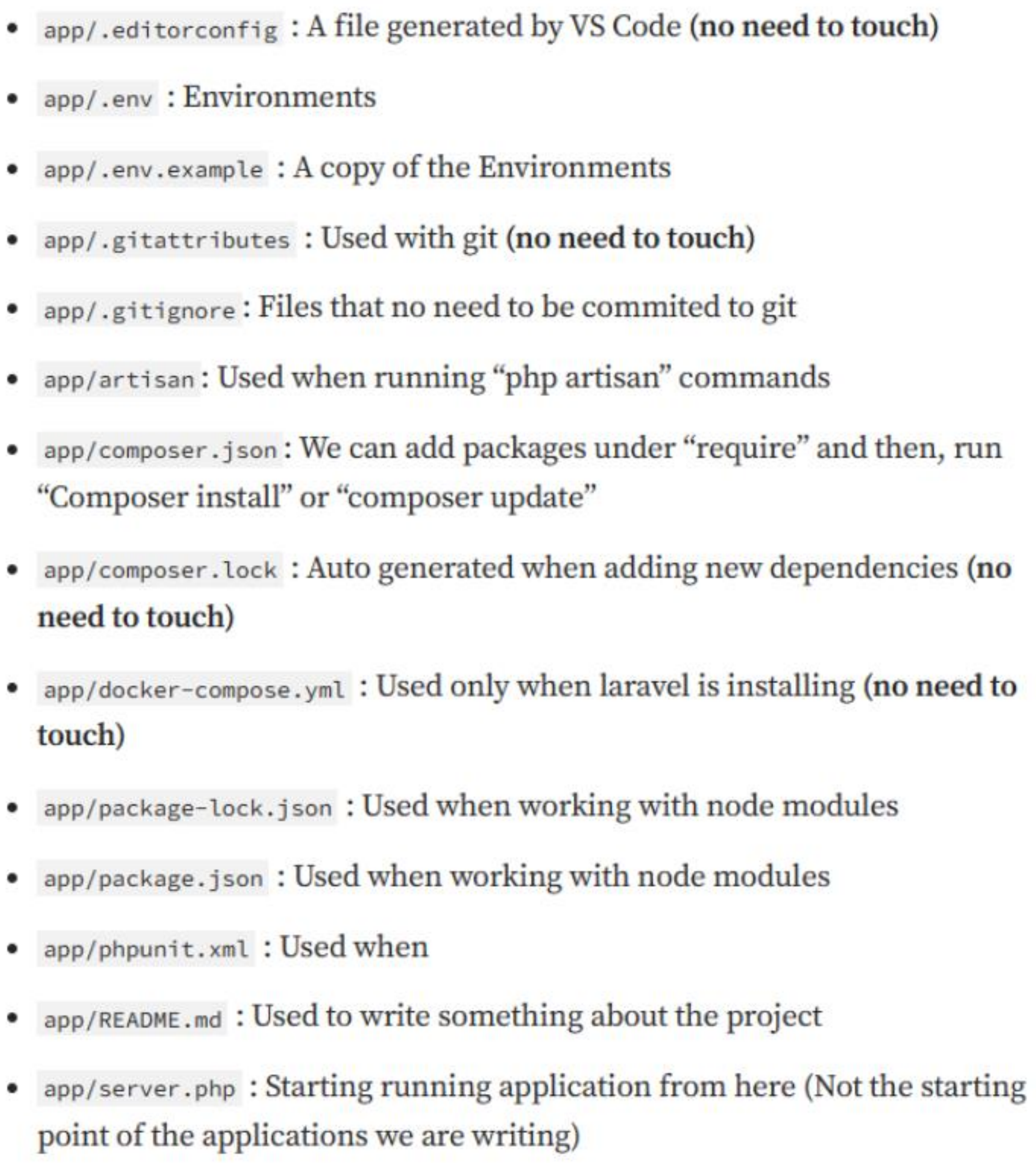
- app/ resources: All the resources which is related to the Views (V in MVC)
- app/routes : Routings (navigations) for the web pages

- app/storage: storage folder cannot be accessed by all the users. This
- folder can be used to store user specific data which need to be restricted

- from the public access.
- app/tests: Test cases

- app/vendor: All the packages we are install using Composer will relies in
- this folder

Files



- `app/.editorconfig` : A file generated by VS Code **(no need to touch)**
- `app/.env` : Environments
- `app/.env.example` : A copy of the Environments
- `app/.gitattributes` : Used with git **(no need to touch)**
- `app/.gitignore` : Files that no need to be commited to git
- `app/artisan` : Used when running "php artisan" commands
- `app/composer.json` : We can add packages under "require" and then, run "Composer install" or "composer update"
- `app/composer.lock` : Auto generated when adding new dependencies **(no need to touch)**
- `app/docker-compose.yml` : Used only when laravel is installing **(no need to touch)**
- `app/package-lock.json` : Used when working with node modules
- `app/package.json` : Used when working with node modules
- `app/phpunit.xml` : Used when
- `app/README.md` : Used to write something about the project
- `app/server.php` : Starting running application from here (Not the starting point of the applications we are writing)

Hardware and software requirement

System Requirements

Operating System (minimum): Windows 10, MacOS Monterey, Linux Ubuntu 20.04 LTS

Software Requirements:

- - PHP 7.4 or higher
- - MySQL 5.7 or higher
- - Composer
- - Node.js 12.x
- - XAMPP (for local development)

Hardware requirements (minimum):

- **Processor:** Intel(R) Core (TM) i3-10th generation
- **Speed:** 2.60 GHz
- **RAM:** 4 GB
- **Storage:** 118 SSD

Future enhancement

- **Machine Learning for Spending Insights:** Implement machine learning algorithms to analyze spending patterns and provide personalized financial advice or insights.

- **Integration with Financial Institutions:** Enable direct linking to users' bank and credit accounts for real-time transaction data, which would automate transaction entries and provide up-to-date financial information.

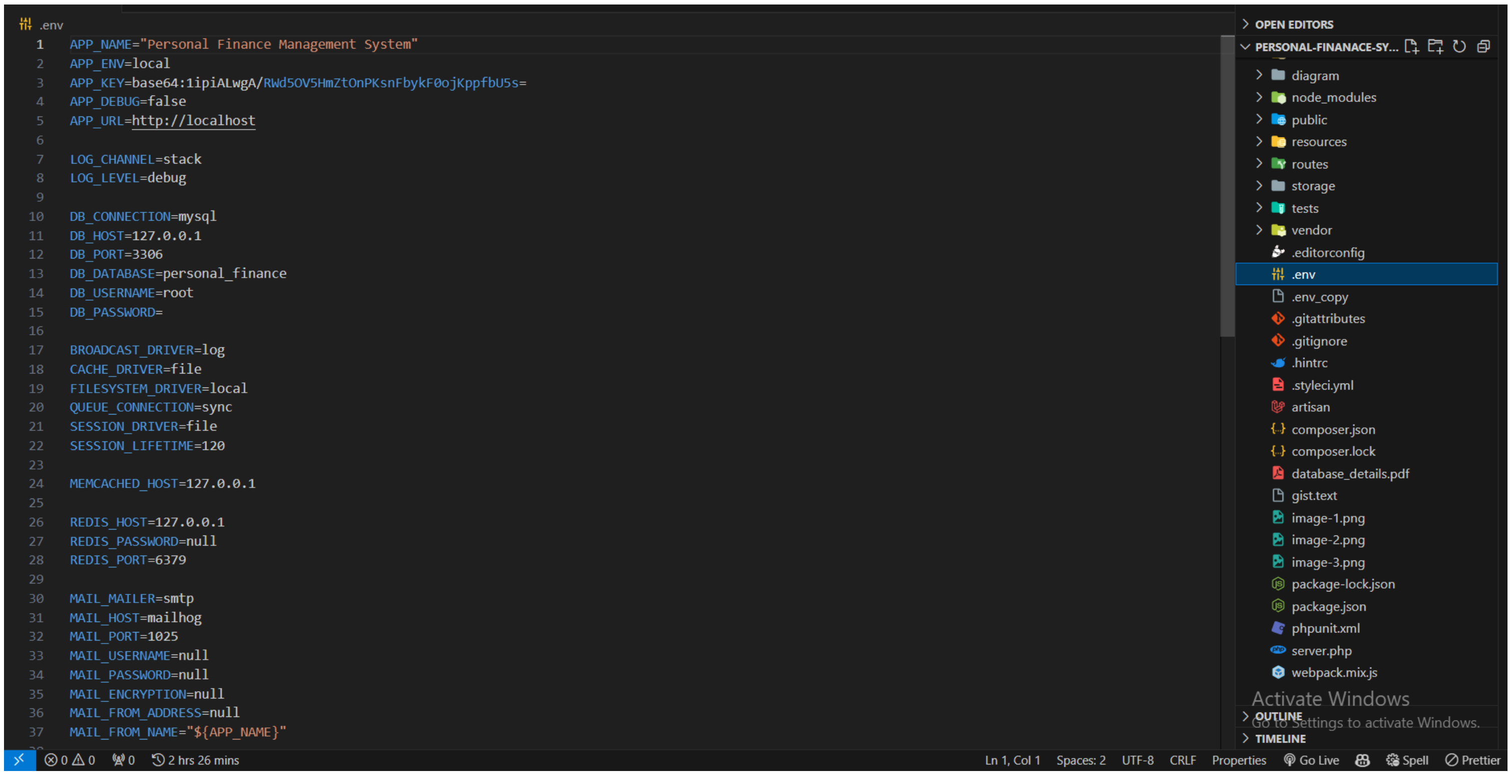
- **Mobile App Development:** Develop a mobile application for iOS and Android to provide users with on-the-go access to their financial data and tools.

- **Enhanced Reporting Tools:** Offer more detailed and customizable reports, including predictive budgeting and forecasting features based on historical data.

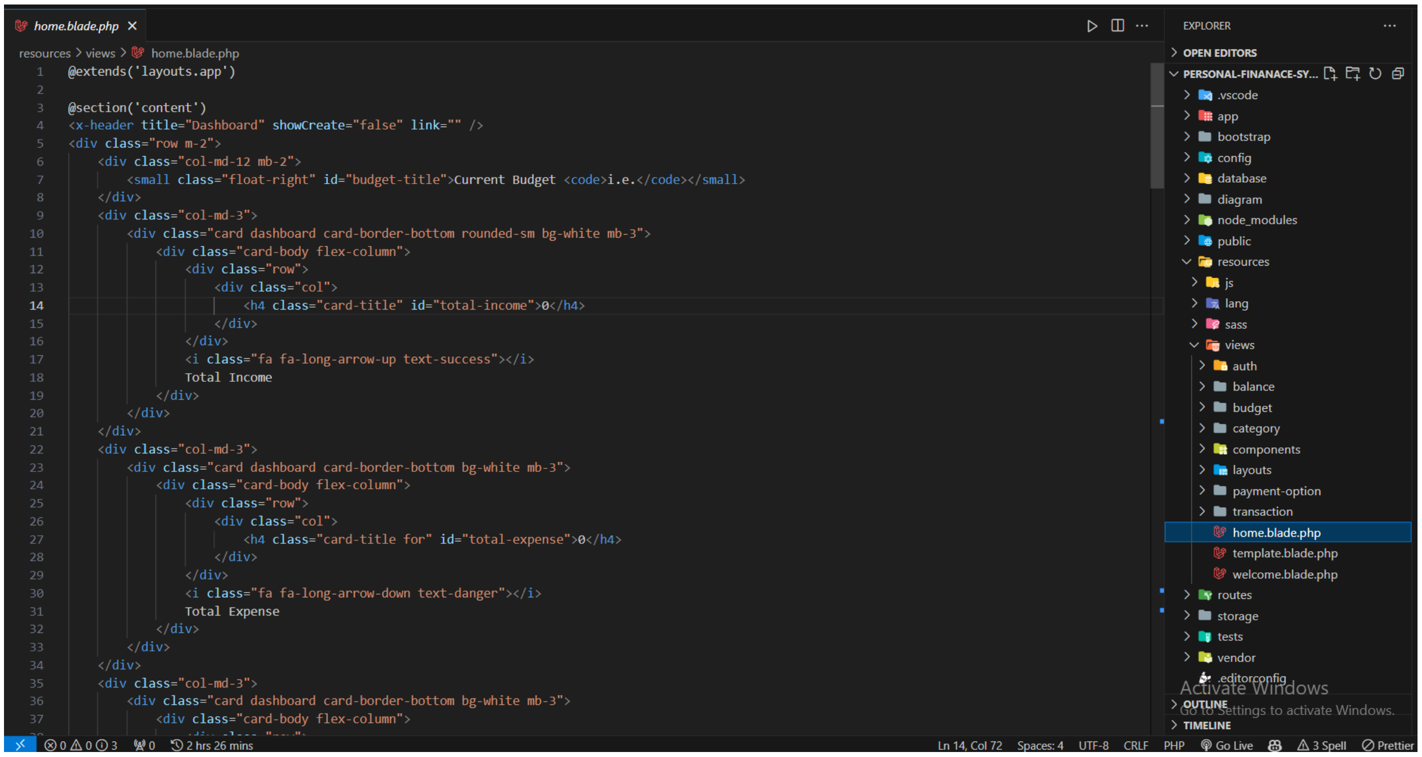
- **Investment Tracking:** Incorporate investment tracking and analytics, allowing users to monitor their investment portfolios and receive alerts on significant market changes.

- **Notifications and Alerts:** Implement customizable notifications for budget limits, bill payments, and financial goals.
- **User Customization:** Allow users to customize the dashboard and reports to prioritize the information they find most important.
- **Multi-Currency Support:** For users who deal with multiple currencies, provide features to track and manage transactions in different currencies with real-time conversion rates.
- **Debt Management Tools:** Integrate tools to help users manage and plan for debt repayment, including calculators for mortgages, loans, and credit card debts.
- **User Collaboration:** Add features that allow family members or partners to collaborate on budgets and financial planning within a shared environment.

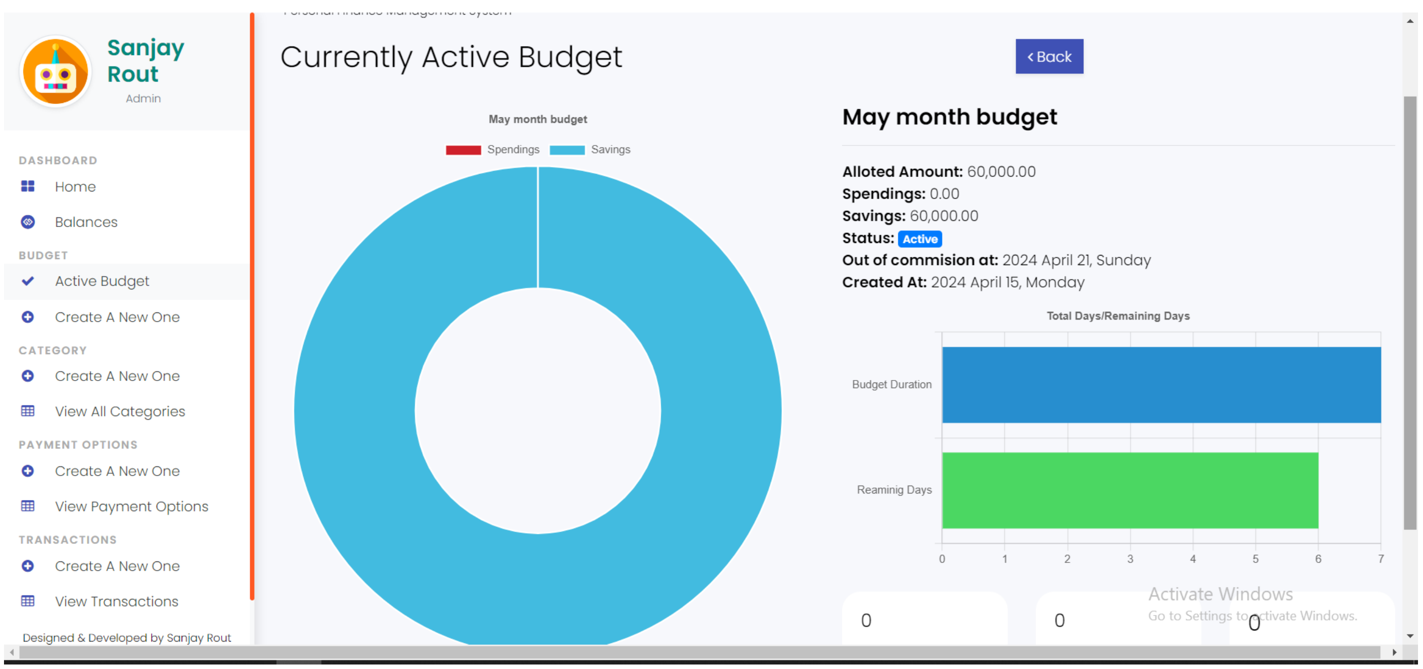
Screenshot of program and software



```
APP_NAME="Personal Finance Management System"
APP_ENV=local
APP_KEY=base64:1ipiALwgA/RWd5OV5HmZtOnPKsnFbykF0ojKppfbU5s=
APP_DEBUG=false
APP_URL=http://localhost

LOG_CHANNEL=stack
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=personal_finance
DB_USERNAME=root
DB_PASSWORD=

BROADCAST_DRIVER=log
CACHE_DRIVER=file
FILESYSTEM_DRIVER=local
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"
```



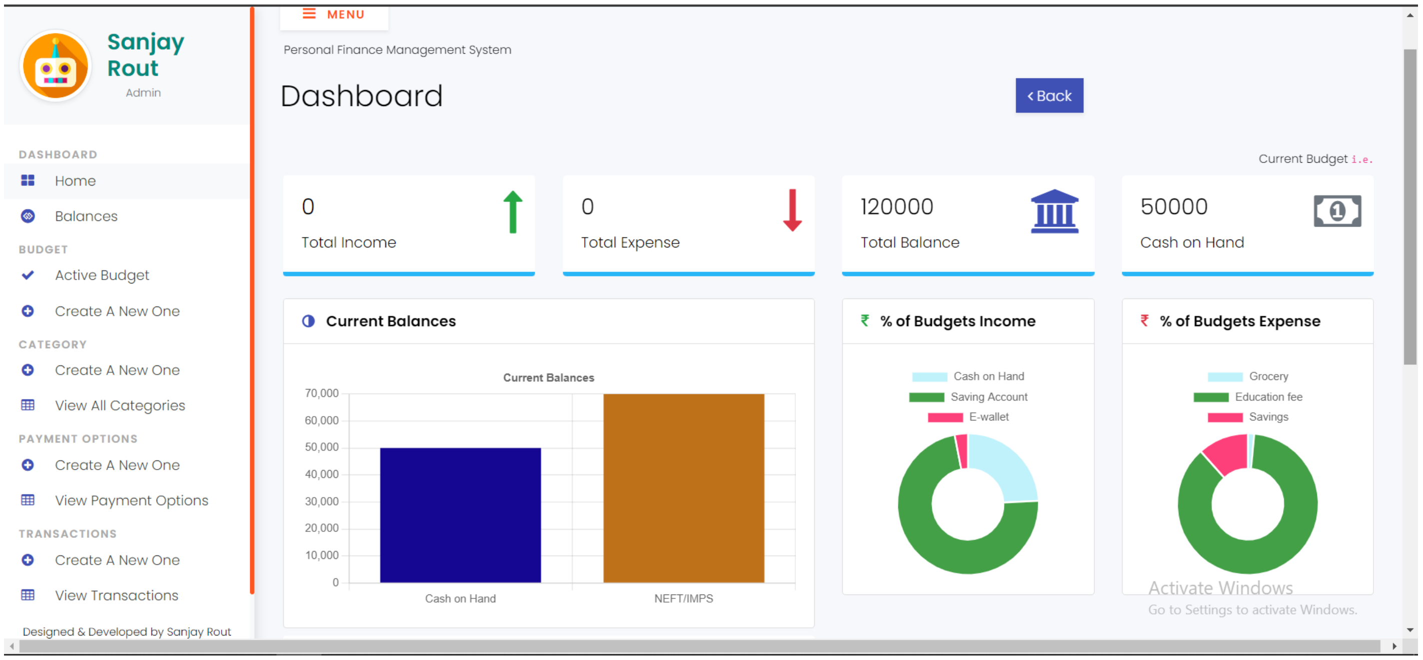
```
@extends('layouts.app')

@section('content')
<x-header title="Dashboard" showCreate="false" link="" />
<div class="row m-2">
    <div class="col-md-12 mb-2">
        <small class="float-right" id="budget-title">Current Budget <code>i.e.</code></small>
    </div>
    <div class="col-md-3">
        <div class="card dashboard card-border-bottom rounded-sm bg-white mb-3">
            <div class="card-body flex-column">
                <div class="row">
                    <div class="col">
                        <h4 class="card-title" id="total-income">0</h4>
                    </div>
                </div>
                <i class="fa fa-long-arrow-up text-success"></i>
                Total Income
            </div>
        </div>
    </div>
    <div class="col-md-3">
        <div class="card dashboard card-border-bottom bg-white mb-3">
            <div class="card-body flex-column">
                <div class="row">
                    <div class="col">
                        <h4 class="card-title for" id="total-expense">0</h4>
                    </div>
                </div>
                <i class="fa fa-long-arrow-down text-danger"></i>
                Total Expense
            </div>
        </div>
    </div>
    <div class="col-md-3">
        <div class="card dashboard card-border-bottom bg-white mb-3">
            <div class="card-body flex-column">
```



Currently Active Budget

May month budget

Alloted Amount: 60,000.00
Spendings: 0.00
Savings: 60,000.00
Status: Active
Out of commision at: 2024 April 21, Sunday
Created At: 2024 April 15, Monday



Sanjay Rout
Admin

DASHBOARD
Home
Balances
BUDGET
Active Budget
Create A New One
CATEGORY
Create A New One
View All Categories
PAYMENT OPTIONS
Create A New One
View Payment Options
TRANSACTIONS
Create A New One
View Transactions

Designed & Developed by Sanjay Rout

MENU

Personal Finance Management System

Dashboard

‹ Back

Current Budget i.e.

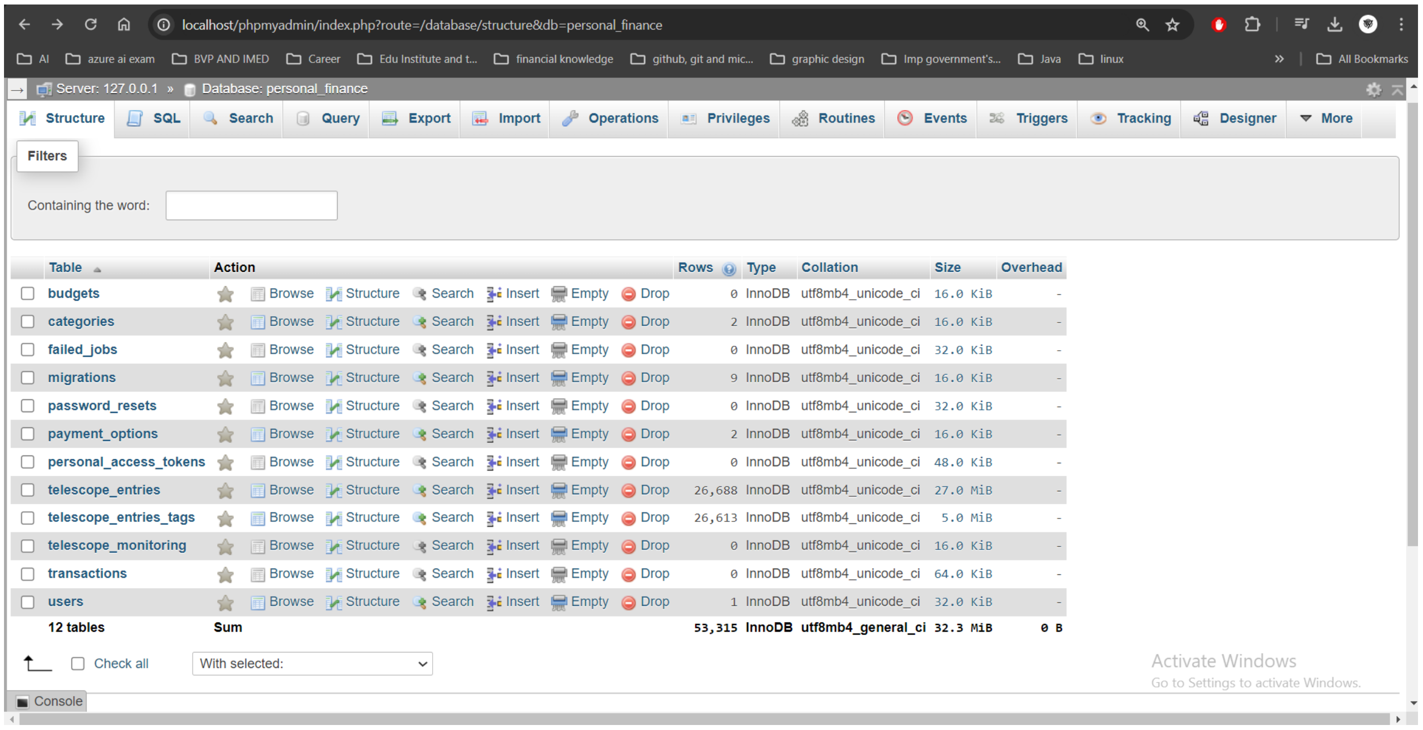
0	Total Income
0	Total Expense
120000	Total Balance
50000	Cash on Hand

Current Balances

% of Budgets Income

% of Budgets Expense

Activate Windows
Go to Settings to activate Windows.



localhost/phpmyadmin/index.php?route=/database/structure&db=personal_finance

AI | azure ai exam | BVP AND IMED | Career | Edu Institute and t... | financial knowledge | github, git and mic... | graphic design | Imp government's... | Java | linux | All Bookmarks

Server: 127.0.0.1 » Database: personal_finance

Structure | SQL | Search | Query | Export | Import | Operations | Privileges | Routines | Events | Triggers | Tracking | Designer | More

Filters

Containing the word:

Table	Action	Rows	Type	Collation	Size	Overhead
budgets	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
categories	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
failed_jobs	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
migrations	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
password_resets	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
payment_options	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
personal_access_tokens	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
telescope_entries	Browse Structure Search Insert Empty Drop	26,688	InnoDB	utf8mb4_unicode_ci	27.0 MiB	-
telescope_entries_tags	Browse Structure Search Insert Empty Drop	26,613	InnoDB	utf8mb4_unicode_ci	5.0 MiB	-
telescope_monitoring	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
transactions	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	64.0 KiB	-
users	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
12 tables	Sum	53,315	InnoDB	utf8mb4_general_ci	32.3 MiB	0 B

Check all | With selected:

Activate Windows
Go to Settings to activate Windows.

Console

Reference

- https://www.investopedia.com/terms/p/personalfinance.asp (Investopedia)
- https://scholarworks.lib.csusb.edu/cgi/viewcontent.cgi?article=4009&context=etd-project (Real-time finance management system)
- https://laravel.com/ (laravel website)