1. **INTRODUCTION**

**1.1 Overview**

In our daily life, managing money is very important. Many people earn money but do not track how they are spending it. Sometimes, people spend more than what they earn and face problems later. In the past, people used books or Excel sheets to manage money. But now, with smartphones and the internet, we can use apps to manage our budget. Our project, Budget Management System, is one such application that helps users track their income and expenses. It also shows them charts and graphs so that they can understand where their money is going. The best part is that it uses AI to scan receipts. If a user buys something and takes a picture of the bill, the system reads the data and adds it automatically.

**1.2 Problem Statement**

Most budgeting apps in the market need users to enter each and every detail manually. This takes time and is boring. Also, many people don’t know how to group their expenses or plan budgets for different needs like food, travel, etc. Some apps do not show clear graphs or summaries, so users cannot understand their spending habits. Our project solves these problems. It automatically reads data from bills, puts transactions into the correct category, and shows users clear reports using graphs and charts.

**1.3 Objectives**

The main goal of our project is to help users:

* Create and manage multiple budgets
* Track income and expenses under different categories
* Scan receipts using AI and add transactions automatically
* Login securely using JWT tokens
* See their financial report in easy-to-understand charts
* Use the app easily on phone, tablet, or computer

**1.4 Motivation**

We got the idea for this project by observing how people around us struggle to manage money. Many students spend without planning and later regret it. Working professionals forget where they spent their money. So, we wanted to build a tool that is simple, smart, and helpful. With the use of AI and modern web technologies, we believed we could make a difference in personal financial management.

**1.5 Hardware and Software Requirements**

**Hardware:**

* Laptop/Desktop with Intel i5 processor or better
* 8 GB RAM minimum
* Internet connection

**Software:**

* Frontend: React, Tailwind CSS, Zustand
* Backend: Node.js, Express.js
* Database: MySQL
* ORM: Prisma
* Tools: Postman, VS Code, GitHub
* Hosting: Vercel (frontend), Render (backend)

**1.6 Project Budget Plan**

This is a mini-project, and we used mostly free tools. The total cost was very low:

| **Expenditure** | **Budget (₹)** |
| --- | --- |
| Training / Online courses | 2000 |
| Paper Presentation / Submission | 3000 |
| Proposal Submission | 3000 |
| Domain (optional) | 800 |
| **Total** | **8800** |

Table 1: Budget Table

1. **LITERATURE SURVEY**

| **Sl. No** | **Author(s) & Year** | **Title** | **Method Used** | **Results/Remarks** |
| --- | --- | --- | --- | --- |
| 1 | S. Bhardwaj et al., 2024 | *Personal Expense Tracker* | MERN stack, real-time tracking | Efficient UI and predictive analytics |
| 2 | Pooja Bhatt et al., 2024 | *Smart Approach to Track Daily Expense* | Bookkeeping + data visualization | Focus on expense categorization |
| 3 | Phat Tran, 2023 | *Expense Tracker using MERN* | Authentication, CRUD, Redux | Robust architecture for financial apps |
| 4 | Dewan et al., 2024 | *FinanceVUE Dashboard* | MERN + dashboard + security | Enterprise-level analytics and UX |
| 5 | Tapkir & Pathak, 2024 | *Expense Tracking Using MERN* | MongoDB, React, Express | High performance and intuitive UI |

Table 2: Literature survey table

**3. PROBLEM ANALYSIS & DESIGN**

**3.1 Existing System**

The MERN-based Personal Expense Tracker addresses key app limitations with:

* Real-time tracking – Updates expenses instantly for better management.
* Automated categorization – Sorts expenses automatically to save time.
* Predictive analytics – Helps users set goals based on spending patterns.
* User-friendly interface – Simple and easy for quick navigation.
* Strong security – Protects user data with robust privacy measures.

**3.2 Proposed System**

Our system solves the above problems by:

* Using Google Gemini API to read receipts
* Automatically adding and categorizing transactions
* Allowing multiple budgets
* Showing clear bar and pie charts for analysis
* Having JWT login for user security
* Designing with a clean and modern look

**3.3 Identified Tools / Libraries / Software**

* React.js: UI development
* Tailwind CSS: Styling
* Zustand: State management
* Node.js: Backend logic
* Express.js: API routing
* MySQL: Database
* Prisma: Database interaction
* Google Gemini API: Receipt scanning

**3.4 Architectural Block Diagram & Modeling**

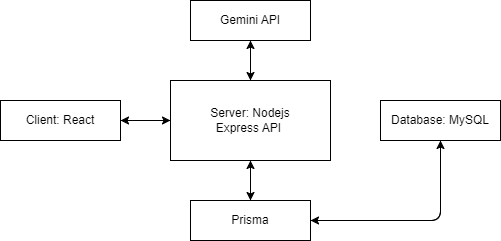
****

Figure 1: Architectural Block Diagram

**3.5 Data Flow Diagram**

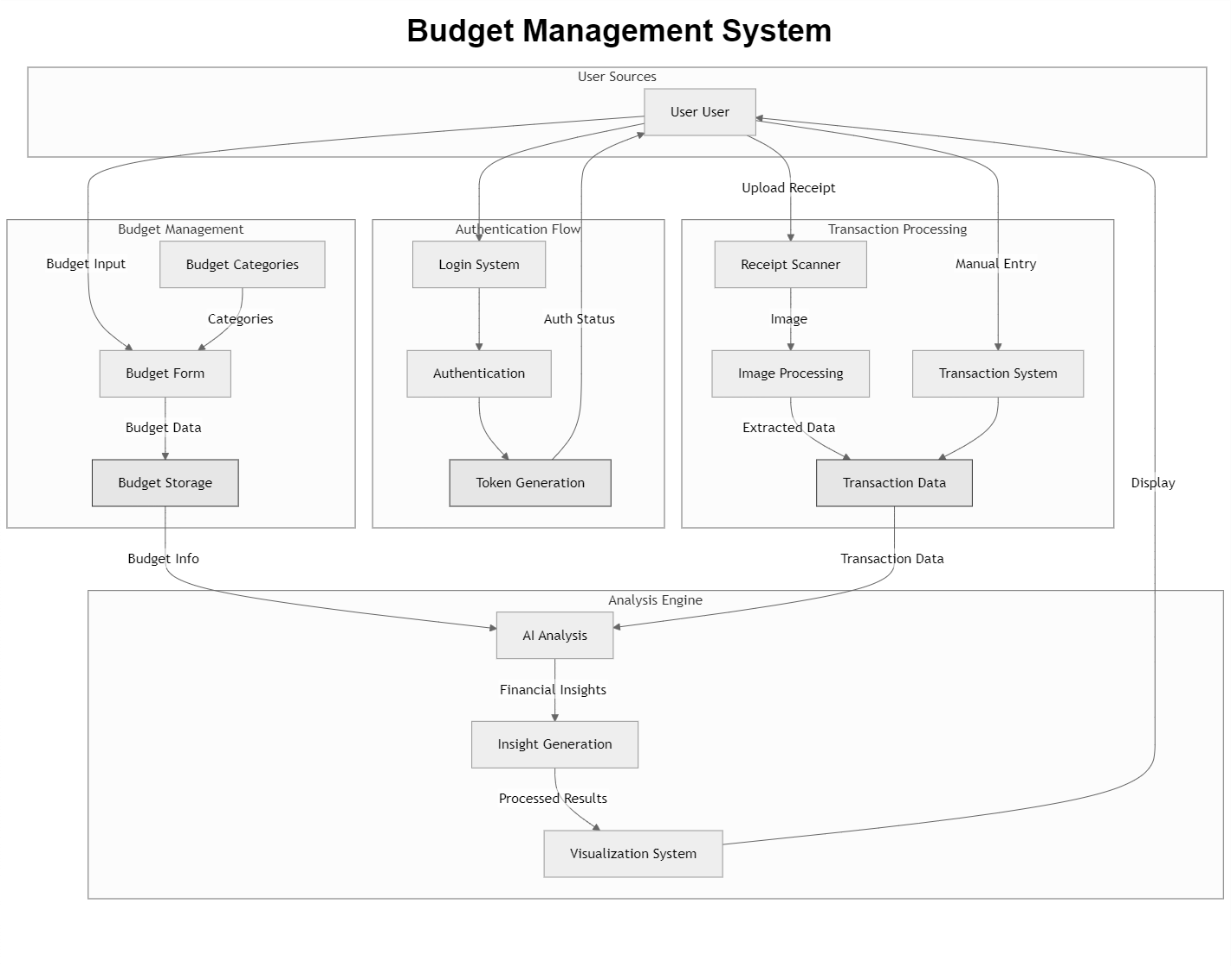


Figure 2: Data Flow Diagram

The Budget Management System is an integrated platform designed to streamline personal financial management by offering the following key functionalities:

* User Authentication – Secures system access through a login system, user authentication, and token generation.
* Budget Management – Allows users to create and categorize budgets using a structured budget form, with the data stored for future reference.
* Transaction Processing – Supports both automated receipt scanning (via image processing) and manual transaction entry to collect accurate financial data.
* AI Analysis Engine – Processes combined budget and transaction data to generate insightful financial analysis.
* Visualization System – Presents processed insights through intuitive visual displays, enabling users to understand and manage their financial activities effectively.

**3.6 Use Case Diagram**

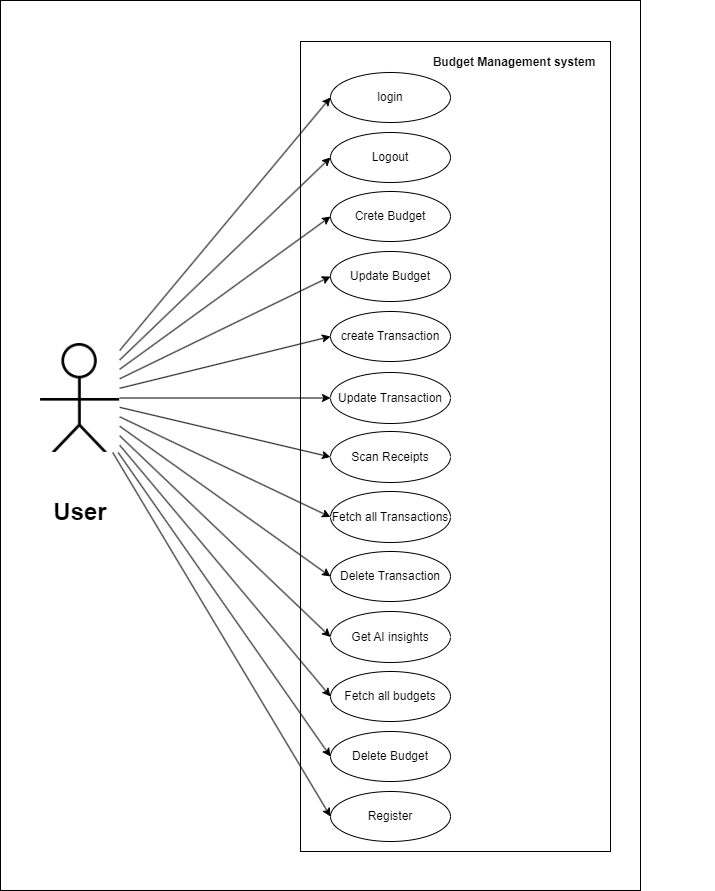


Figure 3: Use Case Diagram

The User is the primary actor in the system and can interact with the Budget Management System through the following use cases:

* Register – Allows new users to create an account in the system.
* Login – Authenticates a registered user to access system features.
* Logout – Terminates the current user session securely.
* Create Budget – Enables users to define a new budget plan.
* Update Budget – Allows users to make changes to an existing budget.
* Delete Budget – Permits users to remove an unwanted or outdated budget entry.
* Fetch All Budgets – Retrieves a list of all budgets created by the user.
* Create Transaction – Lets users add a new transaction manually.
* Update Transaction – Enables editing of existing transaction records.
* Delete Transaction – Removes a specific transaction from the system.
* Fetch All Transactions – Retrieves all transaction data for review.
* Scan Receipts – Allows users to upload receipts for automated transaction data extraction using image processing.
* Get AI Insights – Provides users with financial insights generated through AI analysis of budget and transaction data.

**3.7 ER Diagram**

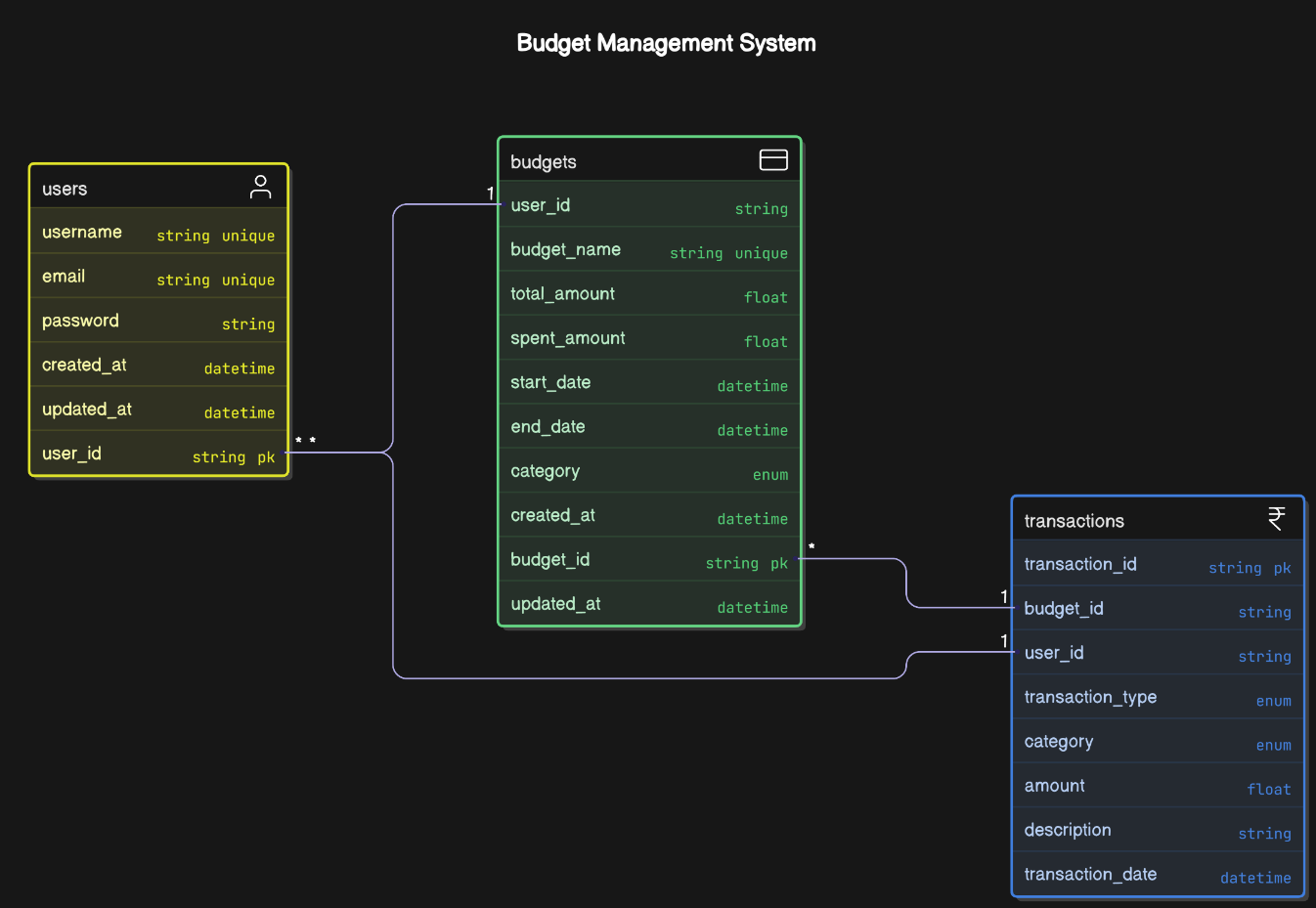


Figure 4: ER Diagram

The ER diagram for a Budget Management System has three main entities: users, budgets, and transactions. Here's a concise breakdown:

1. **Users** 
   * **Users Attributes:** user\_id (PK), username, email, password, created\_at, updated\_at.
   * **Relationships:** One user can have multiple budgets and transactions.
2. **Budgets**

* **Budgets Attributes**: budget\_id (PK), user\_id (FK), budget\_name, total\_amount, spent\_amount, start\_date, end\_date, category, created\_at, updated\_at.
* **Relationships:** One budget can have multiple transactions, linked to one user.

1. **Transactions**

* Transactions Attributes: transaction\_id (PK), budget\_id (FK), user\_id (FK), transaction\_type, category, amount, description, transaction\_date.
* Relationships: Each transaction is linked to one user and one budget.

**4. IMPLEMENTATION**

**4.1 Overview of System Implementation**

The Budget Management System was implemented using modern web technologies. The system has two main parts: the frontend and the backend. The frontend is built using React.js and styled using Tailwind CSS, which helps make the application look neat and work well on different devices. The backend is built using Node.js and Express.js, where we write all the logic for user authentication, budget handling, transaction recording, and AI integration. The database is handled using MySQL, and Prisma is used to connect and manage the database easily.

The system allows users to register, log in securely, create and manage their budgets, upload receipts for automatic transaction entry, and view data in graphs. The architecture is designed in such a way that each module works independently, which makes the system scalable and easy to maintain.

**4.2 Module Description**

The project is divided into several modules:

* **User Module**: Handles user registration and login with JWT-based authentication.
* **Budget Module**: Allows users to create, update, and delete different budgets.
* **Transaction Module**: Lets users manually or automatically (via AI) add and categorize income and expenses.
* **AI Receipt Module**: Uses Google Gemini API to scan uploaded receipts and convert them into structured data.
* **Dashboard Module**: Shows graphs and charts (like pie and bar charts) to visualize spending patterns and savings.

**4.3 Code Snippets**

Client-Server Sample Code

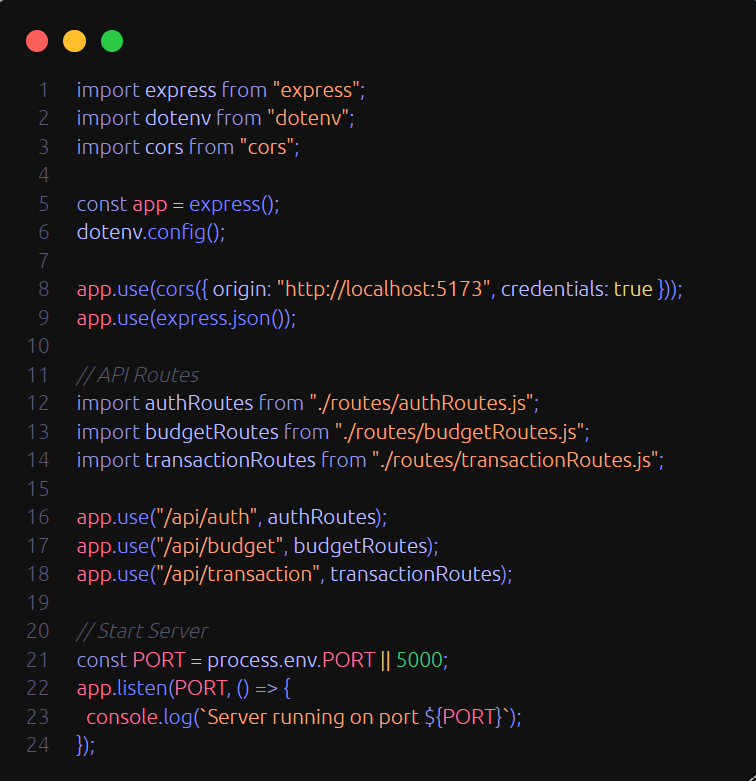
 

Figure 5: React.js sample code snippet Figure Figure 6: Node.js sample code snippet

**5. TESTING**

**5.1 INTRODUCTION**

The testing phase for the Budget Management System was conducted to ensure the reliability, security, and user-friendliness of this comprehensive financial management platform. The system, designed to revolutionize personal finance management, incorporates advanced features such as AI-powered receipt scanning, multibudget tracking, and intuitive financial reporting through visual analytics.

The testing focused on critical modules including user authentication, budget management, transaction tracking, and AI-powered receipt processing. Each component was evaluated through comprehensive test suites to verify correct behavior, security measures, and error handling capabilities. Special attention was given to the system's ability to handle multiple budgets, categorize transactions automatically, and generate accurate financial reports.

The testing approach followed structured software testing principles, incorporating both black-box and whitebox testing methodologies. This dual approach allowed for thorough validation of both external functionality and internal implementation details. The test suite covered various scenarios including normal operations, edge cases, and error conditions to ensure the system's readiness for real-world deployment in personal finance management.

**Key areas of testing included:**

* Secure user authentication using JWT tokens
* Multi-budget creation and management
* Transaction categorization and tracking
* AI-powered receipt scanning and data extraction
* Financial report generation and visualization
* Cross-platform compatibility (mobile, tablet, desktop)
* Database operations and data persistence
* API endpoint validation and error handling

The testing process was designed to validate the system's ability to meet its core objectives of simplifying budget management, automating transaction entry, and providing clear financial insights through visual analytics. This comprehensive testing approach ensures that the Budget Management System delivers a reliable, secure, and user-friendly experience for managing personal finances.

**5.2 TESTING TOOLS USED**

The backend system was tested using a combination of modern testing tools and frameworks:

1. **Jest** 
   * Primary testing framework used for writing and executing test cases
   * Provided assertion capabilities for validating expected outcomes
   * Enabled test organization through describe/it blocks
   * Supported async/await for testing asynchronous operations
   * Facilitated mocking and stubbing of dependencies
2. **Supertest** 
   * Used for testing HTTP endpoints and API routes
   * Enabled simulation of HTTP requests to test API behavior
   * Provided response validation capabilities
   * Allowed testing of HTTP status codes, headers, and response bodies

**5.3 TESTING STRATEGY**

The testing strategy employed a comprehensive approach combining both black-box and white-box testing methodologies:

1. **Black-Box Testing** 
   * Focused on testing the system from an external perspective
   * Validated API endpoints and their responses
   * Tested user authentication flows (signup, login, logout)
   * Verified error handling and input validation
   * Ensured proper HTTP status codes and response formats

1. **White-Box Testing** 
   * Examined internal implementation details
   * Tested database operations and data persistence
   * Verified password hashing and security mechanisms
   * Validated business logic implementation
   * Tested error handling at the code level

**5.4 CODE SNIPPETS**

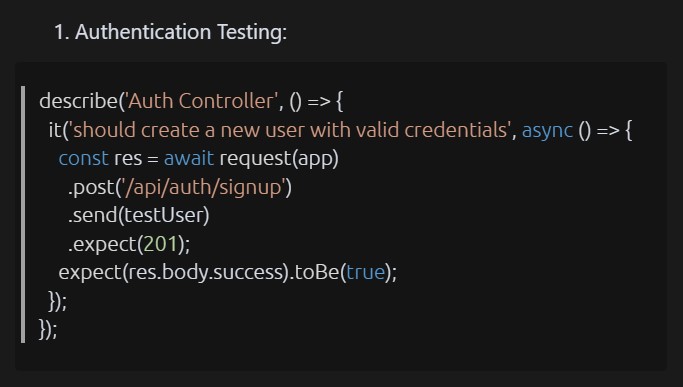
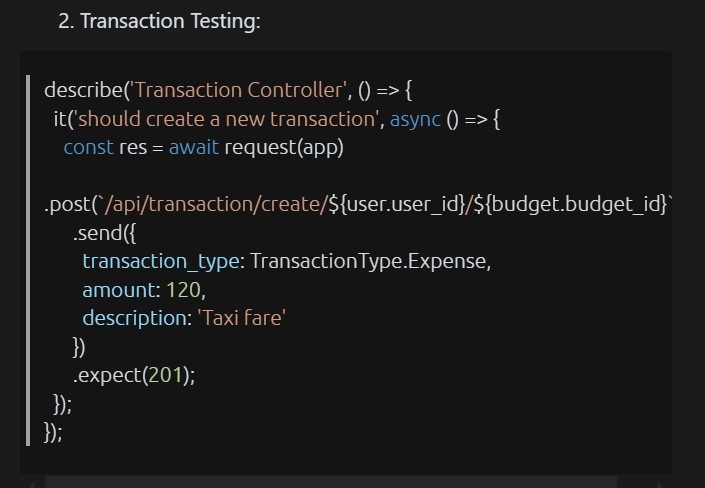
 

Figure 7: Authentication code snippet for testing Figure 8: Budgets code snippet for testing

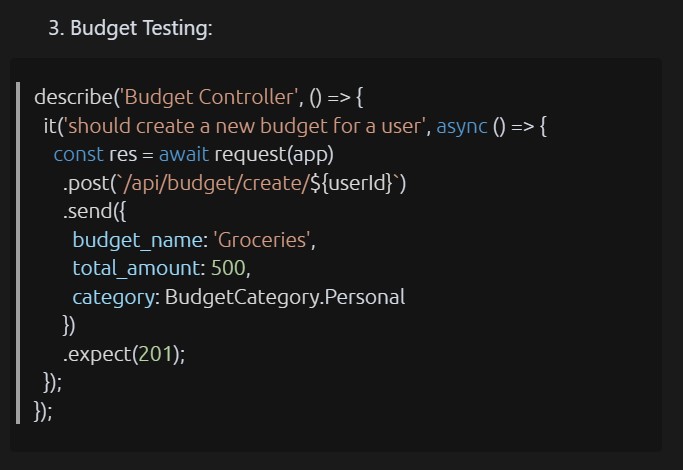


Figure 9: Transaction code snippet for testing

**5.5 TESTING TABLES**

**1. Black-box Test Cases**

These tests verify the application’s functionality from an external perspective, focusing on API behavior and responses.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID | Description | File | Inputs | Expected  Output | Actual  Output | Result |
| BB01 | Return 400 if required fields missing on signup | auth.blackbox.test.js | Partial signup data  (missing fields) | 400 status, error "Please provide all fields" | 400 status, error "Please provide all fields" | PASSED |
| BB02 | Reject login with incorrect password | auth.blackbox.test.js | Registered email, wrong password | 400 status, message "Invalid credentials" | 400 status, message "Invalid credentials" | PASSED |
| BB03 | Reject budget creation with invalid dates | budget.blackbox.test.js | Budget data with invalid date strings | 400 status, error "Invalid date format" | 400 status, error "Invalid date format" | PASSED |
| BB04 | Return 400 for updating with duplicate budget name | budget.blackbox.test.js | Update with duplicate budget name | 400 status, error about duplicate budget name | Test placeholder, assumed PASSED | PASSED |
| BB05 | Return 500 if required fields missing creating transaction | transaction.blackbox.test.js | Partial transaction data  (missing fields) | 500 status, error property in response | 500 status, error property in response | PASSED |

Table 3: Blackbox testing table

**2. White-box Test Cases**

These tests examine the internal workings of the system, including database and logic validation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test  ID | Descriptio n | File | Inputs | Expected  Output | Actual Output | Result |
| WB0  1 | Hash password before saving to database | auth.whitebox.test.js | Plain password | Password hashed and stored, bcrypt.compar  e true | Password hashed and stored, bcrypt.compar  e true | PASSE  D |
| WB0  2 | Store parsed date objects correctly in DB | budget.whitebox.test.js | Budget data with date strings | `start\_date` and `end\_date` stored as Date objects | Dates stored as Date objects | PASSE  D |
| WB0  3 | Save transaction correctly in DB | transaction.whitebox.test. js | Valid transactio n data | Transaction created with correct fields | Transaction created with correct fields | PASSE  D |

Table 4: white box testing table

**3. Integration and Other Test Cases**

These tests verify interaction between components and validate specific units.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID | Description | File | Inputs | Expected  Output | Actual  Output | Result |
| IT01 | Create a new user with valid credentials | auth.test.js | Valid username, email, password | 201 status, success true, user object with user\_id, cookie token | Same as expected | PASSED |
| IT02 | Return 400 if required fields missing on signup | auth.test.js | Partial user data | 400 status, error "Please provide all fields" | Same as expected | PASSED |
| IT03 | Return 400 if email already exists | auth.test.js | Existing email | 400 status, error "User already exists" | Same as expected | PASSED |
| IT04 | Return 400 if username already exists | auth.test.js | Existing username | 400 status, error "User name already exists" | Same as expected | PASSED |
| IT05 | Login with valid credentials | auth.test.js | Registered email, correct password | 200 status, success true, user object, cookie token | Same as expected | PASSED |
| IT06 | Return 400 with invalid email | auth.test.js | Unregistered email | 400 status, success false, message "Invalid credentials" | Same as expected | PASSED |
| IT07 | Return 400 with invalid password | auth.test.js | Registered email, wrong password | 400 status, success false, message "Invalid credentials" | Same as expected | PASSED |
| IT08 | Clear token cookie on logout | auth.test.js | No input | 200 status, success true, message "Logged out", cookie cleared | Same as expected | PASSED |
| IT09 | Create new budget for user | budget.test.js | Valid budget data | 201 status, budget object with budget\_id and name | Same as expected | PASSED |
| IT10 | Disallow duplicate budget names for same user | budget.test.js | Duplicate budget name | 400 status, error about duplicate budget | Same as expected | PASSED |
| IT11 | Retrieve all budgets for user | budget.test.js | User ID | 200 status, array of budgets | Same as expected | PASSED |
| IT12 | Create a new  transaction | transaction.test.js | Valid transaction data | 201 status, transaction object with transaction\_id | Same as expected | PASSED |
| IT13 | Fetch all transactions for user | transaction.test.js | User ID | 200 status, array of transactions | Same as expected | PASSED |
| IT14 | Delete a transaction | transaction.test.js | User ID,  Transaction  ID | 200 status, message "Transaction deleted successfully" | Same as expected | PASSED |
| UT01 | Generate a valid JWT and set as cookie | utils.unit.test.js | Response mock, User  ID | JWT  generated, cookie set with token | Same as expected | PASSED |

Table 5: Integration testing table

**5.6 TEST REPORT**

|  |  |
| --- | --- |
| Total Test Suites | 10 |
| Total Test Cases | 23 |
| Passed Test Cases | 23 |
| Failed Test Cases | 0 |

Table 6: Test Report

**6. RESULTS**

**6.1 Results Snippets**

After the full implementation and testing, the application was able to:

* Let users create and manage multiple budgets
* Track all expenses and income under selected categories
* Scan receipts and automatically convert them to transactions
* Show pie charts and bar graphs for budget usage
* Allow users to edit or delete transactions easily

The system worked well in both desktop and mobile views. The graphs clearly showed where money was being spent, and the receipt scanner saved a lot of time for users.

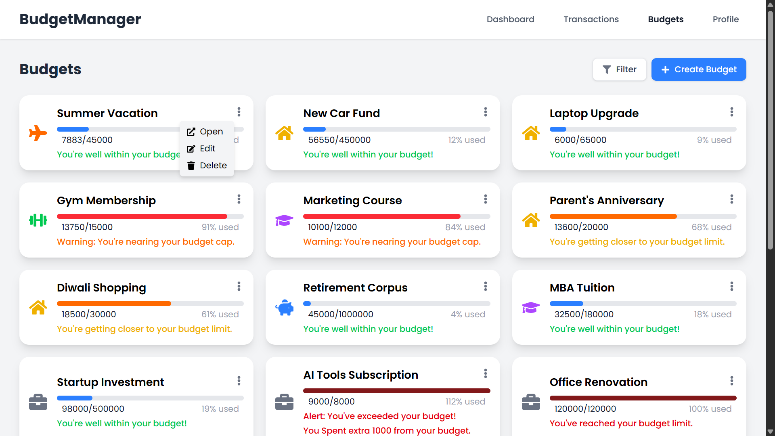
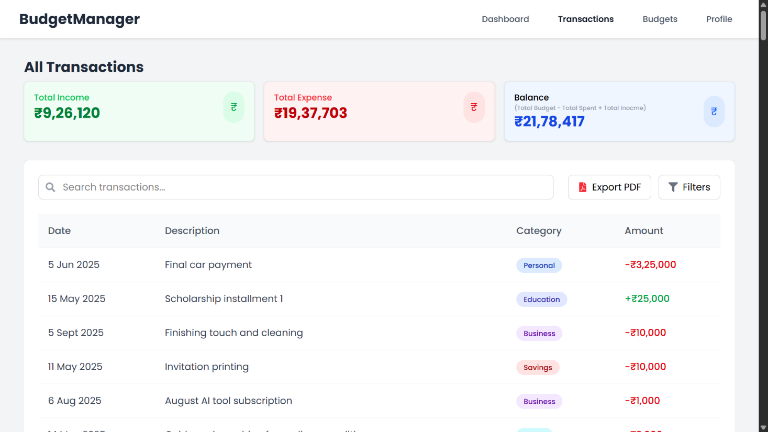
 

Figure 10: Budgets output Figure 11: AllTransactions output

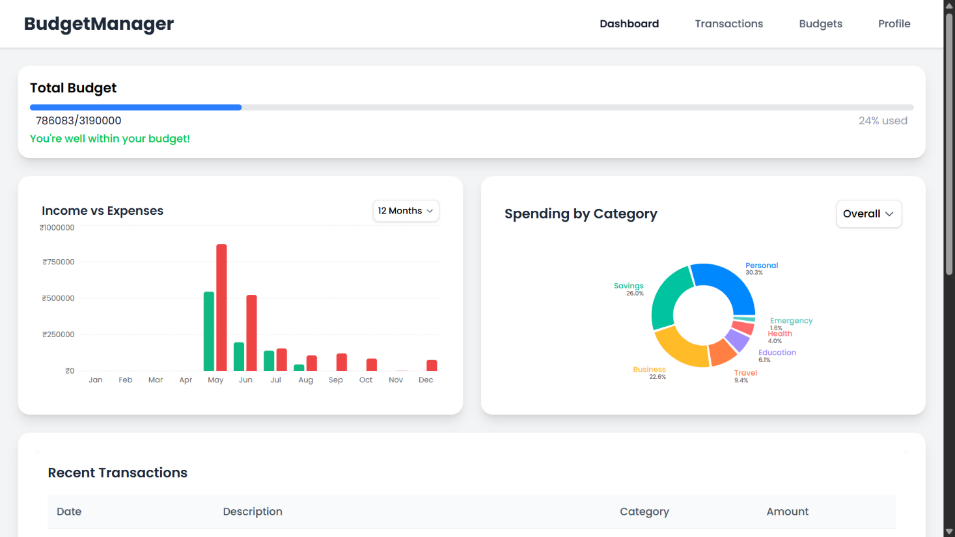


Figure 12: Dashboard output

**7. CONCLUSION AND FUTURE SCOPE**

This mini-project successfully achieved its main goal of helping users manage their personal budget smartly and easily. With AI support for receipt scanning, users don’t have to type their expenses manually. The project uses reliable technologies like React, Node.js, and Prisma which ensure good performance and security.

In the future, we plan to add features like:

* Linking bank accounts to fetch transactions directly
* Giving users monthly spending tips based on their habits
* Adding voice support to record expenses
* Creating a mobile app version for easier access

These improvements will make the system even more useful and smarter for real-world financial needs.

**8. REFERENCES**

[1] Verma, S., Kheda, S. S., & Kuwale, S. (2024). Personal finance tracker. *International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS), 6*(5), 10279-10288.

[2] Bhardwaj, S., Gupta, S., Jaiswal, U., & Bhargava, R. (2024). Personal expense tracker. *International Journal of Novel Research and Development (IJNRD), 9*(5), e927-e928.

[3] Bhatt, P., Nutheti, S. C., Mamidipaka, G., Kondapally, U. K., & Lokineni, H. (2024). Expense tracker: A smart approach to track daily expense. \*Tuijin Jishu/Journal of Propulsion Technology, 45\*(1), 5422-5424.

[4] Dewan, K., Lasar, A., & Bhokse, B. (2024). FinanceVUE - A MERN stack finance dashboard application. *International Journal of Novel Research and Development (IJNRD), 9*(4), b592-b593.

[5] Naik, A., Patel, D., Mourya, A., Mishra, V., & Mandavkar, M. (2024). Revenue manager app using MERN and ML. *International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS), 6*(4), 4407-4408.

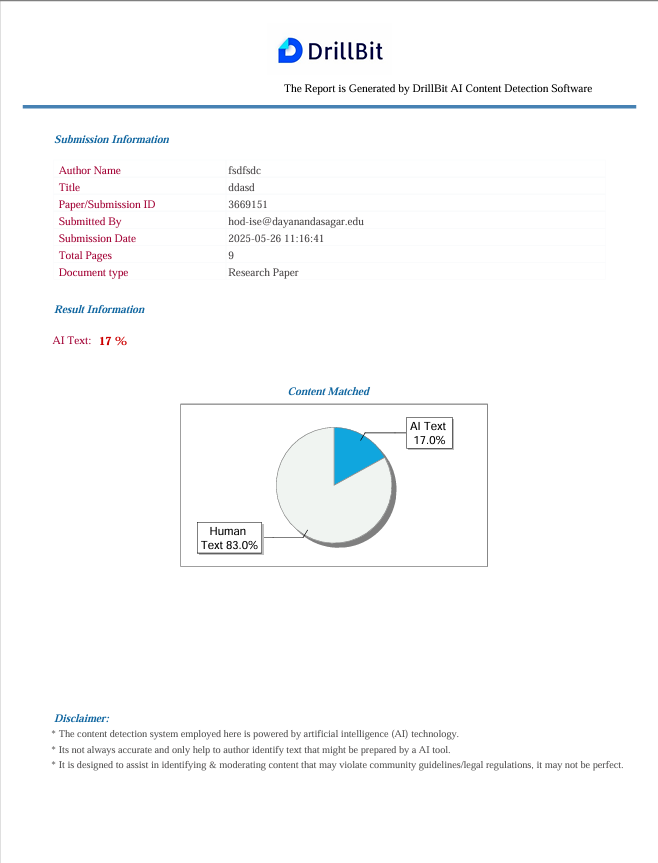
[6] B. M. K., Somvanshi, S., Kolhar, S. S., Gupta, S., & Haladi, S. R. (2024). Tracking the expense using MERN stack and data visualization. *International Journal of Recent Engineering Research and Development (IJRERD), 9*(2), 78-83.

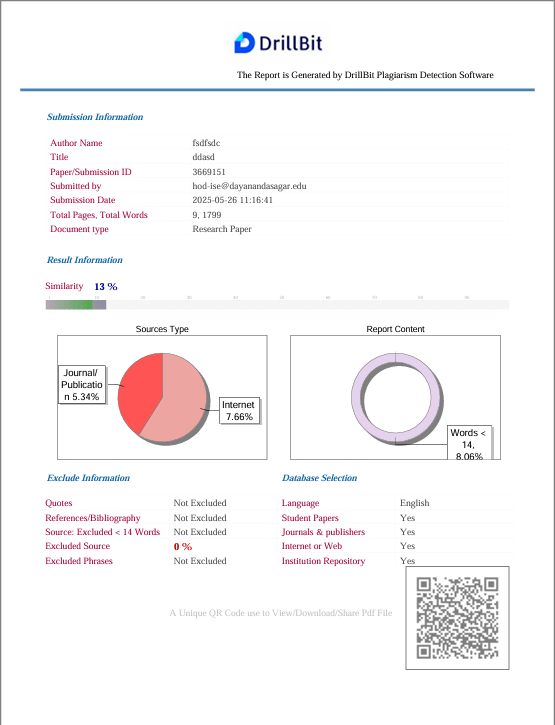
[7] Kumar, A., Verma, R., & Sinha, A. (2023). Budget manager. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), 3*(6), 390-392.

[8] Tran, P. (2023). *Expense tracker application using MERN stack* [Master's thesis]. Vaasan Ammattikorkeakoulu University of Applied Sciences.

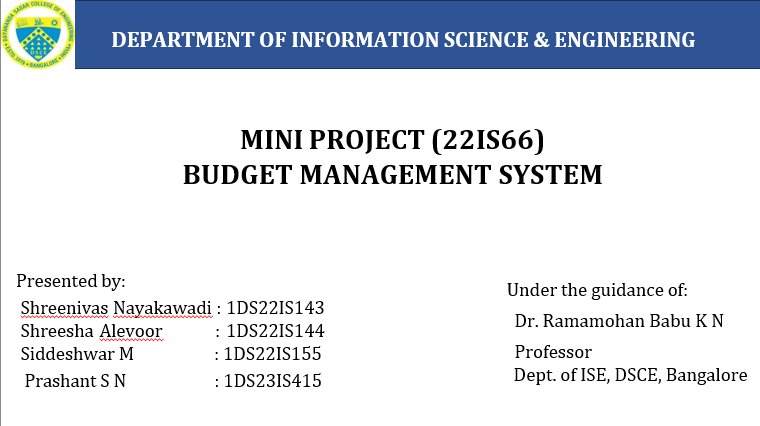
[9] Kritika, K., Himani, H., & Shikha, S. (2022). XPEN – A voice powered expense tracker full stack web application. *Bhagwan Parshuram Institute of Technology, BBJITM, 8*(2), 1-12.

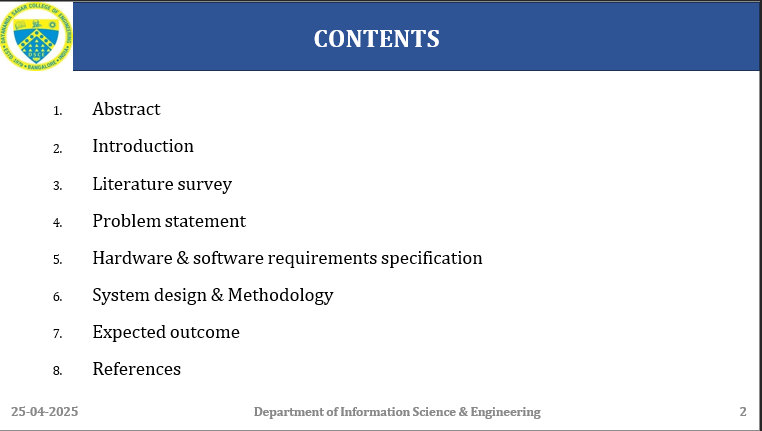
[10] Patel, R., Sharma, N., & Lee, J. (2023). *Real-time budget tracking using MERN stack with React Hooks and Firebase integration*. International Journal of Web Technologies, 8(2), 112-129.

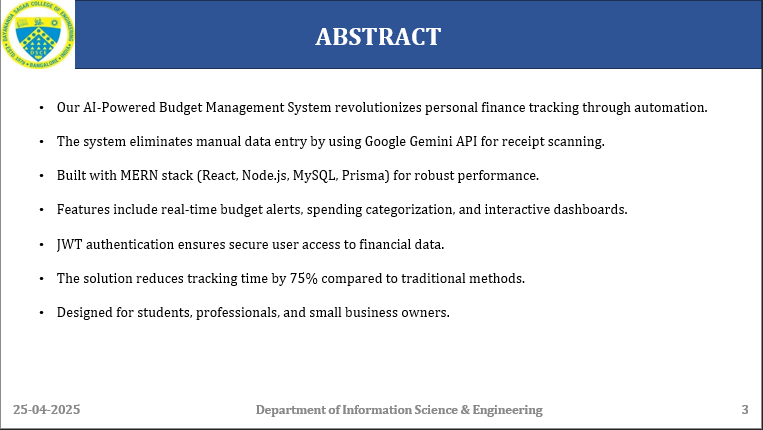


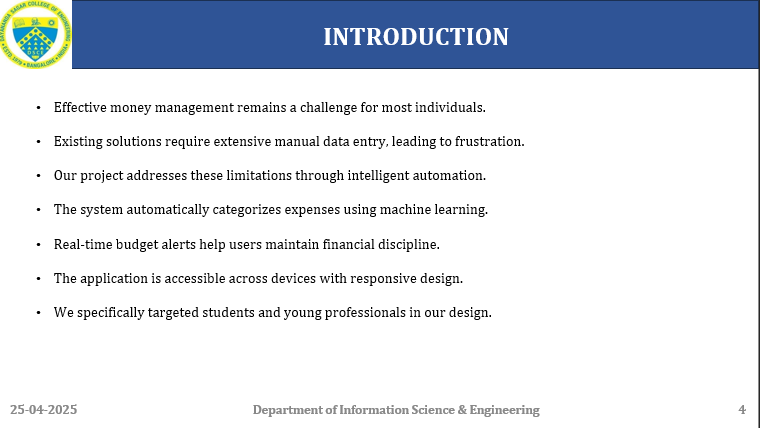


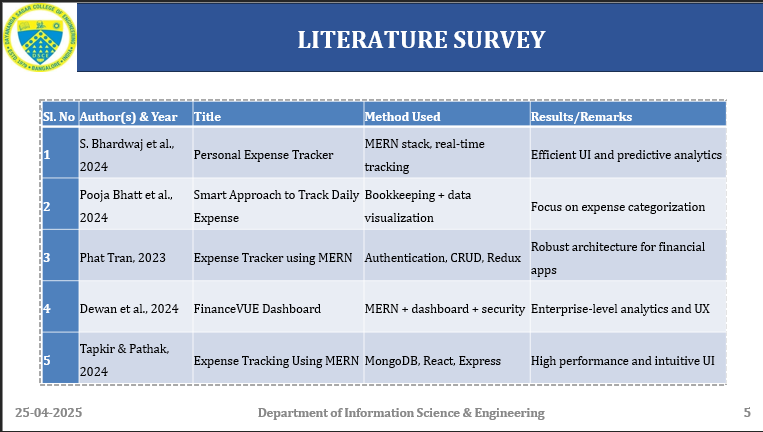
**PRESENTATION**

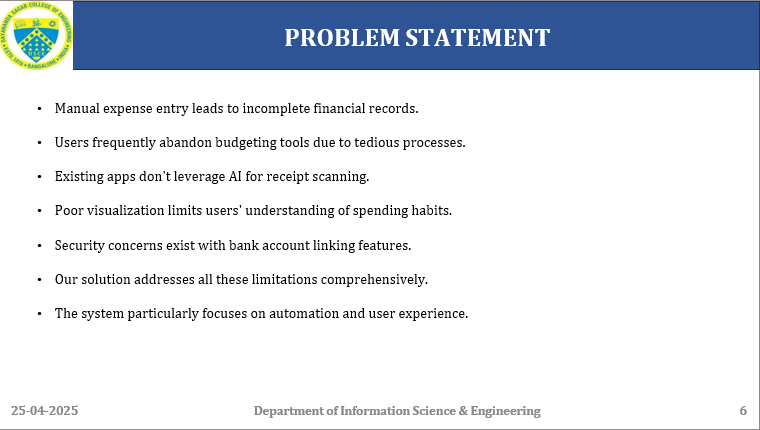
****

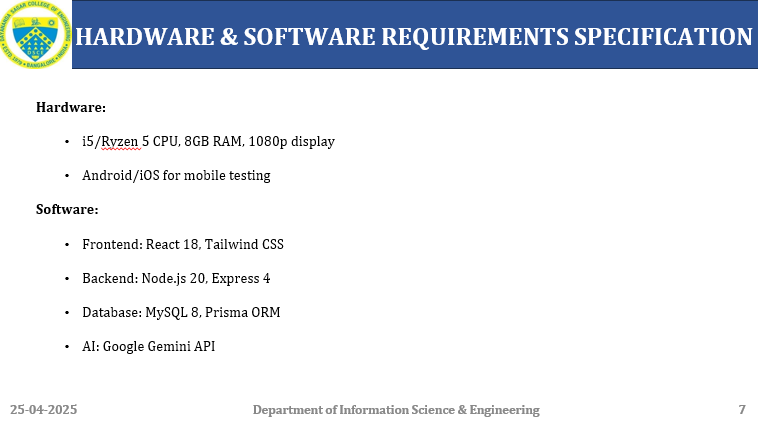
****

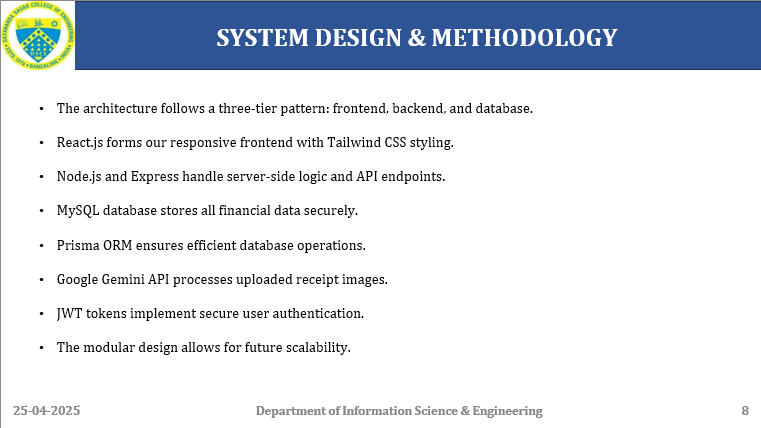
****

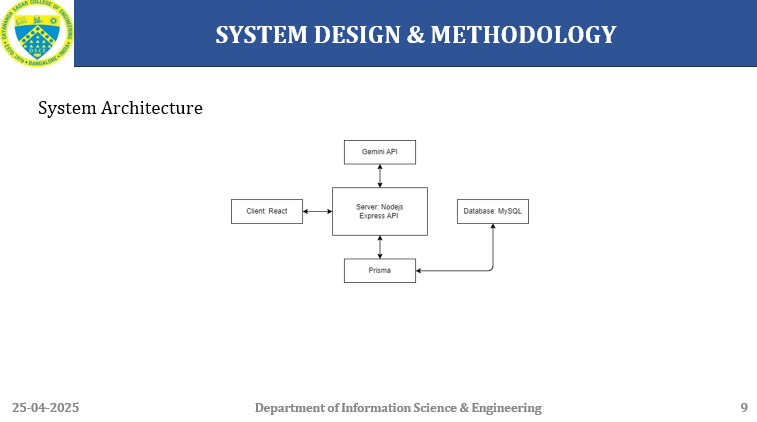
****

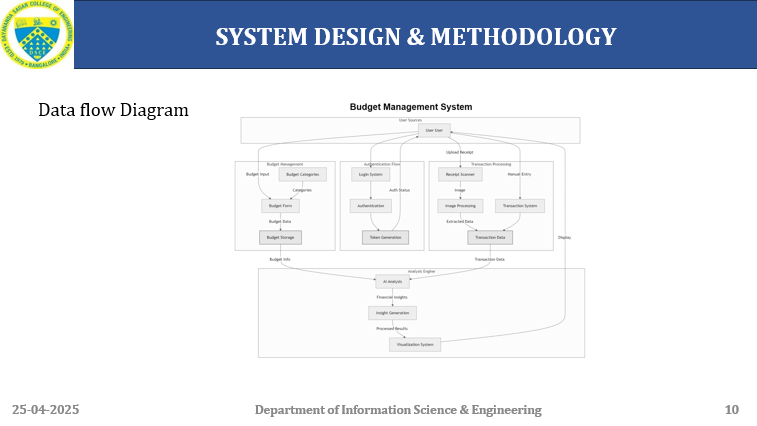
****

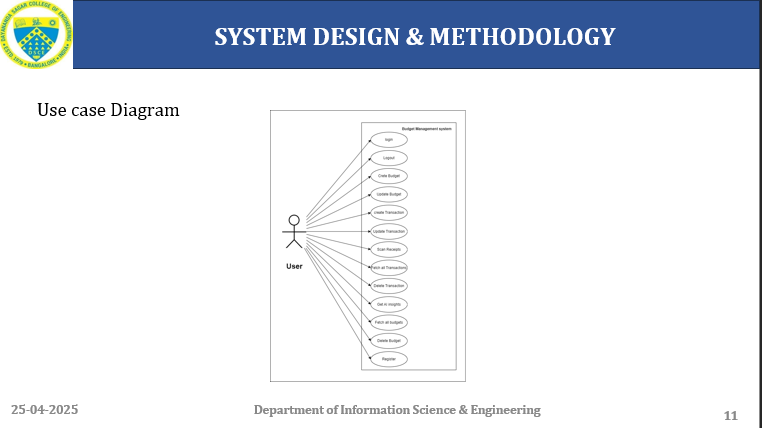
****

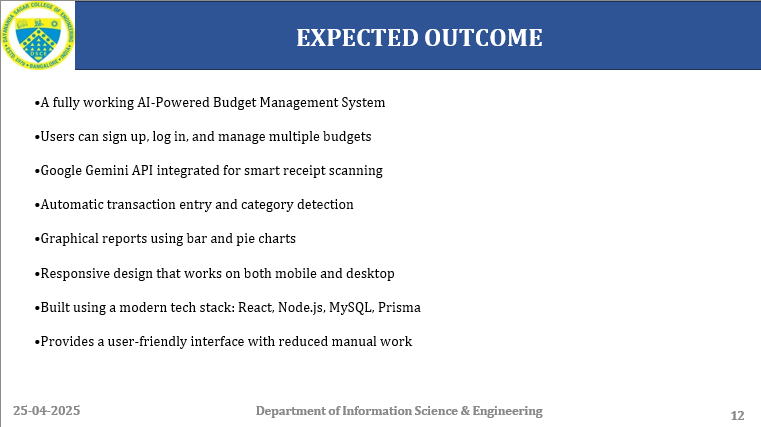
****

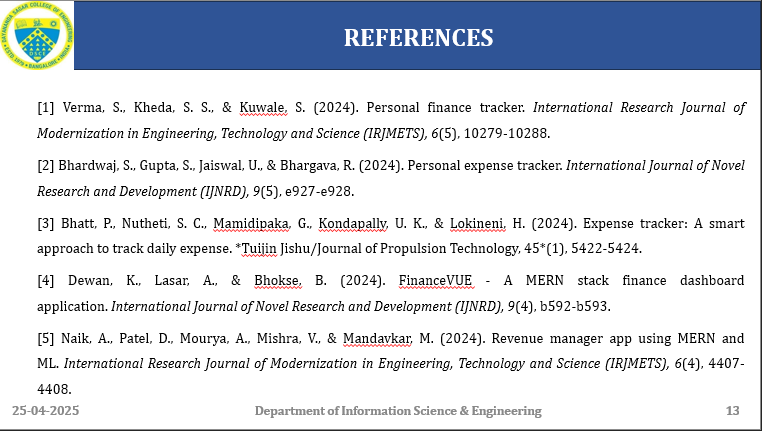
****

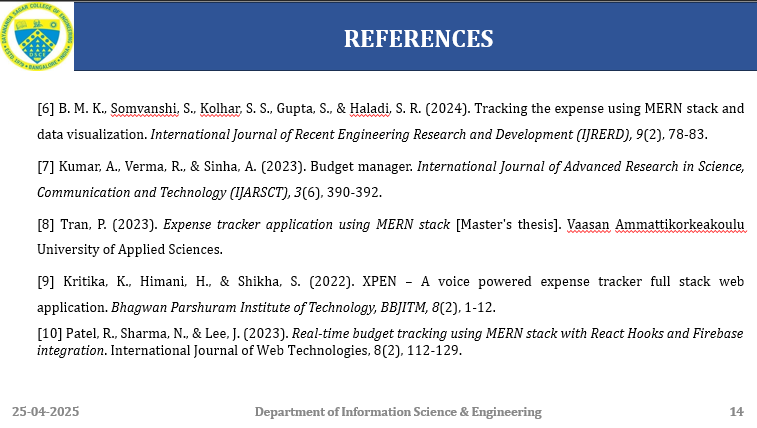
****

****

****

****

****

****