**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 19 th July 2024 |
| Team ID | SWTID1721040922 |
| Project Name | Project – WalletWatch – the expense tracker app |
| Maximum Marks |  |

**Project Title: WalletWatch-Expense Tracker app**

Date: 19th July 2024

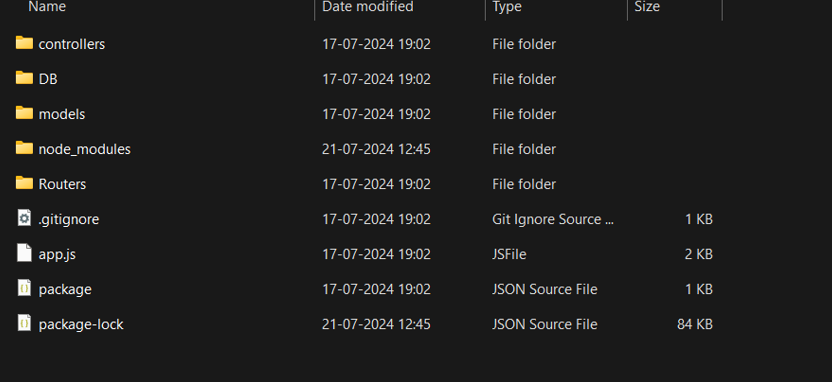
Prepared by: M.Madhuri Nagalakshmi,Rohan Singh

**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for theWalletwatch-The expense Tracker project.

**Technologies Used**

* **Backend Framework:** Node.js with Express.js
* **Database:** MongoDB
* **Authentication:** JWT

**Project Structure**  
Provide a screenshot of the backend project structure with explanations for key directories and files.



**app/**

* **\_\_init\_\_.py**: Initializes the Flask app and sets up the configurations, database, and blueprints for routes.
* **config.py**: Contains configuration settings for different environments (development, testing, production).

**app/models/**

* **\_\_init\_\_.py**: Imports all the models and initializes the database.
* **user.py**: Defines the User model with attributes like username, email, password, etc.
* **expense.py**: Defines the Expense model with attributes like amount, date, category\_id, user\_id, etc.
* **category.py**: Defines the Category model with attributes like name, description, user\_id, etc.
* **budget.py**: Defines the Budget model with attributes like amount, start\_date, end\_date, user\_id, etc.

**Key Directories and Files**

**app/routes/**

* **\_\_init\_\_.py**: Registers the blueprints for different routes.
* **auth.py**: Contains routes for user authentication (register, login, logout).
* **expenses.py**: Contains routes for managing expenses (create, read, update, delete).
* **categories.py**: Contains routes for managing categories (create, read, update, delete).
* **budgets.py**: Contains routes for managing budgets (create, read, update, delete).
* **reports.py**: Contains routes for generating reports.
* **recurring\_expenses.py**: Contains routes for managing recurring expenses.
* **notifications.py**: Contains routes for managing notifications.

**app/schemas/**

* **\_\_init\_\_.py**: Imports all the schemas.
* **user.py**: Defines schemas for user-related data validation.
* **expense.py**: Defines schemas for expense-related data validation.
* **category.py**: Defines schemas for category-related data validation.
* **budget.py**: Defines schemas for budget-related data validation.

**app/services/**

* **\_\_init\_\_.py**: Imports all the service modules.
* **auth\_service.py**: Contains business logic for authentication-related operations.
* **expense\_service.py**: Contains business logic for expense-related operations.
* **category\_service.py**: Contains business logic for category-related operations.
* **budget\_service.py**: Contains business logic for budget-related operations.

**app/utils/**

* **\_\_init\_\_.py**: Imports all the utility modules.
* **email.py**: Contains utility functions for sending emails.
* **validators.py**: Contains custom validators for data validation.

**app/main.py**

* The main entry point of the application where the Flask app is created and run.

**migrations/**

* Contains files related to database migrations (e.g., version files generated by Alembic).

**tests/**

* **\_\_init\_\_.py**: Imports all the test modules.
* **test\_auth.py**: Contains unit tests for authentication-related functionalities.
* **test\_expenses.py**: Contains unit tests for expense-related functionalities.
* **test\_categories.py**: Contains unit tests for category-related functionalities.
* **test\_budgets.py**: Contains unit tests for budget-related functionalities.

**Project Root**

* **.env**: Environment variables for configuration.
* **.gitignore**: Specifies files and directories to be ignored by Git.
* **requirements.txt**: Lists all the dependencies required for the project.
* **README.md**: Contains an overview and documentation of the project.

**API Endpoints**  
A summary of the main API endpoints and their purposes:

1. **Authentication Endpoints**:
   * **POST /api/register**: Register a new user.
   * **POST /api/login**: Log in a user and return an authentication token.
   * **POST /api/logout**: Log out the user and invalidate the authentication token.
   * **POST /api/password-reset**: Send a password reset link to the user's email.
   * **POST /api/password-reset/confirm**: Confirm the password reset with a new password.
2. **User Profile Endpoints**:
   * **GET /api/profile**: Retrieve the user profile information.
   * **PUT /api/profile**: Update user profile information.
   * **DELETE /api/profile**: Delete the user account.
3. **Expense Endpoints**:
   * **GET /api/expenses**: Retrieve a list of all expenses for the authenticated user.
   * **POST /api/expenses**: Add a new expense.
   * **GET /api/expenses/{id}**: Retrieve details of a specific expense by its ID.
   * **PUT /api/expenses/{id}**: Update an existing expense by its ID.
   * **DELETE /api/expenses/{id}**: Delete an expense by its ID.
4. **Category Endpoints**:
   * **GET /api/categories**: Retrieve a list of all expense categories.
   * **POST /api/categories**: Add a new expense category.
   * **GET /api/categories/{id}**: Retrieve details of a specific category by its ID.
   * **PUT /api/categories/{id}**: Update an existing category by its ID.
   * **DELETE /api/categories/{id}**: Delete a category by its ID.
5. **Budget Endpoints**:
   * **GET /api/budgets**: Retrieve a list of all budgets for the authenticated user.
   * **POST /api/budgets**: Add a new budget.
   * **GET /api/budgets/{id}**: Retrieve details of a specific budget by its ID.
   * **PUT /api/budgets/{id}**: Update an existing budget by its ID.
   * **DELETE /api/budgets/{id}**: Delete a budget by its ID.
6. **Report Endpoints**:
   * **GET /api/reports/summary**: Retrieve a summary report of expenses.
   * **GET /api/reports/category/{id}**: Retrieve a report of expenses for a specific category.
   * **GET /api/reports/date-range**: Retrieve a report of expenses within a specific date range.
7. **Recurring Expenses Endpoints**:
   * **GET /api/recurring-expenses**: Retrieve a list of all recurring expenses.
   * **POST /api/recurring-expenses**: Add a new recurring expense.
   * **GET /api/recurring-expenses/{id}**: Retrieve details of a specific recurring expense by its ID.
   * **PUT /api/recurring-expenses/{id}**: Update an existing recurring expense by its ID.
   * **DELETE /api/recurring-expenses/{id}**: Delete a recurring expense by its ID.
8. **Notifications Endpoints**:
   * **GET /api/notifications**: Retrieve a list of notifications for the authenticated user.
   * **POST /api/notifications**: Add a new notification.
   * **DELETE /api/notifications/{id}**: Delete a notification by its ID.

**Integration with Frontend**  
The backend communicates with the frontend via RESTful APIs. Key points of integration include:

* **User Authentication:** Tokens are passed between frontend and backend to handle authentication.
* **Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction.

**Error Handling and Validation**  
Describe the error handling strategy and validation mechanisms:

* **Error Handling:** Centralized error handling using middleware.
* **Validation:** Input validation using libraries like express-validator.

**Security Considerations**  
Outline the security measures implemented:

* **Authentication:** Secure token-based authentication.
* **Data Encryption:** Encrypt sensitive data at rest and in transit.