Expense Tracking System for Hostel Students

1. Introduction

Expense Tracking System for Hostel Students is specifically designed to provide intuitive tools that help students **track, categorize, and analyse** their daily expenses, ultimately promoting smarter financial habits. The platform enables efficient **expense recording and categorization** for both personal and shared spending, along with **interactive visualizations** that uncover spending patterns and trends. Through **insightful analytics**, the system empowers users to better manage their budgets and make informed financial decisions. Built on a robust and scalable tech stack—**Streamlit** for a lightweight and interactive frontend, **Fast API** for high-performance backend services, and **MySQL** for reliable and structured data storage—the solution is designed to be **scalable, maintainable, and user-friendly**, fostering financial awareness and responsibility among hostel students.

2. System Goals & Objectives

The core goals and objectives of the Expense Tracking System for Hostel Students are detailed below to ensure comprehensive functionality, performance, and user satisfaction:

Comprehensive Expense Management

- **Capture Expenses**: Allow students to record expenses with key details such as date, amount, category, and descriptive notes.
- **Modify Expenses**: Provide the ability to edit existing expense records to correct errors or update information.
- **Categorization**: Facilitate meaningful classification of expenses (e.g., Rent, Food, Shopping, Entertainment, Other) for more effective financial analysis.

Robust Data Persistence

- **Reliable Storage**: Securely store all expense data in a **relational database (MySQL)**.
- **Data Integrity**: Enforce data accuracy and consistency through **database constraints** and **backend validation** mechanisms.

Intuitive User Interface (UI)

- **Simplicity & Ease of Use**: Design a clean, minimalistic UI using **Streamlit**, ensuring effortless expense entry and navigation.
- **Visual Feedback**: Provide users with immediate feedback for successful operations or errors to enhance usability and confidence.

Powerful Data Analytics & Visualization

- **Category-wise Breakdown**: Display analytical summaries showing how expenses are distributed across various categories.
- **Month-wise Breakdown**: Visualize spending trends over time, enabling students to track

financial behaviour month-by-month.

- **Interactive Charts**: Use **bar charts** and other interactive elements to represent data in a user-friendly and engaging way.
- **Tabular Data**: Supplement visualizations with detailed **tables** showing totals, percentages, and breakdowns for deeper insights.

Scalability & Performance

- **Modular Architecture**: Adopt a layered design separating frontend (Streamlit), backend (FastAPI), and database (MySQL) to support independent development and scaling.
- **Efficient API Endpoints**: Build optimized endpoints using **FastAPI** to ensure rapid data processing and reduced latency.
- **Database Optimization**: Implement indexing and query tuning strategies in MySQL for high-performance data access and retrieval.

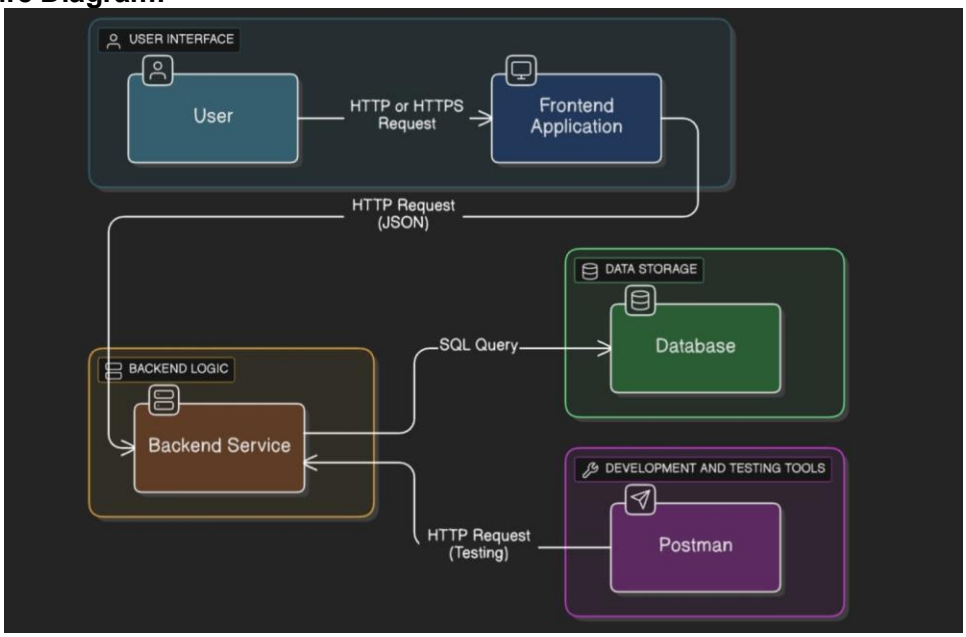
Security & Data Integrity

- **Input Validation**: Apply comprehensive server-side validation to filter out invalid or potentially harmful inputs.
- **Error Handling**: Ensure graceful error handling with clear feedback for users and actionable debugging information for developers.
- **Data Backup Strategy (Planned)**: Prepare for future integration of database backup solutions to safeguard against data loss.

Architecture Overview

The system adheres to a robust three-tier architecture, promoting separation of concerns, scalability, and maintainability.

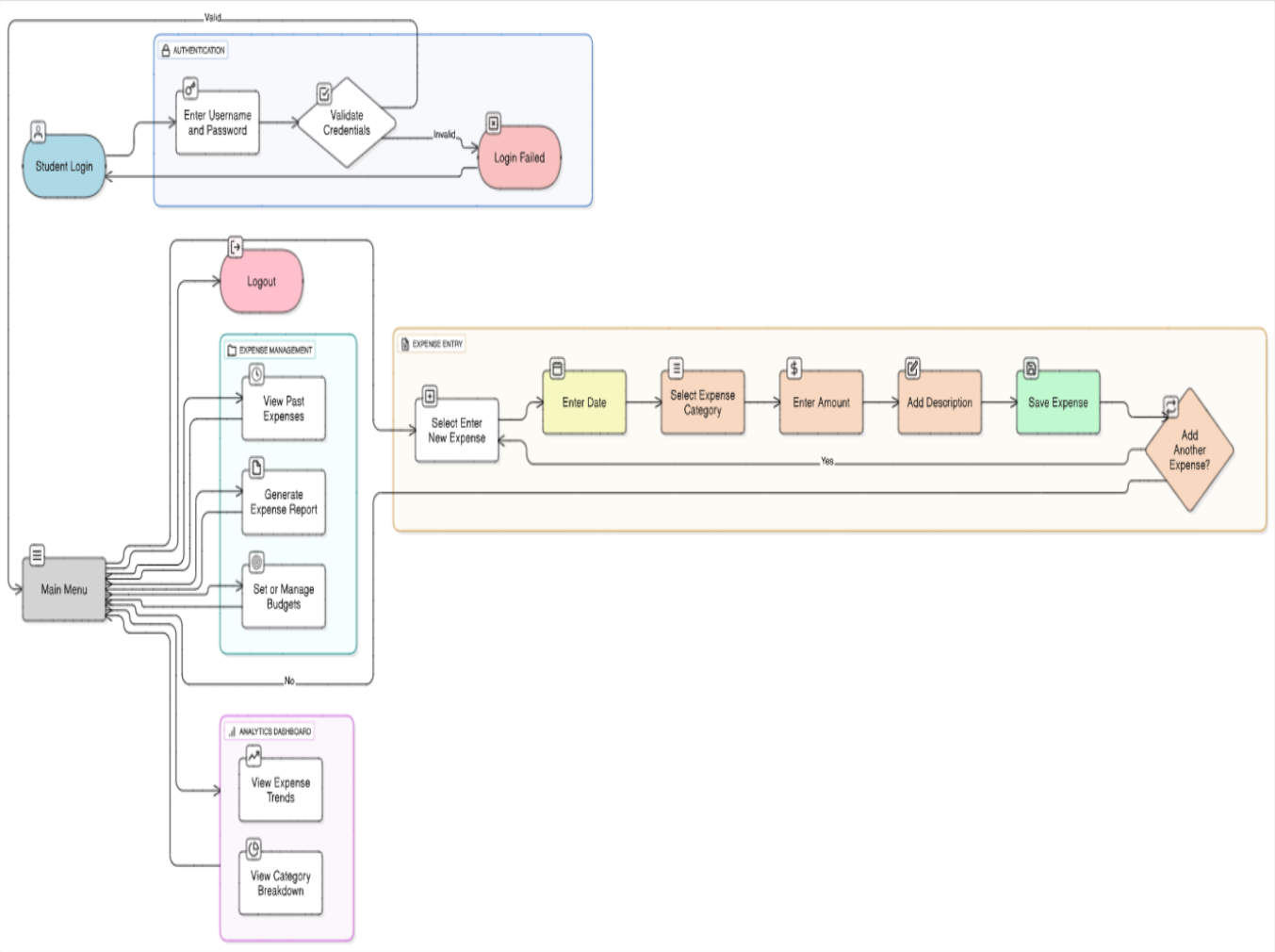
Architecture Diagram:



Technology Stack Justification:

- **Frontend (Streamlit):** Chosen for its rapid development capabilities, ease of use for creating interactive web applications with data visualizations, and Python-native environment, aligning with data analytics goals. Its simplicity allows developers to focus on functionality rather than complex web development frameworks.
- **Backend (FastAPI):** Selected for its high performance (comparable to NodeJS and Go), asynchronous capabilities, automatic interactive API documentation (Swagger UI/ReDoc), and strong type hinting with Pydantic. It's ideal for building efficient and maintainable RESTful APIs.
- **Database (MySQL):** A widely adopted, open-source relational database known for its reliability, robust feature set, scalability, and strong community support. It's well-suited for structured data like expense records.
- **API Testing/Development (Postman):** An industry-standard tool for testing REST APIs, allowing developers to send requests, inspect responses, and automate API testing during development.

WEB APPLICATION OVERVIEW:



3.Detailed Components

3.1 Presentation Layer (Streamlit Application)

The Streamlit application will be the user-facing component, providing a clean and responsive interface.

- **Core Pages/Modules:**
  - **"Add/Update" Expense Form:**
    - **Date Picker:** For selecting the expense date (defaulting to current date).
    - **Amount Input:** Numeric input field for the expense amount.
    - **Category Dropdown:** Pre-defined categories (e.g., Rent, Food, Shopping, Entertainment, Other) to ensure consistency.
    - **Notes Text Area:** Optional free-form text field for detailed descriptions.
    - **Add/Submit Button:** Triggers the API call to save or update the expense.
    - **Dynamic List/Table:** Displays recently added or all expenses with options to edit or delete individual records (leveraging Streamlit's data editor or custom components).
    - **Success/Error Messages:** Clear feedback messages displayed after submission (e.g., "Expenses updated successfully!").
  - **"Analytics By Category" View:**
    - **Date Range Selectors:** Two date pickers (Start Date and End Date) to define the period for analysis.
    - **"Get Analytics" Button:** Triggers the API call to fetch categorized expense data.
    - **Bar Chart (Plotly/Matplotlib integrated with Streamlit):** Visual representation of total spending per category.
      - X-axis: Expense Categories.
      - Y-axis: Total Amount Spent.
      - Bars ordered by amount (descending).
    - **Summary Table:** A data table displaying:
      - Category: Name of the expense category.
      - Total: Sum of expenses for that category.
      - Percentage: Proportion of that category's spending relative to the total for the selected period.
  - **"Analytics By Months" View:**
    - **Automatic Period Selection:** Automatically fetches data for recent months (e.g., last 3-6 months) or allows a user to define a custom year.
    - **Bar Chart (Plotly/Matplotlib):** Visual representation of total spending per month.
      - X-axis: Month Name (e.g., August, September, October).
      - Y-axis: Total Amount Spent.
    - **Summary Table:** A data table displaying:
      - Month Number: Numerical representation of the month.
      - Month Name: Full name of the month.
      - Total: Sum of expenses for that month.
- **Navigation:** Clear tab-based or button-based navigation to switch between the "Add/Update," "Analytics By Category," and "Analytics By Months" sections.

3.2 Application Layer (FastAPI Backend)

The FastAPI application will be the brain of the system, handling all business logic

and orchestrating data flow.

- **API Endpoints (Detailed):**
  - **Expense Management:**
    - POST /expenses:
      - **Request Body:** {"date": "YYYY-MM-DD", "amount": 123.45, "category": "Food", "notes": "Lunch at canteen"}
      - **Validation:** Ensure date is valid, amount is positive number, category is from a predefined list, notes is string.
      - **Action:** Inserts a new record into the expenses table.
      - **Response:** {"message": "Expense added successfully!", "id": 123} or error details.
    - PUT /expenses/{expense_id}:
      - **Path Parameter:** expense_id (integer)
      - **Request Body:** {"date": "YYYY-MM-DD", "amount": 123.45, "category": "Food", "notes": "Updated note"} (partial updates allowed)
      - **Validation:** As above, plus expense_id existence check.
      - **Action:** Updates the corresponding record in the expenses table.
      - **Response:** {"message": "Expense updated successfully!"} or error details.
    - GET /expenses:
      - **Query Parameters (Optional):** start_date=YYYY-MM-DD, end_date=YYYY-MM-DD, category=Food
      - **Action:** Retrieves expense records based on filters.
      - **Response:** List of expense objects [{"id": 1, "date": "...", "amount": ..., "category": "...", "notes": "..."}, ...].
    - GET /expenses/{expense_id}:
      - **Path Parameter:** expense_id (integer)
      - **Action:** Retrieves a single expense record.
      - **Response:** {"id": 1, "date": "...", "amount": ..., "category": "...", "notes": "..."} or 404 if not found.
    - DELETE /expenses/{expense_id}:
      - **Path Parameter:** expense_id (integer)
      - **Action:** Deletes the specified expense record.
      - **Response:** {"message": "Expense deleted successfully!"} or error details.
  - **Analytics Endpoints:**
    - GET /analytics/category:
      - **Query Parameters:** start_date=YYYY-MM-DD, end_date=YYYY-MM-DD (required)
      - **Action:** Queries expenses table, groups by category, sums amount, calculates percentage for each category within the date range.
      - **Response:** [{"category": "Rent", "total": 9850.00, "percentage": 80.70}, {"category": "Food", "total": 1015.00, "percentage": 8.32}, ...]
    - GET /analytics/month:
      - **Query Parameters (Optional):** year=YYYY (default to current year)
      - **Action:** Queries expenses table, groups by month and year, sums amount.

        - **Response:** [{"month_number": 8, "month_name": "August", "total": 12205.00}, {"month_number": 9, "month_name": "September", "total": 4750.00}, ...]
- **Data Models (Pydantic):** FastAPI will utilize Pydantic models for request and response validation, ensuring data consistency and providing automatic documentation.
- **Database Interaction Layer (SQLAlchemy ORM or raw SQL):** While not strictly ORM-heavy for this scale, using a lightweight ORM like SQLAlchemy Core or direct mysql.connector with prepared statements will be employed for secure and efficient database interactions.
- **Dependency Injection:** FastAPI's dependency injection system will be used for managing database connections and other reusable components.

3.3 Data Layer (MySQL Database)

The MySQL database will be the persistent storage for all application data.

3.3.1 Database Schema (Detailed):

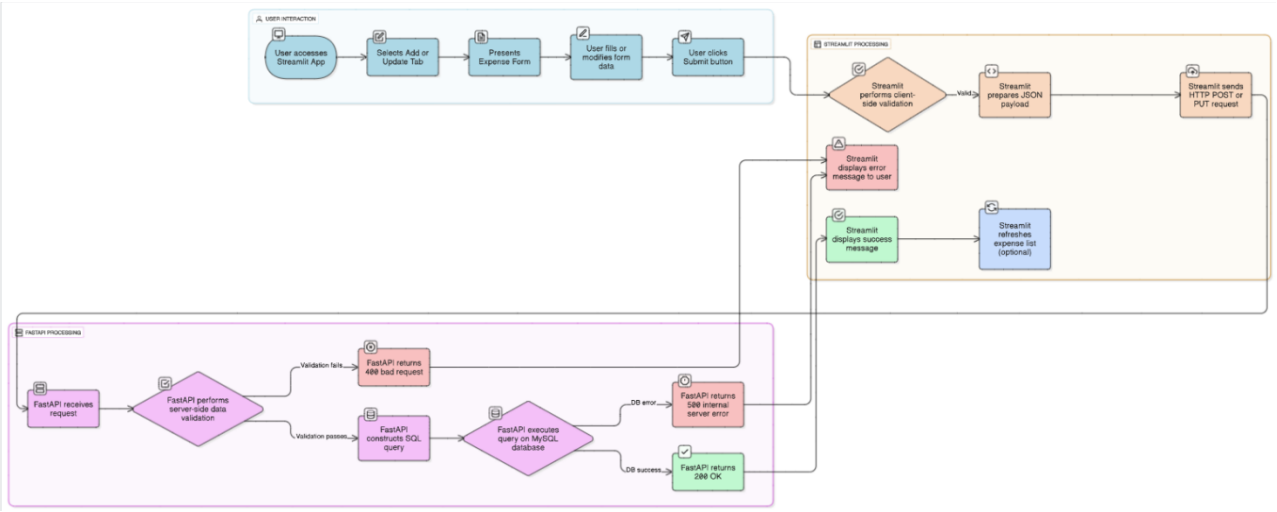
```
CREATE DATABASE IF NOT EXISTS `expense_tracker`;

USE `expense_tracker`;

CREATE TABLE IF NOT EXISTS `expenses` (
    `id` INT AUTO_INCREMENT PRIMARY KEY,
    `date` DATE NOT NULL COMMENT 'Date when the expense occurred',
    `amount` DECIMAL(10, 2) NOT NULL COMMENT 'Amount of the
expense, up to 2 decimal places',
    `category` VARCHAR(50) NOT NULL COMMENT 'Category of the
expense (e.g., Rent, Food, Shopping)',
    `notes` TEXT NULL COMMENT 'Optional detailed notes about the
expense',
    `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT
'Timestamp when the record was created',
    `updated_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT 'Timestamp of the last update to the
record'
);

-- Indexes for performance
CREATE INDEX idx_expenses_date ON `expenses` (`date`);
CREATE INDEX idx_expenses_category ON `expenses` (`category`);
```

3.3.2 **Data Consistency:** NOT NULL constraints ensure essential data is always present. DECIMAL(10,2) ensures accurate financial calculations.

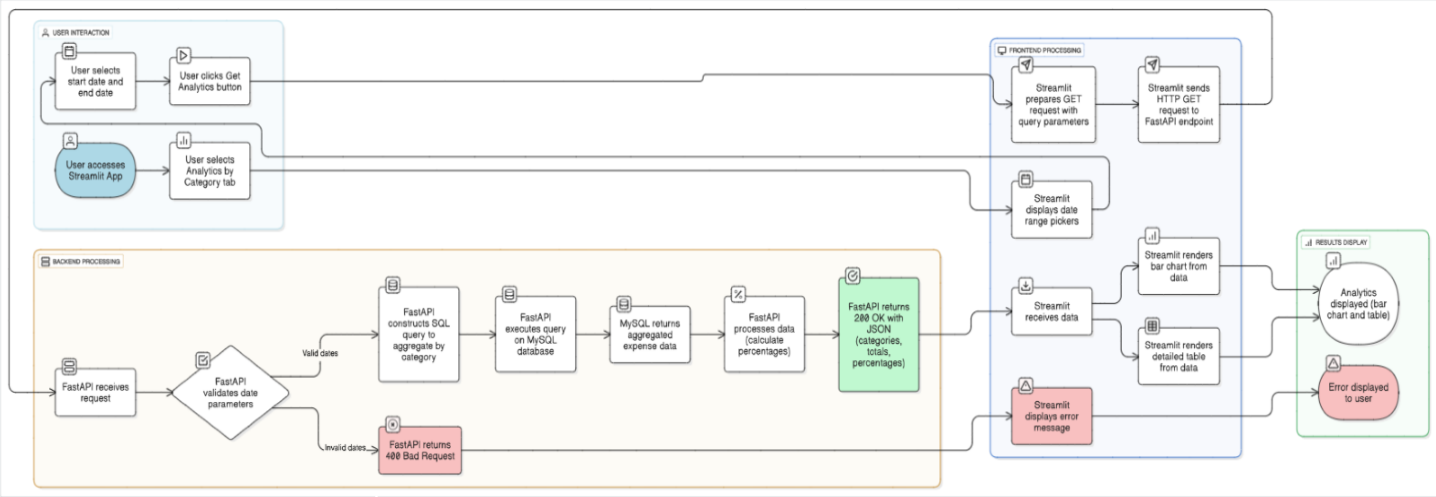
3.3.3 **Scalability Considerations:** Proper indexing (idx_expenses_date, idx_expenses_category) will be crucial for optimizing queries related to date ranges and category-based analytics, especially as data grows.

4 Detailed System Flowchart

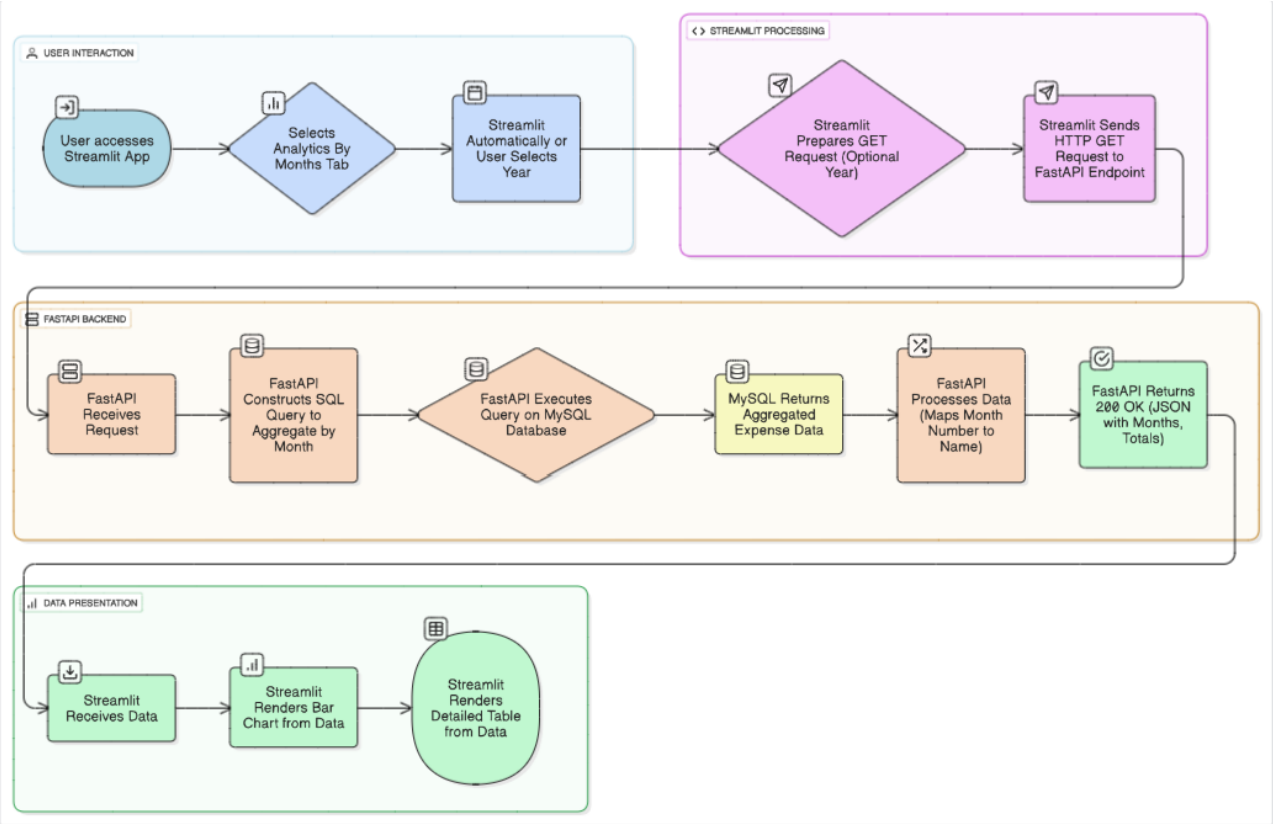
Expense Addition/Update Flow



4.1Analytics by Category Flow



4.2Analytics by Month Flow



5 Data Flow (Detailed)

1. **User Input (Streamlit):** A student interacts with the Streamlit UI, entering expense details into forms or selecting date ranges for analytics.
2. **Frontend Request (Streamlit to FastAPI):** Upon user action (e.g., clicking "Submit," "Get Analytics"), Streamlit constructs an appropriate HTTP request (POST, PUT, GET) containing the relevant data (as JSON in request body or URL parameters). This request is sent to the FastAPI backend.
3. **Backend Request Handling (FastAPI):** FastAPI receives the incoming HTTP request.
    - **Routing:** The request is directed to the correct API endpoint based on its URL path and HTTP method.
    - **Validation (Pydantic):** FastAPI, using Pydantic models, automatically validates the incoming data against predefined schemas. If validation fails, an immediate error response is sent back to Streamlit.
    - **Business Logic:** If validation passes, FastAPI executes the associated business logic. This involves preparing data for the database, performing calculations (e.g., for analytics), or orchestrating database operations.
4. **Database Interaction (FastAPI to MySQL):** FastAPI connects to the MySQL database.
    - **Query Construction:** SQL queries (INSERT, SELECT, UPDATE, DELETE) are

dynamically constructed based on the request and business logic. Parameterized queries are used to prevent SQL injection vulnerabilities.
    - **Query Execution:** FastAPI sends the SQL query to the MySQL server.
5. **Data Retrieval/Modification (MySQL):** MySQL processes the SQL query, retrieves or modifies data, and sends the result set or status back to FastAPI.
6. **Backend Response Generation (FastAPI):** FastAPI receives the data/status from MySQL.
    - **Data Transformation:** For analytics, raw database results are further processed (e.g., summing, calculating percentages) into a format suitable for the frontend.
    - **JSON Response:** FastAPI constructs an HTTP response, typically with a JSON body containing the requested data or a success/error message, and an appropriate HTTP status code (e.g., 200 OK, 201 Created, 404 Not Found, 500 Internal Server Error).
7. **Frontend Data Rendering (Streamlit):** Streamlit receives the HTTP response from FastAPI.
    - **Error Handling:** If an error response is received, Streamlit displays a user-friendly error message.
    - **Data Visualization:** If data is received successfully, Streamlit uses its capabilities (or integrates with libraries like Plotly/Matplotlib) to render interactive charts and populate data tables, presenting the information visually to the user.
    - **UI Update:** The UI is updated to reflect the new state (e.g., "Expenses updated successfully!").

6 Future Enhancements

These future enhancements will significantly increase the system's value and robustness:

- **User Authentication & Authorization:**
    - **User Accounts:** Implement a user registration and login system.
    - **Session Management:** Use secure session tokens (e.g., JWT) to maintain user sessions.
    - **Personalized Data:** Ensure each user can only view and manage their own expense data.
- **Recurring Expenses Management:**
    - **Definition:** Allow users to define expenses that occur regularly (e.g., monthly rent, weekly allowance).
    - **Automation:** Automatically generate these expenses on their due dates.
    - **Notifications:** Send reminders for upcoming recurring payments.
- **Budgeting Feature:**
    - **Budget Creation:** Enable users to set monthly or weekly budgets for specific categories.
    - **Budget Tracking:** Visual dashboards showing spending against allocated budgets.
    - **Alerts:** Notify users when they are approaching or exceeding their budget limits.
- **Data Export Functionality:**
    - **Formats:** Provide options to export expense data in common formats like CSV, PDF, or Excel, facilitating external analysis or record-keeping.
- **Notifications System:**
    - **In-App Alerts:** Pop-up notifications for successful operations, budget alerts, or upcoming expenses.

    - **Email/SMS Integration:** (Advanced) Integrate with email or SMS services for out-of-app notifications for critical alerts.
- **Advanced Analytics:**
    - **Trend Analysis:** Identify spending trends over longer periods.
    - **Comparison:** Allow comparison of spending across different months or categories.
    - **Custom Reports:** Enable users to generate custom reports based on various criteria.
- **Search and Filter:**
    - **Full-text Search:** Ability to search notes or categories for specific keywords.
    - **Advanced Filters:** More granular filtering options for expense lists (e.g., by amount range).

7 Conclusion

This detailed HLD document provides a comprehensive blueprint for the "Student Expense Tracking System with Data Analytics." By meticulously outlining the architecture, components, data flow, and future enhancements, it serves as a robust guide for the development team. The strategic choice of Streamlit, FastAPI, and MySQL ensures a powerful, user-friendly, and scalable application that will significantly aid hostel students in managing their finances effectively. This systematic approach will streamline the development process and lead to a high-quality, impactful product.