Ghar Ki Bachat – Hackathon Documentation

Team Name: Nexora

GitHub Repository: https://github.com/pushpeshpant02/Ghar-Ki-Bachat

Team Members And Roles:
1.Pushpesh Pant : Backend
2.Palak Pandey : Frontend
3.Manas Dobal: Database
4.Rohit Kumar Rathore: AI Models

1. Overview

Ghar-Ki-Bachat is a personal finance tracking system that allows users to add expenses, categorize transactions, view insights, and receive alerts about their financial habits. The system is designed to be fast, simple to use, and highly reliable, with built-in scalability and failure-handling capabilities**.**
PROTOTYPE WEBSITE LINK: https://pushpeshpant02.github.io/Ghar-Ki-Bachat/

2. System Architecture

The architecture follows a modular, layered approach to keep the application flexible, easy to maintain, and efficient.

2.1 Frontend Layer

- Handles user input (adding expenses, editing entries, viewing dashboards).
- Performs basic validation before sending data to the backend.
- Stores temporary data using local storage when offline.

2.2 Backend Layer

- Processes requests from the frontend.
- Applies business logic such as categorization, filtering, and validation.
- Ensures transactions are stored safely in the database.
- Handles authentication and user management.

2.3 Database Layer

- Stores all expense records, categories, budgets, and user profiles.
- Ensures fast retrieval and analytics-friendly structure.
- Maintains data consistency using ACID transactions.

2.4 AI Insight Layer

- Analyzes user expenses to find spending patterns.
- Generates alerts, cost-saving suggestions, and category breakdowns.
- Uses rule-based logic and ML-ready structure for future enhancements.

2.5 Analytics and Dashboard Layer

- Aggregates data from the database.
- Displays monthly reports, category charts, and spending trends.
- Provides alerts when spending crosses budget thresholds.

3. Data Flow Diagram (Explanation)

The system follows this clean data flow:

1. User Input
   The user adds an expense (amount, category, notes).

2. Categorization
   The system categorizes the expense automatically or uses a predefined rule.

3. Database Storage
   The validated data is stored safely in the database using transactions.

4. AI Insight Layer
   Data is analyzed to detect trends or generate alerts.

5. Dashboard + Alerts
   Insights are shown immediately as charts, summaries, and warnings.

4. Scalability Strategy

If the system grows from a few hundred users to one million users, it should continue working smoothly without slowing down or crashing. To achieve this, we follow these steps:

1. **Split the system into smaller parts**
   Instead of building one big application that does everything, we divide features into separate modules.
   For example: expenses, income, budgets, goals, advisory.
   Each part can grow independently without affecting others.

2. **Add more servers when needed**
   If one server receives too many requests, it will become slow.
   To prevent this, we can add more servers and divide the traffic between them.
   This way no single machine gets overloaded.

3. **Use cloud services**
   Cloud providers like AWS, Azure or Google Cloud automatically give more CPU, RAM, and storage when traffic increases.
   This means the system can grow without manually upgrading hardware.

4. **Store data efficiently**
   We optimize the database by indexing common fields like user_id, category, date.
   This allows the database to fetch data much faster, even when millions of records exist.

5. **Cache frequently used results**
   Some results do not change often, such as monthly totals.
   Instead of calculating them every time, we store them temporarily in fast memory.
   This reduces load on the database.

6. **Keep the backend stateless**
   The backend should not store user session data in memory.
   This allows any server to handle any user request.
   It makes it easy to add or remove servers without confusion.

7. **Use background jobs for heavy tasks**
   Time-consuming processes like analytics or report generation should run in the background, not during user requests.
   This keeps the main app fast and responsive.

8. **Maintain backup and failover systems**
   If one server or database fails, another one should take over automatically.
   This ensures the system keeps running without downtime.

5. Failure Handling

To ensure smooth functioning, the system incorporates multiple layers of protection.

1.**Server becomes slow or stops working**

Challenge:Too many people use the app at the same time and the server becomes slow or crashes.
Solution:We keep backup servers ready. If one server stops, the other one starts working automatically. This way the app keeps running.

2. Internet connection problem

The user's internet may stop while they are entering data.
Solution:
The app saves the data in the mobile or browser temporarily. When the internet comes back, the app sends the saved data to the server automatically.

3. Data not saved properly

Challenge: Sometimes the app may save only half data if something goes wrong in between, which can corrupt the record.
Solution: We use a method where data is saved fully or not saved at all. This keeps the database clean and avoids mistakes.

4. AI gives wrong or confusing suggestions

Challenge: The AI might misunderstand the data and show wrong advice.
Solution: We check the AI output. If it looks wrong or incomplete, we show a simple message like "Try again later" instead of giving wrong advice.

5. App crashes because of bugs

Challenge: Sometimes there are coding mistakes that can crash the app.
Solution: We add error checking so that if something unexpected happens, the app does not crash. It just shows a safe error message and keeps running.

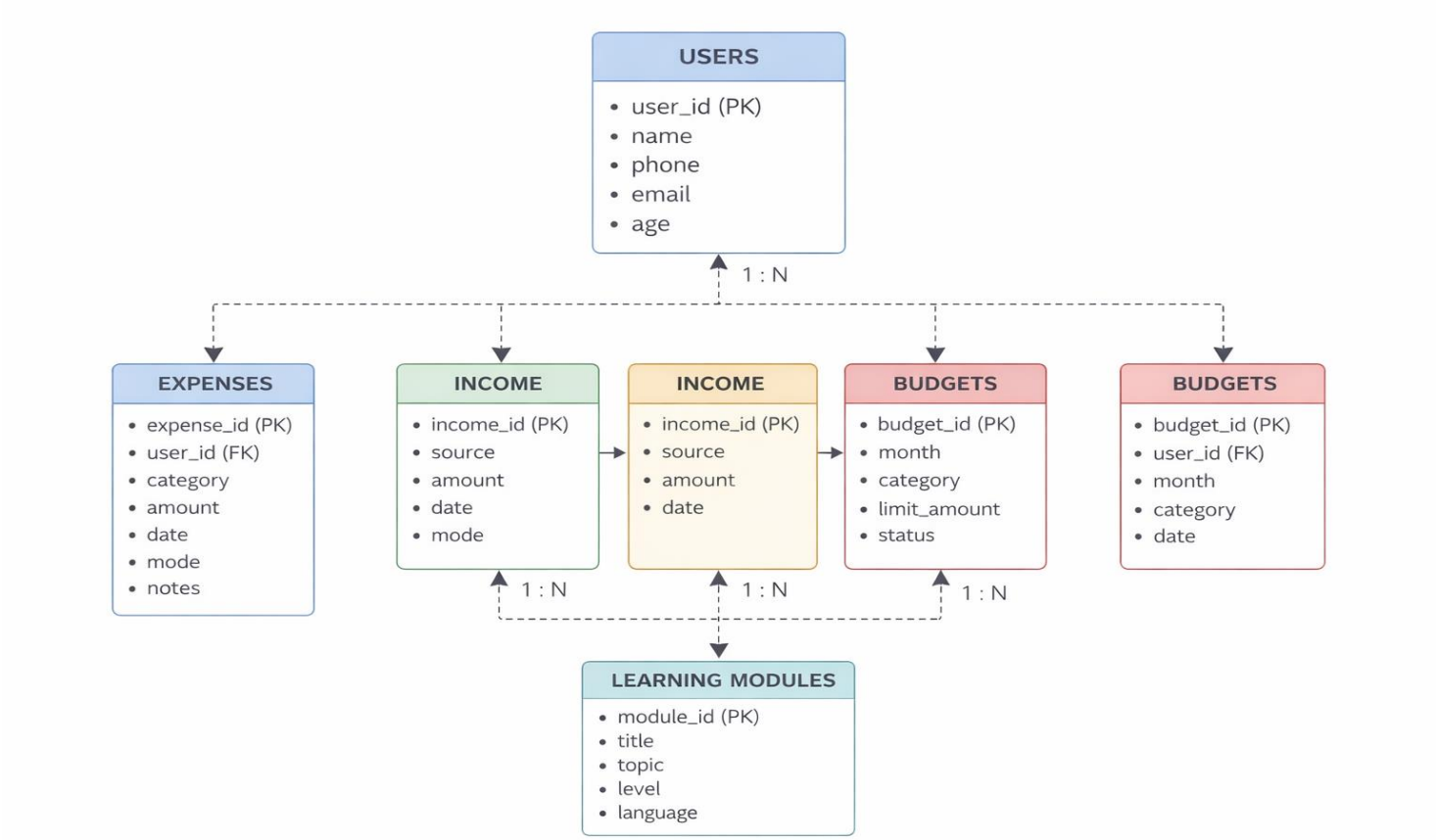
6. Database stops responding

Challenge: Database might become slow or stop due to high load.
Solution: We use another database in backup. When the main one fails, the backup starts working automatically so users do not notice anything.

7. Files or data get deleted by mistake

Challenge: A big problem is losing data because of accidental deletion or system failure.
Solution: We take backups every day and also every hour for important information. If something goes wrong, we can restore the latest backup.

6. Database Structure



This diagram shows how all the data in the system is organized and connected.

1. Users Table (Main Table)
All other data belongs to a user.
Each user has:

- user_id
- name
- phone
- email
- age

This is the parent table.

2. Expenses
Every user can have many expense entries.
Each expense saves:

- category
- amount
- date
- payment mode
- notes

3. Income
Each user can have multiple income records.
It stores:

- income source
- amount
- date
- mode

4. Budgets
Users can set different budgets for different categories.
Each entry tracks:

- month
- category
- limit amount

5. Goals
Users can set saving/financial goals.
Each goal stores:

- goal name
- target amount
- timeline
- status

6. Advisory Logs
This table stores what advice the AI gives to each user.
It includes:

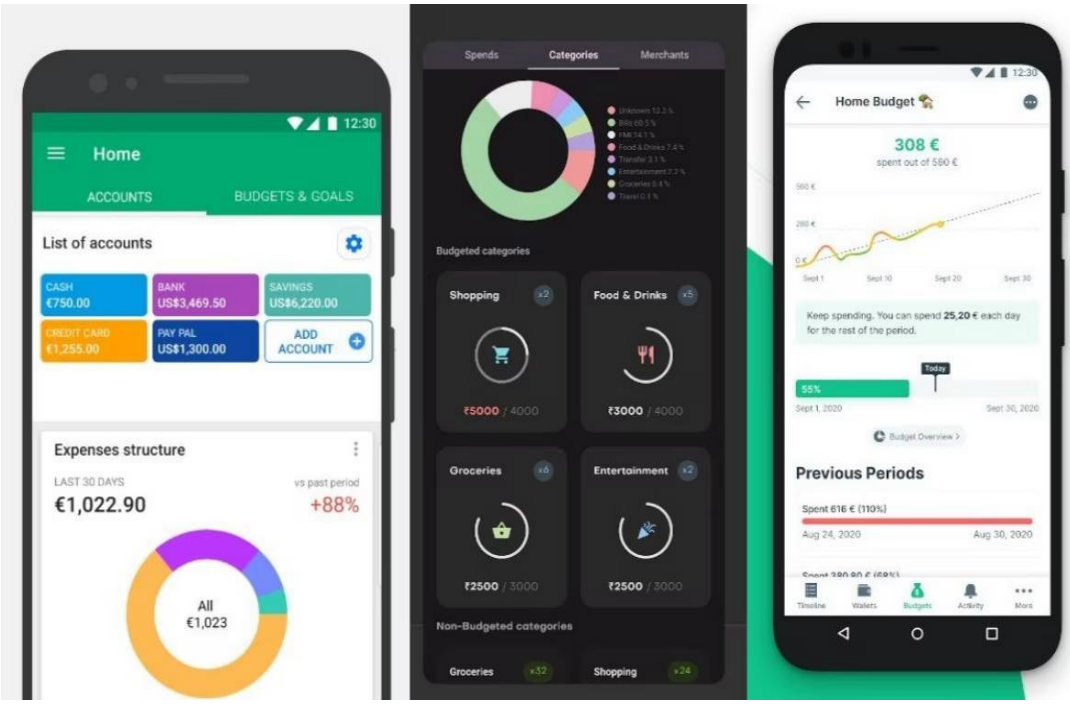
- suggestion
- risk level
- date

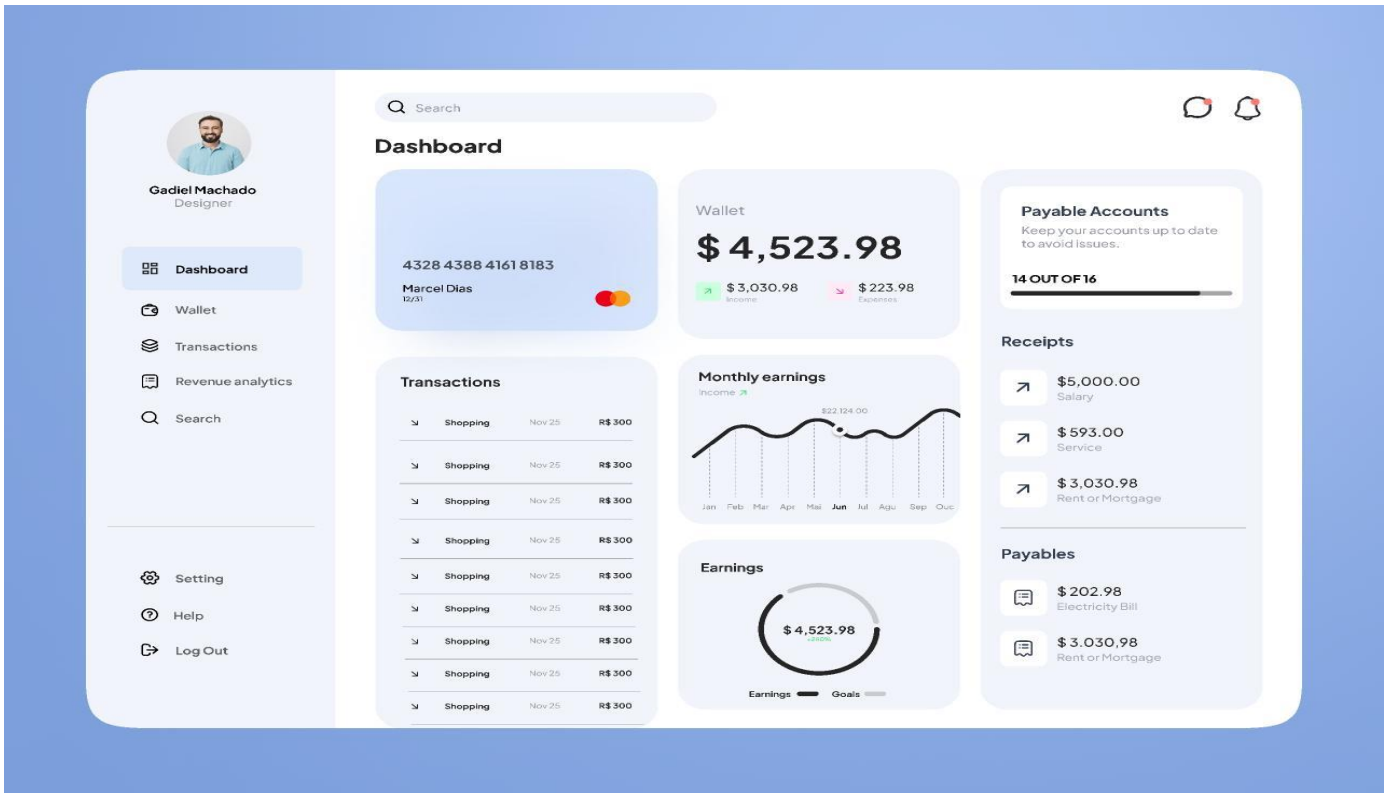
7. Learning Modules
This is separate from users.
It contains educational content:

- title
- topic
- level
- language

7.UI DESIGN :

Below images show the ui design how our website or will look :





Conclusion

The “Ghar-Ki-Bachat” system provides a complete, user-friendly solution for managing personal finances. By integrating expense tracking, income management, budgeting, goal setting, AI-driven insights, and educational modules, the system empowers users to make informed financial decisions.