FEDERAL URDU UNIVERSITY OF ARTS, SCIENCE AND TECHNOLOGY

Federal Urdu University of Arts, Science and Technology

Advance Programming Final Project

Smart Transaction Sorter & Analytics System

Version 1.0 - Technical Documentation

Prepared by:

Mahhin Shahzad

Supervised by:

Mr. Muhammad Aqib Rauf
Lecturer, Department of Computer Science

Date: December 30, 2025

Contents

Chapter 1: Introduction

1.1 Project Overview

The global financial landscape has undergone a paradigm shift over the last decade, transitioning from traditional brick-and-mortar banking to agile, digital-first solutions. In Pakistan, this revolution is spearheaded by the rapid adoption of fintech platforms and digital wallets such as Easypaisa, SadaPay, NayaPay, and JazzCash. These services have democratized access to banking, allowing millions of unbanked individuals to perform transactions, pay bills, and manage funds directly from their smartphones.

However, while the transactional side of fintech has matured, the analytical capabilities available to the end-user have lagged significantly. Users have instant access to their transaction history, often downloadable as raw CSV (Comma Separated Values) or PDF files, but they lack the tools to interpret this data. Raw data is inherently static; it tells a user *what* they bought, but not *how* their spending patterns are evolving over time.

The **Smart Transaction Sorter and Analytics System** is developed to address this specific analytical gap. It serves as a sophisticated Business Intelligence (BI) layer that sits on top of the raw data provided by existing banking infrastructure. By leveraging modern web technologies and Artificial Intelligence, the system automates the tedious process of financial tracking, transforming unintelligible spreadsheets into dynamic, actionable financial insights.

1.2 Problem Statement

Despite the availability of digital transaction records, personal financial management remains a high-friction activity for the average user. The core problem is the disconnection between ”Data Access” and ”Data Utility.”

Currently, a user wishing to analyze their monthly expenses faces a significant barrier to entry:

1. **Manual Labor & Fatigue:** The primary method for analysis is manual entry into spreadsheet software like Microsoft Excel. For a user with hundreds of transactions per month, manually tagging each row (e.g., labeling a fuel pump transaction as ”Transport”) is prohibitively time-consuming and prone to human error.

2. **Lack of Contextual Intelligence:** Standard rule-based systems are often too rigid. For instance, a payment to ”University Canteen” might be classified as ”Education” by a bank’s generic algorithm, whereas the user considers it ”Food.” Existing tools lack the personalized context to make this distinction automatically.

3. **Privacy Concerns with Third-Party Apps:** While international apps like Mint or YNAB exist, they often require direct bank credentials or sell user data to third-party advertisers.

There is a lack of a locally relevant, privacy-focused solution that processes data without exposing banking credentials.

This project solves these issues by creating a secure, user-controlled environment where data ingestion is automated, classification is handled by AI, and privacy is guaranteed through isolated database scoping.

1.3 Project Objectives

The strategic objective of this project is to develop a fully functional, standalone web application that automates the ”Data-to-Dashboard” pipeline. The specific technical goals are:

- **Secure Authentication:** To implement a robust user management system that ensures data isolation, preventing cross-user data leakage.
- **Robust Ingestion:** To develop a validation engine capable of parsing complex CSV structures from various local banks, ensuring only valid financial data enters the system.
- **AI-Driven Classification:** To integrate a pre-trained Large Language Model (LLM) capable of Zero-Shot Classification, allowing the system to understand transaction context without extensive manual training.
- **Interactive Visualization:** To move beyond static tables by rendering financial data into interactive charts (Pie, Bar, Line) that allow users to drill down into their spending habits.

1.4 Scope of Work

The scope defines the boundaries of the current release (Version 1.0).

- **In-Scope:**
  - Development of the Django backend and SQLite/PostgreSQL database.
  - Implementation of the Hugging Face Transformers pipeline for categorization.
  - Frontend development using Bootstrap and Chart.js.
  - Deployment of CSV parsing logic for standard bank formats.
- **Out-of-Scope:**
  - Direct API integration with banking servers (Open Banking APIs are not yet widely accessible in Pakistan).
  - Real-time transaction alerts (SMS reading).
  - Mobile App development (The current scope is limited to a responsive Web App).

Chapter 2: Background Study & Literature Review

2.1 The Local Fintech Landscape

To understand the necessity of the Smart Transaction Sorter, one must first analyze the capabilities and limitations of the current market leaders in Pakistan's digital payments space.

2.1.1 SadaPay NayaPay

These two Electronic Money Institutions (EMIs) have set the standard for user experience in Pakistan. They offer sleek mobile applications that allow users to view their transaction history in real-time. However, their analytical features are rudimentary.

- **Limitation:** While they allow users to tag transactions manually, this data is often locked within the mobile app. The ”Statement Export” feature provides a PDF or CSV, but the custom tags are often lost or formatted poorly in the export [**SadaPayDocs**].
- **Gap:** There is no web interface for deep analysis. A user cannot generate a ”Yearly Report” to see how their grocery spending has fluctuated over 12 months.

2.1.2 JazzCash Easypaisa

As the largest branchless banking operators, they handle millions of transactions daily. However, their primary focus is strictly transactional (sending money, paying bills).

- **Limitation:** The transaction history features are often buried in complex menus. The ”Schedule of Charges” and other documentation [**JazzCashSOC**] focus on fees rather than data accessibility.
- **Gap:** These platforms are ”Walled Gardens,” making it difficult for users to extract their data for personal budgeting.

2.2 Global Expense Tracking Solutions

Internationally, the market for Personal Finance Management (PFM) software is mature, with giants like Intuit Mint (now Credit Karma) and YNAB (You Need A Budget).

2.2.1 Why Global Tools Fail Locally

Despite their advanced features, these tools struggle to gain traction in the Pakistani market for several reasons:

1. **Bank Sync Issues:** Global tools rely on aggregators like Plaid to connect to bank accounts. Pakistani banks generally do not support these open banking standards, rendering the ”Automatic Sync” feature useless.

2. **High Friction Imports:** Without auto-sync, users are forced to use the ”File-Based Import” features of tools like YNAB [3]. However, these tools often require CSV files to be in a very specific, rigid format (e.g., specific date formatting MM/DD/YYYY), which often conflicts with the formats exported by local Pakistani banks.

3. **Cost Prohibitive:** Many of these tools charge monthly subscriptions in USD, which is not viable for the average local student or freelancer.

2.3 The Technological Gap

The comparative analysis reveals a clear ”Technological Gap.” On one side, we have local banks with raw data but no analytics. On the other side, we have global analytics tools that cannot connect to local data.

The **Smart Transaction Sorter** fills this specific gap. It is a ”Middle Layer” solution. It does not require direct bank integration (solving the connectivity issue) but offers advanced analytics (solving the insight issue). By using AI, it removes the friction of manual formatting, making it the ideal solution for the local context.

Chapter 3: System Requirements Specification (SRS)

This chapter outlines the formal requirements derived from the stakeholder analysis. These requirements serve as the governing contract for the system's development.

3.1 Functional Requirements

The functional requirements are categorized by module.

3.1.1 Security Management Module

The security module is critical for maintaining user trust.

ID	Requirement Description
SRS-01	The system shall allow new users to securely register by providing a username, unique email address, and a strong password.
SRS-02	The system shall enforce a password complexity policy (minimum 8 characters, alphanumeric).
SRS-03	The system shall store user passwords using the PBKDF2 hashing algorithm with a SHA-256 digest to prevent rainbow table attacks.
SRS-04	The system shall implement session management, automatically logging users out after a period of inactivity to prevent unauthorized access.

3.1.2 Category Management Module

This module empowers users to personalize the system.

ID	Requirement Description
SRS-08	The system shall provide an interface for users to Create, Read, Update, and Delete (CRUD) custom spending categories.
SRS-09	The system shall ensure category names are unique per user to prevent data ambiguity during aggregation.
SRS-10	The system shall provide visual feedback (color coding) for categories to enhance the dashboard experience.

3.1.3 Data Ingestion Module

This module handles the complexity of file parsing.

ID	Requirement Description
SRS-14	The system shall accept CSV files via a drag-and-drop web interface.

SRS-15	The system shall validate the file extension (.csv) before attempting to parse the file.
SRS-16	The system shall validate the internal structure of the CSV, checking for essential headers ("Date", "Description", "Amount").
SRS-17	The system shall handle date parsing errors gracefully, supporting multiple formats (DD-MM-YYYY and MM-DD-YYYY).

3.1.4 Intelligent Sorting (AI Core)

The AI module is the differentiating feature of this system.

ID	**Requirement Description**
SRS-20	The system shall utilize the Hugging Face `pipeline` API to perform Zero-Shot Classification on transaction descriptions.
SRS-21	The system shall dynamically map the user's custom categories as the "candidate labels" for the AI model.
SRS-22	The system shall flag transactions with a low confidence score ($< 50\%$) as "Uncategorized" for manual review.

3.2 Non-Functional Requirements

- **Performance:** The system must process a batch of 500 transactions in under 60 seconds to ensure a fluid user experience.
- **Scalability:** The database schema must be normalized to support thousands of transactions per user without significant query latency.
- **Privacy:** The system must adhere to a strict "Data Isolation" policy. A user's financial data must never be accessible to other users or the system administrators.
- **Availability:** The web application should be deployable on standard cloud infrastructure (Heroku/AWS) with 99.9% uptime.

Chapter 4: System Architecture & Detailed Design

This chapter bridges the gap between requirements and implementation. It presents the logical flow of data through the system, detailed via Unified Modeling Language (UML) sequence diagrams.

4.1 System Overview (High-Level Design)

The System Sequence Diagram (SSD) below provides a ”Black Box” view of the system. It illustrates the primary success scenario, detailing the linear flow from user authentication to the final visualization of the dashboard. This high-level view establishes the boundaries of the system and the external actors involved.

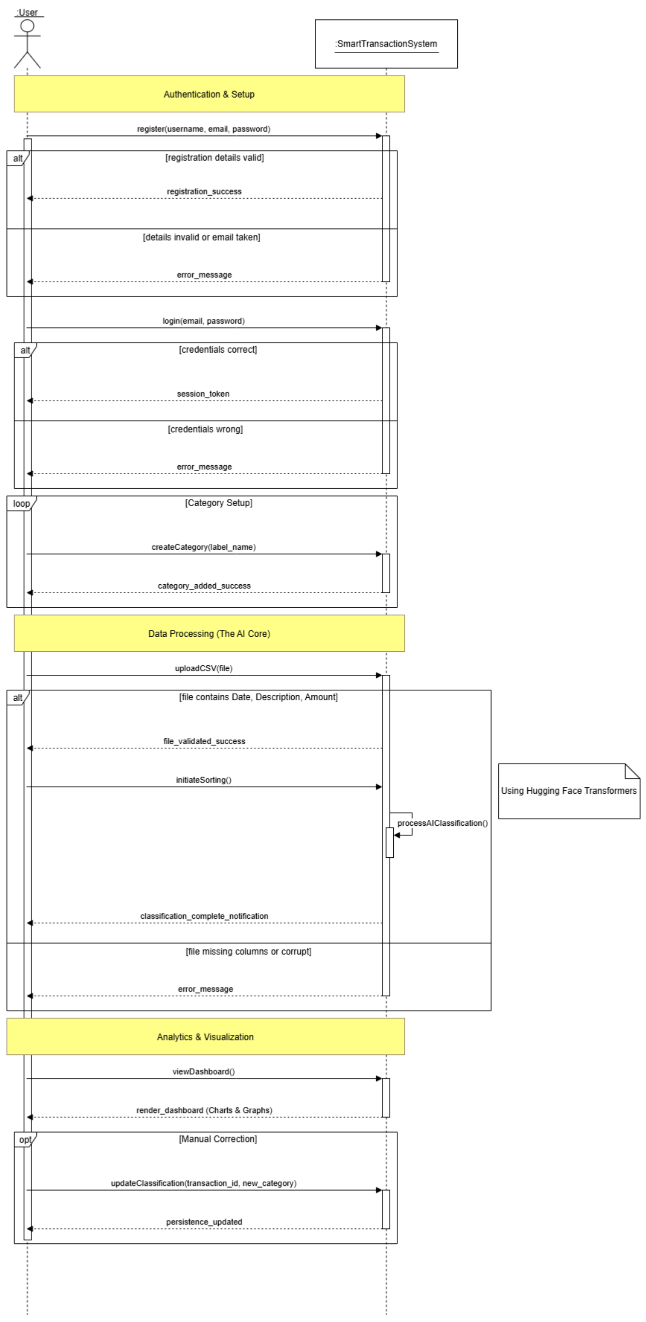
:User
:SmartTransactionSystem
Authentication & Setup
register(username, email, password)
alt
[registration details valid]
registration_success
[details invalid or email taken]
error_message
login(email, password)
alt
[credentials correct]
session_token
[credentials wrong]
error_message
loop
[Category Setup]
createCategory(label_name)
category_added_success
Data Processing (The AI Core)
uploadCSV(file)
alt
[file contains Date, Description, Amount]
file_validated_success
initiateSorting()
processAIClassification()
Using Hugging Face Transformers
classification_complete_notification
[file missing columns or corrupt]
error_message
Analytics & Visualization
viewDashboard()
render_dashboard (Charts & Graphs)
opt
[Manual Correction]
updateClassification(transaction_id, new_category)
persistence_updated

Figure 1: System Sequence Diagram (SSD) showing the overall process flow.

4.2 Detailed Module Design

The system architecture is modular, following the Django MVT (Model-View-Template) pattern. Each module encapsulates specific business logic.

4.2.1 User Access & Authentication Module

This module handles the security barrier of the application. It acts as the gatekeeper, ensuring that only authenticated users can access the sensitive financial features.

1. **Registration Flow:** The user submits their details. The system validates the email uniqueness and strength of the password. Upon success, the password is salted and hashed using PBKDF2 before being stored in the `User` table.

2. **Login Flow:** The system takes the input password, re-hashes it using the stored salt, and compares it to the database entry. If they match, a secure session token is issued.

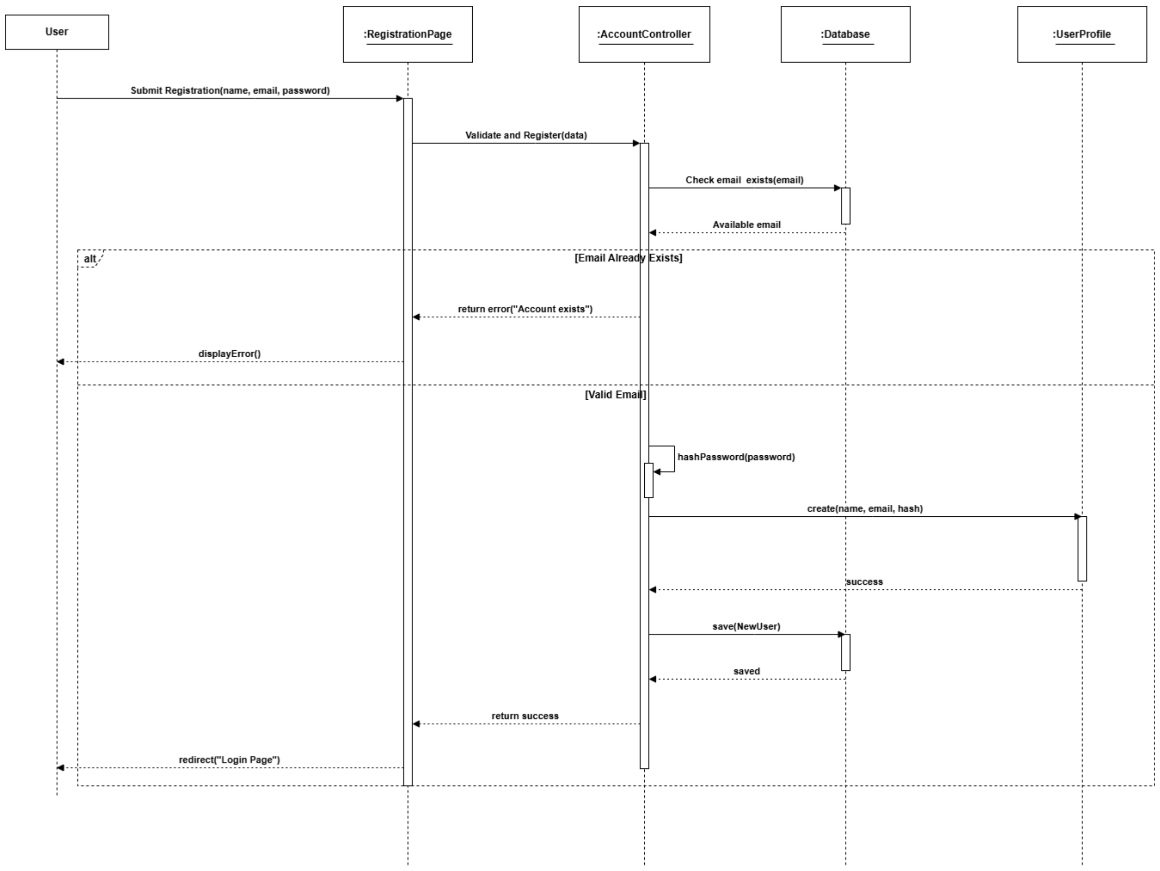

Figure 2: Sequence Diagram: User Registration Process.

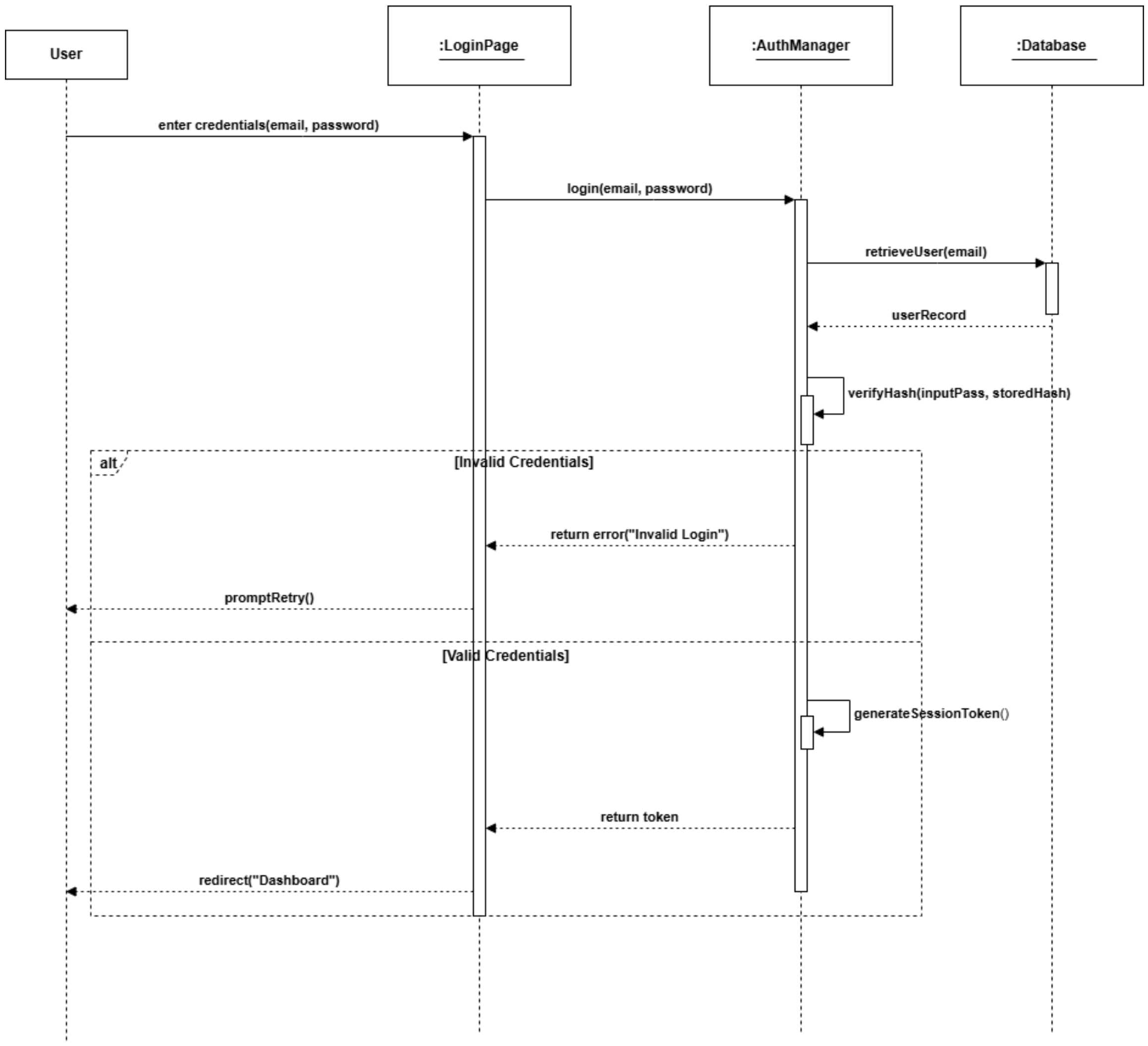

Figure 3: Sequence Diagram: Secure Login Process.

4.2.2 Configuration Module (Category Management)

This module allows the user to define the ”labels” that the AI will use. Unlike rigid banking apps, this system allows full CRUD operations.

The sequence diagram below illustrates the checks performed when a user adds a category. The `CategoryController` must first verify that the label (e.g., ”Gym”) does not already exist for that specific user ID to avoid duplicate data aggregation later.

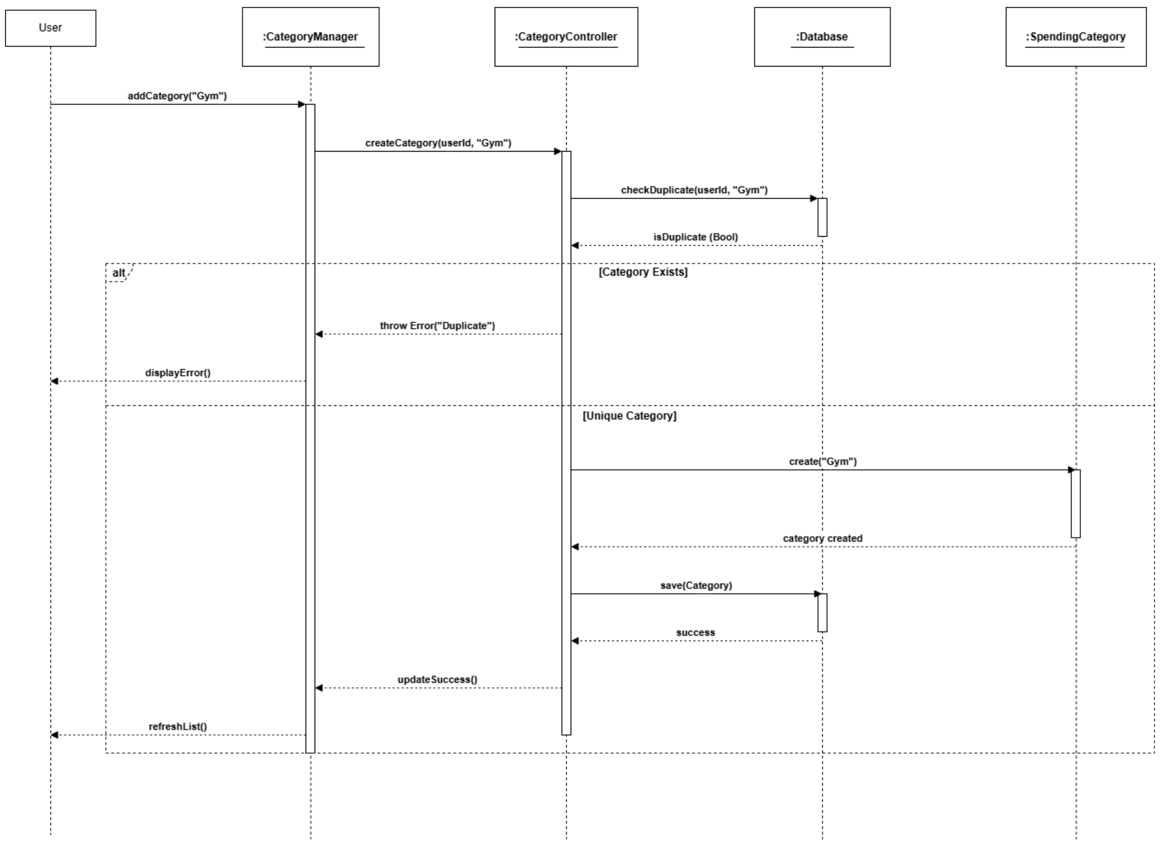

Figure 4: Sequence Diagram: Manage Spending Categories.

4.2.3 Data Ingestion Module

This module handles the ETL (Extract, Transform, Load) process. It is responsible for bridging the gap between the user's raw file and the system's database.

The process involves several validation steps:

1. **Extension Check:** Ensuring the file ends in `.csv`.
2. **Header Validation:** Using the Pandas library to read the first row and ensure columns like "Date", "Description", and "Amount" exist.
3. **Data Cleaning:** Stripping currency symbols (e.g., "PKR 500" → 500) to ensure the data is mathematically usable.

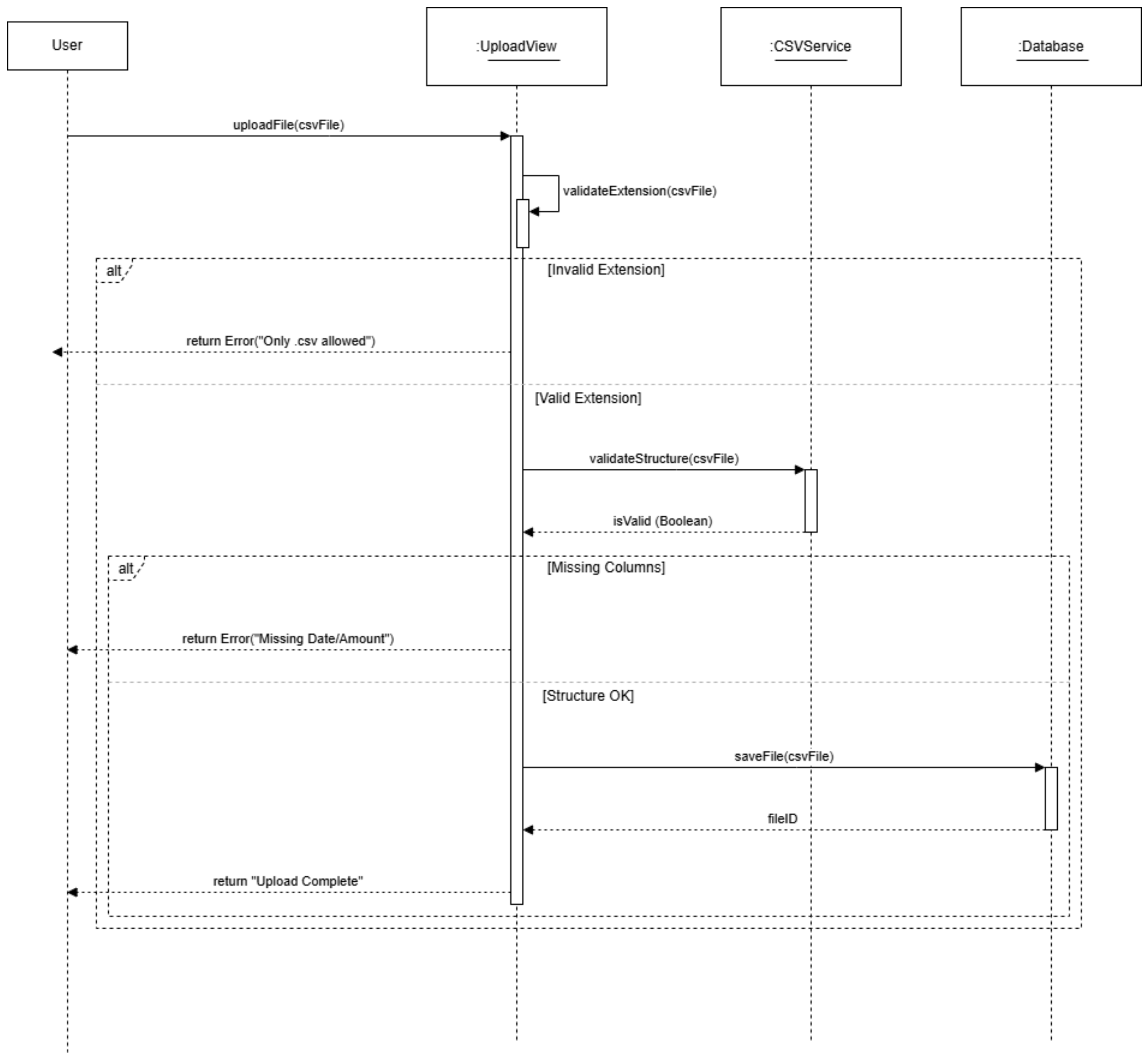

Figure 5: Sequence Diagram: CSV Upload and Validation.

4.2.4 Intelligent Sorting Module (AI Core)

This is the central processing unit of the application. The sequence diagram below details the iterative loop used to classify transactions.

The AI Loop:

- The system retrieves the user's custom categories (the "Candidate Labels").
- It iterates through every row of the uploaded CSV.
- The Description text is sent to the Hugging Face Zero-Shot model.
- The model returns a confidence score. If the score is high (e.g., > 0.7), the category is automatically assigned. If low, it is flagged.
- Finally, the labeled transaction is updated in the database.

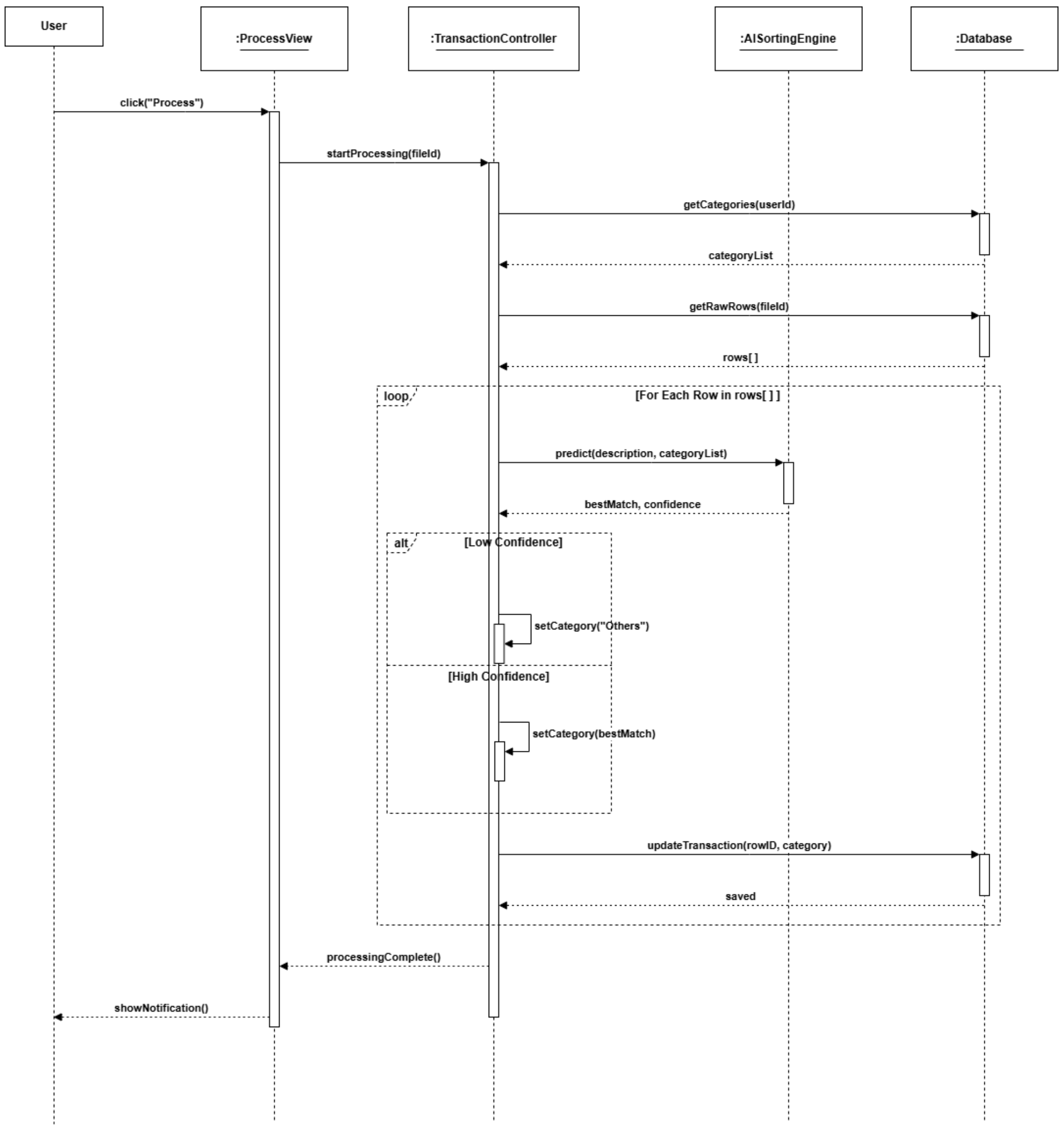

Figure 6: Sequence Diagram: AI Transaction Processing.

4.2.5 Analytics & Visualization Module

This module aggregates data from the database and prepares it for the frontend.

The sequence diagram shows the interaction between the View and the Template. The View performs database aggregation queries (e.g., Summing all expenses where Category = 'Food'). This aggregated data is serialized into JSON format and passed to the frontend, where the Chart.js library renders it into visual graphs.

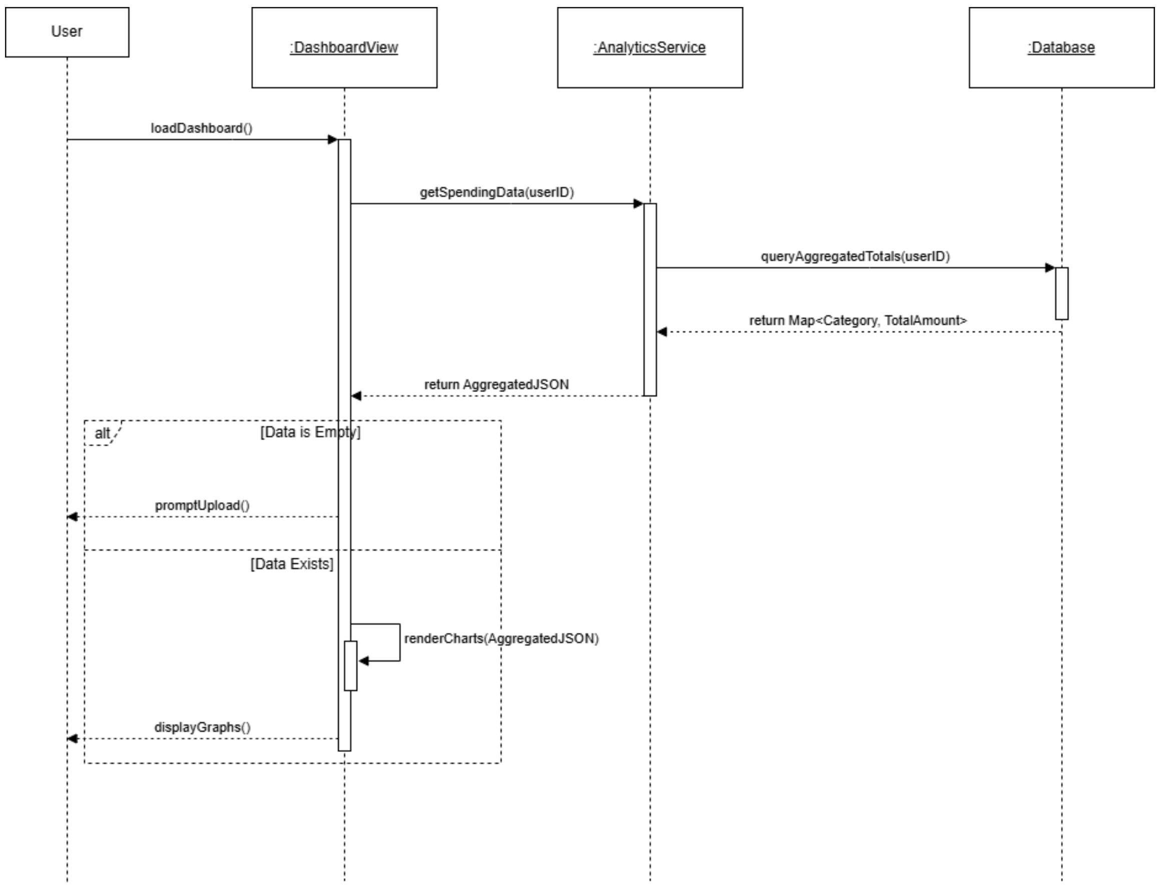

Figure 7: Sequence Diagram: View Analytics Dashboard.

4.2.6 Data Maintenance Module

To comply with privacy requirements and GDPR principles, users retain full ownership of their data. The sequence diagram below illustrates the ”Hard Delete” process. When a user requests deletion, the system does not just hide the data; it permanently removes the records from the PostgreSQL tables, ensuring they cannot be recovered.

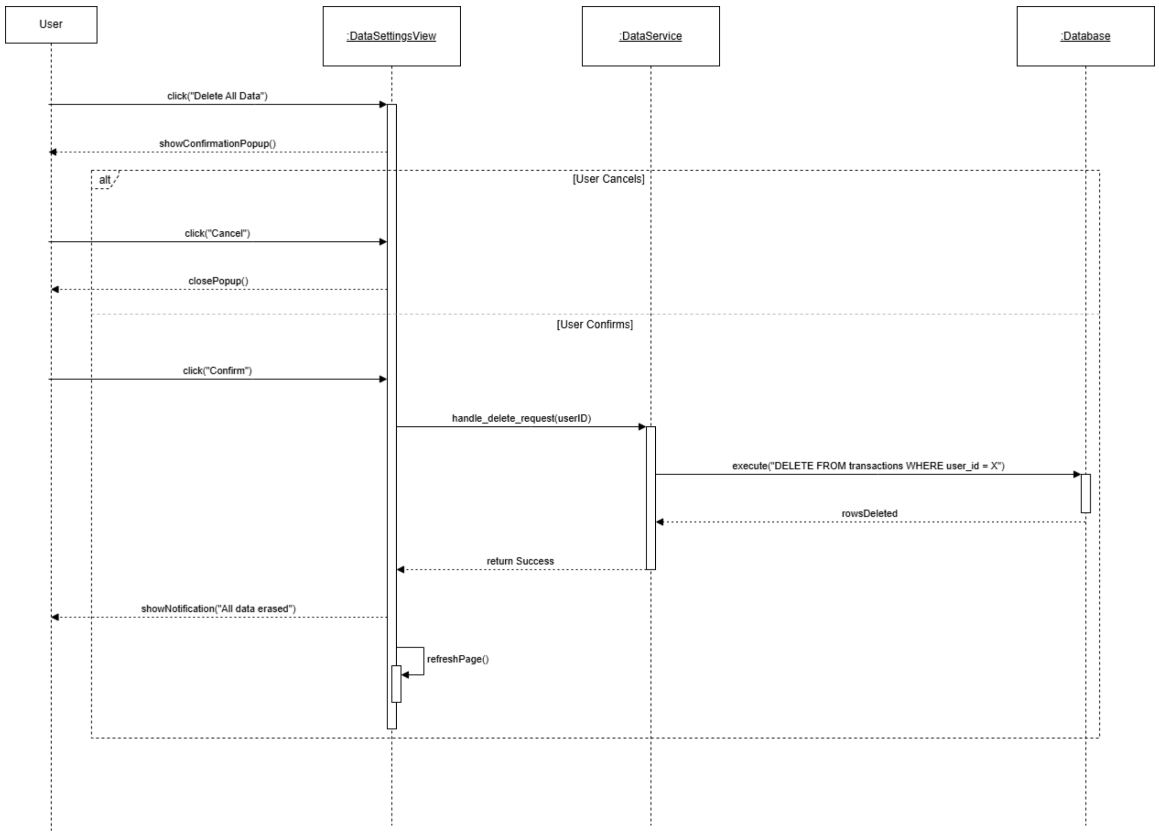

Figure 8: Sequence Diagram: Delete Transaction Data.

4.2.7 Administrative Module

The Admin module allows system administrators to oversee the health of the application. The Administrator can view user statistics (e.g., total registered users) and manage account statuses, but strictly cannot view the raw financial data of users due to the data isolation architecture.

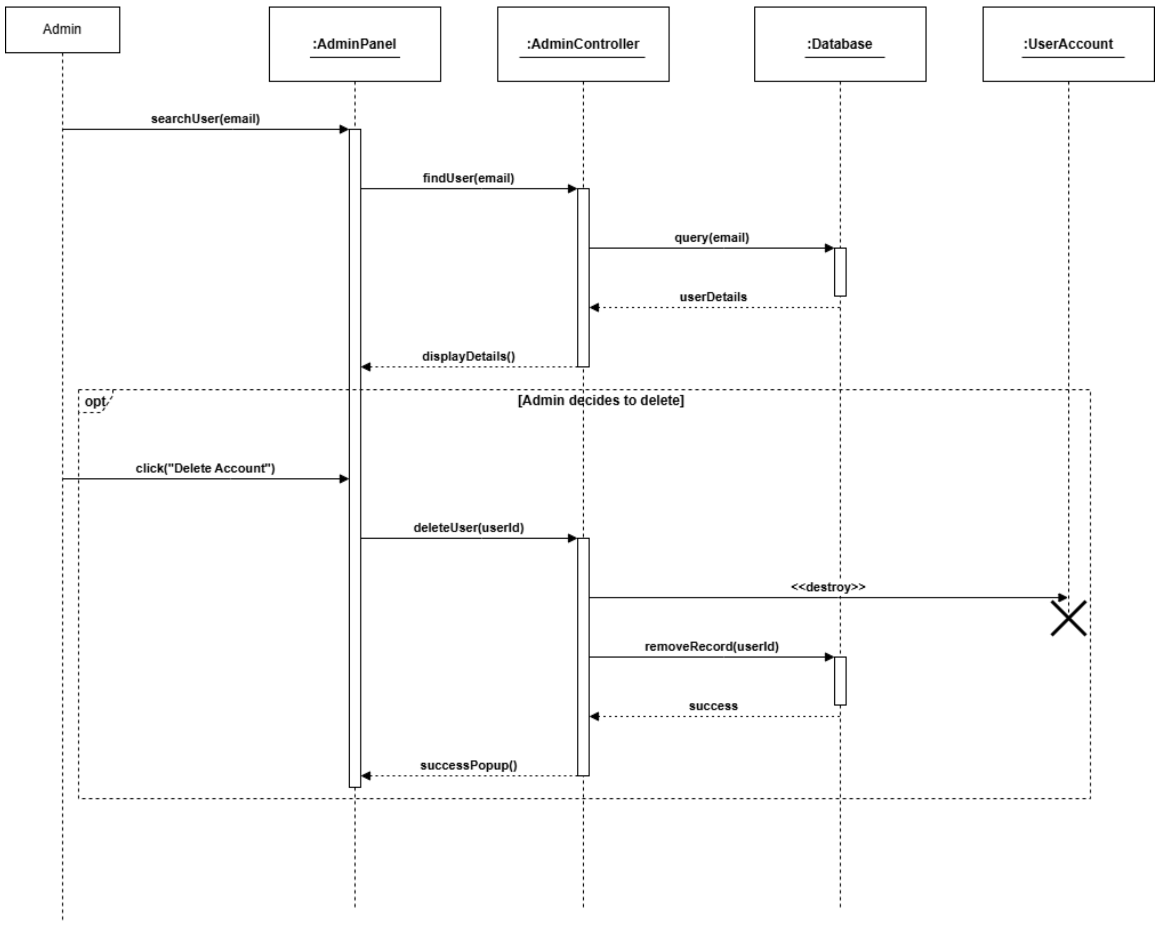

Figure 9: Sequence Diagram: Admin User Management.

Chapter 5: Methodology

The development of the Smart Transaction Sorter followed a structured software engineering approach to ensure reliability, scalability, and code maintainability. By adhering to industry-standard practices, the project ensures that the integration of the AI components with the web framework is seamless and robust.

5.1 Software Process Model

The project utilized the **Iterative and Incremental** process model. Given the complexity of integrating a probabilistic AI model with a deterministic web framework, a linear Waterfall model would have been too rigid.

The Iterative model allowed for the project to be broken down into manageable cycles:

1. **Phase 1 (Core Architecture):** Building the basic Django setup, Secure Authentication (SRS-01 to SRS-06), and database schema.
2. **Phase 2 (Data Engineering):** Developing the Data Ingestion Module, specifically CSV parsing logic and validation (SRS-14 to SRS-19).
3. **Phase 3 (AI Integration):** Integrating the Hugging Face library and refining the Zero-Shot Classification model for local transaction descriptions.
4. **Phase 4 (UI & Visualization):** Building the interactive dashboard using Chart.js to ensure the interface is responsive.

5.2 Process Workflow

The system operates on a linear pipeline, which serves as the backbone of the user experience. The workflow is designed to minimize user friction while maximizing data integrity.

1. **Input:** User uploads a raw CSV file.
2. **Validation:** System checks for structural integrity (Date, Amount columns).
3. **Intelligence:** AI engine processes valid data and assigns categories.
4. **Output:** Dashboard renders interactive charts.

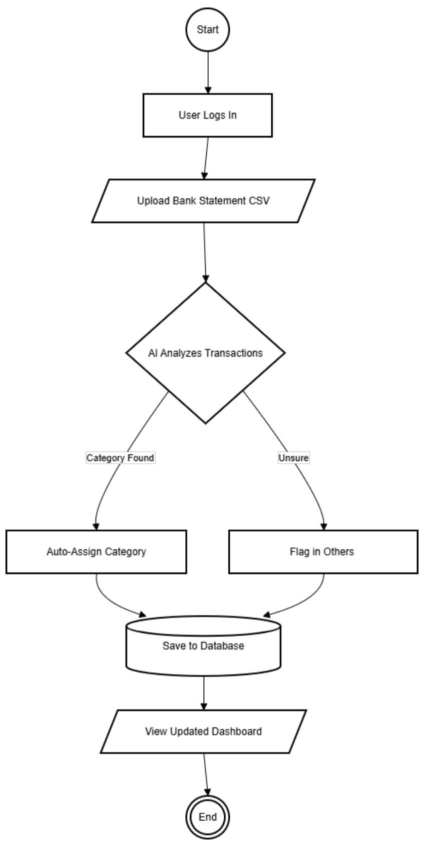

Figure 10: Process Workflow: From CSV Upload to Dashboard Visualization.

5.3 Project Timeline

The project development was executed over a span of 30 weeks. The timeline below illustrates the parallel development phases, ensuring that core backend features were stabilized before the complex AI integration began.

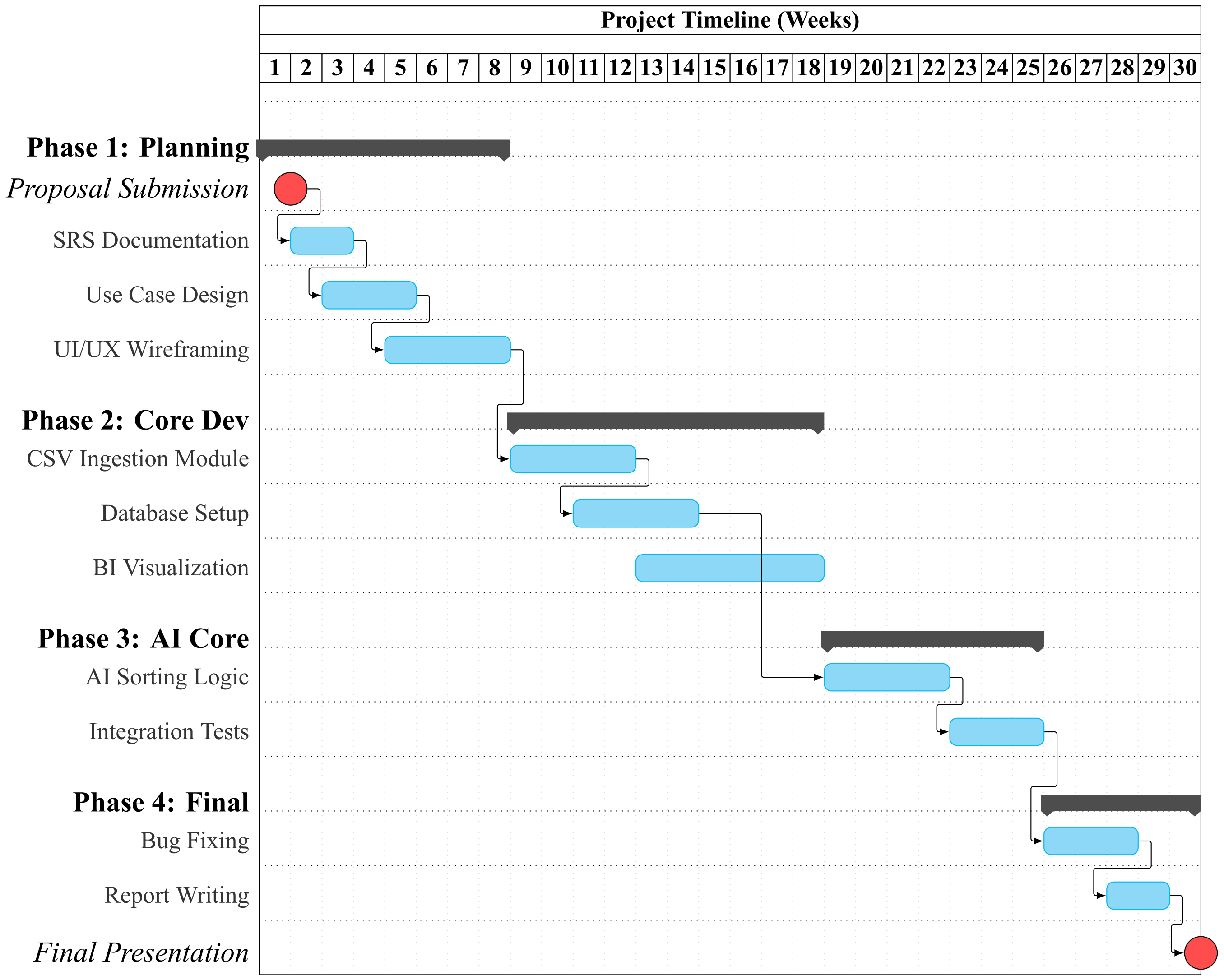

Figure 11: Gantt Chart outlining the 30-week development schedule.

==

Chapter 6: Implementation Results

The Smart Transaction Sorter is Partially operational. The results below focus on quantitative metrics such as processing speed and classification accuracy.

6.1 Comparative Analysis

Metric	**The Challenge (Manual)**	**Our Technical Solution**	**Result**
Speed	Manual Entry ($\approx$ 45 mins)	Automated Parsing	$<$ **60 Secs**
Accuracy	Variable / Human Error	AI Classification	**80% Accuracy**
Security	High Risk (Cloud apps)	Isolated Database	**Private**
Usability	Static Rows	Interactive Charts	**Dynamic**

Table 5: Performance Comparison Table.

Chapter 7: Conclusion

The Smart Transaction Sorter represents a significant step forward in personal finance management. By combining the flexibility of custom categories with the power of AI-driven sorting, it eliminates the manual labor that plagues traditional expense tracking. The system successfully demonstrates that high-end Business Intelligence is not just for corporations, but can be made accessible to the everyday user through intelligent software design.

References

[1] Intuit Mint. *How do I export my transaction data?* 2025. URL: https://mint.intuit.com/support/ (visited on 10/23/2025).

[2] NayaPay. *How can I view my documents on NayaPay?* 2025. URL: https://help.nayapay.com/article/251-how-can-i-view-my-documents-on-nayapay (visited on 10/23/2025).

[3] YNAB Support. *File-Based Import — A Guide*. 2025. URL: https://support.ynab.com/en_us/file-based-import-a-guide-Bkj4Sszyo (visited on 10/23/2025).