Project Documentation
(Franzy)

Section: A

Submitted by

	Participant Reg Number	*Participant Name*
1	**Sp-23/BSSE/029**	**Muhammad Abdullah**

Submitted To: Sir Farhan Sarwar

Department of Software Engineering,
Lahore Garrison University, Lahore
[Date of Submission: 05/12/2025]

Table of Contents

Project Documentation 1

Franzy 4

1. Introduction 4

1.1 Overview 4

1.2 Problem Statement 4

1.3 Objectives 4

1.4 Audience 4

1.5 Definitions and Acronyms 4

2. Overall Description 4

2.1 Product Perspective 4

2.2 Product Features 4

2.3 User Classes and Characteristics 5

2.4 Operating Environment 5

2.5 Constraints 5

2.6 Assumptions and Dependencies 5

3. System Requirements Specification 5

3.1 Functional Requirements 5

3.2 Non-Functional Requirements 5

3.3 Business Requirements 5

3.4 User Requirements 6

3.5 System Requirements 6

4. UML Diagrams 6

4.1 Use Case Diagram 6

4.2 Use Case Descriptions 6

4.3 Class Diagram 6

4.4 Activity Diagram 7

4.5 Sequence Diagram 8

4.6 Deployment Diagram 9

5. System Architecture 9

5.1 Architectural Overview 9

5.2 Module Breakdown 9

6. Technologies and Tool 9
6.1 Programming Languages 9
6.2 Frameworks 9
6.3 Libraries 9
6.4 Storage 10
7. Testing Plan 10
7.1 Test Strategy 10
7.2 Sample Test Cases 10
7.3 Validation and Verification 10
8. Future Enhancements 10
9. Conclusion 10

Franzy

1. Introduction

1.1 Overview

Franzy is a personal finance dashboard that enables users to upload bank transaction CSV files, automatically categorize their expenses, and visualize financial data in an intuitive interface. The system allows users to add custom categories and save categorization rules locally for future uploads. Franzy provides both tabular and graphical insights into personal expenses and payments, helping users analyze their spending habits effortlessly.

1.2 Problem Statement

Traditional bank statements are unorganized and time-consuming to analyze manually. Users often spend considerable time calculating category-wise expenses, which is error-prone and repetitive every month. There is a need for a simple, intelligent, offline solution that automates categorization, provides summaries, and visualizes spending.

1.3 Objectives

- Automate categorization of transactions using keyword rules
- Enable addition of custom categories and keywords
- Provide clear visual insights including pie charts and summaries
- Maintain an offline, secure environment without cloud dependency
- Offer an intuitive and editable interface for all users

1.4 Audience

- Final year project evaluators and instructors
- Students and developers interested in finance dashboard applications
- Users seeking simple offline tools for personal finance management

1.5 Definitions and Acronyms

Term	Description
CSV	Comma-Separated Values
JSON	JavaScript Object Notation
UI	User Interface
UX	User Experience
PKR	Pakistani Rupee

2. Overall Description

2.1 Product Perspective

Franzy is a standalone desktop or browser-based application using Streamlit. The product leverages CSV files as input and JSON files for local persistent storage. Plotly is used for data visualization. Franzy does not rely on any external databases or APIs, ensuring offline usability.

2.2 Product Features

- CSV Upload: Load bank statements in .csv format
- Transaction Parsing: Clean and process Date, Amount, Details, Debit/Credit columns
- Auto Categorization: Assign transactions to categories based on keywords

- Editable Table: Modify categories directly in the interface
- Persistent Storage: Save category rules in categories.json
- Visual Dashboard: Pie charts and summaries for expenses and payments
- Custom Category Management: Add, remove, or edit categories and keywords

2.3 User Classes and Characteristics

User Type	Description
Basic User	Uploads CSV files, views categorized expenses, and visualizations
Advanced User	Manages categories and keywords, edits transactions
Developer	Maintains application, modifies JSON storage, and improves functionality

2.4 Operating Environment

- Python 3.8 or higher
- Streamlit framework
- Operating Systems: Windows, Linux, macOS
- Browser: Chrome, Edge, Firefox

2.5 Constraints

- CSV files must conform to required column structure
- Local-only storage, no cloud integration
- Manual keyword addition required for uncategorized merchants

2.6 Assumptions and Dependencies

- Input CSV files are correctly formatted
- JSON file for categories is accessible and writable
- Pandas correctly parses dates and amounts

3. System Requirements Specification

3.1 Functional Requirements

- **CSV Upload**: Accept and validate bank statement CSV files
- **Transaction Processing**: Parse and clean Date, Amount, Details, Debit/Credit fields
- **Categorization Engine**: Auto-assign categories based on keywords
- **Category Management**: Add, edit, remove categories; update JSON
- **Editable Transactions Table**: Modify transaction categories
- **Visualization**: Display pie charts and category-wise summaries
- **Payment Summary**: Calculate and display total credits

3.2 Non-Functional Requirements

- **Performance**: Process up to 10,000 transactions in under 3 seconds
- **Usability**: Intuitive interface for non-technical users
- **Reliability**: Automatic saving of category rules
- **Portability**: Run on any system with Python and Streamlit
- **Maintainability**: Modular code for easy future updates
- **Security**: Data remains local and private

3.3 Business Requirements

- Reduce manual effort in expense tracking
- Improve user awareness of financial behavior
- Deliver offline accessibility and simplicity
- Enable personalized categorization rules

3.4 User Requirements

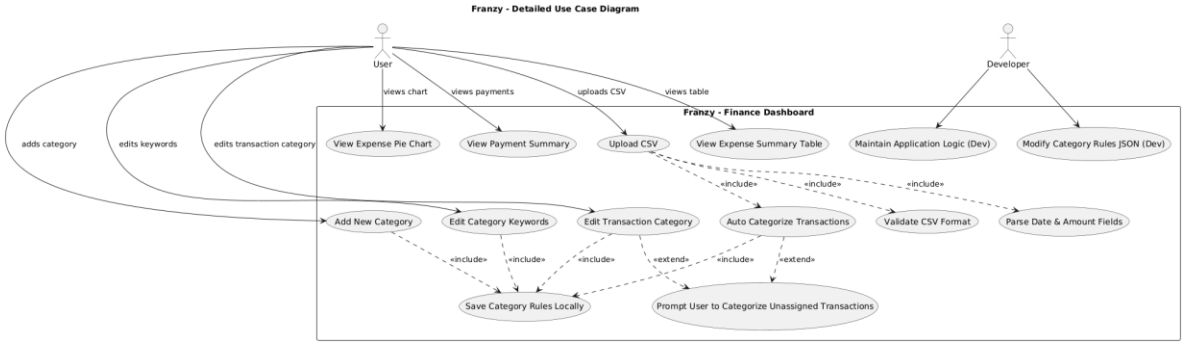
- Upload CSV files easily
- Modify categories with a simple UI
- View categorized expenses clearly
- Analyze financial behavior using charts and summaries

3.5 System Requirements

- Python 3.8+
- Libraries: Streamlit, Pandas, Plotly, JSON
- Minimum RAM: 2GB

4. UML Diagrams

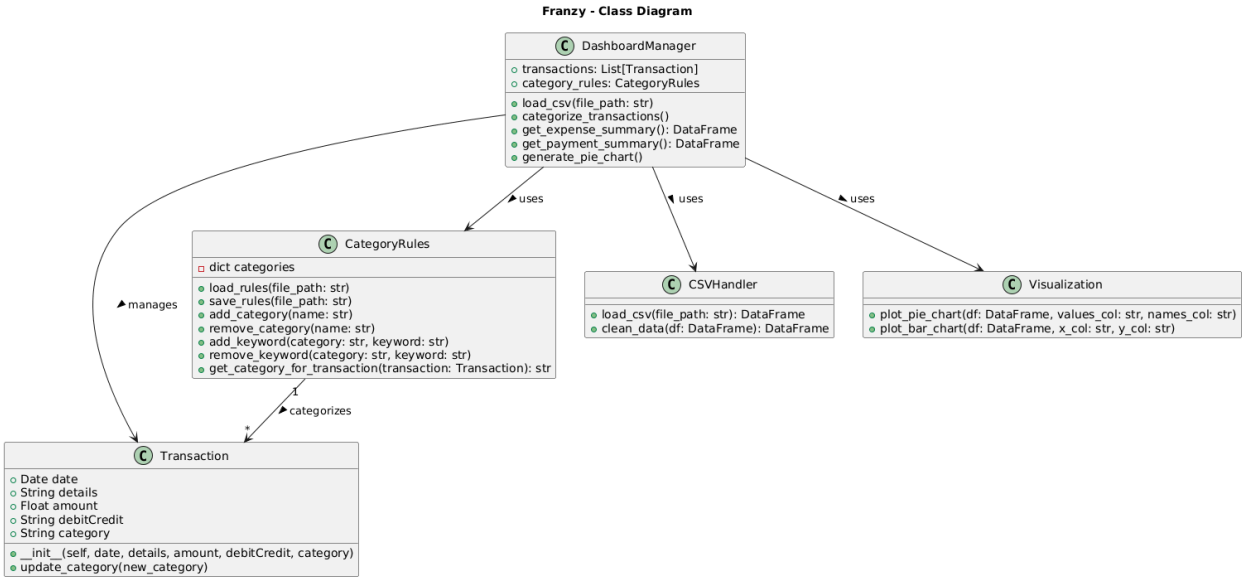
4.1 Use Case Diagram



4.2 Use Case Descriptions

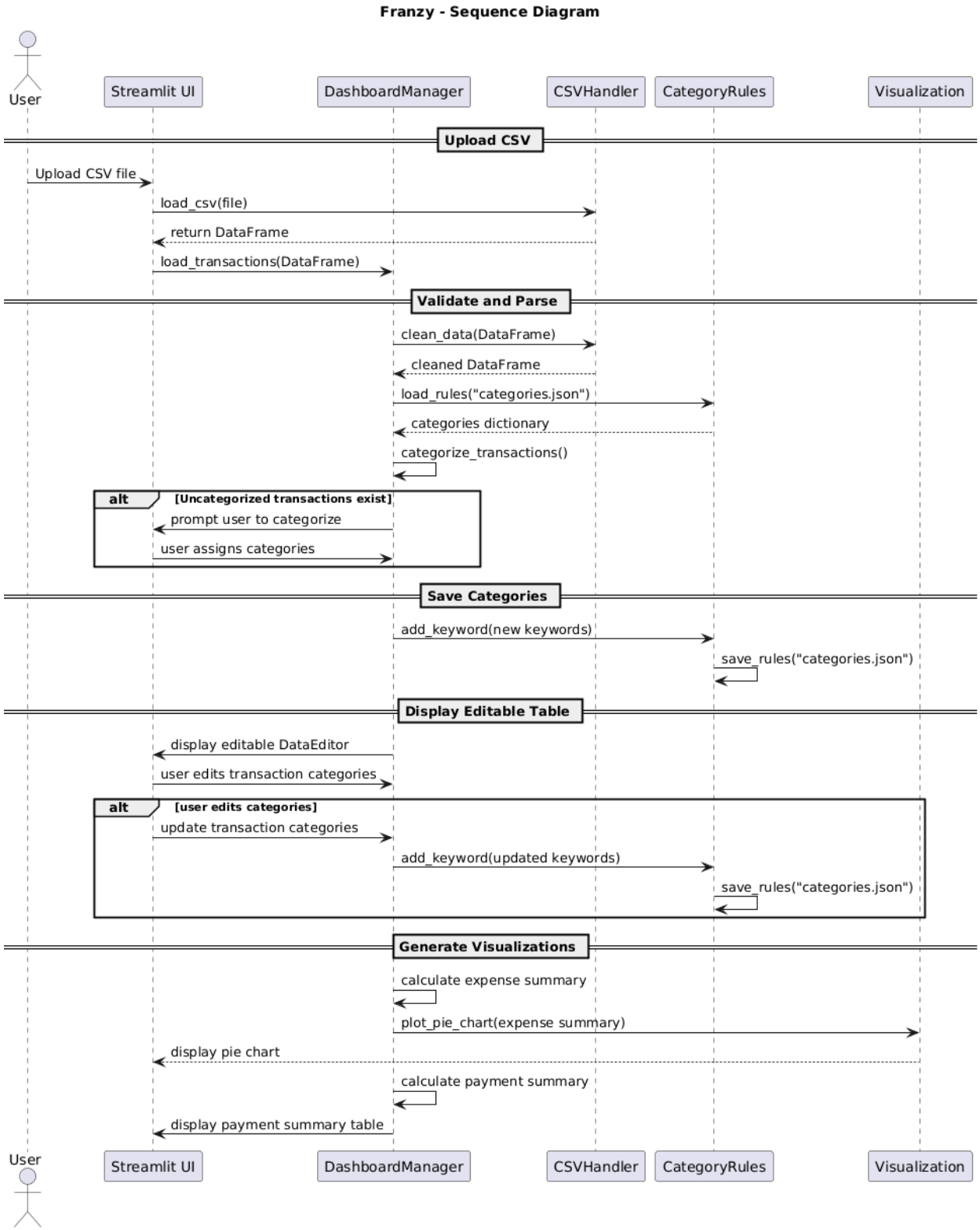
Upload CSV: User uploads CSV, system validates structure, and loads data.
Auto Categorize: System assigns categories using stored keywords.
Add/Edit Category: User can add new categories or modify keywords.
View Summary/Charts: System generates visual summary of expenses and payments.

4.3 Class Diagram

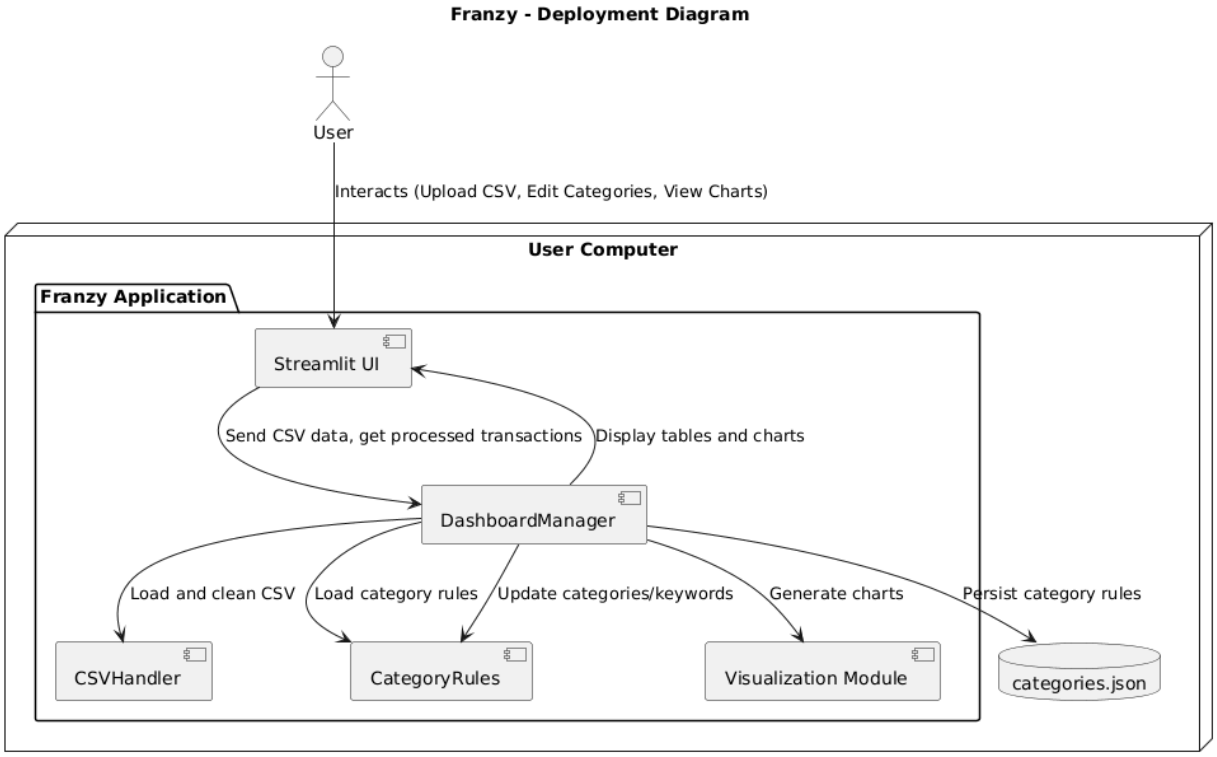


4.4 Activity Diagram

4.5 Sequence Diagram



4.6 Deployment Diagram



5. System Architecture

5.1 Architectural Overview

Franzy follows a modular architecture with:

- UI Layer: Streamlit interface
- Processing Layer: Pandas-based transaction processing and categorization
- Storage Layer: JSON file for persistent categories
- Visualization Layer: Plotly for charts

5.2 Module Breakdown

- CSV Handling Module
- Categorization Engine
- Category Manager Module
- Visualization Module
- Editor Module
- Storage Module

6. Technologies and Tool

6.1 Programming Languages

- Python 3.8+

6.2 Frameworks

- Streamlit

6.3 Libraries

- Pandas

- Plotly
- JSON

6.4 Storage

- categories.json for rules and keywords

7. Testing Plan

7.1 Test Strategy

- Unit testing for categorization logic
- Functional testing for CSV upload and processing
- UI validation testing for data editor and charts
- Performance testing for large datasets

7.2 Sample Test Cases

Test Case	Expected Result
Upload valid CSV	Data loaded successfully
Upload invalid CSV	Error message displayed
Add new category	JSON updated with category
Edit transaction category	Keyword saved in JSON
View pie chart	Chart rendered correctly

7.3 Validation and Verification

- Validation ensures features meet user requirements
- Verification ensures system behaves as expected

8. Future Enhancements

- Cloud synchronization
- AI-assisted categorization
- Budgeting and alerts
- Mobile application version
- PDF/Excel export of reports
- Multiple account merging

9. Conclusion

Franzy provides a complete, offline solution for personal finance management. It automates transaction categorization, enables custom rules, visualizes expenses, and provides clear insights for users. The system is modular, scalable, and designed to be user-friendly while maintaining data security and local storage.