****

**COMSATS University Islamabad**

**Department of Computer Science**

SRS Documentation

for

**Project Title**

***Expense Tracker***  
Version 1.0

**Submitted By:**

**Ahsan Shahid SP23-BSE-029**

**Muhammad Umer** **SP23-BSE-038**

**BSE-5**

**SupervisorAamir Shabbir Parre**

Submission Date: (6/4/2025)

Original Version 1.0

Contents

[1. Introduction 3](#_Toc201220514)

[1.1 Purpose 3](#_Toc201220515)

[1.2 Scope 3](#_Toc201220516)

[1.3 Definitions 4](#_Toc201220517)

[2. Overall Description 4](#_Toc201220518)

[2.1 Product Perspective 4](#_Toc201220519)

[2.2 User Classes 5](#_Toc201220520)

[2.3 Operating Environment 5](#_Toc201220521)

[2.4 Design and Implementation Constraints 5](#_Toc201220522)

[3. Functional Requirements 6](#_Toc201220523)

[3.1 User Management Module 6](#_Toc201220524)

[3.1.1 Authentication 6](#_Toc201220525)

[3.1.2 Profile Management 6](#_Toc201220526)

[3.2 Expense/Income Tracking 6](#_Toc201220527)

[3.3 AI Features 6](#_Toc201220528)

[3.3.1 Auto-Categorization 6](#_Toc201220529)

[3.4 Analytics & Reporting 7](#_Toc201220530)

[3.5 Admin Panel (Optional) 7](#_Toc201220531)

[4. Non-Functional Requirements 7](#_Toc201220532)

[4.1 Performance 7](#_Toc201220533)

[4.2 Security 8](#_Toc201220534)

[4.3 Usability 8](#_Toc201220535)

[4.4 Reliability 8](#_Toc201220536)

[5. System Features (Use Cases) 8](#_Toc201220537)

[5.1 AI-Powered Auto-Categorization 8](#_Toc201220538)

[5.2 Budget Alert 8](#_Toc201220539)

[5.3 User Management Use Cases 9](#_Toc201220540)

[6. External Interface Requirements 9](#_Toc201220541)

[6.1 User Interfaces 9](#_Toc201220542)

[6.2 Software Interfaces 9](#_Toc201220543)

[6.3 Hardware Interfaces 9](#_Toc201220544)

[Software Design Specification (SDS) 10](#_Toc201220545)

[System Overview 10](#_Toc201220546)

[Architectural Design 11](#_Toc201220547)

[Folder Structure (Simplified) 11](#_Toc201220548)

[Data Models 11](#_Toc201220549)

[Security 12](#_Toc201220550)

[Front-End Implementation 12](#_Toc201220551)

[Technologies Used 12](#_Toc201220552)

[Pages & Components 12](#_Toc201220553)

[Notable Features 12](#_Toc201220554)

[Back-End Implementation 13](#_Toc201220555)

[Technologies Used 13](#_Toc201220556)

[Routes 13](#_Toc201220557)

[Screenshots 13](#_Toc201220558)

[How to Run (Deployment Instructions) 15](#_Toc201220559)

# 1. Introduction

## 1.1 Purpose

**This document defines the functional and non-functional requirements for the Expense Tracker—a web-based personal finance management application built on the MERN stack with AI integration for auto-categorization. The system is designed to simplify expense and income tracking by providing an intelligent categorization feature using Hugging Face Transformers, while keeping the overall design simple and manageable for a student project.**

## 1.2 Scope

**In Scope:**

* **User Management Module:**
  + **Authentication using JWT-based email/password login.**
  + **Basic profile management for updating personal details.**
* **Expense/Income Tracking:**
  + **CRUD operations for managing transactions.**
* **AI-driven Auto-Categorization:**
  + **Using Hugging Face Transformers to automatically classify expenses.**
* **Data Visualization:**
  + **Interactive charts and summaries.**
* **Budget Alerts and Notifications:**
  + **Alerting users when nearing set budget limits.**
* **Responsive Web Design:**
  + **Ensuring usability across desktops, tablets, and mobile devices.**

## 1.3 Definitions

| **Term** | **Definition** |
| --- | --- |
| **JWT** | **JSON Web Token for secure authentication.** |
| **NLP** | **Natural Language Processing for text analysis.** |
| **MVC** | **Model-View-Controller architectural pattern.** |

# 2. Overall Description

## 2.1 Product Perspective

**System Type: Standalone Web-based application.  
Integration Points:**

* **AI Integration:**
  + **Hugging Face Transformers for NLP-based auto-categorization.**
* **Analytics:**
  + **Google Analytics for monitoring usage and performance.**

## 2.2 User Classes

| **User Class** | **Description** |
| --- | --- |
| **End User** | **Tracks personal expenses/incomes and views analytics.** |
| **Admin** | **Manages user accounts and monitors system performance.** |

## 2.3 Operating Environment

* **Frontend: React.js, Chart.js, TailwindCSS.**
* **Backend: Node.js, Express.js, MongoDB.**
* **Browsers: Chrome, Firefox, Safari, Edge.**
* **Devices: Desktop, tablet, mobile (responsive design).**

## 2.4 Design and Implementation Constraints

* **Security:**
  + **Secure communication via HTTPS.**
  + **JWT-based session management.**
* **Performance:**
  + **Page load time under 2 seconds.**
* **Scalability:**
  + **Basic scalability requirements suited for a student project without enterprise-level demands.**

# 3. Functional Requirements

## 3.1 User Management Module

### 3.1.1 Authentication

* **JWT-based Login/Signup:  
  Secure authentication and session management using JSON Web Tokens with email and password.**

### 3.1.2 Profile Management

* **Profile Updates:  
  Users can view and update personal details and preferences.**
* **Account Settings:  
  Manage basic notification preferences and privacy settings.**

## 3.2 Expense/Income Tracking

| **ID** | **Requirement** | **Description** |
| --- | --- | --- |
| **FR-01** | **Add Transaction** | **Users can log a transaction with amount, auto-detected category (via AI), date, and description.** |
| **FR-02** | **Edit/Delete Entry** | **Users can modify or remove existing transactions.** |
| **FR-03** | **Recurring Expenses** | **Support for setting up repeating transactions (e.g., monthly rent).** |

## 3.3 AI Features

### 3.3.1 Auto-Categorization

* **Functionality:  
  Automatically classify transaction descriptions using a Hugging Face Transformer model. For example, convert “Uber ride” to the “Transport” category.**
* **Implementation Approach:**
  1. **Data Preparation:  
     Collect historical transaction data with category labels.**
  2. **Model Setup:  
     Utilize pre-trained NLP models from Hugging Face and fine-tune if necessary.**
  3. **Endpoint Integration:  
     Implement a RESTful API endpoint (e.g., /api/categorize) in the Node.js backend that processes transaction descriptions and returns the predicted category.**
* **MERN Integration:  
  The React frontend calls this endpoint when a new transaction is created or updated.**

## 3.4 Analytics & Reporting

| **ID** | **Requirement** | **Description** |
| --- | --- | --- |
| **FR-04** | **Monthly Summary** | **Generate reports comparing income versus expenses and net savings.** |
| **FR-05** | **Export Data** | **Allow users to download reports in PDF/CSV format.** |

## 3.5 Admin Panel (Optional)

| **ID** | **Requirement** | **Description** |
| --- | --- | --- |
| **FR-06** | **User Analytics** | **View aggregated spending patterns and key user metrics.** |

# 4. Non-Functional Requirements

## 4.1 Performance

* **NFR-01:  
  Page load times must be less than 2 seconds.**
* **NFR-02:  
  The system should handle a reasonable number of concurrent users suitable for a small-scale deployment.**

## 4.2 Security

* **NFR-03:  
  All data exchanges must be encrypted using HTTPS.**
* **NFR-04:  
  Use bcrypt for secure password storage and hashing.**

## 4.3 Usability

* **NFR-05:  
  The application must be mobile-friendly and responsive.**
* **NFR-06:  
  Ensure compliance with WCAG 2.0 AA standards for accessibility.**

## 4.4 Reliability

* **NFR-07:  
  Achieve a high level of uptime during development and testing phases.**
* **NFR-08:  
  Implement automated daily backups to safeguard data.**

# 5. System Features (Use Cases)

## 5.1 AI-Powered Auto-Categorization

* **Scenario:  
  A user enters a transaction description such as “$15 for lunch at Cafe Nero.” The system sends this data to the auto-categorization service that uses a Hugging Face Transformer model to predict the category “Food.” The suggested category is then confirmed by the user before saving the transaction.**

## 5.2 Budget Alert

* **Scenario:  
  A user sets a budget limit (e.g., $500/month for food). The system monitors spending and sends an alert when the expenditure nears the set limit.**

## 5.3 User Management Use Cases

* **User Registration & Login:  
  Users sign up using an email/password combination, secured via JWT.**
* **Password Reset:  
  A secure workflow for users to reset their forgotten passwords.**
* **Profile Management:  
  Users can update their personal information and adjust account settings as needed.**

# 6. External Interface Requirements

## 6.1 User Interfaces

* **Dashboard:  
  Displays summary cards, charts, and recent transaction lists.**
* **Expense Form:  
  A user-friendly interface for entering new transactions, with auto-complete and suggestions based on historical data.**

## 6.2 Software Interfaces

| **Integration** | **Purpose** | **Technology** |
| --- | --- | --- |
| **Hugging Face Transformers** | **Auto-categorization via NLP** | **REST API integration in Node.js backend** |
| **RESTful APIs** | **Communication between frontend and backend** | **Node.js, Express.js** |

## 6.3 Hardware Interfaces

* **Server:  
  Hosted on AWS EC2 or Render.**
* **Database:  
  Managed via MongoDB Atlas.**

**Appendix A: Tech Stack**

| **Layer** | **Technology** |
| --- | --- |
| **Frontend** | **React.js, Chart.js, Tailwind CSS** |
| **Backend** | **Node.js, Express.js, MongoDB** |
| **AI** | **Hugging Face Transformers** |

**📑 Index**

1. Title Page
2. Index Page
3. **Part 2**: Software Design Specification (SDS)
4. **Part 3**: Front-End Implementation
5. **Part 4**: Back-End Implementation
6. Screenshots

# Software Design Specification (SDS)

## System Overview

The AI-Powered Expense Tracker allows users to register, login, add income or expenses, and get real-time category predictions for each transaction using Hugging Face's NLP models.

## Architectural Design

* **Frontend:** React.js
* **Backend:** Express.js + Node.js
* **Database:** MongoDB (local or Atlas)
* **AI Integration:** Hugging Face Model API (joeddav/xlm-roberta-large-xnli)
* **Authentication:** JWT stored in localStorage

## Folder Structure (Simplified)

Expense\_Tracker/

├── client/ # React Frontend

│ └── src/pages/ # Login, Register, Dashboard Pages

├── server/ # Express Backend

│ ├── routes/ # API Routes (auth, transactions)

│ ├── utils/ # predictCategory.js (AI)

│ ├── models/ # Transaction.js, User.js

│ └── index.js # Entry point

## Data Models

**1. User**

{

"name": "String",

"email": "String",

"password": "String (Hashed)"

}

**2. Transaction**

{

"title": "String",

"amount": "Number",

"type": "income | expense",

"userId": "ObjectId",

"category": "String",

"rawCategory": "String",

"date": "Date"

}

## Security

* JWT used for authentication.
* Protected routes using PrivateRoute.js

## Front-End Implementation

### Technologies Used

* **React.js**
* **Tailwind CSS** (or optional custom CSS for styling)
* **Chart.js** for pie charts
* **React Router** for navigation

Pages & Components

* /login: Login page with localStorage check
* /register: Registration page
* /dashboard: Main dashboard page (protected)
* Components: Logout button, transaction form, pie chart, filter buttons

### Notable Features

* Real-time transaction updates
* Dynamic pie chart
* Conditional UI rendering (login redirect, auth check)
* Responsive design (works on small screens)

## Back-End Implementation

Technologies Used

* **Express.js**, **MongoDB**, **Mongoose**, **bcryptjs**, **jsonwebtoken**

### Routes

**/api/auth/register**

* Registers a new user (password hashed)

**/api/auth/login**

* Verifies credentials and returns JWT token + user ID

**/api/transactions/add**

* Adds transaction, uses predictCategory.js to assign category via Hugging Face API

**/api/transactions/:userId**

* Gets all transactions for a user

**/api/transactions/summary/:userId**

* Returns income, expense, balance, and breakdown

**/api/transactions/:id [DELETE]**

* Deletes a transaction by ID

**AI Integration: predictCategory.js**

* Uses fetch with HUGGINGFACE\_TOKEN to call Hugging Face inference API
* Model: joeddav/xlm-roberta-large-xnli
* Predicts most relevant category from list (Food, Bills, Shopping, etc.)

### Screenshots

* Login Page

A screenshot of a login screen

AI-generated content may be incorrect.

* Dashboard Page (Transactions List)

A screenshot of a computer

AI-generated content may be incorrect.

* Add Transaction Form
* A screenshot of a computer

  AI-generated content may be incorrect.
* Pie Chart View

A blue and red pie chart

AI-generated content may be incorrect.

* Category Prediction Examples

A screenshot of a computer

AI-generated content may be incorrect.

### How to Run (Deployment Instructions)

**Prerequisites**

* Node.js & npm
* MongoDB (local or Atlas)

**Setup**

cd server && npm install

cd ../client && npm install

**Environment Variables (server/.env)**

MONGO\_URI=<your\_mongo\_uri>

HUGGINGFACE\_TOKEN=<your\_hf\_token>

**Start the App**

# In one terminal:

cd server && npm start

# In another:

cd client && npm start

**Final Notes**

* This project showcases full-stack skills including user auth, AI inference, state management, and responsive UI.
* Hugging Face NLP allows real-world category predictions.
* MongoDB makes the system scalable for multiple users.

**Thank you!**