**Project Initiation**

**Project Proposal**

* **Project Title:** *Smart Spend: AI-Powered Personal Finance Manager*  
  **Client:** University students and young professionals seeking an intelligent, user-friendly financial management tool.
* **Client Requirements:**  
  The client requires a modern financial management solution that can:
* Automatically detect and categorize user expenses using AI, minimizing manual effort.
* Analyze financial habits to deliver personalized insights and helpful suggestions.
* Track recurring bills and send timely reminders so users never miss a due date.
* Provide a clean, responsive, and visually appealing interface that works seamlessly across devices.
* Prioritize the highest standards of security and privacy to protect users' sensitive financial information.
* **Objectives:**  
  This project aims to create a personal finance app that works smoothly on both web and mobile, using the power of AI to help users stay on top of their finances with less effort. The main goals include:
* Making it easy for users to upload receipts, with AI handling the data extraction automatically.
* Instantly categorizing expenses and predicting future spending based on trends.
* Spotting spending habits, unusual transactions, and areas where users can save money.
* Keeping users engaged with personalized insights and timely alerts that help them make smarter financial choices.
* **Scope:**  
  The scope of Smart Spend includes:
* **Cross-platform development** using React Native for the frontend and .NET Core with SQL Server for a strong and scalable backend.
* **AI integration** with OCR to scan and extract data from receipts, and TensorFlow for smart, predictive financial analytics.
* **High-level security** following ISO 27001 standards to ensure user data remains safe and protected.
* **Responsive and user-friendly design** tailored for both mobile and desktop platforms, ensuring a smooth and intuitive experience.
* **Robust testing and deployment** with CI/CD pipelines, automated testing tools, and manual user testing to deliver a reliable and polished app.

**Cost and time estimation**:

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Estimation Inputs:**

* **New Code:** 8,500 SLOC
* **Reused Code:** 1,300 SLOC
* **Modified Code:** 985 SLOC
* Adjustment factors: Understanding (0.3), Unfamiliarity (0.16)

**Key Configuration Parameters:**

* **Scale Drivers:** High precedence, very high architecture/risk resolution, nominal flexibility.
* **Cost Drivers:**
  + **Product factors:** High reliability, very high complexity, extra high reusability.
  + **Personnel factors:** Very high application and platform experience, nominal language/toolset experience.
  + **Platform factors:** High platform volatility, nominal time constraints.
* **Labor Rate:** $5,000 per person-month

**Monte Carlo Simulation:**

* **Iterations:** 10,000
* **Software Equivalent Size (KSLOC):** Most values ranged between **7.8 to 10 KSLOC**, with the peak around **8–10 KSLOC**.

**Estimated Effort and Cost:**

* Based on the tool’s output, the **estimated person-months** ranged mostly around **7.8 to 10**, with moderate variation.
* Assuming an average of **9 person-months**, the **total cost** is estimated to be around:  
  **9 person-months × $5,000 = $45,000**

**Project Planning**

* **WBS for the project**

A computer screen shot of a chart

AI-generated content may be incorrect.

Smart Spend is all about helping users take control of their finances with the help of AI. To build this smart, secure, and user-friendly platform, the project is broken down into six key phases—each focused on getting us closer to a polished, high-impact personal finance app.

**1. Planning (SMAR-1)**

Before jumping into development, we lay the groundwork by answering the “what, when, who, and how” of the project.

* **Define Scope:** Clarify what Smart Spend will deliver—AI expense tracking, receipt scanning, predictions, and personalized insights.
* **Deliverables:** List all the things we’ll produce: the app itself, reports, documentation, etc.
* **Schedule:** Map out a realistic timeline with clear milestones.
* **Resources:** Identify the team, tools, and technologies we need.
* **Risk Management:** Spot any potential issues early and have a plan to deal with them.

**2. System Design (SMAR-2)**

This is where we architect the solution—everything from how data flows to how the app looks and feels.

* **Database Schema:** Structure how financial data like transactions, categories, and user info will be stored.
* **ER Diagram:** Visually map relationships between key data elements.
* **UI Design:** Craft a clean, intuitive interface that’s responsive for both mobile and desktop users.

**3. Development (SMAR-3)**

Now it’s time to build!

* **Backend:** Develop the core logic and s using .NET Core and SQL Server.
* **Frontend:** Design the app interface using React Native to ensure it works well on both Android and iOS.
* **AI Features:** Implement smart features like OCR for reading receipts and machine learning for predicting spending.
* **Project Management:** Keep everything on track with regular check-ins, version control, and task tracking.

**4. Testing (SMAR-4)**

We thoroughly test everything to make sure it all works smoothly and securely.

* **Unit Testing:** Test small code units to ensure they do what they’re supposed to.
* **Integration Testing:** Make sure all components—frontend, backend, and AI—work together without issues.
* **System Testing:** Test the entire app.
* **User Testing:** Get feedback from real users to improve experience and usability.
* **Performance Testing:** Check how the app performs under load.
* **Security Testing:** Ensure the app protects user data and complies with security standards.

**5. Deployment (SMAR-5)**

Once it’s tested and polished, we prepare to launch!

* **DevOps:** Set up automated pipelines to build, test, and deploy the app efficiently.
* **Data Migration:** Move any necessary data to the production environment.
* **Launch:** Release SmartSpend to users through app stores and the web.
* **Training:** Provide helpful onboarding material or walkthroughs to make it easy for users to get started.

**6. Maintenance (SMAR-6)**

After launch, we keep improving and supporting the app.

* **Bug Fixing:** Address issues users report.
* **Updates:** Continuously release improvements and new features.
* **Monitoring:** Track the system’s health and user activity.
* **Support:** Offer assistance through support channels to help users with questions or problems.
* **Backup:** Regularly back up data to prevent any potential loss.

**Bug Tracking & Technical Debt**

We keep track of known issues and areas that need extra attention during development, including:

* Backend not syncing properly
* Recurring test cases failing
* Backend inconsistencies during system testing

These are flagged for improvement to ensure a smooth experience.

**Project Schedule:**

A screenshot of a computer

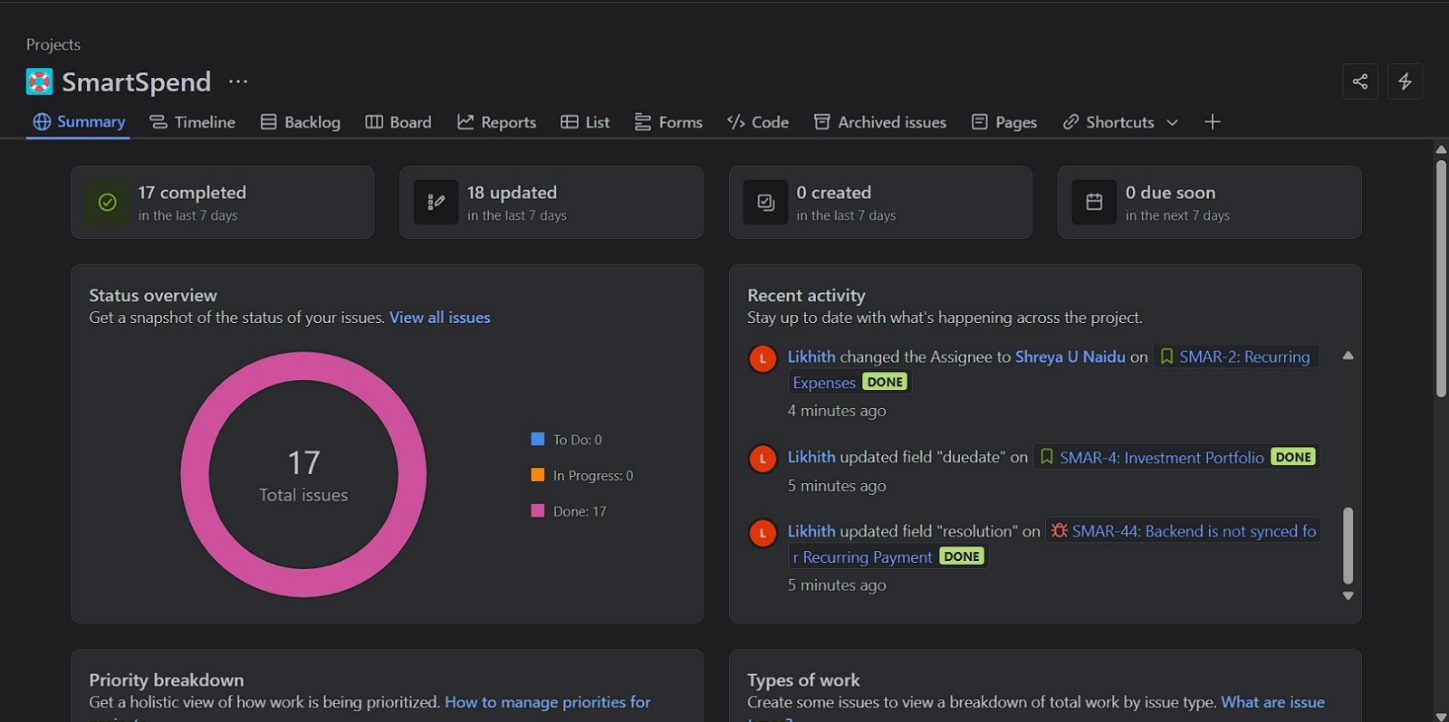
AI-generated content may be incorrect.

**Sprint Backlog**

A screenshot of a computer

AI-generated content may be incorrect.

**Overall Summary of the Project**



**Risk Management**

**Risk Identification:**

| **Risk** | **Explanation** | **Category** |
| --- | --- | --- |
| **Data Security and Privacy Risk** | The application handles sensitive user financial data. Weak encryption, misconfigured database rules, or security breaches could expose this data. | Technical |
| **Scope Creep Risk** | As the project progresses, additional features may be requested, expanding the original scope and affecting timelines and resources. | Organizational |
| **Integration and Compatibility** | Issues might occur while integrating third-party s (e.g., currency conversion) or ensuring compatibility across browsers and devices. | Technical |
| **Performance and Scalability Risk** | The system may slow down or fail under heavy user load or with real-time operations, especially if not optimized for scale. | Technical |
| **User Adoption and Training Risk** | Users may find advanced features like forecasting, budget ratios, or AI suggestions confusing, leading to low adoption and satisfaction. | External / Organizational |

**Risk Analysis**:

**Risk Assessment Table**

| **Risk** | **Explanation** | **Category** | **Likelihood (1–5)** | **Impact (1–5)** | **Risk Level** |
| --- | --- | --- | --- | --- | --- |
| Data Security and Privacy Risk | Sensitive financial data could be exposed due to weak encryption or poor security rules. | Technical / Security | 4 | 5 | 20 |
| Scope Creep Risk | Additional features may be requested beyond the original scope. | Organizational / Project | 4 | 4 | 16 |
| Integration and Compatibility Risk | Issues when integrating s or ensuring cross-browser/device compatibility. | Technical | 3 | 4 | 12 |
| Performance and Scalability Risk | Real-time updates and user growth may lead to slowdowns or system crashes. | Technical / Performance | 3 | 3 | 9 |
| User Adoption and Training Risk | Users may find advanced AI features confusing, leading to low usage. | Organizational / External | 2 | 3 | 6 |

**Risk Matrix**

|  | **Impact: 1** | **Impact: 2** | **Impact: 3 (Medium)** | **Impact: 4** | **Impact: 5 (High)** |
| --- | --- | --- | --- | --- | --- |
| **Likelihood: 5** |  |  |  |  |  |
| **Likelihood: 4** |  |  |  | Scope Creep (16) | Data Security (20) |
| **Likelihood: 3** |  |  | Performance (9) | Integration (12) |  |
| **Likelihood: 2** |  |  | User Adoption (6) |  |  |
| **Likelihood: 1** |  |  |  |  |  |

**Risk Mitigation Plan**:

**Risk Mitigation Plan**

| **Risk** | **Category** | **Mitigation Strategies** |
| --- | --- | --- |
| **Data Security & Privacy** | Security Risk | - Implement **AES-256 encryption** for data storage and transfer. - Enforce strict **database access controls**.  - Align with **ISO 27001** compliance for managing data privacy and security. |
| **Scope Creep** | Project Management Risk | - Define a **clear project scope** in the proposal stage.  - Establish a **formal change control process** for evaluating new feature requests. |
| **Integration & Compatibility** | Technical Risk | - Conduct **early integration testing** of external s and modules.  - Use **reliable, well-documented third-party s** to ensure long-term compatibility. |
| **Performance & Scalability** | Performance Risk | - Optimize database operations using **indexing and pagination**.  - Implement **client-side caching** for repeated queries.  - Perform **load testing** using tools like Apache JMeter. |
| **User Adoption & Training** | Operational / Organizational Risk | - Design an **intuitive user interface** with onboarding walkthroughs.  - Provide **tutorial videos, FAQs, and in-app help** to support users at every step. |

**Project Quality Assurance**

**Quality Assurance Plan**

| **QA Area** | **Focus** | **Strategies & Tools** |
| --- | --- | --- |
| **Code Quality Assurance** | Maintain clean, consistent, and reliable code | - Conduct **code reviews** via GitHub pull requests.  - Follow **branching strategy** (feature, bugfix, dev).  - Use **ESLint** to enforce coding standards. |
| **Automated Testing** | Ensure code correctness and stability | - Perform **unit testing** (e.g., using PyTest or Django's built-in test suite).  - Conduct **integration testing** with **Postman** or **Cypress**.  - Use **GitHub Actions** for **Continuous Integration (CI)** to auto-run tests on every commit. |
| **Manual Testing** | Validate usability and functionality | - Do **UI/UX testing** across multiple browsers/devices.  - Conduct **User Acceptance Testing (UAT)** with real users to gather feedback before release. |
| **Performance & Load Testing** | Ensure responsiveness and scalability | - Monitor speed using tools like **Firebase Performance Monitoring**.  - Simulate load with **Apache JMeter** to test app under stress. |
| **QA Standards & Documentation** | Maintain standards and issue tracking | - Maintain detailed **documentation** and testing reports. - Track bugs using tools like **Jira** or **Trello** for structured resolution workflow. |

**Quality Assurance Strategy**

**1. Code Quality Assurance**

Maintaining clean, readable, and maintainable code is foundational for team collaboration and long-term project success. We enforce a consistent coding style using tools like **ESLint** (for JavaScript) and rely on structured **GitHub pull requests** for code reviews. A **branching strategy** is followed (e.g., feature/, bugfix/, and dev/ branches) to isolate changes and prevent unstable code from entering production.

**2. Automated Testing**

Automated testing ensures early detection of bugs and code regressions. We implement:

* **Unit tests** using pytest or Django’s test framework to test individual components and business logic.
* **Integration tests** via Postman or Cypress to validate the interaction between modules and s.
* All tests are integrated with **GitHub Actions**, enabling Continuous Integration (CI). This setup automatically runs test suites on every push or pull request, ensuring no broken code is merged into the main branch.

**3. Manual Testing**

Manual testing plays a vital role in identifying usability issues and UI/UX bugs that automated tools may miss. We:

* Conduct **cross-browser and cross-device testing** to ensure the interface works seamlessly across environments.
* Perform **User Acceptance Testing (UAT)** with end-users to simulate real-world scenarios and collect usability feedback before deploying to production.

**4. Performance & Load Testing**

To ensure our application is scalable and responsive under different loads:

* We use **Firebase Performance Monitoring** or browser dev tools to track page speed and responsiveness in real-world usage.
* We simulate concurrent users and stress-test the backend using tools like **Apache JMeter**, helping identify bottlenecks and performance limitations early in the development cycle.

**5. QA Standards & Documentation**

Consistency is enforced through well-defined QA processes and documentation:

* All tests and results are thoroughly documented, ensuring traceability and accountability.
* Bugs, tasks, and improvements are tracked using project management tools like **Jira** or **Trello**, ensuring structured workflows and timely resolution.
* Each release is accompanied by testing reports and changelogs for better transparency and future audits.

**Design**  
  
**UML Class Diagram:**

A class diagram is like a blueprint that helps us understand the structure of a software system before we build it. It’s part of the Unified Modelling Language (UML) and shows us what the system is made of and how different parts connect.

Here’s what it includes:

* **Classes:** Think of these as the main building blocks—representing real-world things like "User," "Transaction," or "Account."
* **Attributes:** These are the details or data each class holds. For example, a “User” class might have a name, email, and password.
* **Methods (Operations):** These are the actions a class can perform—like logging in, making a payment, or generating a report.
* **Relationships:** This show how classes are connected—like one class inheriting from another, working together, or depending on each other.

**Step 1: Domain Analysis and Class Identification**

* **Authentication**: Handles user login, registration, sessions.
* **Financial Core**: Manages expenses, transactions, and accounts.
* **Features**: Includes recurring transactions, receipts, and shared expenses.
* **Analytics**: Supports categorization, forecasting, and budget alerts.
* **Integration**: handling, currency conversion, email alerts.
* Key classes identified: User, Account, Transaction, Category, Budget, Recurring Transaction, Split Expense, Currency Converter, Notification Service.

**Step 2: Class Definition and Properties**

**Transaction**:

* Attributes: transactionID, amount, date, description, category.
* Methods: convertCurrency(), isRecurring().

**User**:

* Attributes: userID, username, email, password.
* Methods: login(), register().

**Budget**:

* Attributes: monthlyLimit, dailyLimit.
* Methods: checkLimit(), sendAlert().

**Step 3: Establishing Relationships**

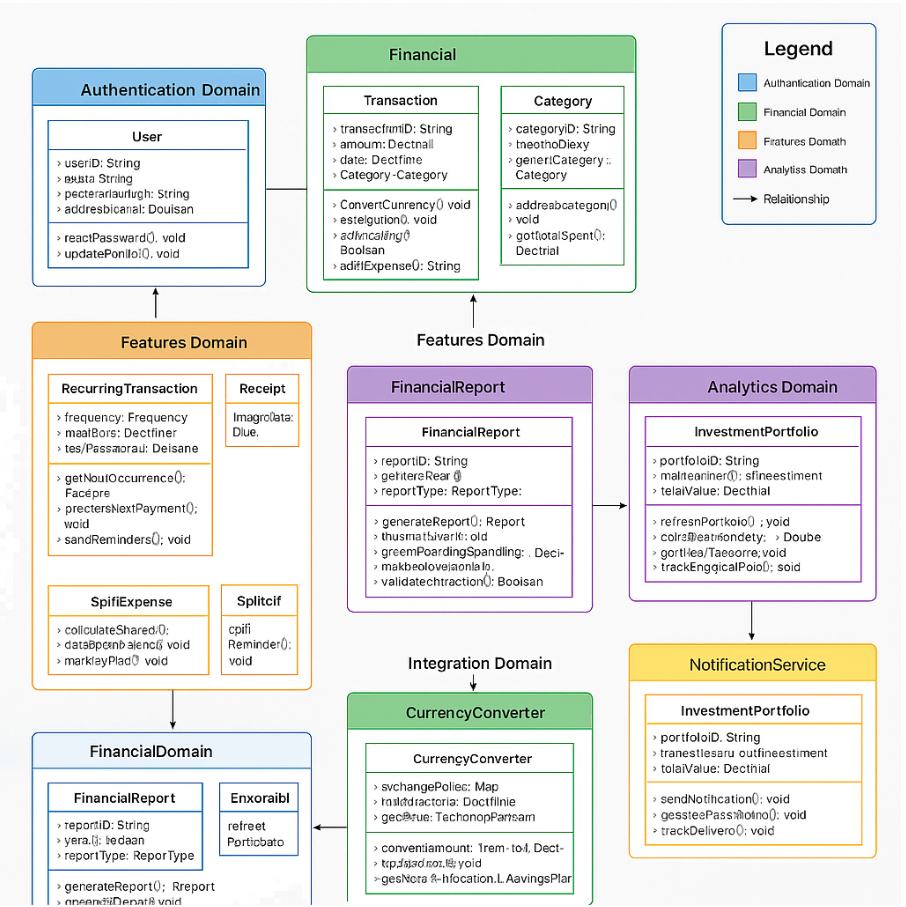
* Relationships mapped with cardinality:
* One User can own multiple Accounts.
* Each Transaction belongs to one Account and one Category.
* One Budget is associated with a User.
* RecurringTransaction inherits from Transaction.
* Dependency established from NotificationService to User for alert purposes.
* Aggregation used for SplitExpense, where one transaction involves multiple users.

**Step 4: Visual Organization and Layout**

* Classes grouped visually by domain for clarity.
* Core functional classes like User, Transaction, and Category placed centrally.
* Used UML notations like solid lines (association), arrows (inheritance), and diamonds (aggregation).
* Added a color-coded legend for domain-specific grouping.

**Step 5: Refinement and Validation**

* Walked through real user scenarios like adding an expense, receiving a forecast alert, or sharing a transaction to validate relationships and responsibilities.
* Refined class responsibilities and method coverage.
* Added missing service classes like Currency Converter and Notification Service.
* Final UML diagram made visually appealing and logically structured for both developers and stakeholders.

****

**ER DIAGRAM**

An Entity-Relationship Diagram (ERD) is the ideal choice for our Smart Spend AI-Powered Personal Finance Manager project for several compelling reasons:

While a class diagram helped us visualize the software architecture and object interactions, the ERD serves a different but equally crucial purpose—it provides a clear blueprint of our database structure, which is the foundation of our entire application.

Smart Spend is fundamentally a data-intensive application that manages financial information across multiple domains (transactions, accounts, budgets, investments). The ERD gives us a precise view of how this data will be organized and related in our database, making it an essential tool for both development and communication.

The ERD explicitly shows all entity attributes with their data types and clearly defines primary and foreign keys—critical information that isn't as visible in class diagrams. For a financial application where data integrity is paramount, these explicit relationships are vital to ensure no financial data is ever orphaned or incorrectly associated.

**Key Entities and Relationships in the Smart Spend ERD**

**Key Entities**

1. **USERS**: Stores user account information including login credentials and personal details
2. **ACCOUNTS**: Represents financial accounts like checking, savings, or credit cards
3. **TRANSACTIONS**: Records individual financial transactions (purchases, deposits, etc.)
4. **CATEGORIES**: Classification system for transactions (groceries, entertainment, etc.)
5. **BUDGETS**: Spending limits set by users for specific categories
6. **RECURRING\_TRANSACTIONS**: Scheduled regular payments like subscriptions or bills
7. **RECEIPTS**: Digitized records of physical or electronic receipts
8. **SPLIT\_EXPENSES**: Transactions shared between multiple people
9. **SPLIT\_PARTICIPANTS**: Individuals involved in a split expense
10. **INVESTMENT\_PORTFOLIOS**: Collections of investment assets
11. **INVESTMENTS**: Individual investment holdings
12. **FINANCIAL\_INSIGHTS**: AI-generated observations about financial patterns
13. **USER\_PREFERENCES**: User settings and customization options

**Key Relationships**

1. **USER to ACCOUNT**: One-to-many (1) - A user can have multiple accounts
2. **USER to BUDGET**: One-to-many (1) - A user can set multiple budgets
3. **USER to USER\_PREFERENCE**: One-to-one (1:1) - Each user has one set of preferences
4. **ACCOUNT to TRANSACTION**: One-to-many (1) - An account can have many transactions
5. **ACCOUNT to RECURRING\_TRANSACTION**: One-to-many (1) - An account can have multiple recurring transactions
6. **CATEGORY to TRANSACTION**: One-to-many (1) - A category can be applied to many transactions
7. **CATEGORY to BUDGET**: One-to-many (1) - A category can have multiple budgets (e.g., across different users)
8. **CATEGORY to RECURRING\_TRANSACTION**: One-to-many (1) - A category can be applied to many recurring transactions
9. **CATEGORY to CATEGORY**: One-to-many (1) - Self-referential relationship for parent-child categories
10. **TRANSACTION to RECEIPT**: One-to-zero-or-one (1:0..1) - A transaction may have one receipt
11. **TRANSACTION to SPLIT\_EXPENSE**: One-to-zero-or-one (1:0..1) - A transaction may be split
12. **SPLIT\_EXPENSE to SPLIT\_PARTICIPANT**: One-to-many (1) - A split expense involves multiple participants
13. **USER to INVESTMENT\_PORTFOLIO**: One-to-many () - A user can have multiple investment portfolios
14. **INVESTMENT\_PORTFOLIO to INVESTMENT**: One-to-many (1) - A portfolio contains multiple investments
15. **USER to FINANCIAL\_INSIGHT**: One-to-many (1) - A user can receive multiple financial insights

A screenshot of a computer

AI-generated content may be incorrect.

**Implementation**

| Component | Tool / Framework | **Why It Was Chosen** |
| --- | --- | --- |
| Backend | Django + Django REST Framework | Offers a secure, scalable backend with built-in user authentication, admin interface, and ORM for database queries. Perfect for rd development. |
| Frontend | React + Vite + TailwindCSS | React ensures a dynamic and responsive user interface, Vite speeds up development builds, and TailwindCSS allows for fast, customizable UI styling. |
| Database | PostgreSQL | A robust relational database with strong support for transactions, ideal for managing complex financial data. |
| AI/Analytics | TensorFlow, Pandas, Scikit-learn | Used for implementing predictive features like forecasting expenses and categorizing transactions using Machine Learning. |
| Data Visualization | Chart.js | Offers interactive and appealing charts to help users visualize their spending trends and financial health. |
| File Handling | pandas, openpyxl | Used for importing/exporting transactions from/to CSV/XLSX, making it easier for users to manage their data externally. |
| Notification System | Django Email, SMS Gateway s | Sends reminders for due bills, spending limits, and important alerts, enhancing user control and peace of mind. |
| Security | SSL Encryption + ISO 27001 Guidelines | We implemented secure protocols and followed international standards to ensure financial data is protected. |
| Testing & CI/CD | Postman, GitHub Actions, JMeter | Used for testing, automation of test pipelines, and load testing to ensure performance at scale. |
| UX Tools | Figma / Canva (for UI prototyping) | Helped design clean, accessible, and intuitive user interfaces, especially for mobile and desktop views. |

A green and white chart with text

AI-generated content may be incorrect.

**Smart Spend Technical Stack: Key Points**

1. **Backend Framework:** Django with Django REST Framework provides robust authentication, admin interfaces, and a powerful ORM for database interactions.
2. **Frontend Framework:** React paired with Vite (build tool) and Tailwind CSS (styling library) creates a responsive, high-performance user interface with accelerated development.
3. **Database:** PostgreSQL delivers strong ACID compliance, complex relationship support, and advanced querying capabilities for reliable financial data storage.
4. **AI/Analytics:** TensorFlow enables predictive modelling and time-series forecasting for spending patterns and financial projections.
5. **Data Visualization:** Chart.js handles interactive financial graphs while CSV/XLSX export functionality provides data portability.
6. **Development Approach:** Web-first design with responsive capabilities, suited for Agile/Kanban methodology and supported by automated testing.

**Images of Our Front-End:**

**DASHBOARD:**

A screenshot of a computer

AI-generated content may be incorrect., Picture

A screenshot of a computer

AI-generated content may be incorrect., Picture

**CODE FOR DASHBOARD:**

A screenshot of a computer program

AI-generated content may be incorrect.

**INCOME PAGE:**

A screenshot of a computer

AI-generated content may be incorrect., Picture

A screenshot of a computer

AI-generated content may be incorrect., Picture

**CODE FOR INCOME PAGE:**

A screenshot of a computer

AI-generated content may be incorrect.

**EXPENSES PAGE:**

A screenshot of a computer

AI-generated content may be incorrect., Picture

**AI GENERATED FORECAST OF EXPENSES:**

A screenshot of a computer

AI-generated content may be incorrect., Picture

A graph showing a line

AI-generated content may be incorrect., Picture

A screenshot of a computer

AI-generated content may be incorrect., Picture

A screenshot of a computer

AI-generated content may be incorrect., Picture

A screenshot of a computer

AI-generated content may be incorrect., Picture

**CODE FOR LOGIN PAGE:**  
A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer screen

AI-generated content may be incorrect.

**Testing**

**Manual Testing Table**

| **Test Case** | **Test Steps** | **Expected Result** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- |
| **1. User Login** | Enter valid credentials and click “Login” | User should be redirected to dashboard | As expected | Pass |
| **2. User Registration** | Fill registration form with valid inputs and submit | Account is created and redirected to login page | As expected | Pass |
| **3. Add New Expense** | Fill in amount, category, description, date, then submit | Expense should be saved and shown in transaction list | As expected | Pass |
| **4. Edit Expense Entry** | Click edit icon on an expense, modify fields, and update | Expense details should be updated and reflected in the UI | As expected | Pass |
| **5. Delete Expense** | Click delete icon on an expense and confirm deletion | Expense entry should be removed from list | As expected | Pass |
| **6. View Expense Summary** | Navigate to dashboard page | Visual summary of expenses with charts is displayed | As expected | Pass |
| **7. Expense Trend Prediction** | Trigger GET request to //predict-expense/ from Postman | System returns JSON with next month's predicted expense and chart image is generated | As expected | Pass |
| **8. Financial Chatbot Response** | Send POST to //chatbot/ with question: "What is my biggest expense?" | Chatbot should return an AI-generated insight based on transaction history | As expected | Pass |
| **9. Invalid Form Submission** | Leave required fields blank in Add Expense form and submit | System should show validation error messages | As expected | Pass |
| **10. Unauthorized Access** | Try accessing a protected endpoint (like /predict-category/) without logging in | Server returns an authentication error or 403 forbidden | As expected | Pass |

**Automated Testing**

**Authentication Module – Integration Testing**

| **Module** | **Test Case** | **Type** | **Expected Result** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- | --- |
| Authentication | Login request | Automated | Valid output returned on successful login | Output returned successfully | Pass |
| Authentication | User registration request | Automated | User created in database with is\_active=False | Inactive user created | Pass |
| Authentication | Email validation for existing email | Automated | Error message: “email in use” | Message returned correctly | Pass |
| Authentication | Email validation with valid new email | Automated | Status 200 with email\_valid: true | Valid response received | Pass |
| Authentication | Login request with inactive user | Automated | Error shown: “Account is not active” | Proper error shown | Pass |
| Authentication | Invalid credentials login | Automated | Message: “Invalid credentials” | Error message shown | Pass |

**Expense Module – Integration Testing**

| **Module** | **Test Case** | **Type** | **Expected Result** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- | --- |
| Expense | Add new expense and reflect in list | Automated | Expense saved and shown in transaction history | Appears in list | Pass |
| Expense | Edit expense updates record | Automated | Updated amount and description visible in dashboard | Updated successfully | Pass |
| Expense | Delete expense removes entry | Automated | Expense removed from database | Deleted successfully | Pass |
| Expense | Prevent future date expense | Automated | Form error displayed for invalid date | Validation error shown | Pass |
| Expense | Empty description in add expense | Automated | Error message for missing required field | Error shown | Pass |
| Expense Stats | Get expense summary by category | Automated | JSON with correct category totals returned | Accurate summary returned | Pass |
| Expense Search | Search by description or category | Automated | Matching expenses returned in response | Expected entries found | Pass |
| Expense Sort Feature | Sort expenses by amount descending | Automated | Highest expense appears first | Sorted correctly | Pass |

**User Income, Bills & Limits – Integration Testing**

| **Module** | **Test Case** | **Type** | **Expected Result** | **Actual Result** | **Status** |
| --- | --- | --- | --- | --- | --- |
| Bill Management | Add a new bill and verify database entry | Automated | New bill is created with correct due date | Bill saved correctly | Pass |
| Bill Management | Edit bill and verify update | Automated | Updated bill reflects in listing | Updated successfully | Pass |
| Bill Management | Delete bill and ensure removal | Automated | Deleted from DB and removed from list | Deleted successfully | Pass |
| Expense Limit | Set daily expense limit | Automated | Daily limit is saved and validated | Limit saved | Pass |
| Expense Limit | Add expense below limit | Automated | No warning shown, expense saved | As expected | Pass |
| Expense Limit | Add expense above limit | Automated | Warning shown but expense still saved | Warning + entry saved | Pass |
| Statistics View | Load dashboard stats (daily/monthly/yearly) | Automated | Charts and summaries appear | Stats displayed | Pass |
| Forecast | Predict next month's expenses | Automated | Returns prediction + line chart (.png) | JSON + image returned | Pass |
| Chatbot | Chatbot answers financial question | Automated | AI-generated insight/suggestion shown | Valid reply received | Pass |

**Maintenance**

**1. Conversational AI Assistant**

Imagine managing your finances by simply asking, “How much did I spend on food last week?”—without having to click through multiple pages. That’s what a conversational AI assistant could bring to Smart Spend. By integrating a smart, natural language chatbot, users would be able to interact with the app through simple voice or text-based queries. This feature would not only improve accessibility for users with disabilities or older individuals unfamiliar with complex digital interfaces, but it would also introduce a more natural, hands-free experience. Whether you're walking, driving, or just multitasking, a conversational assistant can help you check your spending, set reminders, or even recommend saving tips—all through a quick and intuitive chat.

**2. Gamification Elements**

Let’s face it—managing finances isn’t always exciting. But what if it could be? Gamification can add a motivational twist to financial planning. By introducing elements like achievement badges, milestone rewards, and progress bars, users will feel a sense of accomplishment when they hit their budgeting goals or avoid unnecessary expenses. For example, staying under your grocery budget for three consecutive months could unlock a "Smart Saver" badge. These game-like mechanics encourage users to stay consistent with their financial habits, celebrate small wins, and stay emotionally engaged with their money journey. Over time, this can lead to more mindful spending and better financial discipline—all while making the process enjoyable.

**3. Offline Mode**

Internet access isn’t always guaranteed, especially when traveling, commuting, or in rural areas. That’s why an offline mode would be a valuable enhancement. This feature would let users add expenses, edit entries, and view past transactions even without an active connection. Then, once they’re back online, all the data would sync seamlessly to the cloud. Offline support makes the app more resilient and dependable in real-life scenarios, ensuring that your financial logging is uninterrupted no matter where you are or how patchy your internet might be.

**4. Bank Account Synchronization**

One of the most time-consuming aspects of finance tracking is manual data entry. To eliminate that hassle, future versions of SmartSpend could integrate with trusted financial data aggregators like Plaid or Yodlee. These services allow users to link their bank accounts securely so that income and expense transactions are automatically imported into the app. This real-time syncing ensures the data stays current and accurate without the need for repetitive input. It also opens the door to more powerful insights, like detecting spending trends across accounts or flagging unusual activity. For users, this means less work and more control over their complete financial picture.

**5. Two-Factor Authentication (2FA)**

As Smart Spend grows and handles increasingly sensitive user data, robust security becomes a must. Implementing Two-Factor Authentication (2FA) adds an extra layer of protection by requiring users to verify their identity through a secondary method—like a one-time code sent to their email or mobile phone. This makes it much harder for unauthorized users to access someone’s account, even if they somehow manage to obtain the password. In today’s digital world where cyber threats are real, 2FA brings peace of mind, reinforcing user trust and protecting their financial integrity.

**6. Budget Sharing and Family Accounts**

Money management isn’t always a solo activity. Families, partners, and roommates often share responsibilities when it comes to budgeting. A shared budgeting feature or family accounts setup would allow multiple users to manage a common budget under one roof. Each member could log expenses, monitor collective savings goals, and get notified of important financial events. Think of it like a group chat, but for your wallet—transparent, collaborative, and perfectly suited for households that want to stay financially aligned. It’s a game-changer for parents teaching kids about spending, or couples managing monthly bills together.

**Roles and Responsibilities**

**Student A: Varad Nair (1002161475)- Lead Developer (Frontend + backend)**

**Student B: Sai Likhith Reddy Kummathi (1002228347) - Data Analyst, AI Model Integrator**

**Student C: Shreya Umesh Naidu (1002196595) - Frontend, Testing**

**Student D: Mounisha Putta (1002240539) - UI/UX Designer, Testing**

**References**

<https://create.microsoft.com/en-us/templates/gantt-charts>

<https://statics.teams.cdn.office.net/evergreen-assets/safelinks/1/atp-safelinks.html>

<https://ksailikhithreddy08.atlassian.net/jira/software/projects/SMAR/summary>

<https://mavsuta-my.sharepoint.com/:w:/g/personal/vxn1475_mavs_uta_edu/ESGKsckTHM5MhuSKBwCebskBZepOKkcRbpAZhy03VuFLzw?wdOrigin=TEAMS-MAGLEV.p2p_ns.rwc&wdExp=TEAMS-TREATMENT&wdhostclicktime=1743307732487&web=1>