Software Project Management Plan

Financial Management: Full Stack Web Application

HashSlingingSlashers

Jeffery Kimbrow, Madeline Brothers, Yasmin Hassan, and Samantha Saunsaucie

[1 Overview 2](#_Toc1110839705)

[1.1 Purpose and Scope 3](#_Toc93473632)

[1.2 Success Criteria 3](#_Toc593970631)

[2 Planning and Organization 3](#_Toc1053393953)

[2.1 Team Organization 3](#_Toc1510383863)

[2.2 Project Communications 3](#_Toc1636203159)

[2.3 Tools 3](#_Toc775972940)

[2.4 T-Shirt and Budgeting 3](#_Toc305808901)

[3 Work Plan 4](#_Toc1792253905)

[3.1 Activities and Tasks 4](#_Toc2073453158)

[3.2 Schedule for Managing Time 4](#_Toc22125777)

[3.3 Release Plans 4](#_Toc1350586788)

[3.4 Iteration Plans 4](#_Toc338760644)

[4 Quality Plan 4](#_Toc873858148)

[4.1 Project Architecture 4](#_Toc1146312293)

[4.2 Project Measurements 4](#_Toc479260634)

[4.3 Risk Management 5](#_Toc1646454594)

[4.4 Verification and Validation Plan 5](#_Toc1717991704)

## 1 Overview

### 1.1 Purpose and Scope

The Financial Management Web Application offers a safe and easy way for anyone to keep track of their own money. The goal is to help people better understand and manage their spending by giving them tools like budget tracking, transaction history, account transfers, and tailored financial advice.  
 The goal of this project is to make managing personal finances easier, raise awareness about money issues, and make budgeting tools easier to find and use, especially for people who want a simple and centralized way to see and manage their financial data.

We will be using Python, PostgreSQL, C, and JavaScript to design and build a full-stack financial web app for this project. The app has:

* User registration and login using secure authentication (MFA and encrypted passwords).
* A customizable dashboard that shows account balances and reports of spending.
* Managing transactions provides options to search, filter, and put them into groups.
* Planning a budget and keeping track of goals.
* Moving money between linked accounts.
* AI chatbot that can answer questions and give personalized financial help.
* Features for administration and operations include backing up data

**Project Goals**

* Make a safe and user-friendly web software for managing finances that lets users see their budgets, transactions, and account balances all in one place.
* Give people the tools they need to make smart financial choices by giving them personalized insights, spending summaries, and visual reports.
* Support many languages and make sure that the user interface is clean, responsive, and works on all devices to make sure that everyone can use it.
* Add an AI chatbot to give users specialized help with their finances, help them make budgets, and answer their questions right away.
* Deliver a high-quality solution that is hosted in the cloud that shows that you can work together, plan, and know how to do things technically within the time frame of the project.
* Align with Commerce Bank's promise to be open to all users, promote innovation, and teach people about money.

**Project Objectives**

* build a full-stack online app using Python, PostgreSQL, and JavaScript, C to design
* Make a secure method for users to sign up, log in, and maintain their profiles.
* Build backend services for transactions, budgeting, chatbots, and reporting on finances.
* Create a responsive front-end interface that supports multilingual functionality and gives users a smooth experience on all devices.
* Add and connect AI chatbot to help users with questions, show them how to use the app, and provide them customized financial advice.
* Test the functionality, performance, and security to make sure they meet accessibility and data protection standards.
* Get the final documentation, presentation, and deployment of the system ready before the project is due.

### 1.2 Success Criteria

Success criteria outline the particular requirements that must be fulfilled for the Financial Management Web Application project to be considered successful. These criteria give us a way to measure how well the project is doing and what the end results will be. Each criterion is meant to be measurable and verifiable, to ensure that the goals and objectives of the project have been fully achieved. The following criteria will determine how well the Financial Management Web Application does:

**Completion of the function**

* All of the necessary functionality, such as authentication, transactions, budgeting, a chatbot, and profile management, have been fully implemented and tested.
* Without any problems and system errors, users may register, log in, see their transactions, adjust their budgets, and interact with the chatbot

**Security and Reliability**

* Multi-factor authentication (MFA) is in place and working as it should.
* All private information, like passwords and bank information, is encrypted and kept safe.
* During testing, the application stays up all the time and shows that it can handle a normal user load.

**Performance and Usability**

* The time it takes for a page to load and respond meets the performance requirement
* The app's interface is easy to use, works well on both desktop and mobile platforms, and looks the same on both.
* Accessibility and support for several languages have been checked and passed usability testing.

**Quality assurance and testing**

* Unit, integration, and end-to-end tests are completed with minimal unresolved defects.
* Security and accessibility scans are completed, and all high-priority issues are resolved.

**Deployment and Documentation**

* The final version is successfully deployed to a cloud environment.
* The documentation, which includes the user guide and technical reports, is all there and correct.
* A successful final presentation is given to the bank, showing them the system's main features and benefits.

## 2 Planning and Organization

### 2.1 Team Organization

The development team, named HashSlingingSlashers, consists of four members, each assigned distinct responsibilities to ensure the successful execution of a full-stack financial management application.

* **Project Manager**: *Jeff Kimbrow*  
   Jeff oversees the overall planning and coordination of the project. His responsibilities include developing the project plan, assigning tasks, managing risks, verifying development progress, maintaining project documentation, and organizing weekly team meetings. In addition to these duties, Jeff also serves as the team's cloud security engineer.
* **Programmers (3 Members)**:  
   All programmers contribute to application architecture planning, code documentation, and active participation in weekly meetings. They are expected to provide updates on their progress, share insights, and demonstrate completed tasks.
  + *Madeline Brothers*: Responsible for the design and development of the AI chatbot tool, along with backend development.
  + *Yasmin Hassan*: Leads the creation and maintenance of project documentation, along with backend development.
  + *Samantha Saunsaucie*: Focuses on front-end development, including UI/UX implementation and client-side functionality.

### 2.2 Project Communications

The HashSlingingSlashers team utilizes a variety of tools to ensure effective communication and collaboration throughout the project's lifecycle.

* **Discord** is the primary communication platform, used for daily messaging and scheduling weekly team meetings.
* **Box** serves as the central repository for all project documentation, including the Software Requirements Specification (SRS), Project Plan, and Architecture Plan.
* **Jira/Confluence** is used to manage tasks, track progress, and assign roles, providing transparency and accountability across the team.
* **Figma** is the chosen tool for UI/UX prototyping, allowing team members to collaborate on and review design elements in real time.
* **GitHub** is used for version control, where all code is stored, reviewed, and maintained.
* **Zoom** is used for remote meetings.

### 2.3 Tools

The HashSlingingSlashers team utilizes a comprehensive technology stack to support development, deployment, and security of the full-stack financial management application:

* **Backend Technologies**:
  + *Python*, *C*, and *Java* for core backend services.
* **Frontend Technologies**:
  + *JavaScript*, *HTML*, and *CSS* for responsive and interactive UI development.
* **Database Management**:
  + *PostgreSQL* for structured relational data storage.
* **Caching**:
  + *Redis* is implemented to manage in-memory data caching, improving system performance and reducing latency.
* **Containerization and Deployment**:
  + *Docker.io* is used to containerize application components for consistent deployment.
  + Cloud deployment is performed using platforms such as *Amazon Web Services (AWS)* and *Azure Virtual Machines*, enabling scalable and reliable infrastructure.
* **API Gateway and Load Balancing**:
  + *NGINX* is implemented as a reverse proxy and gateway to manage traffic routing and load balancing.
* **AI Integration**:
  + The application integrates a pre-trained*BERT Financial Model* from *Hugging Face*, specifically fine-tuned for financial language processing. This model enhances the AI chatbot's ability to understand and respond to finance-related queries with context-aware accuracy.
* **Security**:
  + *CSRF* protection and *JWT* (JSON Web Tokens) are used for secure session handling and API authentication.
  + *OWASP ZAP* is employed for automated vulnerability scanning to identify and mitigate common web application threats.

### 2.4 T-Shirt and Budgeting

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Task Name | Start | Finish | Duration | September/ October 2025 |
| 1 | Project Charter | 09/19/2025 | 09/22/2025 | 2 days | (S) |
| 2 | Software Requirements Document | 09/26/2025 | 10/1/2025 | 5 days | (M) |
| 3 | Architecture Document | 10/13/2025 | 10/17/2025 | 4 days | (M) |
| 4 | Project Plan | 10/10/2025 | 10/17/2025 | 7 days | (M) |
| 5 | Backend Boiler Plate | 10/3/2025 | 10/11/2025 | 8 days | (L) |
| 6 | Frontend Boiler Plate | 10/3/2025 | 10/12/2025 | 9 days | (L) |
| 7 | UI Prototype | 10/3/2025 | 10/17/2025 | 14 days | (L) |

## 3 Work Plan

### 3.1 Activities and Tasks

* Task Name
* Task Description
* Owner
* Effort Estimate
* Planned start and stop dates
* Actual start and stop dates
* Dependencies among other tasks

### 3.2 Schedule for Managing Time

### 3.3 Release Plans

**Oct 17, 2025** Prototype UI: Basic page layouts and navigation are ready. include login, dashboard, transaction, transfer, profile, budget, and chatbot.

**Oct 23, 2025** Boilerplates: Set up and push backend and frontend projects to GitHub

**Nov 7, 2025** Core features working (login, dashboard, profile). Basic database connection and API tested.

**Nov 21, 2025**  Full functionality complete (transactions, budgets, session management). Testing and security scans in progress.

**Dec 5, 2025** App fully deployed with clean UI, tested features, and presentation materials ready.

### 3.4 Iteration Plans

**Iteration 1: Oct 17–23, 2025**

**Focus:** Prototype and initial setup  
 **Goal:** Create the initial structure and visual prototype of the application.  
 **Tasks:**

* Make prototypes for the Login, Dashboard, and Transaction pages.
* Create project repositories and set up boilerplates for the backend (Python) and frontend (HTML/CSS/JavaScript).
* Set up the PostgreSQL database and check that you can connect to it.
* Set up basic page navigation and layouts that work on desktop and mobile
* Do some early tests to make sure the configuration worked

**Iteration 2: Oct 24-Nov 7**

**Focus:** Backend development

**Goal:** The development process requires creating fundamental backend services while establishing API connections between frontend and backend systems.  
 **Tasks:**

* Use Python to set up an authentication service (signup, login, MFA).
* Set up a User Profile Service to keep track of user information.
* Develop and test API endpoints for login and registration.
* Create frontend web pages for authentication and connect to backend.
* Implement security controls
* Conduct unit testing and fix initial bugs.

**Iteration 3: Nov 8-20**

**Focus:** Features of the Dashboard and Transactions

**Goal:** Enable users to view and manage financial data.  
 **Tasks:**

* Build the Accounts and Ledger Service in Python.
* Make transaction endpoints, like view, filter, search,
* Create dashboard pages that provide balances and summaries of transactions.
* Generate basic statements and reports.
* Set up object storage for saving user data.
* Conduct unit and performance testing

**Iteration 4: Nov 21–Dec 5**

**Focus:** Final Testing, Budgeting, and Chatbot  
 **Goal:** Finalize all services and prepare the application for deployment.  
 **Tasks:**

* Implement Budgeting & Insights Service in Python (spending summaries, goal tracking).
* Integrate Chatbot Service
* Add Admin/Ops Service for system management and backups.
* Perform security and end-to-end testing.
* Prepare presentation materials and documentation.

## 4 Quality Plan

### 4.1 Project Architecture

The system uses aDocker-based microservices architecture.  
 Each service (authentication, accounts, transfers, budgeting, chatbot, and admin) runs in its own container and communicates through an NGINX API Gateway using secure RESTful APIs.  
 Docker ensures environment consistency, simplified deployment, and easier debugging.  
  
 Key architectural quality elements include:

* **Scalability:** Achieved through Docker/Kubernetes orchestration with horizontal scaling of backend services.
* **Security:** TLS encryption, secure cookie handling, and CSRF protection are embedded in every layer.
* **Observability:** A dedicated Observability Service captures logs, traces, and metrics for continuous monitoring and fault detection.
* **Reliability:** Daily automated backups to object storage and transactional integrity enforced by PostgreSQL ensure consistent data recovery.
* **Performance:** Redis caching and asynchronous message queues optimize responsiveness under load.

4.2 Project Measurements   
To ensure measurable quality outcomes, quantitative metrics will be collected across the development lifecycle.

|  |  |  |
| --- | --- | --- |
| Phase | Measurement | Source |
| Development | Code coverage ≥ 85 % | Local testing, GitHub Actions logs |
| Testing | ≥ 95 % test pass rate | PyTest or Selenium reports |
| Deployment | Rollback rate < 3 %; deploy < 10 min | CI/CD pipeline logs |
| Operations | Uptime ≥ 99.9 %; MTTR < 24 h | Docker and monitoring tools |
| Security | 0 critical vulnerabilities | OWASP ZAP and Greenbone Scan |

### 4.3 Risk Management

|  |  |  |
| --- | --- | --- |
| Risk Category | Description | Mitigation Strategy |
| Technical | Integration failure between services or API version mismatches | Version-controlled APIs, contract testing, and integration staging environments |
| Security | Unauthorized data access or credential compromise | Apply least-privilege roles, continuous vulnerability scanning |
| Data Integrity | Data corruption or inconsistency between services | Audit logs, and scheduled consistency checks |
| Performance | Latency spikes under heavy concurrent load | Implement Redis caching, async task queues, and autoscaling policies in Docker/Kubernetes |
| Schedule | Development delays due to overlapping responsibilities | Weekly sprint reviews and milestone tracking |
| Operational | Cloud downtime or deployment failures | Use container health checks, automated rollbacks, and deploy to multiple availability zones |

### 4.4 Verification and Validation Plan

**Validation Activities**

**User Acceptance Testing (UAT)**: Conducted with representative roles (admin, customer, multilingual user).

**End-to-End Testing:** Validates data flow through Docker network topology (frontend ↔ API ↔ database).

**Performance Testing**: Confirms response time ≤ 2 s under expected load.

**Stakeholder Review Demos:** Verify feature alignment with financial management goals and AI chatbot responsiveness.