**CS673 Software Engineering** 

**Team 2 - SoloSavings**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Will Aftring | Team Leader | *William Aftring* | 09/10/2023 |
| Andrew Ouellette | Configuration Leader | *Andrew Ouellette* | 09/10/2023 |
| Whitney To | QA Leader | *Whitney To* | 09/10/2023 |
| Christan Pratt | Requirements Leader | *Christan Pratt* | 09/11/2023 |
| Hanwen Zhang | Security Leader | *Hanwen Zhang* | 09/10/2023 |
| Maryam Hussein | Design and Implementation Leader | *Maryam Hussein* | 9/10/2023 |
| Xin Zhao | Flexible Leader | *XIN ZHAO* | 9/12/2023 |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
|  |  |  |  |
| **1.0** | **Will Aftring** | **09/25/2023** | **Minor** |
| **1.0** | **Andrew Ouellette** | **09/25/2023** | **Config change** |
| **0.5** | **Christan Pratt** | **09/11/2023** | **Add Requirements** |

[Overview](#_g6igqliy7rm)

[Related Work](#_bf21eadgjj29)

[Proposed High level Requirements](#_rgyo4hi9stmq)

[Management Plan](#_ts358bsdtbcv)

[Objectives and Priorities](#_nxeeppkjxgn4)

[Risk Management (need to be updated constantly)](#_tk7yixobah8p)

[Timeline (need to be updated at the end of each iteration)](#_iksrndohvx29)

[Configuration Management Plan](#_j5uvivmxqcsp)

[Tools](#_dzly5b9kz982)

[Deployment Plan if applicable](#_sd8zu6r3jisd)

[Quality Assurance Plan](#_vra5ptwu59qx)

[Metrics](#_vwjduhc9wuah)

[Code Review Process](#_hx3eaiwb8v3m)

[Testing](#_l9xnpmd6hh0y)

[Defect Management](#_5amsh8h9f0c7)

[References](#_pd9euov6m4du)

[Glossary](#_ty3i2nqffhtc)

# Overview

Our project is a budgeting web application. The motivation for this is to be able to provide high level analyses of spending and income trends. This is to allow users who have inconsistent spending or income to be able to better bucketize and denote what the source of a debit or credit to their bank account would be. The primary users of this project would be individuals who are conscious of their spending habits or have inconsistent sources of income (freelancers, consultants, etc…). The basic functionality will be to track credits and debits to the user's account, assign those to a specific group defined by the user, and add relevant comments or notes to that adjustment. These notes and adjustments will be linked to invoice items that can be sent out to clients of the user. This allows for a clear association of work and income.

# Related Work

Similar products:

* You Need A Budget (YNAB): [Gain Total Control of Your Money with YNAB - YNAB](https://www.ynab.com/)
* Mint by Intuit: [Budget Tracker & Planner | Free Online Money Management | Mint (intuit.com)](https://mint.intuit.com/)
* Quickbooks: [QuickBooks®: Official Site | Smart Tools. Better Business. (intuit.com)](https://quickbooks.intuit.com/)

The differentiating factor is the all-in-one management of income and expenditures from the user. Where each of the individual softwares may be able to address part of the picture, our project allows for clear association of invoiced work done and income.

# Proposed High level Requirements

Before the project team can design and implement the budget web application proposed, the team needs to understand what the system is intended to do. “The process of understanding what’s wanted in an application” is called requirement analysis. Following conversations with key stakeholders, and initial market research outlined below are the high-level requirements:

* 1. Functional Requirements

**Essential Features:**

* + 1. Login – Users should be able to login to the system to access their profile and transaction history (8 - 16 person hours)
    2. Track Cash Flow - Users should be able to add incoming and outgoing cash flow so that they can be analyzed in the future. (8 - 16 person hours)
    3. Comments – As a user enters a transaction, they should be able to add comments to have additional context as needed.(4 - 8 person hours)
    4. View Entered Transactions – Users should be able to view the history of transactions entered. (2 - 4 person hours)
    5. User Interface – Users should access the web application so that they can view it in a web browser (8 - 16 person hours)
    6. Budget Goals – Users should be able to enter their financial goals (2 - 4 person hours)

**Desirable Features:**

* + 1. Bank Connectivity – Users should be able to connect to a bank so that they can download transactions to avoid manual entry. (Or have the user be able to add sub-accounts that they can manually input the transactions from their bank accounts.)
    2. Transaction Classification – As the system reviews transactions, it should be able to broadly classify the transaction category
    3. Invoice API – Users should be able to connect to a invoice so that they can retrieve transactions to avoid manual enter

**Optional Features:**

* + 1. Budget Analytics – As the system classifies transactions, it should be able to conduct analytics based on the user’s goal to provide recommendations.
  1. Nonfunctional Requirements
     + **Reliability and Availability**
       1. The Budget app shall have a monthly uptime average of at least 98%.
       2. The Budget app should be available 24/7/365 with downtime limited to only maintenance windows.
     + **Performance**
       1. The Budget app should be able to produce a transaction summary report within 2 seconds with an average transaction count of 200 (baseline)
       2. The budget app should be able to handle a throughput of 1000 transactions per minute (baseline)
     + **Security**

Security is an essential part of the software development process for our SoloSavings project especially when personal finance is involved.

Here are some considerations when develop this project:

* + - 1. Authentication: Login credentials should meet the password requirements to access to a personal account, password needs to be hashed and salted before storage.
      2. Authorization: logged in user can only access and see his/her own data. The user cannot see or access other people’s data.
      3. Data encryption: sensitive personal information such as user credentials(password) should be encrypted to securely store data at rest.
      4. Input validation: to prevent common security vulnerabilities such as SQL injection, XSS, CSRF; we need to ensure code quality, and Github offers code scanning tools which we should take advantage of.
      5. Secure API: CRUD operations will be implemented in SoloSavings, using HTTPS to encrypt communication between client and server.
      6. Third-party security: before implementing any third-party library, make sure the dependencies are reliable and well maintained. Regularly update dependencies are encouraged.
      7. Soft deletion: we are dealing with sensitive data, a soft deletion might be important for backup purposes in case we need to recover data.
      8. Session management: some users might forget to logout their app, as a budgeting app handling sensitive data, we should securely manage user sessions with timeout and token-based authentication.
      9. Secure SDLC: maintain documentation, provide security training, develop an incident response plan, secure continuous integration and, continuous deployment (CI/CD) configuration management
      10. The budget app should ensure defined password lengths and complexity for user credentials. The application should check the password using RegEx\*, eg. minimum eight characters, at least one uppercase letter, one lowercase letter, one number and one special character: "^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$"
      11. The budget app should be secured using certifications to encrypt network traffic.
      12. Users should not be able to see other users’ information or transaction history.
      13. The budget app should encrypt data at rest due to the sensitivity.
      14. The budget app should be PCI\* compliant to allow integrations with banks.
    - **Maintainability**
      1. The average time for the web development team to repair a level 1 defect shall be no greater than 2 hours.
    - **Error Handling**
      1. The budget app should log all errors into the log database.
      2. The budget app should warn the user when an error is encountered during data entry to avoid data integrity issues.
      3. The budget app should allow a maximum of 3 retries when a system or connectivity issue error

# Management Plan

## Objectives and Priorities

(Please describe your project objectives with highest priority first. Project Goals can include but not limited to complete all proposed (essential) features, deploy the software successfully, the software has no known bugs, maintain high quality, etc )

here are the project objectives for our budgeting web application built using Java/Spring boot :

**1. Complete All Proposed Essential Features:**

Ensure that all essential features outlined in the project scope are developed and implemented successfully. These features are fundamental to the core functionality of the budgeting web application.

**2. Run the Software Successfully:**

Achieve a successful running of the budgeting web application without any major issues. This involves proper installation, configuration, and user accessibility.

**3. Ensure High Data Security:**

Prioritize data security by implementing robust encryption mechanisms and access controls to protect sensitive financial information.

**4. Maintain High Software Quality:**

Uphold a high level of software quality by adhering to coding standards, conducting thorough testing, and ensuring that the software performs reliably without frequent errors or crashes.

**5. Achieve User Satisfaction:**

Focus on user satisfaction by creating an intuitive and user-friendly interface.

**6. Comprehensive Testing and Bug-Free Release:**

Conduct comprehensive testing, including unit, integration, functional, and performance testing, to ensure a bug-free release of the budgeting web application.

**7. Scalability and Performance:**

Design the application to be scalable.

**8. Document Design and Architecture:**

Maintain complete and up-to-date documentation of the design and architecture of the budgeting web application. This documentation will facilitate future maintenance and development efforts and ensure that team members have a clear understanding of the application's structure and components.

## Risk Management (need to be updated constantly)

**Ongoing Design Reviews:** Continue to conduct comprehensive design reviews to identify and rectify design-related issues promptly.

**Usability Testing:** Continue usability testing to ensure that the application remains user-friendly and addresses any usability concerns.

**Task Tracking:** Maintain a robust project management and task tracking system to prevent duplication of work and ensure proper task sequencing.

**Security Measures:** Continue to implement security measures, including encryption, access controls, and vulnerability assessments, to protect user data.

**Risk Management Sheet:**

[**Risk Management link**](https://docs.google.com/spreadsheets/u/0/d/1a388ZvM9NL4rqaEczXLPcGGdKg5b4213qVf_CjkZnq0/edit)

## Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration)

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 0 | Create an initial project plan.  Determine the technology stack.  Iteration 0 Kickoff Meeting | 1.Establish version control using GitHub.  2.Share the"Team.md" document with all team members.  3. team meeting to discuss project details.  4.Create a Jira board for monitoring and managing our tasks. 1.Define the project's scope and objectives.  2.Identify key stakeholders and their roles.  3.Establish project goals and success criteria. | 2 hours/ 1 hour  1 hour/ 1 hour  3 hours/ 2.30 hours  1 hour/ ½ hour  2 hours/1 hour  1 hour/30 mins  2 hour/2 hour |
| 1 | 1.Frontend and Backend Design  2.Project Setup  3. User Account Management  4. ​​Transactions Management  5. User Interface Implementation  6. Creating a 7.comprehensive testing plan | 1.Complete the design of both the frontend and backend components of the application.  2.Set up the project using Spring Boot and create the necessary folder structure.  3.Implement user account creation.  4. Implement user account authentication.  5.Create the initial UI for Home page and the dashboard  6.Write JUnit test cases to verify the functionality of the create user account and Transaction | 3 hours/2 hours  2 hours/ 3 hours  1 hour/  2 hours/  1 hour/  1 hour/ 2 hours |
| 2 | 1. Improve frontend outlook, additional components  2. Additional features implementation  3. Update session token implementation  4. Security Enhancement with new features/endpoint added  5. Testing  6. Code review | 1. Implement outlook and connectivity for UI with new features added such as analytical features. Add missing implementation from previous iteration  2. Add more features from backend such as analytical features, goal setting feature, remind features, to be discussed more  3. Implement JWT token session and replace the previous iteration’s implementation.  4. Review security and see if need to add more endpoints protection/permission  5. Add more tests for new functionalities, cover older functionalities that have not been tested. Implement other types of tests such as integration regression tests. Continue with manual testing and provide reports  6. Review other’s code, approve PR and making sure code is working after merge | 6 hours     8 hours     4 hours    2 hours    6 hours        1 hour |
| 3 | 1.Security Enhancements  2.Code Review and Testing  3.Monitoring and Logging:  4.Error Handling  5.User Documentation  6.User Experience  7.User Acceptance Testing  8.Bug Fixing | 1.Review and enhance user authentication and authorization mechanisms. Ensure that only authorized users can access specific parts of the application.  2.Comprehensive Code Review.  3.Monitoring and Logging.  4.Enhance error handling to provide informative error messages and gracefully handle unexpected situations.  5.Create user documentation or guides to help users navigate the application, including registration, login, and transaction processes.  6. Optimize the user interface for a better user experience. Gather team feedback and make necessary improvements to make SoloSavings easy to the user  7. Conduct user acceptance testing with our team as a tester to gather feedback and identify any usability issues or bugs  8. Address and resolve any issues or bugs identified during UAT | 2 hours  2 hours  1 hour  2 hours  2 hours  3 hours  4 hours  5 hours |

# Configuration Management Plan

## Tools

* + - Git
    - Eclipse/IntelliJ
    - Heroku/AWS
    - Java/Spring Boot
    - JSP
    - Jira
    - GitHub Flow
    - Docker
    - GitHub Actions
    - Postman
    - JUnit / Mockito
  1. Code Commit Guideline and Git Branching Strategy
     + We will employ the “GitHub Flow” branching strategy. This is a better alternative to say the “GitFlow” strategy because we would like to keep the master code in a constant deployable state and support CI/CD processes. Team members can create their own branches and commit and push their work once finished. Once pushed a team member can submit a pull request. We will then be able to compare and review all changes to the current master branch. Once reviewed the code will be merged to the master branch and a release will be created. Master branch will be protected and require at least one reviewer before merging into.

## Deployment Plan if applicable

* + - This is currently to be determined, however a possible solution would be to deploy our application via Heroku or AWS.

# Quality Assurance Plan

## Metrics

*The result of these metrics will be reported in the progress report/ iteration summary sheet*

| Metric Name | Description |
| --- | --- |
| KLOC | Measure the size of the code base |
| # files | Count the total # of files in the project |
| # of classes | Quantify the total number of classes used. |
| # of methods | Quantify the total number of methods/functions used. |
| Cyclomatic complexity | Measure the complexity of the codebase, identifying areas that may need refactoring. |
| Defect rate (defects / KLOC) | Calculate the number of defects found per 1,000 lines of code. |
| # of defects | Track the total number of defects reported and resolved. |
| # test cases | Count the total number of test cases in the test suite. |
| Test case pass rate | Calculate the percentage of test cases that pass successfully. |
| Test coverage | Measure the percentage of code covered by test cases. |
| Person hours used | Track the total effort expended in person-hours during the project. |
| # of user stories completed | Quantify the user stories implemented and marked as completed. |

* 1. Coding Standard

Naming Conventions:

* + - Follow consistent and meaningful naming conventions for variables, functions, classes, and files.
    - Use camelCase for variables and functions (e.g., totalIncome, calculateSavings).
    - Use PascalCase for class names (e.g., UserAccount, TransactionHistory).
    - Use lowercase with hyphens for file and folder names in web development (e.g., user-profile.html, css/styles.css).

Code Formatting:

* + - Maintain a consistent code formatting style throughout the project.
    - Use consistent indentation (e.g., tabs or spaces) and specify the number of spaces or tabs.
    - Maintain proper alignment of code blocks and statements.

Error Handling:

* + - Implement consistent error handling practices, including proper use of try-catch blocks or exception handling mechanisms.
    - Log meaningful error messages with relevant context to aid debugging.

Code Reusability:

* + - Promote code reusability by creating modular and reusable functions, classes, and components.
    - Avoid duplicating code when similar functionality is needed in multiple places.

Version Control:

* + - Use a version control system (Git) to track changes to the codebase.
    - Follow branching and merging best practices for collaborative development.

Security Practices:

* + - Adhere to security best practices, including input validation, protection against SQL injection, and cross-site scripting (XSS) prevention.

Testing Standards:

* + - Follow testing conventions and naming patterns to make tests easily identifiable.

Performance Optimization:

* + - Optimize code for performance by following best practices, such as minimizing HTTP requests, using efficient algorithms, and optimizing database queries.

## Code Review Process

Participants:

* + - All team members involved in the development process will participate in code reviews.
    - The code review process will include both the design and implementation leader and another team member reviewing code.
    - Developers will also review each other's code

Pull Requests (PRs):

* + - Code reviews will be conducted using pull requests through Git and GitHub
    - Developers will create a pull request when they have completed a feature or fix.
    - A pull request can be approved and merged into the main branch once it meets the defined criteria and receives approvals from designated reviewers.

Checklists:

* + - Code formatting and style adherence.
    - Compliance with coding standards.
    - Functional correctness and logic.
    - Error handling.
    - Performance considerations.
    - Security best practices.
    - Documentation and comments.
    - Code modularity and reusability.

Review Feedback:

* + - Reviewers will provide constructive feedback on the code changes made in the pull request.
    - Feedback should be clear, specific, and actionable, focusing on areas that need improvement.
    - Reviewers can use code comments within GitHub to annotate specific lines of code.

## Testing

Types of testing:

* + - **Unit Testing**: Developers will write unit tests for individual functions and methods to verify their correctness.
    - **Integration Testing**: The QA leader will focus on integration testing to ensure different components of the application work together seamlessly.
    - **Manual Testing**: Testers will perform manual testing to evaluate the overall functionality and user experience.

Testing tools and framework:

* + - **Unit Testing**: Use testing frameworks like Jest for JavaScript and JUnit for Java.
    - **Integration Testing**: Leverage Postman for API testing.
    - **Manual Testing**: Conduct manual testing using web browsers.

Personnel Responsibility:

* + - **Developers**: Responsible for writing and maintaining unit tests for their code.
    - **QA Leader**: Focus on integration testing and coordinating manual testing efforts, execute manual tests, create test cases, and report defects.

Testing objectives:

* + - **Unit Testing Objectives:** Verify that individual code components (functions, methods, modules) behave correctly and handle edge cases appropriately.
    - **Integration Testing Objectives:** Ensure that different parts of the application integrate seamlessly, data flows correctly between components, and APIs function as expected.
    - **Manual Testing Objectives:** Evaluate the user interface, usability, and overall functionality of the application. Identify any user experience issues and report defects.

## Defect Management

Defect management tool:

* + - We will use issues on Jira as our defect management tool.

Types of defects:

* + - **Bugs**: Functional issues that cause the application to behave incorrectly or not as expected. These can range from critical show-stoppers to minor glitches.
    - **Enhancements**: Requests for new features, improvements, or optimizations in the application's functionality.
    - **Documentation Issues**: Problems related to documentation clarity, accuracy, or completeness.
    - **User Interface (UI) and User Experience (UX) Issues**: Problems related to the look and feel of the application, including usability, accessibility, and design issues.
    - **Performance Issues**: Defects related to the application's speed, responsiveness, or resource consumption.
    - **Security Vulnerabilities:** Identification and management of security-related defects, such as vulnerabilities and data exposure risks.

Defects management:

* + - **Reporting defects**: anyone can report defects and should provide clear and detailed information about the defect, including the steps to reproduce, expected behavior, and actual behavior.
    - **Prioritizing defects**: defects should be prioritized based on the severity and impact.
    - **Assignment and Resolution**: assign defects to responsible team members who will investigate, make changes to fix the issues.
    - **Testing and Verification**: QA leader will verify that the defect has been fixed correctly and confirm the resolution.
    - **Closure and communication**: once defects are resolved, mark issues as closed in Jira and keep everyone informed about the status of defect resolution.

# References

(For more details, please refer to the encounter example in the book or the software version of the documents posted on blackboard. )

# Glossary

* KLOC : Kilobytes of Lines of Code
* XSS: Cross-Site Scripting
* CSRF: Cross-Site Request Forgery
* Regex: Regular Expressions
* PIC Compliance: Payment Card Industry Data Security Standard