**CS673 Software Engineering**

**Team Rocket - Project RCM: Rental Car Management**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Alisa Belousova | Configuration Lead | *Alisa Belousova* | 9/9/2023 |
| Nick Cruz | Team Leader/ Requirements Lead | *Nickolas Cruz* | 9/9/2023 |
| Devon Duddley | QA Lead and Design and Implementation Lead | *Devon Dudley* | 9/9/2023 |
| Kris | Security Lead | *Cangqing (Kris) Wang* | 9/9/2023 |
| Chenghao Ye | Design and Implementation Lead | *Chenghao Ye* | 9/10/2023 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 2 | [Alisa Belousova](mailto:alisa007@bu.edu) | 09/21/2023 | Write code review process section, write defect management section, write deployment plan section, update glossary |
| 3 | [Alisa Belousova](mailto:alisa007@bu.edu) | 09/24/2023 | Write requirements section, update tools section, write code commit guidelines, write metrics section |
| 4 | [Alisa Belousova](mailto:alisa007@bu.edu) | 09/25/2023 | Write testing section, write coding standard section, write timeline section |
| 5 | [Alisa Belousova](mailto:alisa007@bu.edu) | 10/16/2023 | Update requirements, timeline |

[1. Overview](#_g6igqliy7rm)

[2. Related Work](#_bf21eadgjj29)

[3. Proposed High level Requirements](#_rgyo4hi9stmq)

[4. Management Plan](#_ts358bsdtbcv)

[a. Objectives and Priorities](#_nxeeppkjxgn4)

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

[c. Timeline](#_iksrndohvx29)

[5. Configuration Management Plan](#_j5uvivmxqcsp)

[a. Tools](#_dzly5b9kz982)

[b. Code Commit Guideline and Git Branching Strategy](#_nriosngc2qi9)

[c. Deployment Plan](#_sd8zu6r3jisd)

[6. Quality Assurance Plan](#_vra5ptwu59qx)

[a. Metrics](#_vwjduhc9wuah)

[c. Code Review Process](#_hx3eaiwb8v3m)

[d. Testing](#_l9xnpmd6hh0y)

[Unit Testing](#_jaqwcl1nzdgq)

[Integration Testing](#_s114djqqovuh)

[Manual Testing](#_exa9x2sxz4eb)

[e. Defect Management](#_5amsh8h9f0c7)

[7. References](#_pd9euov6m4du)

[8. Glossary](#_ty3i2nqffhtc)

# Overview

(Please give an overview of your project. It should include the motivation, the purpose and the potential users of the proposed software system, the basic functionality of the proposed software system and the possible technology stack to be used. )

The purpose of this application is to help car rental companies manage their companies. This would primarily be done by helping them take care of their inventory (type of cars they own, specific details of the cars they own, etc) and their maintenance schedules.

# Related Work

(Please describe any similar software systems that you have found through the online research, and the differences between your software and those software systems.)

There are various business management tools, such as Proofhub, bit.ai, and timely, but our app is focused on helping car rental companies sort and organize their assets. As car rentals generally need to have a significant amount of resources, it would be helpful to have an app that can stay up to date with those resources, as well as ensuring that things do not get lost due to an extensive catalog.

# Proposed High level Requirements

* 1. Functional Requirements  
     (For each functional requirement, please give a feature title and a brief description using the following format: As (a role), I want to (action), so that (value).)
     1. Essential Features (the core features that you definitely need to finish):

(For each essential features, please give a rough estimation in terms of person hours or an range of person hours)

Registration: 12 hours

As a new user, I want to be able to create a new account by providing my registration information, so that I can access the application's features and personalize my experience.

Login: 12 hours

As a registered user, I want to be able to sign in to my account with my username and password, so that I can access my personalized content and perform actions that require authentication.

Inventory list: 8 hours

As an authenticated user, I want to see the total cars count and list of all cars so that I’m aware of my inventory.

Search car: 8 hours

As a user, I want to allow for quick look up of vehicles based on various criteria such as car id, manufacturer, model, type, so that the user can quickly find what they are looking for.

Create and edit car: 16 hours

As an authenticated user, I want to allow users to add vehicles, remove vehicles, and adjust vehicle information such as: manufacturer, type, year, license, mileage, model, color, seats, condition and oil change, so that the user can be fluid in keeping track of their resources.

Maintenance tracker: 4 hours

As an authenticated user, I want to see dates of any maintenance, past or future, so that the user can ensure their vehicles are taken care of, but also check prior cases as needed.

Desirable Features (the nice features that you really want to have too):

Calendar 12 hours

As an authenticated user, I want to be able to see scheduled maintenance dates on the calendar, so that they may be visualized easily.

* + 1. Optional Features (additional cool features that you want to have if there is time):

Maintenance reminder

As a maintenance worker, I want to notify users of maintenance that is overdue or upcoming.

* 1. Nonfunctional Requirements
* The password has to be crypto hashed and salted in order to maintain integrity
* All pages should have fluid design in order to support various screen sizes

# 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 )

* + 1. Develop the necessary pages needed
       1. Home page, login page, signup page, maintenance page, etc.
    2. Ensure that each page can accomplish its main function
       1. Add page adds new vehicle with unique ID
       2. Edit page edits/removes
       3. Maintenance page displays a car’s maintenance history
       4. Maintenance editor allows editing of a car’s maintenance history
    3. Create the database and ensure that information is stored properly in the database and properly accessible by the application
       1. If information about car x is called then information about car y should not be populated.
    4. Ensure that the username and password system work properly
    5. Ensure that the site is easy to operate and has a clean simple design

## Risk Management (need to be updated constantly)

(Please write a summary paragraph about the main risks your group identified and how you plan to manage these risks. (<https://docs.google.com/spreadsheets/d/1zyuz-gFFX54Zf4VquNnzSNqwHtns-p_QF2r0VEanW_8/edit#gid=0> ) for detailed risk management.

**Risk Management Sheet Link:**

The main risks are people leaving the project, having too large of a scope, and having conflicts within the team .

In terms of people leaving, we have had 2 people leave the project to date, and in those cases work that was supposed to be completed by them was undone and left the team in a scramble to sort through the work and complete it.

In the second case, having too large of a scope could lead to the team struggling to complete the project due to being unable to complete the parts necessary to the project.

The third large risk comes with working within a team. As part of any team, people have differing perspectives, views, and ideas. It will be necessary to have conversations and compromise when necessary to not hinder the development of application.

## Timeline

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 1 | Registration and login | Develop user registration system  Implement form validation  Develop user login system  Implement session management  Conduct security testing  Conduct manual testing for registration and login | Estimated: 50 hours  Actual: 135 hours |
| 2 | Inventory list, create and edit car | Develop UI for inventory listing  Develop backend services for inventory management  Add sorting and filtering options  Implement image upload feature for cars  Conduct manual testing for inventory | Estimated: 40 hours  Actual: 97 hours |
| 3 | Search and etc | Develop search bar component  Develop backend services for search  Conduct manual testing for search functionality | Estimated 40 hours  Actual hours: 135 |

# Configuration Management Plan

## Tools

In this project, a toolset is established to facilitate efficient development, version control, and continuous integration and continuous delivery (CI/CD).

**Git**, as a distributed version control system, empowers us to work on code collaboratively while maintaining a clear history of changes. It enables the team to seamlessly track modifications, synchronize work among team members, and resolve any code conflicts efficiently.

**GitHub**, chosen platform for hosting the project's repository, complements Git perfectly. It offers a centralized hub for the codebase, enhancing collaboration with features like pull requests, issue tracking, and code reviews. These features will streamline communication among team members and provide a structured approach to addressing code changes.

In addition to Git and GitHub, the team leverages popular Integrated Development Environment (IDE) tools such as **Visual Studio Code** and **WebStorm**. The choice depends on personal preference but both of these IDEs are renowned for their versatility, providing features like intelligent code completion, debugging capabilities, and extensive plugin ecosystems.

**GitHub Actions** is an integrated CI/CD solution within the GitHub ecosystem, enabling us to automate various stages of the development process, from building and testing to deployment. By doing so, the team ensures that the codebase remains reliable and deployable throughout the project's lifecycle, ultimately leading to an efficient and dependable software development process.

**TailwindCSS** was chosen for its utility-first approach which allows for more efficient and maintainable code. This approach minimizes the risk of bugs and inconsistencies in the styling of the application. The extensive customization options and responsive design utilities make TailwindCSS a versatile choice for creating a cohesive and adaptive user interface.

**MongoDB** was selected for its flexibility in handling diverse data types and its scalability. The project requires a database that can adapt to changing requirements and handle large volumes of data efficiently. MongoDB’s JSON-like format for data storage facilitates easy data retrieval and integration with other technologies in the stack, making it an ideal choice for ensuring a robust data layer.

**Next.js** (built on React, Node.js, and Express) offers a well-rounded solution for developing scalable and performant web applications. The framework’s features such as Server Side Rendering (SSR) and Static Site Generation (SSG) are essential for optimizing the application’s performance and SEO.

**Jira** is a popular project management tool known for its structured approach to task tracking and its ability to adapt to different project management methodologies like Scrum and Kanban. It was chosen to ensure that every team member is aligned with project goals and deadlines, and to maintain transparency and manage workload efficiently.

Communication is key in any collaborative project. **Discord** was chosen for its real-time communication features, including voice, video, and text channels. It provides a centralized platform for team discussions, helping to address project developments and challenges promptly. Its user-friendly interface and wide range of integrations make it an effective tool for maintaining clear communication throughout the project lifecycle.

With this well-rounded toolset, the team is well-prepared to tackle the challenges of the project and deliver a high-quality product that meets project goals and exceeds expectations.

## Code Commit Guideline and Git Branching Strategy

Our team uses Github flow branching strategy. GitHub flow is a lightweight, branch-based workflow:

* Main Branch: This is the primary branch where the stable version of the project resides. All changes eventually get merged back into this branch.
* Feature Branches: For every new task or feature, a separate branch is created from the main branch. This helps in isolating changes and makes it easier to manage the codebase. The name of the branch should be descriptive.

Before committing code, team member should ensure that commit satisfies following criteria:

* Code Quality: The code adheres to the project’s coding standards and guidelines.
* Functionality: The code achieves its intended purpose without introducing bugs.
* Atomic Commits: Each commit should represent a single logical change. This makes it easier to understand the history and to identify and revert changes if needed.
* Descriptive Commit Messages: Commit messages should clearly describe the changes made to assist other developers in understanding the purpose of the changes.

Pull requests are central to the collaboration aspect of our workflow.

* Creation: Once the feature is completed and tested, a pull request is created to merge the feature branch into the main branch.
* Review & Approval: PRs should be reviewed by other team members. Code review process is described in the “Code review process” section.
* Conflict Resolution: If there have been changes to the main branch since the feature branch was created, they are pulled and merged into the feature branch to resolve any conflicts before the PR is merged.
* Merge: After approval and conflict resolution, the PR is automatically merged into the main branch.

## Deployment Plan

To deploy the RCM system, our team plans to leverage the integration between Vercel and GitHub, optimizing our workflow for efficiency and simplicity. Vercel is a Frontend-as-a-Service product, simplifying the process for engineers to deploy next.js applications.

First, configuration lead will create a project on Vercel and link it to our GitHub repository. During this step, the repository admin authorizes Vercel to access our repository, enabling automatic deployments. Environment variables will be set up securely on Vercel to ensure the application functions correctly.

With Vercel and GitHub integration, every push to the main branch or any feature branches will trigger an automatic deployment. Vercel will build and deploy the application, making the latest version accessible via a unique URL. For every pull request created on GitHub, Vercel will generate a preview deployment. This enables us to review and test the changes in a live environment before merging.

By integrating Vercel with GitHub, we aim to streamline our deployment process, ensure continuous delivery, and maintain the high availability and performance of our application.

# Quality Assurance Plan

## Metrics

(Describe the metrics to be used in the project to measure the quality of your software. Each metric should be measurable and quantifiable. Examples of metrics include product complexity (LOC, # of files, # of classes, # methods, cyclomatic complexity, etc.) , defect rate (# of defect per KLOC), # of test cases, test case pass rate, cost (# of person hours used), # of user stories completed, etc. **The result of these metrics should be reported in the progress report/ iteration summary sheet.**)

| Metric Name | Description |
| --- | --- |
| Code coverage | Code Coverage is a metric that measures the extent to which the source code of a program is executed when a particular test suite runs. We measure it as a percentage, with higher percentages indicating more thorough testing. This metric also helps us to identify which parts of the codebase have been tested and which are at risk due to a lack of testing. |
| Test pass rate | Test Pass Rate is a metric that represents the percentage of test cases that have passed during a testing cycle. It is calculated by dividing the number of passed test cases by the total number of executed test cases. This metric provides insight into the quality and reliability of the software, with higher pass rates generally indicating higher software quality. |
| Defect rate | Defect Rate is a metric that represents the number of defects identified over a specific period during a testing phase. For our project we express it as the number of defects per sprint. |

* 1. Coding Standard

**Formatting and Indentation:** We use prettier, a tool to format files in various languages, to ensure formatting consistency such as: indentation size and style, maximum line length, spacing etc. It’s integrated into our CI/CD to check formatting is adhering to the rules. Prettier can also automatically fix formatting issues on save or whenever a command is run, making it easy for developers to use.

**Naming Conventions**: We have decided to adopt camelCase for variable, function, and method names, while PascalCase should be used for class names and react components. We integrated eslint to CI/CD pipeline to check on naming conventions. It’s also advisable to avoid using abbreviations and acronyms that are not universally understood to ensure clarity.

**Comments and Documentation:** We will be commenting on any functions we add to the program. This will ensure that each function will be mapped out, easy to understand by others, and will not cause confusion.

**Code Structure:** We integrated eslint to CI/CD pipeline to check on the code structure as well. Team should organize import statements with third-party libraries/modules first, followed by project-specific imports.Related functions and variables should be grouped together within files and use modules and directories to organize code by feature or functionality.

**Error Handling:** Vercel will log any errors, both on frontend and backend, providing a consistent approach to error logging and reporting across the application. The job of the developer is to provide meaningful error messages and, where applicable, handle errors gracefully to improve user experience.

**Code Readability:** Developers should create variables, functions, and classes that have meaningful and descriptive variable names and break down complex functions into smaller, single-responsibility functions for improved readability and maintainability.

**Consistency:** We have both prettier and eslint to enforce coding standards consistently across the entire codebase.

**Version Control and Collaboration:** Our team follows GitHub Flow branching strategy for managing development and releases. Developers should write clear and concise commit messages that describe the changes made and the reasoning behind them and conduct code reviews to ensure that the code meets quality standards and to share knowledge among team members.

**Performance and Efficiency:** Each developer will regularly profile and optimize code to address performance bottlenecks. During development and review we will consider the time and space complexity of algorithms and opt for more efficient solutions where applicable, but be mindful of the trade-offs between code readability, maintainability, and performance.

## Code Review Process

(Everyone should review all documents to be submitted. Here you will mainly describe how the code review will be done. Who will review the code, e.g. design or implementation leader will review all code or team members review each other’s code. Do you use pull requests for the code review? Is there a checklist to help review? What feedback should the reviewer provide?)

Our team is committed to conducting code reviews for all documents to be submitted in order to ensure the highest level of code quality and maintainability.

All team members will be assigned to review each other's code. This peer-review system fosters mutual learning and allows for the identification of any potential issues or improvements that may have been initially missed.

To facilitate the review process, the team will employ the use of pull requests (PRs). This method enables the team to examine proposed changes before they are merged into the main branch.

Feedback provided by the reviewers should be constructive, specific, and actionable. It should focus on identifying areas for improvement, offering solutions, and encouraging best practices.

Through this collaborative and structured code review process, we aim to enhance the quality, reliability, and maintainability of our codebase, ultimately contributing to the success of our projects.

## Testing

### Unit Testing

Unit testing is performed continuously using Jest by developers as they write/modify the code. These tests are executed regularly through a CI/CD pipeline. Objectives of unit testing are:

* Verify the correctness of individual units/components.
* Identify and fix bugs at an early stage.
* Refactoring support.

### Integration Testing

Integration Testing using playwright Node.js library to automate browser tasks conducted once the individual tasks are tested and integrated into the software. The QA leader will schedule and oversee this activity. Objectives of integration testing are:

* Ensure seamless interaction between different units/components.
* Identify and fix interface bugs.
* Validate end-to-end system workflow.

### Manual Testing

We use manual testing after automated testing, to catch any issues that might have been missed. All found issues are logged into github issues and addressed.

Regular reports are generated detailing the results of the testing phases. Feedback is looped back to the developers for immediate resolution of identified issues. The QA Leader ensures that all tests are documented, and results are available for review.

## Defect Management

Our team will utilize GitHub Issues for managing defects in our projects. GitHub Issues is a robust tracking tool that integrates seamlessly with project codebase and workflow. This tool will aid us in identifying, assigning, and addressing defects efficiently, providing a collaborative platform for team communication and resolution progress tracking.

The types of defects we will be looking at include, but are not limited to:

* Syntax Defects: These occur due to incorrect code syntax, which can be caught by compilers or interpreters (Sommerville, 2011).
* Logical Defects: Arising from errors in the program's logical structure, these defects can lead to incorrect results (Sommerville, 2011).
* Semantic Defects: These defects arise when the code implemented doesn’t match the intended requirements or specifications (Sommerville, 2011).
* Runtime Defects: Errors that cause unexpected behavior or crashes during the software’s execution fall under this category (Sommerville, 2011).
* Performance Defects: These defects relate to the software’s efficiency, affecting its speed, memory usage, and overall performance (Sommerville, 2011).
* Security Vulnerabilities: These are critical defects that can expose the software to various malicious attacks (Sommerville, 2011).

Once a defect is identified and logged in GitHub Issues, the relevant developer will be notified and assigned to address the issue. The assignment will be based on the nature of the defect and the expertise required for its resolution. For instance, security defects might be assigned to a security lead. The assigned developer will be responsible for investigating the defect, proposing a solution, implementing the fix, and verifying the resolution.

# References

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

1. Sommerville, I. (2011). Software Engineering (9th ed.). Addison-Wesley.

# Glossary

1. RCM: Rental Car Management
2. Repo: Repository
3. PR: Pull Request
4. IDE: Integrated Development Environment
5. CI: Continuous Integration
6. CD: Continuous Deployment