Table 1: Revision History

Date	**Developer(s)**	**Change**
Sept. 26th, 2022	All Members	Initial document creation.

Development Plan
Sayyara Automotive Matcher

Team 27, Kappastone
Tevis Doe, doet
Caitlin Bridel, bridelc
Gilbert Cherrie, cherrieg
Rachel Johnson, johnsr12
Harkeerat Kanwal, kanwalh
Himanshu Aggarwal, aggarwah

The purpose of this document is to outline the course of action for Kappastone's project, as agreed upon by all team members. The document will discuss the team's plan for collaboration, the proof of concept demonstration, project workflow and scheduling.

1 Team Meeting Plan

The team plans to meet weekly on Tuesdays from 5:30 PM to 7:30 PM EST. The meetings will be held virtually on Discord unless otherwise communicated. Meeting minutes will be created prior to the meeting by the meeting arbiter. The meeting arbiter will also conduct the meeting to maintain focus on meeting topics. Meeting topics will primarily include the delegation of tasks, issues and concerns, and questions. Notes will be taken by the meeting scribe and posted to the team's discord in the meeting-notes channel. These notes will summarize what was discussed in the meeting and include action items for each team member and a reminder of the next deliverable due date.

2 Team Communication Plan

The team plans to use a combination of Discord and GitHub issues to communicate throughout this project. Each method of communication will be used for a different purpose as described below.

2.1 Discord

Discord will be used as the teams primary method of communication. In our team's discord server, all team members are able to discuss progress with current

tasks and deliverables, ask and answer questions, send important information, etc. Channels have been set up for general discussion, meeting notes and important links. While working on the project, further channels may be added to organize flow of information. The voice channel in the team's discord server will also be the location of most meetings.

A second discord server has been created by the project supervisor and is accessible by all team members. This server will be used by the project supervisor to relay important information about the project and by team members to update the supervisor on progress and ask any questions.

2.2 GitHub Issues

Github Issues will be used to track progress on tasks and action items for all team members. Team members will also use issues to delegate work to other members, provide feedback on tasks and address bugs or other issues. All team members will be responsible for adding new issues with necessary labels, milestones, comments and attachments.

3 Team Member Roles

Team members were given roles based on personal strengths to make sure that all aspects of the project are effectively managed. The breakdown of the roles and responsibilities is given in Table 2.

Table 2: Breakdown of team member roles

Team Member	**Role(s)**	**Role Description**
Tevis Doe	Git Expert	Responsible for ensuring that the git workflow is understood and followed by all team members. This team member is also the go-to person for questions about git.
	Developer	Responsible for contributing to the creation, testing and reviewing of project code.
Caitlin Bridel	Developer	Responsible for contributing to the creation, testing and reviewing of project code.
Gilbert Cherrie	Developer	Responsible for contributing to the creation, testing and reviewing of project code.
Continued on next page		

Table 2 – continued from previous page

Team Member	Role(s)	Role Description
Rachel Johnson	Documentation Expert	Manages the overleaf project and makes final document approval by ensuring document follows specified checklist. This member will also answer any questions about LaTeX formatting and standards.
	Meeting Scribe	Responsible for making note of any important meeting discussions, action items, and future due dates and posting information in team's Discord.
	Developer	Responsible for contributing to the creation, testing and reviewing of project code.
Harkeerat Kanwal	Meeting Arbiter	Conducts all meetings and keeps team on-track during meeting time. This member is responsible for creating the meeting agenda prior to meetings based on previous week's work and discussions.
	Team Liaison	Acts as point-of-contact for teaching assistants and professor. This member will communicate any given information to the team.
	Developer	Responsible for contributing to the creation, testing and reviewing of project code.
Himanshu Aggarwal	Backend Technology Expert	Provides any help with understanding backend technologies and oversees work done on the backend.
	Developer	Responsible for contributing to the creation, testing and reviewing of project code.

4 Workflow Plan

This section of the document will go over the project's use of git and GitHub to manage code and documentation, and other related parts of the project.

4.1 Development Workflow

This subsection will include a short list on the general workflow a developer will take to contribute to the project. More details will be given in the following subsections.

The workflow is as follows:

1. Identify the addition to be added to the repository.
2. Create an issue with a description of the addition to be added, with the relevant labels and milestones assigned to it.
3. Create a branch that corresponds to the issue created.
4. Develop the addition on the created branch.
5. Commit and push changes to the remote repository.
6. Create a pull request once the addition is complete.
7. Merge the branch into main after approval from at least 1 team member and successful completion of unit testing via continuous integration.
8. Close the issue associated with the branch and delete the branch.

4.2 Issue & Branch Management

The general rule of development on this project is to keep additions as small as possible, which carries over into the creation of both issues and branches. This subsection will detail the general rules of issue creation as well as some tips on how to manage branches.

4.2.1 Rules of Issue Creation

The first step to contributing is to identify the addition to be added. This can be anything, a bug, an addition to the code, improvements to the documentation, etc. Whatever the addition, the first step should always be to create an issue on GitHub.

The only guideline for creating an issue is to keep it as atomic as possible. Issues such as "Update Development Plan" or "Build Back End" should be avoided, as they are either too vague and could grow exponentially from their initial purpose in the case of the former, or are too all encompassing in the first place in the case of the latter. Large issues create large pull requests, which are difficult to review and will slow down development. Instead, smaller, more focused issues should be used. If an addition is too large for a single issue, consider breaking it down into smaller, separate issues that can be completed in a sequence, if not even independently.

When creating an issue, consider also if the changes made will affect other parts of the project. Adding new methods may require updates to documentation, or additional tests to be written. New issues should be made as required to encompass these changes.

There will be no specific template for issue descriptions. A brief description of the addition and an abstract description of how to implement it is suggested to be included.

4.2.2 Issue Nomenclature & Labelling

There is no specific nomenclature of issues, and the naming will most likely differ depending on the addition to be added. Contributors should use their best judgement to name their issues, however generally they should conform to the following examples:

- Bug reports should be named with a short description of the bug. Ex. "MathService incorrectly adds 1 plus 1 to equal 3".
- Features to be added should be a short description of the feature. Ex. "Add new API to fetch shop owners to showOwnerController".
- Issues that update sections of the documentation should be more strict, and include both the documentation name and the section to be updated. Ex. "Development Plan - Workflow Plan Rev 0".

Once an issue is created it should use the following labels provided to categorize itself:

- back end - For improvements related to the server-side of the project.
- front end - For improvements related to the client-side of the project.
- full stack - For improvements that touch upon both the front end and the back end of the project.
- documentation - For improvements that relate to the documentation of the project.
- CI/CD - For improvements that relate to continuous integration in the project.

Labels can be added (or removed) as needed.

4.2.3 Milestones

This project will use GitHub Milestones (as well as GitHub projects, covered in Project Management) to group related tickets together. When an issue is created, it will be assigned a milestone that best fits the issue. Milestones will subsequently include related issues, assigning a due date to them. Milestones do not necessarily need to be linked to a specific deliverable, however they oftentimes will end up related due to the structure of the project.

4.2.4 Branch Nomenclature & Management

Unlike issues, branches have strict nomenclature that relates them to issues. This is because of another rule: branches should be linked to issues, that is, one branch should correspond to one issue. Because of this, the branch name should be directly related to the issue name, with the following nomenclature:

- For bug fixes: fix-⟨issue number⟩-⟨brief description⟩
- For additions to the code: feature-⟨issue number⟩-⟨brief description⟩
- For additions to the documentation: doc-⟨issue number⟩-⟨brief description⟩

Naming branches in this fashion not only allows for easy traceability of a given branch to its related issue, but also allows you to see at a glance what part of a project a branch is updating.

Once an issue has been made, a branch can be created to implement the improvement it details. The branch should only implement what the corresponding issue details; if additions are being added that are not in the scope of the issue, consider updating it to include the new features or creating a new issue altogether.

Commits made to a branch do not need to follow any nomenclature. There is no strict rule on how often work should be committed to git, nor how often to push commits to the repository, it is good practice, however, to commit and push frequently to avoid the potential loss of work.

4.2.5 Pull Requests

Once a branch is ready to be reviewed, and it is pushed to the remote GitHub repository, a pull request can be created. A pull request should include a brief description of the improvement and how the branch implements it, along with a link to the original issue that spawned the branch. Additionally, it should be named similarly to the linked issue.

A minimum of 1 reviewer is required to merge a branch into master, however, ideally two approvals should be given to the pull request before merging. Additionally, continuous integration will be used to run tests on the branch, and are required to pass before merging. If either of these methods of review reveal that more work is needed on an addition for it to be fully functional, consider whether it could be done in a new issue or if it is required before merging the pull request.

After a successful merge, the corresponding issue can be closed if it is resolved, and the branch associated with it can be deleted.

4.3 Project Management

GitHub projects will be used along with milestones to manage issues. Issues ready to be worked on that are due in the next milestone will be placed on the

GitHub project's Kanban board. The columns of the board will be used for the following:

- TODO - Issues that are ready to be assigned to a contributor.
- In Progess - Issues that are actively being worked on.
- Ready for Review - Issues that have an associated branch in pull request.
- Done - Issues that have closed after their associated branches are merged into main.

This allows contributors to keep track of what is being worked on, what still needs to be worked on, and what is ready to be reviewed. By keeping the project board updated, contributors can keep track of the progress of the project as a whole.

5 Proof of Concept Demonstration Plan

For our proof of concept demonstration in mid November we plan to be able to deliver a product with all core functionalities and features built out and ready to demonstrate. The core features of the product for shop owners are being able to filter, search and update work orders, view and respond to quote requests from vehicle owners and being able to manage appointments with vehicle owners. The core features of the product for vehicle owners are being able to search for a service provider, submit quote requests and book appointments with service providers. These are the features of the product that our group along with our project supervisor have decided to be core features and our plan for the proof of concept demonstration is to be able to have a working user interface demo of our product that will be able to demonstrate all these core features of the product from a front-end point of view. We don't plan on having all the back-end functionality complete for the proof of concept demonstration but we plan on having a very rudimentary front-end product that may not be finished stylistically but will allow us to at least demonstrate all the user features. As well as these main features we plan on ensuring that our proof of concept demonstration is all built using a common dashboard and code base rather than having a separate demo built for shop owners and vehicle owners. We plan on being able to demonstrate our proof of concept working on both mobile and computer devices since the finished product is meant to be multi-platform.

Various risks that exist for this proof of concept demonstration plan are promising too many features, having the same dashboard operational for two different user types, and creating a web app that is capable of working on multiple platforms. The first risk is with over promising features in the finished product that won't allow us to be able to complete the core functionality and having a working user work flow. To overcome this risk we will ensure our proof of concept demonstration has all core features included so that we can demonstrate a complete user workflow and ensure our application is functional.

The second risk is having a common dashboard and code base that works for both user types, shop owners or technicians and vehicle owners or customers. To overcome this risk we will demonstrate our proof of concept using a common dashboard that will be operational and contain relevant features based on the two user types rather than building two separate proof of concepts for each. The final risk is ensuring that our application is multi-platform and works on both mobile devices and computers. To overcome this risk we will be conducting our proof of concept demonstration plan on both mobile and computer devices to ensure the application is functional on both platform types.

6 Technology

This section of the document will cover the technology to be used in the project as a whole.

6.1 Programming Languages & Testing Frameworks

The output of this project will be a progressive web application (PWA), and so various different programming languages and testing frameworks are required. The following sections will cover programming languages for both the back end and the front end of the project.

6.1.1 Back End

For the back end it was decided to use Python as the base programming language, using Django as the framework. This combination was chosen based on both on comfort of the developers and ease of use. Django is a powerful framework that has extensive documentation on the internet to make development easier, and has many built in features that will be used. Django has built in support for Pylint, our chosen linter, through the Pylint-Django library, and also has built in documentation generation. This will be used to keep the code clean and readable to both contributors and outside observers. The most helpful part of Django that will be used in this project is its built in libraries, however. To successfully complete this project we will need to have different levels of user authentication to serve different data to different users, which Django supports from the beginning with its built-in libraries to handle user authentication and permissions.

To test the back end, pytest will be used. Unit tests will be written using the pytest libraries, and run later using continuous integration. pytest was chosen, again, because of the comfort of the contributors having used it in the past, but it is also a robust framework that has direct integration with Django through the pytest-django library, streamlining the unit testing of our back end. Additionally, pytest has built-in code coverage analysis, which will support the creation of tests that cover more branches of code.

6.1.2 Front End

The front end will use Next.js, which is a framework built on React, along with TypeScript as the programming language. Next.js is a powerful framework with many built-in features that the project will use, including native support for TypeScript which some other frameworks lack. In addition to base Next.js, the project will also be using the plugin next-pwa to easily produce the output required from the project, a PWA that can be installed on mobile devices. Additionally, Next.js has built-in support for ESlint, the linter for the front end, allowing the project to more easily adhere to coding standards without having to import additional libraries. Next.js does provide a built-in back end server, however it was decided to not be robust enough to be used on its own, which is why Django will be used as well.

As the testing frameworks for the front end, Cypress and Jest will be used. Both support many different types of front end testing, but we will be using Jest primarily for unit testing, and Cypress for end to end testing, which are their respective specialties. Additionally, like pytest in the back end, Cypress also provides built-in code coverage analysis, allowing the contributors to write tests that cover more branches of the code.

6.2 Continuous Integration

This project will utilize continuous integration tools to manage branches being merged into the main branch. Using GitHub actions, both back end tests using pytest and front end tests using Cypress will, ideally, be required to pass before a branch is able to be merged into develop. Tests will not be run on the main branch as the team does not have the ability to continuously run tests using GitHub. It is hoped that by running tests on branches before they are merged that the main branch will not need tests to be run on it.

If possible in the team's capacity, GitHub actions will also run branches in pull request through the linters for the front end and back end, ESlint and pylint respectively, as needed.

6.3 Performance

No specific tool will be used to measure performance of the web app. The team will likely use various different online performance tools, such as GTmetrix or web.dev. In addition to these online tools, end to end tests will be used to run through different interactions users have with the web app, which will reveal areas of the app where performance suffers. By using a combination of these tools along with manual testing, the performance of the web app will be accurately measured.

6.4 Tools

The main tools used to build the web app are the hosting services Heroku and Vercel. Heroku will be used to deploy the back end, and it has built in

integration with PostgreSQL, allowing us to easily host a database as well. Vercel will be used to host our front end, as it has some great integration with Next.js. But most of all, both services have free packages, which was the main reason they were chosen.

7 Coding Standard

Since various coding languages are being used in this project, this will require different coding standards that we as a group have agreed to adhere to. For our front-end Typescript code we will be using ESLint to ensure our code is linted and adheres to the ESLint standards. ESLint is a very common Javascript and Typescript linter and by using this it ensures our Typescript code adheres to modern day standards. For our back-end code we will be using Python and the Python code standard we will be adhering to is the Pep8 standard which will be accomplished by using the Pylint python linter. For code formatting we will also be using the Prettier plugin to ensure a common formatting for all of our code. By using these coding standards, linters and formatting tools it will ensure a common pattern across our whole code base which will make our code base easier to maintain.

8 Project Scheduling

Most project scheduling will be done on GitHub, while minor, informal discussions about milestone timelines and task delegation may occur on Discord. Specifically, our major milestones and their deadlines will be recorded using GitHub Milestone, as elaborated on in section 4.2.3, and issue progress will be tracked using GitHub Issues.

When faced with a large task, the task will be broken into smaller parts and an individual issue will be created for each. When it is time to assign tasks, the distribution of work will start on a volunteer basis. If there remains a task that must be assigned but no one has volunteered for yet, it will be assigned to the person with the lightest workload for that milestone. This process will be repeated until all tasks for the given milestone have been assigned.

Here is an overview of the project's major milestones and their deadlines:

Problem Statement, Development Plan	September 26
Requirements Document Revision 0	October 5
Hazard Analysis 0	October 19
V&V Plan Revision 0	November 2
Proof of Concept Demonstration	November 14-25
Design Document Revision 0	January 18
Revision 0 Demonstration	February 6-February 17
V&V Report Revision 0	March 8
Final Demonstration (Revision 1)	March 20-March 31
EXPO Demonstration	April TBD
Final Documentation (Revision 1) - Problem Statement - Development Plan - Requirements Document - Hazard Analysis - Design Document - V&V Plan - V&V Report - User's Guide - Source Code	April 5