Management Report

Roles and Responsibilities

Project Roles

The assigned member is responsible for leading the team in their allocated area. This includes overseeing ticket creation and merge requests, completing more tickets in their area than other members, and conducting peer review sessions to assist other members where needed.

Project area	Assignee
Scraper	Callum
API server	Jonathan
Database management	Callum
Frontend design	Abdullah
Frontend implementation	Jonathan
Deployment	Tung
Testing	Negin
Reports	Different sections allocated to all members

Agile Roles

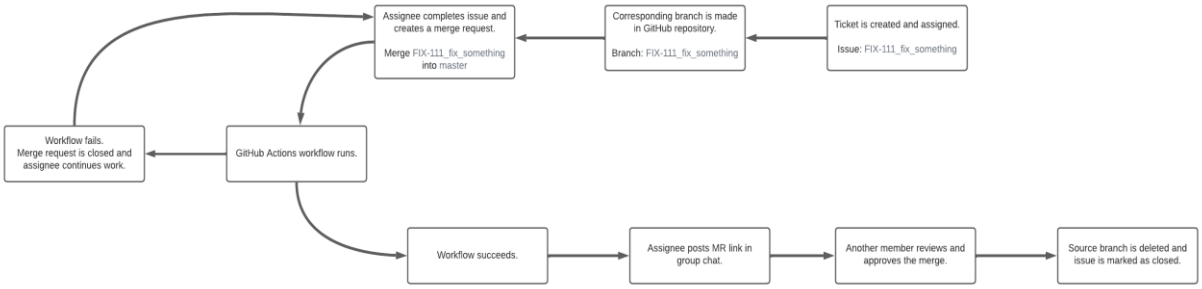
The team will implement Agile team roles to help deliver a strong leadership structure where work is properly prioritised, meetings are fruitful and team members receive proper guidance to complete the vision of the project owner. The roles have been assigned as such:

Role	Description	Assignee
Project Owner	• Prioritises work • Manages scrum backlog • Interacts with stakeholders • Define objectives	Jonathan

Scrum Master	• Interacts with project owner and team members • Provide direction to team members • Facilitate stand-up meetings • Remove external blockers	Callum
Team Member	• Collaborators	Negin, Tung, Abdullah

Git Practices

Overview



Trunk-based development

Trunk-based and git flow are currently the two most popular version control management practices.

We have decided to adopt trunk-based development for managing our GitHub repository as it supports fast-paced development of an MPV. Trunk-based development also helps us achieve CI/CD through regular commits to the master branch and shorter deployment intervals.

Git flow was not selected as it is more suitable for enterprise settings, open-source projects, or extending a product that is already well-established. Git flow's use of multiple primary branches and restricted access to the main branch introduces additional complexity that is not necessary for our project at this stage of development.

We will use the master branch as our 'trunk', with short-lived issue branches. Everyone in the team will have full access to feature branches and the master branch, easing collaboration and allowing for small fixes to be pushed directly to master.

We will set up a CI/CD pipeline which will run tests for every new commit to master. A commit will be automatically denied if the pipeline fails, ensuring our master branch is stable at all times.

Once the pipeline for a merge request passes, the assignee is free to merge the branch. If the assignee would like their code to be peer reviewed prior to a merge, they can post the link to their merge request in the group chat. Once a merge is approved, the source branch is deleted to simplify the repository structure.

Finally, we aim to deploy our master branch fortnightly to avoid deployment complications towards the end of the project.

Naming conventions

Issues

We will distribute and manage work among members using issues posted on our Jira board.

We identify each issue category with a tag. The following is a draft of main categories and their corresponding tags.

Component	Tag
Backend code	BCK
Frontend code	FNT
Test	TST
Patch	FIX
Deployment task	DEP

Each individual issue is uniquely identified using its category tag and a number. The issue name follows the format TAG-NUM_brief_issue_description.

Branches

A new branch will be made for each issue. Each issue should be small and result in a short-lived branch which can be merged into master quickly, preferably within one day. The branch name will be the same as the issue name to make the purpose of the branch clear.

Commits

Each commit comment will be prefixed with the issue identifier followed by a concise subject line - TAG-NUM: brief comment on change. The subject line should be fewer than 50 characters, use the imperative mood, and end without punctuation. If additional information is required, a body message not exceeding 80 characters can be added.

To aid in the creation of consistent commit messages, our team will use a shared .gitmessage file which we will configure as our global commit template. We aim to write meaningful commit messages which succinctly explain the purpose of the changes made.

Automated testing pipeline

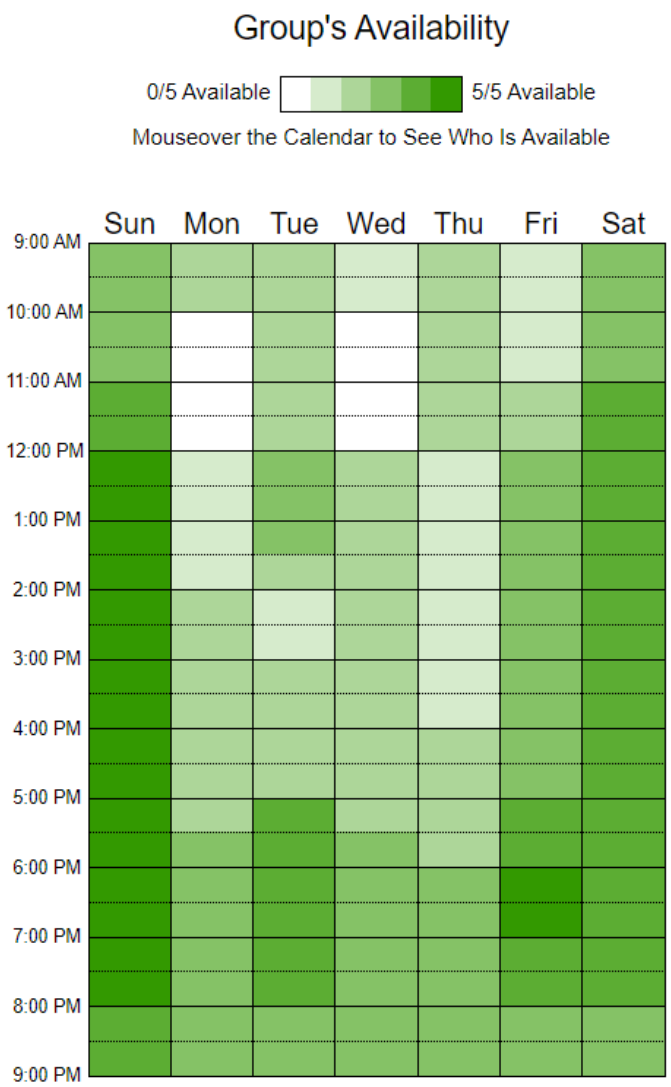
We will use GitHub Actions to set up a workflow which is triggered on merge requests and commits to the master branch. This will run the existing test suite on the code and prevent the

code from being committed if any of the tests fail. This helps maintain a stable master branch and prevent regressions as we add new features.

Team Communication Plan

Team Meetings

Our team has used several tools to implement strong team planning and communication. We started by finding team members' spare time using when2meet.com. In the situation that some members can't afford to attend the meeting, we will tell them the important information that we discuss in the meeting for them to catch up, or if they have any questions then we can discuss it on our team messenger chat.



Since the beginning of the project, we basically don't have any specific meeting time but we usually have three meetings in a week:

- The first meeting often occurs on the weekend before the meeting with the mentor in order to gather information from every team member, clarify the project specifications and make a board of questions of ambiguous details to ask the mentor.
- Second is the meeting with the mentor on Tuesday at 6:30. We can meet earlier than this particular time as we can come up with more questions that we are confused about through our project before meeting the mentor.
- The third meeting that randomly takes place in the week depends on the urgent circumstances as we have to separate our work equally and decide to assign it to each team member. Besides, we have to check if everybody has done their task or has difficulty in doing it.

Hence, our meeting plan will change a lot as this is only the start of the project since we haven't done any implementation, as we might meet more often.

Team Notes

To summarize our meeting plan and notes, We will use the meeting minutes templates provided by Jira and Confluence. Jira and Confluence allows us to create a workspace where we can initiate the plan along with the tasks, store all the meeting minutes and notes in order to keep the project on track. Jira and Confluence are mainly used because we will store all our reports, project plan, documentations and most importantly the Git Hub repository.

Team Communication

We are using three primary tools to communicate:

- One is the messenger chat, mainly used for chatting and notifying every member the meeting time because everybody has a phone and you can easily see the messenger notifications. Besides, we also discuss some important topics that we have to deal with in the meeting.
- The second one is the Microsoft Teams, this is our original group created by our mentor, where we usually interact with him for instance: having a meeting call with our mentor, asking questions and getting the project specifications with answers to our questions.
- The last one is discord, our main server where the team voice chat takes place. This is convenient since we can pop in the room in a single click, we can also create lots of text channels based on different type (resource, questions, brainstorming and so on) in order to share and store information (Links, documentation, samples, etc) and when we come back to find it, you can easily accessed them.

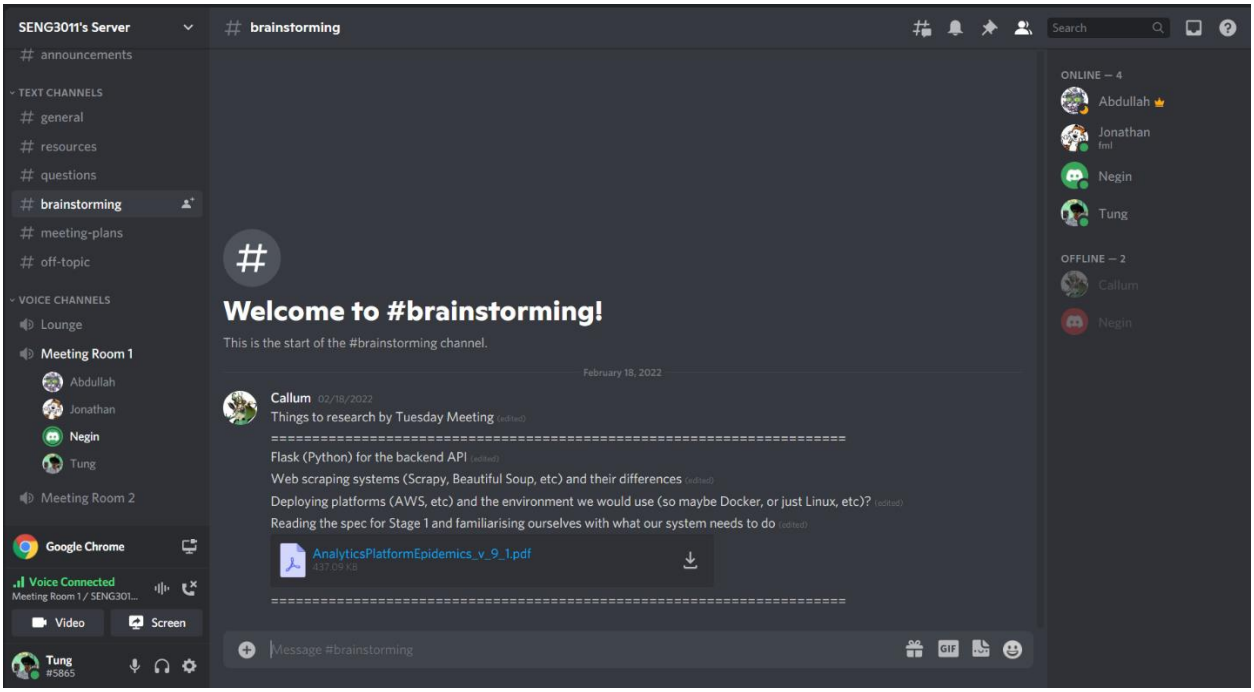


Illustration of our discord server