Project Plan

Introduction

As we are lacking crucial information about the nature of the upcoming project, many ideas may well be drastically changed, or dropped entirely as the project progresses. We intend to form the skeleton of our project using the following ideas.

Project Management

Meeting Schedule & Communication

Meeting Schedule

During our first team meeting, we unanimously agreed upon a fluid meeting schedule. It was agreed that the final topic of discussion in all meetings is to formally organise the time and date of the next meeting. We believe this flexibility will result in more efficient and effective meetings. The regularity of meetings can easily be increased or decreased depending on workload and necessity. Furthermore, this method will allow us to best manoeuvre each individual's ever-changing availability over the coming weeks and months.
The purpose of our regular, in person, meetings is to discuss any issues, ideas and concepts with all members of the team, as well as to provide updates on each individual’s progress. Many important aspects of software engineering and group project work can be difficult to coherently discuss via group messaging, especially as the project grows more complex.

General Communication

We have agreed upon a number of general methods of communication throughout this project.

- Our main means of group communication will be via the application Slack. This was agreed due to its ease of use and a wide range of features. Crucially, group messaging and file sharing is very straightforward.
- A google drive directory has been set up for all team members to publish, share and review any project relevant documentation.
- Additionally, all team members have contact details for each other team member for any necessary one-to-one contact.

Mark Allocation & Selecting Team Leader

Team Governance

Through a democratic process, Greg has assumed the responsibilities of team leader and James of project manager. The team agreed that they would be the best suited for these roles after taking into consideration their experience, skills and demeanour.

Marks Allocation

With regards to mark allocation, the team agreed that at the end of the project we would all ideally agree that all members put in roughly equal work and therefore distribute the marks evenly. However, if it turns out that there were major discrepancies in the amount of work put in from various team members then we would all individually assign a mark for all other members of the team by rating the other team members from 1-5. This will be based on how much they contributed to the success of the project, then the results would be averaged and final individual marks derived. There would be some range decided by the group (e.g. the highest mark could be 120% of the average and the lowest 80%).

Kanban & Job Allocation

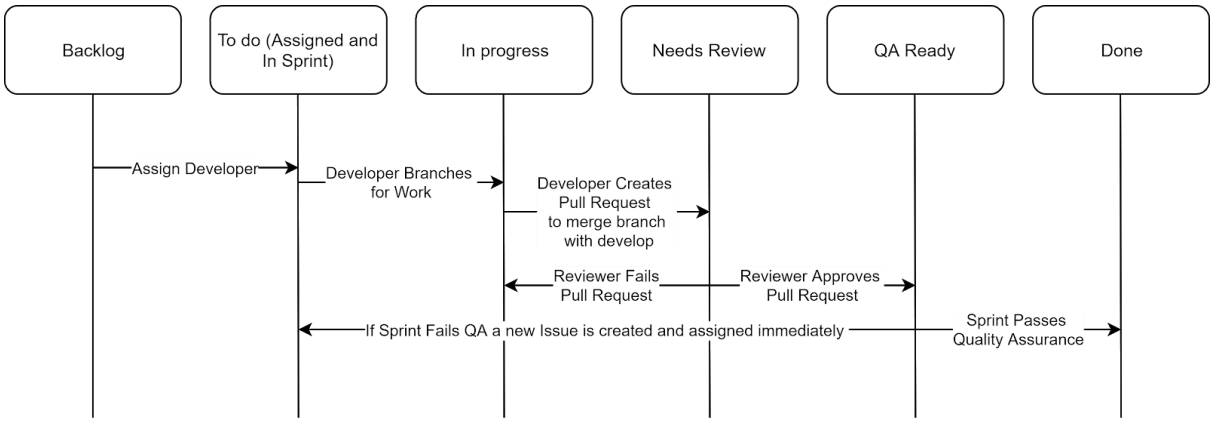
Kanban

We plan on using the Kanban board available in the projects section of a Github Repository. This will allow us to manage the different features and issues which need to be implemented and code reviewed. We will have the following 6 Columns:

- Backlog
- To do (Assigned and In Sprint)
- In Progress
- Needs Review
- QA Ready
- Done

When a Github issue is added to the project then it will automatically be added to the “To do” list. Once a developer has been assigned then it will need to be moved to “Assigned and In

Sprint” column. Once the developer branches from development this allows for the issue to be moved in to “In Progress”. After development has finished the developer will create a pull request to merge the branch with the development branch this will automatically move the task into needs review. At this point the feature is reviewed, and the issue is either approved or pushed back to in progress. Once the issue is approved the code is merged into the development branch and the issue is moved to QA ready. Once all the issues in the sprint are completed the development branch is put through a QA cycle. If there are any new issues they are added to the current sprint and assigned a developer immediately. If the sprint passes QA then all the issues are set to Done. Below is a diagram of the planned flow:



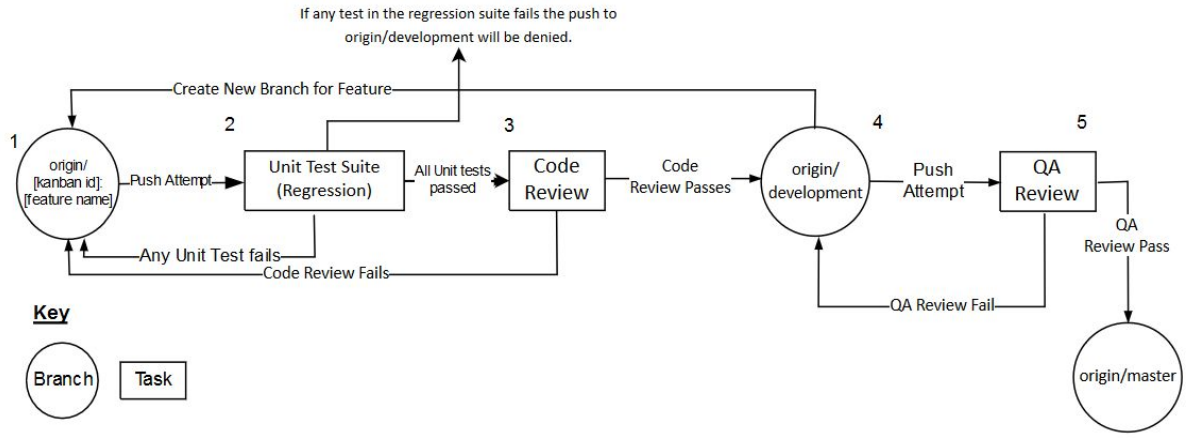
Job Allocation

Issues will be assigned time estimates by everyone in the group this is to ensure that they are as accurate as possible. Issues will then be allocated by James our project manager. This ensures that people with the correct skills or equipment will be able to develop the issue at hand and that work should be balanced between everyone. Code reviews will be assigned by the developer when they submit a pull request. It will be left to the developer to identify the most suitable member of the group to review their code.

Technical Plan

Branching Strategy

Our branching strategy is demonstrated in the diagram below.



Stage 1 - In order to add new features, each developer will branch off origin/development, our naming strategy for these branches will be [developer initials]-[github issue]-[A descriptive feature name (chosen by the developer)].

Stage 2 - Once the developer believes their feature is ready they will write unit tests validating it, this will be added to our regression suite. In order for a developer's feature to be merged into the development branch, their code must first pass all tests residing in this regression suite (including their own). If any of the tests fail the push will be rejected (revert to *Stage 1*).

Stage 3 - After the code passes all tests it must be reviewed. We've adopted a variation of Google's approach in which at least one other developer must review the code, if the reviewer regards the code to be of an adequate standard and cannot identify any defects, the code will enter the development branch. Otherwise, the process will revert to *Stage 1***.**

Stage 4 - The development branch can be regarded as our beta release, as although we've taken measures to ensure the code quality is of a good standard and attempted to weed out any defects, it is unreasonably optimistic to assume all defects will be identified.

Stage 5 - At the end of each sprint, when the development branch is full of unit tested and code reviewed features we will at the point perform a QA review before merging it into master. We will, either manually or if possible with automated tools, run through the app's functionality ensuring all features are working as expected. If we find everything to be satisfactory, the origin/development branch is merged into the origin/master branch and the process will begin again with new features (start again at *Stage 1*). Otherwise, the code will remain in the origin/development branch (revert to *Stage 4*) until sufficient improvements have been made to allow for the passing of the next QA review.

Technologies

IOS applications require Xcode and it is only available on Mac OS. Unfortunately, none of us have a Mac. Therefore the obvious conclusion was to go cross-platform, to have an application on both IOS and Android, as most of us prefer to work at home than at the university computers.

Some cross-platform frameworks we could use are;

- React Native (Written in JavaScript) (Native)
- Apache Cordova (Written in JavaScript) (Hybrid)
- Adobe PhoneGap (Written in JavaScript) (Hybrid)
- Ionic Framework (Written in JavaScript, TypeScript) (Hybrid)
- Apache Weex (Written in JavaScript) (Native)
- iFactr (Written in C#)
- Kivy (Written in Python)

All are open-source frameworks we can use for our development. We have made the decision to run React Native on our front-end, so that our application can run on IOS and Android. The rest of the list we will leave as a reference, in case we decide to change framework. If a back-end is an appropriate approach for the project, we will use Java since everyone is familiar with it.

Web Frameworks for Java Back-end

Assuming using a backend is a suitable approach for the project there would need to be means of communicating between the React front-end and the Java backend. We could manually implement this communication, however, many frameworks already exist making this process both faster and more maintainable.

Some possibilities include:

- Spring framework
- Spark
- JSF
- GWT

However the latter two possibilities are “strictly focussed on building web interfaces”, this is not the goal of our backend as all interfaces will be handled by React.js.

Despite some members of the team being familiar with the Spring framework, in order to use it effectively many concepts would need to be understood (such as autowiring, beans, DAO/Service/Controller structure, etc), it would end up entirely shaping the back-end architecture. Spark seems like a much more lightweight framework hopefully meaning it has less of an architectural impact, leaving us with greater flexibility as the project progresses.

Http Testing Framework

If we need to create a backend for the project we could use a tool to allow us to create some automatic documentation and a testing framework for a backend. This would be useful as it will allow for not only us, but other users to access and test our API. This should make it easier to debug problems and therefore will more likely speed up development. Below is a list of a few different tools which we could use to generate a testing framework.

- Slate
- Swagger
- RAML

Swagger can easily be included in the build process. Slate can build docs automatically, but doesn't have any links to the build process. Thus, it would need to be called as a command line program as a part of the build process, to be used automatically. RAML needs to be written by hand. Therefore we will use swagger as it provides the cleanest solution.

Code Quality

Code Review and Bug Finding

Members may also find problems with code written by other members of the team. If this occurs during a code review then it should be documented in the review and the feature should be set to review fail. Otherwise, an issue should be raised on GitHub, noting the problem. Whilst raising the issue, a solution may be proposed. The original author is responsible for fixing the problems, as they should be the most knowledgeable on the behaviour of that section of code, making fixing the problem easier. It may also be the case that the perceived problem was intended behaviour, in which case the author should attempt to make this behaviour better known by documenting it. Though the original author is responsible for fixing the problem, they may instead manually check and authorise a proposed solution; if one has been provided.

Code Standards

As a team we have agreed to use a tool which will automatically standardise the formatting and style of the code. This approach enables automated formatting to ensure code style consistency. Additionally, individual team members can write the code in the style they prefer and apply the standardised formatting. The team also agreed to not use the braceless-ifs style in Java and instead stick to the more standard approach.

Automation

All members of the team will be writing code, because of this, guidelines for testing will be specified here to reduce workload, improve workflow and promote professional practice. All members of the group will be expected to write unit tests for all features and also any complex methods they write. These tests should be completed to a reasonable standard, verifying that the methods respond correctly to both expected and unexpected arguments.

All the tests will be combined into one testing suite that will be tested against the whole system. This will be our regression suite. This test suite will be used to test each group member's work before they merge their feature branch into the development branch. We will also investigate the applicability of using WebdriverIO to develop an automated UI test suite for the JavaScript code.

To ensure consistency of style throughout the project we will use tools to automatically format the code before it is committed. We will do this by using a git hook to pass any modified files containing source code, to a code formatter, then re-staging and committing the newly formatted files. We intend to use the Clang-Format tool to format the code to the Google standard for both the Java and JavaScript code bases. We are using this tool instead of other formatting tools, as it supports both Java and JavaScript files, and has git integration. This is appropriate for our project as we intend to use both React and Java. Using a single formatter with the same rules will lead to a more consistent style.