Development Plan

Team Meeting Plan

Pre-Development

1. Meet every lab-session to work on the assignments & milestones.
2. Lead Developer to provide progress reports/summaries at the end of each lab.
3. Lead Developer to provide tasks/a plan-of-action for each item the team has to accomplish by the next lab session.
4. Meeting Agenda will have the following information to be filled in during each meeting.
   (a) Previous Agenda items.
      i. Will go through Agenda items from last meeting to see what work is in progress and how much progress has been made.
   (b) List of topics for current week.
      i. Will have Decision section for each topic.
   (c) Work plan for each developer for the week (Homework).

Development

1. Hold stand-up meetings every 2 days discussing our progress
2. Lead Developer to provide daily progress reports/summaries
3. Meet during lab sessions to discuss blocking issues and debug
4. Meeting Agenda will have the following information to be filled in during each meeting.
   (a) Previous Agenda items.
      i. Will go through Agenda items from last meeting to see what work is in progress and how much progress has been made.
   (b) List of topics for current week.
      i. Will have Decision section for each topic.
   (c) Work plan for each developer for the week (Homework).

Team Communication Plan

Pre-Development

1. Communication during this period is to be done through a combination of Microsoft Teams and Facebook Messenger.

2. Teams is to be used as a way to provide weekly updates on progress, upcoming tasks, questions and concerns, and as well as possible issues.

3. Messenger will be used as a way to communicate on a daily basis in terms of providing a way to answer simple questions and track progress on tasks when closer to deadline.

Development

1. All communication shall be done over Teams during Development

2. Weekly updates shall be provided by the Lead Developer to the Team via Microsoft Teams. These updates shall encompass the following (progress summary, upcoming tasks, possible challenges). If there are any issues or any unpredicted set-backs, the update shall also include the following: brief plan of attack, revised plans, updated time-line.

3. Milestones shall be tracked via the Kanban Board on GitLab.

Team Member Roles

Name	Role	Expertise
Aditya Sharma	Developer, QA	React, Node, Python, TS/JS, Git, Latex
Jash Mehta	Lead Developer, QA	React, Node, Python, TS/JS, Git, Latex
Zackary Ren	Developer, QA	React, Node, Python, TS/JS, Git, Latex

- The Lead Developer will handle daily/weekly reports and keep track of progress using the meeting agenda and Gantt chart. Developer responsibilities are encompassed as well.

- The Developer is responsible for creating the features assigned to them and resolving the bugs involved with their feature. Features are assigned based on the SRS created by the team.

- The QAs' (Quality Assurance) is responsible for creating test cases relevant to the feature they are assigned to. This is to ensure that the feature adheres to the correct specification.

Git Workflow Plan

Our Git Workflow shall revolve around the simplified version of the Gitflow Workflow practice. Throughout the development process of the project, we will be leveraging 2 different branches: *master* and *development.* The *master-branch* will store the official release of the project, whereas the *develop-branch* will be the integration branch for all the features/tickets.

We will also be leveraging the concept of a *feature-branch.* Each feature/ticket shall be implemented within a *feature-branch* whose respective parent is the *develop-branch.* This ensures that when the development and testing process for a feature is complete, it can be merged with the *develop-branch* rather than the *master-branch.*

Our project will not be leveraging *release* or *hotfix* branches due to the intended small scale and simplicity of the project. We feel that by adding these branches, we would over-complicate the production, testing and release of our software.

Proof of Concept Demonstration Plan

The most significant risk would be a lack of consistency amongst the developers' work environments. Inconsistencies in software library versions could introduce unnecessarily complicated bugs. We will ensure that all developers' environments are set up the same way to avoid the aforementioned issue.

We believe that unit testing will not be difficult because we have significant experience with unit testing via 2AA4 and 2XB3. However, we believe that integration testing presents certain challenges. This is because we have to mock the API and front-end interactions to simulate different kinds of interactions the end user would have with our app.

The most common roadblock in team-oriented software development is setting up the development environment and tools. The development environment and tools that are being used across everybody's machine needs to be consistent otherwise there is a risk of producing bugs later in the development period that are difficult to resolve. The second part of implementation that might be difficult will be deploying the front-end and back-end. This is because we will be using technologies that we have little hand-on experience with. However, we hope to overcome this burden by focusing all of our attention on 1 task at a time (backend first, then frontend). This ensures that everybody is aware of what is happening in the software and in the event that a bug rises, multiple developers are capable of contributing to its resolution rather than just the developer who made the feature.

There are no libraries that are particularly difficult to install. However, this does not mean that the libraries will be easy to use because we do not have experience with some of them (Flask). By the same token, we plan on leveraging existing documentation and publicly available resources (udemy, stackoverflow and github issues) to ensure that we understand the library and overcome any issues that arise from its use within an appropriate timeframe.

Portability will always be a concern with any software product. However, we plan on ensuring that our software product is portable via the choice of the technologies that we are using. By using React-Native, we are able to cater to customers in both iOS and Android markets. Furthermore, by leveraging Docker, we are able to containerize the application and ensure that it can be run from any environment for development. This significantly improves portability of the software product after release and during development as well.

Technology

- *React-Native*
- *Python - Beautiful-Soup, Selenium & Flask*
- *Prettier - ESLint, Bracket Colorizer*
- *Docker*

As mentioned in the Project Statement, our software product is a mobile application intended for both iOS and Android devices. As a result, we need to leverage a technology that will enable us to produce and deliver a high quality product for both mobile environments.

Normally, in order to produce a product for an iOS and Android device, we would need to leverage *Swift* and *Java* respectively. This would result in either 2 separate code-bases or 1 massive code-base. This would introduce an extra-layer of complexity to the project. Therefore, we have decided to use a cross-platform technology known as *React-Native*. This programming language/framework will allow us to develop for both platforms through a singular codebase.

We will also be leveraging *Beautiful-Soup* & *Selenium* to perform the web-scraping. The backend logic will be hosted by an API and this API will be created via *Flask*.

A CI/CD pipeline will be leveraged to automate builds, testing and deployments. We will be leveraging the built-in CI/CD pipelining support offered by GitLab. This requires the use of a Docker image which is why we will be using Docker.

Coding Style

A strict coding style shall be enforced for the purpose of readability, maintainability and quick iterations. This coding style shall be enforced via the use of tools such as *Prettier*, *ESLint* & *Flake8*. These tools are plugins that can be installed onto our IDE (*VS Code*). These tools will handle things such as spacing, variable naming, maximum-line length etc…

To keep the project simple, we will not be using a state-management system such as *MobX* or *Redux*. Instead, we will be leveraging the built-in state-management tools offered by *React-Native*.

General Rules

1. Keep things simple.
2. Find the root cause of bugs.
3. Put all boundary conditions in 1 place (don't have different edge cases for multiple functions). Handle edge case processing in 1 place.
4. Descriptive, meaningful, unambiguous names.
5. No constants (all constants will be declared in a file in the root directory known as constants.extension).
6. Keep functions small - each function should only do 1 task.
7. Minimize the number of arguments to a function.
8. Add Brief comments on a function if the function name is not sufficient enough to describe it.
9. When testing, only have 1 assert per test and provide each test a meaningful description.

React-Native Rules

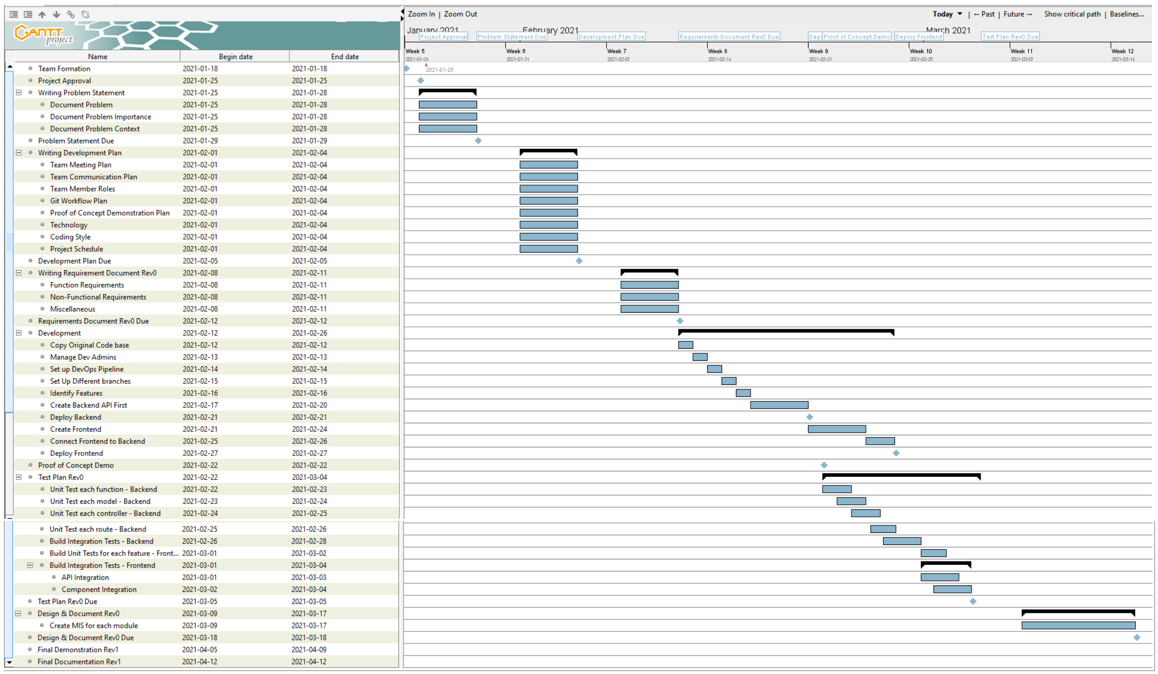
1. We will be using Function Components.
2. Interfaces will have a leading "I" prefacing the name of the variable. For example: "ITodoItem" will represent the interface for a "TodoItem".
3. Props will always end with "Props". For example: "TodoItemProps" will represent the props object being passed into the "TodoItem" component.
4. Stylesheets pertaining to a particular component will always be located at the bottom of the page.

Python Rules

1. Following *Pep8* Style Guide, using *Flake8* as our automated linter.
2. Automatically updates coding style to fit *Pep8* guidelines, therefore the *Flake8* linter provides adherence.

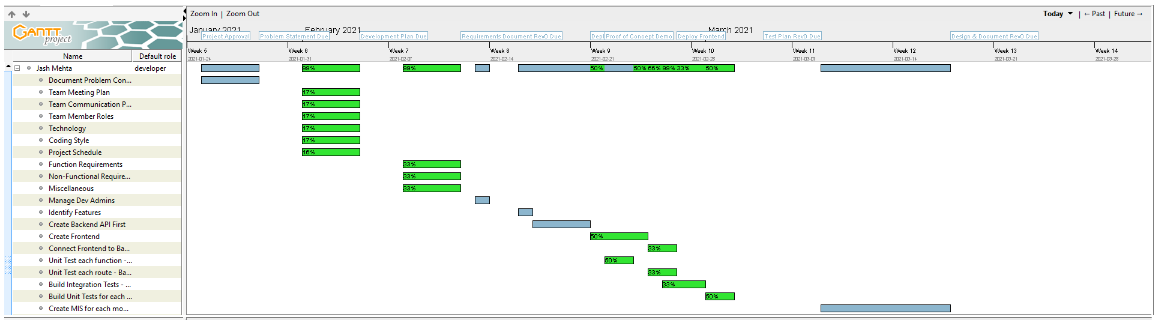
Project Schedule

Gantt Chart

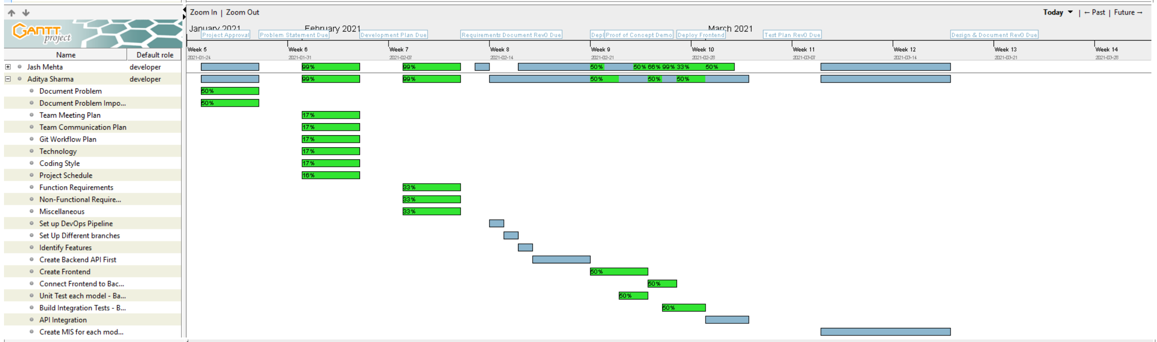


Name	Begin date	End date
Team Formation	2021-01-18	2021-01-18
Project Approval	2021-01-25	2021-01-25
Writing Problem Statement	2021-01-25	2021-01-28
Document Problem	2021-01-25	2021-01-28
Document Problem Importance	2021-01-25	2021-01-28
Document Problem Context	2021-01-25	2021-01-28
Problem Statement Due	2021-01-29	2021-01-29
Writing Development Plan	2021-02-01	2021-02-04
Team Meeting Plan	2021-02-01	2021-02-04
Team Communication Plan	2021-02-01	2021-02-04
Team Member Roles	2021-02-01	2021-02-04
Git Workflow Plan	2021-02-01	2021-02-04
Proof of Concept Demonstration Plan	2021-02-01	2021-02-04
Technology	2021-02-01	2021-02-04
Coding Style	2021-02-01	2021-02-04
Project Schedule	2021-02-01	2021-02-04
Development Plan Due	2021-02-05	2021-02-05
Writing Requirement Document Rev0	2021-02-08	2021-02-11
Function Requirements	2021-02-08	2021-02-11
Non-Functional Requirements	2021-02-08	2021-02-11
Miscellaneous	2021-02-08	2021-02-11
Requirements Document Rev0 Due	2021-02-12	2021-02-12
Development	2021-02-12	2021-02-26
Copy Original Code base	2021-02-12	2021-02-12
Manage Dev Admins	2021-02-13	2021-02-13
Set up DevOps Pipeline	2021-02-14	2021-02-14
Set Up Different branches	2021-02-15	2021-02-15
Identify Features	2021-02-16	2021-02-16
Create Backend API First	2021-02-17	2021-02-20
Deploy Backend	2021-02-21	2021-02-21
Create Frontend	2021-02-21	2021-02-24
Connect Frontend to Backend	2021-02-25	2021-02-26
Deploy Frontend	2021-02-27	2021-02-27
Proof of Concept Demo	2021-02-22	2021-02-22
Test Plan Rev0	2021-02-22	2021-03-04
Unit Test each function - Backend	2021-02-22	2021-02-23
Unit Test each model - Backend	2021-02-23	2021-02-24
Unit Test each controller - Backend	2021-02-24	2021-02-25
Unit Test each route - Backend	2021-02-25	2021-02-26
Build Integration Tests - Backend	2021-02-26	2021-02-28
Build Unit Tests for each feature - Front...	2021-03-01	2021-03-02
Build Integration Tests - Frontend	2021-03-01	2021-03-04
API Integration	2021-03-01	2021-03-03
Component Integration	2021-03-02	2021-03-04
Test Plan Rev0 Due	2021-03-05	2021-03-05
Design & Document Rev0	2021-03-09	2021-03-17
Create MIS for each module	2021-03-09	2021-03-17
Design & Document Rev0 Due	2021-03-18	2021-03-18
Final Demonstration Rev1	2021-04-05	2021-04-09
Final Documentation Rev1	2021-04-12	2021-04-12

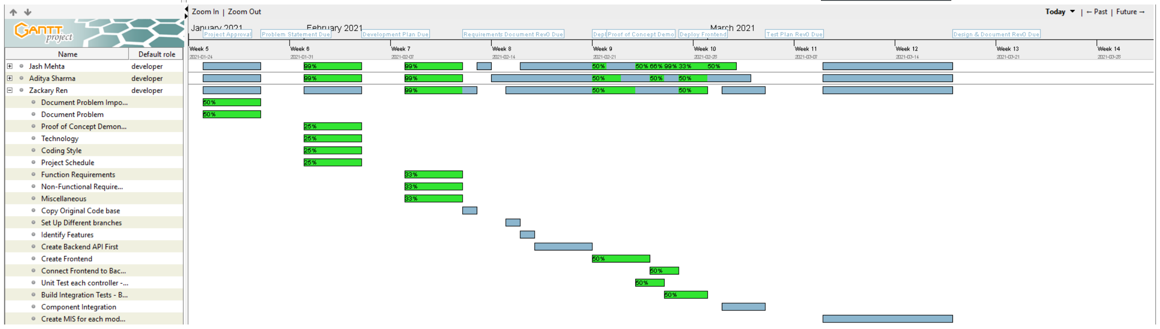
Resource Allocation



Name	Default role
Jash Mehta	developer
Document Problem Con...	
Team Meeting Plan	
Team Communication P...	
Team Member Roles	
Technology	
Coding Style	
Project Schedule	
Function Requirements	
Non-Functional Require...	
Miscellaneous	
Manage Dev Admins	
Identify Features	
Create Backend API First	
Create Frontend	
Connect Frontend to Ba...	
Unit Test each function -...	
Unit Test each route - Ba...	
Build Integration Tests - ...	
Build Unit Tests for each ...	
Create MIS for each mo...	



Name	Default role
Jash Mehta	developer
Aditya Sharma	developer
Document Problem	
Document Problem Impo...	
Team Meeting Plan	
Team Communication Plan	
Git Workflow Plan	
Technology	
Coding Style	
Project Schedule	
Function Requirements	
Non-Functional Require...	
Miscellaneous	
Set up DevOps Pipeline	
Set Up Different branches	
Identify Features	
Create Backend API First	
Create Frontend	
Connect Frontend to Bac...	
Unit Test each model - Ba...	
Build Integration Tests - B...	
API Integration	
Create MIS for each mod...	



Name	Default role
Jash Mehta	developer
Aditya Sharma	developer
Zackary Ren	developer
Document Problem Impo...	
Document Problem	
Proof of Concept Demon...	
Technology	
Coding Style	
Project Schedule	
Function Requirements	
Non-Functional Require...	
Miscellaneous	
Copy Original Code base	
Set Up Different branches	
Identify Features	
Create Backend API First	
Create Frontend	
Connect Frontend to Bac...	
Unit Test each controller -...	
Build Integration Tests - B...	
Component Integration	
Create MIS for each mod...	

Project Review