**CS673 Software Engineering** 

**Team: A-Team**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Bharat Gogineni | Team leader | *Bharat Gogineni* | 05/16/2022 |
| Ruiqi Chang | Requirement Leader | *Ruiqi Chang* | 5/16/2022 |
| Chinmay Bhelke | Design and Implementation Leader | *Chinmay Bhelke* | 5/15/2022 |
| Benoît Clemenceau | QA Leader | *Benoît Clemenceau* | 5/15/2022 |
| Taina Conde | Configuration Leader | *Taina Conde* | 5/23/2022 |
| Chinmay Bhelke | Security Leader | *Chinmay Bhelke* | 5/15/2020 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| Version\_Iteration1 | Taina, Bharat, Chinmay | **May 21th** | Related Work, Timeline,Proposed High-Level Requirements |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Overview](#_heading=h.gjdgxs)

[Related Work](#_heading=h.30j0zll)

[Proposed High level Requirements](#_heading=h.1fob9te)

[Management Plan](#_heading=h.3znysh7)

[Objectives and Priorities](#_heading=h.2et92p0)

[Risk Management (need to be updated constantly)](#_heading=h.tyjcwt)

[Timeline (need to be updated at the end of each iteration)](#_heading=h.3dy6vkm)

[Configuration Management Plan](#_heading=h.1t3h5sf)

[Tools](#_heading=h.4d34og8)

[Deployment Plan if applicable](#_heading=h.2s8eyo1)

[Quality Assurance Plan](#_heading=h.17dp8vu)

[Metrics](#_heading=h.3rdcrjn)

[Code Review Process](#_heading=h.26in1rg)

[Testing](#_heading=h.lnxbz9)

[Defect Management](#_heading=h.35nkun2)

[References](#_heading=h.1ksv4uv)

[Glossary](#_heading=h.44sinio)

# Overview

The Group 2 Project is a Job Tracker Web Application called FollowUp. It is developed with a ReactJS frontend with a NodeJS Backend web server.

Nowadays, it is not unusual to apply for multiple jobs at a time. However, handling many applications at once can be challenging. FollowUp aims to help job candidates better manage the status and next steps of their applications. The goal is to help them not miss any deadlines, better prepare for the upcoming steps, and, therefore, improve their chances of getting the job offer they desire.

In order to achieve this goal, FollowUp enables users to keep track of and make changes to the various activities involved in the lifecycle of a job application - such as prep work, phone screens, coding assessments, in-person interviews - to name a few.

Users can log in to the web application and view the status of their tasks for every job application and this information is pulled from a MySQL Database running on a NodeJS Express Server hosted in the cloud.

Users can make CRUD operations against the various tasks involved for the job application and these changes are persisted to the backend DB through a REST API Interface.

# 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.)a

Huntr -"Track your job applications and keep your job search organized, all in one place"- <https://huntr.co/> - "Huntr keeps track of every detail about your job opportunities regardless of where you found them. Track contacts, notes, dates, tasks, job descriptions, salaries, locations, company data and more. It's like a CRM for your job search."

While Huntr's main goal is to organize job applications in cards, FollowUp's main goal is to help the user not miss any important deadlines for activities related to the job applications.

To accomplish this goal, FollowUp not only organizes the job applications in cards but also distinguishes the cards' colors based on the urgency of the next activity related to that card.

For example, suppose the user is applying for the software engineer position at Amazon. The user adds a card filling out that job application information and adds an activity called "Phone Interview". This activity is going to happen in 3 days. That job card would have the color red because the "phone interview" is in 3 days from now. If another card's next activity date is within 7 days, the color of the card would be yellow. Lastly, if there is another card in which the next activity date is more than 7 days away, the card would be green. This color system will make it clear to the user that an activity is coming for that card that the user needs to prepare accordingly.

Another differentiating feature FollowUp has is the "follow up" alert that appears on a card that hasn't been updated for more than 7 days. This feature helps the user to keep track of job applications that the user hasn't heard from for a while.

# 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)

* + 1. Desirable Features (the nice features that you really want to have too):
    2. Optional Features (additional cool features that you want to have if there is time):
    3. Existing Features (delete this item if your project starts from scratch):
  1. Nonfunctional Requirements
     1. Security requirements

| 3. Proposed High-Level Requirements   1. **Functional Requirements**    1. **Essential Features**       1. **User Account -** Users must be able login/register into the system.       2. **Jobs Dashboard -** Central Portal to monitor/Create and Track Dashboards.Will serve a list of jobs that are currently being monitored based on priority and dates.       3. **Notes -** To help users add notes to keep track of each job       4. **Dates -** To allow users to keep track of important events upcoming.       5. **Color system -** use different colors for the cards to let the user know which card has the most upcoming activity.    2. **Desirable Features**       1. **User Accounts**          1. Self register       2. **Notifications -** To ensure users know their upcoming events       3. **Status -** Let users know the current status of their jobs applied       4. **Follow-up alert -** notify the user if a card hasn't been updated for more than 7 days.    3. **Optional Features**       1. **Customizable configurations - (dark mode and other)**       2. **Front end notifications in real-time**       3. **Company updates(Tweets/Stocks/Press releases)**       4. **Share Job Board with other users**       5. **Track from mail** 2. **Non Functional Requirements**  * **Usability**   Create a simple, understandable design for the users. Use easily recognizable icons. Use an established color system for the cards to make them understandable for the users (red for urgent, yellow for not so urgent, green for not urgent).  The web application should be designed to be accessible to persons of disability such as individuals suffering from color blindness and deafness and the application will be designed to be 508 Compliant   * **Reliability**  The application should have an uptime of >99% when the application is live and the frontend will implement error handling in the case of unavailability. Fallbacks and logging capabilities in case of failures * **Compatibility** * All modern browsers such as Google Chrome , Mozilla Firefox , Apple Safari , and IE Edge must be supported * **Responsiveness**   The application should automatically adjust for modern widescreen display monitors and mobile device screens such as standard 27” LG/Sony monitors and standard and large size Apple and Android smartphones.   * **Security** The backend web server should be secure and unauthenticated users will not be able to make requests to the backend web server without an authorization token |
| --- |

# Management Plan

## Objectives and Priorities

(Please describe your project objectives with the highest priority first. Project Goals can include but are not limited to completing all proposed (essential) features, deploying the software successfully, the software has no known bugs, maintaining high quality, etc )

1. Ensure deliverables of all essential and desirable features are completed.
2. Architect the application such that it is maintainable.
3. Test-Driven architecture to avoid loss of time on regression testing.
4. Ensure good CI/CD practices are utilized for easy deployment/
5. Ensure that all Team members have the best learning experience through interactions with each other.
6. Maintain an ecosystem for the codebase that can be taken forward for future enhancements.
7. Maintain best practices of software engineering by following the 12 Principles of Agile Methodology.

## 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. Then use the separate google sheet for detailed risk management. The template is provided in the same folder with this file. Please provide the link to the sheet.)

For now, our team has identified the following main risks:

1. Time management and availability of the Team: Considering the short duration of the project and that the team members have outside work or personal obligations, the group identified a high risk of not consistently performing the assigned work for the duration of the project. The group plans to manage this risk by communicating to the other members in advance when someone is not available to work on the assignments for the week. In addition, the team leader will distribute the work among the team members keeping in mind the availability of each team member.
2. Scope Creep: The group identified a high risk of unplanned and unaccounted for tasks arising for the duration of the project. To manage this risk, the group will develop a process to ensure requirements are constant and every team member's responsibilities have clarity.
3. Messy Code: the group identified a medium risk that the code is not understandable and is difficult to read. The plan to manage this risk is to use Typescript, define types and isolate components and responsibilities in order to have clean code.
4. Lack of testing knowledge: the group identified a medium risk that Proper Testing has not been implemented for the project. The team plans to use regression testing, unit testing, integration testing, functional testing, and non-functional testing to make sure the project has enough test coverage.

**Risk Management Sheet Link:https://docs.google.com/spreadsheets/d/1R5Lt0mRCwtKGFjtehPlGroS\_AILdAX7S/edit#gid=2024528617**

## Timeline (need to be updated at the end of each iteration)

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours | Presentation Recording Link (5-10 minutes for iteration 1 and 2) |
| --- | --- | --- | --- | --- |
| 1 | User Login  User Signup  User Security | 1. Create frontend to Create Users and Signup 2. Create Database and schema to handle Users and Fields 3. Integrate Backend and Frontend to handle application logic | 20 |  |
| 2 | User Job Creation  User Job Updates | 1. Create and test REST API’s for User CRUD Actions 2. Integrate Functionality with Backend DB from Frontend | 20 |  |
| 3 |  |  |  |  |

# Configuration Management Plan

## Tools

(In this project, we will use Git and Github as the version control tools. Please also specify any other tools to be used, e.g. IDE tools, CI/CD tools, container tools, SAST or DAST tools, and any other DevOps tools)

* + 1. Git for Version Control + Git CLI
    2. Github as the Git Service
    3. Github Pages to host our frontend
    4. GitHub Actions
    5. Mocha and Chai for testing
    6. NodeJS
    7. Create-React-App for client-side code development
    8. AWS EC2 for our backend Web Server
    9. iTerm2 / Powershell to SSH into our Backend
    10. VSCode for Writing Code
    11. Docker
    12. NPM
    13. ExpressJS as our Backend Web Server Framework
    14. Postman for REST API Testing
  1. **Code Commit Guideline and Git Branching Strategy**  
     (Please briefly describe the criteria for the code commitment and the branching strategy used, e.g. what are the branches to be used, how the pull request will be used etc. Here is an article to give you some basic knowledge about different git branching strategies: <https://www.flagship.io/git-branching-strategies/>
* Commit messages should briefly describe what changes were made and follow the rules below, which were extracted from the article <https://cbea.ms/git-commit/>:

1. Capitalize the subject line;
2. Limit the subject line to 50 characters
3. Separate subject from body with a blank line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and why vs. how

* The team will adhere to the GitHub Flow branching strategy described on <https://www.flagship.io/git-branching-strategies/>. Therefore, each developer will create feature branches directly from the main branch. Before merging the feature branch back to the main, the code has to be properly tested. Then, the developer creates a pull request. Another team member reviews the code and approves the pull request if no issues are found.

## Deployment Plan if applicable

(If you plan to deploy your application (e.g. your web application), briefly describe how you plan to deploy your application).

The team is planning to deploy the web application using the GitHub Actions tool in an AWS EC2 instance

# 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** |
| --- | --- |
| LOC | Lines of code. |
| Number of tests | Self-explanatory. |
| Test Coverage | Percentage measures how much our source code is being tested with our unit tests. Divided into: Function coverage, Statement coverage, Branch coverage, and Line coverage. |
| Percentage of tests passed | Counts the number of tests that have passed as a percentage of total tests runned. |
| Total test duration | How long it takes to run the automated tests. |

* 1. Coding Standard

(Describe any coding standard to be used)

* Front end (TypeScript / React):
* Following Google TypeScript Style Guide: <https://google.github.io/styleguide/tsguide.html>
* The standard for javascript will be found at http://www.w3schools.com/js/
* Files should not have more than 500 LOC.
* Using JSX syntax, Function Components, and Hooks.
* Following React+TypeScript Cheatsheets: <https://github.com/typescript-cheatsheets/react>
* Back end (JavaScript / Express):
* Following Google JavaScript Style Guide: <https://google.github.io/styleguide/jsguide.html>
* Files should not have more than 500 LOC.

## 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?)

* Commits should be as concise as possible and easy to review and merge
* Everyone should comment on their code.
* Team member fills out a pull request of his feature branch onto the main branch.
* Tests are being executed by GitHub Actions.
* Build script is being executed to see if it goes through.
* Pull request is reviewed by the Design and Implementation Leader (or another member to whom he has delegated the task).
* If everything goes well, pull request is accepted and the new feature is merged.

## Testing

(Both manual testing and automated testing should be considered. Both unit testing and integration testing should be considered. Briefly describe the testing tools/framework to be used, the personnel involved (e.g. the QA leader will focus on the integration testing and each developer will unit test their own code), when and what types of testing will be performed, the testing objectives, etc)

**Regression testing**: If a new portion is introduced to the original part (for example, new functionality), the entire part should be retested.

**Unit Testing**: individual source code unit tests.Will use Mocha and Chai/Jest. Team members will write the test cases.

**Integration testing**: when assembling developed parts together, we should provide integration testing to ensure that these components function together properly. Will be performed by QA Leader

**Functional testing**: We should provide the required inputs (for example, user profile, messages, and operations) with the actual results with the expected result

**Nonfunctional testing**: includes performance, load/stress, usability, and other factors. Recoverability, compatibility, security, installation, and serviceability are all factors to consider.

Acceptance testing: ensuring that all of the plan's requirements are satisfied.

## Defect Management

(Describe the tool to be used to manage the defect (e.g github issues). The types of defects to look at. The actions or personnel for defect management. )

i) Any team member can report one and give it a severity from P0 - Critical to P3 - Trivial and a Priority high to low

**Criteria of severity**:

* \*Critical: cause the application to crash with a high frequency and fail to meet more than two criteria  
  \*Serious: cause at least one scheduled requirement to fail with no alternate methods.
* \*Trivial: the fault has little impact on the system's usability, and the necessary goals can be easily achieved by working around the flaws.
* \*Medium: a flaw in a system that is being improved, with modifications to the look and scope of the program.

**Priority criteria**: establish the order in which the fault should be repaired.

* \*high: repair as soon as feasible
* \*medium: should be fixed as part of the development process
* \*low: can be fixed after more significant flaws have been fixed

ii)**Types of defects**

* Documentation: Documents, remarks, or messages are illegible or inaccurate.
* When compiling, several syntax and static problems can be found.
* Interface: Incorrect design or usage of interfaces (incomplete, incorrect, or inappropriately utilized class, method, or data type interfaces, objects are invisible, etc.).
* Data: incorrect variable value, variable structure violation
* System: Timing, synchronization, network, hardware, or similar issues
* Defect in the development environment or supporting systems

iii) **Tools**

Every iteration phase, team members who discover flaws should update the GitHub repository. Define defects as specified in the criteria above.

Track defects in a user narrative using "Pivotal Tracker" and debate how to enhance the product. A problem should be committed to GitHub with a clear 'commit name' when it has been changed. "Modified bug1 in View2(Problem 3 is fixed)," for example.

The QA leader should write down the overall number of defects and their causes at the conclusion of each iteration.

# 

# 

# References

TypeScript Standard:

<https://google.github.io/styleguide/tsguide.html>

JavaScript Standard:

<https://google.github.io/styleguide/jsguide.html>

React Standard:

<https://reactjs.org/docs/design-principles.html>

Agile Project Managment Standard:

<https://www.altexsoft.com/whitepapers/agile-project-management-best-practices-and-methodologies/>

# 

# Glossary

(Any acronym used in the document should be explained here)

| **Acronym** | **Meaning** | **Description** |
| --- | --- | --- |
| **NPM** | Node Package Manager | JavaScript Package Manager |
| **JS** | JavaScript | JavaScript is a programming language which is primarily used for client-side and server-side development for web applications |
| **TS** | TypeScript | TypeScript is a programming language and is a superset of JavaScript which gets compiled down to JavaScript at build time |
| **HTML** | HyperText Markup Language | A standardized system for tagging text files to achieve font, color, graphic, and hyperlink effects on World Wide Web pages |
| **CSS** | Cascading Style Sheets | Cascading Style Sheets (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. |
| **CRUD** | Acronym for: create, read, update and delete | Create, read, update and delete are the four functions considered necessary to implement a persistent storage application |