**Table of Contents**

1. **Introduction**
   1. *Purpose*
   2. *Scope*
   3. *Overview*
   4. *Need for the System*
   5. *Similar Apps*
2. **Functional Analysis**
3. **Operational Scenario**
4. **Other Features**
5. **Interfaces with the Device**
6. **Testing the Application**
7. **Sample Use Case**
8. **Glossary**
9. **Space for Updating Versions and Comments**

**1. Introduction**

The following section contains a brief overview of the Software Requirement Specification document. It will serve as an introduction to what the project is and why it is being developed.

1.1 Purpose

This document provides details about the usage and purpose of the Issue Tracker webapp. The general direction of development goals and standards will be provided throughout this document.

1.2 Document Conventions

This document has been written based on the IEEE template for System Requirement Specification documents.

1.3 Scope

The Progress Tracker webapp is being designed for developers such as myself to use as a flexible tool to keep track of the issues and by extension, the progress of development. The intended audience however is the hiring manager as this will be a personal portfolio project, so it is being designed to display different types of features. Therefore the scope of this document is to plan features and keep an updated vision of the project’s goals.

1.4 Need for the System

Development teams use an issue tracker for software bugs while help desk teams use an issue tracker for help desk tickets. It appears to be a ubiquitous tool in the industry and helps with scaling, logging data and documentation. As a utility, it will help enable greater control over projects for developers and organisations alike. It will provide an insight into the development process and design details as well as serving as a personal learning opportunity for greater hiring appeal.

1.5 Similar Apps

Jira

Github Issue Tracker

Enterprise Help Desk Systems

**2. Functional Analysis**

The ability to type an issue into a textbox and submit to save to a database.

Description: This will be a core feature that creates and saves new issues into the database.

Implementation: Issue creation will have its own page. Once saved, the webapp will take you to the specific issue just created. A quick version to create an issue is to be displayed on the landing page itself.

Criticality: This feature is top priority as it is part of the CRUD database operations.

Risks: The wrong information may be entered. The user may forget to save the issue number. We will have user account functionality which will have issues created linked to the user.

Dependency: UI, frontend framework. A database but the logic to perform operations will exist in the controller logic layer according to the MVC model.

Landing page that shows cards layout – left card has top issues, middle card has progress tracker, right card has velocity/sprint/tasks completed data, then a quick submission section bottom.

Description: This will be the initial landing page, after authentication (to be added as an extra feature), serves as a dashboard.

Implementation: A UI featuring 3 cards with number countup library and calls to controller to retrieve data from model.

Criticality: This feature must look professional as it will be the main landing page. Must be easy to use, top priority in terms of design.

Risks: Difficulty to use, could end up with too many dependencies. Could end up cluttered.

Dependency: UI libraries, .NET implementations. Requires data from the model.

Switch to a screen with all issues presented in a table-ish format

Description: The “View All” screen. The page to see current issues.

Implementation: Ideally a WebGL applet can be loaded to show issues visually, this will be explained in more detail in the Other Features section. This page will have 25 issues sorted by oldest alongside html # links that sort by newest, and increase number of issues to 50, 100 and 200. There will also be a search feature sorted by a time range, closed job with oldest and newest, and search by ID or issue tag.

Criticality: High priority. Considered the main “Retrieve” page for CRUD operations.

Risks: Sorting can lengthen development time, bugs can affect sorting issues. Make sure to use code comments. This page can also be highly subjective, a working product is enough until UI cleanup.

Dependency: Third party WebGL library, 3D graphics expertise, database search tools.

User ability to update/delete issues, set urgency levels, deadlines, due dates, issue comments, issue history and notifications.

Description: Urgency levels, deadlines, due dates, issue comments, issue history and notifications are hereby known as Core Issue features.

Implementation: User must be able to set and update Core Issue features at any time. Core Issue features can be deleted too but a changelog tracks all changes, publicly viewable.

Criticality: High priority as these features will perform part of the CRUD operations.

Risks: Bugs could permanently delete database data. Set up an offsite backup solution.

Dependency: Database host, backup software account details.

Calendar integration

Description: Calendar that tracks due dates.

Implementation: Applet on top right of landing page and potentially every page.

Criticality: Medium priority. Core features can be finished fist, but a calendar should be kept in mind as code progresses.

Risks: Lengthen development time, could look unprofessional. Keep it simple.

Dependency: Will try and minimise third party tools, however it is probable that we will use a third party library.

Auth0 integration

Description: Authentication feature built into the webapp.

Implementation: Added after core webapp is built.

Criticality: Medium priority. It is essential for the purposes of fully completing the project.

Risks: Google could get hacked, data is not under our control.

Dependency: Auth0 Google software.

Attachments to be allowed to include with issues/comments.

Description: Allows users to upload attachments like pictures to issues.

Implementation: Feature to be included on the issue creation and update screens.

Criticality: Medium priority. Core features can be finished fist.

Risks: Lengthen development time, could enable DDOS attacks and/or bandwidth problems.

Dependency: Internet speeds, data file formats.

**3. Operational Scenario**

Landing Page:

1. Landing page served by the controller after Auth0
2. Dashboard/WebGL display will show summary of issues
3. List of issues will either be on the right, or be accessible via a link “Show All Issues”
4. At bottom of page, textbox widget will be available with “Quick Issue Submission”
5. Another button “Create an Issue” clearly hyperlink
6. Each issue displayed has an edit feature

Issue Creation:

1. Textbox with HTML/editor features around it
2. Submit issue

Issue Update:

1. Issue page, each issue has its own unique page
2. Update, delete, read comments, and other info displayed
3. Changelogs shown

Dashboard Logs:

Access Log:

1. Shows all database operations with time and user
2. Shows all logins including IP address and other info about user

Submission Log:

1. Shows all issues submitted
2. Open or closed
3. Time frame (default – last 2 weeks)

About Us:

1. Contact Us
2. Textbox submission – email or text (leave email or phone number)

Calendar:

1. Reachable on any page
2. Filter by urgency

Hamburger Menu:

1. Reachable on any page
2. Takes you to all pages
3. Logout (if Auth0 implemented)

**4. Other Features**

Chatbox:

Chat with others on chatbox - uses websocket to cloud server

Messaging connectivity:

Telstra mobile API

WebGL custom feature

Main UI page a JS feature that shows a spinning pole kind of thing that is a timeline going bottom to top and you can see it from different directions, possibly webGL to show it off, also have button that filters by urgency level

Route optimisation and/or business list

Google maps API

Login/project information

Github API

Project Board

Project Owner, Kanban, Emails/Announcements

**5. Interfaces**

Initially planned to be a webapp using MVC design pattern. Frontend UI will consists of .NET framework and C# language. As MVC is modular, the frontend will exist almost entirely separately to the logic, therefore new versions of the frontend can be implemented without affecting the business logic of the application. The backend server-side language will consist of Java on top of either a Google Firebase or AWS solution, or both.

**Hardware**:

Server – Google Firebase/AWS  
Frontend – .NET webapp  
Keyboard inputs through web browser

**Software:**

Web Browser (compatible with: Firefox, Google Chrome, Internet Edge, Safari)  
C# .NET application served through web browser  
Database CRUD operations using controller logic  
Auth0 authentication library

**6. Testing the Application**

**Testing Process**

1. The web application will be tested by myself. It will be shown to friends and family with opportunities given for them to create, retrieve, update and delete issues. They will provide feedback on the UI and designs I have used.

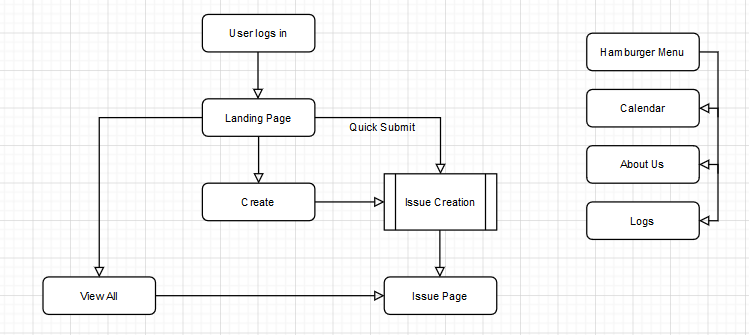
2. I will participate in a code review and check for any security flaws or system bugs. They will provide feedback on my code quality and style.

3. I will use this project as a portfolio example. I will receive feedback from hiring managers and interviewers on the efficacy and design from a hiring perspective.

4. This project will be hosted on the internet, preferably using Google Firebase. Other options will be explored and tested.

**7. Sample Use Case**

Figure A.



**8. Glossary**

|  |  |
| --- | --- |
| **Word or Phrase** | **Definition** |
| Core Features | Urgency levels, deadlines, due dates, issue comments, issue history and notifications |
| MVC | Model View Controller – web design pattern |

**9. Space for Updating Versions and Comments**

Changelog:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version Number** | **Removed Features** | **Added Features** | **Date Published** |
| 0.1 | - | Created document | 14/05/2020 |

Useful Links:

Github:  
https://github.com/andrewpublic/job-meercat

Jira:  
https://andrewpublic.atlassian.net/secure/RapidBoard.jspa?rapidView=1&projectKey=JM10