CPSC 3720 SPRING 2019

TrackMagnet

Embla (Team E)

Cassady Campos
Logan Wilkie
Nwabunor Onwuanyi

October 8/2019

TABLE OF CONTENTS

Introduction 3

Project Management 4**Error! Bookmark not defined.**

Risk Management 4

Project Schedule 5

Design 6**Error! Bookmark not defined.**

Appendices 6

INTRODUCTION

This is the initial proposal for Team Embla's CPSC 3720 RESTful system project, with team members Logan Wilkie, Cassady Campos and Nwabunor Onwuanyi. This document will go over the initial design philosophy for the Issue Tracking System we have been tasked with creating, including design principles, team management tactics and the initial schedule for the project, ending the proposal with a collection of REST endpoint designs and prototype web-layout templates. Our goal for this project is to produce a consistent, working and easy to use version of the software at the end of each sprint, with a completed version at the end that we are happy with and can hopefully use and improve upon in our future endeavours.

PROPOSAL

We want to create an Issue Tracking System that is easy to view, sort, edit and navigate. Multiple workspaces can be created to sort through user projects, keeping issues separate and focused on the current workspace. The tracking system will have any issues shown in a Board view, with clean-cut categories showing the stages an issue is going through, allowing users to sort the issues with several options. Users will be allowed to add their name to the list of users, as well as view other user profiles with their ID#'s, tasks assigned to them, and total tasks completed by that user. The list of features currently planned are as follows:

- Issues
    - Create an issue
        - Set title
        - Set Status
        - Set Priority
        - PDF Screenshot of issue (optional)
    - Get an issue (information, history, comments on the issue, timestamps)
    - List All issues (because of the board layout, this is fairly intuitive)
    - Sort Issues (sort by date, priority, ID, assignee)
    - Assign an Issue to a user
    - Update an Issue
        - Update Title
        - Update Status
        - Add PDF screenshot
    - Delete an Issue
- Comments
    - Add comment to an issue
    - Get a comment from an issue
    - List all comments on an issue
    - Update a comment of an issue
    - Delete a comment from an issue

- Users
    - Create or Add a user to the workspace
    - List all users
    - Remove a user
- Workspace Functionality, separation of different groups using the system

The prototype images for the user interface can be found in the appendix (1,2,3). There will be 3 main tabs: Issues, Users, and Help/About. 'Issues' will bring up the board-like view of the current issues, placed into 4 sections: Backlog, In Progress, Reviewing, and Closed. Ideally, the issues will work their way left to right, showing their progress towards being fixed. Clicking on an issue will bring bring the issue up on a full webpage, allowing the user to scroll through comments, as well as provide additional comments for the issue. The 'Users' tab will bring up the list of users, showing their information including ID, # of issue assigned to them. The 'Help/About' will show the user manual and FAQ/System Info. We are hoping for a modern yet simple looking interface, where most of the information is not hidden behind a series of menus. Everything should be displayed up front for the user to see and navigate as they please, all the while avoiding clutter and overwhelming the user with information.

PROJECT MANAGEMENT

RISK MANAGEMENT

Design Issues:

If the planned project design is too big, we will look for small areas with little to no effect on the system to cut out. Additionally, we can look for any function that can be cut down to lessen the size of the project to be able to meet the deadlines. The biggest of these "too big" sections is most likely to be our webpage HTML layout. If our planned design does not mesh well with the actual implementation, changes will need to be made to accommodate the implementations, most likely simplifying the webpage by a significant amount.

If we get to the point that we need to make design changes, we will proceed differently depending on the type of change. If it's a small or minor change, the group member will tell others first, then proceed to make the change(s), and finally push the final product to the rest of the group. If its a major change, we will bring it up to all other group members, and we all will discuss what to change and how to update the design, coming together to brainstorm a solution as quickly as possible.

A large part of our design issues will be from our background in C++ programming and the idea of carrying states forward and backward. As discussed in lecture, the idea of states for a RESTful system are much different than that of a typical C++ project, and it is something we don't fully have a grasp on yet.

Estimation Issues:

If a part takes longer than anticipated, we all will help out a little bit (but not so much as to get in each others way and everyone else can still get their tasks done) to get it done. If it's still not done, then we will provide additional help to the parts that need it most.

People Issues:

In the case of an additional member (last minute) who gets assigned to the group, each group member will catch them up to speed on the parts they are currently working on. It would be the responsibility of the new member to review older portions of the project and understand them. Initially, we might have the new member buddy-up with an existing member to get them started, or work with a smaller, easy-going task. In the case of unproductive group members, or those who don't show up/don't do their portion, we will give them a few warnings before it becomes an issue with Dr. Anvik.

In the case of someone being sick, each member will just work on their parts of the project. If one team member's portion requires the functionality of the sick team member work, the sick member will explain their functionality to the rest of the group, and work from there until all functionality is implemented or restored. If this solution is not effective, then we will shuffle the assigned jobs around team members to accommodate.

Learning & Tools Issue:

In the case of inexperience with new tools, it's on that person to ask for help with it or learn how to operate the tool.

In the case of a steeper than expected learning curve, the person with the most knowledge of the tool (if able to) explains how the tool works. Otherwise this it's up to them to learn how the tool works individually.

PROJECT SCHEDULE

Sprint 1	**Sprint 2**	**Sprint 3**
from Oct 14 to Oct 25	**from Oct 28 to Nov 8**	**from Nov 11 to Nov 22**
Create a new issue **Get a new issue** **Create user** **Delete user** **Update an issue** **Delete an issue**	**List all users** **List all issues** **Assign an issue** **Set title of an issue** **Update title of an issue** **Add comment to issue** **Get comment from issue**	**Get all of an issues comments** **Update a comment of an issue** **Delete a comment from an issue** **Set issue priority** **Get issue timestamp** **Sort issue**

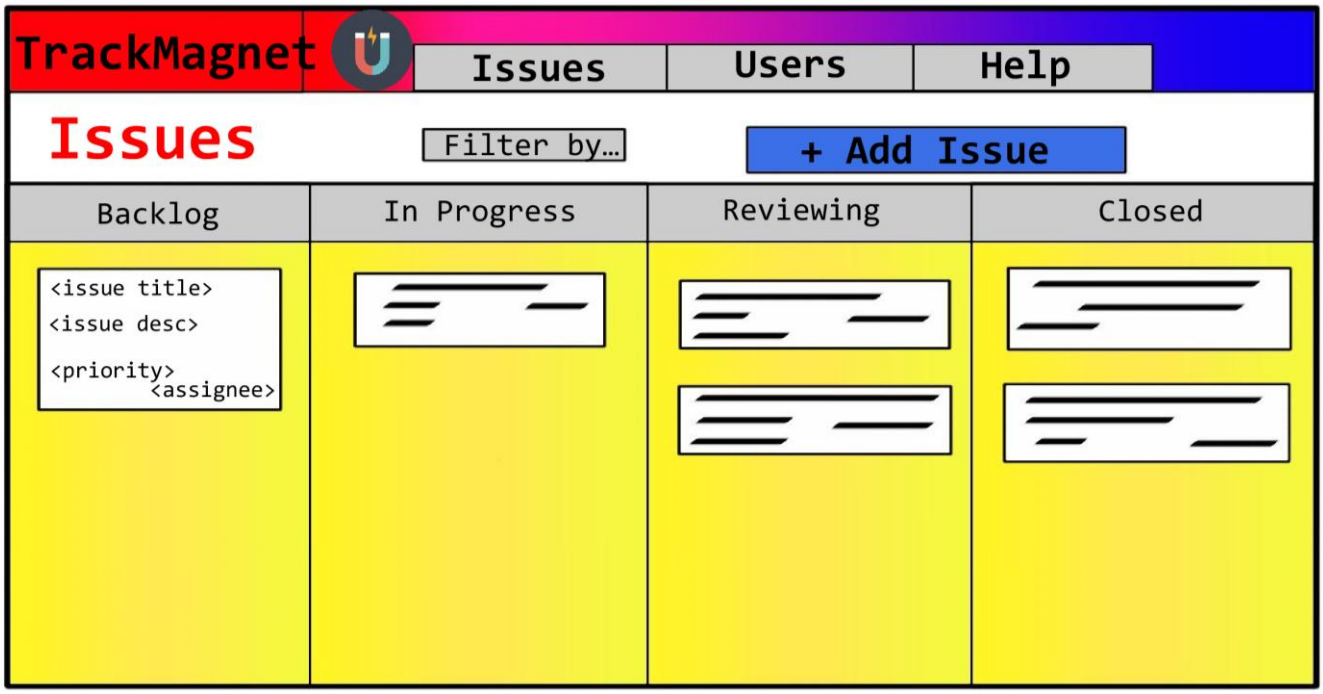
Coverage check **Style check** **Documentation**	**Coverage check** **Style check** **Documentation**	**Coverage check** **Style check** **Documentation**

DESIGN

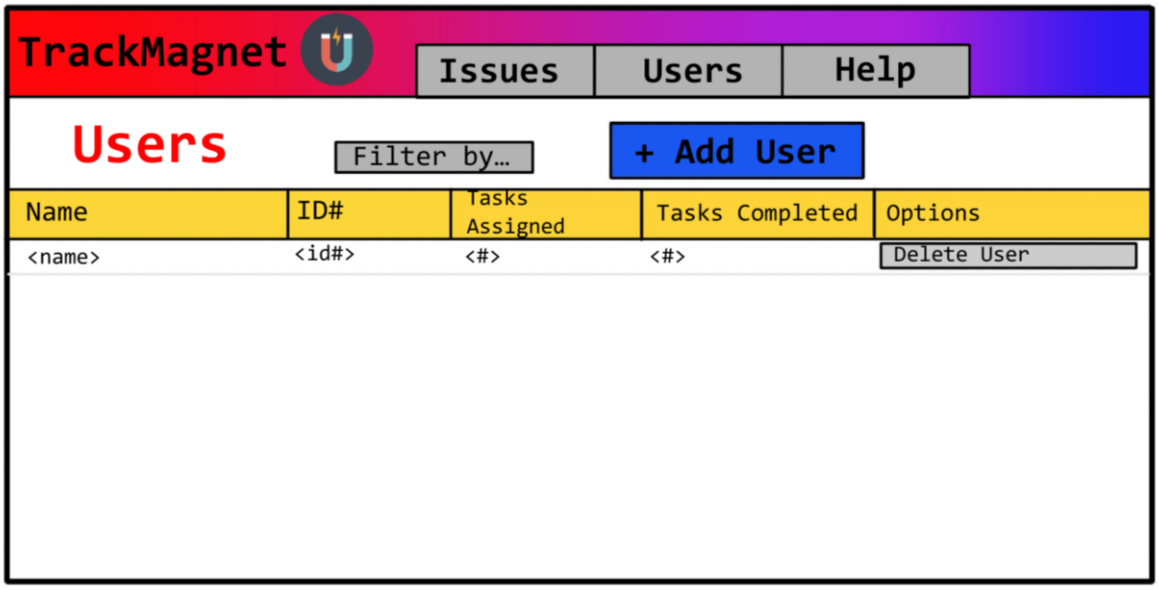
See Appendices (4) for endpoint design.

APPENDICES

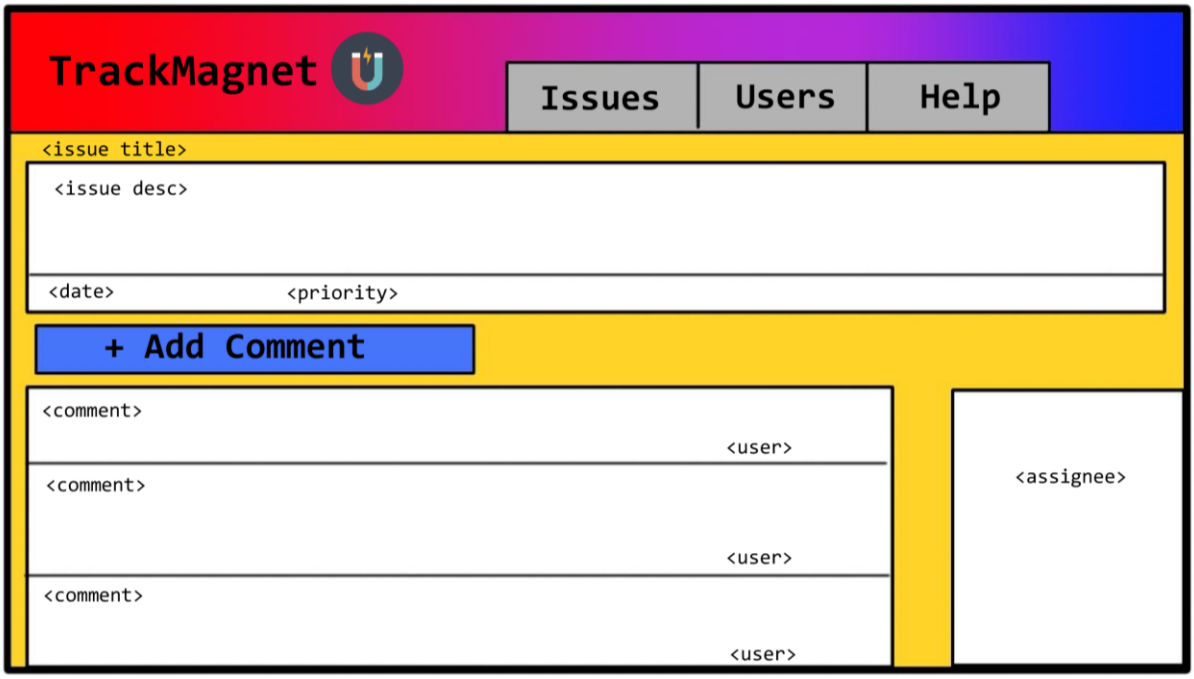
(1) Issue Tracking Board



(2) User Tab



(3) Comments on an Issue



(Colours are added to help distinguish windows, not necessarily a final decision)

(4)

Use Case	**REST endpoint**	**JSON Response**
Create a new issue	/issues/create?	{“result”: “<num>”} Returns the created issue ID
Get Issue	/issues/get?id=<num>	{ “issue”: { “id”: “<num>”, “title”: “<string>”, “description”: “<string>”, “status”: “<enum>”, “assignedTo”: “<num>”, “assignedFrom”: “<num>”, “comments”: “<comments>”, “dateCreated”: “<date>”, “priority”: “<enum>”} }
List all issues	/issues/getAll	{ “result”: “issue”: { “id”: “<num>”, “title”: “<string>”, “description”: “<string>”, “status”: “<enum>”, “assignedTo”: “<num>”, “assignedFrom”: “<num”, “comments”: “<comments>”, “dateCreated”: “<date>”, “priority”: “<enum>”} . . . “issue”: { “id”: “<num>”, “title”: “<string>”, “description”: “<string>”, “status”: “<enum>”, “assignedTo”: “<num>”, “assignedFrom”: “<num>”, “comments”: “<comments>”, “dateCreated”: “<date>”, “priority”: “<enum>”}
Assign an issue	/issues/assign?issueId=<num>, userId=<num>	{“response” : “Issue Assigned Successfully”}
Update an issue	/issues/update?issueId=<num>	{“response”: “Issue Updated Successfully”

Delete an issue	/issues/delete?issueId=<num>	{“response”: “Issue Deleted Successfully”}
Set title of an issue	/issues/setTitle?issueId=<num>, title=<string>	{“response”: “Issue title set successfully”}
Update title of an issue	/issues/updateTitle?issueId=<num>, title=<string>	{“response”: “Issue title updated successfully”}
Set status of an issue	/issues/setStatus?issueId<num>, status=<string>	{“response”: “Issue status set successfully”}
Update status of an issue	/issues/updateStatus?issueId=<num>, status=<string>	{“response”: “Issue status updated successfully”}
Add comment to issue	/comments/add?userId=<num>, issueId=<num>, comment=<string>	{“response”: “Comment added successfully”}
Get comment from issue	/comments/get?commentId=<num>	{“comment”: “id”: “<num>”, “userId”: “<num>”}
Get all of an issues comments	/comments/getAll?issueId=<num>	{“response”: {“comment”: “id”: “<num>”, “userId”: “<num>”}, . . . {“comment”: “id”: “<num>”, “userId”: “<num>”}
Update a comment of an issue	/comments/update?commentId=<num>, comment=<string>	{“response”: “Comment updated successfully”}
Delete a comment from an issue	/comments/delete?commentId=<num>	{“response”: “Comment deleted successfully”}
Create user	/user/add?username=<string>	{“response”: “User created successfully”}
Delete user	/user/delete?userId=<num>	{“response”: “User deleted successfully”}

List all users	/user/getAll	{“response”: {“user”: “userId”: “<num>, “username”: “<string>”}, . . . {“user”: “userId”: “<num>, “username”: “<string>” }
Set issue priority	/issues/setPriority?priority=<string>	{“response”: “Issue priority set successfully”}
Get issue timestamp (time it has been assigned to someone)	/issues/getTimestamp?issueId=<num>	{“result”: “<num>”} or {“error”: “Issue not yet assigned”}
Sort issues (date, priority, ID, assignee)	/issues/sort?filterBy=<enum>	{ “result”: “issue”: { “id”: “<num>”, “title”: “<string>”, “description”: “<string>”, “status”: “<enum>”, “assignedTo”: “<num>”, “assignedFrom”: “<num”, “comments”: “<comments>”, “dateCreated”: “<date>”, “priority”: “<enum>”} . . . “issue”: { “id”: “<num>”, “title”: “<string>”, “description”: “<string>”, “status”: “<enum>”, “assignedTo”: “<num>”, “assignedFrom”: “<num>”, “comments”: “<comments>”, “dateCreated”: “<date>”, “priority”: “<enum>”}