CPSC 3720 FALL 2020

Jalapenos Issue Tracker

Team Jalapenos

Everett Blakley
Blake McLachlin
Steven Trinh

October 16th, 2020

TABLE OF CONTENTS

INTRODUCTION

Team Jalapeno is responsible for the development of an issue tracking system. An issue tracking system allows developers to effectively communicate with each other about bugs and issues in the software and provides a transparent interface for the project. It allows the software team to quickly detect and report bugs during all stages of the software lifecycle and allows the team to be more organized because reports will be easily accessible, filterable, discussable, and easy to understand.

As part of the project, Team Jalapeno has chosen the following optional attributes:

- Voting on issues
- Grouping users (developers, testers, managers)
- Keeping track of the reporter
- Search by multiple attributes

In this document, we will outline the key structure of our team, risk management, project proposal, the design of the system, and the project schedule.

PROJECT MANAGEMENT

TEAM ORGANIZATION

For the development of our issue tracker, the team structure is as follows:

Team Lead and Software Developer: Everett Blakley

Software Developer: Blake McLachlin

Software Developer: Steven Trinh

The team has decided to appoint Everett as our team lead because he has the most hands-on experience and skill as a software developer. As a result, he has a deeper understanding of the direction the team should move towards during all stages of the software lifecycle. All team members will contribute actively to the development of the project, with Everett acting as the scrum master, guiding the development along in an efficient manner.

TEAM WORKFLOW

The team will be utilizing the Gitflow Workflow model (Atlassian, 2020) (see Appendix A) to ensure seamless development processes:

- The **master** branch contains the official release history
- The **develop** branch serves as the integration point for new features.
- **Feature** branches (branched off **develop**) is where new features are implemented
- When a feature is complete, the feature branch is merged into **develop**
- At the end of a sprint, a **release** branch is created off **develop**, and then merged to **master**

There are more features to the Gitflow workflow. However, for this project, this amount of detail should be sufficient.

RISK MANAGEMENT

In this section, we will outline the potential risks the group may come across during the implementation of the issue tracker. With each risk, we will discuss the group's way of addressing the issue and provide a potential solution.

The team planned a project that is too large (i.e. "eyes bigger than stomach")

- In the case that the team underestimated the complexity of the project and milestones begin to become unrealistic, we will meet as a team to discuss the potential changes to be made to ensure competition before each deadline. A potential solution would be to dissect our project into minimum requirements (Planned), additional requirements (Refined) and optional requirements (Future). This will allow the team to understand the importance of certain elements and to redirect our focus on the big items, while removing resources from features that aren't necessarily needed.

The team underestimated how long parts of the project would take.

- During stand-up meetings, team members will identify if tasks are not going to be completed on time. If this is the case, the team lead will work with the developer responsible to resolve the issue. If more resources are needed to complete the task, another developer may be brought in to help complete the task. If the task needs to be broken down into smaller, more manageable tasks, the team will discuss how to best do that. In general, these instances will be dealt with in a timely manner, on a case-by-case basis.

Major changes to design are needed during implementation

- Change is to be expected during the project design and implementation because as a team, we must adapt to new changes in requirements and continuously monitor the situation to make the most optimal decisions. In the case of a major re-design, we will meet as a team to discuss the problems with the current design and determine potential workarounds or solutions. The team will analyze each potential design change in terms of its effectiveness, the amount of resources required to implement it, and the estimated time to complete the change. Hopefully, this will provide the team a new direction and motivate the team to keep pushing forward. The ideal solution for us as a team is to not get hung up on all the work that we have already completed but to think positive and keep morale high.

The team is not meeting the appropriate standards for testing and quality assurance.

- Since the team will be utilizing CI/CD to automatically test the application, quality will be easy to assess. At the outset of the project, the team will determine a minimum goal in terms of code coverage, with the expectation that every component of the application meet these requirements. If it becomes apparent that these goals are not being met, the team will meet and determine if the goals were too ambitious, or if it is the fault of the developers who are not meeting targets.

The team did not plan for risk-mitigation and as a result, the team had to compromise and deliver poor quality code.

- The team will discuss the development schedule before we dive into the project. This will allow the team to allocate a certain period before the deadline to ensure the code is meeting the requirements and progressing smoothly. Having a buffer zone before the deadline is important

to allow the team room to breathe and to fix potential and unforeseen issues that may occur. Risk mitigation is important to ensure that the team has the opportunity to deliver quality code.

Addition or loss of team member (i.e. someone dropped the courses, a new person joins the team)

- The addition or loss of a team member is a significant issue that needs to be addressed immediately by the team. During the meeting, we will discuss how to properly reallocate our team's resources and tasks to ensure the project can be completed on time. In the case of the loss of a member, we will have to reallocate the missing person's tasks to the other members. In the case of the addition of a team member, we will need to reallocate an existing team member to get them up to speed concerning the requirements, design, and the current stage of our implementation. However, at the same time, we will need to ensure that the project remains on track during the redistribution of resources.

Unproductive team member(s)

- We will meet as a team to discuss the potential issues that cause the unproductivity of a team member such as personal issues, gaming addiction, lack of experience, not understanding the task or requiring more guidance. It is important for us a team to understand the context of the unproductivity before making rash decisions to maintain a positive team environment.

Team member(s) lacking expected background

- Our team believes in the importance of helping others, especially fellow team members. If a team member is lacking the experience or skill to complete their task, the rest of the team will offer guidance and if necessary, we will delegate that team member to a simpler task to help build their confidence. Everything is a learning experience, and we want to build a positive team environment. This is also a good opportunity to practice paired programming to help learn a new skill.

Illness or unanticipated life event (e.g. death of family member)

- The team will meet and discuss with the affected team member about what they are able to handle and consequently, reallocate our resources accordingly. The team understands that unexpected things may occur and if life hits you, sometimes you must respond and put your focus elsewhere. We will monitor the situation closely and adapt to changes as they occur because we are a team built on spice and everything nice.

Inexperience with new tools

- In the case that new tools are introduced, the team will meet and determine if the tool is worth the investment of resources. If so, we will dedicate time to understanding the workings of the tool(s). If something is unclear to another member, the team will work together to find solutions and answers to questions.

Learning curve for tools steeper than expected

- If the tool has not become an integral part of our development process, we will search for less complex alternatives or tools that the team is already experienced with. We do not want to waste time learning a tool that is too complex and can potentially over-complicate the whole process, leading to multiple headaches. If we are already too invested with the tool, the team

will have to live with the choice and study the tool references diligently. We may have to re-evaluate the cost-benefit of keeping the tool(s) versus starting over with a new tool.

Tools don't work together in an integrated way

- Similar to the above, we will need to meet as a team and re-evaluate the design of our project to find alternative tools that will work together seamlessly.

SOFTWARE DESIGN

In this section, we will outline the main features of our issue tracking system and dive deeper into the design of our project as well as a rough draft of our project schedule.

PROPOSAL

ADDITIONAL FEATURES

The team has decided on the following additional features:

- Search for issues by multiple attributes (selected searching method)
- Voting on Issues (selected additional attribute)
- Reporter Tracking (selected additional attribute)
- Grouping Users (selected additional attribute)

USER STORIES

MINIMUM SCOPE

At the bare minimum, all users will be able to perform the following actions:

- As a User, I can sign up, using my name and email
- As a User, I can "login" to the issue tracker
- As a User, I can edit my name and email on my profile
- As a User, I can delete my account
- As a User, I can create issues, and act as the Reporter of the issue
- As a User, I can update and delete issues that I created
- As a User, I can create comments on issues
- As a User, I can update or delete comments that I created
- As a User, I can vote on issues so that the team is able to communicate the importance placed on each issue.
- As a User, I can search for issues using multiple attributes so that it is easy to find the information I need

Our group has decided to add user group attributes as well. The groups will be Developer, Tester, and Manager. Thus, for each of those groups, the users will be able to perform the following actions:

- As a Developer, I can assign myself to an issue that I created
- As a Developer, I can change the status of an issue that I created
- As a Tester or a Manager, I can assign a User to any issue
- As a Tester or a Manager, I can change the status of any issue

- As a Manager, I can change other Users' user group attribute (to developer, tester, or manager)
- As a Manager, I can update and delete other Users' issues
- As a Manager, I can update and delete other Users' comments
- As a Manager, I can delete other Users' profiles

FUTURE SCOPE

Time permitting, the team will try to implement the following features:

- As a User, I can subscribe to certain issues and be notified of changes to those issues upon logging in
- As a User, I can tag other users in comments so that it is easier to communicate with those specific users regarding the content of that issue and/or a specific comment.

DESIGN

The following table displays all the REST endpoints for the application.

Use Case	REST endpoint	JSON Response
Get all issues	`GET /issues/`	`{"status": 200, "data": [<issue>]}`
Get an issue by id	`GET /issues/:id`	`{"status": 200, "data": "<issue>"}`
Create an issue	`POST /issues/` `Body: <issue>`	`{"status": 200}` `Or {"status": 400, "message": "<error message>"}`
Update an issue	`PUT /issues/:id` `Body: <issue>`	`{"status": 200}` `Or {"status": 400, "message": "<error message>"}`
Delete an issue	`DELETE /issues/:id`	`{"status": 200}` `Or {"status": 400, "message": "<error message>"}`
Get all users	`GET /users/`	`{"status": 200, "data": [<user>]}`
Get a user by id	`GET /users/:id`	`{"status": 200, "data": <issue>}`
Create a user	`POST /users/` `Body: <user>`	`{"status": 200}` `Or {"status": 400, "message": "<error message>"}`
Update a user	`PUT /users/` `Body: <user>`	`{"status": 200}` `Or {"status": 400, "message": "<error message>"}`
Delete a user	`DELETE /users/:id`	`{"status": 200}` `Or {"status": 400, "message": "<error message>"}`

Get comments for an issue	`GET /issues/:issueId/comments`	`{"status": 200, "data": [<comment>]}`
Get a comment for an issue by id	`GET /issues/:issueId/comments/:id`	`{"status": 200, "data": <comment>}`
Create a comment	`POST /issues/:issueId/comments Body: <comment>`	`{"status": 200}` Or `{"status": 400, "message": "<error message>"}`
Update a comment	`PUT /issues/:issueId/comments/:id Body: <comment>`	`{"status": 200}` Or `{"status": 400, "message": "<error message>"}`
Delete a comment	`DELETE /issues/:issueId/comments/:id`	`{"status": 200}` Or `{"status": 400, "message": "<error message>"}`
Query issues based on query parameter	`GET /issues?<paramName>=<value>`	`{"status": 200, "data": [<issue>]}` Or `{"status": 400, "message": "<error message>"}`

DESIGN RATIONALE

Following this REST endpoint naming conventions (restfulapi.net, 2020) guide, the team has decided to treat each of the three main resources (issues, comments, and users) as a single entity or collection of entities, which our REST endpoints respond to upon HTTP requests from the client. This program will make use of multiple HTTP request methods (GET, POST, PUT, and DELETE) to interact with the backend service, passing data objects in the request/response body.

For all endpoints that use the GET method, if no resource is found, the service will return an error. Otherwise, it will return the resource(s). For PUT, POST, and DELETE requests, if the request is invalid (i.e. the object was missing required fields), the service will return a descriptive error message. Otherwise, the service will return a success message with an empty body.

In addition, when it comes to querying issues, we will use query parameters on the GET issues/ endpoint and return all issues that satisfy the query. If the query is valid, but no issues meet the criteria, the response will be empty. If the query is invalid (i.e. contains a parameter that does not exist), the service will return an error.

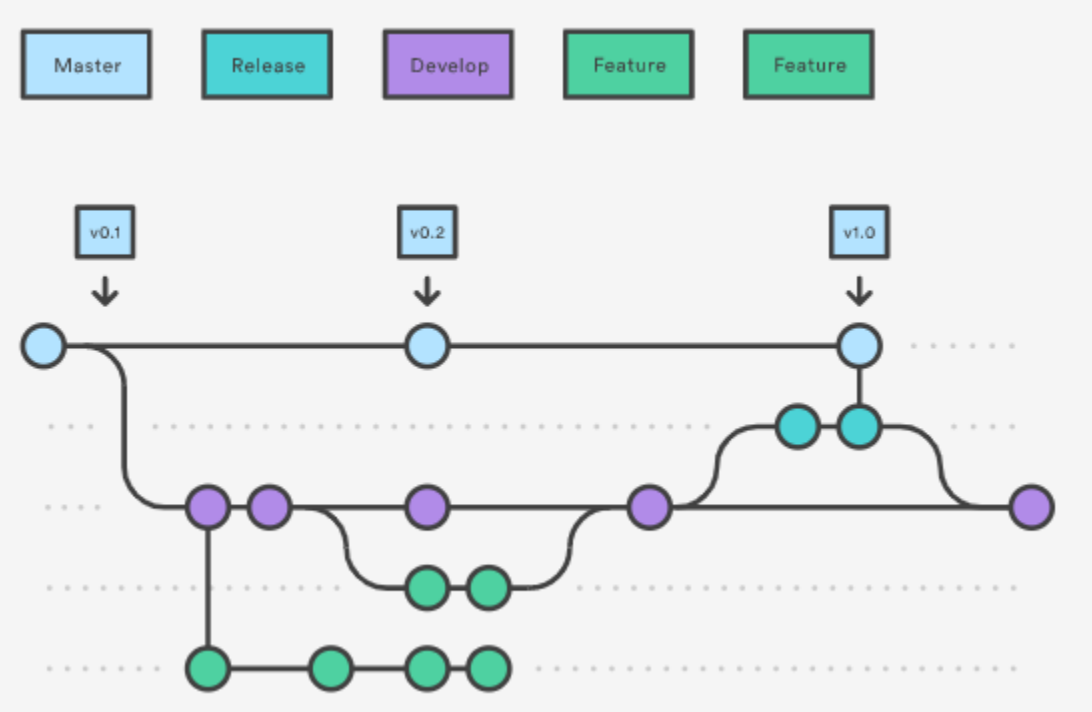
PROJECT SCHEDULE

Sprint 1	Sprint 2	Sprint 3
• Get the development environment set up for each team member	• Get, Create, Update and Read Issues • Get, Create, Update and Read Comments	• Incorporate NCurses • Finalize testing and documentation

• Learn the C++ *restbed* framework via reading documentation and making small test applications • Create sample JSON data • Get the back-end setup (I.e. try getting a simple get request working.) • Get, Create, Update and Read Users	• Get, Create, Update and Read Issues • Working on the CLI • Implement selected attributes/searching method • Data association (e.g. user **assigned** to an issue)	• Implement Refined/Future features

APPENDICES

APPENDIX A: GITFLOW WORKFLOW



WORKS CITED

Atlassian. (2020). *Gitflow Workflow*. Retrieved from Bitbucket Tutorials: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

restfulapi.net. (2020, 10 16). *REST Resource Naming Guide*. Retrieved from REST API Tutorial: https://restfulapi.net/resource-naming/