University of Lethbridge

3720 group project proposal

Issue Tracker Express

Team Aegir

Lorenzo Conrad
Mathew Richards
Steven Deutekom

October 7, 2019

Contents

List of Figures

List of Tables

1 Introduction

The goal of this project is to build an issue tracking system. An issue tracking system is a list of issues, or tickets, that can be dynamically modified, deleted and changed as needed by several different users. Organizations employ issue tracking systems to simplify their workflow and prioritize which issues need to be addressed first.

Issue tracking systems are essential for the smooth operation of many large projects. As problems arise they can be added to an issue tracking system so that everyone is aware that there is a problem. This helps to keep everyone informed about the state of the project and for critical needs to be addressed quickly.

For example, the University of Lethbridge maintains an issue tracking system in their IT Department. Any time a University student or employee has an issue or needs a service performed by the IT Department, they can submit a ticket. This ticket is then assigned a priority and assigned to a certain IT Services employee. The employee updates the ticket as their work progresses. When the work is done the issue is marked as closed. This is a typical workflow used with most issue tracking systems.

This proposal include several different sections. First, the specific service that will be built is described. This include a description of the general workings of the system and a list of its features. The next section will provide project management details. Outlines of the processes and project organization, team composition, risk management, and project schedule will be given. Finally, an initial API design will be given. It will list the request and associated endpoints that will be available with the service.

2 Proposed Service

We are proposing to build a RESTful issue tracking service called Issue Tracker Express. This section will examine what functionality will be available with the service. It will also detail some aspects of the system's implementation and identify some of the tools and dependencies. The features proposed as additions to the specification are listed in **RED**.

2.1 Issues

Issue Tracker Express main purpose is to allow users to log new issues with a specific title, status, and description. Issues will support updating their description, status, and priority. Users will also be able to comment on issues. It will be possible to generate list of all current issues. This will allow developers and project stakeholders to track and the progress that is being made on each issue. Issues will also support

priorities and tags so that issues can be organized more effectively. When listing issues it will be possible to filter them by priority, tag, or user. It will also be possible to search for a keyword or phrase in issue titles.

Proposed Features

- Create a new issue
- Get an issue by its unique identifier
- List all current issues
- Update an issue
- Delete an issue
- Set issue title
- Update an issue's title
- Set an issue's status
- Update the status of an issue
- **Issue priority**
- **Issue tagging**
- **Issue filtering**

2.2 Users

The system will also support unique user profiles. This will help to keep track of all the people that are involved in a project. These profiles will be simple, but offer personalization through user descriptions and a selection of avatars. Profiles will be easy to create, delete, and update allowing for a flexible system as people come and go from the project over time.

Proposed Features

- Add a new user
- List all current users
- Delete a user
- **Profile blurb and avatar**
- **Edit a user profile**

2.3 Comments

Commenting on issues will also be an important part of the service. This will allow those involved to add thoughts and updates to issues as progress is made toward a solution. It will be possible to create, delete, and update comments. When viewing an issue it will be possible to list all of its comments so users can stay up to date.

Proposed Features

- Add to an issue
- Get a comment from an issue
- Get all of an issue's comments
- Update a comment of an issue
- Delete a comment from an issue

2.4 Implementation Details

The back end for the service will be written in C++. It will use the **Restbed** framework (footnote) for the RESTful system. To handle JSON in requests and responses the **nlohmman/json** library (footnote) will be used. These two libraries will be the only back end dependencies for the project.

Testing will be automated using googles *gtest* and *gmock* frameworks. test coverage will be checked with *gcov*. Static analysis will be done with *cppunit* and style checking with *cpplint*. Memory usage will be checked using *valgrind*. The project will be maintained on *Gitlab* and will use a continuous integration server to ensure the integrity of the build.

There will be a simple front end client implemented as well. This will demonstrate some of the usage and possibilities for Issue Tracker Express. We will use html and a small amount of javascript to make this work, but to keep it simple. An example image is available in appendix A.

3 Project Management

This will be a relatively large project to undertake. This section details some of the practices that will be used to ensure that the team is successful. These practices will also be important for learning how software is developed in a professional setting. Each member of the team will be given specific responsibilities to make sure this run smoothly. Possible risks that may slow down progress or cause portions of the project to

go unfinished are also listed below. Finally, a tentative breakdown of the 3 sprints that the project will be completed in is given.

3.1 Processes and Organization

The team will follow *Agile* practices for this project. Using the list of features above a product backlog will be created. From this backlog items will be taken to form three two week sprints. The goal at the end of each sprint will be to have a working project. These sprints will be natural measuring points to assess productivity. In the last sprint, once all core functionality has been implemented, refactoring and improving the prototype UI will take place. Throughout the process TDD will be employed to make sure that the team can be confident in the working order of system components.

The team will also engage in weekly meetings to help measure productivity and discover areas that need more or less attention. After each sprint a retrospective will be done to see what can be done better. If needed adjustments to the upcoming sprint backlog will be made. The retrospectives will be documented to keep track of the project and be used when reflecting on current successes and failures. With continuous feedback and reflection from the team the project can evolve and adapt to deliver a successful and high quality product that meets the need of its users.

3.2 Roles and Responsibilities

In organizing this project we defined roles for team members. Steven will act as the scrum master for the group. He will be responsible for making sure that the sprint retrospectives are done, as well as to make sure that agile practices are being followed to the best of the teams abilities. He will also be managing pull requests and maintaining the build. Lorenzo has considerable experience with web technologies and he will be responsible for the prototype client UI. He will also be instrumental in ensuring a stable and useful UI. Mathew has good instincts for usability and features. He will focus on ensuring the service has what it needs to be useful to users. All group members will be contributing equally to developing the system. This will include coding, testing, and documentation. There will also be constant communication to ensure that the team remains on track and all members will motivate one another and help out when issues need to be resolved.

3.3 Risk Management

All three team members have worked on group projects. They understand the risks involved when working on a project of this size during a full time semester. Any difficulties that arise could result in schedules not being met or features not being completed.

There are potential obstacles due to the nature of a student semester. If a team member were to get sick during a sprint additional effort and coordination from the others will be required. If necessary we are all familiar with using secure shell apps to work remotely.

With all the new technologies being used there is a learning curve for Lorenzo and Mathew who have less experience with larger C++ projects. They are both committed to learning to use the new tools and seek help when needed. Since Steve has more experience with these type of projects he will have to be ready to guide the other team members when they need it. Working closely and constantly communicating will be the best way for the team to make sure that new tools and technologies do not hurt productivity or quality.

Some additional risks are present with our individual team members. There is a risk that Lorenzo will get too carried away with the front-end UI. There is also risk Steve will strive to "gold-plate" the project. He is also a very talkative person and occasionally needs to be reminded to focus. Mat is sometimes easily distracted by recreational interests so Steve and Lorenzo must sometimes remind him to return to the task at hand (no reddit or Star Wars!). To mitigate these risks all team members must work together to keep attention where it needs to be. They must also strive to prioritize the core components of the system and make sure they stick to the schedule.

3.4 Proposed Schedule

The team will have weekly stand up meetings every Monday at 2 pm. Sprints will occur in two week segments beginning when the proposal phase of the project is finished.

Sprint one will be for getting the dependencies working together and implementing a few important features. The goal will not be to get a lot of features implemented as the team will need time to get familiar with the tools being used. Once a couple features are working and the system seems stable then work on features can begin. The prototype UI will begin its implementation in this phase, but will remain as simple as possible. The sprint backlog for this phase will be conservative, but if time permits some features will be moved from sprint two.

Sprint Two will be where the bulk of the service is implemented. Many of the features will not require much to add to the system once it is set up. During this phase it will be a priority to implement as many of

the main features as is possible. The hope will be to have most of the specification completed in this sprint. However, it may not be possible so anything that needs to be will be moved into sprint three.

Sprint 3 will be the home stretch and at this point the team should be very familiar with the processes and tools that are being used. The last few extra features will be implemented here. During this phase the UI will be completed and enhanced if there is time. The user documentation will also be completed as the system should be relatively well understood at this point. Also, if it is possible the team may identify some areas of the code that could be refactored to deliver a higher quality product. At the end of the sprint the release should contain a fully working back end and a useful front end to demonstrate the working of the service.

The nature of Agile software development is to try and always keep some working release so that it can be used to guide future development. Agile is also about being flexible and adaptable. As time goes on our releases and previous work will help to guide the future processes. While the proposed schedule and sprint backlogs (in appendix B) will be followed as much as possible the team will not hesitate to make adjustments if needed.

4 Design

This section contains an explanation of the general response structure. It also lists possible RESTful endpoints and their parameters. The goal is to have a relatively stable and intuitive API for the service before work gets to far along on the systems. Hopefully, during the first sprint any issues with the current design will be worked out. Some examples of the JSON responses from some of the methods are listed in appendix C.

4.1 Responses

The general structure of a response will be to contain a status element. This element will indicate whether the request was successful. If it was successful the status will be "ok". However, if not successful the status will be "failed". If status is "ok" then the response will also contain a field called "contents". This field will have a JSON object with the necessary information in it. It may be a single object or a list of objects depending on the particular request. If the status is "failed" then the response will contain a field called "error". The error will be a JSON object with an error code and a message. This way the caller can easily determine the error from the code, but also have a message that gives a more human friendly error message.

These error codes will be detailed in the user documentation once the team is more aware of what issues might be encountered when handling requests.

4.2 API

Service	Method	Endpoint	Parameters
Create/update issue	POST	/trackexp/issue	title, description, priority, assignee, tags
Get Issue(s)	GET	/trackexp/issues	id, tags, keyword, status
Remove Issue	DELETE	/trackepr/rm_issue	id
Create/update user	POST	/trackexp/user	id,name,blurb, picture
Get users(s)	GET	/trackexp/users	id,name
Remove User	DELETE	/trackepr/rm_user	id
Create/update comment	POST	/trackexp/comment	issueId, text
Get comments	GET	/trackexpr/comments	issueId, id
Remove Comment	DELETE	/trackepr/rm_comment	id

Table 1: Service Endpoints

Appendices

A Client Examples

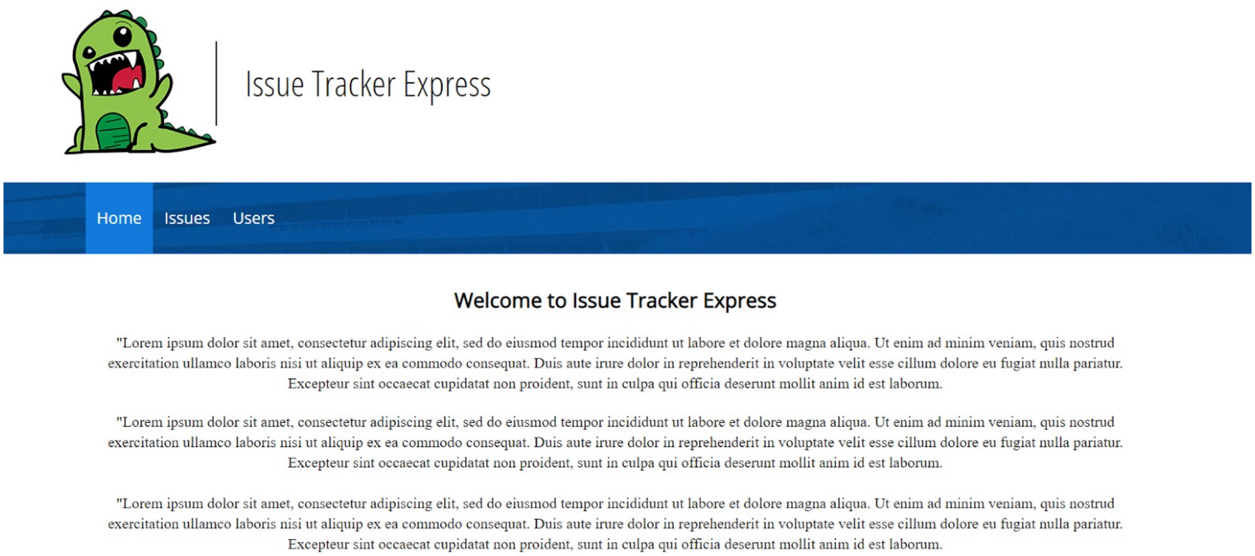

Figure 1: Home Page and General Layout

B Sprint Breakdowns

Sprint 1

- Core system with Restbed and nlomman/json
- Issues: createIssues, getIssues, listAll
- Users: addUsers
- Basic web interface

Sprint 2

- Issues: update, delete, setTitle, updateTitle, setStatus, updateStatus, updatePriority, searchTitle, addTag, removeTag, filter
- Users: listAllUsers, deleteUsers , editProfile

Sprint 3

- Comments:
- Documentation: Service manual, Prototype front end instructions
- Final release organization
- Refactoring

C Response Objects

There are some sample responses for issues and an error. The other responses for users and comments will be very similar except with slightly different fields.

- Get Issue

```
{
    "status": "ok",
    "content":
        {
            "id": 1234,
            "title": "Issue title",
            "description": "Issue description.",
            "status": "open",
            "priority": "high",
            "assignee":
                {
                    "id": 293023,
                    "name": "A user",
                },
            "tags": ["OS", "Critical"],
        }
}
```

- List of Issues

```
{
    "status": "ok",
    "content":
        [{
            "id": 1234,
            "title": "Issue title",
            "description": "Issue description.",
            "status": "open",
            "priority": "high",
            "assignee":
                {
                    "id": 293023,
                    "name": "A user",
                },
            "tags": ["OS", "Critical"],
        },
        {
            "id": 83420,
            "title": "Issue title2",
            "description": "Issue description.",
            "status": "open",
            "priority": "low",
            "assignee":
                {
```

```
                    "id": 79879,
                    "name": "Another user",
                },
            "tags": ["UI", "Colors"],
        }]
}
```

- Remove Issue

```
{
    "status": "ok",
    "content":
        {
            "id": 1234,
        }
}
```

- Error

```
{
    "status": "ok",
    "content":
        {
            "code": 4,
            "error": "some error message that explains the error better.",
        }
}
```