Issue Tracking System Project

1 Overview

In this project your team will create a web-based issue tracking system.

2 Objectives

Some of the objectives of this project are:

- Work collaboratively as a team to build a useful real-world product.
- Gain experience in service-oriented software engineering.
- Follow an agile methodology to develop a software product.
- Use good software engineering practices to develop software.

3 Project

The goal of the project is to create a web-based issue tracking system.

3.1 Minimal Requirements

The system will provide (at minimum) the following functionality:

- Issues
    - Create
    - Get
    - List all
    - Assign
    - Update
    - Delete
    - Set title
    - Update title
    - Set status
    - Update status
- Comments
    - 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
- Users
    - Create / Add

- List all
- Remove a user

Your team is to add 4 additional features of the team's choosing. You can do more than this if you want to and have time.

Choose:

- 3 features from *Attributes*
- 1 feature from *Searching*

Attributes	Searching
• Component – what system component does the issue affect? • OS – what operating system does the issue affect? • Priority – how important is the issue? • Reporter – who reported the issue? • Voting – how important is this issue to the community? • Type – what kind of issue is it (Feature? Bug? Task?) • User can select a profile picture from a fixed set of images. • Users can be grouped (e.g. developers/testers/managers)	• Find using a single attribute (e.g. all issues for Windows 10) • Find using two attributes (e.g. all high priority items of type "Feature") • Find by any number of attributes.

If your team wants to do a different feature, please contact Dr. Anvik for approval. Most likely you will be able to do it, however this check is to help students avoid features shown to be more work than is expected for the project. You can get ideas for features by looking at:

- Bugzilla REST API (https://wiki.mozilla.org/Bugzilla:REST_API)
- Jira REST API (https://docs.atlassian.com/software/jira/docs/api/REST/latest/)

The following items are not recommended as they could become a functionality quagmire from which your team may sink into and never be seen again, beyond the scope of the project, or not worth the effort for what benefit they will provide.

- User authentication (i.e. user names and passwords)
- Attachments
- Use of encryption (military-grade or otherwise) of data
- Use of Javascript User Interface libraries (e.g. React, AngualrJ, or VueX).

3.2 Issue Fields

An issue is to have the following fields (at minimum):

- Identification number
- Title
- Description (this will be Comment #1)
- Status (one of `New`, `Assigned`, `Fixed`, `WontFix`)
- AssignedTo (a user id)
- Comments

3.3 Interface

Create a **simple** and **useable** console interface for your application.

If you would like to use a library such as `ncurses`, that is acceptable as long as using the library does not require updates to the CS department's environment.

4 Constraints

4.1 Design Constraints

1. **REST service.** The functionality of the web service will be provided as REST endpoints.
2. **JSON Data Format.** The data format for messages is to be `JSON (JavaScript Object Notation).`

4.2 Process Constraints

1. **Agile for development process.** The project is to be completed using Agile practices as discussed in class. There should be evidence of:
    a. ***Standup meetings.*** These need not be every day, but should be regular. Show this to the marker by having a team member (Scrum Master) post a brief summary in the notebook of the team's MS Teams channel to show this is happening.
    b. ***Kanban.*** Use a board on GitLab to show your project/sprint backlogs and progress.
    c. ***Iterative development.*** This will be demonstrated through sprints, merged branches / merged pull requests and commits.
    d. ***Continuous integration.*** The team will create a `Makefile` and GitLab CI configuration file that:
        i. Compiles the project
        ii. Runs unit & integration tests
        iii. Runs static analysis
        iv. Runs memory leak testing (only for the "service" classes, **not** the client or server classes which will hang the CI server)
        v. Runs style checking
        vi. Generates source documentation
    e. ***Release planning.*** Each sprint must end with a "shippable" product. This will be indicated with version tags in the repository:

i. *v.*1.0 – the project proposal document.
ii. *v.*1.1 – the product for Sprint #1.
iii. *v.* 1.2 – the product for Sprint #2.
iv. *v.* 1.2 – the completed project at the end Sprint #3.

f. ***Coding standards***. These will be enforced using `cpplint.py`. You can choose which rules to not run, but try to have as many rules run as possible.

2. **`C++` for the programming language.** You will need to use g++9.1 to compile with the `restbed` library.
3. **`restbed` for the REST application framework.** The framework is already installed in the labs.
4. **Header-only JSON library.** Use the `nlohmann/json` (https://github.com/nlohmann/json) header-only `JSON` library
5. **`GitLab` for version control, issue tracking, continuous integration and project planning/management.** A private Git repository will be created for each team to use during the project.
6. **`MS Teams` for communication.** A private channel will be created for each team to communicate. The marker will examine this from time for evidence of your software development process in action.
7. **PDF for reports.** All written documentation (i.e. reports, user manual, and endpoint documentation) will be provided in `PDF` format in a folder called `docs` in the project's repository.
   a. Sprint retrospective reports have filename `Retrospective-Sprint<#>.pdf` where `<#>` is the sprint number.
   b. Web service documentation has filename `REST_API.pdf`
   c. User manual has filename `User_Manual.pdf`
8. **`doxygen` for generating source code documentation.** The generated files are to be found in a `docs/code` directory.
9. **`cpplint.py` for style checking.** You can choose which rules to not run, but try to have as many rules run as possible.
10. **`gcov` for code coverage.** This need only be on the "service" classes which are unit tested. You will need to use `gcov9.1`. Also `lcov` does not work with `g++9.1`.
11. **`cppcheck` for static analysis.**

5 PROJECT DELIVERABLES

5.1 PROJECT PROPOSAL

Submit a proposal for the REST service that your team is planning to build. A proposal template is provided on Slack for you. See the appendix for details.

5.2 Sprint Review

Your team will provide a video presentation to the class at the end of each sprint. The video will be uploaded onto the course's Notebook to a page in the Collaboration→Project section. The presentation is to contain (as indicated):

	Sprint 1	*Sprint 2*	*Sprint 3*	*Sprint 4*
An introduction of the team	✓	✓	✓	✓
An overview of the project's design	✓	✓	✓	✓
An explanation of the additional features your team has chosen	✓	✗	✗	✗
What your sprint plan was	✗	✓	✓	✓
How close you came to achieving your plan	✗	✓	✓	✓
What problems you encountered and how you solved them	✗	✓	✓	✓
Your plan for the next sprint	✓	✓	✓	✗
What you will you do differently in the next sprint	✓	✓	✓	✗
A demonstration of the working software	✗	✓	✓	✓

5.3 Sprint Retrospectives

Complete three 2-week sprints. Following each sprint you will submit a sprint retrospective report that has detailed answers to the following questions:

a. What was your sprint plan?
b. How close did you come to achieving your plan?
c. What problems did you encounter and how did you solve them?
d. What is your plan for the next sprint, if there is one?
e. What will you do differently in the next sprint, if there is one?

5.4 Web Service Documentation

This document specifies the REST endpoints for your web service. The document must be logically organized and the end points described in such a manner that a web developer that wants to use your service will know how to use the service and what to expect as output from the endpoints.

5.5 User Manual

The user manual is a guide for using the simple web interface you created for your web service. The manual must be logically organized so a user knows how to use your application. Include screenshots as much as possible to guide the user.

6 Grading

Your project will be graded based on:

- Proposal & Sprint retrospective reports **[20% of project grade]**
  - Quality of writing.
  - Detail of answers to Scrum questions and retrospective.
- Sprint presentations **[20% of project grade]**
  - Quality of presentation.
- A working web service at the end of each sprint **[30% of the project grade]**
- Web service documentation. **[5% of project grade]**
- User manual. **[5% of project grade]**
- Evidence of the use of good software engineering practices (e.g. agile development, version control, unit testing, static analysis, continuous integration) throughout the project (See Process Constraints for details) **[20% of the project grade]**

7 Getting Started

An example REST-service is provided as a starting point for you. The REST service provides the functionality of a simple calculator (add, subtract, multiply, divide). It can be found at http://gitlab.cs.uleth.ca/course3720/examples/Examples/CalcREST. Use the example as a starting point for building your own REST service.

APPENDIX: PROJECT PROPOSAL

The project proposal document will contain the following:

1. **Title Page** showing:
    a. the name of the issue tracking system,
    b. team name and logo,
    c. the team letter on Moodle
    d. names of team members (only list those that contributed to the design of the project and the report)
    e. due date
2. **Table of Contents.** It should be on its own page.

3. **Introduction**. The introduction provides an overview of the entire document. A person should be able to get a clear idea of what the project is about from this section. The introduction explains what an issue tracking system does and why yours will be the best ever (a.k.a. a work of creative fiction). The introduction ends with a preview of the major sections that follow.

4. **Proposal**. Describe the web service and application that your team is proposing to build. Give as much detail as possible, including:
    a. Prototype images of the web client interface (can be hand drawn).
    b. Use cases (i.e. product backlog items).
        i. As all projects will have the same base set of features, use a different font colour to quickly identify the features that are unique to your web service.

5. **Project Management.** Provide a description of how you will complete the project, and address any foreseeable problems. This section will have two subsections:
    a. ***Risk Management:*** Describe how the team will address foreseeable risks that could prevent the team from completing the project. Examples of risks include, but are not limited to:
        i. Requirements/Design/Estimation
            1. The team planned a project that is too large (i.e. “eyes bigger than stomach”).
            2. The team underestimated how long parts of the project would take.
            3. Major changes to design are needed during implementation.
        ii. People
            1. Addition or loss of team member (i.e. someone dropped the course, a new person joins the team)
            2. Unproductive team member(s)
            3. Team member(s) lacking expected background
            4. Illness or unanticipated life events (e.g. death of family member)
        iii. Learning & Tools
            1. Inexperience with new tools
            2. Learning curve for tools steeper than expected
            3. Tools don’t work together in an integrated way
    b. ***Project Schedule*****:** An initial project plan showing which features will be completed in which sprint. A table, set of tables, or bullet lists are acceptable.

6. **Design**. Provide the planned design of the REST endpoints. Indicate for each endpoint the use case that it services. For brevity, you can omit the protocol, hostname and port from the endpoints. For example:

Use Case	REST endpoint	JSON Response
Add two numbers	`/calc/add?first=<num>,second=<num>`	`{"result": "<num>"}`
Subtract two numbers	`/calc/sub?first=<num>,second=<num>`	`{"result": "<num>"}`
Multiply two numbers	`/calc/mult?first=<num>,second=<num>`	`{"result": "<num>"}`
Divide two numbers	`/calc/div?first=<num>,second=<num>`	`{"result": "<num>"}`
		`{"error": "Division by zero"}`

7. **Appendices.** Any figures or tables that are more than half of a page should be put in the appendices and reference in the report text. Omit this section if there is no content.