**CS673F16 Software Engineering**

**Team 1 - Project Name**

**Project Proposal and Planning**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Steve Black | Team Leader, Design Lead | *Steve Black* | 10/11/16 |
| Anthony Valencia | Configuration Lead | *Anthony Valencia* | 10/11/16 |
| Demetrius Glover | Environment and Integration Lead | *Demitrus Glover* | 10/09/16 |
| Yash Bavishi | Requirement Lead | *Yash Bavishi* | 10/11/16 |
| Kevin Ding | Implementation Leader | *Kevin Ding* | 10/11/16 |
| Kevin Sperling | QA Lead | *Kevin Sperling* | 10/11/16 |
| Bakyt Bakayeva | Security Lead | *Bakyt Bakayeva* | 10/11/16 |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| 1 | Steve Black, Yash Bavishi, Kevin Sperling, Anthony Valencia | 10/3/2016 | Initial Document |
| 2 | Steve Black | 12/12/2016 | Update Procedures and References |
|  |  |  |  |
|  |  |  |  |

[Overview](#_87t9hln2vjz0)

[Related Work](#_mps353x5ezyl)

[Detailed Description](#_fg3z0hpd4q9v)

[Management Plan](#_ds8oyr75pnh1)

[Process Model](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.27177f40uci)

[Risk Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.a4oqwntk3mw)

[Monitoring and Controlling Mechanism](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.ywdoc2clc9yt)

[Schedule and deadline](#_tadq5mb0pici)

[Quality Assurance Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.72e1f4uawy2r)

[Metrics](#_b2haznn3yyz2)

[Standard](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.vc72k6dweldv)

[Inspection/Review Process](#_f1c69ifi68h7)

[Testing](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.r5d5mhtlf0kq)

[Defect Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.54a4wuncjg1c)

[Process improvement process](#_jhct37ebxxpn)

[Configuration Management Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.hw41vg4ykxen)

[Configuration items and tools](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.bwlb4d4vdox2)

[code commit guidelines](#_yyauft6zr9hw)

[References](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.8mva2050iy7t)

[Glossary](#_ty3i2nqffhtc)

# Overview

The purpose of developing a REST API to the existing issue tracker software is to allow for automatic posting of new issues, issue updates, and issue messages via external software such as a chat client or via events like git commits. This will enhance the workflow of anyone using the issue tracker as part of the existing software suite, or in an integrated fashion with other programs. In our experience, issue trackers have been effective repositories for tracking ongoing testing issues, defects, and features; hence, by increasing the ability of end users to easily input information to the issue tracker, we hope to enhance their development efficiency.

# Related Work

Our implementation of issue tracker is similar to other software with this feature: It’s meant to be a tool to tackle issues that come with planning and implementation stage in projects. A communication tool that keeps track of bugs/issues that come up along in a project. That way the team is able to keep track of them and come up with a plan to remove them.

Many software exists on the market that takes varying approaches for the same problem.

Some specific examples of issue trackers currently in the market are:

1. Roundup (open-source Python implementation, backed by SQL RDBMS of choice)
2. JIRA (proprietary, developed by Atlassian)
3. Github (built-in issue tracker as add-on to version control services)

The difference is in implementation and presentation. Our focus is allowing storage of files and issues and the capability to reference them in the middle of conversation with our added chat feature- similar to embedded links you can do in conversation, the REST API will allow us to reference the files and issues we’ve uploaded in simple and easy to use way.

# Proposed High level Requirements

Existing Features:

1) Users can log in to the system if they have an account.

2) Users can create, assign, view, and edit issue

3) Issues can be sorted

4) Dashboard

New Features:

a. Functional Requirements

i. Essential Features

1) Create REST API for the application

2) Replace the existing chat client with a more tightly-integrated version

3) Chat client can send messages to the tracker to add comments to the issues via API

4) Chat client can create new issues via API

5) Chat client or other outside services can change the status (Open, Fixed, Closed) status of issues via API

6) Support for multiple databases. To Isolate and deploy the database on a separate server, using RDS.

ii. Desirable Features

1) Integration to pipe git messages into issue tracker

iii. Optional Features

1) Documents can be attached to comments and requirements tracker

b. Nonfunctional Requirements

Reserved before fully researching on existing project

# Management Plan

# (For more detail, please refer to SPMP document for encounter example)

## Process Model:

For this project Agile process model will be used. The project will be developed in 3 iterations (4 if required) of 3 weeks each. Each iteration will encompass a particular set of requirements to be fulfilled and any changes that are demanded by the team and/or Prof. Zhang. Agile process model will ensure all the team members will be actively involved in the development process while working on their lead roles and with it performing any and all general tasks assigned to them. Agile allows us to establish learning goals at the start of an iteration and gives us time to reflect upon the lessons learned before the start of the next iteration. The team shall use Paired Programming for the less experienced members.

## Objectives and Priorities:

The main objective of this project is to learn and apply good software engineering techniques on the development of an efficient RESTful API for the Issue Tracker. Priorities are subject to the iterations and the requirements / changes.

Priorities:

* Iteration 1:
  + Training the team for development in the new technologies.
  + Creating a Docker Container for ease of development and setting up the environment in everyone’s computer.
  + Delivering the iteration requirements.
* Iteration 2:
  + Using Paired Programming approach for the learning members of the team to ease the development process for them.
  + Delivering iteration requirements:
    - Adding comment to issue via API call.
    - Fixing previous issues requirements.
* Iteration 3:
  + Iteration requirements:
    - Modifying multiple fields of an issue via API call.
    - Enabling chat client to create new issues.
    - Enabling chat client to add comments to issues.
    - Enabling chat client to modify issue.
    - Attaching files to issues via comments.

## Risk Management (need update constantly):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | **Risk Title** | **Likelihood**  **1-10** | **Impact 1-10** | **Retirement Cost**  **1-10** | **Priority (11-L)\***  **(11-I)\*R** | **Mitigation Plan** |
| 1 | Limited technical experience in the technologies to be used. | 7 | 8 | 3 | 36 | Training of team members and use of online tutorials |
| 2 | Communication Failure | 7 | 5 | 5 | 120 | Regular communication over slack and in-person meetings |
| 3 | Delayed Delivery | 6 | 7 | 5 | 100 | Constant monitoring and updating schedule |
| 4 | Issues in integration due to different dev environments | 5 | 4 | 5 | 210 | Setting up Docker containers |

## Monitoring and Controlling Mechanism:

a. Progress shall be monitored over each weekly meeting and weekly reports shall be submitted by each member.

b. Team communication is facilitated through slack ‘#team-1’ channel.

c. Requirements shall be established at the start of each iteration and tracked on pivotal tracker.

d. Schedule monitored and updated each week on Google Calendar.

e. Use of pivotal tracker to monitor bugs.

In addition to the overall monitoring by the team leader, each team member is responsible for the progress of work under their own lead role as mentioned below:

* + Team lead, Design lead - Steve Black
  + Configuration lead - Anthony Valencia
  + Environment and Integration lead - Demetrius Glover
  + Requirement Lead - Yash Bavishi
  + Implementation Lead - Kevin Ding
  + QA lead - Kevin Sperling
  + Security Lead - Bakyt Bakayeva

## Schedule and deadlines (need update constantly):

* Initial project planning - 10/03/2016
  + Submission of SPPP
  + First presentation
* Iteration 1 - 10/24/2016
* Iteration 2 - 11/14/2016
* Iteration 3 - 12/05/2016

# Quality Assurance Plan

# (For more detail, please refer to SQAP document for encounter example)

## Metrics

* + 1. Quality metrics will be based off of unit test coverage, UI automation coverage, and number of issues/bugs.

Process metrics will be based on work points completed and hours worked via our weekly reports.

Product metrics be based on code complexity. The measurement tool is yet to be determined.

* + 1. Results (to be completed at the end of each iteration)
       1. [Process Metrics](https://docs.google.com/spreadsheets/d/1rcMobR85OhHuqMvl53P3JSatVs2Cs7kNnxxY3x6QZcM/edit#gid=1652354667)
       2. Quality Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Iteration 1** | | | | |
| **Unit Tests** | **UI Automation Tests** | **Manual testing** | **API Tests** | **Bugs** |
| - 0 | - 6 inoperable | - 0 | - 0 | - 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Iteration 2** | | | | |
| **Unit Tests** | **UI Automation Tests** | **Manual testing** | **API Tests** | **Bugs** |
| - 0 | - 4 inoperable  - 2 Passing | - 5 Passing - 1 Failing | - 0 | -1 Edit Issue Description (Fixed) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Iteration 3** | | | | |
| **Unit Tests** | **UI Automation Tests** | **Manual testing** | **API Tests** | **Bugs** |
| - 0 | - 5 Passing | - 15 Passing | - 13 Passing | - 1 Unable to Messages in chat(Fixed) |

## Standard

The coding standards we will be following will the be as defined by the code

Foundation. For example, for the python code, we will be following the python foundations own style guide found here: <https://www.python.org/dev/peps/pep-0008/>

## Inspection/Review Process

Code submission will be in the form of a pull request. At that point, the rest of the team will review the code and either allow it to be merged or specify changes that need to be made.

## Testing

Testing will consist of a combination of unit tests, UI automation using selenium and manual testing. Unit tests must created for new code and run before a pull request. Hopefully, we will be able to use a continuous integration tool like TravisCI so that we can have unit and UI automation run for each pull request. Test results will be documented in a shared spreadsheet.

## Defect Management

(e.g. describe the criteria of defect, also in terms of severity, extend, priority, etc. The tool used to management defect, actions or personnel for defect management)

A defect is anything that interferes with the expected result of the program. We can categorize the defects or bugs into three categories:

**High** - Anything that breaks the application or causes major functional issues that would severely impact the user - i.e., not being able to log into the application

**Medium** - Anything that interferes with users workflow but does not hinder the completion of the workflow. General annoyances with workarounds - i.e., a window is sized incorrectly but the user can manually resize it and use it.

**Low** - Anything that might be an edge case that will not impact the vast majority of users - i.e., having 100 windows open simultaneously causes the app to crash or the font doesn’t match the rest of the page.

We will be using Pivotal Tracker in order to manage bugs.

# Configuration Management Plan

(For more detail, please refer to SCMP document for encounter example)

## Configuration items and tools

**Django, git, Docker, RDS, EC2/ECS, API Gateway, Ansible**

Django will serve as the web development framework. The existing application is written in Django, so it will be easiest to maximize deliverables if the framework is kept.

The deployment environment will utilize Docker containers. Docker containers will help minimize setup environments as it applies to development and deployment. It will help enforce some consistency between the range of development operating systems. This requires removing virtualenv. Python virtual environments are dependent on the OS (e.g. pyodbc, cython, compilation differences). The virtualenv is mutable. While VMs get part of the way toward converging the development landscape, they are heavy and not much of a scalable deployment solution. Docker containers work well with many available scalable deployment services such as ECS. A small set of bash and batch scripts will be created for building and running containers for various dev environments.

We hoped to achieve the following items, but we opted to simplify the scope of the project:

1. Dedicated database server
   1. Another enhancement this project will hopefully deliver is to isolate and deploy the database on a separate server, using RDS. This makes enforcing access restrictions much more streamlined. Furthermore, RDS deployment comes with CloudWatch monitoring which will be invaluable for debugging database issues.
2. Configuration management with Ansible
   1. If time/scope permits, Ansible playbooks will be defined and used as a configuration management solution. Ansible has great functionality for controlling AWS. Another feature that would be nice to deliver if we manage to pack a year's worth of of work into 16 weeks is to utilizing the AWS API Gateway as the accessor of the data layer. Benefits include low cost, scalability, monitoring, streamlined dev (run multiple API versions).

## Change management and branch management

There will be two branches used by developers:

* **master**: the origin of all branches. Develop originated from master, and the cumulative changes made in the course of the project are merged back to master upon conclusion.
* **develop**: all developers will check out the develop branch, or fork their own repositories. Developers who work on develop directly will be responsible for making sure that they rebase their changes on top of the current HEAD of develop (to avoid merge conflicts). Developers who fork their own repositories will submit pull requests to Steve (who manages the main repository), and he will merge their changes into develop.

## Code commit guidelines

* Everyone clones the central repository (**bebingando/cs673f16-team1.git**) to obtain the source code.
* Fork repository to work on individual features.
* Before committing to origin master, rebase to pull any upstream changes. When rebasing, each developer is responsible for their own merge resolution. Assistance may be sought with any developers who have made intermediary commits.
* Each commit must be accompanied by a commit message. The commit message must have the name of the issue/task if one is associated with the commit.
* Developers are expected to update their user story issues in Pivotal tracker with details about any departures from the existing requirement, and notes about testing the features.

# References

* 1. Best Practices for Designing a Pragmatic RESTful API
     1. http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api
  2. Docker
     1. https://docs.docker.com/
  3. Django Viewsets
     1. http://www.django-rest-framework.org/api-guide/viewsets/

# Glossary

* 1. API
     1. Application Programming Interface
  2. REST
     1. REpresentational State Transfer
  3. JSON
     1. JavaScript Object Notation