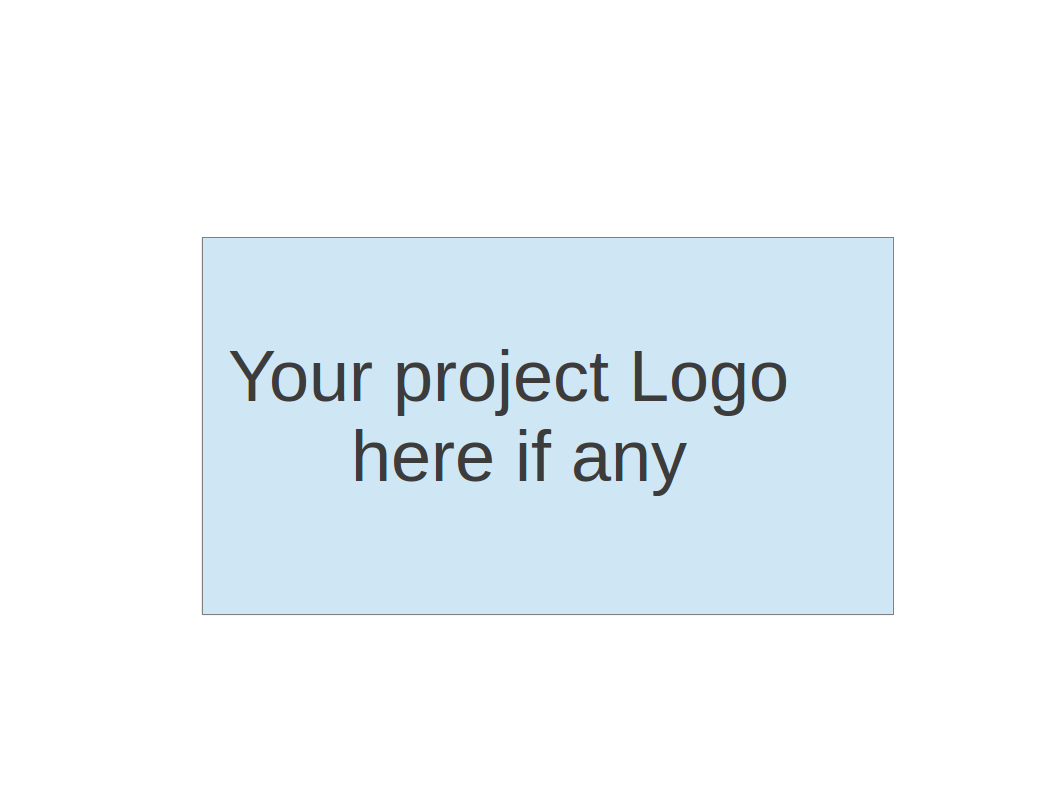
**CS673S16 Software Engineering** 

**Team X - Project Name**

**Software Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Oliver Nowak | Team Lead | O No | 1/21/2017 |
| Son Nguyen |  | Sn | 1/21/2017 |
| Emily Birmingham |  | EB | 1/21/2017 |
| William Baldoumas | Sub-Team Lead, Environment & Integration Leader | WB | 1/21/2017 |
| Hanhan(Michael) Song |  | HS | 1/21/2017 |
| Mike Trinh |  | MT | 1/21/2017 |
| Ashutosh Singh |  | AS | 1/21/2017 |
| Raziel Melchor |  | RM | 3/12/2017 |
| Betsy Berkey |  | BB | 3/12/2017 |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
|  |  |  |  |
|  |  |  |  |

[Introduction](#_gjdgxs)

[Software Architecture](#_30j0zll)

[Design Patterns](#_1fob9te)

[Key Algorithms](#_3znysh7)

[Classes and Methods](#_2et92p0)

[References](#_tyjcwt)

[Glossary](#_3dy6vkm)

# Introduction (Subteam 3.1)

The Project Management Tool suite consists of a Requirements tracker, Issue tracker and Communication tool.

The communication tool is a web-based application that provides a centralized location for team communication. It allows users to send instant messages to other users of the application, in an isolated context called a “Room”.

This document describes the implementation and framework for building the Communication tool. This document includes software architecture, design patterns and sub systems.

# Software Architecture (Oliver)

* Components:
  + Communication Tool
    - Uses websockets for real-time communication
    - Uses nodeJS/jQuery for dynamic content
* Database Technologies
  + SQLite (local datastore)
  + MySQL (server datastore)
* MVC dependencies
  + NodeJS
    - API calls from client
  + jQuery
    - Client-side Data operations
  + Socket.io
    - Websocket interface for real-time chat
  + Django Web Framework
    - ORM, REST endpoints, HTML view templating
* Architecture and Control Flow
  + <https://github.com/CS673S17-Team-3/Final_Project/blob/develop/ARCHITECTURE_OVERVIEW.md>
* File System Overview
  + <https://github.com/CS673S17-Team-3/Final_Project/blob/develop/FILE_SYSTEM_OVERVIEW.md>
* Future Plans
  + Continuous Integration
  + Documentation
  + Repo Cleanup
  + Simple features centered around Room / Chat management

# Design Patterns (Subteam 3.1)

In this section, you shall describe any design patterns used in your software system.

* The Model-View-Controller (MVC) pattern consists of three separate components that allow a software’s users to modify and access the software’s information. Models contain the data and logic for the app. Views are the visual representation of the models. Finally, controllers act as the medium between the two, allowing users to update the model and the model to update the view.
* Django uses the MVC pattern in a slightly different way. It is often referred to as an MVT framework because it consists of a model, a view, and a template. The model in Django acts the same as a normal MVC. The controller in Django is referred to as the view. The view is the Python callback function for the particular URL. This function describes which data is presented in the template.
* In the Communication Tool, the model contains the room, message and userroom models that are stored in the database. Users can see and use the template, which interacts with the model (creating teams, sending messages) by sending URLs that are handled by the view.

# Key Algorithms (Subteam 3-A)

In this section, you shall describe any key algorithms used in your software system, either in terms of pseudocode or flowchart.

(We either need to add new features or improve old code to incorporate legitimate algorithms.)

* (People you may know?) DFS/Disjoint Datastructure
  + Based on common room presence
* We can have encrypting and decrypting for nodejs to enhance security (<http://lollyrock.com/articles/nodejs-encryption/>) or hasher algorithms like MD5 (<https://www.npmjs.com/package/node-django-hashers>)

# Classes and Methods (Subteam 3-B)

This part can be a reference to automatic generated document for all classes and methods.

* For our Python stack we will be using Django’s admin documentation generator, found here: <https://docs.djangoproject.com/en/1.8/ref/contrib/admin/admindocs/>
  + Accessible from the Django admin login.
  + After logging in, one can view the automatically generated documentation for models, views, templates, etc.
  + Flexible in that we can include anything we want in the main docstrings, including method names, required parameters, general comments, etc.
  + Allows for use of special tags in the docstrings, such as the following:
    - Models - :model: `model\_name.model\_name\_goes\_here`
    - Views - :view: `view\_name.vew\_name\_goes\_here`
    - Template tags - :tag: `tagname`
    - Template filters - :filter: `filtername`
    - Template - :template: `path/to/template.html`
* At the moment we are not using any kind of automated documentation generation for our Javascript stack.

# References

* Websites/tools used
* Class slides

<https://docs.djangoproject.com/en/1.10/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names>

* <https://docs.djangoproject.com/en/1.8/ref/contrib/admin/admindocs/>

# Glossary

* Glossary of this document