**CS673S15 Software Engineering** 

**Group Project - Team 1 Group 3**

**Software Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Joe | QA Leader |  |  |
| Ya-Lan (Amy) | Requirement Leader |  |  |
| Lin-Kei Tseng (Ted) | Design Leader |  |  |
| Chun-Kai (Kenny) | Implementation Leader |  |  |
| Jerrold | Group1 Leader |  |  |
| Samer | Environment and  Integration Leader | Samer Abu-Nasser | 2/15/2015 |
| Josh | Configuration Leader |  |  |
|  |  |  |  |

**Revision history**

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

[Introduction](https://docs.google.com/a/bu.edu/document/d/1kvgsHDkLsMkmw5bW5AlIs7PSUhJbZyN3nJUf200Il3U/edit#heading=h.87t9hln2vjz0)

[Software Architecture](https://docs.google.com/a/bu.edu/document/d/1kvgsHDkLsMkmw5bW5AlIs7PSUhJbZyN3nJUf200Il3U/edit#heading=h.buttcq9i221r)

[Design Patterns](https://docs.google.com/a/bu.edu/document/d/1kvgsHDkLsMkmw5bW5AlIs7PSUhJbZyN3nJUf200Il3U/edit#heading=h.x18fj36s1121)

[Key Algorithms](https://docs.google.com/a/bu.edu/document/d/1kvgsHDkLsMkmw5bW5AlIs7PSUhJbZyN3nJUf200Il3U/edit#heading=h.mtfbusfb0eq3)

[Classes and Methods](https://docs.google.com/a/bu.edu/document/d/1kvgsHDkLsMkmw5bW5AlIs7PSUhJbZyN3nJUf200Il3U/edit#heading=h.7ucksmkf6rzx)

[References](https://docs.google.com/a/bu.edu/document/d/1kvgsHDkLsMkmw5bW5AlIs7PSUhJbZyN3nJUf200Il3U/edit#heading=h.15tmymhipvdv)

[Glossary](https://docs.google.com/a/bu.edu/document/d/1kvgsHDkLsMkmw5bW5AlIs7PSUhJbZyN3nJUf200Il3U/edit#heading=h.8n34lvocupub)

# Introduction

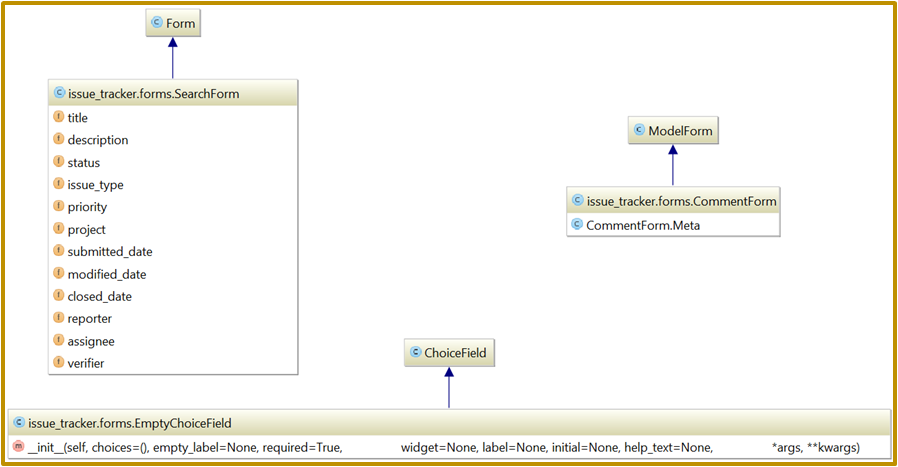
Our software system is an issue and bug tracker. It is designed to work as part of a larger suite of applications geared towards project management. Our primary goal is ease of use, followed by accurate and effective management of information, as well as providing a clean and attractive interface for the user

# Software Architecture

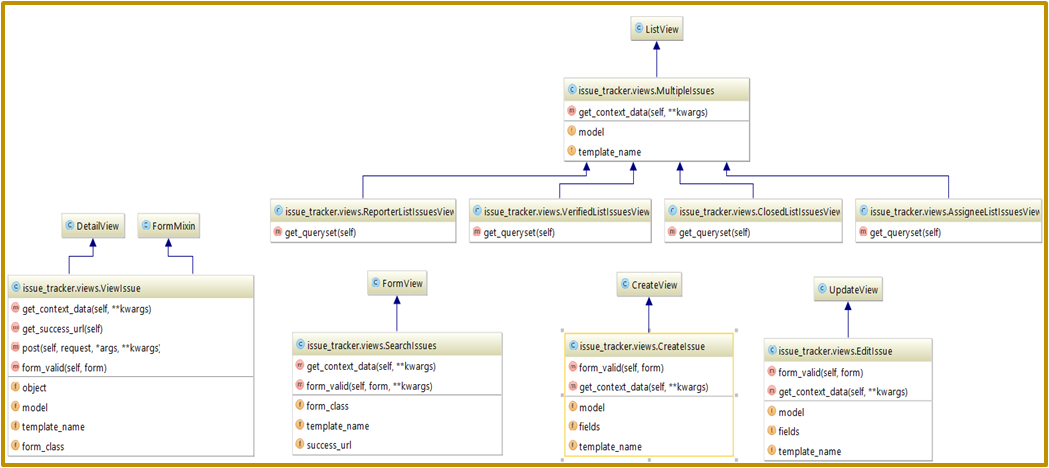
In this section, you will describe the decomposition of your software system, which include each component (which may be in terms of package or folder) and the relationship between components. You shall have a diagram to show the whole architecture, and **class diagram** for each component. The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too. **Database design** should also be described if used.

The base framework we are using to construct our web app is the Django framework, in the language python. The front end stylings are powered by Bootstrap. Testing of the app utilizes the built in unittest module of Django, as well as ui coverage provided by selenium webdriver.

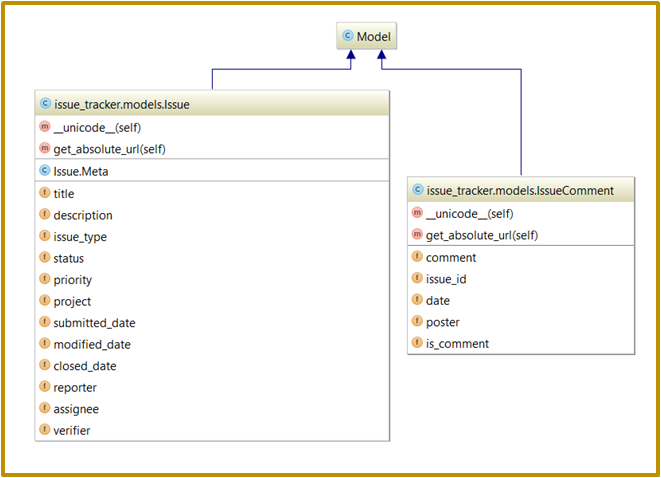
Class Diagram [ Form ]:



Class Diagram [ View ]:



Class Diagram [ Model ]:

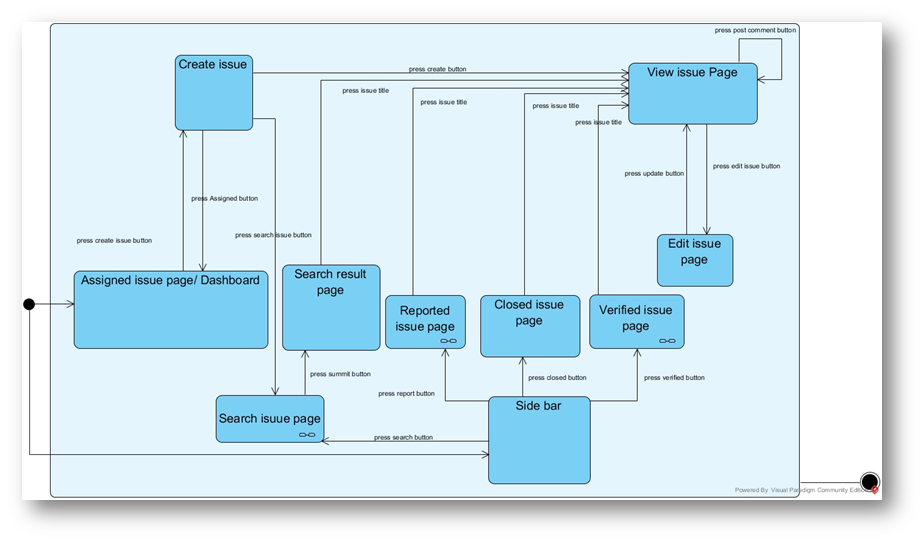


# Design Patterns

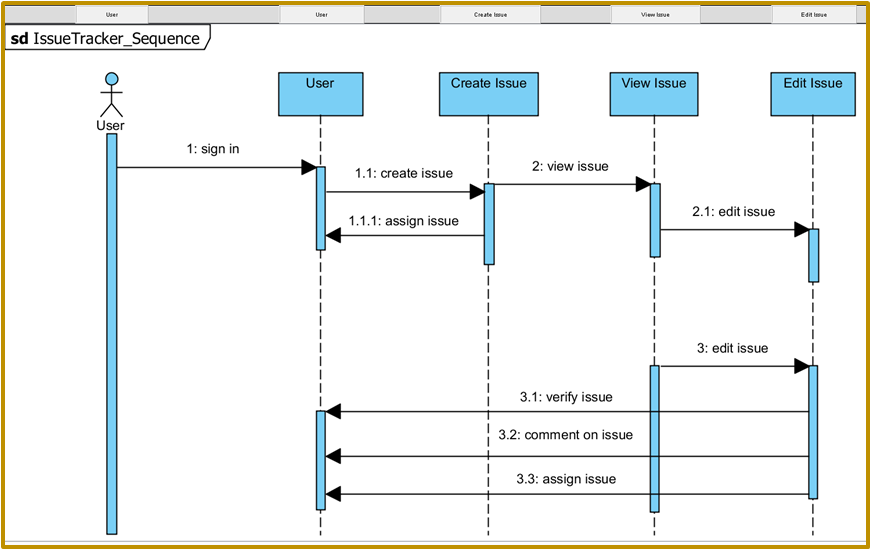
In this section, you shall describe any **design patterns** used in your software system.

We are following the overarching MVT design pattern used by Django for our project. On a smaller scale we’d like to implement page object test construction in selenium if time allows.

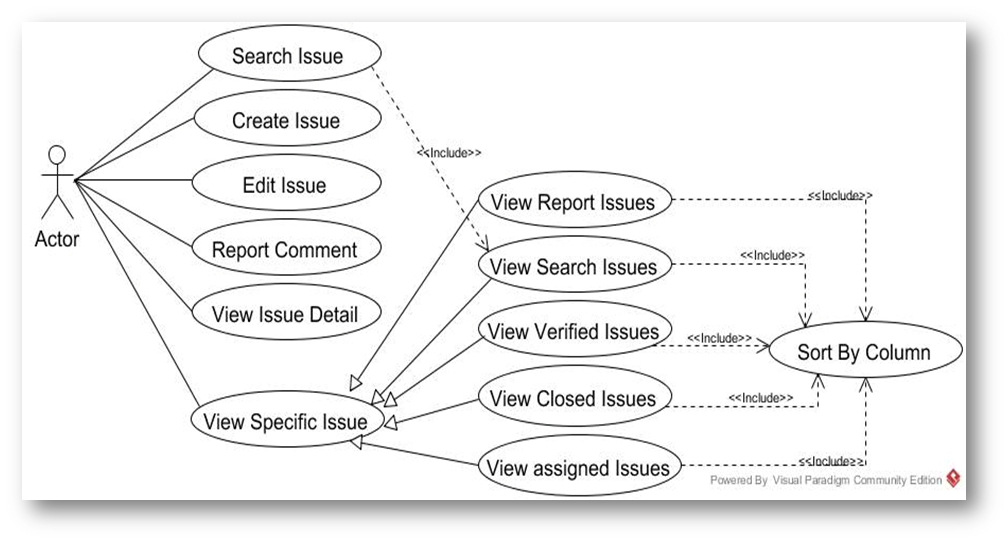
State Diagram:



Sequence Diagram:



Use Case Diagram:



# Key Algorithms

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

As an information storage and delivery system, we don’t have much in they way of heavy algorithms. We do have some functionality worth mentioning here, specifically the algorithms that drive our search functionality, and the algorithm under development that will govern state transitions of issues and bugs in the app.

# Classes and Methods

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

# References

<http://www.django-rest-framework.org/>

<https://www.djangoproject.com/>

<http://getbootstrap.com/>

<http://www.seleniumhq.org/>

<http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm>

# Glossary

**Testing Documentation**

Owing to the late stages of integration, all of our existing test suite was run on a locally generated instance of the issue tracker application. By implementing the page object design we created a fairly flexible infrastructure that allowed us to modify tests quickly and only according to specific templates when they were altered. This also allowed us to develop some very clear test cases. Essentially each subset of tests found in a file is linked to a page that we operate on the front end, and while they may interact with each other, the goal of those tests is to test that page specifically.

Our coverage of the app’s functions and processes is near-100%. We had full coverage of error functionality, search, navigation, login, issue creation, issue edits, and sorting features. We had high coverage of interactive visible elements, but were not complete there (we did not have a test confirming quantity displays in the sidenav were accurate, for example).

All of our current tests were 100% satisfied at the time of integration, and based on their reports we were confident of a functioning product when integrating with the other teams.