**CS673S16 Software Engineering**

**Team 3 - ProTeam**

**Software Design Document**



**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| **1.0** | **Josh Wildey** | **4/23/2016** | **Initial draft** |
| **2.0** | **Josh Wildey** | **4/24/2016** | **Modified/added to Server and Front end Architecture** |
| **3.0** | **Josh Wildey** | **4/25/2016** | **Added Server API Reference** |

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Danny Chiu | -Team Leader  -QA Leader  -Requirement Leader | *Danny Chiu* | 4/24/2016 |
| Brian Kabuye | -Implementation leader  -App Developer | *Brian Kabuye* | 4/26/2016 |
| ChihYung Wu | -Design Leader  -App Developer | *ChihYung Wu* | 4/27/2016 |
| Ayodele Awoleye | -Configuration Leader  -Server Developer |  |  |
| Josh Wildey | -Environment and Integration Leader  -Server Developer | *Josh Wildey* | 4/25/2016 |

# 

Table of Contents

[Introduction](#h.sjrqwwih7hkd)

[Purpose](#h.wl3da4vawnfd)

[Overview](#h.m5nhkg6v4fnh)

[Scope](#h.87s0a8yl07gb)

[ProTeam Android Application](#h.39lusqa2jdx2)

[Overview](#h.r6g5sa9furb5)

[Design Patterns](#h.sbl4sxit1kpz)

[Software Architecture](#h.6dlvvnh5iyqq)

[GUI Design](#h.w7rq46x9xo6y)

[Classes and Methods](#h.gdg7isknxkmf)

[Web Server Codebase](#h.sen1973hvu52)

[Overview](#h.osr8k082pmvt)

[Design Patterns](#h.3t0gmdewkfi3)

[Software Architecture](#h.2hrex1xx0zbs)

[Classes and Methods](#h.dawqqplpk6fa)

[Automatically Generated Package/Class Diagrams](#h.ewyxqv7y7e8f)

[API Interface Reference](#h.nddumd36du0r)

[Get Token](#h.3k5tiywq9358)

[Sign Up](#h.3zw41jsse12i)

[User Details](#h.mil6bdyocyl6)

[User Projects](#h.aufhex0xq9l)

[User Project Details](#h.d2bt9vanvuc6)

[User Stories](#h.qzd40jwdv1p4)

[References](#h.pfdnqihmw5h)

[Acronyms](#h.w5munlqw9m68)

# 

# 

# Introduction

# Purpose

This document is intended to describe the design and architecture of the system. Design decisions will be explained and diagrams will be shown to give a visual representation of the system.

# Overview

The Project Management Tools website ([www.3blueprints.com](http://www.3blueprints.com)) was developed to give software development teams a means of managing and tracking progress of the software application development. The three main functionalities that it provides are requirements tracking, issue tracking and chat rooms. Requirements tracking provides a way to generate user stories that define the required functions of the software being developed. The issue tracker provides a way to keep track of found software bugs. The chat rooms provide an easy method of communication amongst development teams and a way to share ideas, comments and concerns.

The ProTeam Android application is a mobile application that adds an additional interface to the website that is intended to provide all the same capabilities as the website. This mobile application requires communication with the web server through a REST API to give it access to all the same information that the website provides through the browser. This document is therefore broken up into two major sections each describing the Android app itself changes made to the server codebase.

# Scope

The initial design effort for the ProTeam Android App focused on the Requirement Tracker aspect of the website and therefore this document will be focused there however the dashboard of the App GUI will contain buttons for the other two functions for future development.

The App uses the REST API of [www.3Blueprints.com](http://www.3blueprints.com) and so the focus of the web server development will be focused on that aspect.

# ProTeam Android Application

# Overview

The ProTeam Android Application is built using Android Studio using the Marshmallow Android Framework.

# Design Patterns

The primary design pattern used for the ProTeam App was Model-View-Controller as it most commonly is for Android development. Views are created via XML style files and have buttons/fields defined with them. The Controllers the grab information that it needs from models and populates the views from models or vice versa.

Singleton is also used which restricts instantiation of certain classes to only one object.

# Software Architecture



# Front End Classes and Methods

**MainActivity class:** This class handles the first home\_activity layout, a user interacts with upon launching up. When user clicks on signIn button, they are prompted to Login screen button. When user clicks on SignUp button, they are prompted to signUp page.

**SignIn class:** User information is validated in this class upon sending it to *ServerRequest* for authentication. All validation check stops are handled in this class in relation to username and password inserted.

**Register class** : User information entered from signUp\_activity layout is validated in register class before it’s sent to *ServerRequest*. All check stops relating to right email format, username and password validation are all handled in register clas.

**User class**: When a user is authenticated, a new instance of that user is created. User class will holds tokens of authenticated users

**userLocalStore:** The purpose of this class is to limit on amount of requests and response sent to server. The ideology was to have setters and getters store where user information is stored or changed before it’s committed or sent to server. It would improve on scalability of easy data access and help reduce on traffic

**ServerRequest:** Most of back end communication is done in this class. It uses Async Task capabilities to retrieve responses intuitively. Requests and responses are handled and sent in json format to back end api.

**Profile Activity:** Upon user’s successful login, welcome screen is displayed. This is handled in this class.

**Project:** This class handles and manipulates projects retrieved from api. When user requests for projects from api, *ServerRequest* class retrieves response from server and assigns requests to Project class that displays stories to the *DashBoard* class

**GetUserCallback** : This is an Interface to hold an abstract void method *done()* that's called when *ServerRequest* retrieves a response from the Api.

**GetProjectCallback** : This is an Interface to hold an abstract boolean method *done()* that's called when *ServerRequest* retrieves a response from the Api.

**DashboardActivity**: This is the activity where all the projects created by the logged-in user will be displayed.

**DatePickerFragment:** This is a dialog fragment which is used to allow user to select a date when he/she is signing up for an account in the SignUpActivity.

**IceBox:** This is an activity where all the iceboxes related to a project will be displayed. Users will navigate to this activity when select a project from the project list in the Dashboard activity.

**IterationAddDialogFragment:** This is a dialog fragment used to allow users to create a new iteration for a project.

**IterationEditFragment:** This is a dialog fragment allowing users to edit an existing iteration of a project.

**ProjectAddFragment:** This is a dialog fragment allowing users to add a new project in the DashboardActivity.

**ProjectEditFragment:** This is a dialog fragment allowing users to edit an existing project in the ProjectInfoActivity.

**ProjectInfoActivity:** This is an activity allowing users to view the information of a specific project when users select a specific project from the DashboardActivity.

**ProjectLayout:** This is class extended from the RelativeLayout class, which is used to represent each project in the Project ViewList in the Dashboard activity.

**ProjectListAdapter:** This is an adapter class that sits between the ProjectLayout and the ViewList widget in the Dashboard activity, and allows manipulation on the project objects in the list.

**ProjectProfileActivity:** This is the activity which users will navigate to after they select a project from the list in the Dashboard Activity. This activity will also set up the tabs that will be used by the TabFragment class.

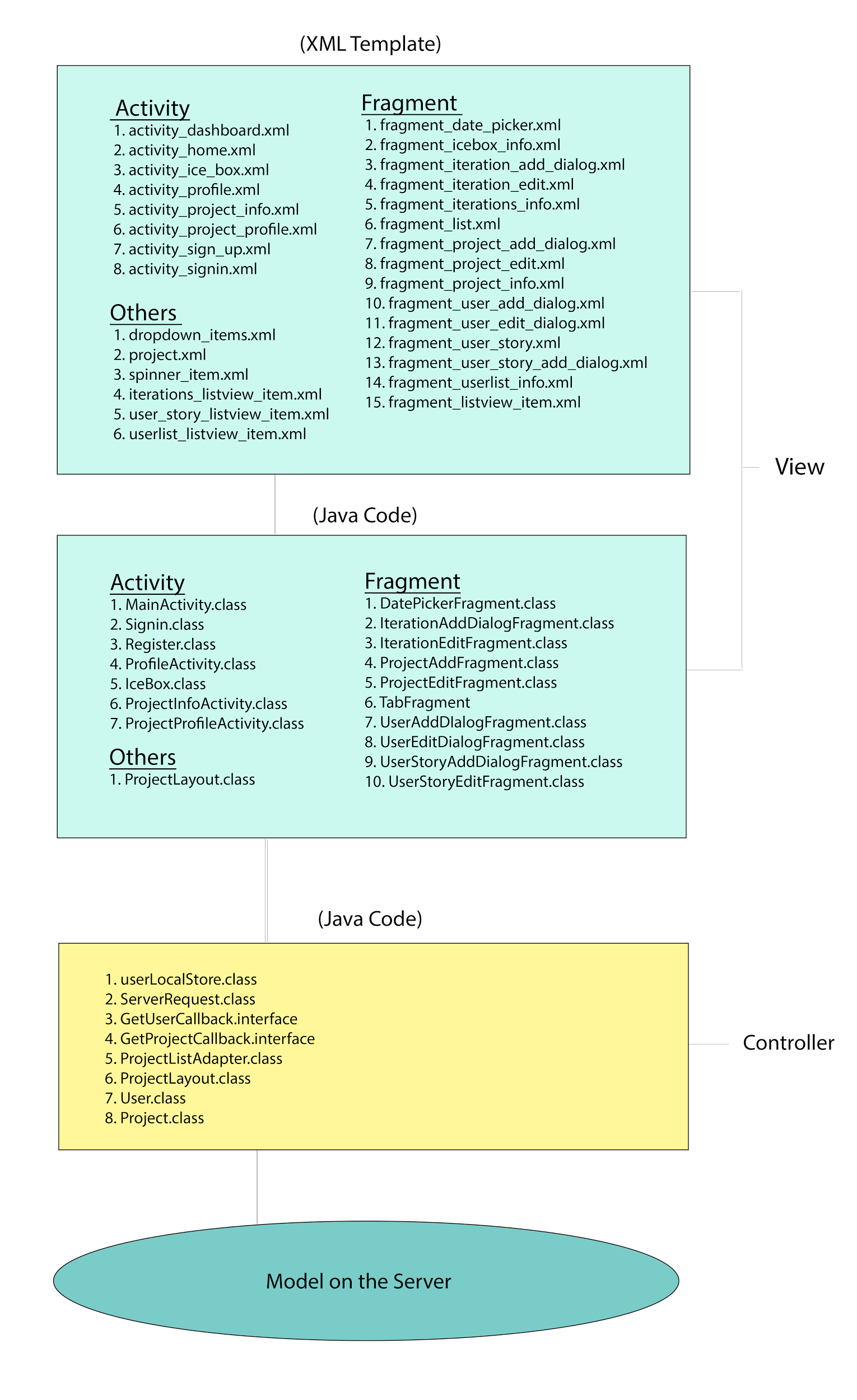
**TabFragment:** This is the fragment which is contained by the ProjectProfileActivity class and has all the logics for navigation between between different tabs defined inside the ProjectProfileActivity class.

**UserAddDialogFragment:** This is a dialog fragment which allows an admin user to assign more users to a specific project owned by this admin user.

**UserEditDialogFragment:** This is a dialog fragment which allows an admin user to edit the existing users assigned to existing projects owned by this admin user.

**UserStoryAddDialogFragment:** This is a dialog fragment which allows an user to create a new user story for a specific project owned by this user.

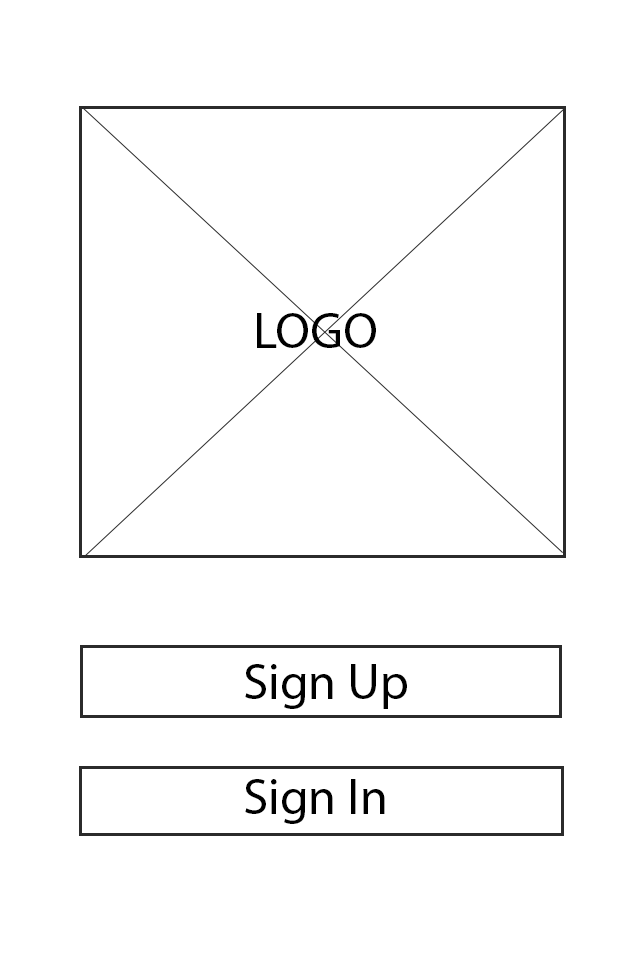
**UserStoryEditFragment:** This is a dialog fragment which allows an user to edit an existing user story for a specific project owned by this user.



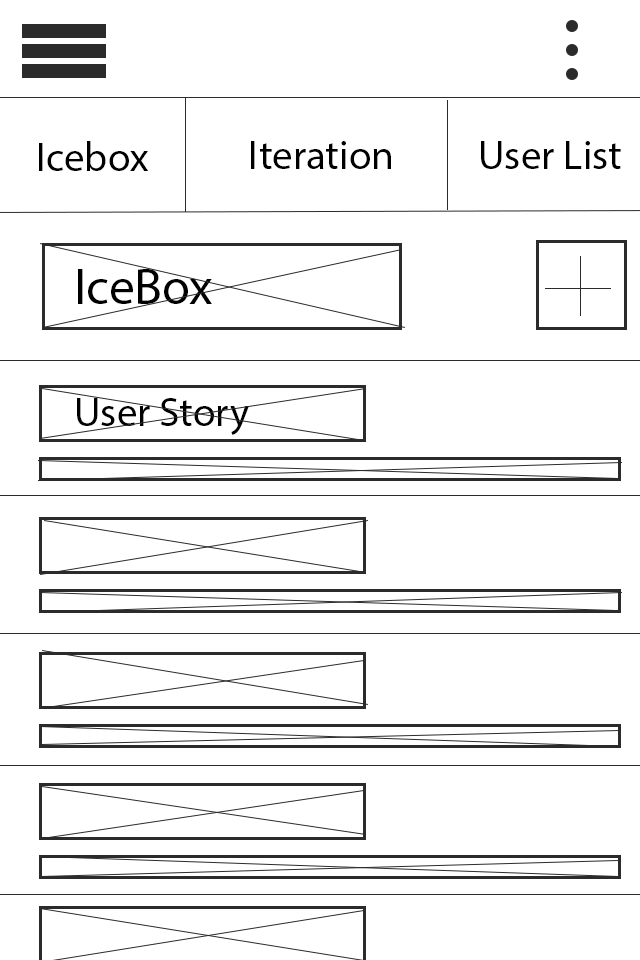
**Front-End Structure**

# GUI Design

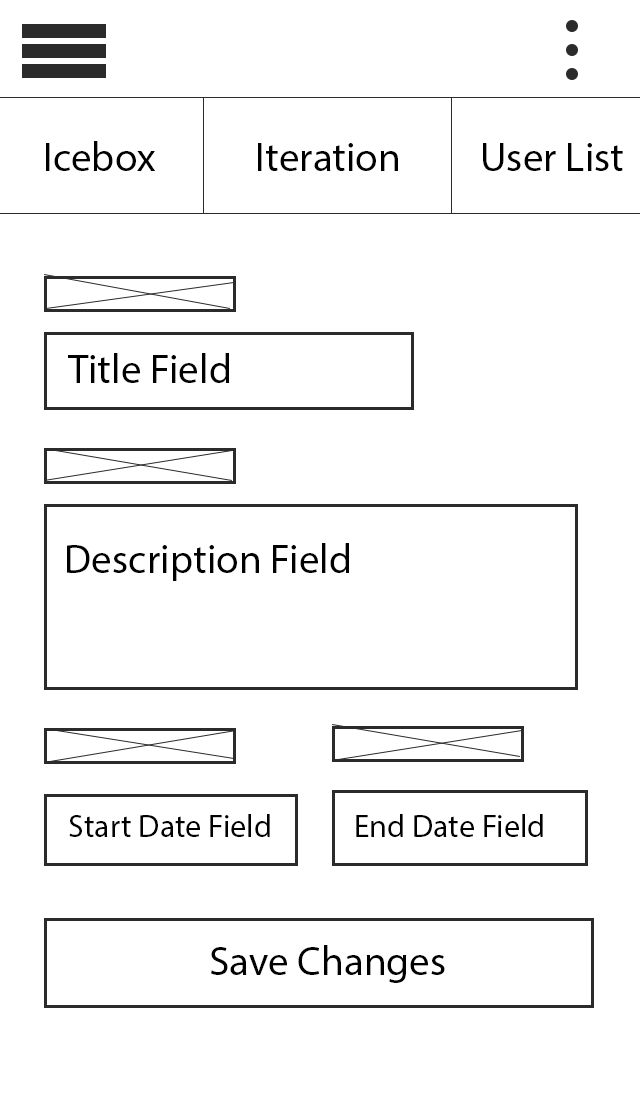
The GUI was designed to be user friendly and to reflect a similar interface that the website provides. When a user first starts the App they should be prompted with option to either sign-up or sign-in. On the sign-in screen, user should be able to authenticate by entering their credentials and clicking submit. The user will then be brought to their dashboard where they can choose between the Chat rooms, Issue Tracker and Requirements Tracker. Once the user clicks on the Requirements Tracker button the list of projects will be displayed on the following screen but only those associated with the user. If user selects one of the projects be brought to the project description screen. At top of the screen should be buttons that allows user to switch between the iteration screen and icebox screen. Each of those screens will display list of user stories. If user selects one of these user stories, user will be shown the user story detail page. Below is a set of concept drawings and set of screenshots taken from development.



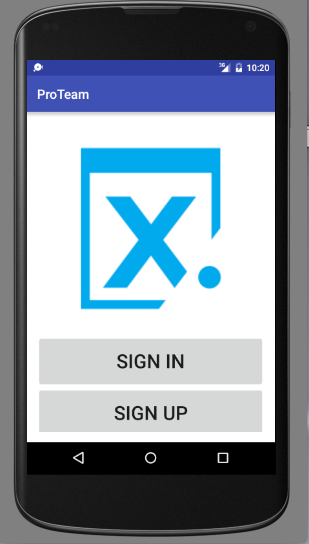
**Concept Welcome Screen**



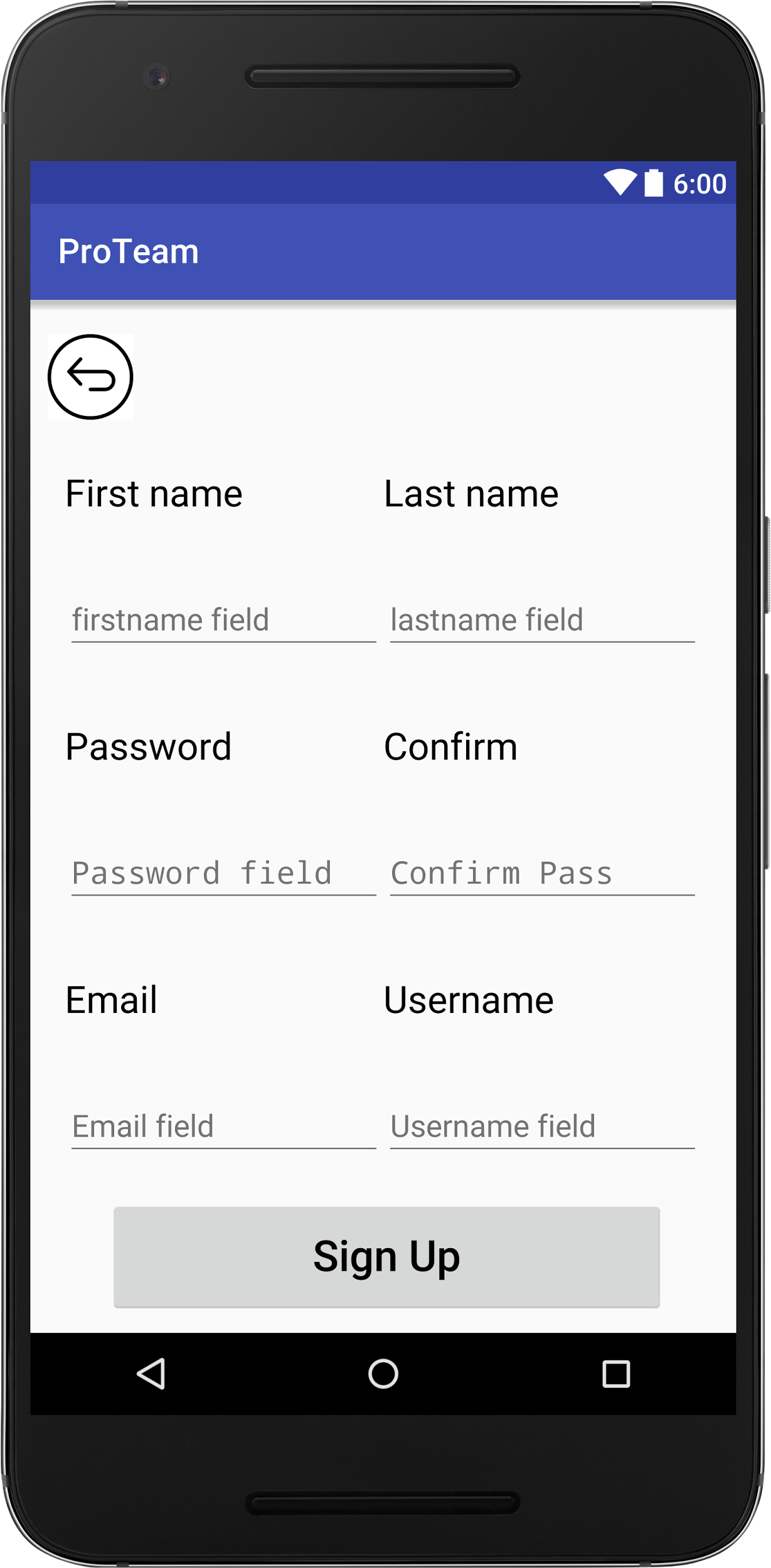
**Concept Icebox Screen**



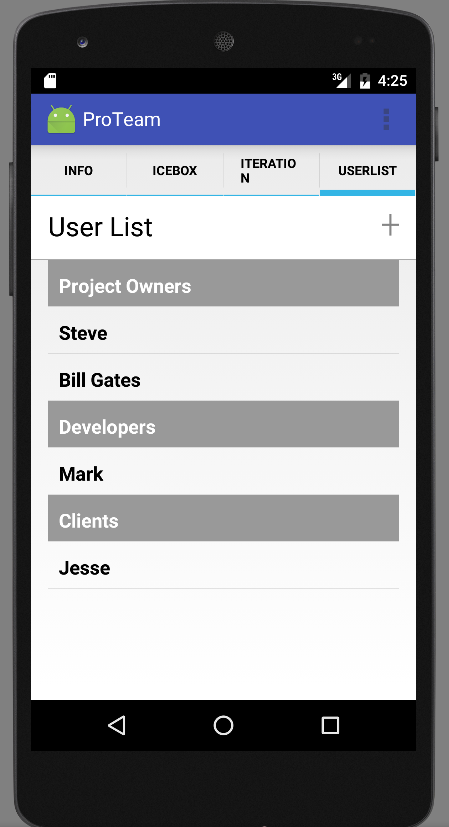
**Concept Iteration Screen**



**Welcome Screen**

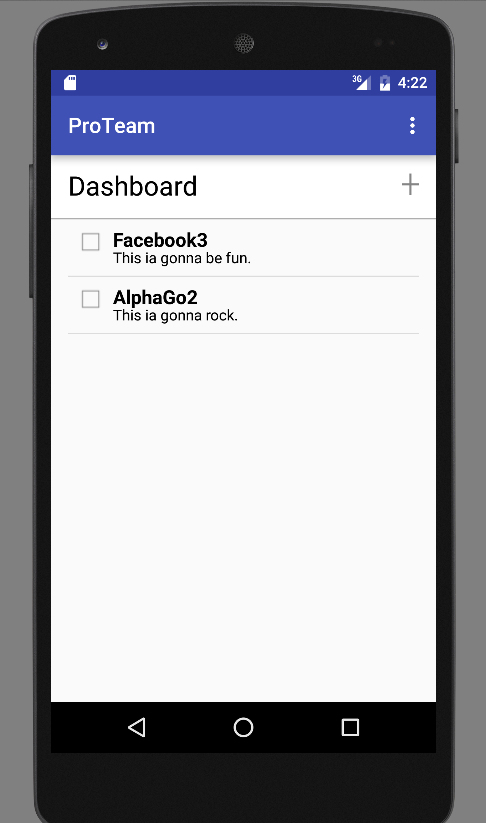


**Signup Screen**

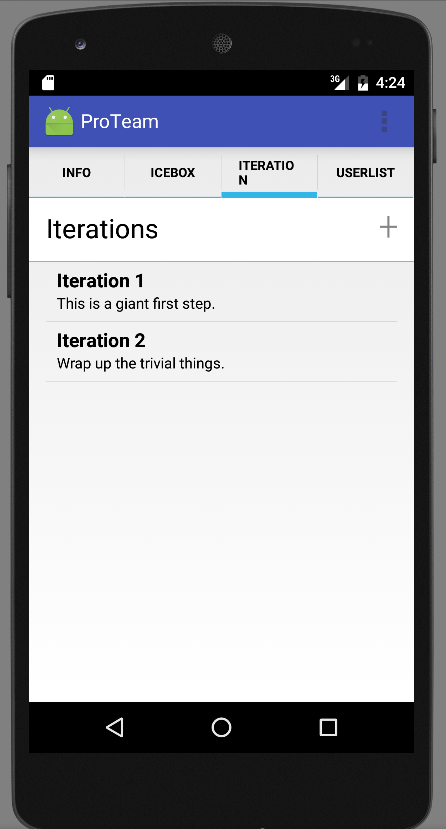


**User List Screen**

**Icebox Screen**



**Dashboard Screen**



**Iteration Screen**

# Classes and Methods

See ProTeamJavaDoc.zip. Open index.html to access the top level documentation.

# Web Server Codebase

# Overview

The web server codebase uses a Django python server. In addition to the base Django framework, the web server also utilizes the following python packages:

* Django-cors-headers
* Django-extensions
* Django-filters
* Djangorestframework
* Django-restframework-jwt (newly added)
* Django-sslserver (newly added)

The Django-restframework-jwt package was added for development of the ProTeam application as well as the Django-sslserver. The restframework-jwt package was added for JSON web token authentication through the API. The sslserver package was added on to provide HTTPS encryption so plain text usernames and passwords aren't transmitted across the web. With the introduction of the SSL server capabilities comes a SSL certificate for the website. This certificate creates errors when creating HTTP requests generating untrusted website errors so the SSL server was not used during development of the ProTeam App.

Before starting the software development effort for the ProTeam App the REST API only exposed users and chat rooms. The effort put forth for the ProTeam App expanded upon the REST API interface to expose projects and user stories.

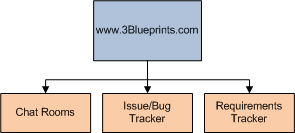
# Design Patterns

No new design patterns were introduced when expanding the REST API for the ProTeam App but the Django framework uses Model-Template-Control which is a variation of Model-View-Control architecture at its core and thus is the design pattern used for the web server. The models defined in the Django framework are then used for models in the database. The web server also uses the Factory Pattern which dynamically loads objects instead of having to know and load exactly what the server needs when it starts up.

# Software Architecture

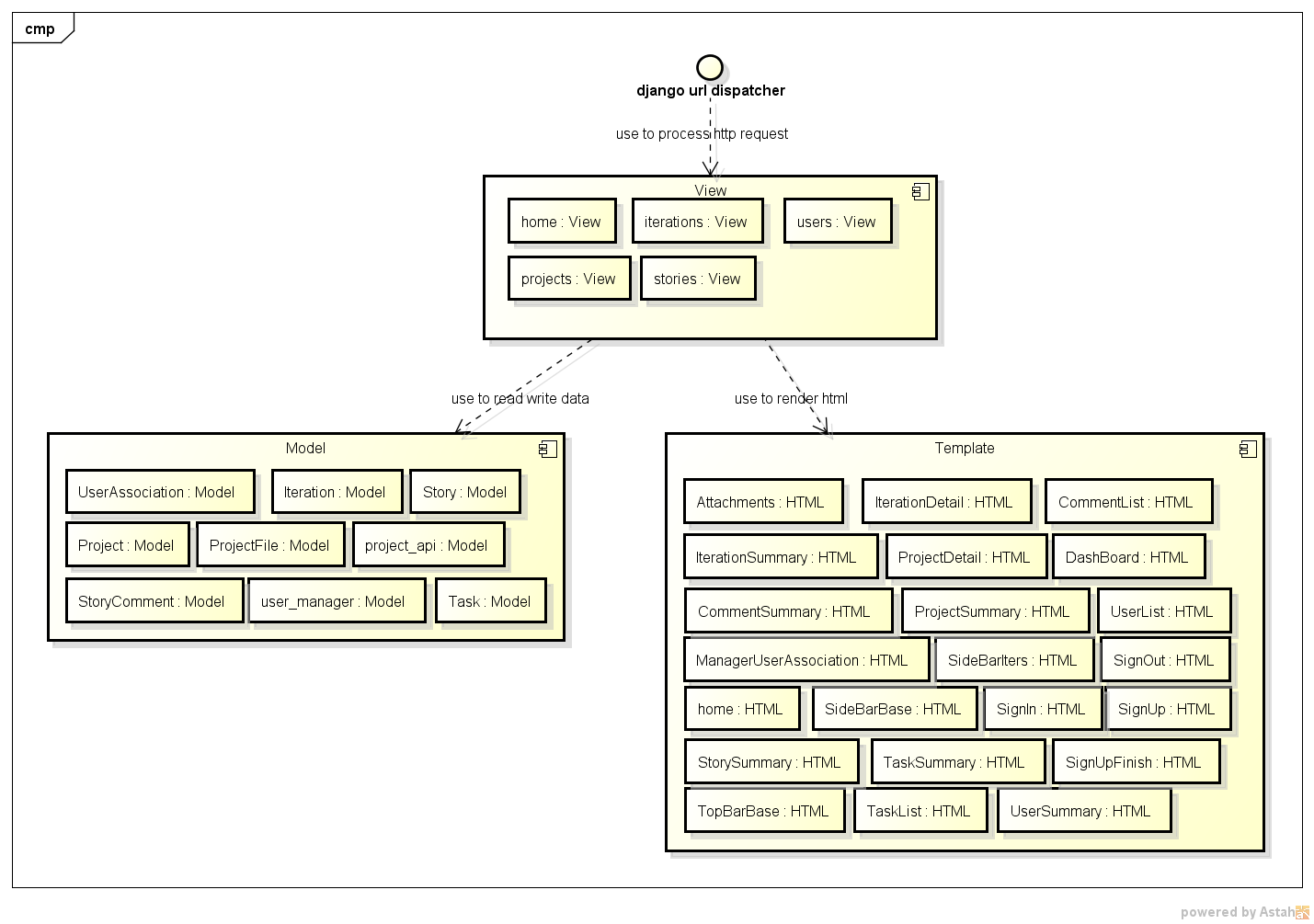
The web server is broken into three main functionalities:

* Chat Rooms
* Issue/Bug Tracker
* Requirements/User Story Tracker



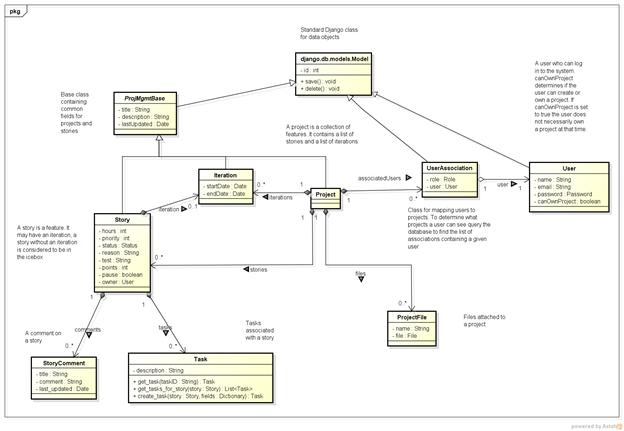
**Overall Architecture**

Further dissecting each component of the web server, it can then be separated into to the models, views and templates as seen in the diagram below.



**MVT Architecture relevant to Requirements Tracker [1]**

Below is a diagram that shows the models that are relevant to the Requirements Tracker. These were minimally modified during development to add more member variables and accessors but their hierarchy was not modified. More details about what was changed can be found in [Section 3.4. Classes and Methods](#h.dawqqplpk6fa).



**Models relevant to Requirements Tracker [1]**

The REST API does not have to use templates so those were unchanged throughout the development of the ProTeam App. Expanding on the REST API only required changes to the Views and Models of the MVT architecture. Members and methods were added to models and views to be able to support the REST API. Details on specific changes can be found in [Section 3.4. Classes and Methods](#h.dawqqplpk6fa).

The database used for the development was SQLite. It is the default database used by Django which is why it was chosen and it provided sufficient performance for the development of the ProTeam App. The design of the database is handled by the Django framework and based upon the design of the Models defined in the server codebase.

# Classes and Methods

**No Changes**

Help on package comm:

NAME

comm

PACKAGE CONTENTS

admin

migrations (package)

models

serializers

tests

urls

views

FILE

\project-mgmt-backend-master\comm\\_\_init\_\_.py

**No Changes**

Help on package issue\_tracker:

NAME

issue\_tracker

PACKAGE CONTENTS

admin

filters

forms

management (package)

migrations (package)

models

serializers

tests (package)

urls

utils

views

viewsets

FILE

\project-mgmt-backend-master\issue\_tracker\\_\_init\_\_.py

**No Changes**

Help on package project\_router:

NAME

project\_router

PACKAGE CONTENTS

admin

migrations (package)

models

urls

views

FILE

\project-mgmt-backend-master\project\_router\\_\_init\_\_.py

Help on package ProTeam:

NAME

ProTeam

PACKAGE CONTENTS

settings (Modified)

* Added JWT token
* Added SSL server
* Modified database location
* Defined permission classes
* Defined authentication classes

urls (Modified)

* Added get token URL
* Added signup URL
* Added URLs to support API expansion

wsgi

FILE

\project-mgmt-backend-master\proteam\\_\_init\_\_.py

Help on package requirements:

NAME

requirements

PACKAGE CONTENTS

admin

migrations (package)

models (package) (Modified)

req\_urls

templatetags (package)

tests (package)

views (package) (Modified)

FILE

\project-mgmt-backend-master\requirements\\_\_init\_\_.py

Help on package migrations:

NAME

migrations

PACKAGE CONTENTS

0001\_initial

0002\_project\_owner

0003\_auto\_20160226\_2328

FILE

\project-mgmt-backend-master\requirements\migrations\\_\_init\_\_.py

Help on package models:

NAME

models

PACKAGE CONTENTS

base

files

iteration

project (Modified)

* Added owner field
* Added users field
* Added stories field
* Added accessor for project name

project\_api

serializers (Modified)

story (Modified)

story\_comment

task

user\_association

user\_manager

FILE

\project-mgmt-backend-master\requirements\models\\_\_init\_\_.py

Help on package templatetags:

NAME

templatetags

PACKAGE CONTENTS

require\_tags

FILE

\project-mgmt-backend-master\requirements\templatetags\\_\_init\_\_.py

problem in tests - ImportError: No module named 'ui'

Help on package views:

NAME

views

PACKAGE CONTENTS

forms

home

iterations

projects

stories (Modified)

story\_comments

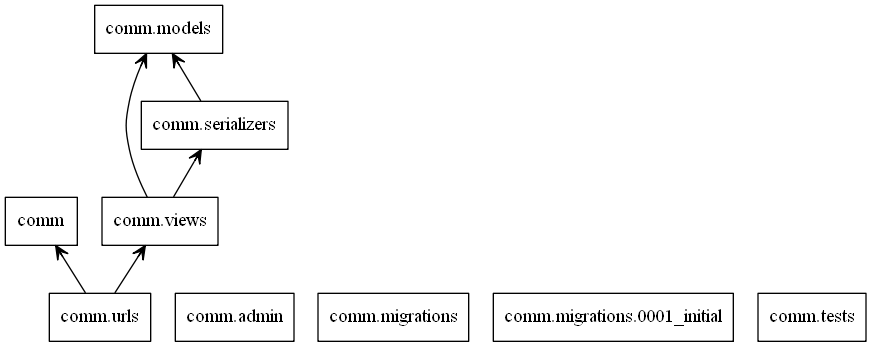
users

views (Modified)

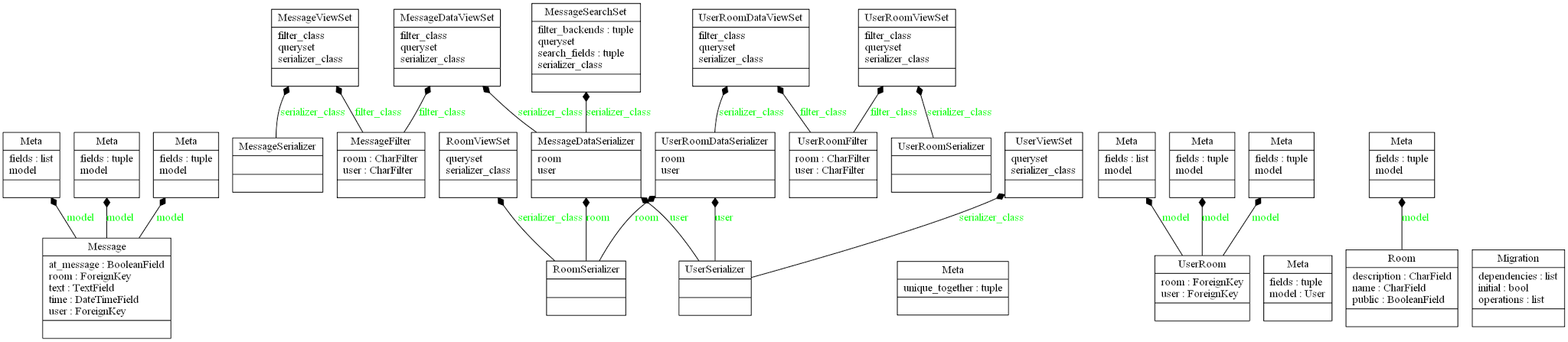
FILE

\project-mgmt-backend-master\requirements\views\\_\_init\_\_.py

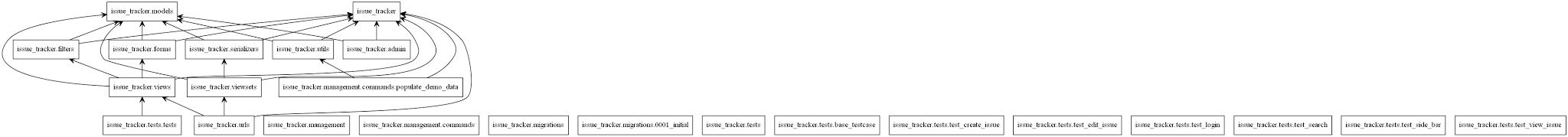
# Automatically Generated Package/Class Diagrams



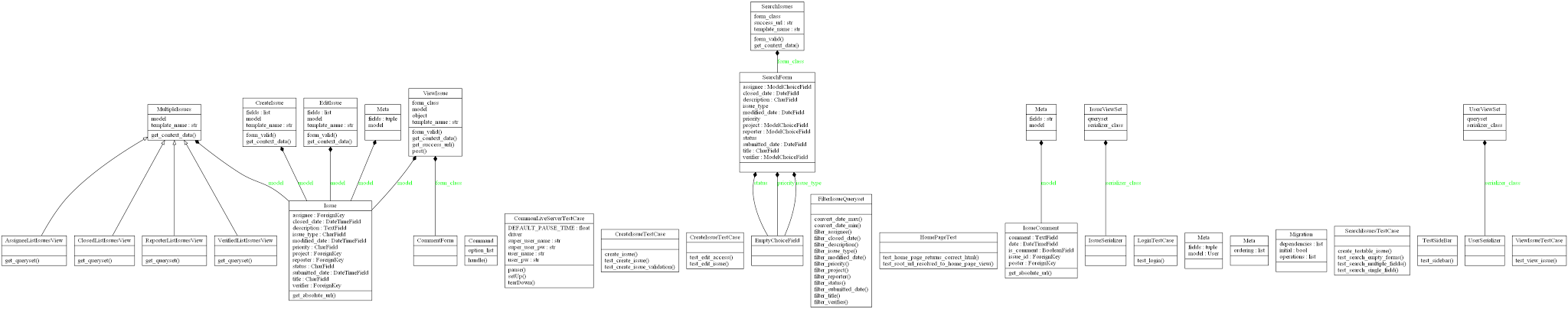
**Package Diagram for Comms**



**Class Diagram for Comms**



**Package Diagram for Issue Tracker**

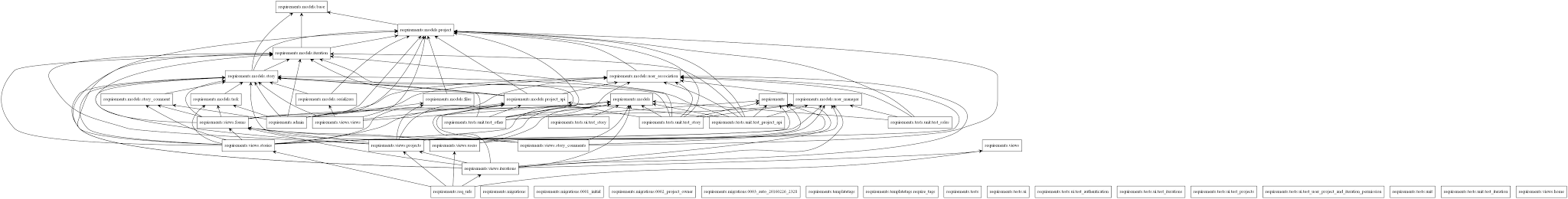
**Class Diagram for Issue Tracker**

packages_project_router.png

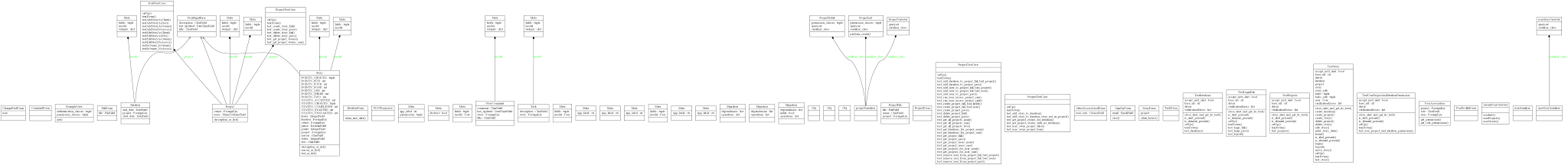
**Package Diagram for Project Router**

packages_ProTeam.png

**Package Diagram for ProTeam**



**Package Diagram for Requirements**



**Class Diagram for Requirements**

# API Interface Reference

The API Interface Reference it intended to provide the mobile developer to with URI details of how to access information from the server. The current server does not have a domain name and is therefore accessed by its IP address (128.197.103.77). Accessing the web server is accessed by using the following base URL:

<http://128.197.103.77>

The functions that are exposed by the API are strings that are appended to the end of the base URL shown above. The API is built on HTTP, is [REST](http://en.wikipedia.org/wiki/Representational_State_Transfer) and it:

* Uses predictable, resource-oriented URLs.
* Uses built-in HTTP capabilities for passing parameters and authentication.
* Responds with standard HTTP response codes to indicate errors.
* Expects parameters in JSON
* Returns [JSON](http://en.wikipedia.org/wiki/Jsonobjects).

The API returns standard HTTP response codes:

|  |  |
| --- | --- |
| **Code** | **Description** |
| 200 | No error |
| 400 | Bad Request - Often missing a parameter |
| 401 | Unauthorized - valid token not provided |
| 402 | Request Failed - valid parameters but request failed |
| 404 | Not Found - item doesn’t exist |
| 500, 502, 503, 504 | Server Errors - Something wrong on server’s end |

# Get Token

/get-token/

Get Token is the method of authentication with the server. Use this method to retrieve an access token for use with communication to the server.

**Request**

POST

Include username and password parameters

**Parameters**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| Username | *String* | User’s username |
| Password | *String* | User’s password |

**Request Sample**

curl --data "username="josh"&password="josh"" http://128.197.103.77/get-token/

**Response**

|  |  |  |
| --- | --- | --- |
| **Data** | **Type** | **Description** |
| Response | *String* | JSON representation of either error message or JSON Web Token |

**Response Sample**

{"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFpbCI6ImZha2VAZ21haWwuY29tIiwidXNlcm5hbWUiOiJqb3NoIiwiZXhwIjoxNDYxNTMxNDk5LCJ1c2VyX2lkIjoyfQ.0dQGEuurlCmUruRK4N1c885r5PHUrXMIaxpr4xkjg8g"}

# Sign Up

/apisignup/

Sign up is the method of registering a new user with the server. Use this method to create a new user account on the server.

**Request**

POST

Include username, password, name (first and last), and email parameters as a JSON string.

Note: In the example using cURL, quotes need to be delimited inside the JSON string.

**Parameters**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| username | *String* | User’s username |
| password | *String* | User’s password |
| firstname | *String* | User’s first name |
| lastname | *String* | User’s last name |
| email | *String* | User’s Email address |

**Request Sample**

curl -H "Content-Type: application/json" -X POST -d "{\"username\":\"test\",\"password\": \"test\", \"email\":\"test@test.com\",\"firstname\": \"te\",\"lastname\": \"st\"}" http://128.197.103.77/apisignup/

**Response**

|  |  |  |
| --- | --- | --- |
| **Data** | **Type** | **Description** |
| Response | *String* | JSON representation of either error message or JSON string of new user info |

**Response Sample**

[{"id":18,"username":"test","email":"test@test.com"}]

# User Details

/api/users/$/

User Details is the method of retrieving user information with the server. Use this method to retrieve info of all users, or a specific user from the server.

Note: Replace ‘$’ with user ID for that user’s information or omit ‘$/’ for all user info

**Request**

GET

Note: The JWT token is required in the header of the HTTP request

**Parameters**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| none |  |  |

**Request Sample**

curl -H "Authorization: JWT <token>" -X GET http://128.197.103.77/api/users/<id>/

**Response**

|  |  |  |
| --- | --- | --- |
| **Data** | **Type** | **Description** |
| Response | *String* | JSON representation of either error message or JSON string with user info |

**Response Sample**

{"id":19,"username":"test","email":"test@test.com","last\_login":null}

# User Projects

/api/<username>/projects/

User Projects is the method of retrieving project information associated with a specific user from the server. Use this method to retrieve a list of projects and their info

**Request**

GET

Note: The JWT token is required in the header of the HTTP request

**Parameters**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| none |  |  |

**Request Sample**

curl -H "Authorization: JWT <token>" -X GET http://128.197.103.77/api/<username>/projects/

**Response**

|  |  |  |
| --- | --- | --- |
| **Data** | **Type** | **Description** |
| Response | *String* | JSON representation of either error message or JSON string with a list of project info |

**Response Sample**

[{"title":"josh's project","description":"mine","owner":null,"id":5,"users":["ht

tp://128.197.103.77/api/users/2/","http://128.197.103.77/api/users/3/","http://1

28.197.103.77/api/users/4/","http://128.197.103.77/api/users/5/"]},{"title":"bri

ans project","description":"mine","owner":null,"id":6,"users":["http://128.197.1

03.77/api/users/3/","http://128.197.103.77/api/users/2/"]}]

# User Project Details

/api/<username>/projects/$/

User Projects is the method of retrieving project information associated with a specific user from the server. Use this method to retrieve specific project info from the server.

**Request**

GET

Note: The JWT token is required in the header of the HTTP request

**Parameters**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| none |  |  |

**Request Sample**

curl -H "Authorization: JWT <token>" -X GET http://128.197.103.77/api/<username>/projects/5/

**Response**

|  |  |  |
| --- | --- | --- |
| **Data** | **Type** | **Description** |
| Response | *String* | JSON representation of either error message or JSON string with project info |

**Response Sample**

[{"title":"josh's project","description":"mine","owner":null,"id":5,"users":["ht

tp://128.197.103.77/api/users/2/","http://128.197.103.77/api/users/3/","http://1

28.197.103.77/api/users/4/","http://128.197.103.77/api/users/5/"]}]

# User Stories

/api/<username>/projects/<project\_ID>/userstories/

User Stories is the method of retrieving user stories associated with a project from the server. Use this method to retrieve all user stories associated with a specific project, or a specific user story from the server.

Note: Replace ‘$’ with user ID for that user’s information or omit ‘$/’ for all user info

**Request**

GET

Note: The JWT token is required in the header of the HTTP request

**Parameters**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| none |  |  |

**Request Sample**

curl -H "Authorization: JWT <token>" -X GET http://128.197.103.77/api/<username>/projects/<project\_id>/userstories/

**Response**

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| Response | *String* | JSON representation of either error message or JSON string with user info |

**Response Sample**

[{"title":"fas","description":"ffdafd","owner":null,"project":"http://128.197.10

3.77/api/projects/5/"}]

# References

1. **CS673S15 Software Engineering Software Design Document**. Spring 2015 Team 1 Student Project. Almanzor, Max.

# Acronyms

HTTP: Hyper Text Transfer Protocol

cURL: Tool to transfer data from or to a server

API: Application Programming Interface

JSON: JavaScript Object Notation

SDD: Software Design Document

JWT: Java Web Token