**Final Project**

# **Intermediate Object-Oriented Programming Daniel Swain**

### **Introduction:**

The following documentation represents the scope, design and implementation of the **KFL – Fantasy Competition** (KFL) Android mobile application and associated web service.

The KFL application will complement an existing website I have designed and implemented for a local fantasy AFL competition, providing a simple and intuitive interface for competition entrants to view news articles, view their fantasy team configuration and update their selected players for upcoming rounds.

The application is to be released on the Google Play Store and connect to the [www.kfl.com.au](http://www.kfl.com.au) site using a newly designed and built web service.

### **Scope:**

The KFL application shall have similar functionality to the existing website, but have the user interaction modified to suit the native Android application experience. The KFL web site is already responsive on mobiles so the Android application should provide a concise and easy to use experience, complementing the functionality already available.

The following represent the functional requirements for the KFL application:

* Connect to [www.kfl.com.au](http://www.kfl.com.au) and bring in the latest articles from the website.
* Implement a single article view for each article.
* Allow KFL competition entrants to login to authorise the following actions:
  + retrieval of the user’s player/team roster;
  + retrieval of the user’s selected team for the upcoming rounds; and
  + ability to select and update the user’s selected team in the application and on the web server (which powers live scoring for the competition).

In addition to the above requirements, the KFL application should allow the users to view articles in their devices web browser if desired (on the original site) as well as manually check for new articles via a manual toggle.

### **Design and Implementation:**

#### **Pre-existing information:**

The KFL application will require access to data that is already hosted on [www.kfl.com.au](http://www.kfl.com.au) in the form of news articles generated by the competition admins, user team lists and current user selections for the weekly competition.

Information is currently stored in a single database with separate tables for each of the following:

* The user’s team.
* The user’s selected team (as it is smaller than the user’s full roster).
* All the news articles.
* All the AFL player objects (used for live scoring and to build the team’s)

This information is currently only accessible to the website, which is an existing project I completed at the beginning of 2016 using Python and the Django web framework.

#### **Web service:**

The information listed above is only accessible to the web site (through the Django application), as such, it is not accessible from within an Android application as there are no API endpoints exposed for the web server data. Prior to building the Android application, these API endpoints would need to be added to the web site.

The following endpoints were decided upon for implementation as they provided the functionality required by the Android application:

* /api/user\_team/ - Endpoint for GET requests for a user’s team roster.
* /api/selected\_team/ - Endpoint for GET requests for a user’s selected team.
* /api/selected\_team/:id – Endpoint for PUT/PATCH requests to update a user’s team (using the :id of the selected team in the database)

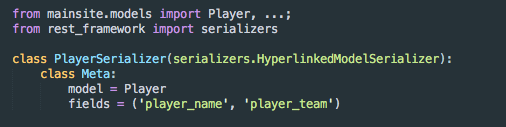
These API endpoints were added by installing the Django-Rest-Framework to the existing Django application. This allows for API routes to be easily configured by setting a URL route in the application’s URL routing file and then build custom Serializer files that parse the web applications data from the database into a JSON response.

To ensure only authorised users can access certain API endpoints (the user\_team and selected\_team endpoints) I installed the Django-Rest-Auth plugin to my application. This application provides the following additional API endpoints that generate user tokens that the Django application uses to validate API requests

* /rest-auth/login/: log the user in (using username, email and password) and generate a token for that user, which is sent back in the JSON response.
* /rest-auth/logout/: log the user out (validated by the above token) and delete their API token from the server.

Once these two plugins were added to the existing application I was able to implement the Web Service for the Android application.

The initial step is to build the JSON serializer python functions that will be used by the API to return data and also parse input data to perform database queries and PUT/PATCH/POST actions.

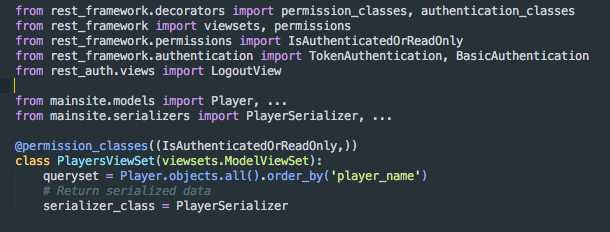


**Excerpt from Serializer.py**

The above image represents an excerpt from the Serializer.py file that returns (or updates) the database models in JSON form. Each database model that requires access from the API needs a serializer class defined.

The PlayerSerializer connects to the the Player model in the database and returns a JSON object containing the Player’s player\_name and player\_team. These simple Serializer files support read and write actions through the rest\_framework.

Once the serializers are written for each API endpoint I build View’s for the API Endpoints as the Django Rest Framework provides a web browsable API, which will be useful for testing.



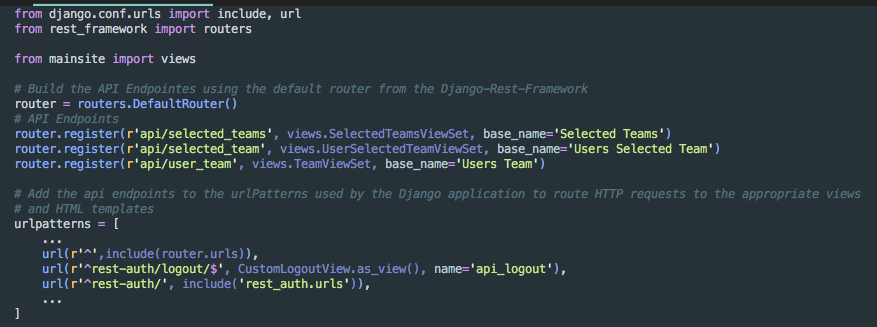
**Excerpt from Views.py**

The above image represents an excerpt from the Views.py file that performs the database query and returns the data as a JSON using the serializer class defined above for the Player model.

It is in the Views.py file that I set the permission\_classes and authentication\_classes for the API endpoints. The player API above will return read only data unless the API request is authenticated by a valid user object.

Similar ViewSet classes were created for the other API endpoints and can be seen in the project code submitted with this report.

The remaining task to get the Web Service API endpoints accessible by the KFL application is to connect them to a URL through the Django application’s URL router file, urls.py.



**Excerpt from Urls.py**

The above image represents all the active API URL endpoints available to the KFL application. These return the view for each API endpoint, which in turn return the JSON through the Serializer class for the API.

These three files can be found in the project directory in the “KFL/mainsite/” directory and are used by the Django application to handle GET, POST and PUT/PATCH requests to the API endpoints defined in urls.py.

#### **Android Application:**

The KFL application shall have a simple and functional design that allows users to complete the required actions as listed in the Functional Requirements quickly and easily.

In order to achieve these goals, the KFL application shall connect to the API endpoints defined in the Web Service and provide a clean UI for user interaction with the application functions.