

Final Project

Intermediate Object-Oriented Programming

Daniel Swain

1.0 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 site using a newly designed and built web service.

2.0 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 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.

3.0 Design and Implementation:

3.1 Pre-existing information:

The **KFL** application will require access to data that is already hosted on 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.

3.2 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.

```

from mainsite.models import Player, ...;
from rest_framework import serializers

class PlayerSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Player
        fields = ('player_name', 'player_team')

```

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.

```

from rest_framework.decorators import permission_classes, authentication_classes
from rest_framework import viewsets, permissions
from rest_framework.permissions import IsAuthenticatedOrReadOnly
from rest_framework.authentication import TokenAuthentication, BasicAuthentication
from rest_auth.views import LogoutView

from mainsite.models import Player, ...
from mainsite.serializers import PlayerSerializer, ...

@permission_classes((IsAuthenticatedOrReadOnly,))
class PlayersViewSet(viewsets.ModelViewSet):
    queryset = Player.objects.all().order_by('player_name')
    # Return serialized data
    serializer_class = PlayerSerializer

```

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.

```

from django.conf.urls import include, url
from rest_framework import routers

from mainsite import views

# Build the API Endpoints using the default router from the Django-Rest-Framework
router = routers.DefaultRouter()
# API Endpoints
router.register(r'api/selected_teams', views.SelectedTeamsViewSet, base_name='Selected Teams')
router.register(r'api/selected_team', views.UserSelectedTeamViewSet, base_name='Users Selected Team')
router.register(r'api/user_team', views.TeamViewSet, base_name='Users Team')

# Add the api endpoints to the urlpatterns used by the Django application to route HTTP requests to the appropriate views
# and HTML templates
urlpatterns = [
    ...
    url(r'^$', include(router.urls)),
    url(r'^rest-auth/logout/$', CustomLogoutView.as_view(), name='api_logout'),
    url(r'^rest-auth/', include('rest_auth.urls')),
    ...
]

```

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.

3.3 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.

The following views are required for the **KFL** application to handle the data returned by the API:

- Article ListView: shows the list of all news articles from the web service. Each article should show a small portion of the full article to minimise the size of each row and
- Individual Article view: show a single Article from the article listview, but show the article’s full text body, expanding on the summary of the article shown in the list view.
- Login View: provides input fields for the username, email and password required by the rest-auth/login API to return an authorisation token for the API’s listed below.
- Roster ListView: shows the list of all the user’s available players from the web service
- Selection ListView: shows the list of all the user’s selected players from the web service
- Selection Edit ListView: allows the user to edit their selections and save them on the web service.

The main activity/view the user sees will be the Article ListView, this will show a loading icon when the initial API GET request is in progress and will change to the Articles list when valid data is returned.

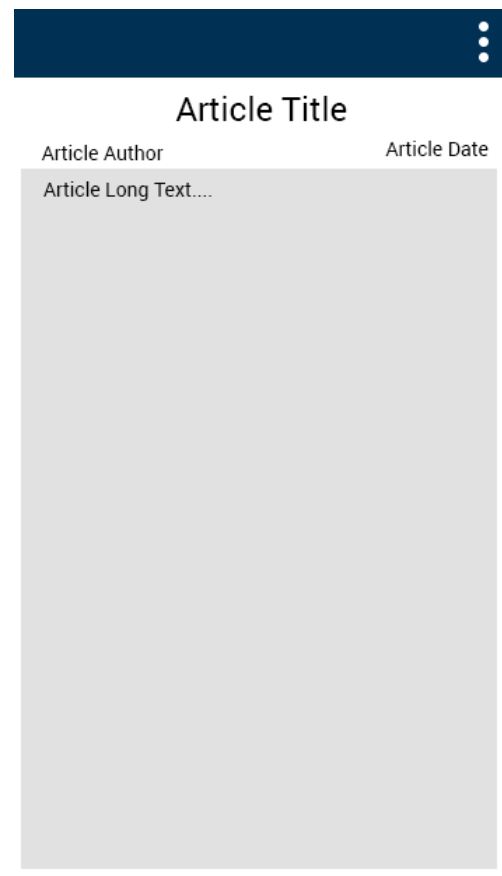
To simplify navigation and to minimise app toolbar clutter, a navigation menu will be utilised and triggered by a hamburger toggle in the nav drawer. This is in keeping with recommended navigation methods in the Android Design guidelines.

Navigation options shall change depending on if the user is logged in or not. When logged in they will see options to view their roster and their selections, as well as log out. When logged out they will see the option to log in. As the articles API can be accessed read only without authorisation, the option to refresh the articles will always be visible.

The following represent the initial mockups for the application.



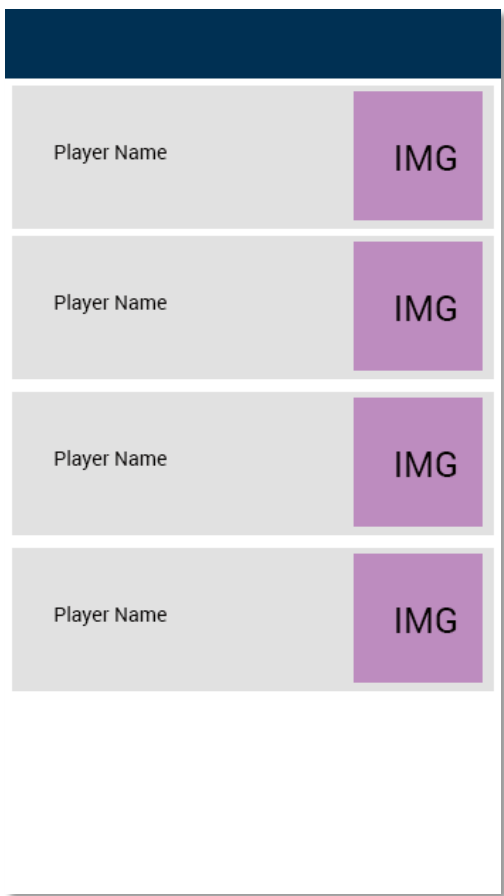
MainActivity Layout



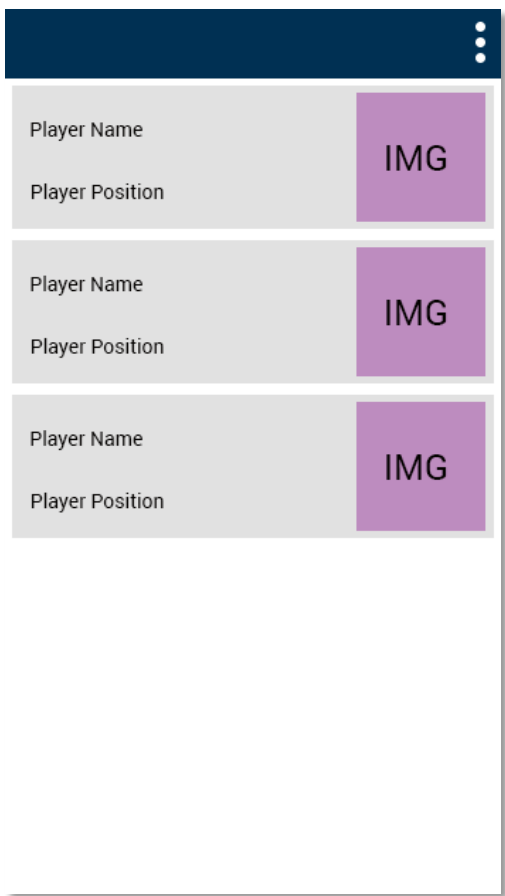
Single Article Layout

The above mockups show the main activity layout with the Article list view and the nav menu hamburger toggle. The image on the right is a mockup of the single article view and features longer article text and an overflow menu to allow the user to open the article in their browser on their phone.

The image in the Main Activity layout represents the article category (one of: news, article, tipping, team of the week, donut review and email archives). These categories are set on the web site and this will help users quickly identify the type of article.



User Roster View



User Selections View

The above two images show the user’s player roster and their team selections. The images on this page represent the AFL team for the player. The User Selection View overflow menu is how the user gets to the edit selections activity.

The image on the left represents the edit selection draft layout. There will be a list of selectors for each of the player positions that a user can select for and a submit button to send the updated selections to the Web Service.

The Player Selection Dropdown will only feature the players from the User's player roster to be shown/chosen.

Selections Edit View

These layouts were built as Activities in Android Studio. To populate the list views and store data from the Web Service requests I build the following helper classes:

- **DatabaseHelper:** Used to connect to the application's SQLite database and add/grab articles, players or selection objects from the database. Other actions include checking for uniqueness and deleting all records in the database.
- **ArticleObject:** An object class representing a single Article object, includes getter and setter methods for the Article properties (title, author, date, summary, long_text, thumbnail, main image).
- **PlayerObject:** An object class representing a single Player object, includes getter and setter methods for the Player properties (player name, player AFL team).
- **SelectionObject:** An object class representing a single Player Selection object, includes getter and setter methods for the Article properties (PlayerObject, player position, player number).
- **ArticleListAdapter:** A custom array adapter to inflate the ListView in the Main Activity layout with the latest articles from the Webservice.
- **RosterListAdapter:** A custom array adapter to inflate the ListView in the User Roster View with the User's player objects from the database and web service (for the latest values).
- **SelectionListAdapter:** A custom array adapter to inflate the ListView in the User Selections View with the User's selection objects from the database and web service (for the latest values).

To perform the web service actions in the background I also built the following Asynchronous Tasks and Helper methods:

- **JSONParser:** This class performs all WebService actions except Logout. It takes the url for the API, the Http method and the postData parameters (i.e. the new selections, or the username, email and password) and connects to the WebService using the Android class HttpURLConnection. Once the connection is completed this class also parses the JSON response from the server into a JSON array and this is sent back to the AsyncTasks below that initiate the web service requests in the background.
- **LoginAsyncTask:** performs the Login call to /rest-auth/login/ with the user's username, email and password via JSONParser. The JSON response from JSONParser includes the user's API Authorisation token on a successful call. This is saved in the application's shared preferences and used for further GET/POST/PUT calls that require authorization (this is so the username and password for the user don't need to be saved).
- **ArticleGetHandler:** This gets the latest articles from api/articles (no authorization required) and then saves the new articles in the database and updates the MainActivity layout with the newest articles at the top.
- **RosterAsyncTask:** This gets the user's player roster from api/user_team/ (using the API Authorisation token from the Login task). The resulting JSON array is converted into PlayerObjects and saved in the database, along with updating the ListView to show the user's new/updated player list. This will remove player's that the user doesn't have any more (i.e. they were removed on the web service). This task is also launched by the Selection Activity to ensure the latest player roster is in the database before enabling the user to perform any selections.
- **SelectionAsyncTask:** This gets the user's selections from api/selected_team/ (using the API authorisation token from the Login task). The resulting JSON array is converted into SelectionObjects and saved in the database, along with updating the ListView to show the user's new/updated selections in the SelectionActivity. This will update the database to include only the current/active selections from the web service.
- **SelectionEditAsyncTask:** This performs the PUT action to api/selected_team/team_id/ by taking the selections from the Selections Edit view and sending the resulting JSON to the server via a PUT action. This updates the Web Service as well as updating the application database to reflect the changes to the selections (only if the server PUT method was successful, otherwise the selections in app won't change as they weren't changed on the server).
- **LogoutAsyncTask:** This performs the log out action on /rest-auth/logout using the user's API token. This DELETE's the API token from the web service, requiring the user to log in again to perform authorised API calls.

With these helpers and asynchronous tasks created the **KFL** application meets the functional requirements as set out in section 2.0 of this report. Note that some asynchronous tasks are private classes inside the activity classes as they don't need to be run by other classes.

For a full view of the code please view the provided Android Studio project folder. I have included extensive documentation for each individual method and action. The above is a summary of the main classes and actions that connect to the web service to meet the functional requirements of the application.

4.0 Testing

4.1 Web Service:

Testing for the web service was completed using the inbuilt web browsable views as discussed in section 3.2 of this report. By building a viewset for each API endpoint it is possible to test each API endpoint on the live site.

To ensure I was not editing selections of any of the live users, I created a test account and test team and performed the following tests on the web service using the browseable API and the Chrome application Postman. I confirmed these tests by using the Admin console I had previously built for the application which allowed me to view web site data live (making it possible to confirm that the changes had been applied in the web site database).

1. Connect to **/rest-auth/login** using POST and provide the test username, email and password.
 - Does an API Token get returned?
2. Connect to **/api/articles** using GET.
 - Does a JSON of all articles get returned?
3. Connect to **/api/user_team** using GET and the Authorisation token in the Http header.
 - Does a JSON of the user's team player roster get returned?
4. Connect to **/api/selected_team** using GET and the Authorisation token in the Http header.
 - Does a JSON of the user's team player selections get returned?
5. Connect to **/api/selected_team/:id/** using PUT and the Authorisation token in the Http header along with the new selections in the Http body as a JSON.
 - Does a JSON of the user's new selections get returned?
 - Does a Success response be sent on successful edits?
 - Does a Failure/Unauthorised response be sent on unsuccessful edits?
6. Connect to **/rest-auth/logout/** using POST and the Authorisation token in the Http header.
 - Does the token get deleted from the Web site database?
 - Does a message get sent back to the Requesting device with the status?
7. Repeat the above tests using invalid data.
 - Does the server notify of the failure?

The Web service passed all the tests above. Sending requests with valid data got the expected response from the server (as listed above) and using incorrect data or a token that had been deleted previously resulted in the server responding with an error message that would be possible to check for in the Android application.

The following pages detail the results from the above tests

Test 1: Login via /rest-auth/login.

Check the credentials and return the REST Token
if the credentials are valid and authenticated.
Calls Django Auth login method to register User ID
in Django session framework

Accept the following POST parameters: username, password
Return the REST Framework Token Object's key.

GET /rest-auth/login/

HTTP 405 Method Not Allowed

Allow: POST, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "detail": "Method \"GET\" not allowed."
}
```

- [HTML form](#)
- [Raw data](#)



The image above shows the web browsable api with the test account details prior to submitting. This method doesn't support GET so there was already a response from the server saying that that method is not allowed.

POST /rest-auth/login/

HTTP 200 OK

Allow: POST, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "key": "eba8d92fbbda7fe10fba7dcdeb02c630dbb5aca7"
}
```

This is the response for a successful call. The user's token is sent in a simple JSON object with the key as the JSON string key. This is very simple for the Android application to get and save in the Shared preference file

Test 2: GET of Articles via /api/articles

Articles List

GET /api/articles/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
[
  {
    "category": "A",
    "title": "DFL Ladder Week 11",
    "image_url": "https://dl.dropboxusercontent.com/u/18672854/DFL/LadderIMG.jpg",
    "thumbnail_url": "https://dl.dropboxusercontent.com/u/18672854/DFL/LadderIMG.jpg",
    "summary": "Click here to view the DFL Ladder after Round 11!",
    "long_text": "<a href= \"https://dl.dropboxusercontent.com/u/18672854/Live%20Scoring/:",
    "author": "Anonymous Reporter",
    "pub_date": "2016-06-07T08:12:51Z"
  },
  {
    "category": "A",
    "title": "Brace Yourself",
    "image_url": "https://dl.dropboxusercontent.com/u/18614642/KFL%20website/bye-weeks.jpg",
    "thumbnail_url": "https://dl.dropboxusercontent.com/u/18614642/KFL%20website/bye-weeks.jpg",
    "summary": "Here come the bye weeks, do you know how they work?",
    "long_text": "Rounds 13, 14 and 15 are the AFL bye rounds. This year, rather than play",
    "author": "Commish",
    "pub_date": "2016-05-30T05:15:23Z"
  },
  {
    "category": "A",

```

The image above shows the successful call to the api/articles endpoint and the JSON array of article JSON objects returned. Again, this can be parsed by the Android application to get individual articles into the database. Also, note that the articles are returned in a sorted order, with the most recent first in the array.

Test 3: GET of User's team via /api/user_team

GET /api/user_team/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
[
  {
    "team_name": "Test Team",
    "players": [
      {
        "player_name": "Aaron Black",
        "player_team": "North Melbourne"
      },
      {
        "player_name": "Adam Cooney",
        "player_team": "Essendon"
      },
      {
        "player_name": "Adam Oxley",
        "player_team": "Collingwood"
      },
      {
        "player_name": "Aidan Corr",
        "player_team": "GWS"
      },
      {
        "player_name": "Billy Evans"
      }
    ]
  }
]
```

A successful response to the api/user_team via an authenticated call. Again, a JSON array is returned with the user's team name, and their player JSON array containing individual player objects for the Android Application to parse and save in the database.

Test 4: GET of User's selected team via /api/selected_team

User Selected Team List

GET /api/selected_team/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

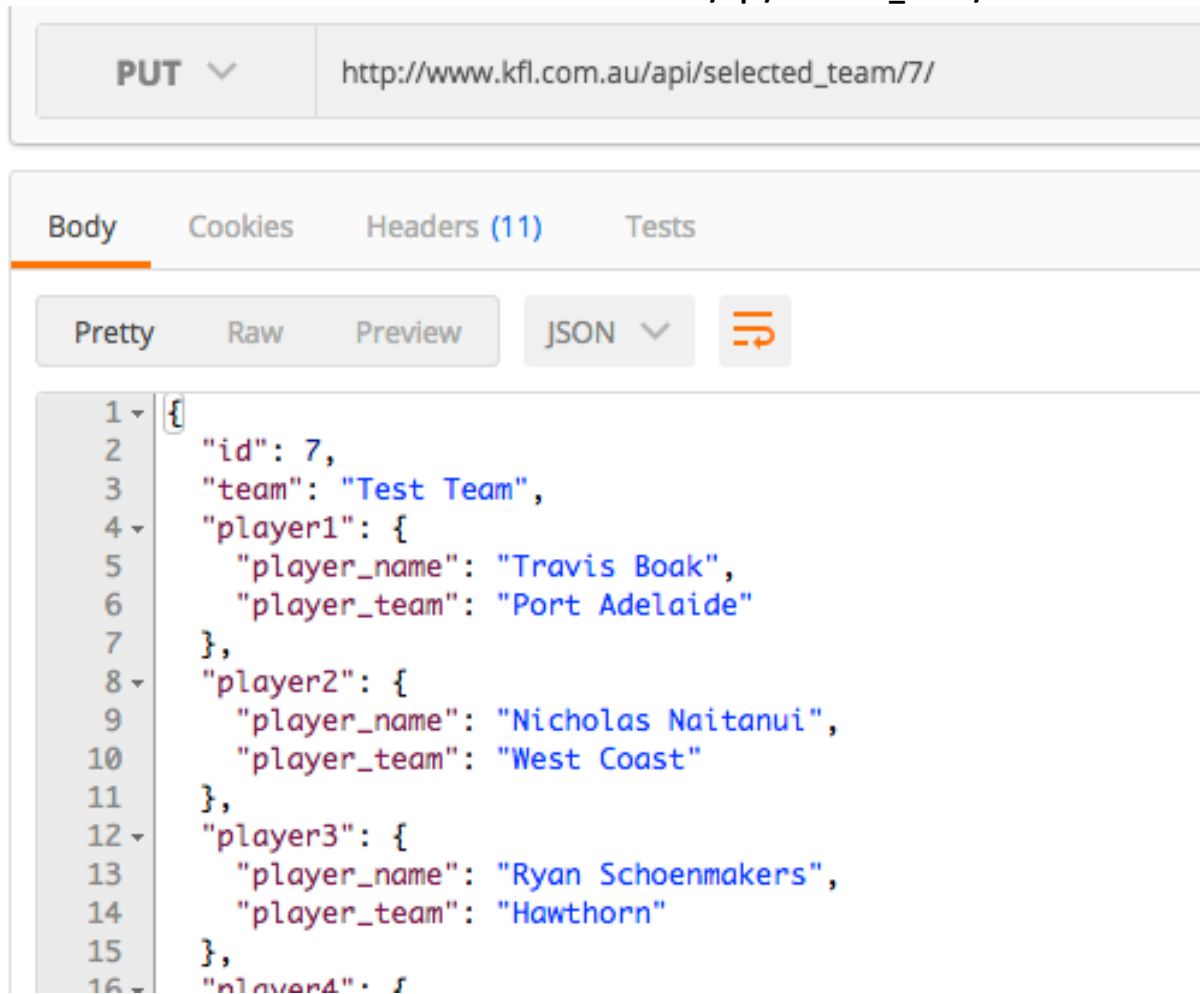
Content-Type: application/json

Vary: Accept

```
[
  {
    "id": 7,
    "team": "Test Team",
    "player1": {
      "player_name": "Nicholas Naitanui",
      "player_team": "West Coast"
    },
    "player2": {
      "player_name": "Travis Boak",
      "player_team": "Port Adelaide"
    },
    "player3": {
      "player_name": "Ryan Schoenmakers",
      "player_team": "Hawthorn"
    },
    "player4": {
```

The image above shows a successful call to the api/selected_team API endpoint. This returns a JSON array containing the user's selected team, the id of that selected team (needed for the PUT action), as well as the players and their positions.

Test 5: PUT of User's selected team via /api/selected_team/:id



The image above is using Postman, but it shows the response from a successful call to the `/api/selected_team/:id` endpoint. This call was done immediately after Test 4 and was used to swap player 1 and player 2.

As the response from the server is the new team list, this can be used to update the selection objects in the application to show the new selection order (or new players).

Test 6: POST to delete user login token via /rest-auth/logout

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** https://www.kfl.com.au/rest-auth/logout/
- Authorization:** Token eba8d92fbbda7fe10fba7dcdeb02c630dbb5aca7
- Headers (1):** key, value
- Body:** Pretty, Raw, Preview (selected). JSON format. The response is:

```
{  "success": "Successfully logged out."}
```

This is the success response from the logout call to the web service. It shows the user was successfully logged out. In the Android application this can be used to determine whether to delete user data or not or to prompt the user to try and log out the user again.

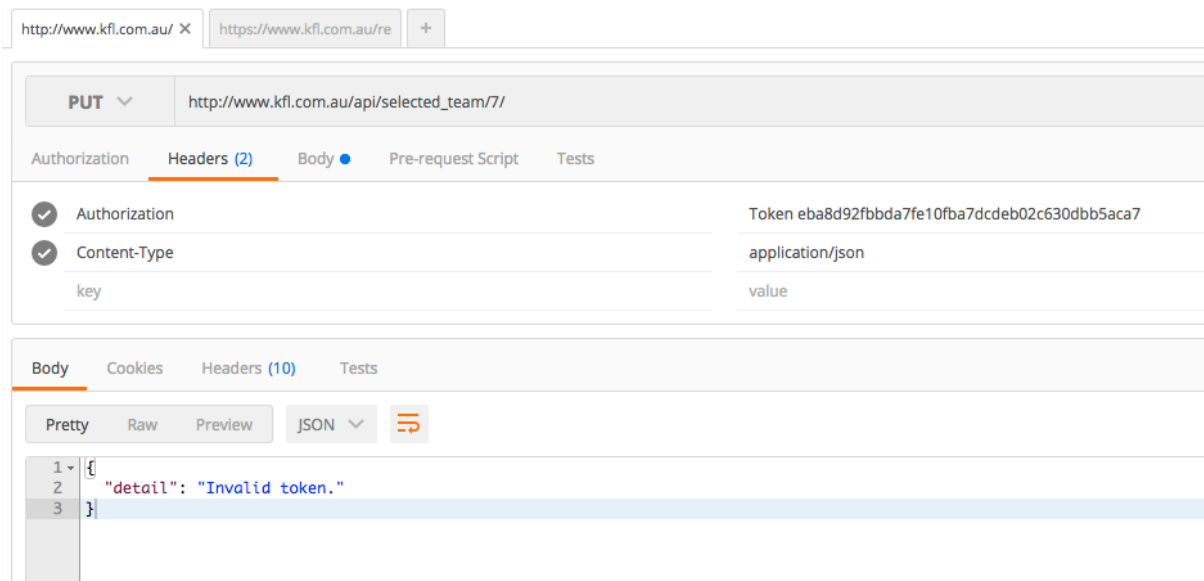
Token to change

----- Go 0 of 2 selected

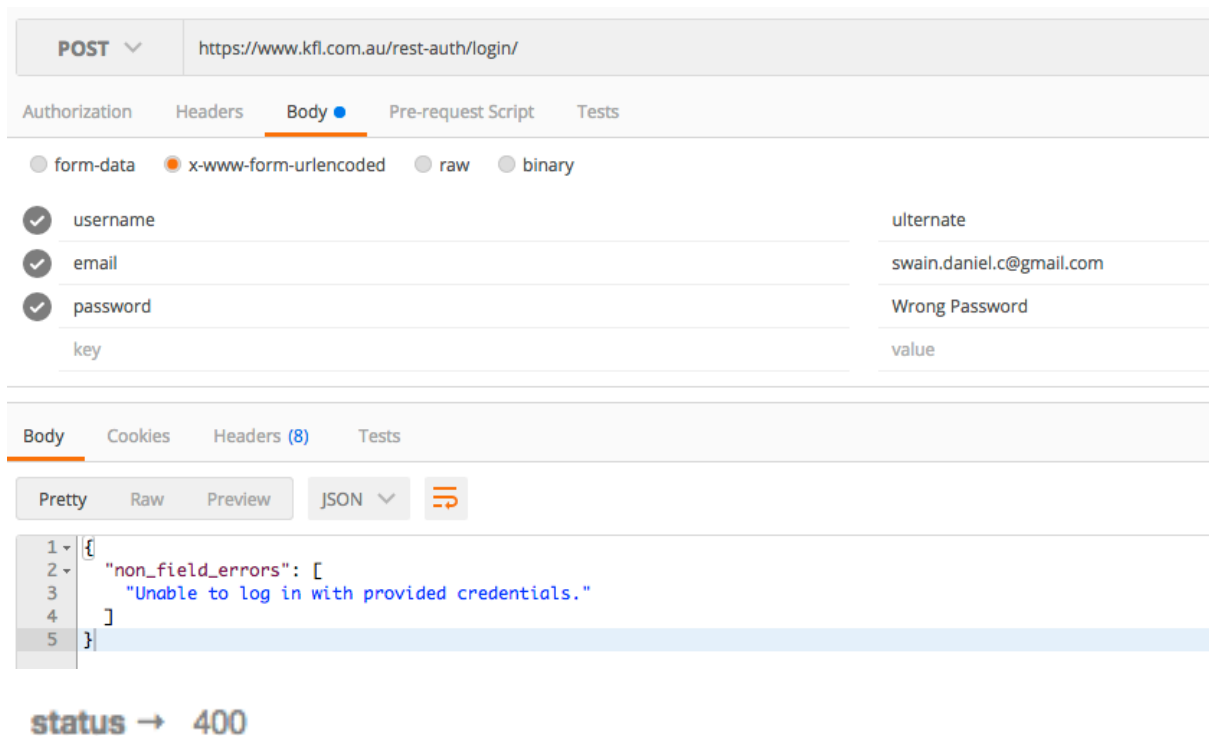
	User	Created
5 [REDACTED]	amobema	June 8, 2016, 2:19 p.m.
[REDACTED]	melkontar	June 7, 2016, 10:09 p.m.

This image shows that the user ulternate no longer has a token (this is a view from the admin console).

Test 7: Testing error response



This image shows what response is sent when an invalid token is used.



A call to `rest-auth/login` with invalid user details will return an error message and HTTP status code 400 Bad Request.

4.1 Android Application:

Two types of testing were completed on the Android application throughout all stages of the project.

These were:

1. Debugging using log messages before and after method calls.
2. Monkey tests: Stress testing the application using an automated testing system built into Android studio.

Stage 1 was a continuous testing process and involved setting log messages around critical activities to determine whether they were performing as expected.

Stage 2 was used close to the end of the project to test the application for any memory issues and stress test the application for extreme events.

Outside of these formal testing regimes I also proactively wrapped my code in try/catch blocks to minimise the occurrence of exceptions such as null pointer exceptions, unsupported character encodings (URL encoding), JSON parsing exceptions and other exceptions that arise from trying to handle HttpURLConnection responses.

The following screenshots demonstrate the logging of the API methods through the debug console in Android Studio. These logs were used to test and confirm assumptions about the connections and the responses from the Web service. All logging was removed prior to submission to the Google Play Store so as not to expose the internal workings of the **KFL** app.

```
D/connection method: GET
D/connection url: https://www.kfl.com.au/api/articles/
D/connection status: 200
D/connection message: OK
```

Successful connection the api/articles, showing the connection message and Http status code

```
D/connection method: GET AUTH
D/connection url: https://www.kfl.com.au/api/user\_team/
D/connection data: token=eba8d92fbbda7fe10fba7dcdeb02c630dbb5aca7
D/connection status: 401
D/connection message: UNAUTHORIZED
W/System.err: at com.android.okhttp.internal.huc.HttpURLConnectionImpl.getInputStream
W/System.err: at com.android.okhttp.internal.huc.DelegatingHttpsURLConnection.getInputStream
W/System.err: at com.android.okhttp.internal.huc.HttpsURLConnectionImpl.getInputStream
```

Unsuccessful connection to /api/user_team as the token is expired. The app doesn't crash due to the surrounding of the HttpURLConnection methods in try/catch blocks (the exception message is included in the screenshot).

```

method: POST
url: https://www.kfl.com.au/rest-auth/login/
data: email=swain.daniel.c%40gmail.com&username=ulternate&password=[REDACTED]
status: 200
message: OK

```

Successful log in of user, also showing the parameterisation of the post data to url encode special characters. The token return was not logged.

```

06-09 11:33:33.910 2396-27352/com.danielcswain.kfl D/connection method: GET AUTH
06-09 11:33:33.910 2396-27352/com.danielcswain.kfl D/connection url: https://www.kfl.com.au/api/selected_team/
06-09 11:33:33.910 2396-27352/com.danielcswain.kfl D/connection data: token=91e423a7ed615d18d0c3f4f2b3312c6e8d7b324f
06-09 11:33:34.589 2396-27352/com.danielcswain.kfl D/connection status: 200
06-09 11:33:34.589 2396-27352/com.danielcswain.kfl D/connection message: OK
06-09 11:33:34.590 2396-27352/com.danielcswain.kfl D/connection resp: [{"id":7,"team":"Test Team","player1":{"player_name":"Travis Boak","player_team":"Port Adelaide"},"player2":{"player_name":"Nicholas Naitanui","player_team":"West Coast"},"player3":{"player_name":"Ryan Schoenmakers","player_team":"Hawthorn"},"player4":{"player_name":"Nicholas O'Brien","player_team":"Essendon"},"player5":{"player_name":"Nicholas Naitanui","player_team":"West Coast"},"player6":{"player_name":"Josh J Kennedy","player_team":"West Coast"},"player7":{"player_name":"Jeremy Cameron","player_team":"GWS"},"player8":{"player_name":"Scott McMahon","player_team":"North Melbourne"},"player9":{"player_name":"Jackson Thurlow","player_team":"Geelong"},"player10":{"player_name":"Scott Pendlebury","player_team":"Collingwood"},"player11":{"player_name":"Nicholas O'Brien","player_team":"Essendon"},"player12":{"player_name":"Scott Pendlebury","player_team":"Collingwood"},"player13":{"player_name":"Tomas Bugg","player_team":"Melbourne"},"player14":{"player_name":"Aaron Sandilands","player_team":"Fremantle"},"position1":"R","position2":"T","position3":"T","position4":"M","position5":"M","position6":"For","position7":"For","position8":"For","position9":"For","position10":"Mid","position11":"Mid","position12":"Mid","position13":"Mid","position14":"FLX"}]

```

Successful GET request to /api/selected_team/. Here the JSONArray returned by the web service is included in the debug log for testing and confirmation that the format is what is expected for the JSON parsing methods in the onPostExecute method call.

```

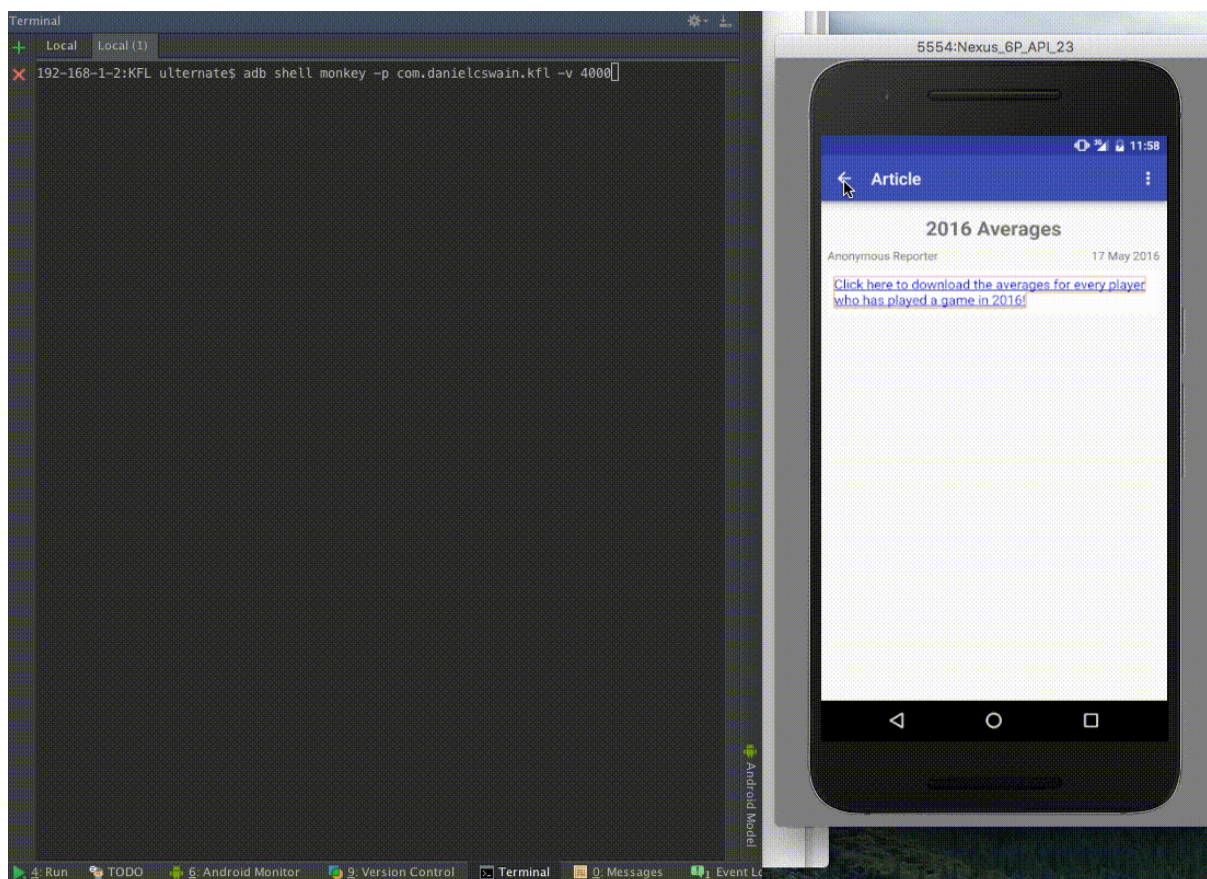
06-09 11:34:05.074 2396-24929/com.danielcswain.kfl D/connection method: PUT
06-09 11:34:05.076 2396-24929/com.danielcswain.kfl D/connection url: https://www.kfl.com.au/api/selected_team/7/
06-09 11:34:05.121 2396-24929/com.danielcswain.kfl D/connection JSON: {"player1":{"player_name":"Nicholas Naitanui","player_team":"West Coast"},"position1":"R","player2":{"player_name":"Travis Boak","player_team":"Port Adelaide"},"position2":"T","player3":{"player_name":"Ryan Schoenmakers","player_team":"Hawthorn"},"position3":"T","player4":{"player_name":"Nicholas O'Brien","player_team":"Essendon"},"position4":"M","player5":{"player_name":"Nicholas Naitanui","player_team":"West Coast"},"position5":"M","player6":{"player_name":"Josh J Kennedy","player_team":"West Coast"},"position6":"For","player7":{"player_name":"Jeremy Cameron","player_team":"GWS"},"position7":"For","player8":{"player_name":"Scott McMahon","player_team":"North Melbourne"},"position8":"For","player9":{"player_name":"Jackson Thurlow","player_team":"Geelong"},"position9":"For","player10":{"player_name":"Scott Pendlebury","player_team":"Collingwood"},"position10":"Mid","player11":{"player_name":"Nicholas O'Brien","player_team":"Essendon"},"position11":"Mid","player12":{"player_name":"Scott Pendlebury","player_team":"Collingwood"},"position12":"Mid","player13":{"player_name":"Tomas Bugg","player_team":"Melbourne"},"position13":"Mid","player14":{"player_name":"Aaron Sandilands","player_team":"Fremantle"},"position14":"FLX"}
06-09 11:34:05.578 2396-24929/com.danielcswain.kfl D/connection status: 200
06-09 11:34:05.578 2396-24929/com.danielcswain.kfl D/connection message: OK
06-09 11:34:05.588 2396-24929/com.danielcswain.kfl D/connection resp: [{"id":7,"team":"Test Team","player1":{"player_name":"Nicholas Naitanui","player_team":"West Coast"},"player2":{"player_name":"Travis Boak","player_team":"Port Adelaide"},"player3":{"player_name":"Ryan Schoenmakers","player_team":"Hawthorn"},"player4":{"player_name":"Nicholas O'Brien","player_team":"Essendon"},"player5":{"player_name":"Nicholas Naitanui","player_team":"West Coast"},"player6":{"player_name":"Josh J Kennedy","player_team":"West Coast"},"player7":{"player_name":"Jeremy Cameron","player_team":"GWS"},"player8":{"player_name":"Scott McMahon","player_team":"North Melbourne"},"player9":{"player_name":"Jackson Thurlow","player_team":"Geelong"},"player10":{"player_name":"Scott Pendlebury","player_team":"Collingwood"},"player11":{"player_name":"Nicholas O'Brien","player_team":"Essendon"},"player12":{"player_name":"Scott Pendlebury","player_team":"Collingwood"},"player13":{"player_name":"Tomas Bugg","player_team":"Melbourne"},"player14":{"player_name":"Aaron Sandilands","player_team":"Fremantle"},"position1":"R","position2":"T","position3":"T","position4":"M","position5":"M","position6":"For","position7":"For","position8":"For","position9":"For","position10":"Mid","position11":"Mid","position12":"Mid","position13":"Mid","position14":"FLX"}]

```

Finally, this image shows the PUT method call successfully working to /api/selected_team/:id/. The initial JSON object is the post data sent in the connection body to the web service whereas the final JSON array is the response from the server confirming the new selection choices.

For the monkey testing ADB is required as it is built into the adb suite. The monkey test suite will send device keypress events inside the emulator (or physical device) and trigger random actions of the application repeatedly.

This stress testing doesn't simulate real world use cases for the application, as thousands of keypress events are sent in short succession, but is useful to automate testing of the app actions. Any action triggered by the monkey tester that has a critical bug or edge case will break the test and be shown to the developer. Coupling monkey testing with details logging and exception catching can increase confidence that the Android application has been tested extensively, with edge cases and potentially critical bugs fully explored.



The screenshot above is a single image from the gif available at <https://dl.dropboxusercontent.com/u/37115861/monkey%20test.gif>. It shows a single monkey test launched by the following command:

- `adb shell monkey -p com.danielcswain.kfl -v 4000`

where `-p` tells the tester what package to test, `-v` for verbose logging and 4000 for the number of device events to send (touch, fling, scroll, screenshot, click, keyboard...).

The only failure in the above test was at the end with the Emulator's browser app crashing. No crashes for the tested app were recorded.

5.0 Publishing

The web service was live immediately as it was added to the existing www.kfl.com.au website hosted via www.pythonanywhere.com. Due to the application sending user information to the web service a SSL certificate was obtained through CloudFlare.

The **KFL** application was uploaded to the Google Play Store on the 28th of May 2016 with only the api/articles endpoint functional in version 1. The latest version (1.3) uses the HTTPS endpoints, added user login and logout, authorised API endpoints such as api/user_team and api/selected_team/(id). This update went live on the 7th of June 2016.

The **KFL** application live on the Play Store ([play store link](#)) meets all the functionality requirements as listed in section 2.0 and repeated below:

- Connect to www.kfl.com.au and bring in the latest articles from the website.
 - Done, the main activity of the application lists the latest articles.
- Implement a single article view for each article.
 - Done, clicking on a single article in the list opens an article activity with that article.
- Allow **KFL** competition entrants to login to authorise the following actions:
 - retrieval of the user's player/team roster (Done);
 - retrieval of the user's selected team for the upcoming rounds (Done); 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). (Done).

Initial reviews from users are positive, with the following review from a **KFL** user coming the day after launch.

REVIEWS

 Download

Average rating	Total ratings	Ratings with reviews
5.000 ★	2	1


REVIEWS




All ratings ▾ All languages ▾ All versions ▾ Add device filter

Yesterday

Sort by most recent ▾



Brad downing 8 Jun 2016 at 12:28  1  0

★★★★★

Brilliant Amazing app, allows me to monitor my fantasy football team without having to load the website. I'm sure it will only get better over time as well!