**NBA App Tech Documentation** by Andres Puga

**Table of Contents**

# [Using given Portfolio as skeleton](#_Portfolio_Documentation)

# Link to Documentation

# [Changes done to skeleton](#_Changes_to_UI)

# [NavBar Customization](#_Navbar)

# [Background Color](#_Background_Color)

# [Image Implementation](#_Image_Implementation)

# [Widget Implementation](#_Widget_Implementation)

# [Populating NBA Team pages](#_Populating_NBA_Team)

# [Adding posts](#_Adding_Posts)

# [Updating Posts](#_Updating_Posts)

# [Deleting Posts](#_Deleting_Posts)

# [Account Information](#_Account_Information)

# [Registering](#_Registering)

# [Logging In](#_Logging_In)

# [Logging Out](#_Logging_Out)

# [User Permissions](#_User_Permissions)

# [Project Testing](#_Project_Testing)

# [Unit Testing](#_Unit_Testing)

# [Selenium Testing](#_Selenuim_Testing)

# **Portfolio Documentation**

Throughout the semester, we have built a student portfolio app that creates student profiles with projects. I have used that as my skeleton for this project, as I believe most people have.

I have built documentation for the beginning of this project here: <https://drive.google.com/drive/u/3/folders/11ZLkEUvtwSeWu2jeficveMQy-yQ2KIXP>

# **Changes to UI**

# Navbar

The look of this web app was designed towards basketball fans, so colors that can resonate with the sport are crucial. A big part of this web app is the navbar, as it will let you travel to each specific team page and back home.   
A close-up of a black and white background

Description automatically generated

Something simple that is easy to pick out is a must, and we went with a dark navbar design to contrast with the background color of the web app. We achieve this navbar design through the base template page with the following code:

A computer screen with text on it

Description automatically generated

# Background Color

An orange background was chosen to be both recognizable and contrast the navbar.

A basketball going through a hoop

Description automatically generated

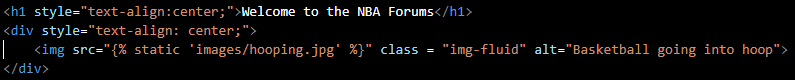
It really helps represent the color of a basketball, which is the central piece to the sport. We also achieve this through the base\_template so that every html page can inherit this background color:

A black background with white text

Description automatically generated

# Image Implementation

Images are important to visualize actions and emotions, and although there are not a lot in this web app, they are crucial to implement for user retention. The image of the web app above shows 2 images in action. The first image on the navbar is implemented through the base\_template so every page has it, but the second image is posted only in the home page with the following code:



A div is used to center the image and will remain that way regardless of screen size thanks to the viewport in the base\_template. All images are stored in a folder within the project named “images” and you can path to which image you want specifically.

# Widget Implementation

The groovehq blog defines a widget as “a component that can be added toa website or application as a stand-along feature.” For my web app, I decided that information is needed to make a proper post about a team. So I went to a sports website (<https://developer.sportradar.com/docs/read/widgets/NBA_Widgets#standings> ) and took their html code to implement onto my main home page:



A screenshot of a computer

Description automatically generated

# **Populating NBA Team Pages**

# Adding Posts

To add posts to the web app, 3 things need to be done:

The first one is that a function needs to be implemented in our views.py in the app that will let the user make a post:

A computer screen shot of text

Description automatically generated

The second thing is that a url needs to be created within the urls.py so that when we click the new post, we can be redirected to the create a post page.



Finally, the html page needs to be created so that when we are redirected, there is actually a page to create the post.

A computer screen shot of a program code

Description automatically generated

With all that, we get this result:

A black and orange rectangle

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Updating Posts

As seen in the last image, there is an update post button. With this, you are able to go into a post created and update the contents of it. To implement this, 2 steps need to be taken.

The first one is to create another function in the views.py that will let someone update the post:

A computer screen shot of a program code

Description automatically generated

And secondly, the urls.py needs to be updated so that he button takes you to the page to update:



We do not need to make a new html page, we will be using the same one as the one used to create a post. We populate it with what the post has and we can change things from there:

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

# Deleting Posts

And the final button in that last screenshot is the delete button, which deletes the posts within the page. We go back to the 3-step process for this function:

First, we must create another function that will process the deleting of the post in the views.py:

A computer screen shot of text

Description automatically generated

Then, we need to update urls.py to take us to a delete page when we click the delete button:



Finally, we do need a new html page that will confirm the deletion of the post we selected:

A computer screen with text and images

Description automatically generated

And we can finally delete our posts:

A red and white sign with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

# **Account Information**

# Registering

To add registration to the web app, we must do 3 things:

First thing is to create a new form in forms.py that will handle the user registration form:

A computer code on a black background

Description automatically generated

Next, you will create a function in the views.py that will actually let us upload the user registration onto the database:

A computer screen shot of a program

Description automatically generated

Finally, we create an html page that will hold the registration form:

A screenshot of a computer

Description automatically generated

A computer screen shot of a program code

Description automatically generated

# Logging In

In order to log into the account registered, we need to create 3 things for it:

We will need to create another form in the forms.py that will handle the authentication form for the logging in:

A black background with white text

Description automatically generated

Second, we will need to create a function in the views.py that will not only allow the logging on, but also use this form to authenticate that the account exists:

A computer screen with text on it

Description automatically generated

Finally, we will create a log in html page that will serve as the interface for the user:

A screenshot of a web page

Description automatically generated

A screenshot of a computer program

Description automatically generated

# Logging Out

Logging out is actually very easy and straightforward. You only need 2 things:

The first one being a function in the views.py that you can customize to redirect where the log out will take you:

A screen shot of a computer program

Description automatically generated

And the second is to implement the users log in and log out onto the base\_template for the whole web app to use:

A computer screen with text

Description automatically generated

# User Permissions

For security reasons, we must implement user permissions for posting, updating and deleting posts in the NBA forum. We accomplish this by creating a wrapper that will handle the checking of the app and implementing it into our views.py:

This is how the wrapper should look:

A screen shot of a computer program

Description automatically generated

And this is how we implement it to the posting, updating, and deleting functions in our views.py. They are added at the top of each of the functions:



With that, you will need to be logged into an account in order to create, update or delete a post in the forum.

# **Project Testing**

# Unit Testing

Creating unit test for any part of the program is crucial to verify that the program runs correctly. We accomplish this by using the django implemented test.py and creating our functions in there:

A black screen with white text

Description automatically generated

A computer screen shot of text

Description automatically generated

This is a model test, but we can also do a views test. These 2 .py files are the most important to check. We can finally run python manage.py test to run the tests.

# Selenuim Testing

The Selenium WebDriver is a framework that will let anyone execute cross-browser tests to check things like user login and registration. We are using it in this web app to test the user login functionality and buttons. We accomplish this by adding the selenium webdriver onto our project through pip install:

$ pip install selenium

Afterwards, we import the selenium webdriver onto our test.py file to let us execute the tests:

A computer screen with text on it

Description automatically generated

Finally, we create the function that will test what we are looking for and run the tests the same way we do unit testing with python manage.py tests:

A computer screen shot of text

Description automatically generated