F.C. Team Possible

Project: World Cup 2014 website

Group Members:

Aseal Yousuf

Jeremiah Martinez

Eros Smith

Kim-yu Ng

Prithvi Shahi

Jungmo Ku

Table of Contents

Problem

Around the world association football, or soccer, has millions of ardent fans and admirers. Despite the need, these millions (even billions) of soccer lovers don't have a single website where all the latest information is gathered and easily accessible. There is a lack of rich media content including video, twitter, stats, and events of the current world cup all collected in one place, especially in english speaking countries such as the United States. At the moment, the average soccer fan has to bounce around many different sites to get the latest information they want, such as match highlights, their favorite player's tweets, and player's statuses. Even though some sites attempt to address this need, there is a lack of design consideration; The currently existing websites lack natural and intuitive interfaces to make all available media content easy to access.

Use Cases

The site's interface allows navigation from a particular point of interest such as a favorite player to a certain match, from one country to another, all the while enjoying the content they desire, because each web component is interlinked with another. In fact, any twitter feed, stat, or video connected to the object of interest is on the site, at most, four clicks away. This eliminates the need of scouring Youtube for the game or player highlights or needing to search twitter to see what players are tweeting. Because of its design innovation, this website will bring together all of the most desired media content of today's World Cup Soccer fans and provide them access to any article of interest quickly at the same time saturating them with other interesting content, facilitating hours of soccer bliss.

Abstract

This report describes the initial stages of building a responsive and attractive website using Django, Bootstrap, CSS, and javascript. This iteration will contain the how-to of setting up the basic framework for the site's layout, developing the API, and working models. The result will yield a static "demo" version of the website with which to share, modify, and experiment with before committing to building the fully dynamic website.

Required technologies

Django

Django is an open source, Python based web-framework. At later iterations, Django delivers the dynamically driven content of the site. In this iteration, however, Django only serves up static pages that mock up the intended final design. Even though they are not yet put into use, the models for the database being delivered are both generated and tested.

1. Setting up Django

Installing Django

The framework of the website utilizes version 1.6.5 of Django because it is the most stable iteration released thus far. Using pip3, Django can be installed in bash with the command:

sudo pip install django==1.6.5

(Other options can be found at https://docs.djangoproject.com/en/1.6/intro/install/#install-django).

Starting a New Project

A new project can be started using the command:

django-admin.py startproject project_name

After this command a new folder with *project_name* should be in the current working directory. Contained within that directory should be another folder with the name *project_name* and a file named *manage.py*. The *manage.py* file runs commands for Django and should not be edited. Inside the second project_name folder there should be four files: *__init__.py, settings.py, urls.py,* and *wsgi.py.* The *__init__.py* file is there just to let Python know that this a Python folder. The *settings.py* and *urls.py* files will be edited during the set up of the static site. The *wsgi.py* file will be used in the configuration of hosting the site on PythonAnywhere.

At this point it will be possible to run a private server at *http://127.0.0.1:8000* using the command:

python3 manage.py runserver

Installing Additional Packages

Next, some additional packages should be installed to help in later production phases using the command:

pip install south django-registration stripe

Then, create a new app using the command:

python3 manage.py startapp app_name

This should create a new folder named *app_name* with files *__init__.py, admin.py, models.py, tests.py*, and *views.py.* Add the string *'signups'* to the the tuple labeled *INSTALLED_APPS* in the *settings.py* in the first folder named *project_name.*

2. Setting Up the Django Database for the Models

Setup MySQL Environment and Installation Guide

The operating system used is Ubuntu 14.04 and the command to install the MySQL connector from Oracle is:

sudo pip3 install --user https://dev.mysql.com/get/Downloads/Connector-Python/mysql-connector-python-1.1.6.tar.gz

Following this, MySQL client has to be installed. The MySQL website details the instruction specific to the three different major operating systems. Next, the *settings.py* file has to be updated to use the oracle Django backend. This changes the default Django database engine from sqlite to MySQL so that the model can interact with a server backed by a MySQL database.

Create Database Scheme from MySQL through terminal

In order to access MySQL as the root user, the command:

mysql -u root -p

has to be run through command prompt terminal. If there are permission problems with running the previous commands with password, the root user privilege of the MySQL has to be flushed.

Grant user access to the database

A new user has to be created for the specific database and the user has to be granted access. This is done by the following commands which are entered in the MySQL interactive command prompt.

```
CREATE USER ‘user’@’localhost’ IDENTIFIED BY ‘mypass’;
CREATE DATABASE any_name;
GRANT ALL ON any_name.* TO ‘user’@’localhost’;
```

Configure the database settings in Settings File

With the newly created user, the Django *settings.py* file has to be modified to recognize the changes made to MySQL i.e. set the location of which database is to be refered to and the data will be stored on that newly created database. This allows the Django model to interact with MySQL when the user is trying to perform data insertion, deletion or modification. The following is an example of the *settings.py* format:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mydb'),
        'USER': 'nick',
        'PASSWORD': 'mypass',
        'HOST': 'localhost',
        'PORT': '3360',
        }
}
```

Create the designed model from *model.py*

Following these steps, the hypothetical model becomes realized and all the tables are created from model design.

```
python3  manage.py syncdb
```

Load required data with Python scripts

Instead of inserting player, county, and match information by hand, an automated script was created and fed data encoded as a dictionary.

Obtain the data in the JSON format first and then process the data i.e. take all necessary bits of information and reformat it in a dictionary format. This reformatted data is also stored in a .json format with a .txt file extension.

python3 manage.py shell < country_insert.py

This is an example command of how to feed in Python script in order to insert country information into the tables.

data: the "country_insert.py", "player_insert_script.py", "match_insert_script.py"

The above step is repeated with this command.

localhost:8000/admin

All the data and information inserted can be inspected through the administrative panel.

3. Building tests for the models using Django unit tests

The required imports:

Add these imports to the *test.py* file:

```
import json
from django.test import TestCase
from name_of_app.models import *
import json #is needed to read the json files that is fed for testing. from django.test import TestCase #is needed so that the unit tests are able to run properly lastly from name_of_app.models import * #are needed to import the models for testing.
```

Make a class for the model tests

An example of a class created in *test.py* :

```
class ModelTestCase(TestCase):
        def test_country_model(self):
                #Dictionary Key: Country Name
                #Dictionary Value: [Country_code, country_rank]
                country_test_dict = {"Brazil": ["BRA", 5]}
                Country.objects.create(country_name="Brazil",
                country_code=country_test_dict1["Brazil"][0],
                rank = country_test_dict1["Brazil"][1]facacd
```

```
Country_Brazil = Country.objects.get(country_name="Brazil")
self.assertEqual(Country_Brazil.country_name, "Brazil")
self.assertEqual(Country_Brazil.country_code, "BRA")
self.assertEqual(Country_Brazil.rank, 5)
```

The test class above creates a new country object using the model by the following code:

```
Country.objects.create(country_name="Brazil",
country_code=country_test_dict1["Brazil"][0],rank = country_test_dict1["Brazil"][1]).
```

The parameters taken in by the country model is the country name, country code and rank. The `Country.objects.get(country_name="Brazil")` code is used to retrieve the country object. Asserts are used to test whether the data inputted to the database are correctly returning the data that is expected. All the nine tests included in the *test_model.py* almost follow the exact pattern of testing. Three of the tests, instead of taking input data from a hard coded dictionary, take in data from a file.

The tests that are checking foreign keys must make a foreign object first and then pass it in as a parameter for the model that is using the foreign key otherwise the test will fail since the model is expecting a the foreign key object. For example:

```
def test_player_model1(self):
        #Dictionary Key: Player full name
        #Dictionary Value: [sur_name, country,Clubname,Position,Birthdate]
        player_test_dict1 = {"Andrea Barzagli": ["Barzagli", "Italy", "Juventus FC", "Defender",
        "1981-05-08"]}

        Country.objects.create(country_name = player_test_dict1["Andrea
        Barzagli"][1])
        c1 = Country.objects.create(country_name = player_test_dict1["Andrea
        Barzagli"][1])

        Player.objects.create(country=c1, sur_name= player_test_dict1["Andrea
        Barzagli"][0],full_name = "Andrea Barzagli" ,clubname =
        player_test_dict1["Andrea Barzagli"][2], position = player_test_dict1["Andrea Barzagli"]
        [3], birth_date =player_test_dict1["Andrea Barzagli"][4])

        player_get = Player.objects.get(full_name = "Andrea Barzagli")
        self.assertEqual(player_get.country.__str__(),
        player_test_dict1["Andrea Barzagli"][1])
        self.assertEqual(player_get.sur_name, player_test_dict1["Andrea Barzagli"][0])
        self.assertEqual(player_get.full_name, "Andrea Barzagli")
        self.assertEqual(player_get.clubname, player_test_dict1["Andrea Barzagli"][2])
        self.assertEqual(player_get.position, player_test_dict1["Andrea Barzagli"][3])
        self.assertEqual(player_get.birth_date.__str__(), player_test_dict1["Andrea
        Barzagli"][4])
```

The country object is being created first then passed in as a parameter for the player model.

One thing to note is the usage of the .__str__() in the context of the country foreign key. This is used so that the name of the country is returned as a string instead of the country object being returned. The .__str__() function in the context get.birth_date is used because the format in which the date is saved is different then the string of the birth date.

Running the tests

Since the database is backed by MySQL, in order to run the unit tests, one has to setup the model mentioned above. The unit test is located at the path: *world_cup/wc_app/test_Model.py*

The following is the command to run the test:

python3 manage.py test

4. Getting Django to Serve Static Files

In order use bootstrap as a frontend, Django must first be set up to serve CSS, JavaScript, and images. Inside *settings.py* add this method at the end of the file:

```
if DEBUG:
    MEDIA_URL = '/media/'
    STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR),"static","static-only")
    MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR),"static","media")
    STATICFILES_DIRS = (
        os.path.join(os.path.dirname(BASE_DIR),"static","static"),
    )
```

Inside *urls.py* include the following imports:

```
from django.conf import settings
from django.conf.urls.static import static
```

Also, add the following code to the end of *urls.py*:

```
if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL,
                          document_root=settings.STATIC_ROOT)
    urlpatterns += static(settings.MEDIA_URL,
                          document_root=settings.MEDIA_ROOT)
```

In the root folder run the command:

python3 manage.py collectstatic

Respond yes to the prompt.

Next setup Bootstrap.

Bootstrap

Bootstrap is a open source front-end framework that provides responsive web-design with little to no effort while offering customizable elements and templates.

1. Setting up Bootstrap

In order to make a sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development, the up-to-date Twitter bootstrap css v3.2.0 was included. Go to http://getbootstrap.com/ and download the code available on the site and include it in the project folder. Then set a path in the base HTML file so that it enables the use of bootstrap.

jQuery library for bootstrap dropdown

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.0/jquery.min.js"></script>
```

Twitter bootstrap css in minified form

```
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
```

Twitter bootstrap default javascript

```
<script src="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script>
<script src="/static/js/bootstrap.min.js"></script>
```

2. Getting Templates

Templates are designs used in designing web pages. To find a template search on Google and choose a template. There are many templates available as open source on the web for bootstrap.

3. Customizing templates using CSS

Include a *custom.css* file to override current css attributes. When using a web browser like Firefox there is an inspect element option which allows the CSS to be viewed and edited within the browser. By selecting the appropriate code, editing it, and placing it in the *custom.css* file the default format can be overridden.

Apiary

Apiary provides a convenient way to mock up a RESTful API. Using the in-site API Blueprint Editor, a mock up can be generated, documented, and shared with team members.

1. Setting up Apiary

Go to www.*apiary.io* and sign in with the project's git hub account.

2. Creating a mock up

Use the Apiary site's embedded editor to create and modify the API mock up.

3. Resulting API

GET /api/countries

Lists the countries in the database

GET /api/countries/{id}

Displays all the details of a specific Country object

GET /api/ players

Lists the players in the database

GET /api/players/{id}

Displays all the details of a specific player object

GET /api/matches

Lists the matches in the database

GET /api/matches/{id}

Displays all the details of a specific match object

PythonAnywhere

PythonAnywhere is a web hosting service as well as an online IDE based on the Python programming language. This acts as a server to host the site. PythonAnywhere has the resources and the environment to setup Django application as well as CSS bootstrap.

1.Setting up PythonAnywhere

Create a new PythonAnywhere account. To clone the github repository on PythonAnywhere go to the bash console. Then at the root directory use the git clone command and drag and drop the clone link on the console to clone.

2. Hosting the site

To allow the site for public access, go to dashboard and under the web tab, make a new web application. Then, there will be a configuration popup, choose manual configuration and choose the corresponding Python version. In this project, the project is setup to environment with Python 3.4.1 support. Next, change the settings on *wsgi.py* as they came with default flask settings which is not related to the project.

```
import os
import sys
if path not in sys.path:
        sys.path.append(path) os.environ['DJANGO_SETTINGS_MODULE'] = 'mvp_landing.settings'
import django.core.handlers.wsgi application = django.core.handlers.wsgi.WSGIHandler()
```

The above is an example of how to link to the project repository that has just been cloned in PythonAnywhere so that the hosting site can recognize where the project is located at. One should change *mvp_landing.settings* to his or her Django application path respectively. The reason to use manual configuration instead of opting for Django configuration on PythonAnywhere is because the default Python, bundled with the Django settings, might require more modification to the project repository as compared to manual configuration of Python that does not serve with the Django module. The reason to do it this way is to publish the Django

application from PythonAnywhere as it was like running the project on localhost without having the need to touch the source files.

Design Choices

Static Web Design

In this project, there are fourteen url patterns in *urls.py* and fourteen functions in *view.py* for each static webpage: one is for the home page (splash page), four pages are for countries, four pages are for players, four pages are for matches, and the last one is an “about us” page. The website has a navigation bar that links to the homepage, the “about us” page and dropdowns for countries, players and matches. The homepage has three links that link to a page containing the countries, a page containing all of the players, and to a page that contains all of the matches.

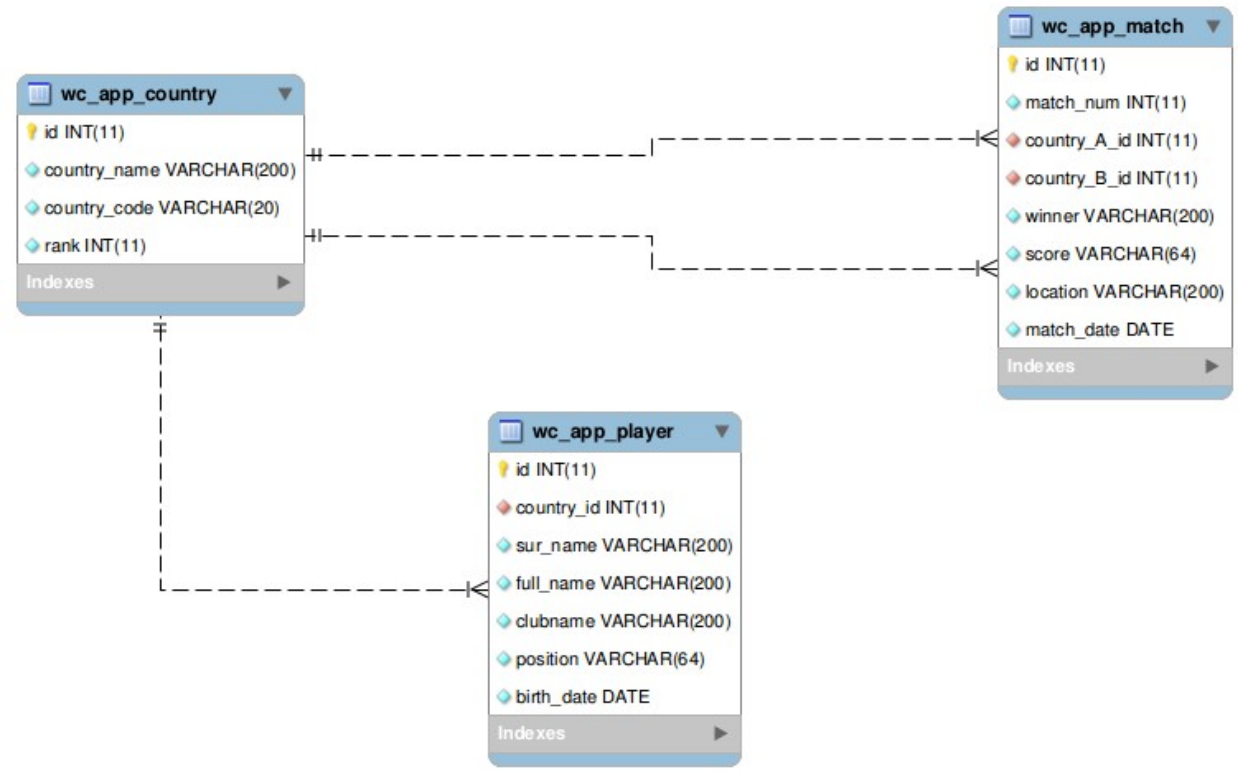
The country page currently contains three countries Brazil, Argentina, and Germany. On each country page there is an embedded Google Map with the country location, Twitter feeds, and the list of players for that nation. In this iteration, there is only one clickable link to a player’s details from the team list. In later iterations, all the players will be linked to their details from the player list since the information will be generated dynamically through interaction with Django model that is incorporated in this web application.

Like the country pages, the player pages also have personal Twitter feeds embedded into the page along with some of their personal and team information, and a photo of themselves. The match pages include information such as score, an embedded Google Map showing the location of the stadium of where the match took place, and an embedded video from ESPN. The about us page consists of the group members names, images, a little information about the group and a soundcloud embedded audio to serve as the theme song of the team.

Django Model Design

The Django models represent the three main subjects of the site. The country model, the player model, and the match model. The country model consists of the country name, a country code and the rank of that country. The player model consists of a foreign key that comes from the country model that links the player to that specific country. The remaining fields contain the player surname, full name, the club he/she plays for, the position he plays and his birthdate. The matches model uses two foreign keys from the country model. This is done to link the two countries playing against each other. The other fields consist of the match number, the winner of the match, the scores of the match, the location where it was held and when it was held.

The Django model generates default primary keys by assigning them to the created id field as integers. The following model view shows the foreign keys on the match table as country_A_id and country_B_id respectively. It makes sense to design the model to have the country as the foreign key in the match table because every single match will have a home team and an away team. By doing the object can be used to access all the attributes from the country class. For example, Brazil is playing against Germany and a user wants to know about the ranking of Brazil in FIFA ranking, he or she can get the rank easily as the rank variable can be accessed through the match model. Same thing goes for player table; every player is associated with a country.



Conclusion

While the implementation for this project is specifically about designing a website that provides information for the current FIFA World Cup, the principles and design could be directly applied to almost any current event of public interest, and more generally to any website that could benefit from scalable and dynamically generated content.