BOĞAZİÇİ UNIVERSITY

SWE 573

www.spotweety.club

An Online Twitter Analyzer for the Popular Songs Shared via Spotify

23 May 2017

Fatih Aracı

Instructor: Suzan Üsküdarlı

Project Documentation

Table of Contents

A. Requirements Specification

1.1 Functional Requirements

Requirements specification for SpoTweety project involves functional and non-functional requirements as listed below.

1. User Accesability
    1.1. Any user shall be able to access the web page. No login required.
2. Top Songs Search Operations
    2.1. User shall enter location info as coordinates.
    2.2. User shall be able to select location in order to provide coordinates.
    2.3. User shall be able to see trend of songs for a given time-line.
3. Collected Data Operations
    3.1. System shall be able to analyze any tweet collected.
    3.2. System shall be able to count number of occurances of songs on tweets.
    3.3. System shall be able to compare the occurances of songs in order to create top 10 songs.
4. Results
    4.1. User shall be able to see the top 10 popular songs as a list.
    4.2. User shall be able to listen the songs directly from the list.

1.2 Non-functional requirements

1. The system shall be both resource-efficient and scalable. That is, as the number of users increases, the system, by adding new resources, should work as if there is only one user. This scalability should be satisfied in a distributed manner.
2. The system should be usable. In other words, users should not need to be trained to use the system.
3. The system should adopt and follow certain quality standards such as W3C Html/XHtml Markup Standards, most common coding/documenting styles(for instance PEP8 for Python)

4. The system shall have responsive user interfaces so that it can be more accessible. It shall support modern web browsers such as Firefox or Chrome and their corresponding mobile versions.
5. The system shall use and contain only open source technologies, libraries, and tools.
6. The system shall be fast enough not to bother its users. Any user interaction should not take longer than 3 seconds in a local development environment, except 3rd party API transactions.
7. The system shall expose an HTTP RESTful API which supports every user interactions which can also be done via the user interfaces of the system. This API shall have complete documentation.
8. The system shall have unit tests and functional tests for the back-end side. Test coverage shall be at least 70%
9. The system shall be able to be deployed in a virtual environment which is completely isolated from operating systems.

B. Software Design Documents

This section contains use case scenarios, UML (Unified Modeling Language) design diagrams and user interface mockups for the designed system.

2.1 Use Case Scenarios

2.1.1 Search Twitter for Popular Songs according to Location

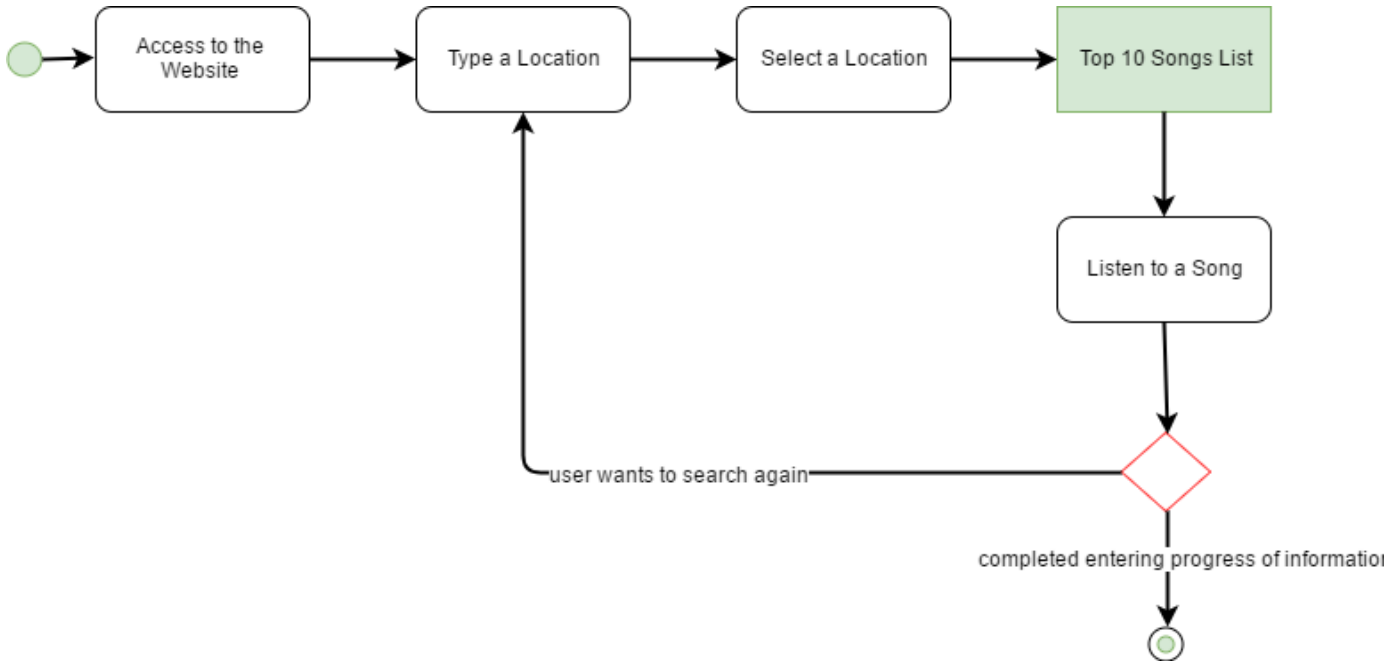
Use Case	Search Twitter for Popular Songs according to Location
Actors	User
Description	User select a location in order to see popular songs around that city.
Flow of events	1. User enters a word to location search box 2. Possible cities and countries around the world are recommended; user selects one of them. 3. User clicks on search button. 4. Top 10 songs that is shared on twitter via spotify listed.

2.1.2 Listen to a Song on a Top10 list

Use Case	Listen to a Song on a Top10 list
Actors	User
Description	After listing top 10 songs user clicks a button to listen that song from spotify
Flow of events	1. User search a location 2. User selects a location and search top songs 3. A name and a link appers for each song in the top 10 list 4. User clicks on related Spotify Link to listen that song.

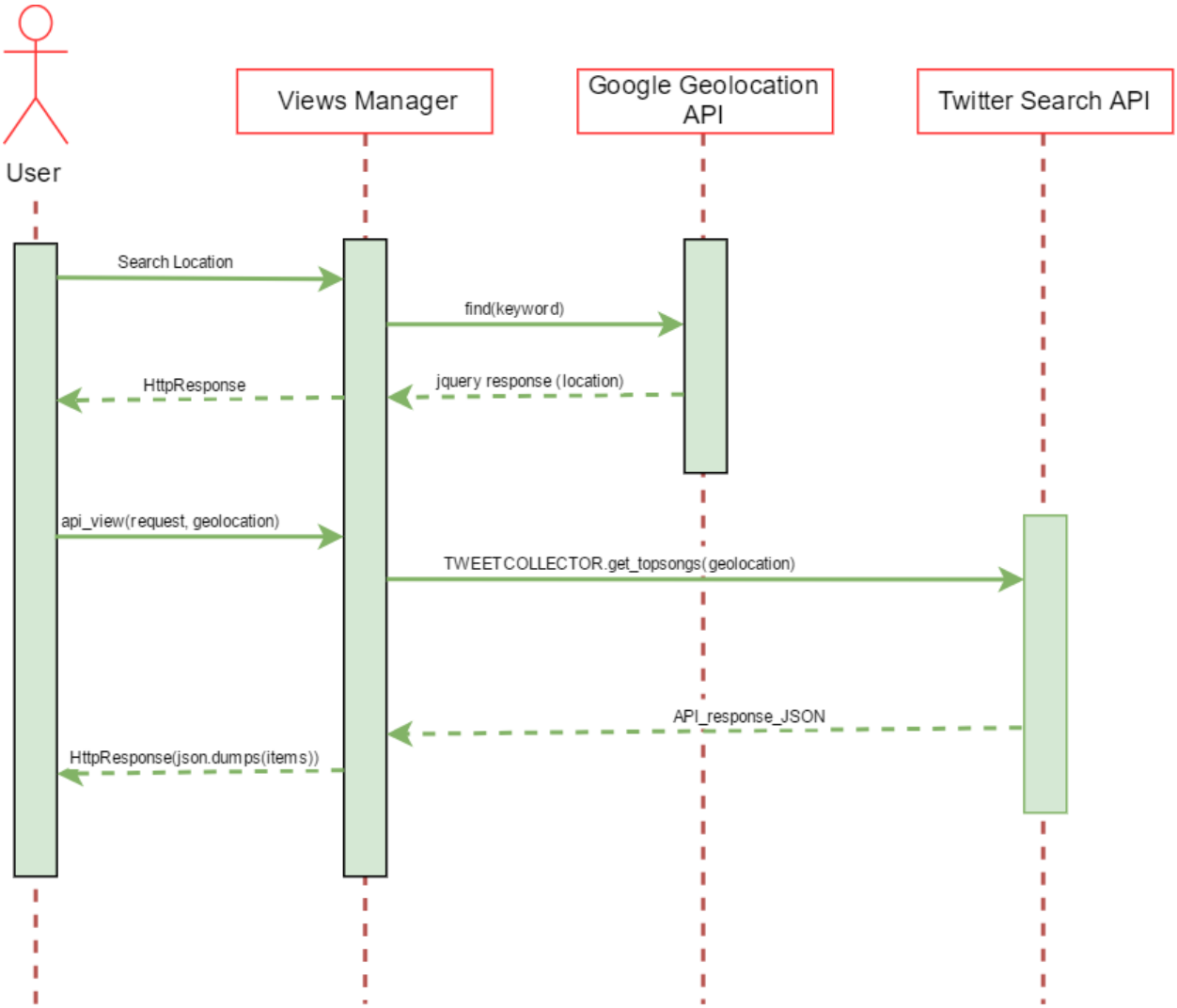
2.3 Activity Diagram

2.3.1 Activity Diagram for Top10 Songs Search

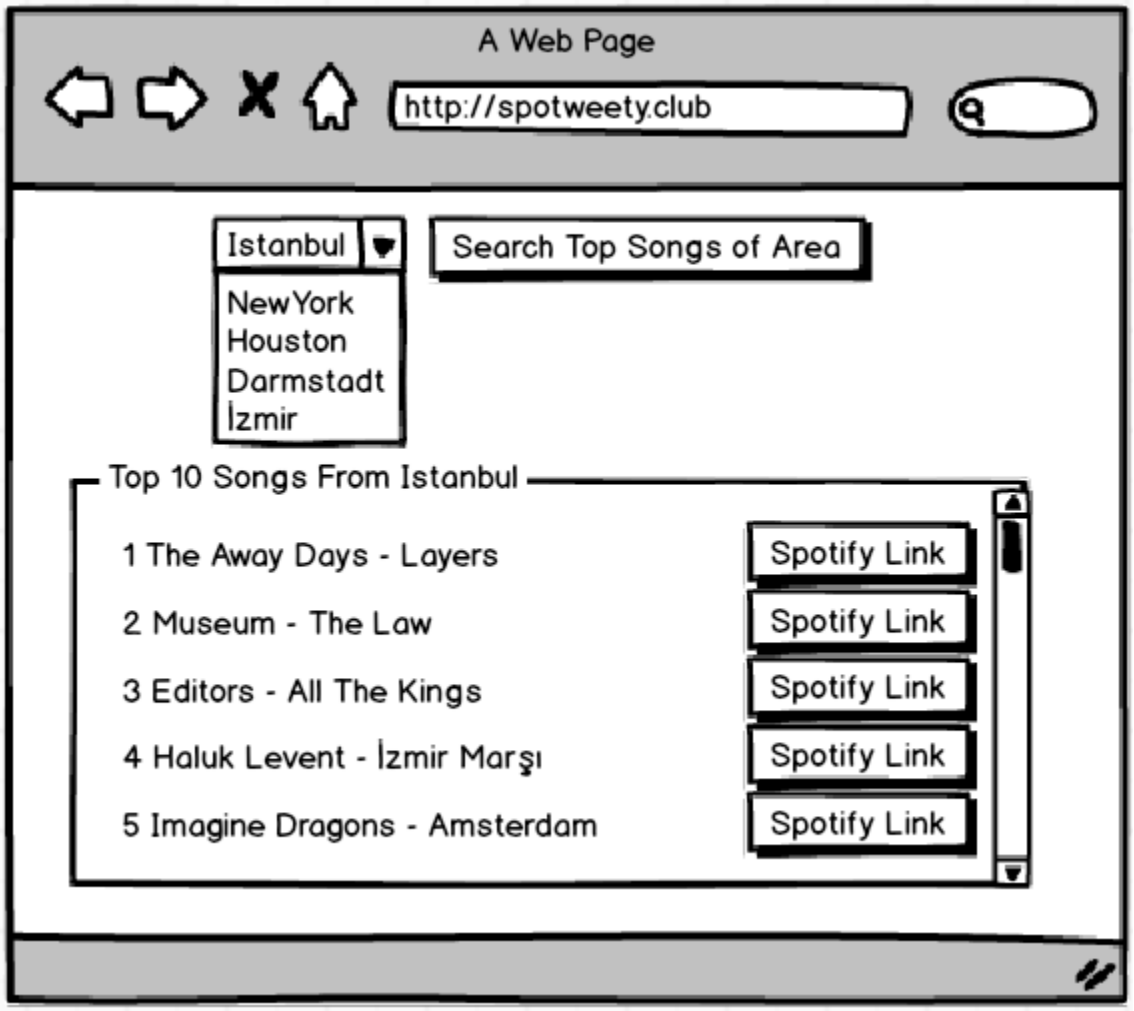


2.4 Sequence Diagrams

2.4.1 Sequence Diagram for Login



2.5 Mockup


A Web Page
http://spotweety.club
Istanbul
NewYork
Houston
Darmstadt
İzmir
Search Top Songs of Area
Top 10 Songs From Istanbul
1 The Away Days - Layers
Spotify Link
2 Museum - The Law
Spotify Link
3 Editors - All The Kings
Spotify Link
4 Haluk Levent - İzmir Marşı
Spotify Link
5 Imagine Dragons - Amsterdam
Spotify Link

3 Project Plan

3.1 Outline

Ref	Description	Beginning	Ending	Spent Hours (Planned)	Spent Hours (Actual)	Predecessor
M1	**Feasibility Study** **TOTAL: 4 WEEKS**	**07/02/2017**	**07/03/2017**	**22**	**25**	
M1.1	Analyze Spotify API and make a small application (1 WEEK)	07/02/2017	14/02/2017	3	5	
M1.2	Research on Twitter API (1 WEEKS)	14/02/2017	21/02/2017	5	7	
M1.3	Research on Web application development environments (2 WEEKS)	21/02/2017	07/03/2017	3	3	
M1.4	Research on popular backend and frontend Web frameworks, tools (2 WEEKS)			10	9	
M1.5	Research on task tracking tools (2 WEEKS)			1	1	
M2	**Requirements Analysis** **TOTAL: 3 WEEKS**	**07/03/2017**	**21/03/2017**	**11**	**17**	
M2.1	Analyze existing data on Twitter (1 WEEK)	07/03/2017	14/03/2017	10	15	
M2.2	Prepare final draft of requirements for validation (1 WEEK)	14/03/2017	21/03/2017	1	2	M2.1

M3	**Design Phase** **TOTAL: 3 WEEKS**	**21/03/2017**	**11/04/2017**	**14**	**20**	
M3.1	Setup design tools and prepare enviorenment (1 WEEK)	21/03/2017	28/03/2017	8	13	
M3.2	Prepare user interface mockups (1 WEEK)	28/03/2017	04/04/2017	1	1	M2.3
M3.3	Compare and decide on system architecture (1 WEEK)			2	3	
M3.4	Prepare class diagram (1 WEEK)			1	1	M2.3
M3.5	Prepare activity diagrams (1 WEEK)	04/04/2017	11/04/2017	1	1	M2.3
M3.6	Prepare sequence diagrams (1 WEEK)			1	1	M2.3
M4	**Development Phase** **TOTAL: 5 WEEKS**	**11/04/2017**	**16/05/2017**	**58**	**84**	
M4.1	Study Django documentation (1 WEEK)	11/04/2017	18/04/2017	5	7	
M4.2	Start an Hello World App on Django (1 WEEK)			3	4	
M4.3	Analyze Twitter API by using Jupyter Notebook (1 WEEK)	18/04/2017	25/04/2017	20	32	
M4.4	Crate API Wrapper and Analyzer for Twitter with python (1 WEEK)	25/04/2017	02/05/2017	5	4	M3.2
M4.5	Code down Django views (1 WEEK)	02/05/2017	09/05/2017	12	15	M3.4, M3.5, M3.6
M4.6	Define URL controllers			3	2	M4.5

	(1 WEEK)					
M4.7	Implement dynamic content of Web pages by integrating static templates, related Django views and Javascript frameworks (1 WEEK)	09/05/2017	16/05/2017	10	20	M4.5, M4.6
M5	**Testing and Deployment Phase TOTAL: 1 WEEK**	**16/05/2017**	**23/05/2017**	**26**	**40**	
M5.1	Apply manual tests and document found issues on GitHub (1 WEEK)	16/05/2017	23/05/2017	3	3	M4.7
M5.2	Solve issues and bugs recently opened (1 WEEK)			10	15	M5.1
M5.3	Code automated unit tests (1 WEEK)			3	2	M5.2
M5.4	Deploy Project to a Web Server and register a domain (1 WEEK)			10	20	M4.7, M5.2, M5.3
	Project is complete TOTAL: 16 WEEKS	**07/02/2017**	**23/05/2017**	**132 HOURS PLANNED**	**187 HOURS ACTUALLY SPENT**	

4 Main Tools Used in the Project

4.1 Environment Tools

- ✓ Piazza – as central communication tool for announcements, notes, questions and collaborative answers
- ✓ GitHub – repository for keeping codes together, version management, issues, wiki page for documentation, projects tab for milestones, task descriptions and task tracking
- ✓ PyCharm IDE – for developing backend part of the project
- ✓ Jupyter Notebook – The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser
- ✓ Balsamique Mockups – for user interface mockup designs
- ✓ WinSCP – for reliable file transfer between local computer and deployment server
- ✓ Digital Ocean – DigitalOcean is a cloud infrastructure provider focused on simplifying web infrastructure for software developers. Used for hosting
- ✓ goDaddy – GoDaddy Inc. is an American publicly traded Internet domain registrar and web hosting company. Used for domain name: spotweety.club
- ✓ Putty – as a network application and terminal emulator for managing deployment process operating system hosted on Digital Ocean

4.2 Frameworks and Libraries

- ✓ Django– as main Web framework in the project for developing intended Web application
- ✓ Bootstrap – as an HTML/CSS framework to design static components of frontend templates
- ✓ Twython – Actively maintained, pure Python wrapper for the Twitter API

- ✓ JQuery – as a Javascript library to manage Ajax calls and dynamic components of Web templates
- ✓ Requests Package – a Python package for sending HTTP requests to external APIs
- ✓ Django REST – for returning raw JSON and testing APIs
- ✓ Nginx – NGINX is an incredibly fast and light-weight web server. We will use it to serve up our static files for our Django app.
- ✓ Unittest – It is the batteries-included test module in the Python standard library. Its API will be familiar to anyone who has used any of the JUnit/nUnit/CppUnit series of tools.

5 Test Results

At this part there are two main tests written. Two libraries are used. These are:

- Unittest
- TestCase from django.test

5.1 Acceptance Tests

apiwrapper.py

- Checks apiwrapper in order to track input and output;
- Possible wrong values are tested

```
import unittest
from api.apiwrapper import TCOL


class TestStringMethods(unittest.TestCase):
  {
  def test_wrapper(self):
    input1 = "41.015137,28.979530,1000km"
    output1 = len(TCOL.get_topsongs(input1))
    return self.assertContains(output1, 10)
```

5.2 Unit Tests

rest.py

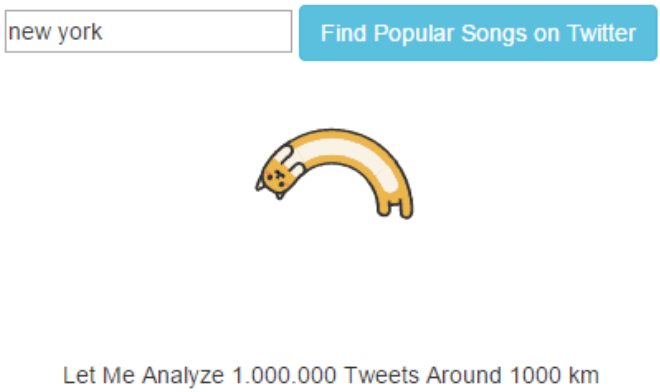
- Checks frontend behaviour against various responses.
- Empty values are tested

```
class EntriesTestCase(SetupTestCase):
  def test_get(self):
    c = self.get_authenticated_client(self.users[0])
    r = c.get("/api/")
    content = r.json()

    self.assertEqual(len(content), 3)
    c2 = self.get_authenticated_client(self.users[1])
    get_response = c2.get("/api/").json()
    self.assertEqual(len(get_response), 1)
```

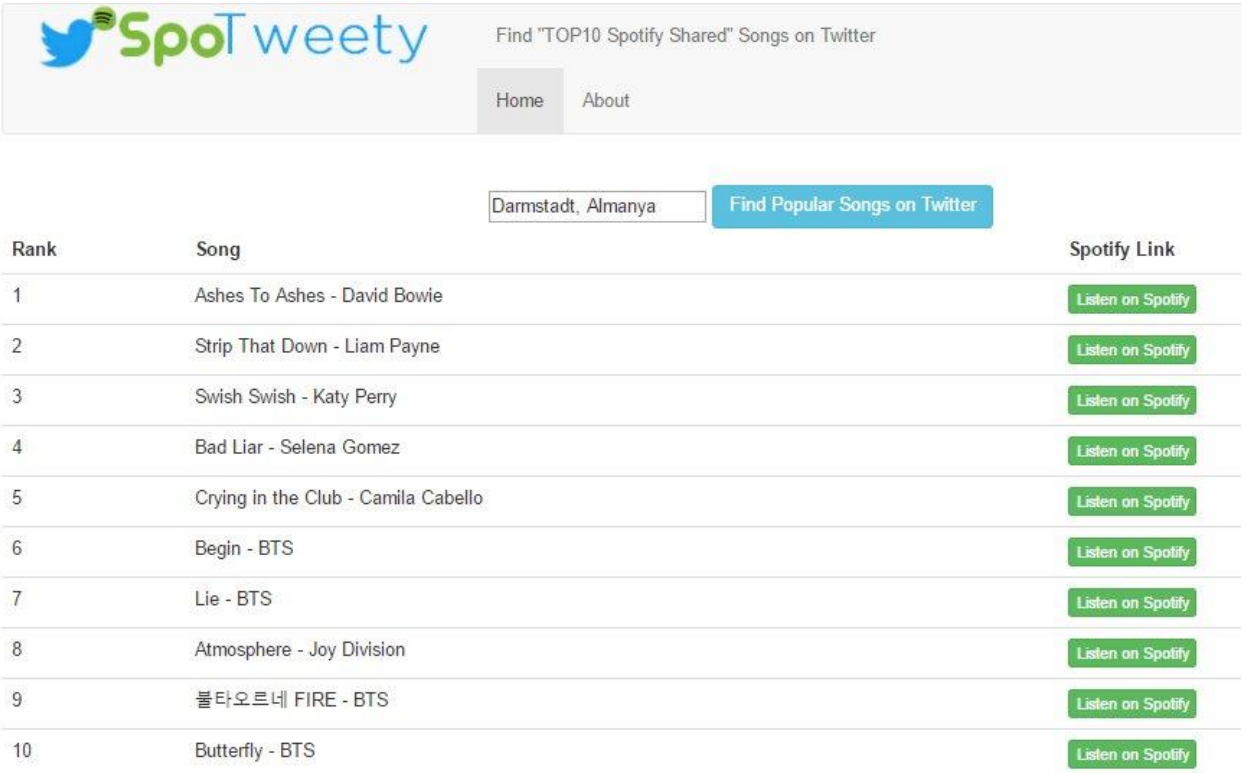
```
All 7 tests passed – 15ms

E:\Programs\Python\Python35-32\python.exe "E:\JetBrains\PyCharm 2016.2
 .3\helpers\pycharm\django_test_manage.py"

Testing started at 7:31 PM ...
Creating test database for alias 'default'...
Destroying test database for alias 'default'...

Process finished with exit code 0

Event Log
7:1  CRLF÷  UTF-8÷  Git: master ÷
```

6 Usage Example

6.1 Search Page



6.2 Result Page



7 System Requirements

7.1 Software Requirements

- Python v2 or v3.5 is required to run Web Application on the server. (Application developed using pythonv3.5)
- Django Web Framework v1.10.3 release is required. Django can be installed after Python is installed
- Django REST Framework v3.5.3 is required. It can only be installed after Python and Django both are installed
- Python Requests Package v2.12.1 is required for API requests to work correctly. It can be obtained using pip installer after Python is installed
- Python Library: django-cors-headers
- Python Library: tywthon
- Debian 7.11 x64 (Any Unix that could run Python and Django)

7.2 Hardware Requirements

- 512 MB RAM
- 20 GB Disk Space

8 References

[1] Twitter Search API Documentation: https://dev.twitter.com/rest/public/search

[2] Spotify API Documentation: https://developer.spotify.com/web-api/

[3] Writing your first Django app: https://docs.djangoproject.com/en/1.11/intro/tutorial01/

[4] Django REST Documentation : http://www.django-rest-framework.org/tutorial/quickstart/

[5] How to use Twitter's Search REST API most effectively: https://www.karambelkar.info/2015/01/how-to-use-twitters-search-rest-api-most-effectively./

[6] nginx Documentation https://nginx.org/en/docs/