Technical Report: The Austinites

Sheeyla Garcia Jesus Hernandez Kyle Nicola
Stephen Ridings Carlos Rodriguez Mark Sandan

August 7, 2014

Contents

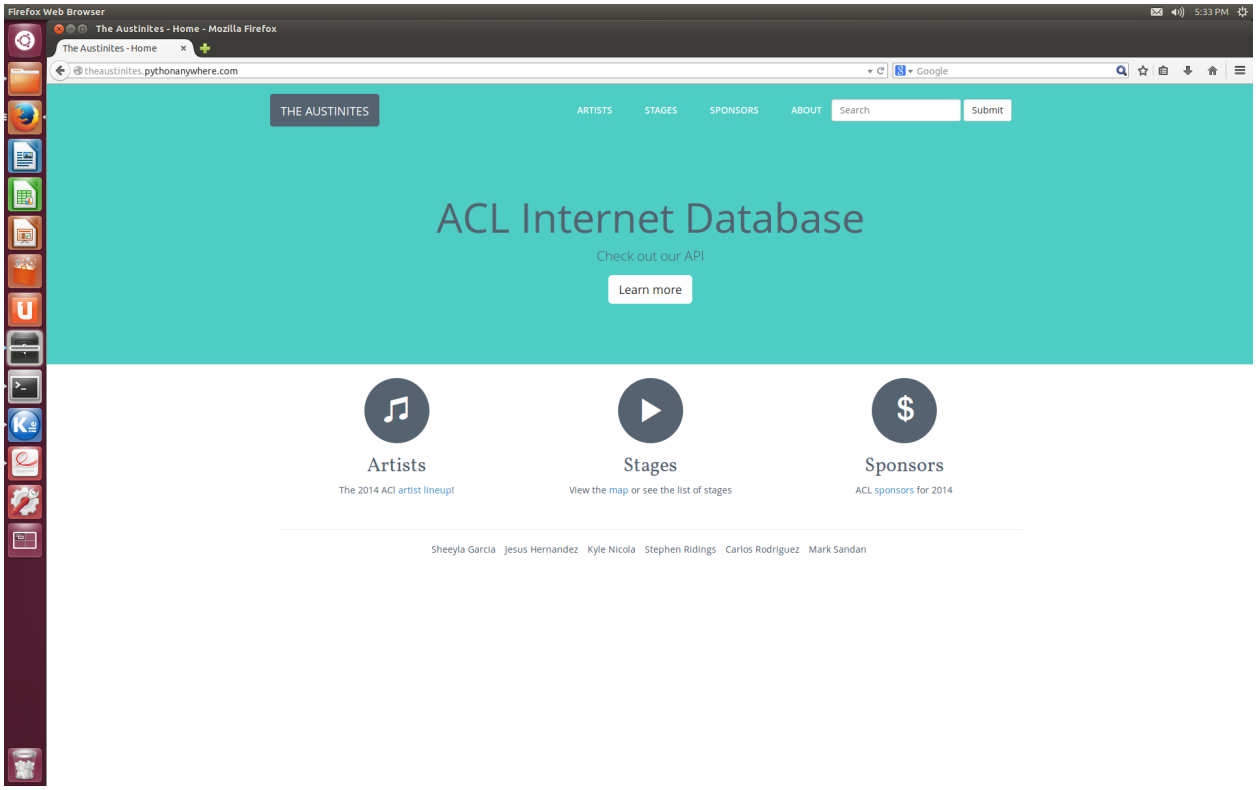

Figure 1: Splash Page for the second iteration of IDB2 showing the group name, buttons for the three main pages, and a link to the API on Apiary. The Carousel template from twitter bootstrap is used. There is also a search bar added on the upper right.

1 Updates

Updates for the third iteration at a glance:

- Search bar and Search Page listing result added
- Augmented models to model time in relation to our models
- Django Rest framework installed to implement the API and test it more rigorously including the new models
- New API in apiary that includes information on the new classes to model time
- New data added to the website that reflects the time relationships between models
- New icons for the twitter, facebook, and youtube links along with an embedded google map on stages page

1.1 Summary

The main changes that have been made can be summarized into three categories: dynamic functionality, REST implementation, and the model refactoring.

To implement a dynamic webpage we are currently using html,css,javascript, and the django template language to access instances of Artists, Sponsors, and Stages to fill the content of the webpages. The REST implementation required installing the django rest framework on the pythonanywhere envioronment which enabled us to return JSON representations of MySQL database instances. The tests for the REST implementation have also been added to the test suite.

The models have been refactored to include media classes that provide the dynamic content for the Artist, Sponsor, and Stage models when they are loaded into the webpage. The media content contains embedded links to youtube, facebook, twitter, etc.

Database loading has been relegated to using python scripts which have been installed as django commands used with manage.py.

2 Introduction

Our website is about the 2014 Austin City Limits (ACL) music festival, an annual three-day American music festival that takes place on 46-acres in Zilker Park located in Austin, Texas. The website design has three main pages: Artists, Stages, and Sponsors along with a splash page. An Artist is allowed to perform on one stage but stages can have many artists perform on them. A sponsor is a business entity that may or may not sponsor a stage. A stage is a physical location in the ACL festival that many artists play on. It can have only one sponsor that sponsors it for any given year. Artists and Sponsors are related through Stages.

The website allows anyone to view pages about the current Artists, Sponsors, and Stages involved in the festival. A user can find links from a specific artist page to the stage they’re playing on as well as the sponsor sponsoring the stage. Similarly

for the stage and sponsor pages. The structure is modeled after the IMDB website (http://www.imdb.com/) where the entities are highly coupled. A problem that we are facing is that the information we need to complete the project hasn't been published as of the date this report except the Artist lineup. Their scheduled performances which include information should be set to release sometime this month.

3 New Features

Our website is about the 2014 Austin City Limits (ACL) music festival, an annual three-day American music festival that takes place on 46-acres in Zilker Park located in Austin, Texas. The website design has three main pages: Artists, Stages, and Sponsors along with a splash page. An Artist is allowed to perform on one stage but stages can have many artists perform on them. A sponsor is a business entity that may or may not sponsor a stage. A stage is a physical location in the ACL festival that many artists play on. It can have only one sponsor that sponsors it for any given year. Artists and Sponsors are related through Stages.

The website allows anyone to view pages about the current Artists, Sponsors, and Stages involved in the festival. A user can find links from a specific artist page to the stage they're playing on as well as the sponsor sponsoring the stage. Similarly for the stage and sponsor pages. The structure is modeled after the IMDB website (http://www.imdb.com/) where the entities are highly coupled. A problem that we are facing is that the information we need to complete the project hasn't been published as of the date this report except the Artist lineup. Their scheduled performances which include information should be set to release sometime this month.

3.1 Search

Currently our search functionality uses the Django haystack version 2.2.0 with the Whoosh python backend engine.

3.2 New Models

3.3 Technology Stack

The technologies used are:

- PythonAnywhere: a web hosting service with python environments supported. Currently we use Python 3.4 and Django 1.6+.
- Twitter Bootstrap 3.2: a web hosting service with python environments supported Python 3.4, Django 1.6.
- Apiary: an online service to provide an API for client-side web access to our databases.
- MySQL: The current database backend using the mysql.connector.django engine.
- MySQL: The current database backend using the mysql.connector.django engine.

4 Design

Using Django's templating language we are able to reuse html files by extending from them. Currently we only have a single base html page that is extended from and uses Twitter Bootstrap. We have 24 pages that are dynamically loaded. The design and content of each page is outlined below.

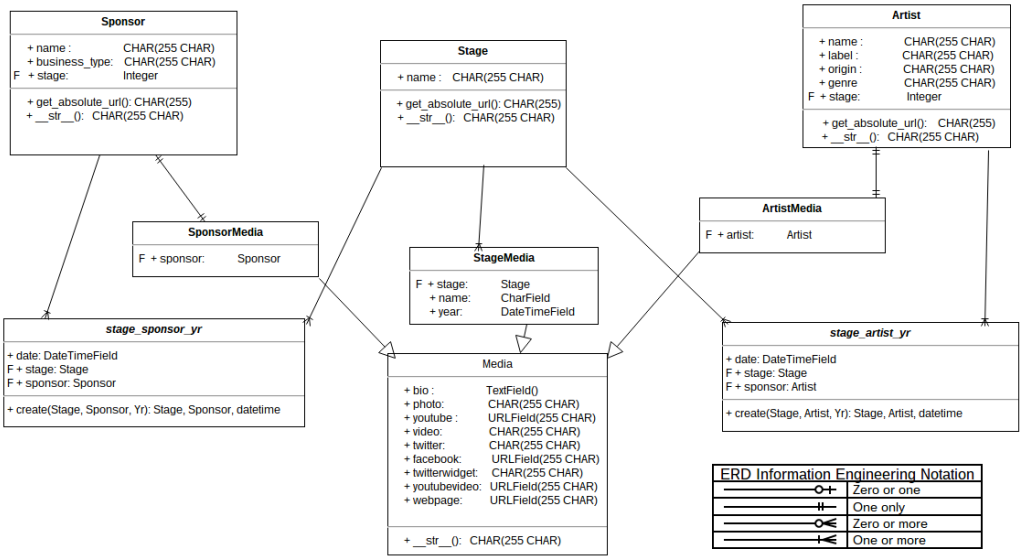

Figure 2: The current UML schema depicting the relationships between the Django models. The main three models are Artist, Sponsor, and Stage each of which have an associated Media page.

4.1 Web Pages

Each web page has basic information about a particular artist, sponsor, or stage involved in the ACL music festival. All pages will include navigation links that will allow the user to go back to the main "splash" page, as well as reach the Artists, Sponsors, and Stages pages.

In future phases we are considering incorporating a search bar inside of the navigation bar, so that the user can search all categories.

Mobile browsers are able to view the webpage correctly and set the height and width to a percentage of the size of the screen. Using Django Templating along with Class-based Generic Views we were able to design a website that dynamically loaded pages using the information from our MySQL database.

4.1.1 Splash Page

- URL: https://theaustinites.pythonanywhere.com/

The "splash" page will be the first page a visitor to the site will see. It provides button style links to all subcategories (Artists, Sponsors, Stages). Currently the page uses the carousel template provided by twitter bootstrap.

4.1.2 Artists

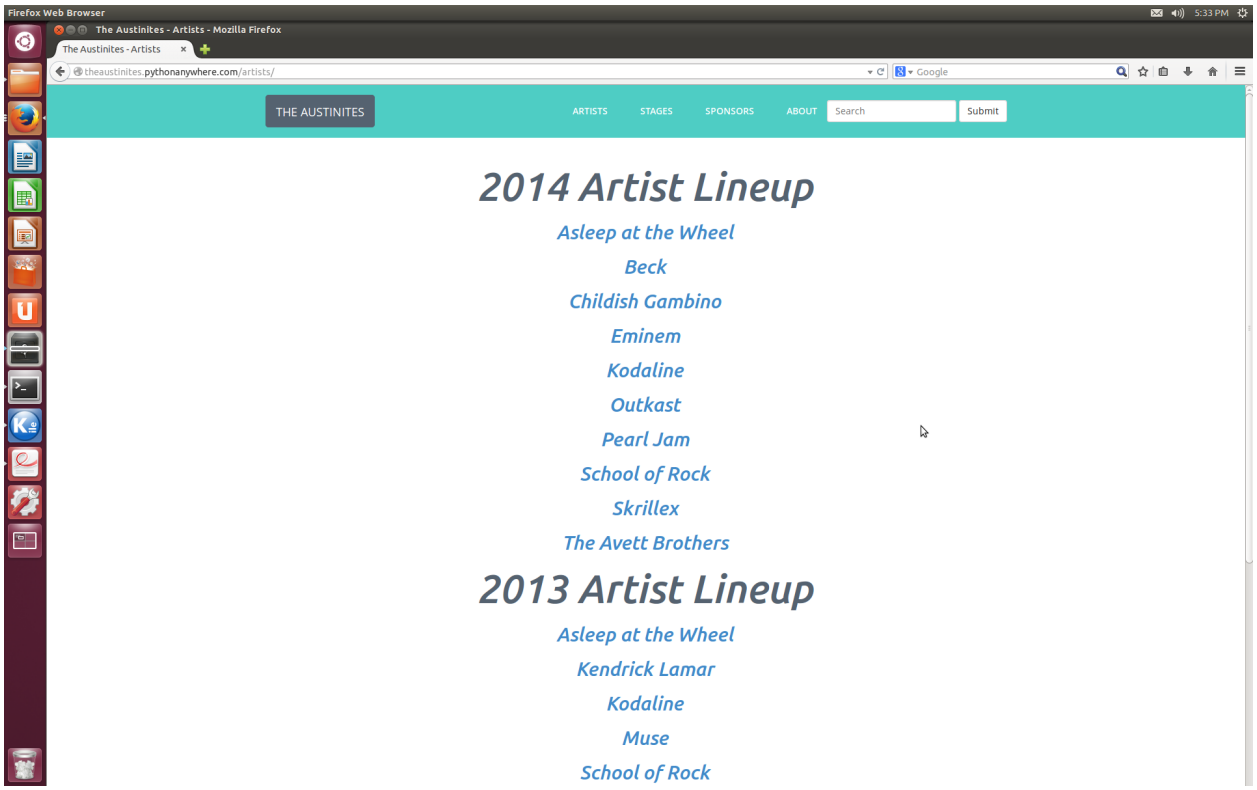

Figure 3: The current UML schema depicting the relationships between the Django models. The main three models are Artist, Sponsor, and Stage each of which have an associated Media page.

- URL: https://theaustinites.pythonanywhere.com/artists

Artist Pages can be reached from the home page as well as from the Stage or Sponsor pages depending on whether the Artist played on a Stage that was hosted by a Sponsor.The page currently lists the dynamic web pages for music artists:

- Outkast
- Eminem
- Pearl jam
- Beck
- Foster The People
- Skrillex
- Lana Del Rey

- Calvin Harris
- Phantogram
- Childish Gambino

Each Artist page includes the Artist name, an Artist photo, the label, the origin, genre, the sponsor,the stage, a bio, the official website to the sponsor, a facebook page, a bio, youtube video, youtube channel, and twitter feed.

4.1.3 Sponsors

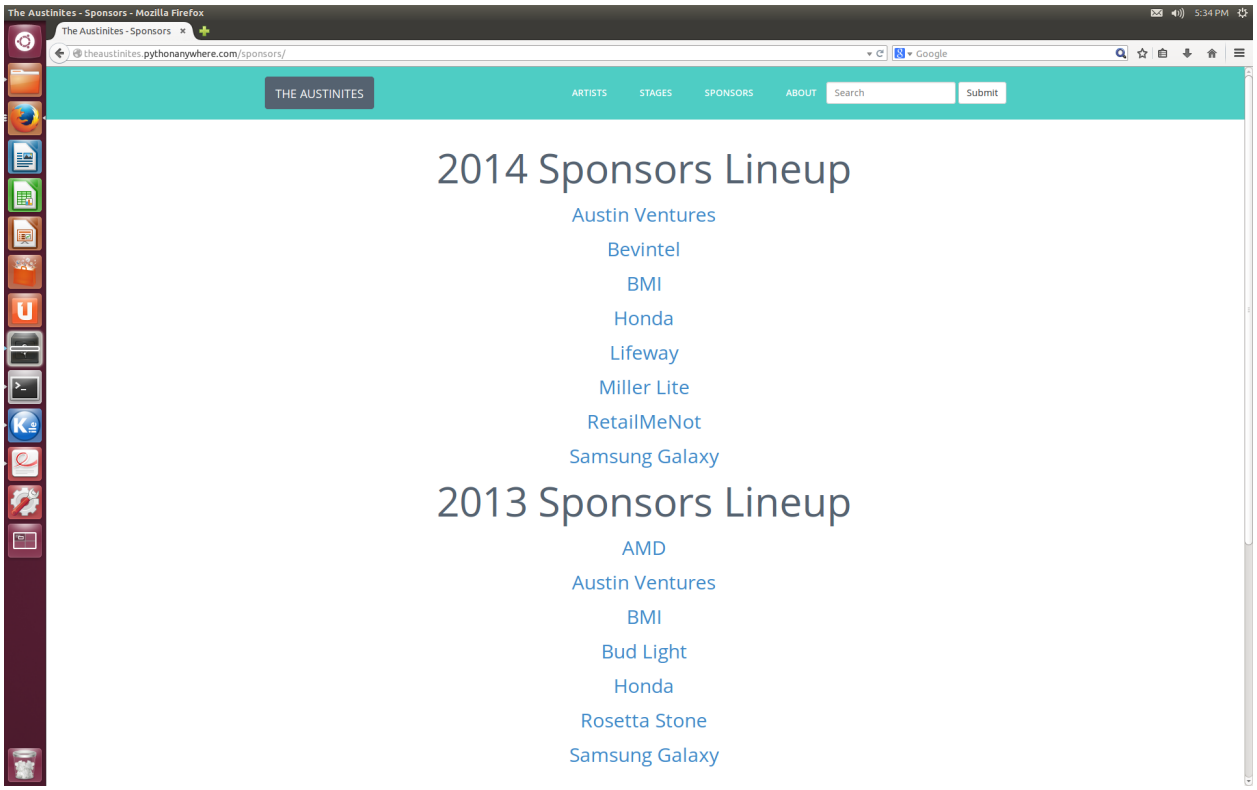

Figure 4: The current UML schema depicting the relationships between the Django models. The main three models are Artist, Sponsor, and Stage each of which have an associated Media page.

- URL: https://theaustinites.pythonanywhere.com/sponsors

Sponsor pages can be reached from the home page as well as from the Artist or Stages pages depending on whether the Sponsor hosted a Stage, and that particular Artist played on that Stage. The page currently lists ACL festival sponsors:

- Honda
- Miller Lite
- Samsung Galaxy
- Austin Ventures

- Bacardi
- BMI (Broadcast Music Inc.)
- Tito's Handmade Vodka
- H.E.B.
- Sweet Leaf Iced Teas
- Western Digital

Each Sponsor page includes the Sponsor name, a sponsor logo, the origin, stage sponsored, the artists playing the stage, the official website to the sponsor, a facebook page, a bio, youtube video, and twitter feed.

4.1.4 Stages

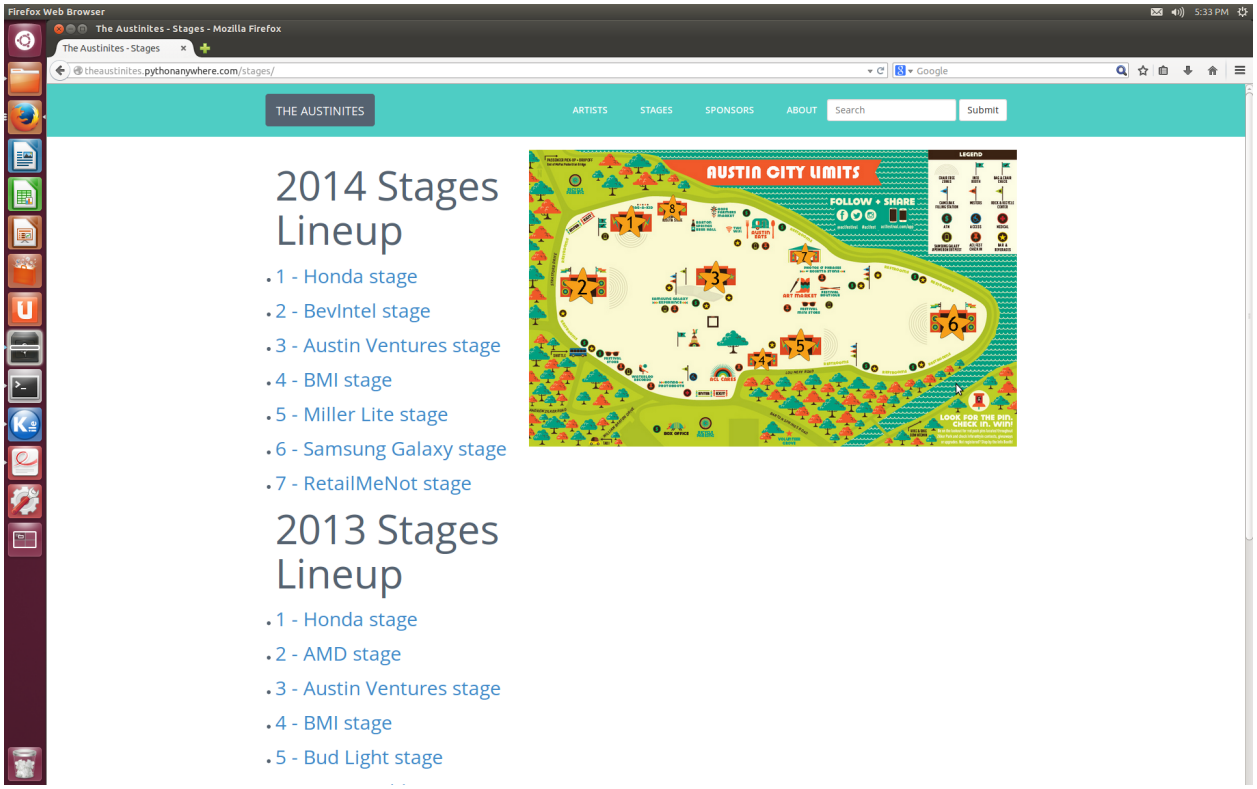

Figure 5: The stages index page which features a map of the ACL festival at Zilker Park in Austin Texas. The numeric mapping is labeled here with stars.

- Location: https://theaustinites.pythonanywhere.com/stages/

Stage pages can be reached from the Home page (splash) as well as from the Artist or Sponsor pages depending on whether the Sponsor hosted a Stage, and that particular Artist played on that Stage.

Each Stage page includes the Stage name, a Stage logo, the artists playing the stage, the official website to the sponsor, a facebook page, youtube video, and twitter feed.

4.2 Templates

In phase one of the project we used Django templates to write static html files. In phase two we have updated our templates to use python code. Along with context from the views, this allowed us to generate a dynamic website. Built-in methods from django models also allows us to access other related instances such as Stages and Sponsors for an Artist. We are able to provide links dynamically as new relationships are being added to the database.

4.2.1 Base Template

Our first template is base.html, which all other templates extend from. The template holds our bootstrap links as well as the main structure for the site. We have separated the base template into sections labeled by the block keyword. Each template that extends base.html can modify these sections to present a different site. This allows us to make new sites without rewriting code.

4.2.2 Model Templates

Each of the three main Models have their own templates. For example Artist has artists.html and artist.html. The plural template shows the list of artists using python to iterate through the artist objects. The singular template uses python to access variables from the objects and call methods to display information.

4.3 Class-based Generic Views

We have rewritten our views for phase two of the project. Previously we used functions to render the html templates. We have moved to Class-Based Generic Views that allow us to easily write views for common tasks.

There are three main types of pages in terms of the generic view design: The splash page that uses the index function view, the Index pages that list sponsors, artists, and stages, and the Detail pages that fill out content on a specific object's template.

4.3.1 ArchiveIndexView

The ArchiveIndexView generic class is used to pass the time relation objects which are automatically sorted by their date attribute. Using this class allowed us to easily pass the list of artists, stages, and sponsors to our templates. Iterating through the list of objects we are able to dynamically load the page with the object's name and link to the other model classes they are related too.

4.3.2 ListView

The ListView generic class is used to pass the list of objects in the Model table into a template. We used this class in ArtistIndex, StageIndex, SponsorIndex. Using this class allowed us to easily pass the list of artists, stages, and sponsors to our templates. Iterating through the list of objects we are able to dynamically load the page with the object's name and link to the other model classes they are related too.

4.3.3 DetailView

The DetailView generic class is used to pass single objects from the database into a template. We used this class in ArtistPage, StagePage, and SponsorPage. This class allowed us to pass an artist, sponsor, or stage to a template. Passing this information to the template allowed for the same template to represent different objects.

4.4 RESTful API

The API allows GET requests to the following models: Stages, Sponsors, Artists. The Members and Photos models have been deprecated since they haven't been used. The following section will detail the attributes for the modules and how the server will respond to the GET requests. See the Links section below to view the Apiary API that outlines the GET responses for the Media models.

4.4.1 Stages

When a GET is called on $[/api/stages]$ it will return a JSON representation of all the Stages in the database. It will be a list of the stages along with their attributes. When a GET is called on $[/api/stages/id]$ it will return a JSON representation of a single Stage in the database with the given id. It will list the stage and all of it's attributes
Example of single stage:

```
{
         "location": 42,
}
```

4.4.2 Sponsor

When a GET is called on $[/api/sponsors]$ it will return a JSON representation of all the Sponsors in the database. It will be a list of the sponsors along with their attributes. When a GET is called on $[/api/sponsors/id]$ it will return a JSON representation of a single Sponsor in the database with the given id. It will list the sponsor and all of it's attributes.
Example of single sponsor response:

```
{
        "id": 42,
        "name": "Sponsor name",
        "industry": "Type of Business",
}
```

4.4.3 Artist

When a GET is called on $[/api/artists]$ it will return a JSON representation of all the Artists in the database. It will be a list of the artists along with their attributes. When a GET is called on $[/api/artists/id]$ it will return a JSON representation of a single Artist in the database with the given id. It will list the artist and all of it's attributes.
Example of single artist:

```
{
        "id": 42,
        "name": "Artist name",
        "label": "Label of artist",
        "origin": "Where the artist was from",
        "genre": "Genre of the artist",

}
```

4.5 Django Models

The Django models created represent the entities we believe are essential to modeling the ACL festival. The following subsections document the attributes and intended functionality of each class instance method. The following sections use *child* and *parent* in the sense of the schema relationship depicted by the UML diagram in Figure 2 and not in the sense of the object oriented inheritance (the only object each model extends is models.Model except the Media classes ArtistMedia, SponsorMedia, and StageMedia that inherit from Media). Currently there are a total of six class.

4.5.1 Artist

The Artist class represents the current Artists playing on a sponsored stage. All Artists will be a child of some stage depending on whether they are playing that Stage or not. The entity relation between Sponosor and Artist The Artist class is implemented using the following:

```
attributes:
- id Primary Key, integer
- name: the unique name of max length 255 characters.
- label: the artist label with a maximum length of 255 characters.
- origin: the place of origin the artist/group formed with a
          maximum length of 255 characters.
- genre: the genre associated with the artist. May span more
         than one. maximum length of 255 characters.
- stage: Foreign Key, integer of type field.

methods:
- get_url(): returns the string "/artists/{id}" . Maximum number of 255 characters.
- __str__(): returns the name string of the artist.Maximum  number of 255 characters
```

4.5.2 Sponsor

The Sponsor class represents the ACL sponsors that sponsor a Stage for the Artist to perform on. This relationship is expressed using the one-to-many relationship between the Sponsor and the Stage class.

The Sponsor class has the following attributes and methods:

```
attributes:
```

```
- id: Primary Key, integer type field
- name: a character type field with a maximum length of 255 characters
- business_type: a character type field with a maximum length of 255 characters
- stage: Foreign Key, integer of type field

methods:
- get.url(): returns the string "/sponsors/{id}". Maximum number of 255 characters.
- __str__(): returns a string that represents the name of the sponsor.
```

4.5.3 Stage

The Stage class is meant to represent the stage that an Artist will perform on. All stages have one sponsor. The Stage class extends from the Django models.Models class.

The Stage class has the following attributes and methods:

```
attributes:
- id: Primary Key, integer type field
- name: a character type field with a maximum length of 255 characters


methods:
- get.url(): returns the string "/stages/%s/{name}" where name is the stage name.
- __str__(): returns a string that represents the name of the stage
```

4.5.4 stage_sponsor_yr

The stage_sponsor_yr class is meant to relate a stage location with a sponsor in a given year. The Stage class has the following attributes and methods:

```
attributes:
- id: Primary Key, integer type field
- date: a character type field with a maximum length of 255 characters
- stage
- sponsor

methods:
- get_absolute_url(): returns the string "/stages/%s/{name}" where name is the stage n
- __str__(): returns a string that represents the name of the stage
```

4.5.5 stage_artist_yr

The stage_artist_yr class is meant to relate a stage location with an artist in a given year. The stage_artist_yr class has the following attributes and methods:

```
attributes:
- id: Primary Key, integer type field
- date: a character type field with a maximum length of 255 characters
- stage
```

```
- artist

methods:
- get_absolute_url(): returns the string "/stages/%s/{name}" where name is the stage n
- __str__(): returns a string that represents the name of the stage
```

4.5.6 Media models

The Media class will represent all of the media associated with Stages, Sponsors, and Artists. All Artists, Sponsors, and Stages will associated with Media in a many-to-one relationship.

The Media class has been included to remove a lot of redundancy in code, and to simplify improving the overall design of our pages. We use the aggregation design pattern for each instance of a model and all Media classes will be an aggregate of all media associated with an Artist, Stage, or Sponsor. The media components included will be images, videos, website,facebook,youtube channel and twitter feeds.

The Media class has the following attributes and methods

```
 attributes:
-id: Primary key, Integer
-bio: A CharField of 255 characters. Biography.
-photo: A CharField of 255 characters.
-youtube: A CharField of 255 characters. url for Youtube Channel
-video: A CharField of 255 characters.
-twitter: A CharField of 255 characters. url for Twitter
-facebook:A CharField of 255 characters. url for Facebook
-twitterwidget: A CharField of 255 characters.
-webpage: A Char field of 255 characters.

methods:
- __str__(): returns a string that represent a stage, sponsor, or artist’s website.

subclasses:
- SponsorMedia: a media subclass that has a one-to-one relationship
                with an instance of the Sponsor class
  attributes:
  - sp: Foreign key, integer type field
- StageMedia: a media subclass that has a one-to-one relationship with
              an instance of the Stage class
  attributes:
  - st: Foreign key, integer type field
- ArtistMedia: a media subclass that has a one-to-one relationship with an
               instance of the Artist class
  attributes:
  - ar: Foreign key, integer type field
```

4.5.7 Stage Sponsor Time relation

The Media class will represent all of the media associated with Stages, Sponsors, and Artists. All Artists, Sponsors, and Stages will associated with Media in a many-to-one relationship.

The Media class has been included to remove a lot of redundancy in code, and to simplify improving the overall design of our pages. We use the aggregation design pattern for each instance of a model and all Media classes will be an aggregate of all media associated with an Artist, Stage, or Sponsor. The media components included will be images, videos, website,facebook,youtube channel and twitter feeds.

4.5.8 Stage Artist Time relation

The Media class will represent all of the media associated with Stages, Sponsors, and Artists. All Artists, Sponsors, and Stages will associated with Media in a many-to-one relationship.

The Media class has been included to remove a lot of redundancy in code, and to simplify improving the overall design of our pages. We use the aggregation design pattern for each instance of a model and all Media classes will be an aggregate of all media associated with an Artist, Stage, or Sponsor. The media components included will be images, videos, website,facebook,youtube channel and twitter feeds.

5 Unit Tests

The unit tests currently implemented reflect the expected functionality of the django models and database. they have been rigorously augmented to test return values and simulations of corner cases. Currently we are using the MySQL database as a test database backend. We assume for each function in the set of tests that the state of the database is reset.There are currently 54 tests in the test suite testing the models and API.

5.1 Django Model Unit Testing

Currently the django models are tested with a MySQL database backend. The database created is temporary. It is created on setup, populated and run to test the attributes and methods of the models we use. Finally, when all tests are run the database is destroyed. The tests are run in a non deterministic order.

5.1.1 Type Test

Total tests: 3

- test_types: creates a Stage, Artist, and Sponsor and asserts their types ar Stage, Artist, and Sponsor respectively

- test_types_relationships1: tests different combinations of creating the stage$_artist_y robject to relate a Stage$ $tests different combinations of creating the stage_sponsor_y robject to relate a Stage and a Sponsor$

5.1.2 StageRelationship Test

Total tests: 2

- test_relationship1: creates a Stage and an Artist and relates them with the stage$_{a}rtist_{y}robject.TestsaccessingtheStageandArtistfromthestage_{a}rtist_{y}r.test_relationship2$: $createsaStageandaSponsorandrelatesthemwiththestage_{s}ponsor_{y}robject.TestsaccessingtheStageandS$

5.1.3 Constraint Test

Total tests: 4

5.1.4 MultiRelation Test

Total tests: 4

5.1.5 CrossRelation Test

Total tests: 2

5.1.6 Simulations Test

Total tests: 1

5.2 RESTful API Test

The REST API is tested using the django rest framework. The django rest framework has a built in library that allows the tests to mimic API clients that interact with the API. Currently only GET requests are allowed. Any other request will return a 405 Method Not Allowed. There are three tests for each of the classes (Stages, Sponsors, Artists, ArtistMedia, StageMedia, and SponsorMedia).

Each test sets up by creating mock relationships and attributes, saving them to the database and then performing various GET requests using the client object from the django rest framework.

5.2.1 Stages API Test

Total tests: 3 One stage with an id of 1 is uploaded into the database with name=$newstage$. On tear down, it is deleted.

- testStageGet: tests the status code of a successful client requesting GET for the first stage.
- testStageGet2: tests the status code is 404 (Not Found) for an unsuccessful GET request for a stage instance that doesn't exist in the database.
- testPost: tests the status code is 405 (Method Not Allowed) for an unsuccessful POST request for the first stage.

5.2.2 Artist API Test

Total tests: 3 A test stage and artist are uploaded into the database. On tear down they are both deleted.

- testStageGet: tests the status code of a successful client requesting GET for the first artist
- testStageGet2: tests the status code is 404 (Not Found) for an unsuccessful GET request for an artist instance that doesn't exist in the database.
- testPost: tests the status code is 405 (Method Not Allowed) for an unsuccessful POST request for the first artist.

5.2.3 Sponsors API Test

Total tests: 3 On setup one stage and one sponsor are created in the database and then deleted on tear down.

- testStageGet: tests the status code of a successful client requesting GET for the first sponsor
- testStageGet2: tests the status code is 404 (Not Found) for an unsuccessful GET request for an sponsor instance that doesn't exist in the database.
- testPost: tests the status code is 405 (Method Not Allowed) for an unsuccessful POST request for the first sponsor.

5.2.4 StageMedia API Test

Total tests: 3 On setup a stage and StageMedia are created in the database. Class attributes for the StageMedia test are created. Both are deleted on tear down.

- test_stage_media_get: tests to see if a client request for the stage media object created returns the 200 OK response code.
- test_stage_media_bad_get: tests to see if a bad client request to a non-existing stage media object created returns 404 Not Found response.
- test_stage_media_post: Tests that POSTS are not allowed by clients by checking that the 405 status code is returned.

5.2.5 ArtistMedia API Test

Total tests: 3 On setup an artist, stage, and ArtistMedia are created in the database. Class attributes for the ArtistMedia test are created. All instances are deleted on tear down.

- test_artist_media_get: tests to see if a client request for the artist media object created returns the 200 OK response code.
- test_artist_media_bad_get: tests to see if a bad client request to a non-existing stage artist object created returns 404 Not Found response.

- test_artist_media_post: Tests that POSTS for artists are not allowed by clients by checking that the 405 status code is returned.

5.2.6 SponsorMedia API Test

Total tests: 3 On setup a sponsor, stage, and SponsorMedia are created in the database. Class attributes for the SponsorMedia test are created. All instances are deleted on tear down.

- test_sponsor_media_get: tests to see if a client request for the sponsor media object created returns the 200 OK response code.
- test_sponsor_media_bad_get: tests to see if a bad client request to a non-existing sponsor media object created returns 404 Not Found response.
- test_sponsor_media_post: Tests that POSTS for sponsors are not allowed by clients by checking that the 405 status code is returned.

6 Six Potters API Testing

7 Links

- Home Page: https://theaustinites.pythonanywhere.com/
- Apiary API: http://docs.aclapi.apiary.io/
- Github: https://github.com/sandan/cs373-idb
- http://www.aclfestival.com/