Technical Report: The Austinites

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

August 7, 2014

Contents

1 Updates

Updates for the third iteration at a glance:

- Search bar and Search Page listing results 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
- An About page has been created to provide an overview of our website
- Splash page and web design have been tweaked to include better navigation and usability

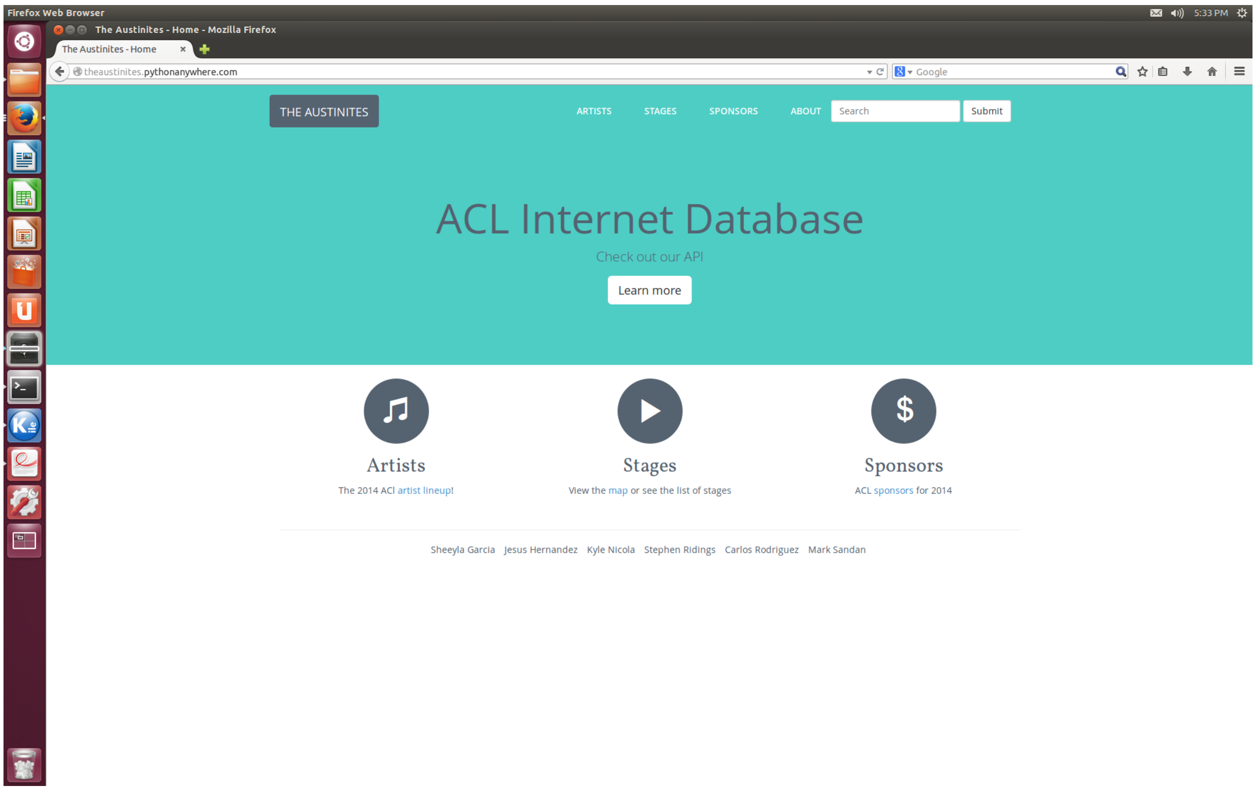

Firefox Web Browser
5:33 PM
The Austinites - Home - Mozilla Firefox
The Austinites - Home
theaustinites.pythonanywhere.com
Google
THE AUSTINITES
ARTISTS
STAGES
SPONSORS
ABOUT
Search
Submit
ACL Internet Database
Check out our API
Learn more
Artists
The 2014 ACl artist lineup!
Stages
View the map or see the list of stages
Sponsors
ACL sponsors for 2014
Sheeyla Garcia Jesus Hernandez Kyle Nicola Stephen Ridings Carlos Rodriguez Mark Sandan

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.

- Search capability implemented using Haystack and Whoosh
- Created a dynamic page implementing another group's API in order to test it
- Created a presentation that overviews all three phases of our concluding IDB project
- Refactored the stages to represent a physical location instead of their sponsor-given information (media) .

1.1 Summary

The main changes that have been made can be summarized into three main categories: search implementation, model refactoring, and website layout tweakings. In order to implement the search functionality we installed Haystack to utilize as the search framework and Whoosh as the search engine.

The models were drastically changed to include intermediary models. These models represented relationships and represent the concept of time, which would allow us to expand our database to include artists, sponsors and stages used over the years.

The website itself was also expanded to include an About page with information about our team and the Splash page was also updated. The tests for the new models and API 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 New Features

This section outlines a more detailed discussion for two new essential features.

2.1 Search

The search framework we used was Django Haystack (see the Links section) along with the Whoosh search engine. Using these tools we were able to construct a seach index on our django models and list the results in a designated search result page. Current features of search are listed below:

- Highlighted search terms for query
- Indexing across the Artist, Sponsor, and StageMedia models
- Lightbox pictures along with search results
- Ngram search
- Contextualized search by limiting scope of search to data either in Artist, Sponsor, or StageMedia.
- A count of the total results found
- a short bio (if the model provides it) slice of 250 characters.
- Search results are linked to the proper url

2.2 New Time Models

Two new models have been added to represent the concept of time. They are essentialy junction tables in a many to many relationship between Artists and Stages and also between Stages and Sponsor (see the UML diagram in figure 5). In addition to linking these classes together they have an attribute called date (see the Django Model detail page in sections 5.5.4 and 5.5.5).

Using these time relationships we are able to order entities by time as well as any other attributes they may have. As a consequence we are able to index a search on these dates to find content in our database by year, genre, label, industy, etc. The motivation for the new relational models is simply to make the website more useful and interesting (see Use Cases).

3 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.

3.1 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.
- Haystack: A simplified search framework that allows you to use third-party search engines as its core.
- Whoosh: a full-text indexing and searching library implemented in pure Python.

4 Use Cases

The Austin City Limits Databases focuses on the following use cases:

- Showcase details of Artists that play at the Austin City Limits music festival
- Allow users to get more information about the sponsors involved
- Allow users to see the history of which artists were sponsored by which sponsors
- Allow users to search the database to find cross-categorized information such as by time (year), genre (rock, hip hop , etc.) , industry (automobile, food/beverage, music/entertainment, etc.)

5 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. The design and content of each page is outlined below.

5.1 Web Pages

Each web page has basic information about a particular artist, sponsor, or stage involved in the ACL music festival. All pages 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.

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.

5.1.1 Splash Page

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

The "splash" page is 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. The search bar is also included and redirects the user to a new search results page listing the results. The splash page can be seen in Figure 1 above.

5.1.2 Artists Page

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

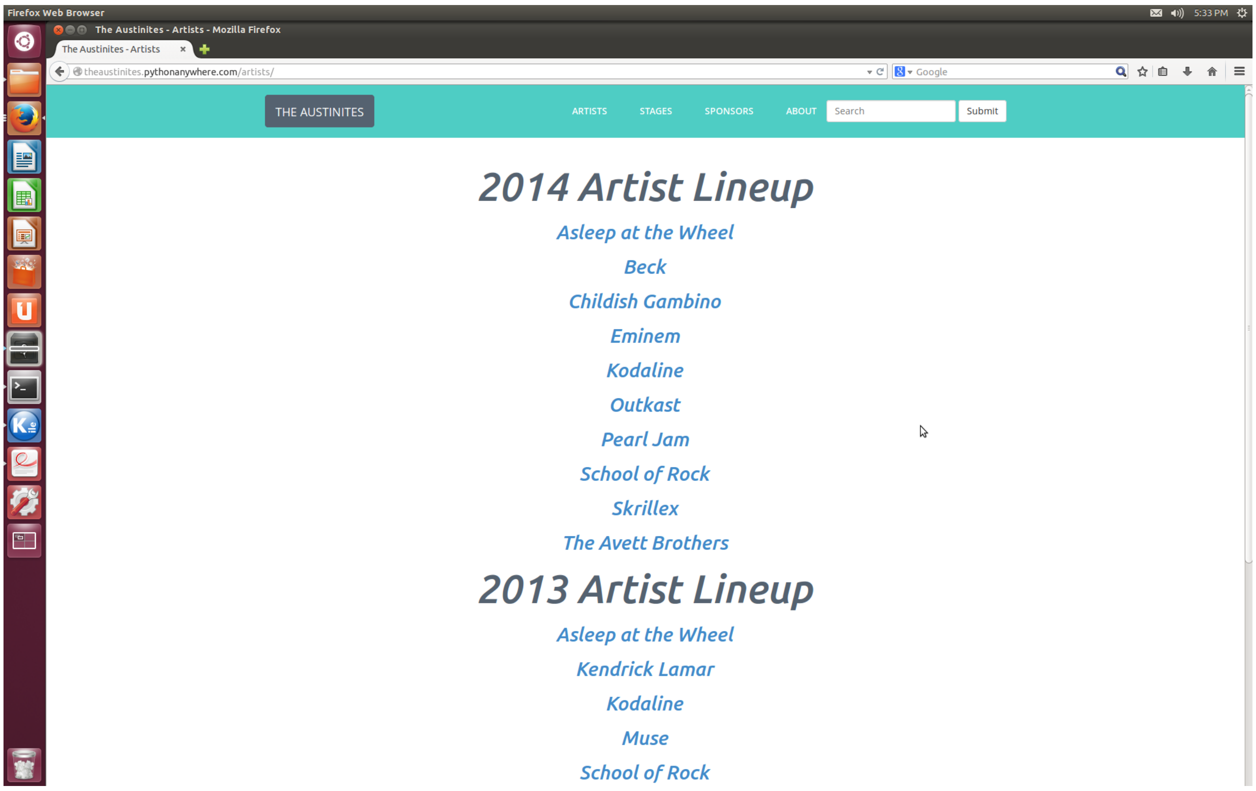
Artist Pages can be reached from the home page as well as from the Stage or Sponsor pages depending on whether the Artist played on a Stage for a given year that was hosted by a Sponsor.The page currently lists the dynamic web pages for music artists: 00 Each Artist page includes the Artist name, an 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.

5.1.3 Sponsors

- 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:

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.



2014 Artist Lineup

Asleep at the Wheel

Beck

Childish Gambino

Eminem

Kodaline

Outkast

Pearl Jam

School of Rock

Skrillex

The Avett Brothers

2013 Artist Lineup

Asleep at the Wheel

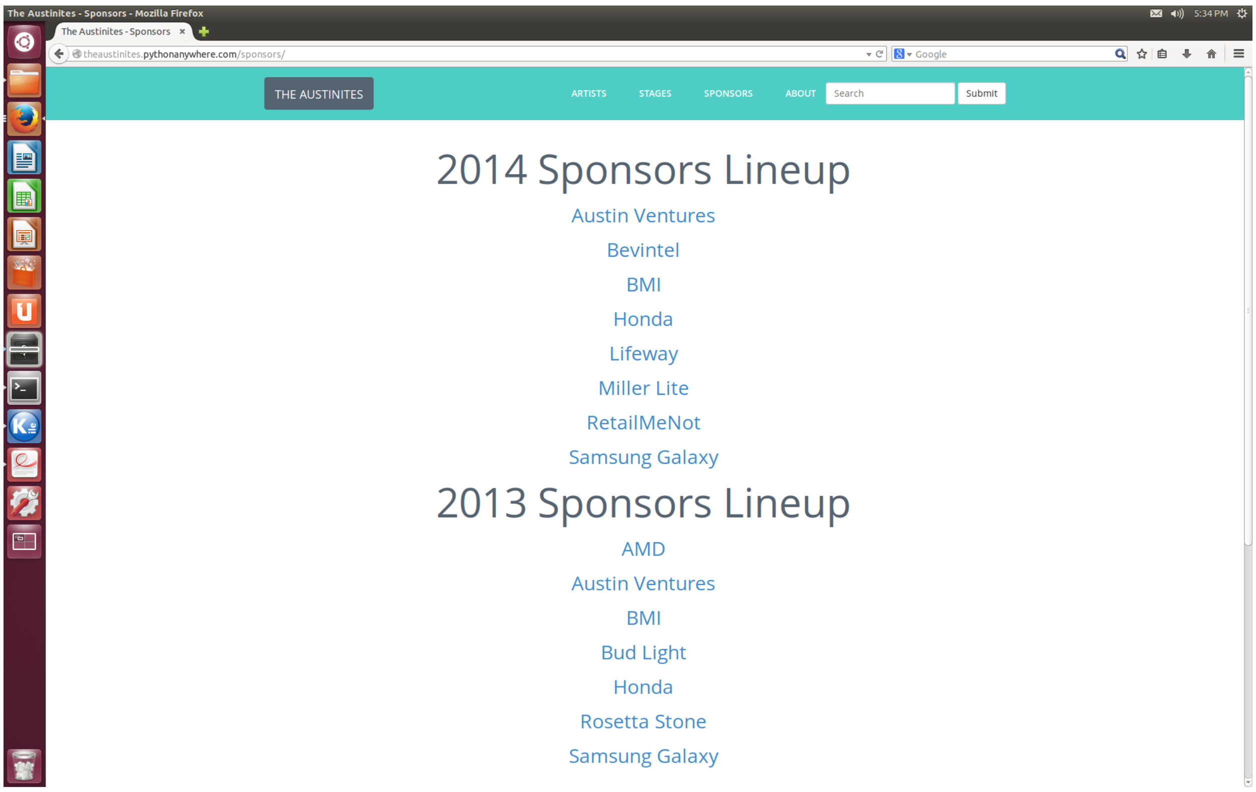
Kendrick Lamar

Kodaline

Muse

School of Rock

Figure 2: The current Artist Index page showcasing the lineup of artists by year


The Austinites - Sponsors - Mozilla Firefox
5:34 PM
The Austinites - Sponsors
theaustinites.pythonanywhere.com/sponsors/
Google
THE AUSTINITES
ARTISTS
STAGES
SPONSORS
ABOUT
Search
Submit

2014 Sponsors Lineup

Austin Ventures

Bevintel

BMI

Honda

Lifeway

Miller Lite

RetailMeNot

Samsung Galaxy

2013 Sponsors Lineup

AMD

Austin Ventures

BMI

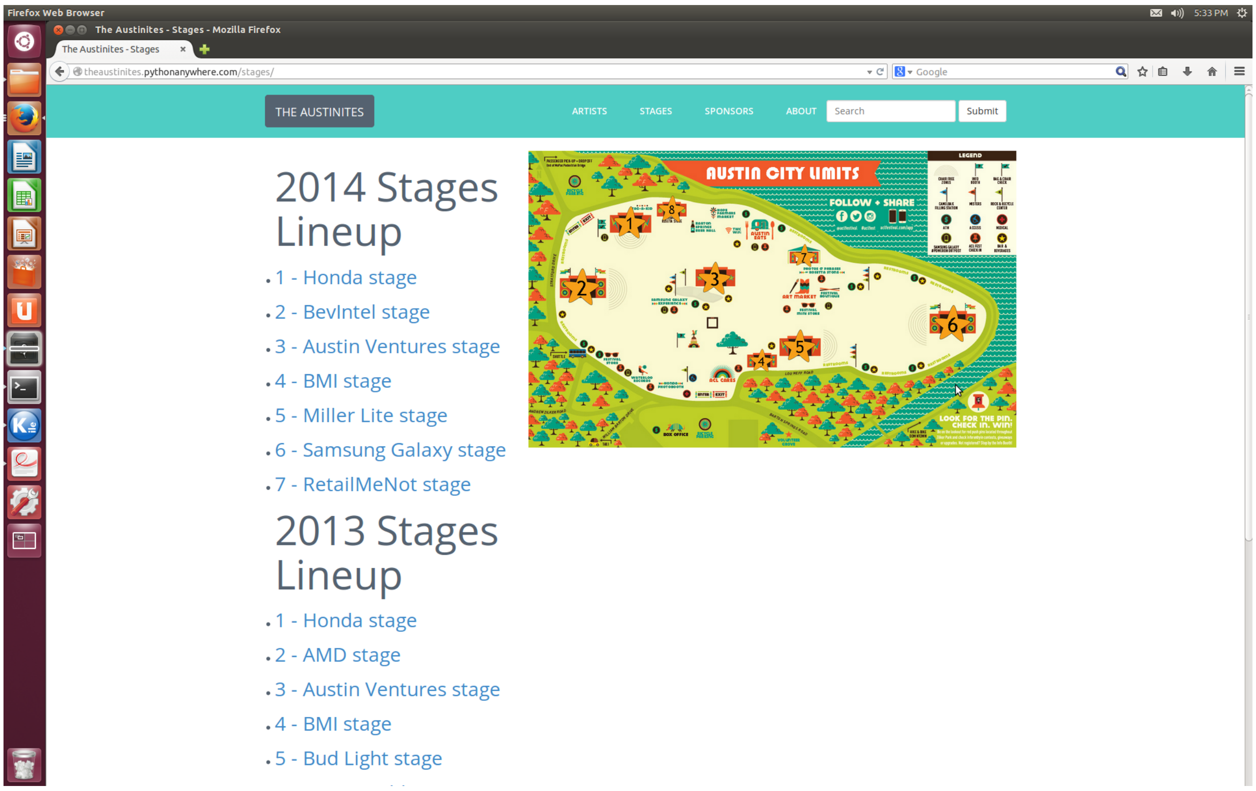
Bud Light

Honda

Rosetta Stone

Samsung Galaxy

Figure 3: The current Sponsor Index page lists the sponsors of stages by year



2014 Stages Lineup

- 1 - Honda stage
- 2 - BevIntel stage
- 3 - Austin Ventures stage
- 4 - BMI stage
- 5 - Miller Lite stage
- 6 - Samsung Galaxy stage
- 7 - RetailMeNot stage

2013 Stages Lineup

- 1 - Honda stage
- 2 - AMD stage
- 3 - Austin Ventures stage
- 4 - BMI stage
- 5 - Bud Light stage

Figure 4: 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.

5.1.4 Stages

- 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.

5.2 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 dynamic django template 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.

5.2.1 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.

5.2.2 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.

5.3 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.

5.3.1 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.

5.3.2 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.

5.3.3 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.

5.4 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.

5.4.1 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,
}
```

5.4.2 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",
}
```

5.4.3 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",


}
```

5.5 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 below 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 nine classes.

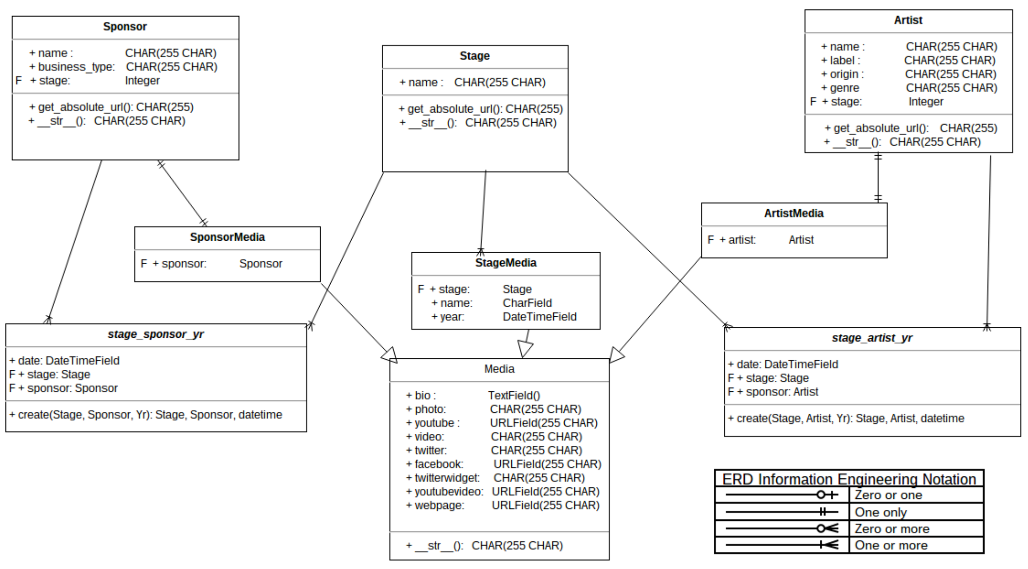

Sponsor
+ name : CHAR(255 CHAR)
+ business_type: CHAR(255 CHAR)
F + stage: Integer
+ get_absolute_url(): CHAR(255)
+ __str__(): CHAR(255 CHAR)
Stage
+ name : CHAR(255 CHAR)
+ get_absolute_url(): CHAR(255)
+ __str__(): CHAR(255 CHAR)
Artist
+ name : CHAR(255 CHAR)
+ label : CHAR(255 CHAR)
+ origin : CHAR(255 CHAR)
+ genre CHAR(255 CHAR)
F + stage: Integer
+ get_absolute_url(): CHAR(255)
+ __str__(): CHAR(255 CHAR)
ArtistMedia
F + artist: Artist
SponsorMedia
F + sponsor: Sponsor
StageMedia
F + stage: Stage
+ name: CharField
+ year: DateTimeField
stage_sponsor_yr
+ date: DateTimeField
F + stage: Stage
F + sponsor: Sponsor
+ create(Stage, Sponsor, Yr): Stage, Sponsor, datetime
Media
+ bio : TextField()
+ photo: CHAR(255 CHAR)
+ youtube : URLField(255 CHAR)
+ video: CHAR(255 CHAR)
+ twitter: CHAR(255 CHAR)
+ facebook: URLField(255 CHAR)
+ twitterwidget: CHAR(255 CHAR)
+ youtubevideo: URLField(255 CHAR)
+ webpage: URLField(255 CHAR)
+ __str__(): CHAR(255 CHAR)
stage_artist_yr
+ date: DateTimeField
F + stage: Stage
F + sponsor: Artist
+ create(Stage, Artist, Yr): Stage, Artist, datetime
ERD Information Engineering Notation
Zero or one
One only
Zero or more
One or more

Figure 5: 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. Th time models (junction tables) are also represented.

5.5.1 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.
methods:
- get_absolute_url(): returns the string "/artists/{id}" . Maximum number of 255 chara
- __str__(): returns the name string of the artist.Maximum  number of 255 characters
```

5.5.2 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
- industry: a character type field with a maximum length of 255 characters

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

5.5.3 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_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
```

5.5.4 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: the associated stage object being sponsored by the sponsor in the given date
- sponsor: The associated sponsor object sponsoring the stage in the given date

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
```

5.5.5 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
```

5.5.6 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
```

6 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.

6.1 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.

6.1.1 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_yr object to relate a Stage and an Artist
- test_types_relationships2: tests different combinations of creating the stage_sponsort_yr object to relate a Stage and a Sponsor

6.1.2 StageRelationship Test

Total tests: 2

- test_relationship1: creates a Stage and an Artist and relates them with the stage_artist_yr object. Tests accessing the Stage and Artist from the stage_artist_yr.
- test_relationship2: creates a Stage and a Sponsor and relates them with the stage_sponsor_yr object. Tests accessing the Stage and Sponsor from the stage_sponsor_yr.

6.1.3 Constraint Test

Total tests: 4

- test_constraint_violation1: Tests that an artist playing on two different stage locations for the same year throws an error (per the model of ACL)
- test_constraint_violation2: Tests that a sponsor sponsoring two different stage locations for the same year throws an error (per the model of ACL)
- test_no_constraint_violation1: Tests that an artist playing on two different stage locations for different years doesn't throw an error (per the model of ACL)
- test_no_constraint_violation2: Tests that a sponsor sponsoring two different stage locations for different years doesn't throw an error (per the model of ACL)

6.1.4 MultiRelation Test

Total tests: 4

- test_access_artist_stage0: Tests accessing the artist from a stage through a stage_artist_yr object by counting the number of artists associated to the stage.
- test_access_artist_stage1: Tests accessing the stage from the artist through a stage_artist_yr object by asserting the associated stage for all artists are the same.
- test_access_artist_stage2: Tests accessing the sponsor from a stage through a stage_sponsor_yr object by counting the number of sponsors associated to the stage.
- test_access_artist_stage3: Tests accessing the stage from the sponsor through a stage_sponsor_yr object by asserting the associated stage for all sponsors are the same.

6.1.5 CrossRelation Test

Total tests: 2

- test_cross_rel1: Tests retrieving stage, sponsor, artist through stage_artist_yr and stage_sponsor_yr from another related stage, artist, and sponsor
- test_cross_rel2: Tests retrieving stage, sponsor, artist from stage_artist_yr and stage_sponsor_yr objects

6.1.6 Simulations Test

Total tests: 1

- test_simulation: Creates a test simulation of accessing data from the models through the relationship objects and making sure the constraints are enforced.

6.2 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.

6.2.1 Stages API Test

Total tests: 9
The tests are separated into three classes below where different aspects of the Stage model is tested. **StagesAllAPITest** $/api/stages/$

- testValidStageGet: tests the status code of a successful client requesting GET for the first stage as well as the number of stages returned.
- testPostNotAllowed tests the status code is 405 (Method Not Allowed) for an unsuccessful POST request for the first stage.

StagesAPITest $/api/stages/location$

- testValidStageGet: tests that the response data is correct and the return code is an HTTP_200_OK
- testInvalidStageGet: tests that a GET to a non-existant stage returns the correct HTTP_404_NOT_FOUND response code
- testPostNotAllowed: tests that a POST to an existing stage returns the expected HTTP_405_METHOD_NOT_ALLOWED response code

StagesYearAPITest $/api/stages/location/year$

- testValidStageGet1: tests that
- testValidStageGet2: tests that
- testInvalidStageGet: tests
- testPostNotAllowed: tests that

6.2.2 Artist API Test

Total tests: 8
The tests are separated into three classes below where different GET requests to the Artist model is tested

ArtistAPITest $/api/artists/id$

- testStageGet: tests that the correct stage is returned for an artist for the given year and that the returned dictionary has the correct number of keys
- testStageGet2: tests that a GET request to an artist that doesn't exist returns the HTTP_404_NOT_FOUND response code
- testPost: tests that a POST to an existing artist is not allowed

ArtistYearAPITest $/api/artists/year/year/$

- testStageGet: tests that a GET to an artist for a specific year returns the correct data to the corresponding stage location
- testStageGet2013: tests that a GET request for a specific year (different than from above) returns the correct data to the corresponding stage location
- testStageGet2: tests that a GET to a non existing artist with a given year returns the HTTP_404_NOT_FOUND response code
- testPost: tests that a POST to an artist is not allowed by checking the 405 response code

ArtistsAllAPITest $/api/artists/$

- testArtistAllGet1: tests that the correct number of artists is returned for a call to all artists

6.2.3 Sponsors API Test

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

- testSponsorGet: tests the status code of a successful client requesting GET for the first sponsor
- testSponsorGet2: 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.

6.2.4 SponsorYear API Test

Total tests: 4
On setup one stage and two stage_sponsor_yr are created in the database and then deleted on tear down.

- testSponsorGet: tests the status code of a successful client requesting GET for a sponsor in year 2012.
- testSponsor2013: tests the status code of a successful client requesting GET for a sponsor in year 2013.
- testSponsorGet2: tests the status code is 404 (Not Found) for an unsuccessful GET request for a 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

6.2.5 SponsorsAll API Test

Total tests: 1
On setup three sponsors are created in the database and then deleted on tear down.

- testSponsorAllGet: tests the length and status code of a successful client requesting GET for a list of all sponsors

6.2.6 StageMedia API Test

$/api/stages/location/media/year/$
Total tests: 4
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_get1: tests to see if a client request for the stage media object created returns the 200 OK response code.
- test_stage_media_get2: tests to see if a client request for the stage media object created returns the 200 OK response code, using a different year than stage_media_get1.
- 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.

6.2.7 StageMediaList API Test

$/api/stages/location/media/$
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 list 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 list 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.

6.2.8 ArtistMedia API Test

/api/stages/id/year/
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.

6.2.9 SponsorMedia API Test

/api/stages/id/media/
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.

7 Six Potters API Testing

The way we chose to implement the required testing of another group's API was to generate random character quotes and their associated character stories. The test page is linked to the website's header for ease of navigation.

8 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/
- http://docs.maraudersmap.apiary.io/