

Technical Report: The Austinites

Sheeyla Garcia
Stephen Ridings

Jesus Hernandez
Carlos Rodriguez

Kyle Nicola
Mark Sandan

July 10, 2014

Contents

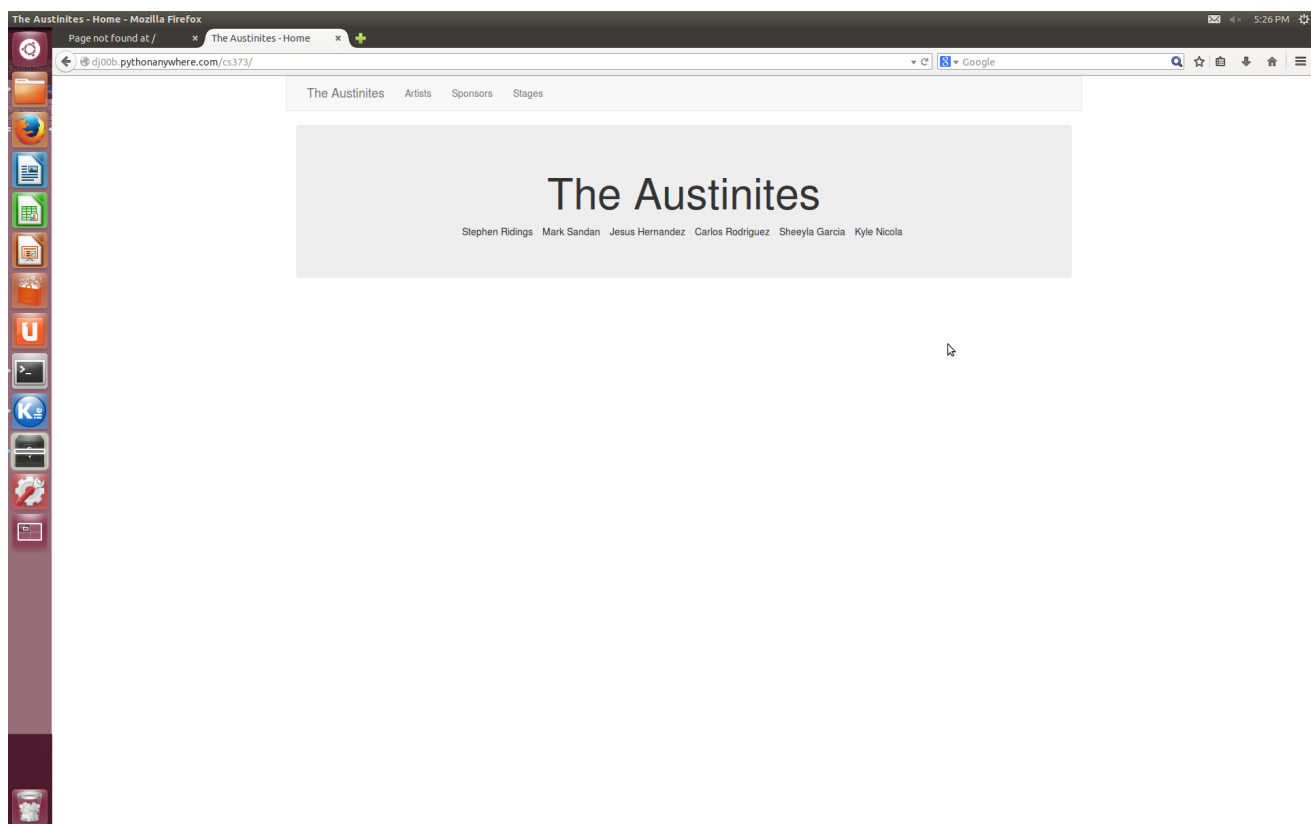


Figure 1: Example Home page showing the group name and the links to the three main pages.

1 Introduction

The website we are designing is about the 2014 Austin City Limits (ACL) music festival. The website design has three main pages: Artists, Stages, and Sponsors. The website allows anyone to view pages about current Artists, Sponsors, and Stages.

The technologies used are PythonAnywhere, Python 3.4, Django 1.6, Twitter Bootstrap 3.2, Apiary, and the database sqlite3. PythonAnywhere is used to host the site using the Django web-framework to construct the necessary models and views to handle HTTP

requests. Twitter Bootstrap is used to organize common data among different types of pages into a base template HTML file.

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 is being written. Currently we are using stubbed relationships to

2 Design

Our current design of the website uses HTML and Twitter Bootstrap to stylize each page and PythonAnywhere to host the page. 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 uses Twitter Bootstrap. We have nine pages in static HTML that provide examples of the web page design and content.

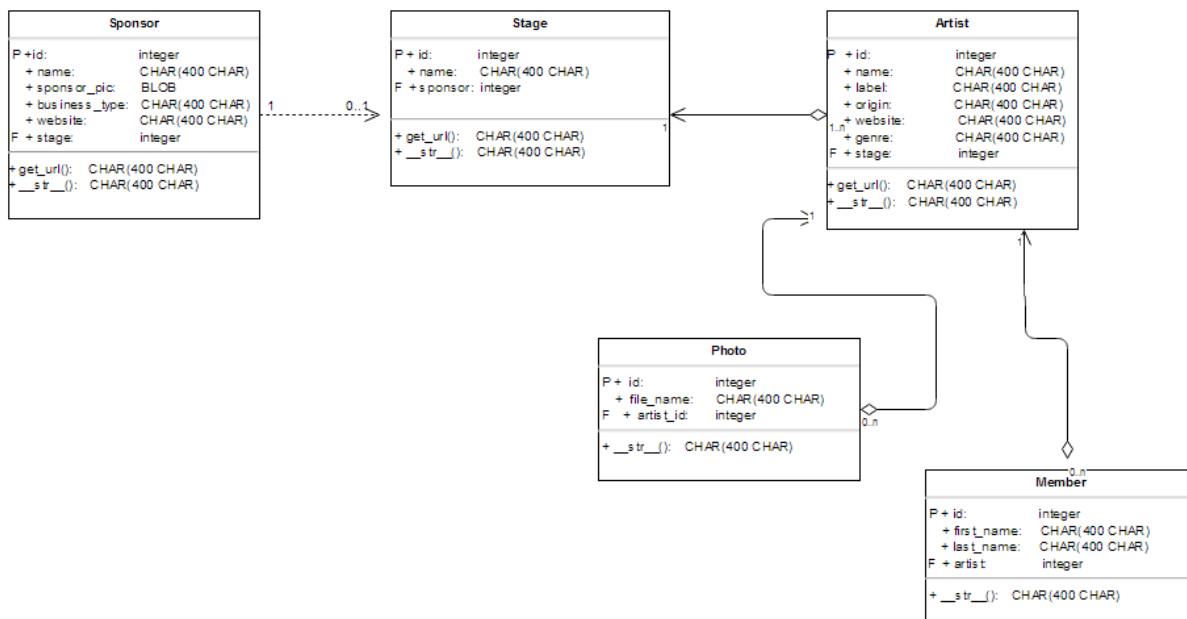


Figure 2: The current UML schema depicting the relationships between the Django models.

2.1 Web Pages

Each web page has basic information about a particular artist, sponsor, or stage involved in the ACL music festival. Each page includes a bio, video, official website, Youtube channel, a Twitter feed, and Picture. All pages will include a navigation bar at the top of their page 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.

2.1.1 Splash Page

The "splash" page will be the first page a visitor to the site will see. It will provide button style links to all subcategories (Artists, Sponsors, Stages).

2.1.2 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.

2.1.3 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.

2.1.4 Stages

Stages pages can be reached from the home page 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.

2.2 RESTful API

The API allows GET requests to the following models: Stages, Sponsors, Artists, Members, and Photos. The following section will detail the attributes for the modules, and how the server will respond to the GET requests.

2.2.1 Stages

When a GET is called on `[/stages]` it will return a HAL+JSON representation all the Stages in the database. It will be a list of the stages along with their attributes. When a GET is called on `[/stages/id]` it will return a HAL+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:

```
{
  "_links": {
    "self": { "href": "stages/42" }
  },
  "id": 42,
  "name": "Stage name"
}
```

2.2.2 Sponsor

When a GET is called on `[/sponsors]` it will return a HAL+JSON representation all the Sponsors in the database. It will be a list of the sponsors along with their attributes. When a GET is called on `[/sponsors/id]` it will return a HAL+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:

```
{
  "_links": {
    "self": { "href": "sponsors/42" }
  },
  "id": 42,
```

```

    "name": "Sponsor name",
    "business_type": "Type of Business",
    "website": "URL of sponsor website",
    "stage": 12
}

```

2.2.3 Artist

When a GET is called on `[/artists]` it will return a HAL+JSON representation all the Artists in the database. It will be a list of the artists along with their attributes. When a GET is called on `[/artists/id]` it will return a HAL+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:

```

{
  "_links": {
    "self": { "href": "artists/42" }
  },
  "id": 42,
  "name": "Artist name"
  "label": "Label of artist"
  "origin": "Where the artist was from"
  "website": "URL to the artist"
  "genre": "Genre of the artist"
  "stage": 22
}

```

2.2.4 Member

When a GET is called on `[/artists/artist_id/members/]` it will return a HAL+JSON representation all the Members in the database for the given artist id. It will be a list of the members along with their attributes. When a GET is called on `[/artists/artist_id/members/id]` it will return a HAL+JSON representation of a single Member in the database with the given id. It will list the member and all of it's attributes

Example of single member:

```

{
  "_links": {
    "self": { "href": "artists/22/members/42" }
  },
  "id": 42,
  "first_name": "Member's first name"
  "last_name": "Member's last name"
  "artist_id": 22
}

```

2.2.5 Photo

When a GET is called on `[/artists/artist_id/photos/]` it will return a HAL+JSON representation all the Photos in the database for the given artist id. It will be a list of the photos along with their attributes. When a GET is called on `[/artists/artist_id/photos/id]` it will

return a HAL+JSON representation of a single Photo in the database with the given id. It will list the photo and all of its attributes

Example of single photo:

```
{
  "_links": {
    "self": { "href": "artists/22/photos/42" }
  },
  "id": 42,
  "file_name": "photo.jpg"
  "artist_id": 22
}
```

2.3 Django Models

The Django models created represent the entities we intend to keep in the database for the future when use dynamically loaded pages. The following subsections document the attributes and intended functionality of each class instance method. When running the tests using the Django framework a temporary.

2.3.1 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. We are assuming that an Artist will play on exactly one stage. The Artist class is implemented using the following:

attributes:

- id: Primary Key, integer type field.
- name: the name of max length 400 characters.
- label: the artist label with a maximum length of 400 characters.
- origin: the place of origin the artist/group formed with a maximum length of 400 characters.
- website: the official website of the artist or fan site if none. Maximum length of 400 characters.
- genre: the genre associated with the artist. May span more than one. maximum length 400 characters.
- stage: Foreign Key, integer of type field.

methods:

- get.url(): returns the string "/artists/{id}" . Maximum of number of 400 characters.
- __str__(): returns the name string of the artist.

2.3.2 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 400 characters
- business_type: a character type field with a maximum length of 400 characters

- website: a character type field with a maximum length of 400 characters
- stage: Foreign Key, integer of type field

methods:

- get.url(): returns the string "/sponsors/{id}". Maximum number of 400 characters.
- __str__(): returns a string that represents the name of the sponsor.

2.3.3 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 400 characters
- sponsor: Foreign Key, integer of type field

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

2.3.4 Photo

The Photo class represents a path to a photo of an Artist. All Photo's are associated with an Artist in a many-to-one relationship. This is expressed with the Foreign key being an Artist that the Photo is associated with. The Photo class has the following attributes and methods:

attributes:

- file_name: A CharField of 400 characters. File name of the photo.
- artist: The ForeignKey relating a Photo to an Artist.

methods:

- __str__(): returns the file_name of the photo.

2.3.5 Member

The Member class is meant to represent the members of the Artist group. The Member class has the following attributes and methods:

attributes:

- first_name: A CharField of 400 characters.
- last_name: A CharField of 400 characters.
- artist: ForeignKey relating the Member to Artist. Integer type field.

methods:

- __str__(): returns the string of form "{first_name} {last_name}" if there is a last anme available. Otherwise "{first_name}" is returned.

3 Unit Tests

The unit tests currently implemented reflect the expected functionality of the django models and database. Currently we are using the sqlite3 database as a backend. We assume for each function in the set of tests that the state of the database is reset. The following give descriptions of each model test.

3.1 Stage Test

Total tests: 7

- `test_create_empty_stage`: Tests that uninitialized (stateless) Stages can be created.
- `test_create_stage` : Tests that the *name* attribute is set correctly when initialized.
- `test_get_stage`: Tests that the *get_stage* method returns the Stage instance we saved into the database. Also tests the filtering functionality of setting the id of the instance.
- `test_stage_name`: Tests the *max_length* property of CharField when setting the name of Stage.
- `test_stage_url_1`, `test_stage_url_2`, `test_stage_url_3`: Tests whether an instance of the desired Stage class is saved in the database table and returns the correct *url* attribute.

3.2 Artist Test

Total tests: 8

- `test_create_artist`: Tests the Artist instance attributes actually match the given values that were passed when constructed (including a Stage instance)
- `test_get_artist`: Tests that an instance of Artist can be retrieved from the database with the attributes used to create it.
- `test_empty_artist`: Tests that an IntegrityError is thrown when an Artist instance is saved in the database without setting a ForeignKey.
- `test_empty_artist_with_stage`: Tests the default values of an Artist instance with a ForeignKey.
- `test_artist_name`: Tests the *max_length* property of CharField when setting the name of Artist.
- `test_artist_url_1`, `test_artist_url_2`, `test_artist_url_3`: Tests that the many-to-one relationship between Artists and Stages is valid. Also tests the *get_url* method returns the correct url according the id of the instance.

3.3 Sponsor Test

Total tests: 6

- `test_empty_sponsor`: Tests that a Sponsor's default values and ID are set correctly.
- `test_save_sponsor`: Tests that the name matches the passed value given when the Sponsor instance was created.
- `test_get_sponsor_name`: Tests the *max_length* property of CharField when setting the name of Sponsor.
- `test_sponsor_url_1`,
- `test_sponsor_url_2`: Tests that the *get_url* is returned in the proper format.

3.4 Member Test

Total tests: 5

- `test_empty_member`: Tests that a Member can be made with no exceptions when created with an Artist. Tests the default values of Member are empty.
- `test_get_empty_member`: ???
- `test_create_member`: Tests that the attributes are correctly set in Member.
- `test_get_member`: : Creates an instance of Stage, Artist and Member with particular values some attributes. They are all saved into the database after being created. It then asserts that the Member instance is not of type None and that all the values passed when constructed are actually the values.
- `test_str_member`:: Creates an instance of Stage, Artist and Member with particular values some attributes. They are all saved into the database after being created. A particular row with a private key is then requested and It then asserts that the the values are equal to the values passed when constructed.

3.5 Photo Test

Total tests: 4

- `test_empty_photo`:
- `test_create_photo`:
- `test_get_photo`:
- `test_str_photo`: