Previous Next

Create a Song Management API

Estimated Time: 4-5 hours

In this assignment you are going to create a basic API for storing and retrieving songs for the application.

Like in the previous lesson there is a repository containing the boilerplate code for the application available. Because the focus of this course is on Python and the server side, the repository also contains the client side code that you'll use to interact with the API. This client side code expects to pull data from API endpoints on the server that you'll write together in this assignment.

Notice how the client-side code is simply served from the *static* directory. In *tuneful/views.py* you have a single route for serving the static *index.html* page which uses the `send_static_file` function so you can avoid the overhead of templating.

To help you to understand a little of what the client-side of the code is doing, have a read through *tuneful/static/js/main.js*. Some of the JavaScript syntax will be unfamiliar to you, but by reading through the comments you should be able to get a high-level understanding of what the code is doing.

Now that you are comfortable with the code let's get started. First of all you should:

1. Create a clone of the repository (`git clone https://github.com/oampo/pip-tuneful-project-template tuneful`)
2. Move into the project's directory (`cd tuneful`)
3. Set up a virtualenv for the project (`python3 -m venv env`)
4. Activate the virtualenv (`source env/bin/activate`)
5. Install the project's dependencies (`pip install -r requirements.txt`)
6. Set up a database for the project (`createdb tuneful`)
7. Set up a test database for the project (`createdb tuneful-test`)

Try running the app using `python run.py` . If you visit the index URL you should see a basic (but non-functioning) user interface. Try opening up the developer console by hitting F12, and selecting the Console tab. You should see that the front-end is failing to complete its requests to the API. Let's start building our simple song management API to fix that. This can be split down into three tasks.

The database

In the file *tuneful/models.py* we are going to need two SQLAlchemy models.

1. The Song model: This should have an integer id column, and a column specifying a one-to-one relationship with a File.
2. The File model: This should have an integer id column, a string column for the filename, and the backref from the one-to-one relationship with the Song.

Each model should have an `as_dictionary` method. The `Song.as_dictionary` method should produce a dictionary which looks something like this:

```
{
    "id": 1,
    "file": {
        "id": 7,
        "name": "Shady_Grove.mp3"
    }
}
```

The `File.as_dictionary` method should just return the `file` element of the song dictionary (i.e. `{"id": 7, "name": "Shady_Grove.mp3"}`).

Try to create those two models, and use Python's interactive interpreter to experiment with adding Song and Files and creating relationships between them.

The GET endpoint

In the *tuneful/api.py* file we need a GET endpoint for `/api/songs` which returns a list of all of the songs as JSON. Try to follow the practice laid out in the previous lesson as you create this, adding tests as you go and trying to make your endpoint well-behaved for clients.

The POST endpoint

In the *tuneful/api.py* file add a POST endpoint for `/api/songs` which allows you to add a new song to the database. The endpoint should look for JSON POST data in the following format:

```
{
    "file": {
        "id": 7
    }
}
```

You should use the file ID to make sure that the file exists in the database, and make the relationship with your new song. Again you should try to follow the

good practice for working with POST data, and write tests to make sure that your endpoint is working as you expect.

Testing it out

Now that you have the beginning of an API try adding a couple of example `File` and `Song` object to the database from the interactive Python interpreter and then visit the front-end. You should be able to see the songs listed in the left column of the site.

While you are in the front-end, try adding a song using the `Add Song` button. You should see a failed request to the `/api/files` endpoint in the console as it tries to upload the file. In the next assignment you'll be looking at how file upload works on a single-page web app and writing the endpoints to get this up and running on your site.

Extension task

Add PUT and DELETE endpoints to your API allowing you to edit and delete songs from the app.

☆ ☆ ☆ ☆ ☆ · Report a typo or other issue

Mark as completed

Previous Next