UDACITY Logout

‹ Return to "Full Stack Web Developer Nanodegree" in the classroom

DISCUSS ON STUDENT HUB ›

Fyyur: Artist Booking Site

REVIEW | CODE REVIEW 5 | HISTORY

Meets Specifications

Congratulations by the excellent effort

I really commend you for this project. You have demonstrated great skills, and proven your knowledge in Python, ask SQLAlchemy and PostgresSQL database.

1. Your code is simple and easy to understand.
2. Your README is well detailed and instructive.
3. Your separation of concerns is good.

You are definitely on the right track. Please take out time to check the Code review for more improvements where necessary.

I wish you the best in future projects.

Kind regards

Data Models

✓ **Correct data types are associated with each field.**

The `Shows` object has a relationship that connects Artists and Venues, and this relationship is of the correct type. In other words, the project demonstrates the ability to appropriately select from the following types of relationships:

- **One-to-one**
- **One-to-many**
- **Many-to-many**

The project demonstrates the ability to appropriately select from the Many-to-many relationship and connects the tables by their primary IDs.

✓ **The code creates a local postgresql database connection.**

Nice. Your code creates a local PostgreSQL database and connects without any issue.

✓ **The `Shows` model has properly set up foreign keys.**

The Artists and Venues models are in third normal form.

Project demonstrates a good grasp of database normalization.

✓ **The code uses SQLAlchemy syntax to completely define the models.**

The code has accurate SQL queries wrapped in SQLAlchemy commands per API endpoint, calling to define data models and serving expected responses per API endpoint.

The code only uses raw SQL where SQLAlchemy wrappers do not suffice, otherwise minimizing use of raw SQL.

Excellent. The model definition is well structured with correct SQLALchemy syntax.

SQL

✓ **The code successfully translates SQLAlchemy code for selecting records from the database into the equivalent PostgresSQL command(s) for selecting records from the database.**

The code demonstrates correct use of `SELECT` and `WHERE` query statements to execute search successfully.

Project Uses SQL syntax to select records from a database.

✓ **`JOIN` statements are used to correctly execute joined queries.**

The code joins tables from existing models to select Artists by Venues where they previously performed, successfully filling out the Venues page with a "Past Performances" section.

The code joins tables from existing models to successfully fill out the Artists page with a "Venues Performed" section.

he code includes correct equivalents in SQL for all corresponding SQLAlchemy statements.

The project uses SQL syntax to correctly `JOIN` relational tables and joint queries. Nice!

✓ **Code connects the New Artist and New Venue forms to a database by successfully using SQLAlchemy to insert new records into the database upon form submission.**

Understands the equivalent SQLAlchemy command in SQL syntax, using INSERT INTO.

Code correctly uses SQL constraints to ensure fields that need to be unique, and fields that are required, are given these constraints on the database level, throwing an error if otherwise.

New records are successfully created to the database upon a form request, and forms are well-formatted with proper validation rules.

Application Quality & Deployment

✓ **Code is decoupled into relevant parts across the files.**

The code includes good use of comments where there is lack of clarity. Where comments are not provided, the code is self-documenting.

Encapsulate querying code in proper places across Models and API endpoints.

1. The project is well decoupled into relevant parts using the principle of separation of concerns.
2. The project makes use of appropriate Models and API endpoints with proper query argument.

✓
- **There are no build or compilation errors in running code and launching the web app.**
- **A user can successfully execute a Search that queries the database.**
- **A user can view a Venue Page with venue and artist information from the database.**
- **A user can view an Artist Page with venue and artist information from the database.**
- **A user can create new venue listing via the New Venue Page.**
- **A user cannot submit an invalid form submission (e.g. using an invalid State enum, or with required fields missing; missing city, missing name, or missing genre is not required).**
- **A user can create new artist listings via the New Artist Page.**
- **A user cannot submit an invalid form submission (e.g. without required fields)**
- **A user can search for an artist from the venue page, and choose them for a show, specifying a date-time.**

The application works as expected.

DOWNLOAD PROJECT

5 CODE REVIEW COMMENTS ›

RETURN TO PATH