# Workshop: Music App

## Skeleton

You are provided with the project's needed **HTML** pages, **images**, and **CSS**.

## Setup

### Creating the Django Project

Create a new **Django** project called "**musicApp**".

### Creating the App

After creating the **Django** project, we are ready to create the **app** we will work with. For instance, we can name it "**musics"**.

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

For clarification, **move the created app inside** the project:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

### Configurations

We need to **add the app** we just created in the **INSTALLED\_APPS** setting:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

### Adding the Templates

The next step is to **move the already-configured folders with HTML templates inside the templates folder.**

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

### Adding the Static Files

Add the provided folders ("**images**" and "**styles**") to the directory. Next, **Django** should find the static files when loading web pages, so write the setting in the **settings.py** file:

Картина, която съдържа текст, Шрифт, софтуер, номер

Описанието е генерирано автоматично

### Adding the URLs (paths)

We want to load each template in the browser using a concrete path - each app should load its templates.

To do that, we should add **urls.py** files in each app:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Then, we can start including them in the main project **urls.py** file. We should import the **include()** function from the Django **urls** module, then we can use the **path()** function to **construct a path**, which will **lead to the app** **urlpatterns**.

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

## Setup SQLAlchemy and Alembic

### Installing SQLAlchemy

**Install** **SQLAlchemy** using the command **"pip intsall sqlalchemy"**:

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

Install the **PostgreSQL** **driver** (as we will be using PostgreSQL as our project database system) using the **command** **"pip install psycopg2"**.

### Configuring SQLAlchemy

* **DATABASE\_URL**: This is a string that specifies the database connection details. In your example, it's set for a PostgreSQL database. The format is **'dialect+driver://username:password@host/database\_name'**.
* **engine = create\_engine(DATABASE\_URL)**: This line creates a SQLAlchemy database engine using the **create\_engine** function. The engine represents a connection to the database and is used to execute SQL queries.
* **Session = sessionmaker(bind=engine)**: This line creates a **Session** class that's used to manage the database transactions. The **bind** parameter specifies which database engine the session should use for its operations.

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

### Installing and initializing Alembic

As our application evolves, we may need to make changes to our database schema. These changes can include adding or modifying tables, columns, indexes, and relationships. Alembic provides a structured way to define these changes in Python code and apply them to the database, ensuring consistency across different database instances (e.g., development, staging, production).

Let us **install the Alembic package** in Django using the **command** **"pip install alembic"**:

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

However, we need to **initialize an Alembic migration environment** for our project. Let us type down the command **"alembic init alembic"**:

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

This way Alembic will create a directory named **"alembic"** in our project's root directory. Inside this directory, Alembic will **create several subdirectories** and **files**, including a **versions directory** for storing migration scripts, a **script.py.mako** **template file** for creating new migrations, and an **alembic.ini** **configuration file**:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

### Configuring Alembic

The **alembic.ini** file is an important configuration file where we will **specify details about our database** **connection**, which will **allow Alembic to** **connect to our database and apply migrations**. Let us open the file and find the **sqlalchemy.url** configuration setting. We will change the connection string to the one we use for this project:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

The **env.py** file serves as the **configuration and entry point for managing database migrations** in a Python project. In this file, we will **specify the database schema** that Alembic **should compare against** when generating database migration scripts. We will set the **target\_metadata** variable to the metadata attribute of the **Base** class:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

## Database

You will need **2 models**:

* **Album**
  + **ID** (Primary Key):
    - Data Type: **Integer**
    - Description: A unique identifier for each album.
  + **Album Name**:
    - Data Type: **String(30)**
    - Description: The name of the album. This field is required, and it must not be left empty. Additionally, it is marked as unique to ensure that no two albums have the same name.
  + **Image URL**:
    - Data Type: **String**
    - Description: A field to store the URL of the album's image. This field is required, and it must not be left empty.
  + **Price**:
    - Data Type: **Float**
    - Description: A field to store the price of the album. This field is required, and it must not be left empty.
  + **Songs**:
    - Description: A relationship to the "**songs**" table, allowing each album to have multiple associated songs. When an album is deleted, its associated songs are also deleted.
* **Song**
  + **ID** (Primary Key):
    - Data Type: **Integer**
    - Description: A unique identifier for each song.
  + **Song Name**:
    - Data Type: **String**
    - Description: A field to store the name of the song.
  + **Album ID**:
    - Data Type: **Integer**
    - Description: A foreign key relationship to the "**albums**" table, indicating which album the song belongs to. This field is required, and it must not be left empty.
  + **Album**:
    - Description: A bidirectional relationship to the "**albums**" table, allowing you to navigate from a song to its associated album. Changes in the **Song** object are reflected in the corresponding **Album** object, and vice versa.

### Defining Models

It is time to **define our models** using SQLAlchemy's ORM. In the **models.py** we will **import** the **declarative\_base** function used to create a **base class** for defining the database tables as **Python classes**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

### Using Alembic

Now, it is time to create a migration and apply it to the database. First, let us generate a new migration script in Alembic with the terminal command **"alembic revision --autogenerate -m "add albums and songs tables"**:

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

Here is a breakdown of what this command does:

* **"alembic revision"** - the Alembic command for **creating a new migration revision**.
* **"--autogenerate"** - the flag that tells Alembic to **automatically generate the migration script** by comparing the current state of the database to the state defined in our SQLAlchemy models.
* **"-m "add albums and songs tables"** - the flag is used to **provide a message** for the migration.

When the **migration script is successfully generated**, it will appear in the **versions** directory:  
Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

Next, let us apply the pending migration with the terminal command **"alembic upgrade head"**: Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

Here is a breakdown of what this command does:

* **"alembic upgrade"** - the Alembic command used to **apply database migrations**.
* **"head"** - the keyword represents the **latest available version of the database schema**. This way we are telling Alembic to apply all pending migrations up to the latest version of the database schema.

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

## Routes

* <http://localhost:8000/> - index page.
* <http://localhost:8000/album>/create/ - create album page.
* [http://localhost:8000/album/details/<id](http://localhost:8000/album/details/%3cid)>/ - album details page.
* [http://localhost:8000/album/edit/<id](http://localhost:8000/album/edit/%3cid)>/ - edit album page.
* [http://localhost:8000/album/delete/<id](http://localhost:8000/album/delete/%3cid)>/ - delete album page.
* <http://localhost:8000/song>/create/ - create song page.

## Views

In **Django**, views are an essential component of web applications that handle the logic for processing and responding to **HTTP** requests.

Views in **Django** are Python functions or classes that take a web **request** as input and return an HTTP response as output. These views are responsible for fetching data from the database, rendering templates, and performing various operations based on the **request**.

To create **views** corresponding to the specified routes in a **Django** web application, we would typically define view functions or classes that handle each of these URLs. Here's how we can create views for the given **routes**:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

## URLs

Inside the **musics** app directory, create a Python file named **urls.py**. In this file, you define URL patterns that specify which **views** should be called for different URL **paths**. The **urls.py** file should contain a variable called **urlpatterns** that is a list of URL patterns. For example, your **musics/urls.py** might look like this:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

### Creating Template Inheritance

If we look closely at each template, we can see that there are many common parts. The head, the header with the navigation bar, and the footer are the same for all templates. We can export them in a separate **.html** file in the project's **template** directory.

Let us open a **base.html** template in the **common** directory and make it a base template:

<!DOCTYPE html>  
<html lang="en">  
  
<head>  
 <meta charset="UTF-8"/>  
 <title>My Music App</title>  
 *<!-- Static Load -->* <link rel="stylesheet" href="/static/css/style.css"/>  
</head>  
  
<body>  
<div id="box">  
  
 *<!-- Navigation Bar -->* <header>  
 <nav>  
 <img src="/static/images/headphones.png" alt="headphones">  
 <a href="#">Home</a>  
 <ul></ul>  
 <ul>  
 <li><a href="#">Create Album</a></li>  
 <li><a href="#">Create Song</a></li>  
 </ul>  
 </nav>  
 </header>  
 *<!-- End Navigation Bar -->* {% block content %}  
 {% endblock %}  
  
 *<!-- Footer -->* <footer>  
 <div>  
 &copy;SoftUni Team 2023. All rights reserved.  
 </div>  
 </footer>  
 *<!-- End Footer -->*</div>  
</body>  
</html>

In Django templates, **{% block %}** and **{% endblock %}** are template tags used to define and delimit named content blocks within a template. These blocks are a fundamental part of the template inheritance system, which allows you to create a base template and override specific parts of it in child templates.

To inherit this base HTML code in every template we need to **extend** the **base.html** file (for this example we are going to work with the **index.html**):

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!-- Catalog with Albums-->* <section id="catalogPage">  
  
 *<!-- If No albums in catalog -->* <p>No Albums in Catalog!</p>  
  
 *<!-- If albums in catalog -->* <h1>All Albums</h1>  
 *<!-- First Card-Box in catalog -->* <div class="card-box">  
 <img src="/static/images/BrandiCarlile.png" alt="Cover Image">  
 <div>  
 <div class="text-center">  
 <p class="name">Name: In These Silent Days</p>  
 <p class="price">Price: $12.80</p>  
 </div>  
 <div class="btn-group">  
 <a href="#">Details</a>  
 </div>  
 </div>  
 </div>  
 *<!-- End First Card-Box in catalog -->  
  
 <!-- Second Card-Box in catalog -->* <div class="card-box">  
 <img src="/static/images/pinkFloyd.jpg">  
 <div>  
 <div class="text-center">  
 <p class="name">Name: The Dark Side of the Moon</p>  
 <p class="price">Price: $28.75</p>  
 </div>  
 <div class="btn-group">  
 <a href="#">Details</a>  
 </div>  
 </div>  
 </div>  
 *<!-- End Second Card-Box in catalog -->  
  
 <!-- Third Card-Box in catalog -->* <div class="card-box">  
 <img src="/static/images/Lorde.jpg">  
 <div>  
 <div class="text-center">  
 <p class="name">Name: Melodrama</p>  
 <p class="price">Price: $7.33</p>  
 </div>  
 <div class="btn-group">  
 <a href="#">Details</a>  
 </div>  
 </div>  
 </div>  
 *<!-- End Third Card-Box in catalog -->* </section>  
  
{% endblock %}

### Loading Static files

In Django, the **load** **static** template tag is used to load and render static files, such as CSS, JavaScript, images, and other assets, in a Django template. Static files are typically used for styling your web pages and adding client-side functionality to your website (here we are working in the **base.html**). **Apply these changes in every template.**

Картина, която съдържа текст, екранна снимка, номер, софтуер

Описанието е генерирано автоматично

## Pages

### Base Template

**Template file**: **"base.html"**

The template is a **base** for **all the templates** in the project. It consists of the **metadata** for the project (in the <head> tag), the **navigation** **bar**, and the **footer** of each web page. On the navigation bar:

* The **"Home"** button leads to the **index** page.
* The **"Create Album"** button leads to the **create album** page.
* The **"Create Song"** button leads to the **create song** page.

There is **no path** to this template, and it is **NOT accessible** via **URL**:

Картина, която съдържа екранна снимка, кръг, текст

Описанието е генерирано автоматично

### Index Page

**Template file**: **"index.html"**

When there are **no albums created**, you can **only** **see** the paragraph **"No Albums in Catalog!"**:

Картина, която съдържа екранна снимка, текст, кръг

Описанието е генерирано автоматично

If there are **album/s, the page should have the following**:

* A navigation bar, as shown below.
* A headline **"All Albums"**
* A card-box for each album, showing the album's **image**, **name,** and **price.** There should be a **button** **"Details"** in each card-box leading to the **album** **details page** for the selected album.

Картина, която съдържа текст, екранна снимка, Мултимедиен софтуер, мултимедия

Описанието е генерирано автоматично

View:

**It is necessary to start our session.**

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

This Django view uses the **SQLAlchemy** **Session** to query all albums from the database, then renders an HTML template (**'common/index.html'**) with the retrieved album data, sending it as context.

We also need to display the need information as we modify the **index.html**.

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!-- Catalog with Albums-->* <section id="catalogPage">  
  
 {% if not albums %}  
 *<!-- If No albums in catalog -->* <p>No Albums in Catalog!</p>  
 {% else %}  
 *<!-- If albums in catalog -->* <h1>All Albums</h1>  
 {% for album in albums %}  
 <div class="card-box">  
 <img src="{{ album.image\_url }}" alt="Cover Image">  
 <div>  
 <div class="text-center">  
 <p class="name">Name: {{ album.album\_name }}</p>  
 <p class="price">Price: ${{ album.price }}</p>  
 </div>  
 <div class="btn-group">  
 <a href="{% url 'details album' id=album.id %}">Details</a>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
 {% endif %}  
  
 </section>  
  
{% endblock %}

### Create Album Page

Template file: **"create-album.html"**

This page loads an **album** **creation form consisting of**:

* An **"Album Name:"** label.
  + The **album name** field must have **max length of 30.**
* An **"Image URL:"** label.
* A **"Price:"** label.
  + The **price** cannot be below **0.0**.
* A button **"Add New Album"**
  + When you **click** on it, **if** the album is **successfully created**, you should be redirected to the **index page**.
  + Otherwise, the form should show the **appropriate validation errors**.

Картина, която съдържа екранна снимка, текст, електроника, кръг

Описанието е генерирано автоматично

View:

Картина, която съдържа текст, екранна снимка, номер, Шрифт

Описанието е генерирано автоматично

In Django, when you submit a form, the form data is typically processed and validated by the Django form class. The **cleaned\_data** attribute of a form is a dictionary containing all the form fields and their corresponding cleaned values.

Form:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

Template:

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!--Create Page-->* <section class="createPage">  
  
 <form method="POST" action="{% url 'create album' %}">  
 <fieldset>  
 <legend>Add Album</legend>  
 <div class="container">  
  
 {{ form }}  
 {% csrf\_token %}  
 <button class="add-album" type="submit">Add New Album</button>  
  
 </div>  
 </fieldset>  
 </form>  
 </section>  
  
{% endblock %}

### Album Details Page

Template file: **"album-details.html"**

This page contains an album's data. It should have the following:

* The **album** **image**
* The **album name**
* The **related songs, joined** by **" | ".**
* The **price** of the album
* An **"Edit"** button that leads to the **edit album page**
* A **"Delete"** button that leads to the **delete album page**

Картина, която съдържа екранна снимка, текст, Мултимедиен софтуер

Описанието е генерирано автоматично

View:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Template:

{% extends 'common/base.html' %}  
{% block content %}  
  
  
 *<!--Details Page-->* <section id="detailsPage">  
 <div class="wrapper">  
  
 <div class="albumCover">  
 *<!-- Album Image -->* <img src="{{ album.image\_url }}" alt="Cover Image">  
 </div>  
  
 <div class="albumInfo">  
  
 <div class="albumText">  
 <h1>Name: {{ album.album\_name }}</h1>  
 <h4>Price: ${{ album.price }}</h4>  
 <h4>Songs:  
 {% for song in album.songs %}  
 <span>{{ song.song\_name }}</span>  
 {% if not forloop.last %} | {% endif %}  
 {% endfor %}  
 </h4>  
 </div>  
  
 <div class="actionBtn">  
 *<!-- Album Buttons -->* <a href="{% url 'edit album' id=album.id %}" class="edit">Edit</a>  
 <a href="{% url 'delete album' id=album.id %}" class="remove">Delete</a>  
 </div>  
  
 </div>  
 </div>  
 </section>  
  
{% endblock %}

### Edit Album Page

Template file: **"edit-album.html"**

On the page, the form must be **filled** with the **data** of the **album** we want **to edit.**

When you click on the **"Edit Album"** button:

* **If** the album is **successfully edited**,you should be redirected to the **home page**, showing a template for a home page with a profile.
* Otherwise, the form should show the **appropriate validation errors**.

Картина, която съдържа екранна снимка, текст, електроника, кръг

Описанието е генерирано автоматично

View:

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

Form:

Картина, която съдържа текст, Шрифт, екранна снимка, бял

Описанието е генерирано автоматично

Template:

{% extends 'common/base.html' %}  
{% block content %}  
  
  
 *<!--Edit Page-->* <section class="editPage">  
 <form method="POST" action="{% url 'edit album' id=album.id %}">  
 <fieldset>  
 <legend>Edit Album</legend>  
 <div class="container">  
 {% csrf\_token %}  
 {{ form }}  
 *<!-- Button for Editing Album -->* <button class="edit-album" type="submit">Edit Album</button>  
 </div>  
 </fieldset>  
 </form>  
 </section>  
  
{% endblock %}

### Delete Album Page

Template file: **"delete-album.html"**

On the page, the form must be **filled** with the **album's data.** When you click on the **"Delete"** button, the **album** is **deleted from the database**, and you should be redirected to the **index page**.

The deleted album should be **no longer visible in the app**.

Картина, която съдържа екранна снимка, текст, електроника, кръг

Описанието е генерирано автоматично

View:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

Form:

Картина, която съдържа текст, Шрифт, екранна снимка, линия

Описанието е генерирано автоматично

Template:

{% extends 'common/base.html' %}  
{% block content %}  
  
 *<!--Delete Page-->* <section class="editPage">  
 <form method="POST" action="{% url 'delete album' id=album.id %}">  
 <fieldset>  
 <legend>Delete Album</legend>  
 <div class="container">  
 {% csrf\_token %}  
 {{ form }}  
 <button class="delete-album" type="submit">Delete Album</button>  
 </div>  
 </fieldset>  
 </form>  
 </section>  
{% endblock %}

### Create Song Page

Template file: **"song-album.html"**

This page loads an **album** **creation form consisting of**:

* A **"Song Name:"** label.
* An **"Album:"** label.
* A button **"Add New Song"**
  + When you **click** on it, **if** the album is **successfully created**, you should be redirected to the **index page**.
  + Otherwise, the form should show the **appropriate validation errors**.

Картина, която съдържа екранна снимка, електроника, кръг, текст

Описанието е генерирано автоматично

Картина, която съдържа екранна снимка, текст, електроника, кръг

Описанието е генерирано автоматично

As we can see there is a **related** song to the album.

View:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

Form:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

In the base form, we override the **\_\_init\_\_** method and add **choices** with all album **names**.

Template:

{% extends 'common/base.html' %}  
{% block content %}  
  
*<!--Create Page-->*<section class="createPage">  
 <form method="POST" action="{% url 'create song' %}">  
 <fieldset>  
 <legend>Add Song</legend>  
 <div class="container">  
 {% csrf\_token %}  
 {{ form }}  
 *<!-- Button for Adding Song -->* <button class="add-song" type="submit">Add New Song</button>  
 </div>  
 </fieldset>  
 </form>  
</section>  
  
{% endblock %}