Absolutely! Here’s a step-by-step guide to creating a Django project with React that stores car make data in SQLite and displays it, along with detailed explanations for each step.

**Step 1: Set Up Your Environment**

1. **Install Python:**
   * **Explanation:** Python is the programming language we will use for the backend with Django. Download Python from the official Python website and follow the installation instructions.
2. **Install Django:**

bash

pip install django

* + **Explanation:** Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. We use the pip tool to install Django.

1. **Install Node.js and npm:**
   * **Explanation:** Node.js is a JavaScript runtime used for running JavaScript on the server-side, and npm is the package manager for Node.js. We need these to create and run the React app. Download Node.js and npm from the official website.

**Step 2: Create a Django Project and App**

1. **Start a New Project:**

bash

django-admin startproject car\_project

cd car\_project

* + **Explanation:** django-admin startproject creates a new Django project named car\_project. We then navigate into the project directory using cd.

1. **Create a New App:**

bash

python manage.py startapp cars

* + **Explanation:** startapp creates a new Django app named cars within the project. An app is a web application that does something, e.g., a blog system or a database of car makes.

1. **Add the App to Settings:**
   * **Explanation:** In car\_project/settings.py, add 'cars' to the INSTALLED\_APPS list so Django recognizes the app and includes it in the project.

**Step 3: Define the Model**

1. **Create the Model:** In cars/models.py:

python

from django.db import models

class CarMake(models.Model):

name = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.name

* + **Explanation:** Models in Django are used to define the structure of our database. Here, we create a CarMake model with a single field name to store car make names. The \_\_str\_\_ method defines how the object is represented as a string.

**Step 4: Create and Apply Migrations**

1. **Make Migrations:**

bash

python manage.py makemigrations cars

* + **Explanation:** makemigrations creates migration files that Django uses to create and modify database tables.

1. **Apply Migrations:**

bash

python manage.py migrate

* + **Explanation:** migrate applies the migrations to the database, creating the CarMake table.

**Step 5: Register the Model in Admin**

1. **Register the Model:** In cars/admin.py:

python

from django.contrib import admin

from .models import CarMake

admin.site.register(CarMake)

* + **Explanation:** Registering the model with the Django admin site makes it possible to manage CarMake entries through the admin interface.

**Step 6: Create a Superuser**

1. **Create a Superuser:**

bash

python manage.py createsuperuser

* + **Explanation:** createsuperuser prompts you to create an admin account. This superuser can log into the admin interface to manage the application.

**Step 7: Set Up URL Routes**

1. **Define URLs in car\_project/urls.py:**

python

from django.contrib import admin

from django.urls import path

from cars import views

urlpatterns = [

path('admin/', admin.site.urls),

path('get\_cars/', views.get\_cars, name='get\_cars'),

path('api/get\_cars/', views.get\_car\_makes, name='get\_car\_makes'),

]

* + **Explanation:** URL routing maps URLs to views. We define routes for the admin interface, the get\_cars page, and an API endpoint to fetch car makes in JSON format.

**Step 8: Create Views**

1. **Create get\_cars View:** In cars/views.py:

python

from django.shortcuts import render

from .models import CarMake

from .populate import populate\_car\_makes

def get\_cars(request):

populate\_car\_makes()

car\_makes = CarMake.objects.all()

return render(request, 'cars/car\_list.html', {'car\_makes': car\_makes})

def get\_car\_makes(request):

populate\_car\_makes()

car\_makes = list(CarMake.objects.values())

return JsonResponse(car\_makes, safe=False)

* + **Explanation:** Views handle the logic for what happens when a user visits a specific URL. get\_cars renders a template with car makes, and get\_car\_makes returns car makes as JSON. populate\_car\_makes populates the database if it’s empty.

**Step 9: Create Template**

1. **Create Template:** In cars/templates/cars/car\_list.html:

html

<!DOCTYPE html>

<html>

<head>

<title>Car Makes</title>

</head>

<body>

<h1>Car Makes</h1>

<ul>

{% for car in car\_makes %}

<li>{{ car.name }}</li>

{% endfor %}

</ul>

</body>

</html>

* + **Explanation:** Templates in Django are HTML files that can include dynamic content. Here, we create a template to display a list of car makes.

**Step 10: Create populate.py Script**

1. **Create populate.py:** In cars/populate.py:

python

from .models import CarMake

def populate\_car\_makes():

if CarMake.objects.count() == 0:

default\_car\_makes = ['Toyota', 'Honda', 'Ford', 'BMW', 'Tesla']

for car\_name in default\_car\_makes:

CarMake.objects.create(name=car\_name)

print("Car makes populated successfully!")

else:

print("Car makes already populated.")

* + **Explanation:** This script contains a function to populate the database with default car makes if it is empty.

**Step 11: Create React App**

1. **Create React App:**

bash

npx create-react-app car-frontend

cd car-frontend

* + **Explanation:** create-react-app is a tool that sets up a new React application with a sensible default structure and configuration. We navigate into the React app directory.

**Step 12: Fetch Data in React**

1. **Install Axios:**

bash

npm install axios

* + **Explanation:** Axios is a popular JavaScript library used to make HTTP requests from the browser. We use it to fetch data from the Django backend.

1. **Create CarList Component:** In src/CarList.js:

javascript

import React, { useEffect, useState } from 'react';

import axios from 'axios';

const CarList = () => {

const [carMakes, setCarMakes] = useState([]);

useEffect(() => {

axios.get('http://127.0.0.1:8000/api/get\_cars/')

.then(response => {

setCarMakes(response.data);

})

.catch(error => {

console.error("There was an error fetching the car makes!", error);

});

}, []);

return (

<div>

<h1>Car Makes</h1>

<ul>

{carMakes.map(car => (

<li key={car.id}>{car.name}</li>

))}

</ul>

</div>

);

};

export default CarList;

* + **Explanation:** This React component fetches car makes from the Django backend using Axios and displays them in a list. useEffect is used to make the HTTP request when the component mounts.

**Step 13: Integrate React Component**

1. **Update App Component:** In src/App.js:

javascript

import React from 'react';

import CarList from './CarList';

import './App.css';

function App() {

return (

<div className="App">

<CarList />

</div>

);

}

export default App;

* + **Explanation:** We update the main App component to include the CarList component, which displays the list of car makes.

**Step 14: Run Both Servers**

1. **Run Django Server:**

bash

python manage.py runserver

* + **Explanation:** This command starts the Django development server on http://127.0.0.1:8000.

1. **Run React Development Server:**

bash

npm start

* + **Explanation:** This command starts the React development server on http://localhost:3000.

**Step 15: Access the React App**

1. **Access the App:** Open your browser and go to http://localhost:3000. You should see the list of car makes.
   * **Explanation:** With both servers running, the React app fetches car make data from the Django backend and displays it.