Build CarModel and CarMake Django Models

Estimated time needed: 90 minutes

A dealership typically manages cars from one or more makers or manufacturers, and customers should be allowed to review the cars they purchased from a dealer.

In this lab, you will create the `CarModel` and `CarMake` models in the Django app.

- A `car model` includes basic information such as its make, year, type, and dealer id.
- A `car make` includes basic information such as name and description.

Environment setup

If your lab workspace has not been reset, you can skip this section. If not, to continue from what you have done previously, do the following:

1. `git clone` or pull from your created GitHub repository.
2. Run the following to set up the Django environment:

```
cd /home/project/xrwvm-fullstack_developer_capstone/server
pip install virtualenv
virtualenv djangoenv
source djangoenv/bin/activate
```

3. Install the required packages by running the following command:

```
python3 -m pip install -U -r requirements.txt
```

4. Run the following command to perform model migrations:

```
python3 manage.py makemigrations
python3 manage.py migrate
python3 manage.py runserver
```

Steps to Build CarModel and CarMake models

You will need to create two new models in `server/djangoapp/models.py`:

- A `CarMake` model to save some data about a car's make.
- A `CarModel` model to save some data about a car's model.

1. Create a car make Django model `class CarMake(models.Model):`
    - Name
    - Description
    - Any other fields you would like to include in a car make
    - A `__str__` method to print a car make object

▾ Click here for a sample

```
class CarMake(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()
    # Other fields as needed
    def __str__(self):
        return self.name  # Return the name as the string representation
```

2. Create a car model Django model `class CarModel(models.Model):`
    - Many-to-one relationship to `CarMake` model (One car make can have many car models, using a ForeignKey field)
    - Dealer Id (IntegerField) refers to a dealer created in Cloudant database
    - Name
    - Type (CharField with a choices argument to provide limited choices such as Sedan, SUV, and Wagon)
    - Year (DateField)
    - Any other fields you would like to include in a car model
    - A `__str__` method to print the car make and car model object

▾ Click here for a sample

```
class CarModel(models.Model):
    car_make = models.ForeignKey(CarMake, on_delete=models.CASCADE)  # Many-to-One relationship
    name = models.CharField(max_length=100)
    CAR_TYPES = [
        ('SEDAN', 'Sedan'),
        ('SUV', 'SUV'),
        ('WAGON', 'Wagon'),
        # Add more choices as required
    ]
    type = models.CharField(max_length=10, choices=CAR_TYPES, default='SUV')
    year = models.IntegerField(default=2023,
        validators=[
            MaxValueValidator(2023),
            MinValueValidator(2015)
        ])
    # Other fields as needed
    def __str__(self):
        return self.name  # Return the name as the string representation
```

3. You need to register the `CarMake` and `CarModel` on the admin site so you can conveniently manage their content (i.e., perform CRUD operations). Refer to the previous Django Admin Site lab for more details.

▾ Click here for sample code

```
from django.contrib import admin
from .models import CarMake, CarModel
# Registering models with their respective admins
admin.site.register(CarMake)
admin.site.register(CarModel)
```

4. Run migrations for the models.

```
python3 manage.py makemigrations
python3 manage.py migrate --run-syncdb
```

Note: The `--run-syncdb` allows creating tables for apps without migrations.

Refer to the previous Django ORM lab for more details:

CRUD on Django Model Objects

Steps to register the CarMake and CarModel models with the admin site

1. First, you must have superuser access on the admin site (if not created before).
2. Please use `root` as the user name and `root` as the password for your reviewer to log in to your app.
3. Open a new terminal and build your client by running the following commands:

```
cd /home/project/xrwvm-fullstack_developer_capstone/server/frontend
npm install
npm run build
```

4. Start the server from the terminal where django application was running if it is not already running.

```
python3 manage.py runserver
```

5. Click the button below to launch the admin page to login with the root credentials.

Django Admin

▾ Click here if you get an error
If you get an error as below, it is caused because the URL has changed. Copy the new application URL and make necessary changes in `server/djangoproj/settings.py`.

Forbidden (403)

CSRF verification failed. Request aborted.

Help

Reason given for failure:

```
Origin checking failed - https://ksundararaja-8000.theiadockernext-1-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai does not match any trusted origins.
```

In general, this can occur when there is a genuine Cross Site Request Forgery, or when Django's CSRF mechanism has not been used correctly. For POST forms, you need to ensure:

- Your browser is accepting cookies.
- The view function passes a `request` to the template's `render` method.
- In the template, there is a `{% csrf_token %}` template tag inside each POST form that targets an internal URL.
- If you are not using `CsrfViewMiddleware`, then you must use `csrf_protect` on any views that use the `csrf_token` template tag, as well as those that accept the POST data.
- The form has a valid CSRF token. After logging in in another browser tab or hitting the back button after a login, you may need to reload the page with the form, because the token is rotated after a login.

You're seeing the help section of this page because you have `DEBUG = True` in your Django settings file. Change that to `False`, and only the initial error message will be displayed.

You can customize this page using the CSRF_FAILURE_VIEW setting.

Take a screenshot for peer review:

1. After you log in to the admin site, please take a screenshot and name it as `admin_login.jpg` or `admin_login.png` for peer review.
2. In addition, you may want to log out as the admin user. You will be redirected to the admin login page again. Take a screenshot and name it `admin_logout.jpg` or `admin_logout.png`.
3. Open `djangoapp/views.py`, import CarMake and CarModel in it after the other import statements in the beginning of the file, and add a method to get the list of cars by including the following code:

```
from .models import CarMake, CarModel
```

```
def get_cars(request):
    count = CarMake.objects.filter().count()
    print(count)
    if(count == 0):
        initiate()
    car_models = CarModel.objects.select_related('car_make')
    cars = []
    for car_model in car_models:
        cars.append({"CarModel": car_model.name, "CarMake": car_model.car_make.name})
    return JsonResponse({"CarModels":cars})
```

4. Open `server/djangoapp/urls.py` and add the path for `get_cars` in it.

```
path(route='get_cars', view=views.get_cars, name ='getcars'),
```

5. Open `server/djangoapp/populate.py` and paste the following code in it to populate data in your database. The data is populated when the first call is made to `get_cars`, if the CarModel is empty. If you wish to add data manually, skip this step.

```
from .models import CarMake, CarModel
def initiate():
    car_make_data = [
        {"name":"NISSAN", "description":"Great cars. Japanese technology"},
        {"name":"Mercedes", "description":"Great cars. German technology"},
        {"name":"Audi", "description":"Great cars. German technology"},
        {"name":"Kia", "description":"Great cars. Korean technology"},
        {"name":"Toyota", "description":"Great cars. Japanese technology"},
    ]
    car_make_instances = []
    for data in car_make_data:
            car_make_instances.append(CarMake.objects.create(name=data['name'], description=data['description']))
    # Create CarModel instances with the corresponding CarMake instances
    car_model_data = [
      {"name":"Pathfinder", "type":"SUV", "year": 2023, "car_make":car_make_instances[0]},
      {"name":"Qashqai", "type":"SUV", "year": 2023, "car_make":car_make_instances[0]},
      {"name":"XTRAIL", "type":"SUV", "year": 2023, "car_make":car_make_instances[0]},
      {"name":"A-Class", "type":"SUV", "year": 2023, "car_make":car_make_instances[1]},
      {"name":"C-Class", "type":"SUV", "year": 2023, "car_make":car_make_instances[1]},
      {"name":"E-Class", "type":"SUV", "year": 2023, "car_make":car_make_instances[1]},
      {"name":"A4", "type":"SUV", "year": 2023, "car_make":car_make_instances[2]},
      {"name":"A5", "type":"SUV", "year": 2023, "car_make":car_make_instances[2]},
      {"name":"A6", "type":"SUV", "year": 2023, "car_make":car_make_instances[2]},
      {"name":"Sorrento", "type":"SUV", "year": 2023, "car_make":car_make_instances[3]},
      {"name":"Carnival", "type":"SUV", "year": 2023, "car_make":car_make_instances[3]},
      {"name":"Cerato", "type":"Sedan", "year": 2023, "car_make":car_make_instances[3]},
      {"name":"Corolla", "type":"Sedan", "year": 2023, "car_make":car_make_instances[4]},
      {"name":"Camry", "type":"Sedan", "year": 2023, "car_make":car_make_instances[4]},
      {"name":"Kluger", "type":"SUV", "year": 2023, "car_make":car_make_instances[4]},
        # Add more CarModel instances as needed
    ]
    for data in car_model_data:
            CarModel.objects.create(name=data['name'], car_make=data['car_make'], type=data['type'], year=data['year'])
```

6. Alternatively, if you wish to add the Car makes and models manually, you can go to the admin page and add them as per your wish. Please note that you will only be allowed to choose from one of the makes and models in these tables to post a review in the later part of the project.

Django Admin

7. Click the `Get Cars` button below to check the list of cars added.

   If you chose to populate, it will show 5 CarMakes and 15 Car Models, 3 under each make. Else it will show what you added.

Get Cars

Take a screenshot for peer review:

8. Please take a screenshot of the car and model and name it `cars.jpg` or `cars.png` for peer review.

9. Click on the `Django Admin` button below and take a screenshot of the car models and save it as `car models.png` or `car models.jpg`.

Django Admin

Commit your updated project to GitHub

Commit all updates to the GitHub repository to can save your work.

If you need to refresh your memory on how to commit and push your code to GitHub in the Theia lab environment, please refer to this lab Working with git in the lab environment

Summary

In this lab, you have created the `CarModel` and `CarMake` models in your Django app.

Author(s)

Lavanya T S

Other Contributor(s)

Upkar Lidder
Yan Luo

Priya