SDEV 2401

Rapid Backend Development

Introduction to Backend Web Development

A LEADING POLYTECHNIC COMMITTED TO YOUR SUCCESS

Agenda

On the right is what we will cover today.

- Review
- How does backend web development work?
- What is the MVC Pattern?
- Let’s take a look at this from a Django Perspective
- Django Architecture and Apps
- Django Views (Controller in MVC)
- Django URL Routing (Needed for Controller in MVC)
- Django Templates (View in MVC)
- Django Models (Model in MVC)
- Django ORM Overview
- Example

Expectations

- No Late Assignments

- No Cheating

- Be a good classmate

- Don’t waste your time

- Show up to class

Review

- In the last lesson we covered:
  - What is backend web development
  - Backend development frameworks options
  - Why use Django for backend development
  - Setting up your environment for Django development
  - Starting your Django project
  - Running the development server
  - Key project files explained

How does backend web development work?

- In order to understand how backend web development works, we need to understand the architecture pattern that is used in backend web development.
- The most common architecture pattern used in backend web development is the Model-View-Controller (MVC) pattern.

This will explain how we take a request from a user and return a response that has fetched data from a database and rendered it in a template.

This is a really good design pattern to learn because it’s used in many different frameworks, languages and other areas of software development.

What is the MVC Pattern?

Model-View-Controller (MVC) is a software design pattern commonly used for developing user interfaces that divides an application into three interconnected components.

- Model: The model represents the data and the business logic of the application. It is responsible for managing the data, including retrieving it from a database, processing it, and updating it as needed.
- View: The view is responsible for rendering the user interface of the application. It displays the data from the model to the user and handles user input.
- Controller: The controller acts as an intermediary between the model and the view. It receives user input from the view, processes it, and updates the model accordingly. It also retrieves data from the model and passes it to the view for rendering.

The MVC pattern is widely used in web development frameworks such as Ruby on Rails, Django, and ASP.NET MVC, among others. It helps to separate concerns and improve the maintainability and scalability of applications.

Let’s take a look at this from a Django Perspective

In Django, the MVC pattern is implemented as Model-Template-View (MTV).

- Model: The model represents the data and the business logic of the application. It is responsible for managing the data, including retrieving it from a database, processing it, and updating it as needed.
- Template: The template is responsible for rendering the user interface of the application. It displays the data from the model to the user and handles user input.
- View: The view acts as an intermediary between the model and the template. It receives user input from the template, processes it, and updates the model accordingly. It also retrieves data from the model and passes it to the template for rendering.

In the next slides we’re going to focus on all of these pieces and talk about them.

Django Architecture and Apps

- In Django for each project we have multiple "apps".
  - Each app is a self contained module that has a specific purpose.
  - Each app has its own models, views, templates, and urls (more on this in the coming slides).
  - This allows for better organization and separation of concerns.
- To create an app in Django, we use the `startapp` command For example, to create a blog app, we would run the following command:

```
python manage.py startapp blog
```

We can then add this app to our project by adding it to the `INSTALLED_APPS` list in the `settings.py` file, so django knows to include it in the project.

Django Views (Controller in MVC)

- Django views are Python functions or classes that handle HTTP requests and return HTTP responses.

Here's a simple example of a Django view function:

```
from django.http import HttpResponse

def hello_world(request):
    return HttpResponse("Hello, world!")
```

- In this example, the `hello_world` function takes an HTTP request as input and returns an HTTP response with the text "Hello, world!". We'll look at how we can hook this up to a url in the next slide.

Django URL Routing (Needed for Controller in MVC)

- Django URL routing is the process of mapping URLs to views in a Django application.

Here’s an example of how to configure URL routing in Django:

```
from django.urls import path

from .views import hello_world

urlpatterns = [
    path('hello/', hello_world, name='hello_world'),
]
```

- Note: We'll be adding this to the main `urls.py` file in the project folder or in the app folder.

Django Templates (View in MVC)

- Django templates are HTML files that contain placeholders for dynamic content.

Here’s an example of a Django template (which uses something called the Django Template Language):

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello, {{ name }}!</title>
</head>
<body>
    <h1>Hello, {{ name }}!</h1>
</body>
</html>
```

- In this example, the \{\{ name \}\} placeholder will be replaced with a dynamic value when the template is rendered.

We’re going to take a really deep dive into templates in the first module of the course.

Django Models (Model in MVC)

- Django models are Python classes that define the structure of the data in a Django application.

```python
from django.db import models

class Recipe(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    ingredients = models.TextField()
    instructions = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

- Models are mapped to database tables using Django's Object-Relational Mapping (ORM) system. On the left is an example of a Django model for a recipe app.
- In this example, the `Recipe` model defines a database table with fields for the recipe's title, description, ingredients, instructions, and creation date.
- To apply this model to the database, we would run migrations using Django's migration framework, with `python manage.py makemigrations` and `python manage.py migrate`.

Django ORM Overview - Creating Data

- The Django ORM (Object-Relational Mapping) is a powerful tool that allows developers to interact with the database using Python code instead of writing raw SQL queries. (we'll be going over this in depth in a later module of the course.)
- The ORM provides a high-level abstraction over the database, allowing developers to create, read, update, and delete records in the database using Python objects and methods.
  - Here's a small example of how to use the Django ORM to create a new recipe:

```
from myapp.models import Recipe

new_recipe = Recipe(
    title="Spaghetti Bolognese",
    description="A classic Italian pasta dish.",
    ingredients="Spaghetti, ground beef, tomato sauce, onions, garlic, herbs",
    instructions="1. Cook spaghetti. 2. Prepare sauce. 3. Combine and serve."
)
new_recipe.save()
```

Django ORM Overview - Querying Data

Here's an example of how to use the Django ORM to query for recipes:

```
from myapp.models import Recipe

recipes = Recipe.objects.all()

for recipe in recipes:
    print(recipe.title)
```

For this course we'll be using this to interact with our database instead of writing raw SQL queries.

Note: We'll still be able to write SQL queries if we need to (for complex queries for a report for example), but for the most part we'll be using the ORM.

Example

Let’s work through an example together.