CAREERFOUNDRY

Python for Web Developers Learning Journal

Exercise 2.1: Getting Started with Django

Learning Goals

- Explain MVT architecture and compare it with MVC
- Summarize Django's benefits and drawbacks
- Install and get started with Django

Reflection Questions

1. Suppose you're a web developer in a company and need to decide if you'll use vanilla (plain) Python for a project, or a framework like Django instead. What are the advantages and drawbacks of each?
   Vanilla Python: Using plain Python means writing code from scratch without using any pre-built frameworks. It gives you full control over your code and can be simpler for small projects. However, it can be more time-consuming and complex for larger projects

because you have to handle everything yourself, like routing, database management, and security

Django (Framework): Django is a web framework for Python that provides a lot of pre-built tools and features for web development. It can speed up development time significantly by providing ready-to-use components for things like database management, authentication, and URL routing. However, it may have a steeper learning curve for beginners, and you might have to adapt to its conventions and structure.

2. In your own words, what is the most significant advantage of Model View Template (MVT) architecture over Model View Controller (MVC) architecture?
   MVT vs. MVC: The biggets advantage of the Model-View-Template (MVT) architecture over Model-View-Controller (MVC) is its simplicity. In MVT, the template layer handles the user interface and presentation logic, which makes it easier to design and manage the frontend. In contrast, MVC requires a more complex setup with separate components for the model, view, and controller, which can be more challenging to understand and maintain, especially for beginners

3. Now that you've had an introduction to the Django framework, write down three goals you have for yourself and your learning process during this Achievement. You can reflect on the following questions if it helps:

   - What do you want to learn about Django?
     I'm excited to see how its framework is formatted and what features it provides

   - What do you want to get out of this Achievement?
     Again, I want to continue to learn about something new.

   - Where or what do you see yourself working on after you complete this Achievement?
     I will likely touchup some of my portfolio then pick up a few YouTube tutorials on Javascript, React, and Python so that I can get better at them.

Exercise 2.2: Django Project Set Up

Learning Goals

- Describe the basic structure of a Django project
- Summarize the difference between projects and apps
- Create a Django project and run it locally
- Create a superuser for a Django web application

Reflection Questions

1. Suppose you're in an interview. The interviewer gives you their company's website as an example, asking you to convert the website and its different parts into Django terms. How would you proceed? For this question, you can think about your dream company and look at their website for reference.
   (*Hint: In the Exercise, you saw the example of the CareerFoundry website in the Project and Apps section.*)

   Analyze the Company's Website:

   Understand the structure of the website, including its pages, navigation, and content.
   Identify the main components, such as the homepage, about page, products/services pages, contact page, etc.

   Map Website Components to Django Terms:

   URLs: Each page of the website corresponds to a URL. In Django, URLs are defined in the urls.py file using the path() function.

   Views: Views in Django are Python functions or classes that handle requests and return responses. Each page of the website would have a corresponding view function or class.

   Templates: Templates in Django are HTML files that define the structure of a page. Each page of the website would have a corresponding template.

   Models: Models in Django define the structure of the database. If the website has dynamic content (e.g., blog posts, products), each type of content would be represented by a model.

   Static Files: Static files such as CSS, JavaScript, and images are served by Django's staticfiles app and stored in the static directory of the project.

   Media Files: Media files such as user-uploaded images are stored in the media directory of the project and served using Django's MEDIA_URL setting.

   Admin Interface: Django provides an admin interface for managing site content. Models registered with the admin interface can be managed through the admin site.

   Create Django Project and App:

   Create a new Django project using django-admin startproject.
   Create a new Django app using python manage.py startapp.
   Define URL patterns, views, models, templates, static files, and media files as needed based on the analysis of the company's website.
   Implement Django Functionality:

Define URL patterns in the urls.py file of the project and app to map URLs to views.
Write view functions or classes to handle requests and return responses.
Create templates for each page of the website.
Define models for dynamic content if applicable.
Configure static and media files settings in the project's settings file.

2. In your own words, describe the steps you would take to deploy a basic Django application locally on your system.

   Create a new Django project using django-admin startproject.
   Create a new Django app using python manage.py startapp.
   Create superuser admin
   Define URL patterns, views, models, templates, static files, and media files as needed
   Implement Django Functionality:

   Define URL patterns in the urls.py file of the project and app to map URLs to views.
   Create templates for each page of the website.

3. Do some research about the Django admin site and write down how you'd use it during your web application development.

   In web app development, I can use Django admin as an internal tool for managing the application's data and users. It's particularly useful for applications where content management is a key feature, such as blogs, content management systems (CMS), or internal tools for managing data. By using Django admin, I can quickly build a functional admin interface without having to write a lot of custom code, saving time and effort

Exercise 2.3: Django Models

Learning Goals

- Discuss Django models, the "M" part of Django's MVT architecture
- Create apps and models representing different parts of your web application
- Write and run automated tests

Reflection Questions

1. Do some research on Django models. In your own words, write down how Django models work and what their benefits are.

Django models are like blueprints for organizing and managing your data in a web application. They are written in Python and help you define how your data should be structured and how it should behave. Think of them as templates that describe the different types of information your application will work with, such as user profiles, blog posts, or products in an online store.

Benefits of Django Models:

1. **Abstraction**: Django models make it easier to work with databases by abstracting away the complex details of database interactions. This means you can focus on writing Python code to manage your data, rather than worrying about the underlying database queries.
2. **Code Reusability**: Once you've defined a model, you can reuse it throughout your application. This saves time and effort by allowing you to define data structures once and use them in multiple places.
3. **ORM (Object-Relational Mapping)**: Django's ORM translates Python code into database queries, so you can work with your data using Python objects. This makes it easier to write and maintain code, as you can use familiar Python syntax.
4. **Database Independence**: Django models provide a level of abstraction that allows you to switch between different database backends (like SQLite, MySQL, or PostgreSQL) without changing your code. This flexibility is useful if you need to change your database setup in the future.
5. **Validation and Constraints**: Django models include built-in validation and constraints to ensure that the data stored in your database meets certain criteria. This helps maintain data integrity and prevents errors in your application

2. In your own words, explain why it is crucial to write test cases from the beginning of a project. You can take an example project to explain your answer.

Writing test cases from the start of a project is like building a safety net for your code. It's really important for a few key reasons:

Bug Detection: Tests help catch mistakes early on, which makes them easier and cheaper to fix.
Code Quality: When you write tests, you're encouraged to write your code in a way that's easier to test. This usually leads to cleaner, more reliable code.
Regression Testing: Tests make sure that when you add new features, you don't accidentally break existing ones. They help keep your app working as it should.
Documentation: Tests act like a user manual for your code, showing other developers how it's meant to be used. This makes it easier for them to understand and work with your code.
Refactoring Safety: When you need to change your code, tests can help you make sure you haven't broken anything in the process.

Confidence in Changes: With a good set of tests, you can make changes to your code with confidence, knowing that if something goes wrong, your tests will catch it.
Client Confidence: Having tests in place shows your clients that you take the quality of your work seriously, which can give them more confidence in your product.

Example: Imagine you're building an online store. If you write tests from the beginning, you'd test things like adding items to the cart, updating quantities, and checking out. This ensures that these critical parts of your store work correctly and continue to work as you add new features

Exercise 2.4: Django Views and Templates

Learning Goals

- Summarize the process of creating views, templates, and URLs
- Explain how the “V” and “T” parts of MVT architecture work
- Create a frontend page for your web application

Reflection Questions

1. Do some research on Django views. In your own words, use an example to explain how Django views work.
   Views are like the traffic cops of your web application. When a user requests a specific URL (like clicking a link or submitting a form), Django's views decide what to do next. They fetch data from the database, process it, and then pass it to a template to create the final HTML page that the user sees. For example, if you have a recipe website, a view might handle the request for a specific recipe page by fetching that recipe's information from the database and passing it to the template to display

2. Imagine you’re working on a Django web development project, and you anticipate that you’ll have to reuse lots of code in various parts of the project. In this scenario, will you use Django function-based views or class-based views, and why?
   If you anticipate reusing lots of code in various parts of your project, class-based views might be more suitable. Class-based views allow you to organize your code more effectively by using inheritance. You can create a base view class with common functionality and then create subclasses that inherit from the base class and add specific behavior as needed. This can lead to more maintainable and reusable code compared to function-based views, where you might end up duplicating code in different views

3. Read Django's documentation on the Django template language and make some notes on its basics.
   Django's Template plate language is a way to dynamically generate HTML pages. Some basics include:

- **Variables**: You can insert variables from your views into your templates using **{{ variable_name }}**.
- **Tags**: Tags provide logic and control flow in templates. For example, **{% if condition %}** allows you to conditionally display content.
- **Filters**: Filters let you modify variables in your templates. For example, **{{ variable|upper }}** will display the variable in uppercase.
- **Template inheritance**: You can create a base template with common elements like headers and footers, and then extend it in other templates to reuse these elements

Exercise 2.5: Django MVT Revisited

Learning Goals

- Add images to the model and display them on the frontend of your application
- Create complex views with access to the model
- Display records with views and templates

Reflection Questions

1. In your own words, explain Django static files and how Django handles them.

   Django static files are files like images, CSS stylesheets, and JavaScript files that are used to style and enhance the appearance and functionality of a Django web application. Django provides a way to manage and serve these static files, making it easy for developers to include them in their projects. Django handles static files by allowing developers to define a **STATIC_URL** in their settings, which specifies the URL where static files should be served from. Developers can then use the **{% static %}** template tag in their HTML templates to reference static files, and Django will automatically serve these files when the template is rendered

2. Look up the following two Django packages on Django's official documentation and/or other trusted sources. Write a brief description of each.

ListView is a Django class-based view that is used to display a list of objects from a queryset. It is commonly used to display a list of items, such as blog posts, products, or any other type of data that can be represented as a list. **ListView** takes care of fetching the queryset, paginating the results if necessary, and rendering the list of objects in a template. Developers can customize the behavior of **ListView** by subclassing it and overriding its methods

DetailView is another Django class-based view that is used to display detailed information about a single object from a queryset. It is typically used to display the details of a single item, such as a blog post, product, or user profile. **DetailView** automatically fetches the object from the queryset based on the URL parameter, renders the object in a template, and provides a context containing the object for the template to use. Similar to **ListView**, developers can customize the behavior of **DetailView** by subclassing it and overriding its methods

3. You're now more than halfway through Achievement 2! Take a moment to reflect on your learning in the course so far. How is it going? What's something you're proud of so far? Is there something you're struggling with? What do you need more practice with? You can use these notes to guide your next mentor call.

   Life has been very busy and as is nearing its end it has kind of become difficult to stay focused. I am happy that I have made it this far and have learned so much.

Exercise 2.6: User Authentication in Django

Learning Goals

- Create authentication for your web application
- Use GET and POST methods
- Password protect your web application's views

Reflection Questions

1. In your own words, write down the importance of incorporating authentication into an application. You can take an example application to explain your answer.
   Incorporating authentication into an application is important for ensuring that only authorized users can access certain parts or features of the application. For example, in a recipe app, authentication ensures that only registered users can add, edit, or delete recipes. This helps protect user data and prevents unauthorized access, providing a secure environment for users to interact with the app.

2. In your own words, explain the steps you should take to create a login for your Django web application.

   To create a login for a Django web application, you would follow these steps:

   - Define a URL pattern for the login page in your app's urls.py.
   - Create a login form using Django's built-in AuthenticationForm.
   - Create a view that handles the login logic, using Django's login function to authenticate the user.
   - Create a template for the login page, including the login form and any other desired content.
   - Update your app's urls.py to include the login URL pattern.
   - Add a link to the login page in your app's templates, typically in the navigation bar.

3. Look up the following three Django functions on Django's official documentation and/or other trusted sources and write a brief description of each.

Function	Description
authenticate()	This Django function is used to authenticate a user based on the username and password. It takes these credentials as arguments and returns the authenticated user object if successful, or None if authentication fails. For example, in a login view, you would use authenticate() to verify the user's credentials.
redirect()	This function is used to redirect the user to a different URL. It takes the URL as an argument and sends an HTTP redirect response to the client's browser, instructing it to navigate to the specified URL. For example, after a successful login, you might use redirect() to send the user to their profile page.
include()	This function is used in Django's URL configuration to include other URLconfs. It allows you to organize your URL patterns into separate files and include them in the

	main URLconf. For example, you might use include() to include a separate URLconf for handling authentication-related URLs, such as login and logout.

Exercise 2.7: Data Analysis and Visualization in Django

Learning Goals

- Work on elements of two-way communication like creating forms and buttons
- Implement search and visualization (reports/charts) features
- Use QuerySet API, DataFrames (with pandas), and plotting libraries (with matplotlib)

Reflection Questions

1. Consider your favorite website/application (you can also take CareerFoundry). Think about the various data that your favorite website/application collects. Write down how analyzing the collected data could help the website/application.

   Analyzing the collected data could help the website/application by providing valuable insights into user behavior, preferences, and trends. This information can be used to improve the user experience, tailor content and offerings to better suit user needs, and make data-driven decisions to enhance overall performance and effectiveness.

2. Read the Django official documentation on QuerySet API. Note down the different ways in which you can evaluate a QuerySet.

   - Using list() to force evaluation and return a list of objects.
   - Using count() to return the number of objects in the QuerySet.
   - Using exists() to check if any objects exist in the QuerySet.
   - Using aggregate() to perform aggregate functions like sum, average, min, max, etc.
   - Using values() to return dictionaries instead of model instances.
   - Using values_list() to return tuples of field values instead of model instances.

3. In the Exercise, you converted your QuerySet to DataFrame. Now do some research on the advantages and disadvantages of QuerySet and DataFrame, and explain the ways in which DataFrame is better for data processing.

QuerySet and DataFrame both have their advantages and disadvantages. QuerySet is more closely tied to the database and is efficient for database operations. It allows for lazy evaluation, which can improve performance by only fetching data when needed. However, QuerySet is limited in terms of data manipulation and analysis capabilities compared to DataFrame.

DataFrame, on the other hand, is a powerful data structure from the pandas library that offers extensive data processing and analysis capabilities. It provides functions for data manipulation, filtering, grouping, and statistical operations. DataFrame is particularly useful for handling large datasets and complex data transformations. However, DataFrame is less efficient for database operations compared to QuerySet and may require more memory and processing power.

In summary, DataFrame is better suited for data processing and analysis tasks where extensive data manipulation and analysis are required. It provides a wide range of functions and capabilities that are not available in QuerySet, making it a valuable tool for data-centric applications

Exercise 2.8: Deploying a Django Project

Learning Goals

- Enhance user experience and look and feel of your web application using CSS and JS
- Deploy your Django web application on a web server
- Curate project deliverables for your portfolio

Reflection Questions

1. Explain how you can use CSS and JavaScript in your Django web application.
   Using static files, then link them in your templates

2. In your own words, explain the steps you'd need to take to deploy your Django web application.

Requirements File: Create a requirements.txt file that lists all of your Python dependencies, which you can generate using pip freeze > requirements.txt.

Procfile: Create a Procfile in the root of your project to tell Heroku how to run your application:c
Copy code web: gunicorn your_project_name.wsgi --log-file -
wsgi.py: Make sure wsgi.py is correctly set up for WSGI servers.
Settings: Configure your settings.py to get database credentials from environment variables, use dj-database-url to configure the DATABASES setting, and set DEBUG to False.
Static and Media Files: Set up WhiteNoise for serving static files and configure your media file storage (e.g., using Cloudinary or Amazon S3 for Heroku).
Git Repository: Initialize a Git repository if you haven't already (git init), commit your changes (git add . and git commit -m "message").
Heroku CLI: Install the Heroku CLI and log in to your Heroku account.
Create Heroku App: Run heroku create to create a new Heroku application.
Add Heroku Remote: Ensure that your Git repository has a remote pointing to Heroku (heroku git:remote -a your-heroku-app-name).
Environment Variables: Set environment variables on Heroku for secret keys and other sensitive settings (heroku config:set SECRET_KEY='your_secret_key').
Deploy: Push your code to Heroku (git push heroku main).
Migrate Database: Run database migrations on Heroku (heroku run python manage.py migrate).
Create Superuser: Create a Django admin superuser on Heroku (heroku run python manage.py createsuperuser).
Final Checks: Visit your Heroku app's URL to ensure it's running correctly.

3. (Optional) Connect with a few Django web developers through LinkedIn or any other network. Ask them for their tips on creating a portfolio to showcase Python programming and Django skills. Think about which tips could help you improve your portfolio.
   Done

4. You've now finished Achievement 2 and, with it, the whole course! Take a moment to reflect on your learning:
   a. What went well during this Achievement?
      I finished it. lol
   b. What's something you're proud of?
      Same answer as above. I am very happy to be done.
   c. What was the most challenging aspect of this Achievement?
      Honestly I don't like how they used the bookstore as an example, sometimes it got confusing.
   d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills?
      Yes, I am more confident but I never really have expectations when it comes to these things. I keep an open mind.

Well done—you've now completed the Learning Journal for the whole course.