Pre-Work: Before You Start Achievement 2

In the final part of the learning journal for Achievement 1, you were asked if there's anything—on reflection—that you'd keep in mind and do similarly or differently during Achievement 2. Think about these questions again:

- Was your study routine effective during Achievement 1? If not, what will you do differently during Achievement 2?
  Yes, my new study routine was actually a lot more effective than for my full stack immersion course! It helped me get through the achievement a lot faster.
- Reflect on your learning and project work for Achievement 1. What were you most proud of? How will you repeat or build on this in Achievement 2?
  I am just proud of the fact that I was able to learn Python and its syntax and how quickly I learned and got familiar with it.
- What difficulties did you encounter in the last Achievement? How did you deal with them? How could this experience prepare you for difficulties in Achievement 2?
  Some of the difficulties I encountered were due to the content given being outdated / old and the frameworks are being updated. I solved these by searching up the solutions from Google to stack overflow, and got the solutions that I needed.

Note down your answers and discuss them with your mentor in a call if you like.

Remember that can always refer to Exercise 1.4 of the Orientation course if you're not sure whom to reach out to for help and support.

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?

The advantages of using Python is that it has a simple easy to read syntax. Django itself helps a lot with quick prototyping, fast deployment, scalability, and more. With Django developers don't have to worry too much about the database setup and such, they can focus on more important feature implementation. Django and Python also have a large supportive community backing them if you ever run into an issue. The drawbacks however, is that Django has its own way of doing things that you must follow, also known as "The Django Way". Which decreases the amount of control a developer has over parts of their project, if you want more control over fine details and such, Django may not be the best option.

2. In your own words, what is the most significant advantage of Model View Template (MVT) architecture over Model View Controller (MVC) architecture?
   The advantage of MVT architecture over MVC is that with MVT you don't have to write the code to fetch the data from the database and then map it to a URL. That is all taken care of with the MVT components. Unlike with MVC, where you would have to take the time to code all of that.

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?
   - What do you want to get out of this Achievement?
   - Where or what do you see yourself working on after you complete this Achievement?

I want to learn how Django works, how to set up a project with it, and what other functionalities I can create with it. With this achievement I want to have a fully working Django project that I can showcase in my portfolio. I want the knowledge that I've gained from this achievement to be enough for me to be able to work on other little projects made with Django.

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.*)

For a movie/shoe website like HBO Max the whole website is a project. Inside that project it will have individual modules that perform different tasks. Such as login to log into the website and get authenticated. Then you can see another module such as a profile to see your information. On the homepage for logged out users there is a Plan module, showing users the different plans available for their service.

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

   To deploy a basic Django application locally on my system I will first create and activate a virtual environment and install Django. Then I will create the django project with django-admin.exe startproject projectname. Then I will rename the project directory folder (in this case project-name) to src so there's no confusion with duplicate names. Next I will create the database by running the first migration with py manage.py migrate and then run the server to check for the success message in the browser. After that I need to create a superuser which will have access to everything such as all CRUD operations and the ability to manage other users. To do that I would use the command py manage.py createsuperuser and then follow the prompts. Afterwards, to confirm the creation I would run the server again and add /admin to the end of the url to log into the new super user that I just created. Once I'm finished I will use CTRL + C to stop running my server and deactivate to quit my virtual environment.

3. Do some research about the Django admin site and write down how you'd use it during your web application development.
   The Django admin site is a powerful, built-in tool that Django provides to facilitate database management tasks for your web application. It offers an out-of-the-box web interface to manage the content of your database models, without having to build CRUD (Create, Read, Update, Delete) interfaces from scratch.

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 python objects that Django web apps use to access and manage data from the database. They define the structure of the data stores in the database. For example, for the recipe app I defined a users and recipes model. Those will be two sections of the entire application that will contain a lot of data with different field types and such.

   The benefits of Django models are that Django does a lot of the work for you. All you do is define the model/its structure and select the database you want to use with it and djando will manage the database communication for you! No worrying about pushing SQL queries to the database.

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.**
   It is important to write test cases from the beginning of the project because tests are preventative. You'll end up maybe uncovering problems that could've made a big issue later on in your project's development. The code also ends up being more reliable the more it is tested on, especially early on. This also saves the code from accidentally being broken by anyone else. In the long run testing from the beginning will save a lot of time.

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.
   A view in Django is the logic that Django runs when a user accesses a URL. Each view is represented as a Python function, or as a method of a Python class that accepts a request, runs Python code, and returns a response. The function can be quite basic, simply accepting a request and returning static information—such as an HTML web page, or JSON data—or the function could be as complex as a class that accepts a request, interacts with the database,

accepts user input (sent as requests), and determines what will be returned to the user as a response.

Every view is associated with a template (like an HTML file) which is what the user ends up seeing. Django selects the view based on the URL the user has typed into their browser, for example to go to the Django admin panel for your project, after the http://127.0.0.1:8000 you would put /admin and that would prompt Django to take you to the corresponding view called admin. That one is provided by default by Django though. Going from a URL to a view and template, Django uses configurations that map URL patterns to views, which is what the developer defines/structures.

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 I have to reuse a lot of code in different parts of the project then I will choose class based views. They are harder to read and write but, they have an advantage in that they allow easy reuse of views/code. Unlike FBV which are harder to use when the code needs to be reused or extended.

3. Read Django's documentation on the Django template language and make some notes on its basics.
   Django's template language is a markup language used for creating dynamic web content within Django web applications. It uses a simple syntax with double curly braces for variables and template tags enclosed in {% %}. Variables allow you to display dynamic data, while template tags control flow and logic within templates. You can also apply filters to modify variable outputs. Template inheritance enables the creation of reusable templates, and for loops and conditional statements make it easy to iterate over data and display content conditionally. Overall, Django's template language provides a powerful and flexible way to generate dynamic web pages while maintaining a clear separation between presentation and application logic

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 has a built-in system to handle static files. When developing locally, Django's development server can serve these files directly. Developer images are saved at the app/model level. But for production, Django provides tools to gather all static files into a single location, making it easier to serve them efficiently using web servers or services specialized for this purpose. User uploaded images aka media are saved at the application/project level. In short, Django's static files system ensures that essential, unchanging files are efficiently managed and delivered to users.

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

Package	Description
ListView	**ListView is a generic view in Django designed to display a list of objects from the database. Instead of writing repetitive code to fetch a list of objects and pass them to a template, developers can use ListView to streamline this process. By default, it fetches all objects of a model and displays them in a list. However, developers can customize which objects to display and how they appear by adjusting attributes and methods of the ListView.**
DetailView	**DetailView is another generic view provided by Django. Its primary purpose is to display detailed information about a single database object. For instance, if you have a recipe application, while the ListView might show a list of all recipes, the DetailView would be used to show the full content of a recipe like its ingredients, description, etc, when a user clicks on it. It expects a primary key in the URL (like index number of that post) to determine which specific object to display and can be customized to adjust its behavior and appearance.**

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.

This specific task was a hefty one and the html syntax used took me by surprise. It's a bit confusing but the more I practice with it, the more I'll get used to it. I'm really enjoying how simple Django is and how often its functions and such provide a way for

Django to do the work for me while all I provide is the path or structure. I'm actually liking Python and Django a lot more than I thought I would, it's something I want to delve more into after the course. The exercise steps are also neatly written and easy to follow for both my books and recipe apps.

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 vital for verifying the identity of users, ensuring data security, protecting user privacy, and granting appropriate permissions. It helps differentiate between legitimate users and potential threats, thereby maintaining the application's integrity and safeguarding user information. Without authentication, sensitive data and functionalities can be exposed to unauthorized individuals, leading to potential breaches and misuse. An example with the recipe app is that I protected the detail view and list view pages so only logged in authenticated users can view them.

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 my Django web application I will go through creating the login view and create an FBV to show the login form based on Django's authentication form. Next I'll create the HTML template for the login page in a templates folder at project level, within another auth folder. After that since that template is not stores within an app I need to tell Django to look elsewhere for the template in the projects settings.py file under the TEMPLATES array. Finally I'll register the url to the project in the projects urls.py folder by importing the login view and adding its path into urlpatterns. Normally before registering the url you'd specify the URL mapping, however since this code is not within an app it can be skipped.

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 function is a part of Django's authentication framework, specifically from django.contrib.auth. It takes credentials as parameters (often a username and password) and returns a user object if the credentials are valid for a backend. If the credentials are invalid, it returns None. It's a core part of the login process in Django-based applications.**
redirect()	**The redirect() function, part of Django's shortcut functions, is used to return an HttpResponseRedirect to a particular URL. It can take a model, a view name, or a URL to redirect to. It's commonly used after form submissions or actions to send the user to a new page.**
include()	**include() is a function used in Django's URL handling. When defining URL patterns, include() allows you to reference other URLconfs, essentially allowing you to "include" other URL configurations. This modularizes the URL structure, making it cleaner and more maintainable, especially in larger projects where you might want to split URL definitions across multiple files or apps.**

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.

   For a website like YouTube, analyzing the collected data for each video and the videos performance (views, likes, comments) is probably what they use to implement recommendations on peoples homepages as that can filter out the better performing videos. Collecting and analyzing in general can help to increase user experience, engagement, and more.

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

   The different ways to evaluate a QuerySet are:

   Iteration: Looping over a QuerySet.
   Slicing: Taking a subset of the QuerySet using slicing.
   Pickling/Caching: Storing a QuerySet for later use.
   repr(): Displaying a QuerySet's output.
   len(): Getting the number of items in a QuerySet.
   list(): Converting a QuerySet to a list.
   bool(): Checking a QuerySet in a boolean context.

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.

While both QuerySets and DataFrames serve to handle and manipulate data, they are best suited for different contexts. A QuerySet is Django's high-level database query and manipulation API, tailored for retrieving and filtering data from relational databases. It offers simple evaluation, caching, and is optimized for SQL database operations. On the other hand, a DataFrame, originating from pythons 'pandas' library, is a powerful data structure for data manipulation and analysis. DataFrames excel in complex data operations like grouping, pivoting, statistical functions, and handling missing data. Their tabular structure facilitates easy data visualization and integration with plotting libraries. In scenarios that involve heavy data processing, DataFrames provide a more robust and versatile toolkit than QuerySets.

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.
   You can use CSS and Javascript in your Django application in many ways. Such as creating a menu or navigation bar, adjusting text alignment, form, table, and button customization.

2. In your own words, explain the steps you'd need to take to deploy your Django web application.
   1. Implement my .gitignore file
   2. Remove settings.py and add/commit/push to my github repository
   3. Add a Procfile with the text "web: gunicorn projname.wsgi –log-file -"
   4. Install gunicorn and dj-database-url with pip
   5. Edit settings.py with Heroku db configuration
   "import dj_database_url
   db_from_env = dj_database_url.config(conn_max_age=500)
   DATABASES['default'].update(db_from_env)"
   6. Pip install psycopg2-binary
   7. Edit static files configuration and paths in settings.py
   8. Pip install whitenoise
   9. Add whitenoise to middleware in settings.py
   10. Change debug to false in settings.py and change the SECRET_KEY
   11. Create requirements.txt with pip freeze >
   12. Run server to check and commit git changes
   13. Create a Heroku account, and in the CLI in the src folder, heroku login
   14. Heroku create proj-name
   15. Git push heroku main
   16. Heroku run python manage.py migrate
   17. Heroku run python manage.py createsuperuser
   18. Heroku open - check website
   19. Add website link into ALLOWD HOSTS in settings.py
   20. Git add, commit, push origin, push heroku
   21. Go to /admin, login with superuser, and add recipe/proj data
   22. Heroku config:set with secret key

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.

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?

My time management during this achievement was a lot better than during my Full stack immersion course. I also feel like I caught on pretty quickly to the python and django syntax pretty quickly.

b. What's something you're proud of?
I'm proud of my good time management and ability to complete tasks quicker as well as how quickly I was able to learn the content. I look forward to learning more in the future. I'm also just very proud of the end product.

c. What was the most challenging aspect of this Achievement?
The most challenging part was probably this last task, I was getting a lot of errors when trying to deploy on Heroku, but with constant research of errors and such I was able to solve it on my own. That helped to improve my problem solving skills.

d. Did this Achievement meet your expectations? Did it give you the confidence to start working with your new Django skills?
Yes it did, I look forward to creating more Python and Django projects in the future and learning more about the language/framework.

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