

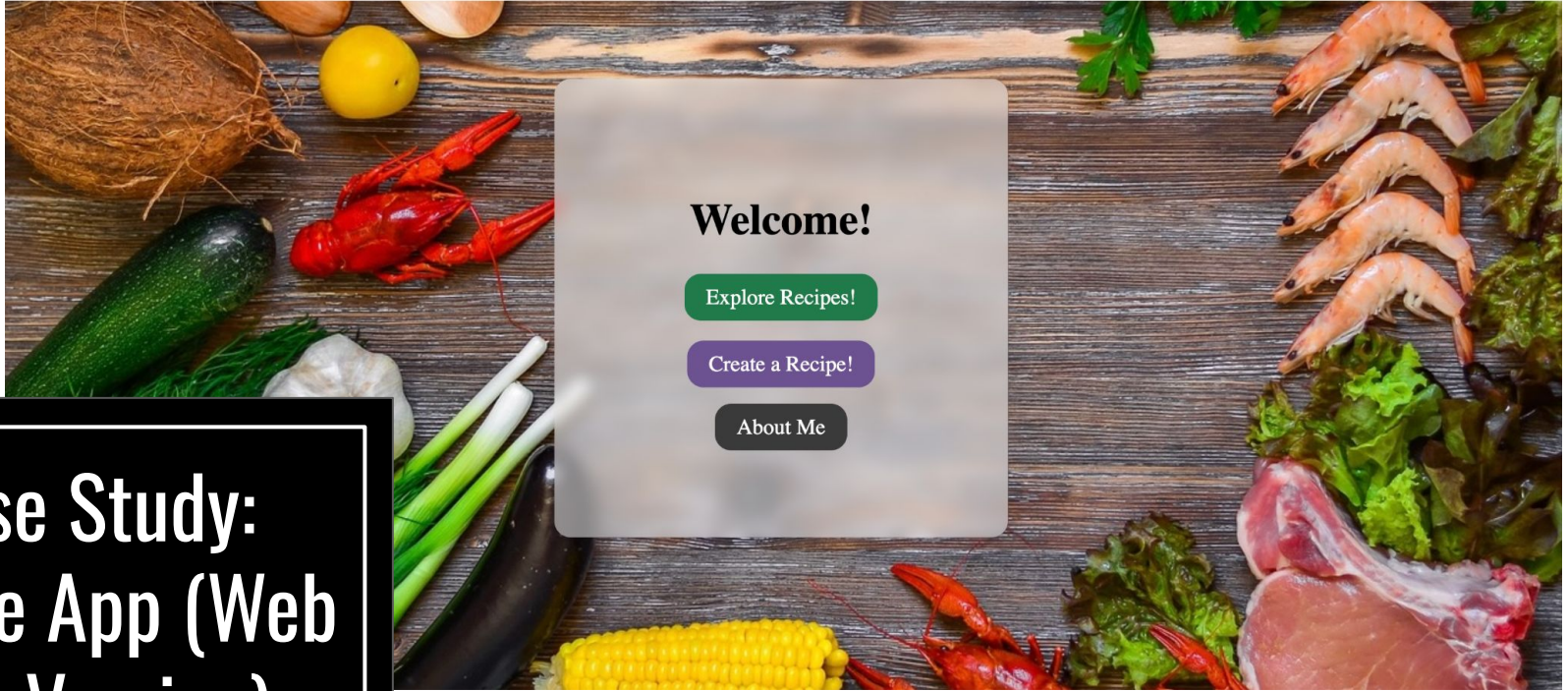
*All Recipes*

*Search*

*Recipe App*

*Profile*

*Logout*



# Case Study: Recipe App (Web App Version)

By: Yufan Xu

# Objective & Purpose

The objective was to create a fully fledged web application version of a recipe app, where users can sign up for an account for full access to the app. Registered users can view recipes created by other users, create their own, and be able to modify and delete their own recipes only. Users should also be able to search and filter recipes, as well as view graphic representations that are displayed alongside the filtered results.

# Tools Used

★ Python

★ Django

★ QuerySet

★ virtualenvwrapper

★ SQLite3

★ PostgreSQL

★ Pandas

★ pip

★ Matplotlib

★ Cloudinary

★ Koyeb

★ WhiteNoise

# PROCESS: The Framework

This application would need a database (to hold the information for each registered user and the recipes) and a frontend (the webpages and interface the user will see when the app is launched). Thus I would benefit from using the Django framework, which would

- 1) handle all the incoming requests from the user (for example, requesting the details of 1 specific recipe),
  - 2) compile the page with the right content based on the data in the database and any custom logic I code using Python, and
  - 3) render the correct web pages in the user's browser via HTML templates.
- Django is a web-development framework with many features ready to be used out of the box, so it can greatly speed up the development process.

# PROCESS: The Framework

Using the Django framework, I set up the “models” for recipes and users by deciding things like the attributes they should have and the types of data for each attribute. These models would be used to create the database tables on the server side. I then considered the various pages the user would encounter while using the app and then created “views” (the logic) and corresponding “templates” (the HTML pages that would present the information to the user). I started with a homepage that all users would see once they open the app, recipe list view (all recipes), recipe details view (for each recipe), and a user profile view. Then later on as more functionality and features are added, I was able to implement these additional features by following Django’s app structure.

# PROCESS: Navigation

Django is structured so I can specify the routes to each module of the project and to each view within each module, but in terms of the user experience, it wasn't easy for users to navigate between the various views and pages (other than directly changing the webpage's url to pull the content we're looking for). To solve this, I created a navigation bar that sits at the top of the interface with links that route users to the appropriate url endpoints, making the navigation much smoother. These links would change over the development process as views and functionality were added.

# PROCESS:

## Search & Visualization

Search functionality and chart visualizations were among the project requirements, so I implemented these within the “recipe” module (as opposed to the “user” module) by creating and displaying a form for users to enter the search criteria. I then used the QuerySet API to extract data from the database, filtering only the recipes that match the search criteria the user inputs into the form. I then converted the queryset to a pandas dataframe, which stores the data in tabular form and allows further data processing.

# Search & Visualization: Challenge

- ★ The default format of the dataframe is a table (one that was not aesthetically pleasing).

	id	book_id	quantity	price	date_created
0	1	5	5	118.55	2021-07-04 17:49:48+00:00
1	3	5	10	237.10	2021-07-01 17:51:48+00:00

Above: example of dataframe table from CareerFoundry course notes

I wanted an interface with a clean minimalistic style that was consistent throughout the app, and the dataframe format didn't fit in. So I further converted the dataframe into a dictionary (a Python data structure that stores values in key:value pairs), which allowed me to access the specific attributes and values of each recipe and style them as desired.



# Search & Visualization: Challenge

- ★ Generating the charts with information I desired (not readily available from the database).

In generating the pie chart (count of recipes by difficulty level), I encountered the issue that the data I wanted to represent wasn't readily available (the "difficulty level" was not a direct attribute of each recipe in the model but was instead calculated automatically by taking the number of ingredients and cooking time attributes). Thus, the dataframe did not include a "difficulty" column. So to access that, I had to first cycle through the filtered results in the queryset to calculate the difficulty level for each recipe and then add a new "difficulty" column to the dataframe. Then I could process that data and extract the value counts of each difficulty level included among the filtered results.

# Search & Visualization: Result



test tea

Difficulty: Easy  
5 minutes



Coffee

Difficulty: Medium  
5 minutes

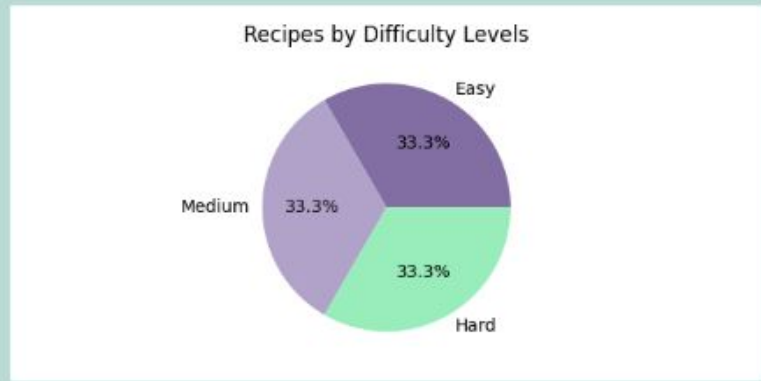
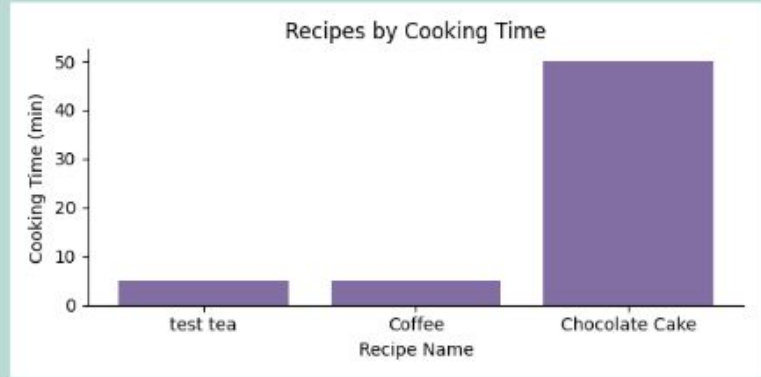


Chocolate Cake

Difficulty: Hard  
50 minutes

## RECIPES SEARCH

Recipe name:  Ingredient:



example of results from recipe search by ingredient "sugar"  
(left: filtered recipes; right: visualization charts)

# PROCESS:

## Implementing Core Functions

The CareerFoundry course notes provide guidance for the project but did not cover the implementation of some core functions, like allowing users to sign up for an account (the “user” model wasn’t required to start) or allowing users to create, modify, and delete recipes. Up until this point, all these functions were done through the Django admin panel, where I created users and recipes for testing during development. Without further guidance, I had to work out the implementation myself.

I first coded forms (including for user signup, recipe creation, and recipe update) and related logic for processing the form input. I then decided where it would make logical sense to display these forms (for example, the signup form available on the login page and the recipe update option available on the details page of a recipe created by the user) and added the forms to the corresponding templates. I chose to incorporate these options as forms and messages in popup dialog boxes, so whole new HTML pages would not be necessary, and these forms and functions would be hidden until the user selects the corresponding button.

# Implementing Core Functions Result



example of “create recipe” function added to homepage as popup dialog  
(above: homepage button; right: popup dialog when clicked)

## Create a Recipe ✕

**Recipe Name:**

**Cooking Time:**

**Ingredients:**

**Description:**

**Instructions:**

**Picture:**  
 No file chosen

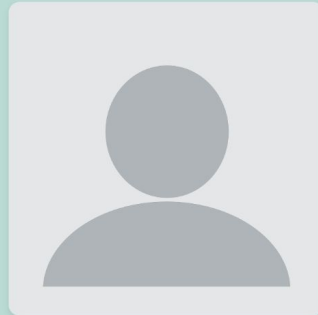
# PROCESS:

## Personal Touch

From a user's perspective, I thought the app could benefit from a little personal touch from the users, so there is more of a face behind each recipe. I had already created the user profile page to display the authenticated user's information (name, photo, and bio). Now this wouldn't be useful if no one else can see it except the current user, so I added a view that would display a selected user's profile along with all their recipes. By clicking the recipe creator's name in a recipe, it brings the user to the creator's page, where they can learn a little about the person behind the recipe. If the logged-in user clicks into their own page, they will also see an additional button to add a new recipe from there as well.

# PROCESS: Personal Touch

example of viewing another user's profile (left) and your own profile (right), both with default profile images (if none is uploaded)



**Test User**

**Bio:**  
Test bio goes here

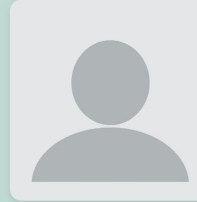
**Recipes by Test User**



**Tea with Honey**



**test tea**



**Yvonne**

**Bio:**

This is a test profile created to test the live web application. You can find out more about me and connect with me by clicking the "about me" button on the home page. Thank you.

[Add Recipe](#)

**Recipes by Yvonne**



**Shrimp Ceviche**

Difficulty: Hard  
30 minutes



**Test**

Difficulty: Intermediate  
100 minutes

# PROCESS: Deployment

Two of the biggest challenges of the project came at the end when I thought everything was set and ready to be packaged for deployment - database configuration and the display of static and media files.

# Deployment: Challenge

#1

## ★ Database Configuration

In development, Django creates a default SQLite database, but this cannot be used in production as it is file based and gets deleted every time the application restarts or if any variables change. So I started to configure the app and database for deployment with Heroku according to the course notes' recommendation, but the deployment doesn't happen successfully. The migrations were not being applied successfully, as if they were being erased or overwritten. After some research I had come to realize that Heroku no longer offers hosting of a PostgreSQL database for free, so I had to pay for a postgres add-on or find an alternative (or else it was still trying to run as a SQLite database). I researched some options for hosting and decided to use Koyeb for this project, as it offered a free web service (for the app) and a free managed PostgreSQL database that could be deployed alongside the web service.



# Deployment: Challenge

## ★ Static and Media Files

The app has static files (the files that don't change during production, like the background images, CSS files, etc.) and media files (recipe images and profile pictures that users upload). To host these files separately from the Django development web server, I had to make some changes. For static files, I utilized WhiteNoise, a middleware package that allows Django web apps to serve their own static files. The media files were trickier, as it turns out both Koyeb and Heroku no longer support saving media files in their free plans.

(right) example of recipes tiles without media images displaying



# Deployment: Challenge

## ★ Static and Media Files

I turned to a 3rd party service to bridge this gap, and I decided on **Cloudinary** for this purpose. The app already allowed users to upload images, so I just needed Cloudinary to provide the ability to store and retrieve these images. The Django Cloudinary Storage package seemed perfect for this use, as it allowed integration with Cloudinary by implementing Django Storage API and only required several lines of configuration. With this integration, my app finally works and looks the way I imagined it.



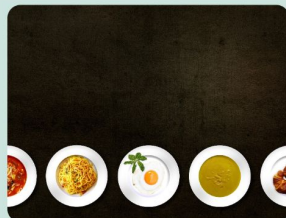
test tea

Difficulty: Easy  
5 minutes



new french toast

Difficulty: Intermediate  
15 minutes



test

Difficulty: Intermediate  
34 minutes

(above) example of recipes tiles with media images displaying properly

Overall I found this to be an enjoyable and rewarding project, as I got to learn about the basics of Python and the Django framework, as well as the configurations it takes to get a Django web app up and running. The Django framework follows a strict structure that may be restricting at times, but for a new developer like myself, it helped guide me to take certain steps.



I met many challenges in the process, but tackling these challenges aided me in gaining a clearer understanding of how Django's structure and flow works. Especially towards the end of the project I was reminded of the fact that change happens quickly in the world of web development (because the course notes were no longer updated!) and the importance of problem solving.



# REFLECTION




For improvements, there are many features that I would want to add when time permits to make the project more complete. This includes options to update user information and delete the profile, as well as username and password reset features. Then some nice-to-haves would be features like categorizing the recipes and the option to "bookmark" or save recipes for quicker reference.



I realized I should've envisioned the end product more thoroughly early on, as I found myself returning to the models to incorporate missing information that I needed from the database. Changing the models midway through a project could cause issues with existing data, and it requires extra work, so next time I will make sure I spend more time defining the models on the first try.



**THANK  
YOU**



Thank you for reading! If interested, you can explore the live application [here](#).