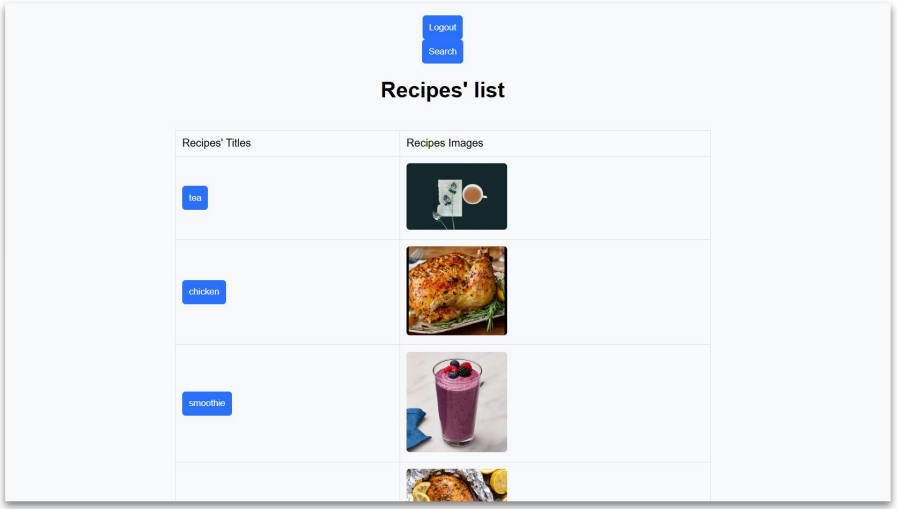
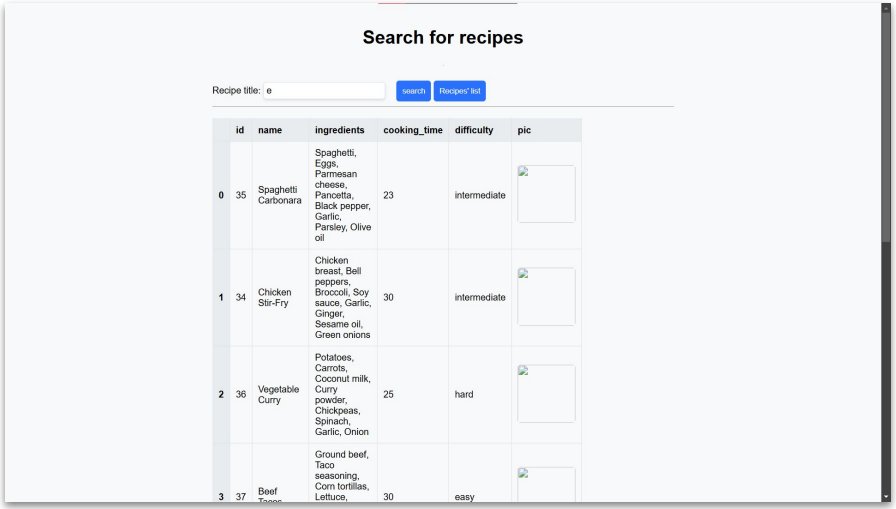
Case Study for Recipe App

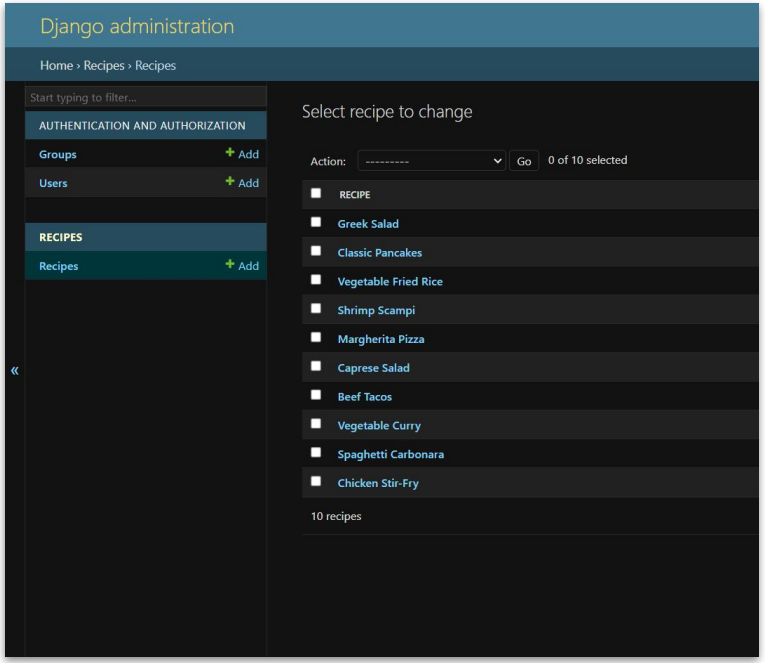




Overview
This recipe application is built with Django and a PostgreSQL database, delivering a seamless experience through an HTML and CSS frontend. Users can securely log in, explore a diverse collection of recipes, and contribute their own.

Purpose & Context
Developed to strengthen backend development skills, this project focuses on database management, user authentication, and data-driven insights while leveraging Django's robust framework.

Objective
The goal was to create a scalable and user-friendly recipe platform, integrating search functionality, automated difficulty ratings, and visual data insights, showcasing expertise in Django, PostgreSQL, and data visualization.



Approach

Django

Built with Django and PostgreSQL, this recipe app lets users log in, explore, and contribute recipes securely. It features search functionality, automated difficulty ratings, and detailed recipe info with robust error handling. The Django Admin interface simplifies management, while data visualizations provide insights into trends.

Django

This project highlights expertise in Django, PostgreSQL, and data analysis, ensuring a scalable and user-friendly experience.

```python
def login_view(request):
    error_message = None
    form = AuthenticationForm()
    # When user hits 'login' button, then POST request is sent is generated
    if request.method == "POST":
        # read the data sent by the form via POST request
        form = AuthenticationForm(data=request.POST)
        # check if the form is valid
        if form.is_valid():
            # get the username and password
            username = form.cleaned_data["username"]  # Read username
            password = form.cleaned_data["password"]  # Read password

            # use Django's built-in authenticate function to check if the user is valid
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                return redirect("recipes:list")  # send user to the desired page
        else:
            error_message = "Oops! Something went wrong."  # error message
    context = {"form": form, "error_message": error_message}
    return render(request, "auth/login.html", context)
```

Using Python

Python's dynamic typing and Django's lazy query execution introduced quirks in handling user input, database queries, and data transformations. The ORM simplifies SQL interactions but can cause unexpected performance issues if queries aren't managed properly. Additionally, Python's whitespace-sensitive syntax enforces readability but requires careful indentation in error handling and data processing. Despite these quirks, Python's flexibility and Django's abstractions made building a scalable, data-driven recipe app seamless.

Challenges

I struggled with serving images on Heroku since its ephemeral filesystem deletes files after each deployment, and I couldn't get an external storage solution like AWS S3 to work properly. Despite multiple attempts to configure Django's media file settings and connect to a third-party service, I ran into authentication issues and deployment errors. Without a paid add-on, hosting images directly on Heroku wasn't an option, and after several troubleshooting efforts, I ultimately couldn't get the setup to function as expected. This challenge highlighted the limitations of free-tier hosting and the need for better media file management in production.

Duration

This project spanned approximately 2 months time to complete. I enjoyed programming with a new language, Python and the tools that Django supplied me with. I would like to do more with Python, even physical applications, like simple electronics.

Thank You For Viewing!

Would you like to connect with me?

My Gmail: VincentAThorne2005@gmail.com