FoodieVerse App

Indulge in endless culinary inspiration!

Made with ❤ By Sumedh Deepak Patkar

Introduction

Introduction

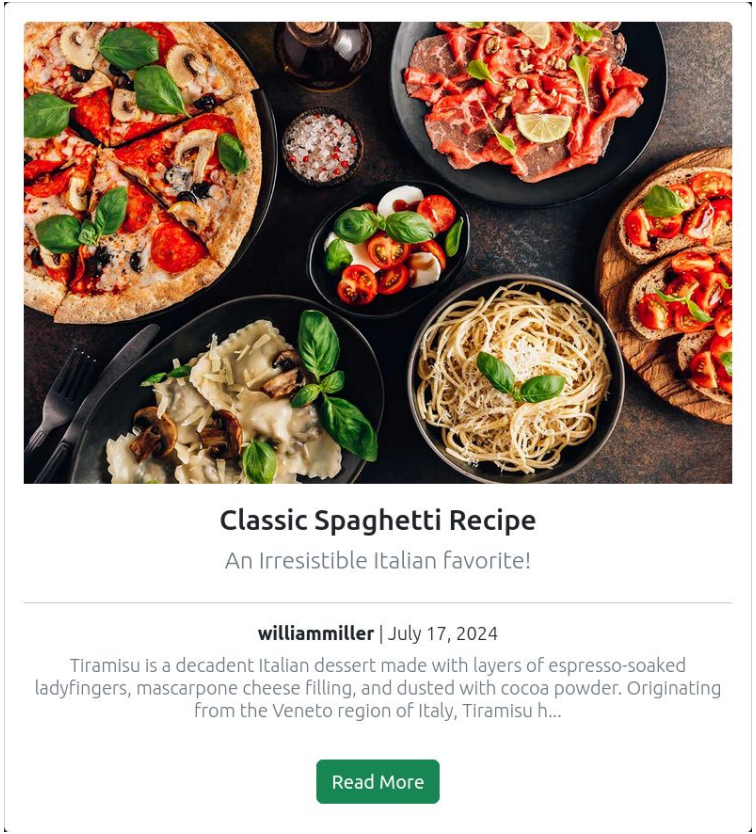
1. What is it?

   A Food Blog application, where users can view **recipes**, discover **cuisines** from other cultures, and also showcase their own **culinary** skills.

2. How is it different?

   Built for everyone - Aspiring and seasoned chefs, recipe enthusiasts, and all foodies

   User Friendly - Responsive design, intuitive navigation, and cross-platform!



Features

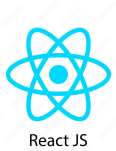
CRUD all the way

- Create Account (SignUp)
- JWT Authentication (Login/Logout)
- Create/Read/Update/Delete (CRUD) a Blog Post
- View all Blog Posts in a Feed

Components and Tech Stack

Frontend

ReactJS and Bootstrap



Cloud and Deployment

Docker, Nginx, Gunicorn. AWS EC2 and S3.

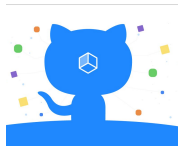
Backend

Django REST Framework and Sqlite3

DevOps automation

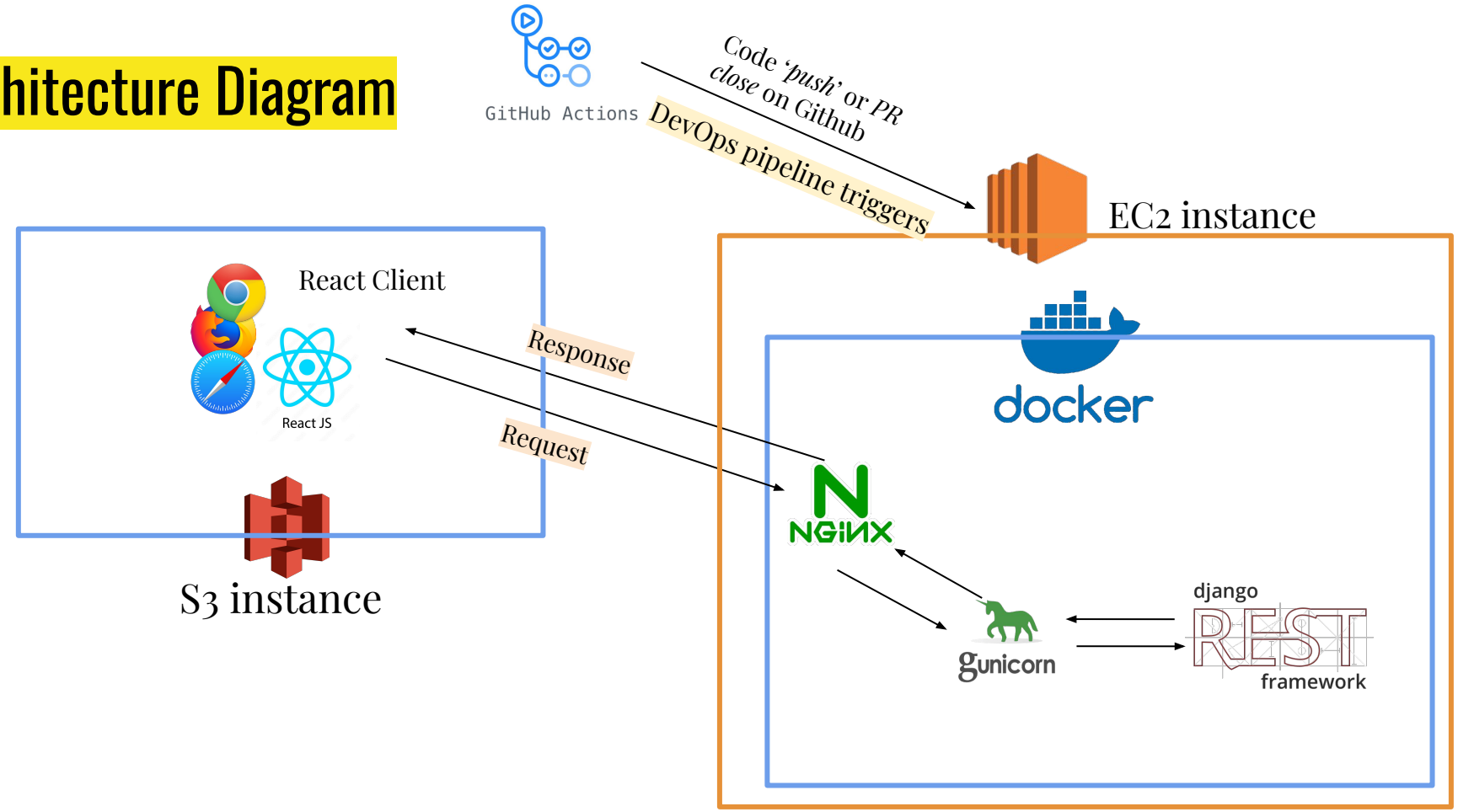
Github Actions and
Github Packages Registry

GitHub Actions



Architecture

Architecture Diagram



Heavy Technical details further...

Backend

1. Creating the Models

Post Model Fields:

- post_id (AutoField, PK)
- title (Unique, String)
- subtitle
- author (ForeignKey: User)
- content
- tags (ManyToManyField)
- publish_date

Tag Model Fields

- name (Unique, CharField)

User (Inbuilt django.contrib.auth)

- Usual ones: username, password, first_name, last_name, email

Backend

2. Writing Serializers

Serialization is converting complex objects or data into a format like JSON or XML that can be

- Transmitted easily
- Interpreted easily by browsers

Deserialization is the opposite

I wrote the following serializers

01. UserSerializer
02. PostSerializer
03. TagSerializer

Highlight - In PostSerializer, I overrode the create() method to create new tags explicitly if they didn't exist, and then add them to the tags[] field of Post model

Backend

3. Creating API Endpoints (urls.py)

- These are inside the ‘blog’ application, redirected from the project urls file

- API specification was written using **OpenAPI** Specification 3.0 (OAS)

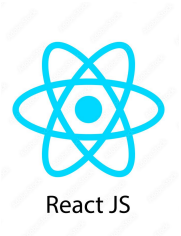
Backend

4. Writing Backend Logic (views.py)

- Created corresponding views for all URLs in the API

- *CreatePost* and *PostDetail* ApiViews together perform all the CRUD operations on Post model

Frontend



- Created Components for
  - Login/Signup/Logout
  - Post Feed
  - CreatePost
  - PostDetail
  - Navbar/Footer

- Used Bootstrap *'Cards'* with *Flexbox* view for the Feed

- Used *axios* library for REST API calls

Cloud ***D*eployment** and ***D*evOps** automation

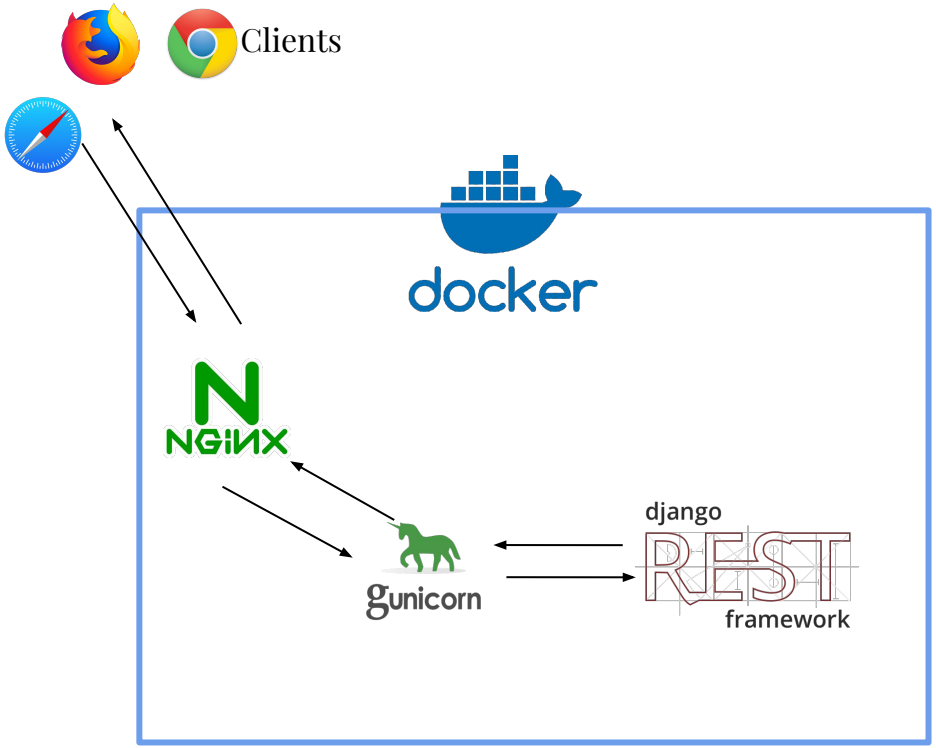
Setting up WSGI Web server and Reverse Proxy Server

Docker Containers

- Setup 2 docker containers
  1. Nginx
  2. REST API with Gunicorn implementing the WSGI

- Used docker-compose to orchestrate both the containers

docker Compose



Deploying React App on AWS S3

Build Step

Built the react app using command

The contents of the *build* folder are production ready

S3 Steps

- Create S3 bucket
- Select configuration (AWS Region, Name, etc)
- Enable static website hosting
- Enter entrypoint (*index.html*)
- Set public access permissions
- Set bucket policy (*GetObject* action needed)
- Upload contents of build folder
- Voila! (Access using given URL)

Deploying Backend on AWS EC2

Docker Compose Step

- Make 3 docker-compose files for dev, ci, and production
- Create nginx.conf file, direct HTTP requests to port 8000
- Build both the *nginx* and *django-api* docker containers

EC2 Steps

- Launch EC2 instance
- Select configuration (I selected *Ubuntu*)
- Set up key-pair
- Copying PUBLIC_IP after instance spawn
- SSH using .pem key to instance
- Perform general apt update
- Deploy app using GitHub Actions DevOps automation (Next Slide)

DevOps pipeline using Github Actions and GPR

Github Actions Steps

- Created **main.yaml** file in .workflows directory
- Build the 2 Docker Images using the ci docker compose file
- Push the images to *Github Packages Registry (GPR)*
- Check if secret variables are available
- Deploy to AWS EC2 (Steps ahead ↠)

Deploy to EC2 Step

- scp the production docker compose yaml file
- Turn off all docker instances running
- Pull the images from *GPR*
- Build the images in production .env
- Check if nginx server works using simple *curl* command (Or on your browser using PUBLIC_IP)
- Access the application!

Future Scope of the Application

Future Scope

- User could upload images in the middle of a blog post.
- AI-based recipe recommendation on feed (user behavior recognition)
- AWS CloudFront for CDN (mentioned in bonus points)

Github Link

https://github.com/Sumedh-Patkar/FoodieVerse-App

Live Demo Link

http://sumedh-roulettech-challenge.s3-website.us-east-2.amazonaws.com/

Thank you for reading!