Contents

Frontend

Per the requirements, I used ReactJS to implement the frontend, which is mainly a one-page brief self-introduction. I have attached the code below. The main page consists of static text and content fetched from components. The content in NameFetcher and DegreeFetcher is retrieved from the backend.

App.js

```
import React from 'react';
import NameFetcher from './components/NameFetcher';
import DegreeFetcher from "./components/DegreeFetcher";

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h3 style={{ textAlign: 'center' }}>My Sample React App for the Junior Software
Engineer application at Roulettech Inc.</h3>
        <NameFetcher />
        <DegreeFetcher />
      </header>
    </div>
  );
}
export default App;
```

DataFetcher.js

```
// src/components/DataFetcher.js
import React, { useState, useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('http://d1blg9mulc439o.cloudfront.net/api/name/')
      .then(response => response.json())
      .then(data => setData(data));
  }, []);

  return (
    <div>
      {data ? <p style={{ marginLeft: '40px' }}>Hello! My name is {data.message}. </p> :
<p>Loading...</p>}
    </div>
```

```
  );
}
export default DataFetcher;
```

DataFetcher.js

```
// src/components/DataFetcher.js
import React, { useState, useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('http://d1blg9mulc439o.cloudfront.net/api/degree/')
      .then(response => response.json())
      .then(data => setData(data));
  }, []);

  return (
    <div>
      {data ? <p style={{ marginLeft: '40px' }}>I am a {data.message}.</p> :
<p>Loading...</p>}
    </div>
  );
}
export default DataFetcher;
```

Backend

I used Django to build the backend according to the requirement. I defined the API routes in the *'urls.py'* file as shown below. I defined the JsonResponse in the *'views.py'* file as shown below:

urls.py

```
from django.urls import path
from .views import get_name, get_degree

urlpatterns = [
    path('name/', get_name),
    path('degree/', get_degree),
]
```

views.py

```
from django.shortcuts import render
# Create your views here.
```

```
from django.http import JsonResponse

def get_name(request):
    return JsonResponse({"message": "Heng Yao"})

def get_degree(request):
    return JsonResponse({"message": "PhD in Computer Science"})
```

Request Diagram

Below is a diagram showing how a request is sent from the frontend to the backend and how a response is sent back from the backend to the frontend.

```
Client (Browser)
       |
       | (1) Request: GET …/api/name/
       v
Frontend (React.js)
      |
      | (2) Request: GET …/api/name/
      v
Backend (Django)
      |
      | (3) View: get_name
      v
      | (4) Response: {"message": " Heng Yao"}
      |
Frontend (React.js)
      |
      | (5) Render: Heng Yao
      v
Client (Browser)
```

Step-by-Step Development and Deployment Summary

1. Set Up the Development Environment:
    a. Install Node.js, npm, Python, and Django.
2. Create the React Frontend:
    a. Set up a simple React component that fetches data from the backend.
3. Create the Django Backend:
    a. Set up a Django project and create API endpoints.
4. Integrate the Frontend and Backend:
    a. Ensure CORS is configured in Django to allow requests from the React frontend.

5. Deployment:
    a. Deploy the frontend to AWS S3 and the backend to AWS EC2.
6. Use AWS CloudFront as a Content Delivery Network:
    a. Configure CloudFront Distribution for S3 and EC2.

Deployment process to AWS

1. Deploying React Frontend to AWS S3
    a. Create an S3 Bucket.
    b. Upload npm build files to S3.
    c. Set up a bucket policy for public access. Adjust later to allow access from CloudFront.
2. Deploying Django Backend to AWS EC2
    a. Launch an EC2 instance.
    b. SSH into the EC2 instance with the pem key file and the instance name.
    c. Set up the environment on EC2, such as installing Python3 and Django Gunicorn.
    d. Migrate the project and deploy using the Gunicorn command. Later, it will be deployed using CloudFront instead.
3. Configuring CloudFront
    a. Create a CloudFront distribution for S3.
    b. Create a CloudFront Distribution for EC2.