Creating a React - Flask application

File setup

Back-end Setup

- Create a **flask-server** folder in the project
  - This can be done with

    ```
    mkdir flask-server
    ```

  - We do this to encapsulate the back-end from the front-end in this project and improve readability
- Create a file in the **flask-server** folder named **server.py**
  - This can be done using

    ```
    touch server.py
    ```

    in bash
  - We have to do this manually with Windows
- Create a virtual environment in the **./flask-server** directory

  ```
  • PS C:\Users\austi\Desktop\React - Flask\flask-server> python3 -m
    venv venv
    PS C:\Users\austi\Desktop\React - Flask\flask-server> .
    \venv\Scripts\activate
  ```

  - This is done to manage python packages and to avoid installing the packages globally
    - We do not want to install globally because this can interfere with other projects on our machine and is overall more error prone.
- Install flask packages

  ```
  • (venv) PS C:\Users\austi\Desktop\React - Flask\flask-server> pip
    install flask
  ```

- Setup back-end API
  - Have a route called members
    - This will return a JSON array of members
  - Connect front-end to the back-end
  - Get the JSON array and display it the screen

```
from flask import Flask

app = Flask(__name__)

# Members api route
@app.route("/members")
def members():
    return {"members": ["Member1", "Member2", "Member3"]}

if __name__=="__main__":
    app.run(debug=True)
```

- debug is set to true because we are in developer mode

- Run this application with

```
PS C:\Users\austi\Desktop\React - Flask\flask-server> python server.
py
```

  - After this we go to **localhost:5000/members** in our browser
  - The link given by the terminal will not work

Front-end setup

- Create a react project that is separate from the back-end

```
PS C:\Users\austi\Desktop\React - Flask> npx create-react-app client
```

  - client is the name for this react application
- Remove unnecessary files in the react app
  - **app.test.js**
  - **index.css**
  - **logo.svg**
  - import for **index.css** in **index.js**
- Connect the front-end to the back-end by setting the same proxy as the back-end
  - We do this by going into the **client/src/package.json**
  - This tells react which port we are running on and so we can make relative requests instead of having to type the full URL

```
"name": "client",
  "version": "0.1.0",
  "private": true,
  #this is where we are adding the proxy
  "proxy": "http://localhost:5000",
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
```

- Replace all starter code in **client/src/app.js**
  - After deleting all of the code, use the **rfce** command to generate boilerplate code for a functional component
  - In this file import **useState** and **useEffect**

```
import React, { useState, useEffect } from 'react'
```

    - **useState** is used for creating a state variable which will contain the data retrieved from the back-end
      - The same state variable will be used to render the data on the page
    - **useEffect** is used to fetch the back-end API on the first render
- Create a state variable in **app.js**

```
function App() {

  const [data, setData] = useState([{}])

  return (
    <div>

    </div>
  )
}
```

- Fetch back-end API

```
• function App() {

    const [data, setData] = useState([{}])

    useEffect(() =>  {
      fetch("/members").then(
        res => res.json()
      ).then(
        data => {
          setData(data)
          console.log(data)
        }
      )
    }, [])

    return (
      <div>

      </div>
    )
  }
```

- In this we are using use effect to fetch the data from the /members route.
- We are then taking what comes out of this and converting it to .json
- Then after this we are using the setData function defined in the state const
- After this we pass in an empty array so that the function only runs once

- Whatever the response from the API is, we set the state to this and use the same state to render the data
- To display the data on the page we use

```
•   return (
      <div>
        {(typeof data.members === 'undefined') ? (
          <p>Loading...</p>
        ): (
          data.members.map((member, i) => (
            <p key={i}>{member}</p>
          ))
        )}
      </div>
    )
```

- This is saying if members is not retrieved or “undefined” then it is showing *Loading…*
- else map each of the members in the array to a p tag and display it

How to run this application

- First start up the back-end server so the front-end can fetch the data

```
• PS C:\Users\austi\Desktop\React - Flask\flask-server> python server.
  py
```

- Next we run the react application as normal since they are going to the same port

```
• PS C:\Users\austi\Desktop\React - Flask\client> npm start
```

- Usually if there are errors when starting the one of the two are not start correctly

Errors

- One I ran into was flask was not being fetched from pip even though I had installed it
  - I ran commands to check the version and where it was located by using

```
• pip --version
  pip show flask
```

  - After I confirmed that it was in the project I restarted and then realized there was a field that was not selected in vs.code for the python language interpreter
  - This solved this problem immediately
- I also ran into an errors when trying to console.log data from the back-end API
  - **localhost/:1 Unchecked runtime.lastError: A listener indicated an asynchronous response by returning true, but the message channel closed before a response was received**
  - This was solved by restarting the back-end server and reloading the front-end

Resources

- https://www.youtube.com/watch?v=7LNl2JlZKHA&t=181s
- https://www.youtube.com/watch?v=mjZmGyEWqkQ
- https://www.youtube.com/watch?v=FIkLJYDDWX0
- https://reactjs.org/docs/getting-started.html
- https://flask.palletsprojects.com/en/2.2.x/