Beginner Project Cycle Documentation: Frontend Development with API Integration

Table of Contents

---

1. Before We Get Started

Objective: Prepare your environment and tools for building a frontend application connected to an API.

1.1 Setting Up Your Tools

Before you can start coding, you need to ensure that your environment is ready. This means installing all the tools that will allow you to write, test, and run your code.

- **GitHub Account**: Sign up for a GitHub account at GitHub if you don't have one already. We will use this for version control, submitting your work, and collaborating.
- **Visual Studio Code (VSCode)**: A code editor is where you'll write and organize your code. Download and install VSCode. It's free, easy to use, and has many helpful features for JavaScript and React development.
- **Node.js & npm**: Node.js is a JavaScript runtime, and npm (Node Package Manager) helps manage dependencies (like React) for your project. Install the latest version of Node.js by following the instructions for your operating system.
- **Terminal/Command Line**: You will be using the terminal (on Mac/Linux) or command prompt (on Windows) to run commands, such as starting your React app or installing packages. Don't worry if you're new to this, we'll guide you through it.

1.2 Getting Started with GitHub Classroom

You will use GitHub Classroom for submitting your work. It allows us to track your progress, give feedback, and review your project.

1. Accept the invitation link provided to join the GitHub Classroom repository.
2. **Fork the repository**: This creates a personal copy of the project for you to work on.
3. **Clone** the repository to your local machine:
   ```
   git clone
   https://github.com/your-username/your-repository.git
   ```
4. **Navigate into your project folder**:
   ```
   cd your-repository
   ```

From this point on, you'll be working within this directory to build your project. Keep your terminal open, as we'll be running several commands.

---

2. Introduction to Frontend Development

Objective: Understand what the frontend is and why it matters in web development.

2.1 What is the Frontend?

The frontend is everything that users see and interact with in a web application. It consists of:

- **HTML (HyperText Markup Language)**: This gives structure to a webpage.
- **CSS (Cascading Style Sheets)**: This styles the webpage, determining how elements look (color, size, layout).
- **JavaScript**: This adds interactivity and functionality to a webpage. For instance, when a button is clicked, JavaScript can trigger an action like displaying new content or sending data.

We will be using **React**, a JavaScript library, to build our frontend. React allows us to build modular components, meaning you can reuse parts of your code and manage the user interface more efficiently.

---

3. Setting Up a React Environment

Objective: Set up a React environment where we will build dynamic and reusable components.

3.1 What is React?

React is a powerful JavaScript library that helps developers build user interfaces efficiently. With React, you don't need to manually update the UI every time something changes in your app. React handles it for you.

3.2 Step-by-Step: Creating a React App

To begin, we will use a tool called `create-react-app`, which sets up a React project for you, including all necessary files and configurations.

1. In your terminal, navigate to your project folder:

   ```
   cd your-repository
   ```
2. Run the following command to create a new React app:

   ```
   npx create-react-app frontend-app
   ```
   - **Tip**: This may take a few minutes to complete. `npx` is a package runner tool that comes with Node.js and allows you to use the `create-react-app` tool without installing it globally.
3. Once the setup is complete, navigate into your new project folder:

   ```
   cd frontend-app
   ```
4. To start the React app and see it in action, run:

   ```
   npm start
   ```
5. Your app will now be running on `http://localhost:3000` in your browser. This is your basic starting point, and we'll build upon this.

---

4. Creating Your First React Component

Objective: Learn how to create reusable components in React and render them in your application.

4.1 What is a Component?

In React, a **component** is a reusable piece of UI. Think of it like a building block. By creating components, you can break up your user interface into independent, reusable pieces that are easy to manage and maintain.

4.2 Step-by-Step: Creating a Component

1. Inside your `src` folder, create a new file called `Welcome.js`.

Add the following code to define a `Welcome` component:

```
function Welcome() {
  return (
    <div>
      <h1>Welcome to My React App!</h1>
      <p>This is your first React component!</p>
    </div>
  );
}

export default Welcome;
```

4.3 Rendering a Component

Now that you have created the `Welcome` component, let's use it in our main app.

1. Open the `src/App.js` file.

Import the `Welcome` component:

```
import Welcome from './Welcome';
```

Replace the default content inside the `App` function with your `Welcome` component:

```
function App() {
  return (
    <div>
      <Welcome />
    </div>
```

```
  );
}

export default App;
```

2. Save the file and go back to your browser. You should now see your `Welcome` component displayed!

4.4 Task: Customize Your Component

- **Prompt**: Change the message inside the `Welcome` component to something more personalized. Add new elements like an image, a list of your favorite hobbies, or a short bio. This is your chance to make the component unique to you.
- **Tip**: You can use standard HTML tags like `<img>`, `<ul>`, and `<li>` inside a React component.

---

5. Introduction to APIs

Objective: Understand what an API is and how it allows the frontend to communicate with the backend or external services.

5.1 What is an API?

An API (Application Programming Interface) is a way for different software systems to communicate with each other. For example, when you log into a website using your Google or Facebook account, the website is using an API to request and retrieve your account details from Google or Facebook.

APIs can be used to send or receive data, such as retrieving user profiles or submitting form data. In this project, we will use an API to fetch data from an external service and display it in our React app.

---

6. Connecting Frontend to an API

Objective: Learn how to fetch data from an API and display it dynamically in your React app.

6.1 Fetching Data in React

React allows us to fetch data using JavaScript's `fetch()` function. This function requests data from a URL (in our case, an API) and returns a promise, which resolves into the data we want to use.

6.2 Step-by-Step: Fetching Data

1. Open your `src/App.js` file.

Modify it to fetch data from an API:

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

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

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then(response => response.json())
      .then(data => setData(data));
  }, []);

  return (
    <div>
      <h1>Posts</h1>
      {data.map(post => (
        <div key={post.id}>
          <h2>{post.title}</h2>
          <p>{post.body}</p>
        </div>
      ))}
    </div>
  );
}

export default App;
```

2. In this example, we are fetching data from a public API that returns a list of posts. We then display each post's title and body.
3. Save the file and see the posts displayed in your app.

6.3 Task: Fetch Your Own Data

- **Prompt**: Modify the API endpoint URL to fetch a different type of data. You can use other free APIs like the Dog API or Cat Facts API. You can experiment by fetching dog pictures, random trivia, or any other kind of fun data.
- **Tip**: Replace `'https://jsonplaceholder.typicode.com/posts'` in the code with the URL of the API you want to use. Be sure to adjust how you display the data depending on the API response. For example, if you're fetching dog images, you might display the images instead of text.

---

7. Deploying Your Project

Objective: Deploy your project online so others can access and interact with your work.

7.1 What is Deployment?

Deployment is the process of making your web application available online for others to use. In this section, we'll walk through how to deploy your React app using **Vercel**. Vercel is a popular platform for deploying React apps because it integrates seamlessly with GitHub and is very simple to use.

7.2 Step-by-Step: Deploying with Vercel

1. **Create a Vercel Account**: Visit Vercel and create an account using your GitHub credentials.
2. **Connect to GitHub**: Once logged in, you'll see an option to connect your GitHub account. This allows Vercel to pull your project directly from GitHub and deploy it.
3. **Import Your Repository**:
    - Click on the "New Project" button.
    - Select the repository you've been working on (the one you forked for this project).
    - Vercel will automatically detect that it's a React app and suggest some basic settings. You can accept the default settings.
4. **Deploy**: Once the project is imported, click "Deploy." Vercel will build and deploy your app. This might take a minute or two.

5. **View Your Live App**: Once the deployment is complete, Vercel will provide a URL where your app is live and accessible from any browser.

7.3 Task: Customize Your Deployment

- **Prompt**: Try customizing the look and feel of your app before deploying it. Add more styling, or add a new feature like a search bar that allows users to filter through the data displayed.
- **Tip**: Deployment is a great way to test your app on different devices. Share your deployment link with friends and family to get feedback or show off your progress!

---

8. Submitting Your Project via GitHub Classroom

Objective: Ensure your project is properly pushed to GitHub and submitted through GitHub Classroom.

8.1 Pushing Your Changes to GitHub

To submit your work for review, you need to push your final code to your GitHub repository.

After completing your project, open your terminal and ensure you are in your project's directory:

```
cd frontend-app
```

Stage all your changes:

```
git add .
```

Commit your changes with a descriptive message:

```
git commit -m "Completed frontend app with API integration"
```

Push your changes to your GitHub repository:

```
git push origin main
```

8.2 Final Submission on GitHub Classroom

Once your code is pushed to GitHub and deployed, you will need to submit the following for final review:

1. **GitHub Repository**: Ensure that your GitHub repository contains all the files for your project, including your `src` folder, the `App.js` file, and the `Welcome.js` component.
2. **Deployment Link**: Include the link to your deployed app in the `README.md` file of your repository. This allows the instructors to access and interact with your live project.
3. **Submit on GitHub Classroom**: Follow the instructions provided by your instructors for submitting your project on GitHub Classroom. Ensure that everything is properly uploaded before the deadline.

8.3 Task: Submit Your Final Reflection

- **Prompt**: Write a short reflection (3-4 sentences) in your `README.md` file about what you learned during the project. Consider discussing any challenges you faced and how you overcame them, as well as any creative elements you added to make the project your own.

---

9. Troubleshooting Common Issues

As you work through your project, you might encounter some issues that can be tricky to solve, especially if you're new to frontend development. Don't worry! Here are some common problems you may face and how to resolve them.

9.1 Node.js and npm Errors

Problem: Node.js or npm Not Installed Correctly
If you see errors like `npm: command not found` or `node: command not found`, it means that Node.js or npm (Node Package Manager) isn't installed or configured correctly.

Solution:

Make sure Node.js is installed. You can verify by running the following command in your terminal:
bash

```
node -v
```

1. This should print the version of Node.js. If not, download and install Node.js from nodejs.org.

If `npm` is not recognized, ensure it is installed by typing:
`npm -v`

2. If this gives an error, reinstall Node.js, as npm comes bundled with Node.

Problem: `npm install` Fails with Dependency Errors
Sometimes, when running `npm install`, you might see errors related to incompatible versions of dependencies.

Solution:

Clear npm cache: This often resolves issues caused by corrupt or outdated packages:

```
npm cache clean --force
```

1. **Install a specific package version**: If the error mentions version conflicts, try installing the required version manually, e.g.:
   `npm install react@17`
2. **Delete `node_modules` and reinstall**: Sometimes, a fresh start helps:
   `rm -rf node_modules`
3. `npm install`

Problem: Missing `package.json`
If you try to run `npm start` or `npm install` and get an error like `No package.json found`, it means your project isn't properly initialized.

Solution: Ensure you're in the correct project directory by running:

```
ls
```

1. If `package.json` isn't listed, you might be in the wrong directory. Navigate to the correct folder.

If the file is genuinely missing, you can reinitialize the project:
`npm init`

2. Follow the prompts to create a new `package.json`.

9.2 API Fetch Errors

Problem: CORS (Cross-Origin Resource Sharing) Error
When fetching data from an API, you might encounter a CORS error that looks like:

```
Access to fetch at 'https://api.example.com/data' from origin
'http://localhost:3000' has been blocked by CORS policy.
```

Solution:

1. **Check the API settings**: Some APIs do not allow requests from certain origins (like `localhost`). Look at the API's documentation to see if it supports CORS, or if there's a specific endpoint for local development.

Use a proxy: If you're working with your own API, enable CORS on the server side. If that's not possible, you can use a proxy to route the requests:
`"proxy": "https://api.example.com"`

Problem: Fetch Fails or Returns an Empty Response
Sometimes the fetch() call completes, but you get an empty array or `undefined` instead of the expected data.

Solution:

1. **Check API endpoint**: Ensure the API URL is correct and functional by opening it in a browser. If it's not returning data there, it's an issue with the API itself.
2. **Check response status**: Use `.then(response => console.log(response.status))` to ensure the response status is `200 OK`.

Handle response properly: Ensure that the response is being correctly converted to JSON:
javascript
Copy code

```
fetch('https://api.example.com/data')

  .then(response => response.json())

  .then(data => console.log(data))

  .catch(error => console.error('Error fetching data:', error));
```

3.

Problem: `Unexpected token < in JSON at position 0`
This happens if the API is returning HTML (like an error page) instead of the expected JSON.

Solution:

1. **Check API URL and parameters**: Ensure that the request URL is correct and that any query parameters or headers are valid.
2. **Check server-side logs**: If you control the API, look at server logs to debug the issue.

9.3 React Errors

Problem: `Module not found`
You might encounter an error like:

```
Module not found: Can't resolve './Welcome'
```

This usually happens if the file you're importing doesn't exist or if there's a typo in the path.

Solution:

1. **Check the file name**: Ensure that the file path is correct and matches exactly, including capitalization (`Welcome.js` is different from `welcome.js`).
2. **Check for missing files**: If the file doesn't exist, create it in the appropriate directory.

Problem: React App Won't Start
If `npm start` fails with an error like `Error: Cannot find module 'react-scripts'`, the dependencies for your project might not be installed correctly.

Solution:

1. Ensure you've run `npm install` inside the project directory to install dependencies.

If that doesn't work, try reinstalling `react-scripts` manually:

```
npm install react-scripts --save
```

2.

Problem: Blank Page with No Errors
Sometimes, you may see a blank page in the browser without any visible errors in the console.

Solution:

1. **Check Console Logs**: Open your browser's developer tools (`F12` in Chrome), go to the Console tab, and check for any error messages or warnings.

Ensure Proper Return in JSX: If a React component is not returning any JSX, it won't render anything:
javascript
Copy code

```
function App() {

  return (

    <div>

      <Welcome />

    </div>

  );

}
```

2.

Problem: State Not Updating Correctly
You may run into a situation where a state variable doesn't seem to update as expected after calling `setState`.

Solution:

Check Asynchronous Nature of State: React's `setState` is asynchronous, so changes may not appear immediately. If you need to update something based on the new state, use `useEffect` to trigger re-rendering:
javascript
Copy code

```
useEffect(() => {
```

```
  // Do something when state updates

}, [stateVariable]);
```

1.

Ensure Correct State Updates: Never mutate the state directly. Always return a new copy when updating:
javascript

```
setData([...data, newItem]);
```

9.4 Deployment Errors

Problem: Deployment Fails on Vercel
If Vercel fails to deploy your app, you may see errors like:

vbnet

```
Build failed due to an error in the React app configuration.
```

Solution:

1. **Check the logs**: Vercel provides detailed build logs. Review them to identify the specific problem (e.g., missing dependencies, incorrect file paths).
2. **Check `package.json`**: Ensure that all necessary dependencies (like `react-scripts`) are listed in `package.json` and properly installed.

Problem: App Works Locally but Not After Deployment
If your React app works fine locally but behaves unexpectedly after deployment (e.g., routing doesn't work or API requests fail), it's often due to differences between local and production environments.

Solution:

1. **Ensure Absolute Paths**: Make sure that all asset paths (like images) are correctly referenced. Use relative paths where possible.
2. **Check CORS for API Calls**: If your API calls don't work after deployment, check if the production environment has different CORS restrictions.

Conclusion

By following this documentation, you have learned the fundamentals of frontend development, built a React application, connected it to an API, and deployed it online. This project demonstrates core skills in web development, including component creation, state management, and data fetching, and provides you with the experience of working with modern tools like React and Vercel.

- **Tip**: Continue building upon what you've learned. You can add more features to your app, try working with different APIs, or explore advanced React concepts such as React Router for navigation or using a state management library like Redux.

We hope this project has been a rewarding learning experience, and we look forward to seeing your final submissions!

Additional Resources

If you want to dive deeper into some of the topics covered, here are a few resources you might find helpful:

- **React Documentation**: https://reactjs.org/docs/getting-started.html
- **Vercel Deployment Guide**: https://vercel.com/docs
- **JavaScript Fetch API Guide**: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Good luck, and enjoy the process of creating something uniquely yours!