Introduction to React and Tailwind

- By Ali Almaamouri
- GitHub: https://github.com/alialm05

What is React?

- React.js is an open-source JavaScript library developed by Facebook (now Meta) for building **user interfaces (UIs)**, especially **single-page applications (SPAs)**. It allows developers to create **reusable UI components** and manage the state of an application efficiently.
- **Key Features of React:**
  - **Component-Based Architecture** – Breaks the UI into reusable pieces.
  - **Virtual DOM** – Improves performance by updating only the necessary parts of the UI.
  - **One-Way Data Binding** – Ensures predictable data flow.
  - **State Management** – Makes UI updates more efficient and dynamic.
  - **Hooks** – Provides a way to use state and lifecycle methods without writing class components.

Why use React?

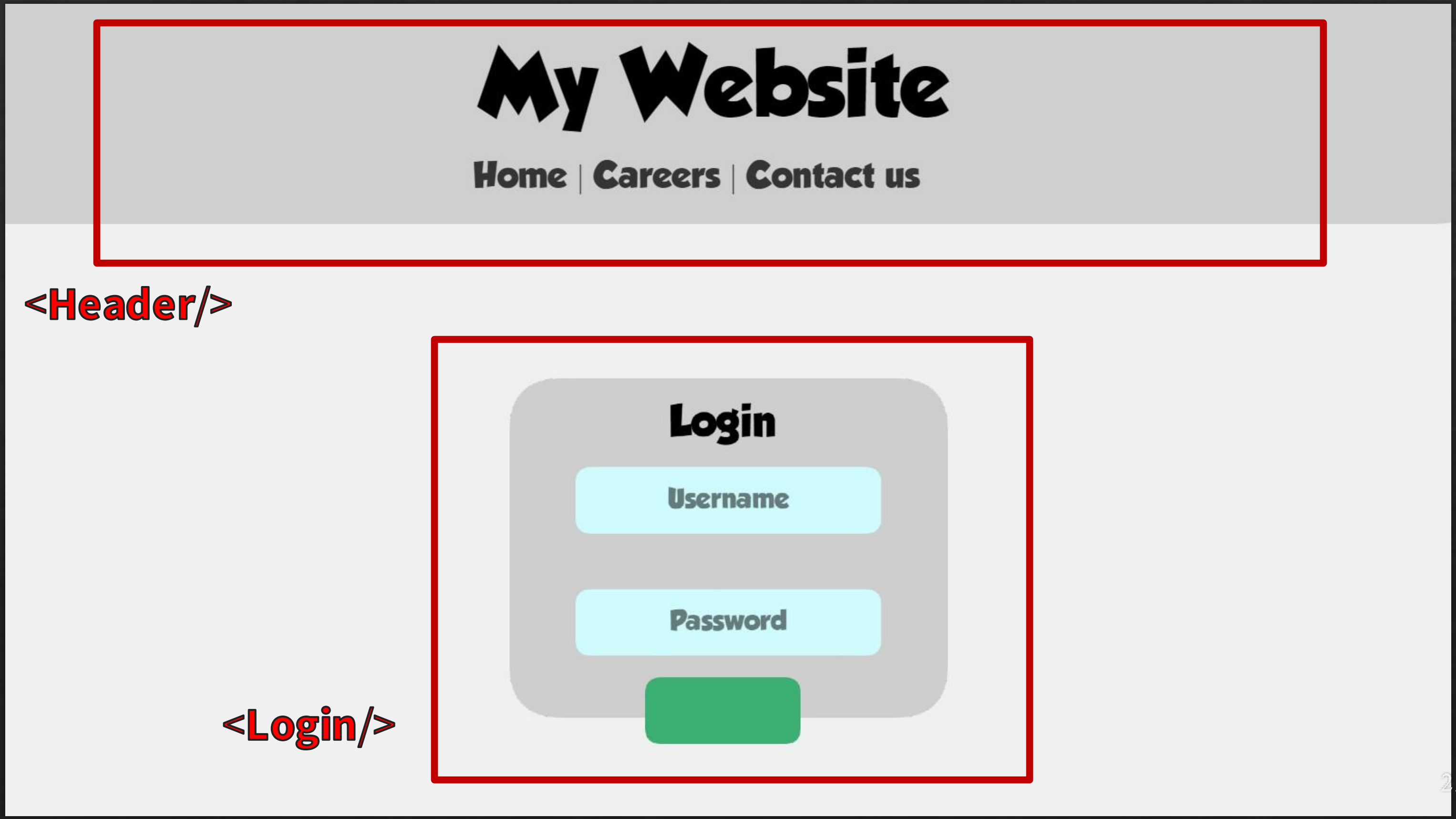
- **Faster Development**
  - React’s component-based system allows developers to **reuse code**, speeding up development.
- **Better Performance**
  - The **Virtual DOM** updates only changed elements, while vanilla JS manipulates the real DOM directly, which is slower.
- **Easier State Management**
  - React’s built-in **state and context API** makes handling UI changes smoother compared to manually tracking state in vanilla JS.
- **Scalability**
  - Large-scale applications benefit from **modular components**, making maintenance easier.
- **Cross-Platform Development**
  - React can be used for web (React.js), mobile (React Native), and even desktop apps (Electron.js).
- **SEO & SSR (Server-Side Rendering) Support**
  - Libraries like **Next.js** make React SEO-friendly.
  - Vanilla JavaScript apps often struggle with SEO optimization.

Reacts Component Structure

- React works in a way where you can reuse components in HTML
- These components are made in a Javascript file (JS or JSX), and can be exported to be used in web pages
- These components can have an active state
- When components appear/disappear on the web app, it is refered as being 'mounted' or 'unmounted'

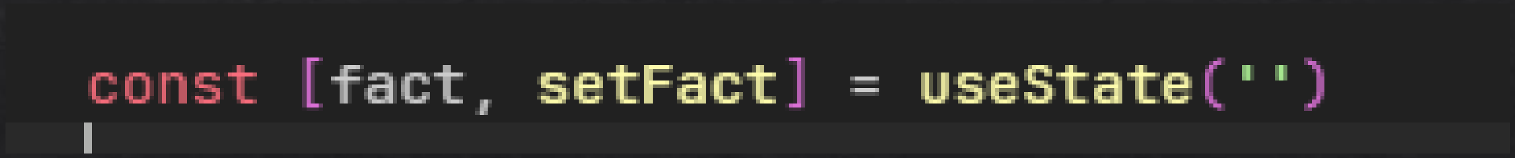
React Components Cont..

- Each components is treated like an HTML tag, except its in JSX



React Hooks

- What is a Hook?
  - A Hook is a special function that lets you “hook into” React features.
- UseState Hook:
  - This hook will let us add a “state” to a react component
  - Similar to a variable, This hook will store information, and help us detect changes that value we stored, ex. The fact string
  - it consists of an array, where the first value is the value we store, and the second is the updater method



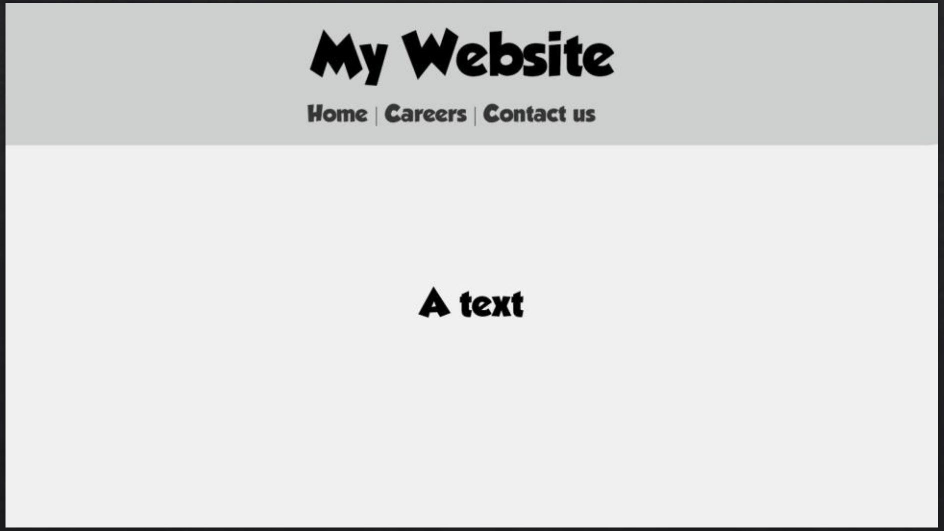
```
const [fact, setFact] = useState('')
```

  - When this value is changed (using setFact), it will also modify (re-render) the changes in the web page as well automatically

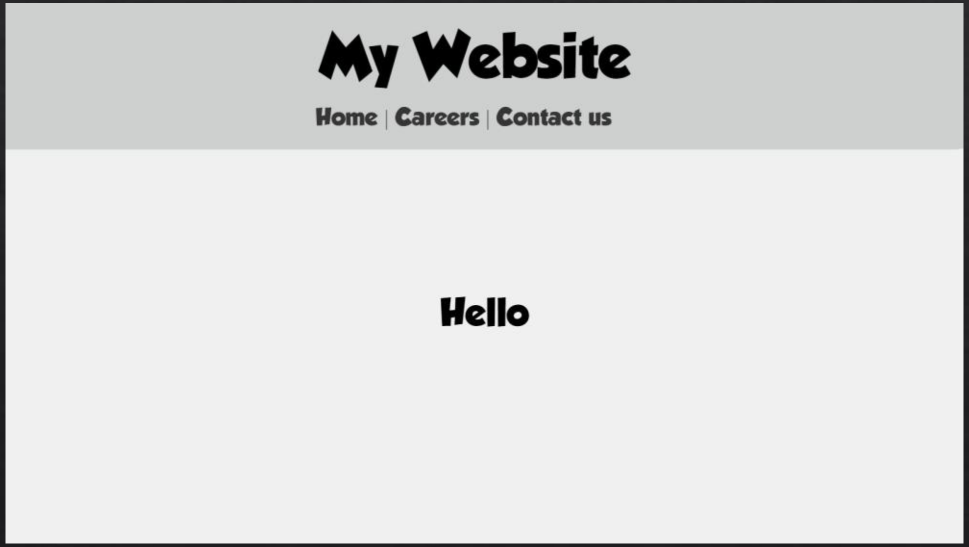
UseState Hooks Cont.

- Take an example of a hypothetical state called “text”, that displays a piece of text on the web-page

```
const [text, setText]= useState(“a Text”)
```



```
setText(“hello”)
```



React Hooks Cont.

- UseEffect Hook
  - This hook allows us to make side effects in components
  - If we want something to happen once a component renders, this is the hook we want to use

```
useEffect(setup, dependencies?)
```

  - An example can be when you want to fetch some data from an API, to display then on the page
  - The `'setup'` will be the function that will run once the current component or dependant component renders

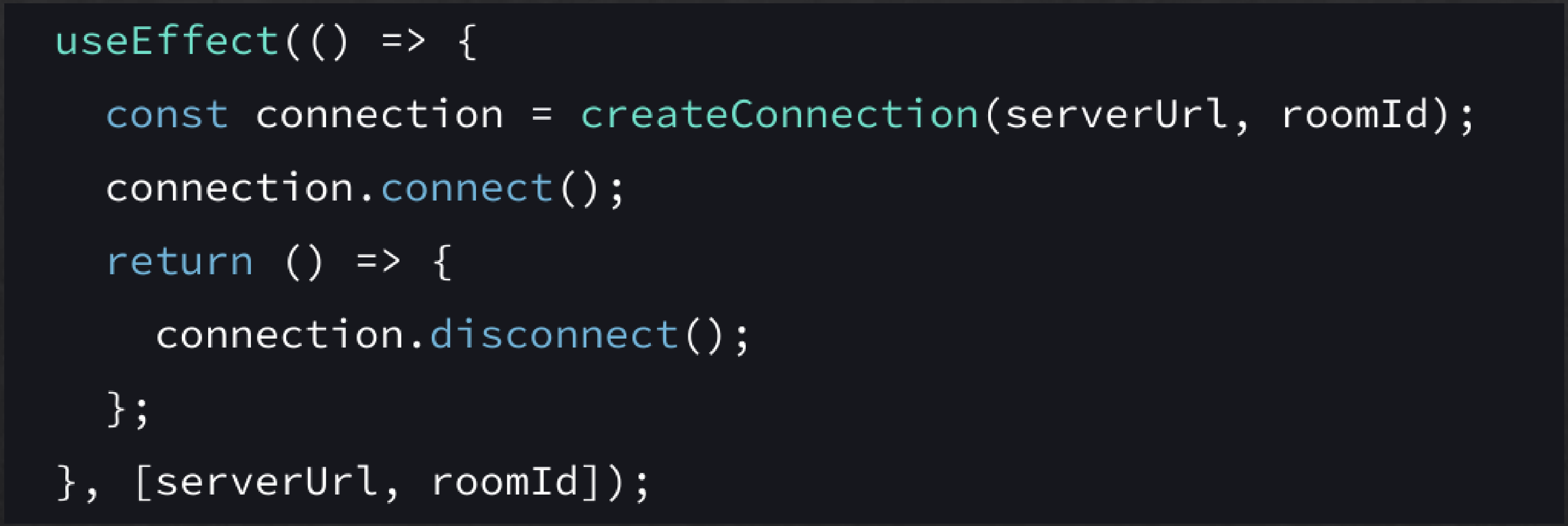
Effect Hook Cont.

- An example

```
useEffect(() => {
  const connection = createConnection(serverUrl, roomId);
  connection.connect();
  return () => {
    connection.disconnect();
  };
}, [serverUrl, roomId]);
```

- In this case, setup function will run whenever serverUrl or roomId is updated, if we only want the setup function to run once on render, we can make the array empty

Effect Hook Cont.

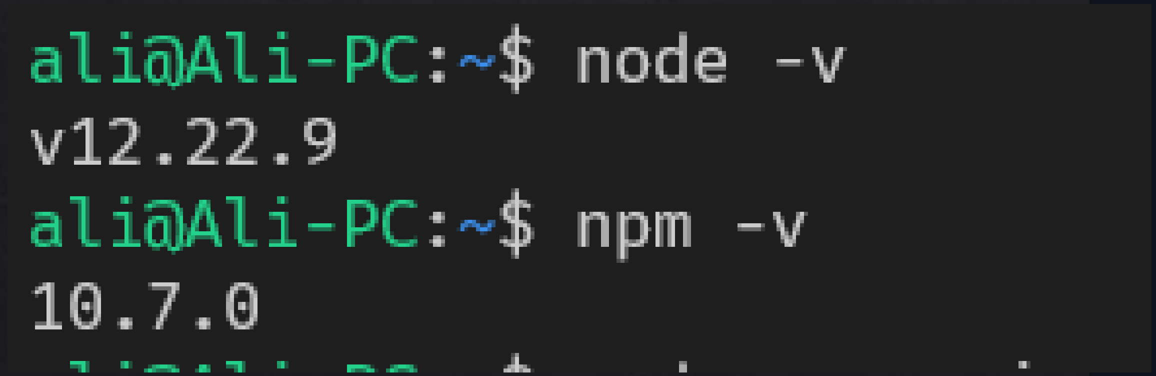


```
useEffect(() => {
  const connection = createConnection(serverUrl, roomId);
  connection.connect();
  return () => {
    connection.disconnect();
  };
}, [serverUrl, roomId]);
```

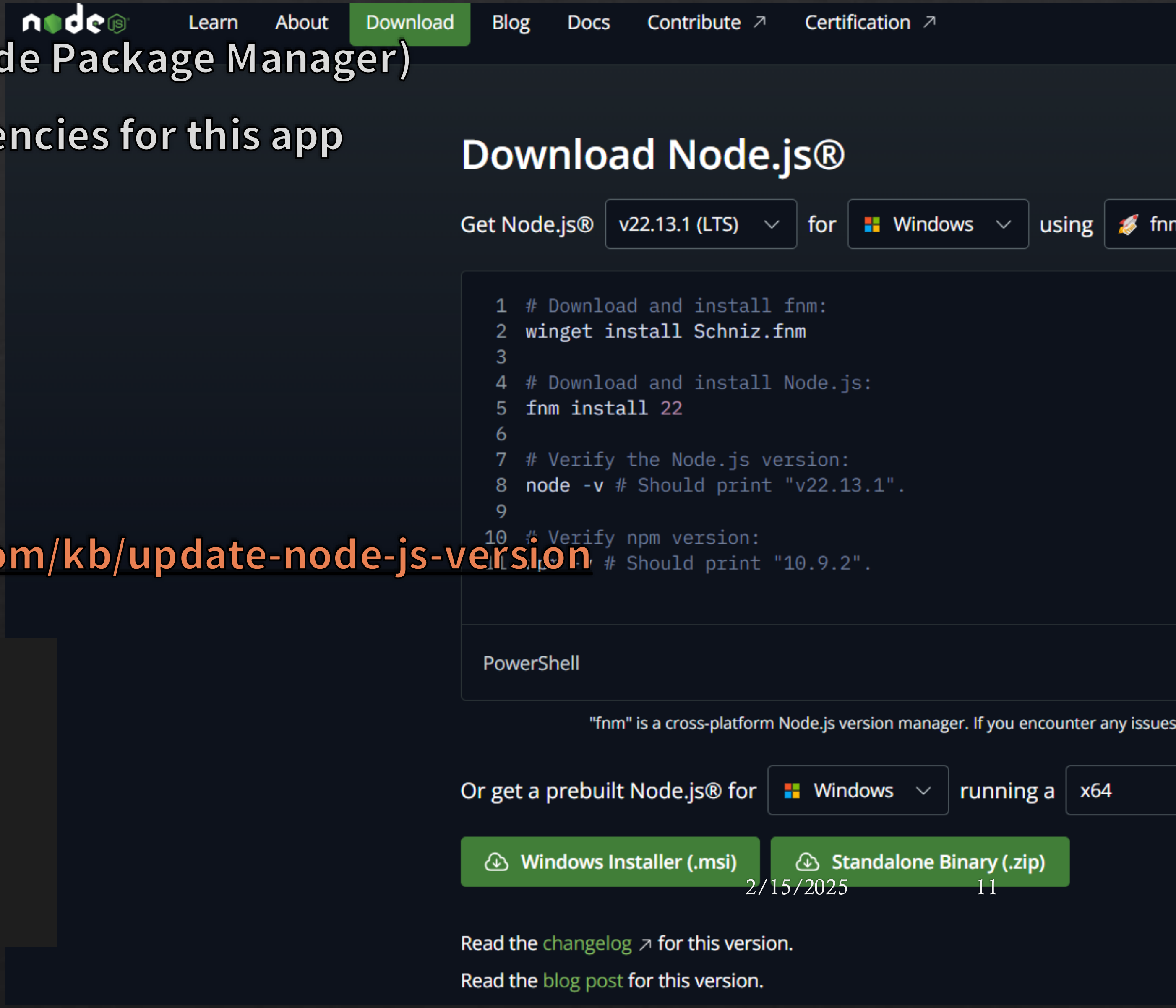
- In the setup function, you also realize there it returns a cleanup function, this is called a 'cleanup' function because it runs once the current component unmounts or unrenders
- In this example, the connection is disconnected because there is not reason to keep the connection if the component is unmounted

To start

- To start with this project, you need NPM (Node Package Manager)
- This will help us install the required dependencies for this app
- https://nodejs.org/en/download
- Verify the installation by typing
  node –v or node --version in the terminal
- Node version 18+ and npm version 10+
  is needed for this project
- If you need to update: https://phoenixnap.com/kb/update-node-js-version



```
ali@Ali-PC:~$ node -v
v12.22.9
ali@Ali-PC:~$ npm -v
10.7.0
```



Download Node.js®

Get Node.js® v22.13.1 (LTS) for Windows using fn

```
# Download and install fnm:
winget install Schniz.fnm

# Download and install Node.js:
fnm install 22

# Verify the Node.js version:
node -v # Should print "v22.13.1".

# Verify npm version:
npm -v # Should print "10.9.2".
```

PowerShell

"fnm" is a cross-platform Node.js version manager. If you encounter any issues

Or get a prebuilt Node.js® for Windows running a x64

Read the changelog ↗ for this version.

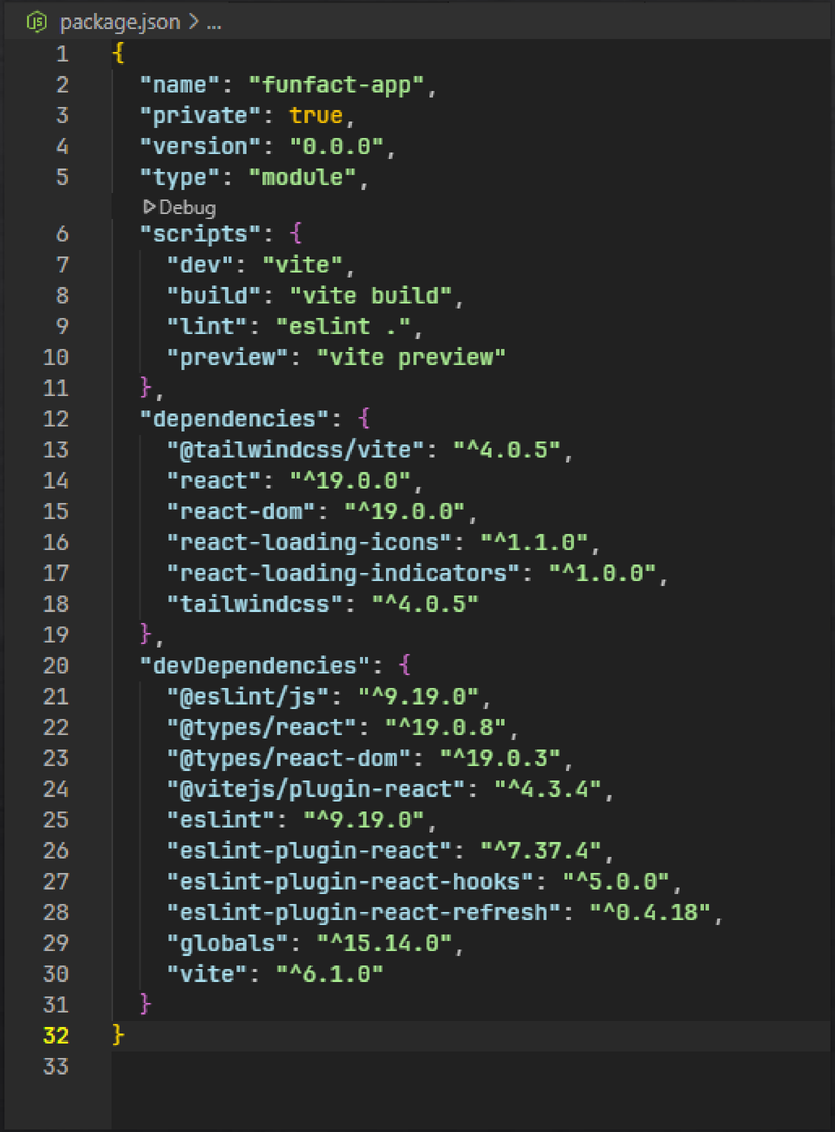
Read the blog post for this version.

Installing and Creating Vite Project

- When we have NPM, we can now use the node package manager to install (if not installed already) Vite and create a new Vite project
- In the Vite docs: https://vite.dev/guide/
- This will create a new Vite project, and if Vite is not installed, it will install it automatically
- Vite will be our toolkit for the project, and we can choose react from the list of frameworks, and Javascript as the language

Package.json file

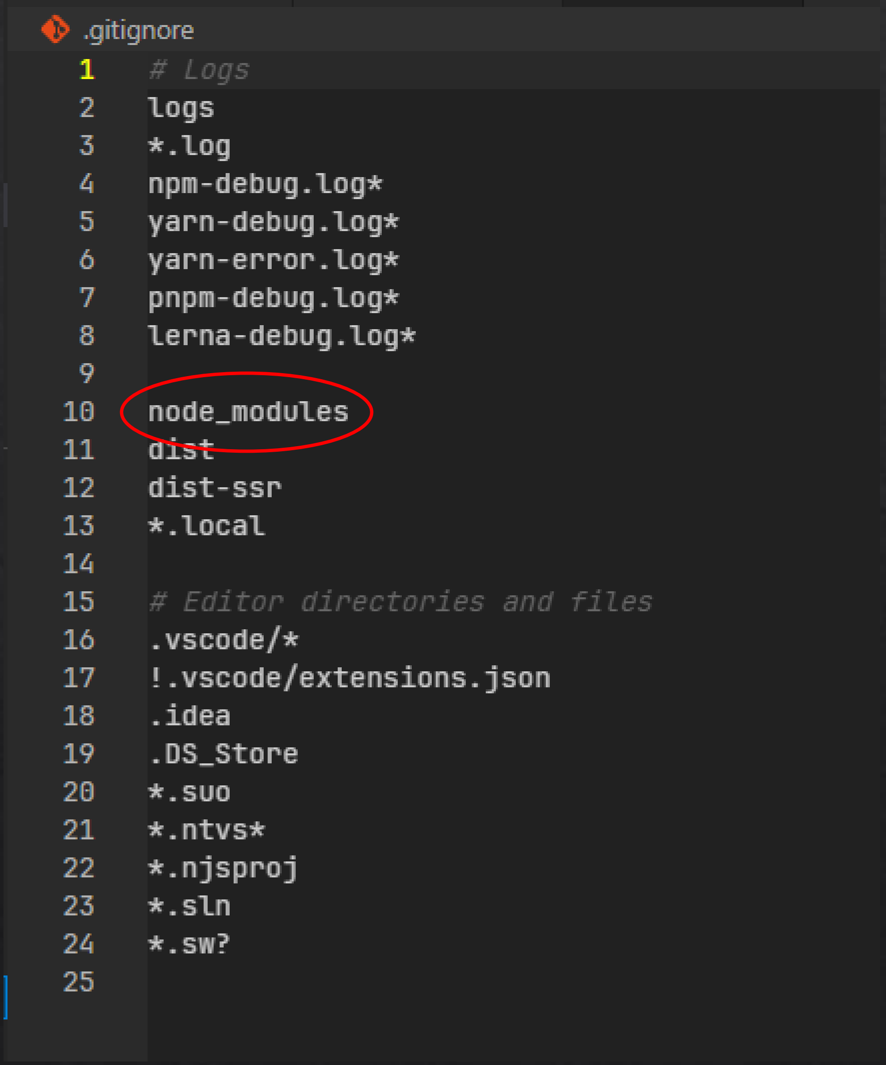
- This file contains some configuration for your projects, including required dependencies?
- These dependencies are like a grocery list, it gives you a list of the dependencies, and you can install them using npm
- This is better than transferring all of the actual packages
- will be helpful when upload to GitHub (it costs less disk space) and moving your projects to other devices
- The command ‘`npm install`’ will install all the required dependencies in this list
- This list also specifies the versions to install for each package
- The difference between dev dependencies and normal dependencies, is that normal dependencies are required in production (ex. react)



```json
{
  "name": "funfact-app",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@tailwindcss/vite": "^4.0.5",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "react-loading-icons": "^1.1.0",
    "react-loading-indicators": "^1.0.0",
    "tailwindcss": "^4.0.5"
  },
  "devDependencies": {
    "@eslint/js": "^9.19.0",
    "@types/react": "^19.0.8",
    "@types/react-dom": "^19.0.3",
    "@vitejs/plugin-react": "^4.3.4",
    "eslint": "^9.19.0",
    "eslint-plugin-react": "^7.37.4",
    "eslint-plugin-react-hooks": "^5.0.0",
    "eslint-plugin-react-refresh": "^0.4.18",
    "globals": "^15.14.0",
    "vite": "^6.1.0"
  }
}

```

.gitignore

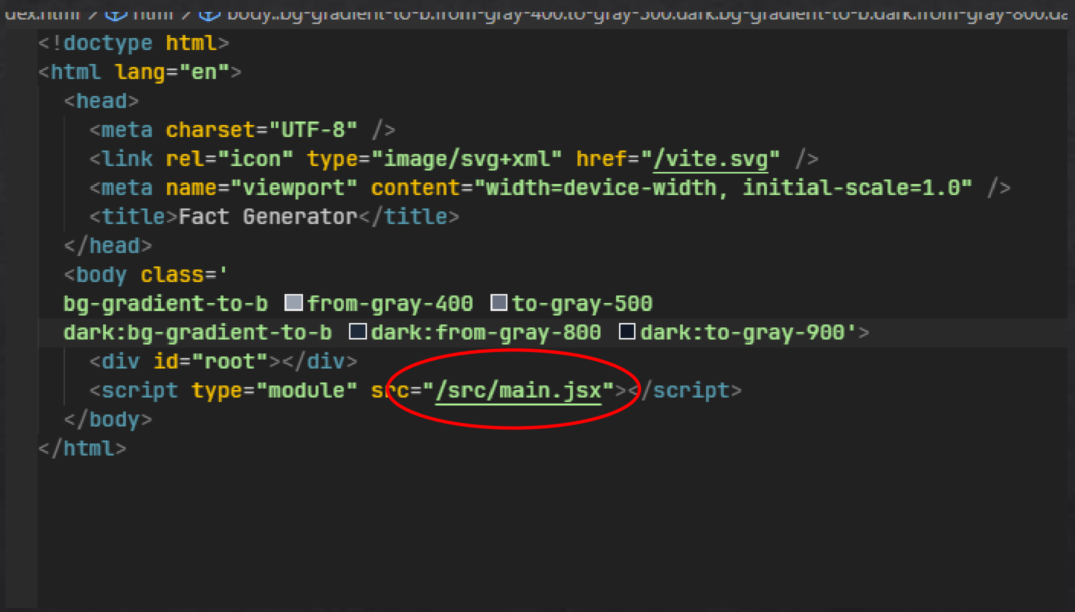
- This file is helpful when you want to ignore some files or directories when upload to GitHub
- Since we have the *package.json* which lists all the dependencies we need, the *node_modules* folder can be ignored since we can just run ‘`npm install`’ to install all the dependencies anyways when working on a new device
- It wouldn’t be feasible to transfer all the module modules because it will take unnecessary amount of space



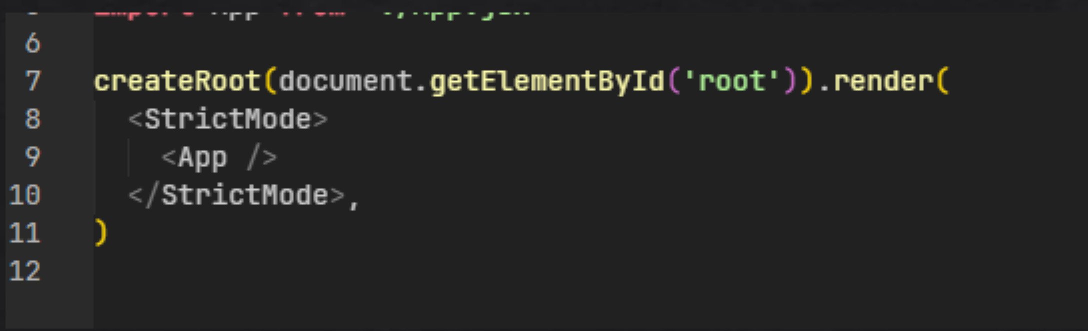
```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

node_modules
dist
dist-ssr
*.local

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?
```

Project Structure

- In our index.html (the root html of our project, we see the script in the body tag called main.jsx)
- In main.js it renders the root of our project, with the custom App
- What is JSX file?
  - Its an extension to JavaScript, where it allows you to write HTML-like code directly within JavaScript
  - JSX requires all HTML attributes to be written in camel case unlike normal HTML attributes:
    - `onClick` instead of `onclick`
    - `className` instead of `class`



```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Fact Generator</title>
  </head>
  <body class='
  bg-gradient-to-b ■from-gray-400 ■to-gray-500
  dark:bg-gradient-to-b □dark:from-gray-800 □dark:to-gray-900'>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```



```jsx

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)

```

Project Roadmap

1. Make a Fact Generator Component

2. Use a useState Hook to store the current fact displayed on screen

3. Make a function to fetch a random fact, and then setting the ‘fact’ state to that fact we received

4. Use a useEffect hook to generate the fact on the first render of the Component

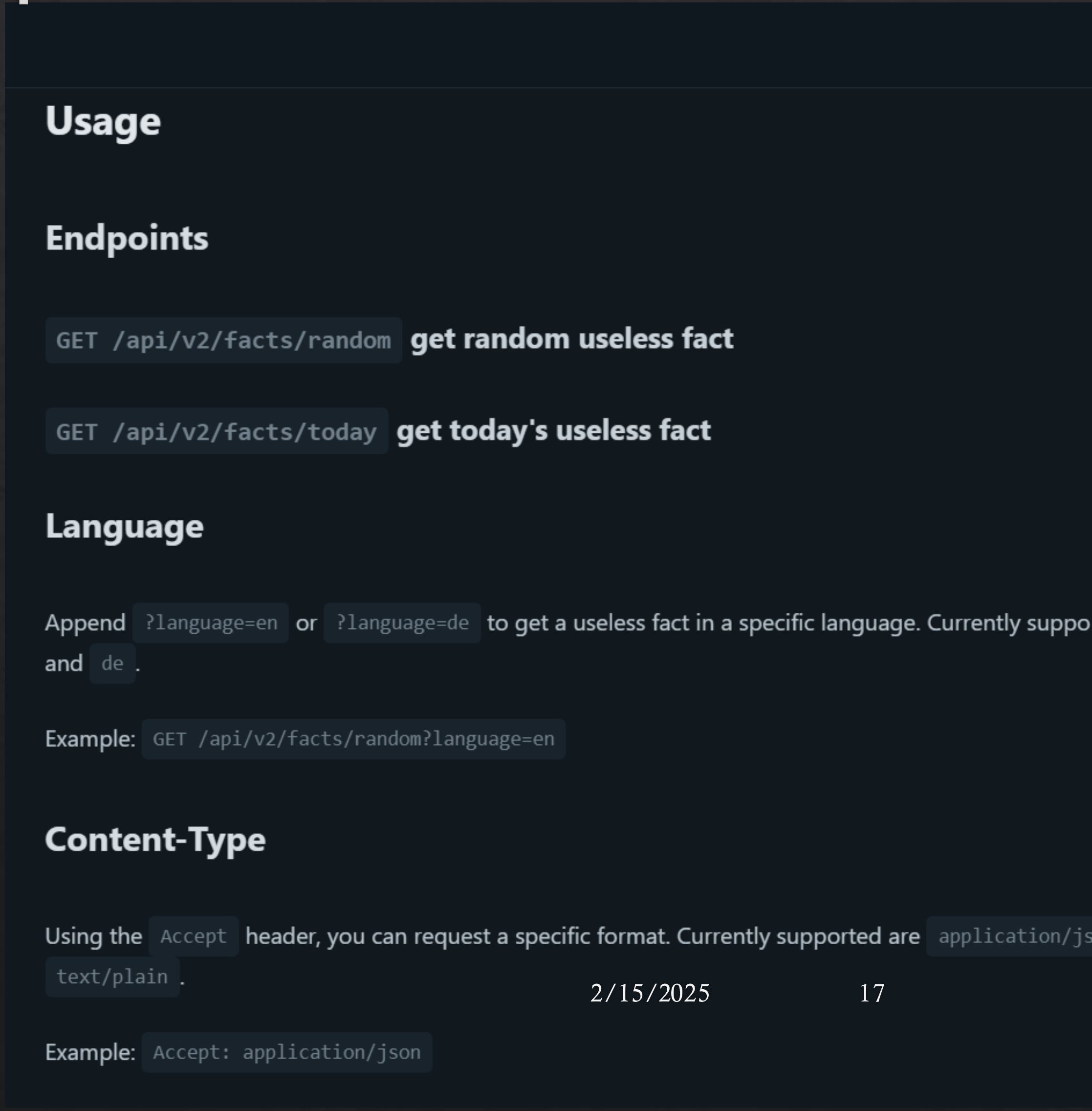
5. Make tweaks

   - Make a loading state, to make sure we don’t send requests while we are still waiting for a response from the API

   - Add some tailwind

Facts API

- We need an API Endpoint to generate these random facts from
- One good endpoint that is free and public is: https://uselessfacts.jsph.pl/
- This is a good API to use since it doesn't require any authorization, and the process will be quicker
- but you can use your own API if you want for this project
- We can use this path given to us fetch a random fact from the endpoint using a GET HTTP Method
- We can test this in the browser, by appending it to the URL



Usage

Endpoints

`GET /api/v2/facts/random` **get random useless fact**

`GET /api/v2/facts/today` **get today's useless fact**

Language

Append `?language=en` or `?language=de` to get a useless fact in a specific language. Currently suppo
and `de`.

Example: `GET /api/v2/facts/random?language=en`

Content-Type

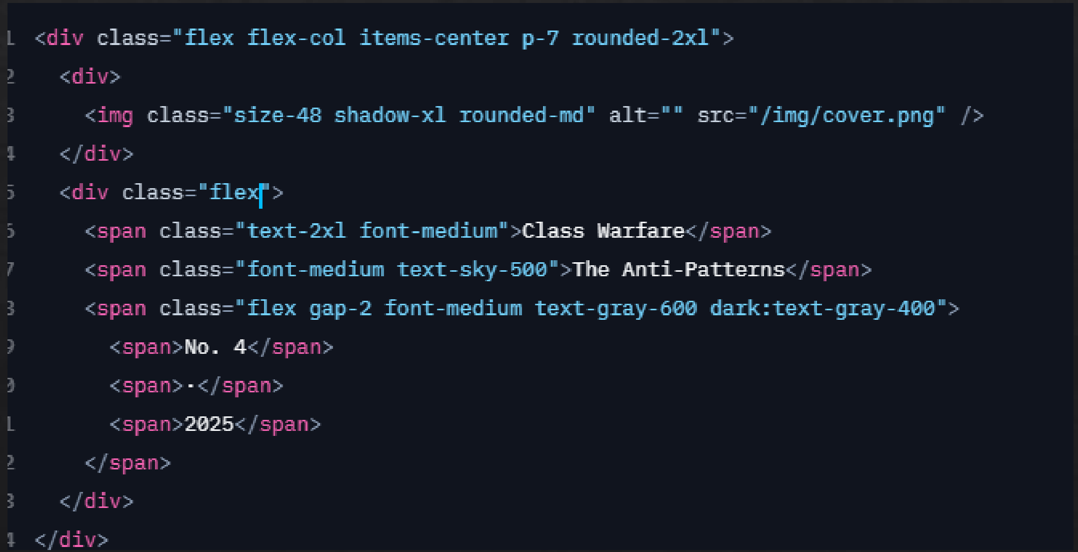
Using the `Accept` header, you can request a specific format. Currently supported are `application/js`
`text/plain`.

Example: `Accept: application/json`

TailwindCSS

- A CSS Framework that gives us re-designed utility classes for easier and direct styling of our HTML components

- Tailwind CSS works by scanning all of your HTML files, JavaScript components, and any other templates for class names, generating the corresponding styles and then writing them to a static CSS file.



```html
<div class="flex flex-col items-center p-7 rounded-2xl">
  <div>
    <img class="size-48 shadow-xl rounded-md" alt="" src="/img/cover.png" />
  </div>
  <div class="flex">
    <span class="text-2xl font-medium">Class Warfare</span>
    <span class="font-medium text-sky-500">The Anti-Patterns</span>
    <span class="flex gap-2 font-medium text-gray-600 dark:text-gray-400">
      <span>No. 4</span>
      <span>·</span>
      <span>2025</span>
    </span>
  </div>
</div>
```

TailwindCSS

- Let’s add it to our project using their documentation:
  - https://tailwindcss.com/docs/installation/using-vite

Final Project

- You can find this project in my GitHub Repository
  - https://github.com/alialm05/fun-fact-generator

Useful links:

- https://legacy.reactjs.org/docs/hooks-state.html
- https://legacy.reactjs.org/docs/hooks-effect.html
- https://react.dev/reference/react/hooks
- Installing Tailwind CSS with Vite - Tailwind CSS
- Getting Started | Vite
- React Loading Indicators
- Random Useless Facts
- https://github.com/alialm05/fun-fact-generator