4 - Intro to React

React education, 2024.

MITHRIL

Overview

- Basics
- JSX
- Components
- Basic routing

MITHRIL

Basics

About React

- React is a JavaScript library for building user interfaces, maintained by Meta (Facebook Inc.) and community of individual developers and companies
- With React you can:
  - Design simple views for each state in your application
  - Update and render just the right components when your data changes
  - Build encapsulated components that manage their own state, then compose them to make complex UIs
- Key terms:
  - SPA - Single-page application
  - Virtual DOM

MITHRIL

About React

- **Single-page application** is an application that loads a single HTML page and all the necessary assets (such as JavaScript and CSS) required for the application to run.
    - Any interactions with the page or subsequent pages do not require request to the server which means page is not reloaded
    - Building a single-page application in React is not a requirement
- **Virtual DOM** is programming concept where representation of a UI is kept in memory and synced with the "real" DOM by a library such as ReactDOM
    - This process is called reconciliation
    - This approach enables declarative API: you tell React what state you want the UI to be in, and it makes sure the DOM matches that state
    - Virtual DOM is usually associated with React elements since they are the objects representing the UI

MITHRIL

Development environment

- Development environment for React apps consists of following technologies:
    - **Node.js** - an open-source JavaScript runtime environment for easily building server-side applications. It's also a runtime that powers many client-side development tools for modern JS libraries and frameworks
    - **npm** (Node Package Manager) - stands for two things, first and foremost, it is an online repository for the publishing of open-source Node.js projects. Second, it is a *command-line utility* for interacting with said repository that aids in *package installation*, *version management* and *dependency management*
        - Other available package managers: Yarn, PNPM, Bun

MITHRIL

Development environment

- **npm** comes bundled with Node.js, so there is no need for separate installation.
- How to use npm?
  - Browse npm packages on npmjs.com
  - Install package by running command in terminal: **npm install <package-name>**
  - Uninstall package by running command in terminal: **npm uninstall <package-name>**
  - Update package by running command in terminal: **npm update <package-name>**
  - When you have a node project with package.json file, you can run terminal command from the root of the project to install all of the dependencies: **npm install**

MITHRIL

Development environment

- When executables are installed via npm packages, npm creates links to them:
  - Locally: npm install <package-name> (links created in /node_modules/bin./)
  - Globally: npm install -g <package-name> (links in global directory, /usr/local/bin on Linux & Mac, %AppData%/npm on Windows)

MITHRIL

Development environment

- **npx -** npm package runner
    - CLI tool whose purpose is to make it easy to install and manage dependencies hosted in npm registry.
    - It allows us to run any sort of Node.js based executable that you would normally install via npm.
    - npx always runs latest version, there is no need to have globally installed package.
    - It can execute a locally installed packages, and packages that are not previously installed.
    - About difference between npx and npm check this discussion.

MITHRIL

Development environment

- **package.json** vs **package-lock.json**
- package.json is information to npm that allows it to identify the project as well as handle the project dependencies.
- package-lock.json is automatically generated for any operations where npm modifies either the node_modules tree, or package.json.
    - It describes a single representation of a dependency tree such that teammates, deployments and continuous integration are guaranteed to install exactly the same dependencies.

MITHRIL

Create a React App

- In order to create React App you will need to have development environment which consists of *Node* and *npm*
- Check you version by executing following in your terminal:
  - *node -v*
  - *npm -v*

MITHRIL

Create a React App

- Several different ways how you can create a React App:
  - Manual setup with compiler and bundler
  - CRA - create-react-app
  - Vite
  - Nx
  - React Meta frameworks - Next.js, Remix, Gatsby, Astro
  - StackBlitz - instant developer environment, zero config (not for real production projects)

MITHRIL

Create a React App

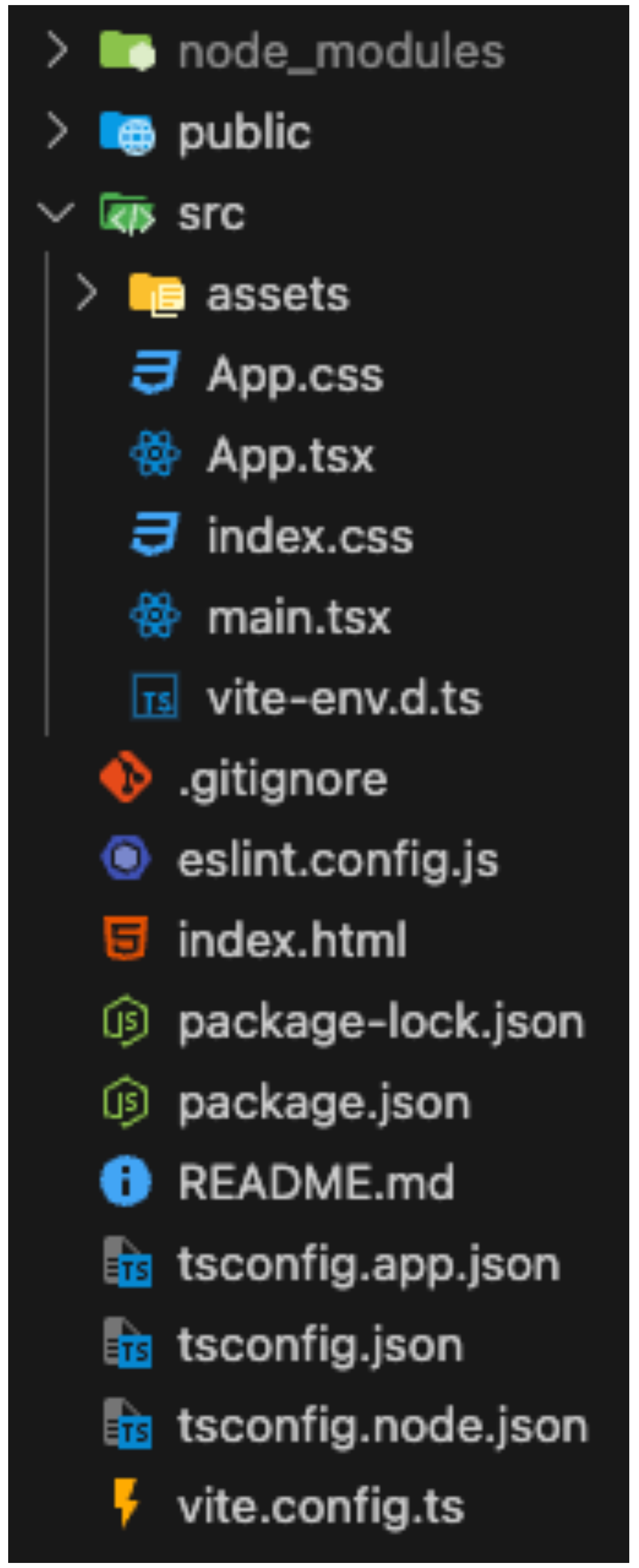
- We will use Vite to scaffold our project
- To create a project, run the following in the terminal:

```
npm create vite@latest my-react-app-name - -  - -template react-ts
```

- Then follow the prompts and instructions!
- Installing project dependencies: **npm install**

File structure

- /node_modules - all of the installed dependencies
- /public - root folder for storing static assets that won't be processed by Vite
- /src - source files and folders of App
  - App.tsx - root component of App
  - App.css - styles for root component
  - main.tsx - renders App
  - index.css - *global* styles
- index.html - entry point of App
- package.json - list of all dependencies and project metadata



```
> node_modules
> public
v src
  > assets
    App.css
    App.tsx
    index.css
    main.tsx
    vite-env.d.ts
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  README.md
  tsconfig.app.json
  tsconfig.json
  tsconfig.node.json
  vite.config.ts
```

MITHRIL

File structure

- React doesn't have opinions on how to put files into folders.
- There are few common approaches popular in the ecosystem:
  - Grouping by *features* or *routes* - all files related to specific feature - JS, CSS, test - are located in the same folder
  - Grouping by *file type* - group similar files together

```
common/
  Avatar.js
  Avatar.css
  APIUtils.js
  APIUtils.test.js
feed/
  index.js
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  FeedAPI.js
profile/
  index.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
  ProfileAPI.js
```

Grouping by feature or routes

```
api/
  APIUtils.js
  APIUtils.test.js
  ProfileAPI.js
  UserAPI.js
components/
  Avatar.js
  Avatar.css
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
```

Grouping by file type

Import and export

- React applications are basically a collection of interactive components
- If we want to create fully-fledged React app with a collection of components, we first need to know the way to use and reuse components that may have been defined elsewhere.
- Inside React app usually we import user-defined components or modules installed with *npm*.
- When we are importing user-defined components we are using relative path to the file. In case of importing npm module, then the name of the module is enough.

MITHRIL

Import and export

- Read more about import and export.

Exporting default export

```
export default function TodoList() {
```

```
import TodoList from "./components/TodoList";
```

Importing default export

Exporting named export

```
export function TodoList() {
```

```
import { TodoList } from "./components/TodoList";
```

Importing named export

Library installed inside node_modules

```
import { FormEvent, useState } from "react";
import { TodoItem } from "./TodoItem";
```

Relative path to the file

MITHRIL

JSX

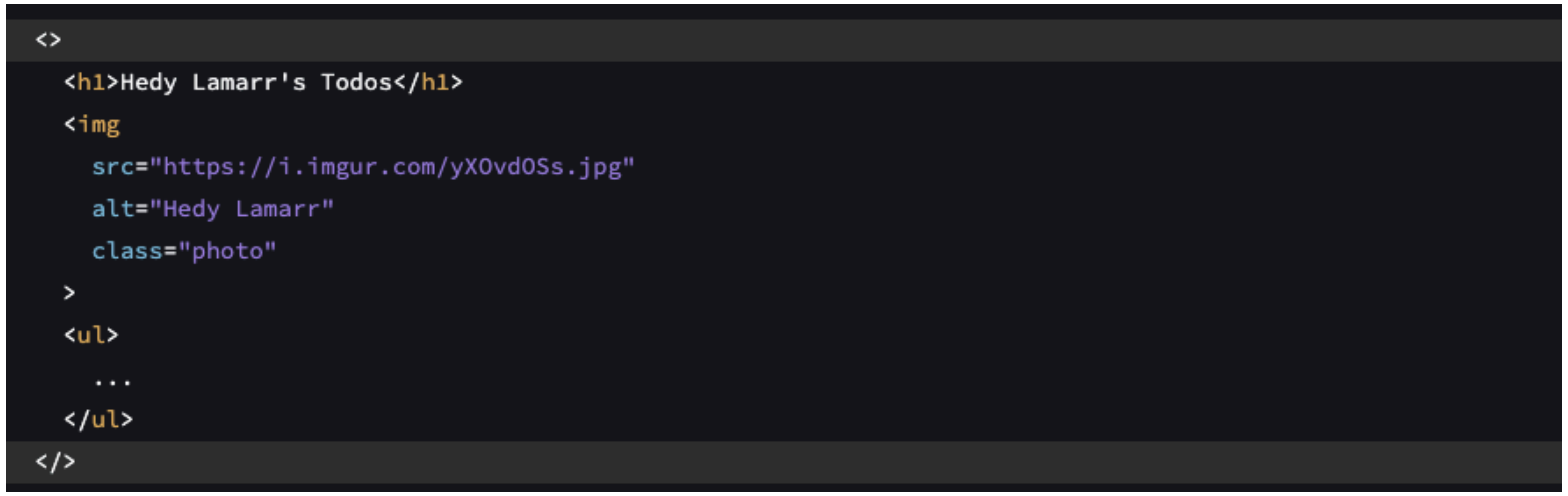
JSX

- JSX stands for JavaScript XML
- *JSX* is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file.
- Fundamentally, JSX just provides *syntactic sugar* for `React.createElement(component, props, ...children)` function.
- JSX is converted into JS with **esbuild** (*Vite* bundler and minifier).

MITHRIL

JSX

- We can define variable and use it inside JSX
- Any valid JS expression inside curly braces can be inside JSX
- Return a single root element - to return multiple elements from a component, wrap them with a single parent tag, like <div></div> or empty tag <></> called Fragment.



```
<>
  <h1>Hedy Lamarr's Todos</h1>
  <img
    src="https://i.imgur.com/yXOvdOSs.jpg"
    alt="Hedy Lamarr"
    class="photo"
  >
  <ul>
    ...
  </ul>
</>
```

MITHRIL

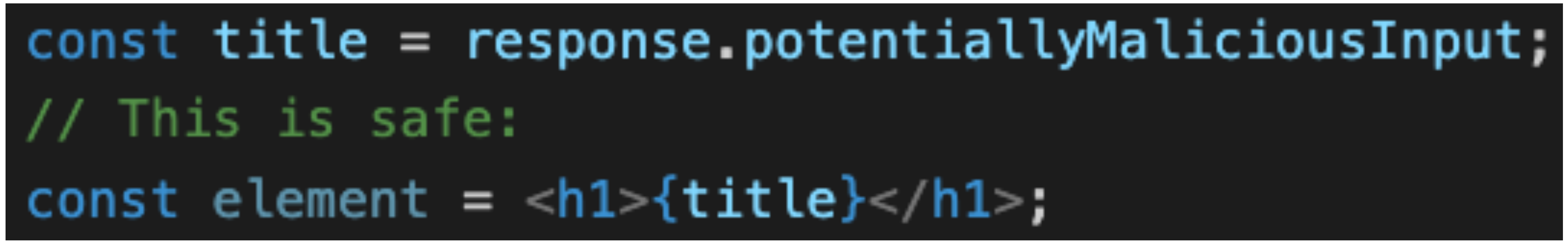
JSX

- Close all the tags - JSX requires tags to be explicitly closed
- Use *camelCase* naming convention for JSX
- Since class is a reserved word, in React you write className instead, named after the corresponding DOM property
- For historical reasons, *aria-** and *data-** attributes are written as in HTML, with dashes.

MITHRIL

JSX

- JSX prevents injection attacks.
- By default React DOM escapes any values embedded in JSX before rendering them.
- You can never inject anything that is not explicitly written in your application.
- Everything is converted to a string before being rendered.
- This helps prevent *XSS* (cross-site-scripting) attacks.



```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

Components

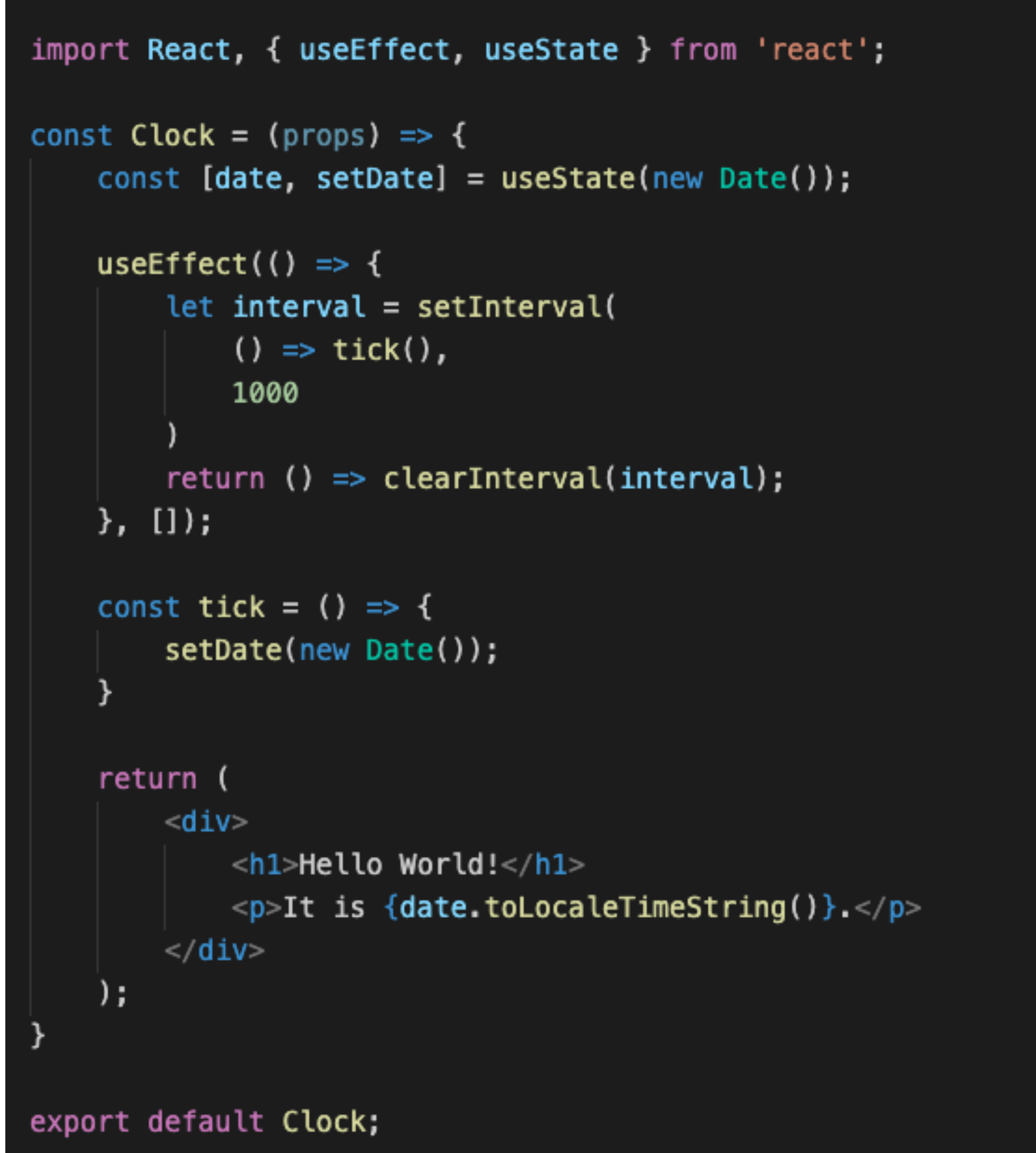
Components

- Component let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Conceptually, components are like JS functions.
- They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.
- React component can be:
  - Class
  - Functional

MITHRIL

Functional component

- Functional components are used in most cases
- Hooks are introduced in 16.8 version of React and they provide us more modular and concise way of handling and writing components
    - **useState()** hook is used to setup initial state of functional component
    - **useEffect()** hook is used to perform side effects, it fires after layout and paint



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

const Clock = (props) => {
    const [date, setDate] = useState(new Date());

    useEffect(() => {
        let interval = setInterval(
            () => tick(),
            1000
        )
        return () => clearInterval(interval);
    }, []);

    const tick = () => {
        setDate(new Date());
    }

    return (
        <div>
            <h1>Hello World!</h1>
            <p>It is {date.toLocaleTimeString()}.</p>
        </div>
    );
}

export default Clock;
```

Basic routing

Basic routing

- In traditional websites, the browser requests a document from a web server, downloads and evaluates CSS and JavaScript assets, and renders the HTML sent from the server. When the user clicks a link, it starts the process all over again for a new page.
- React Router enables "client side routing”.
- React Router, and dynamic, client-side routing, allows us to build a single-page web application with navigation without page refreshing as the user navigates.
- This enables faster user experiences because the browser doesn't need to request an entirely new document or re-evaluate CSS and JavaScript assets for the next page. It also enables more dynamic user experiences with things like animation.

Basic routing

- Install React Router: `npm install react-router-dom`
- Three main parts of React Router:
  - Browser Router - actual router component, highest parent in React app - stateful, top-level component that makes all the other components and hooks work.
  - Route - An object or Route Element typically with a shape of { path, element } or <Route path element>. The *path* is a path pattern. When the path pattern matches the current URL, the element will be rendered.
  - Link - primary means of navigation, similar to <a> tag but prevents page refresh, used to create navigation links
- Read more about React Router here.