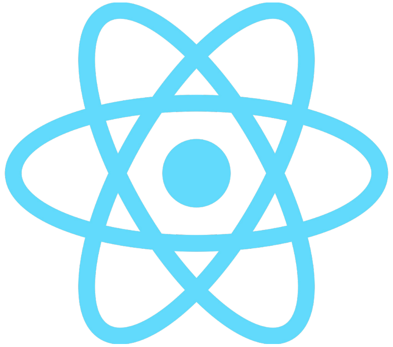


React

in practice

Outline

- **Setup**

  Editor, environment, new project, Create React App

- **Basic Concepts**

  JSX, Components, Props, Lists, Events, Hooks, State

- **Advanced Concepts**

  Context, Error Boundaries, Static Type Checking, Strict Mode

- **Structuring an application**

  Dependencies, UI, Routing, Redux, Some Noteworthy Packages

Editor Setup

- **Visual Studio Code**
- WebStorm
  Integrated Development Environment designed specifically for JavaScript
- Sublime Text
  Support for JSX and TypeScript, syntax highlighting and autocomplete built in
- Vim
  If you are self-harming

https://beta.reactjs.org/learn/editor-setup
https://code.visualstudio.com/

Environment Setup - Option 1: Manual

1. Download and install **Node.js**
2. Select a Package Manager:
    - **npm**
      Already installed in Node.js
    - **Yarn**
      Follow the instructions for the installation

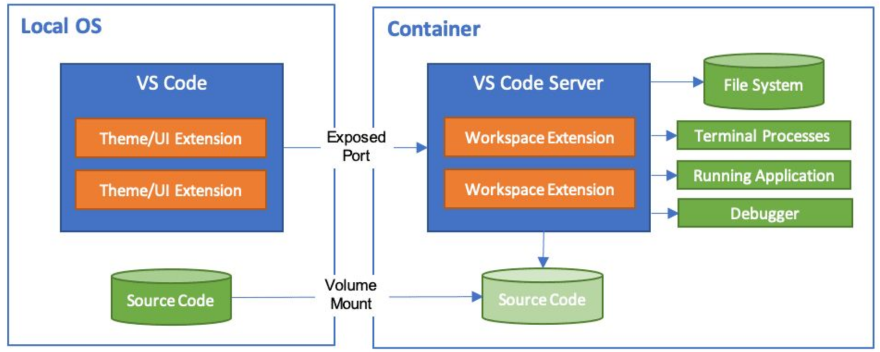
https://nodejs.org/en/download/

https://docs.npmjs.com/

https://yarnpkg.com/getting-started/install

Environment Setup - Option 2: Dev Container

1. Install and configure **Docker** for your operating system
   The suggested way is using **Docker Desktop**, additional steps may be necessary (e.g., enable WSL 2 on Windows)
2. Install **Visual Studio Code**
3. Install the **Remote Development extension pack**



https://code.visualstudio.com/docs/devcontainers/containers

https://docs.docker.com/get-docker/

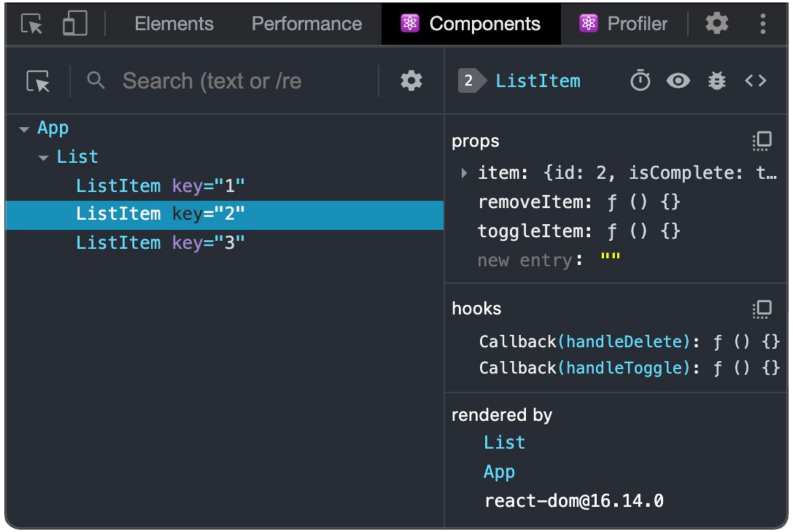
https://aka.ms/vscode-remote/download/extension

React Developer Tools

Use **React Developer Tools** to inspect React components, edit props and state, and identify performance problems.

The Components tab shows you the root React components that were rendered on the page, as well as the subcomponents that they ended up rendering.

By selecting one of the components in the tree, you can **inspect and edit its current props and state** in the panel on the right. In the breadcrumbs you can inspect the selected component, the component that created it, the component that created that one, and so on.



```
▾ App
  ▾ List
      ListItem key="1"
      ListItem key="2"
      ListItem key="3"
```

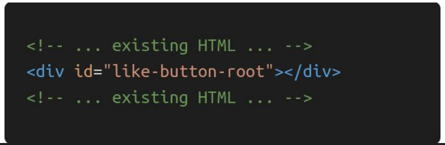
```
props
 ▸ item: {id: 2, isComplete: t…
   removeItem: ƒ () {}
   toggleItem: ƒ () {}
   new entry: ""
hooks
   Callback(handleDelete): ƒ () {}
   Callback(handleToggle): ƒ () {}
rendered by
   List
   App
   react-dom@16.14.0
```

https://beta.reactjs.org/learn/react-developer-tools

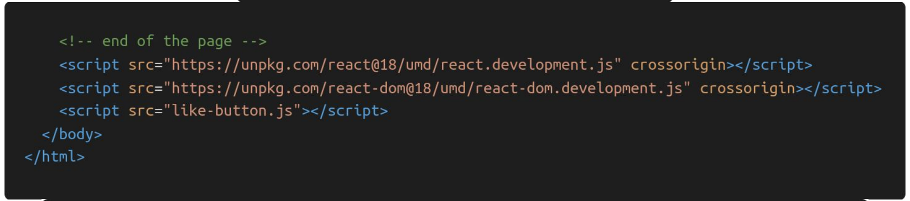
https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en

New Project - Option 1: Add React to a Website

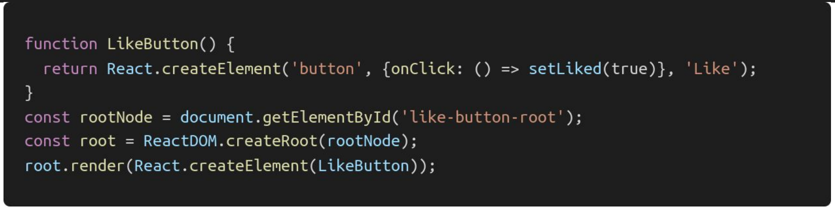
1. Add a root HTML tag
2. Add the script tags
3. Create a React component
4. Add your React component to the page
5. Minify JavaScript for production



```html
<!-- ... existing HTML ... -->
<div id="like-button-root"></div>
<!-- ... existing HTML ... -->
```



```html
    <!-- end of the page -->
    <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
    <script src="like-button.js"></script>
  </body>
</html>
```



```javascript
function LikeButton() {
  return React.createElement('button', {onClick: () => setLiked(true)}, 'Like');
}
const rootNode = document.getElementById('like-button-root');
const root = ReactDOM.createRoot(rootNode);
root.render(React.createElement(LikeButton));
```

If you want to use JSX syntax, additional steps are required…

https://beta.reactjs.org/learn/add-react-to-a-website

New Project - Option 2: Use a Toolchain

Create React App is an officially supported (and the most popular) way to create single-page, client-side React applications. It offers a modern build setup with no configuration.

```
npx create-react-app my-app  # npm 5.2+

npm init react-app my-app  # npm 6+

yarn create react-app my-app  # Yarn 0.25+
```

Popular alternatives:

- Vite (https://vitejs.dev/guide/)
- Parcel (https://parceljs.org/getting-started/webapp/)

https://beta.reactjs.org/learn/start-a-new-react-project#getting-started-with-a-minimal-toolchain

New Project - Option 3: Full-featured Framework

If you're looking to start a production-ready project, **Next.js** is a great place to start.
Next.js is a popular, lightweight framework for static and server-rendered applications built with React.
It comes pre-packaged with features like routing, styling, and server-side rendering, getting your project up and running quickly.

Popular alternatives:

- Gatsby (https://www.gatsbyjs.org/)
- Remix (https://remix.run/)
- Razzle (https://razzlejs.org/)

https://beta.reactjs.org/learn/start-a-new-react-project#building-with-a-full-featured-framework

https://nextjs.org/

New Project - Option 4: Custom Toolchain

You may prefer to create and configure your own toolchain. A toolchain typically consists of:

- A **package manager** lets you install, update, and manage third-party packages. Popular package managers: *npm* (built into Node.js), *Yarn*, *pnpm*.
- A **compiler** lets you compile modern language features and additional syntax like JSX or type annotations for the browsers. Popular compilers: *Babel*, *TypeScript*, *swc*.
- A **bundler** lets you write modular code and bundle it together into small packages to optimize load time. Popular bundlers: *webpack*, *Parcel*, *esbuild*, *swc*.
- A **minifier** makes your code more compact so that it loads faster. Popular minifiers: *Terser*, *swc*.
- A **server** handles server requests so that you can render components to HTML. Popular servers: *Express*.
- A **linter** checks your code for common mistakes. Popular linters: *ESLint*.
- A **test runner** lets you run tests against your code. Popular test runners: *Jest*.

But you can also set up your own JavaScript toolchain from scratch...

https://beta.reactjs.org/learn/start-a-new-react-project#custom-toolchains

Create React App: Selecting a Template

You can optionally start a new app from a template by appending `--template [template-name]` to the creation command.

```
npx create-react-app my-app --template [template-name]
```

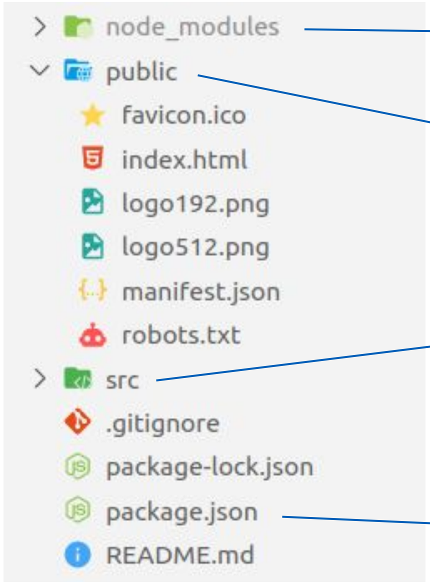
You can find a list of available templates by searching for "cra-template-*" on npm.

Example: Creating a TypeScript App

```
npx create-react-app my-app --template typescript
```

https://create-react-app.dev/docs/getting-started/#selecting-a-template

Create React App: Folder Structure



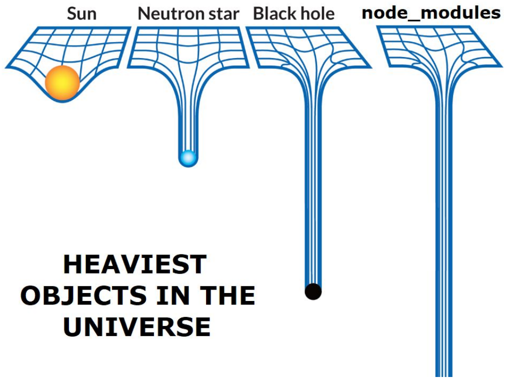
```
> node_modules
v public
    favicon.ico
    index.html
    logo192.png
    logo512.png
    manifest.json
    robots.txt
> src
  .gitignore
  package-lock.json
  package.json
  README.md
```

node_modules → The folder where npm (or Yarn) installs all the **dependencies**

public → **Root** folder that gets served up as your React app

src → Contains all the **sources** of the React app (JavaScript/TypeScript and CSS files)

package.json → **Project Metadata**



https://create-react-app.dev/docs/folder-structure

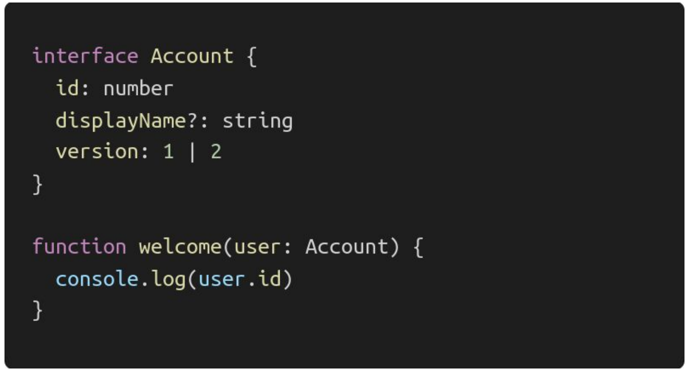
Create React App: Available Scripts

- `npm start`
  Runs the app in the development mode. Open `http://localhost:3000` to view it in the browser. The page will reload if you make edits. You will also see any lint errors in the console.
- `npm test`
  Launches the tests implemented in the project.
- `npm run build`
  Builds the app for production to the build folder. It correctly bundles React in production mode and optimizes the build for the best performance.
- `npm run eject`
  You don't have to ever use this command.

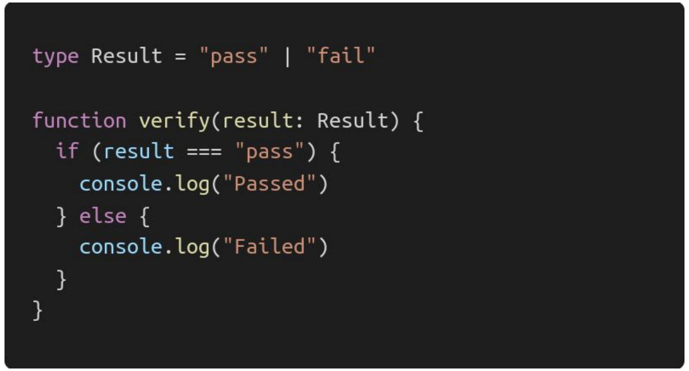
https://create-react-app.dev/docs/available-scripts

TypeScript

“TypeScript is **JavaScript with syntax for types**”



```
interface Account {
  id: number
  displayName?: string
  version: 1 | 2
}

function welcome(user: Account) {
  console.log(user.id)
}
```



```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

The JavaScript language: https://javascript.info/js

JavaScript Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript

The TypeScript Handbook: https://www.typescriptlang.org/docs/handbook/intro.html

Components

React apps are made out of **components**. A component is a piece of the UI (user interface) that has its own logic and appearance. A component can be as small as a button, or as large as an entire page.

```
const MyButton = () => <button>I'm a button</button>
export default MyButton
```

React component names must always **start with a capital letter**, while HTML tags must be lowercase.

A component must be **pure**, meaning: it should not change any objects or variables that existed before rendering and, given the same inputs, a component should always return the same JSX.

https://beta.reactjs.org/learn#components

Arrow Function Expressions

```
(param) => expression  // The parentheses are optional with one single parameter
(param_1, ..., param_N) => expression
```

An **arrow function expression** is a compact alternative to a traditional function expression, but is limited and can't be used in all situations.

```
const func = (function (a) {
  return a + 100;
});
```

```
const func = a => a + 100;
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Props

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “**props**”) and return React elements describing what should appear on the screen.

```
type UserThumbnailProps = {
  img: string,
  url: string,
}

export const UserThumbnail = (props: UserThumbnailProps) => (
  <a href={props.url}><img src={props.img} /></a>
)
```

Props are read-only: all React components must act like pure functions with respect to their props.

https://beta.reactjs.org/learn#components

Conditional Rendering

```
let content;
if (isLoggedIn) {
  content = <AdminPanel />;
} else {
  content = <LoginForm />;
}
return (
  <div>
    {content}
  </div>
);
```

```
<div>
  {isLoggedIn ? (
    <AdminPanel />
  ) : (
    <LoginForm />
  )}
</div>
```

```
<div>
  {isLoggedIn && <AdminPanel />}
</div>
```

https://beta.reactjs.org/learn#conditional-rendering

List and Keys

```
const products = [
  { title: 'Cabbage', id: 1 },
  { title: 'Garlic', id: 2 },
  { title: 'Apple', id: 3 },
];

export const ShpoppingList = () => (
  <ul>
    {products.map(p => <li key={p.id}>{p.title}</li>)}
  </ul>
)
```

Keys help React identify which items have changed, are added, or are removed.
Keys should be given to the elements inside the array to give the elements a **stable identity**.

https://beta.reactjs.org/learn#rendering-lists

Handling Events

You can respond to events by declaring **event handler** functions inside your components:

```
const MyButton = () => {
    const handleClick = () => {
        alert("You have clicked the button")
    }
    return <button onClick={handleClick}>Click Here!</button>
}
```

Do not call the event handler function: you only need to **pass it down**.

https://beta.reactjs.org/learn/responding-to-events

Hooks and Function Components

Hooks (React ≥ 16.8) let you use React features without writing a class.

Rules of Hooks:

1. Only call Hooks **at the top level**. Don't call Hooks inside loops, conditions, or nested functions.
2. Only call Hooks **from React function components** (and custom hooks). Don't call Hooks from regular JavaScript functions.

In JavaScript, all functions work like closures. A **closure** is a function, which uses the scope in which it was declared when invoked (not the scope in which it was invoked).

"With hooks, beginners no longer need to learn about 'this' to avoid shooting themselves in the foot."

Closures:

https://reactjs.org/docs/hooks-intro.html

Hooks

- **useState**
- **useEffect**
- **useContext**
- useReducer
- useCallback
- useMemo
- useRef
- ...

```
const [state, setState] = useState(initialState);
```

```
useEffect(() => {
  // This run after the render is committed to the screen
  return () => {
    // This run when the component leaves the screen
  }
}, [
  // Array of values that the effect depends on
  // Leave it empty if you want to run the effect only once
])
```

https://reactjs.org/docs/hooks-reference.html

Managing the State

```
export const MyButton = () => {
  const [count, setCount] = useState(0)

  function handleClick() {
    setCount(prev => prev + 1)
  }
  return <button onClick={handleClick}>Clicked {count} times</button>
}
```

- Do not modify the state **directly**
- State updates may be **asynchronous**
- State updates are **merged**

https://reactjs.org/docs/state-and-lifecycle.html

Props vs State

Two types of "model" data in React:

- **Props are like function arguments.**
  They let a parent component pass data to a child component and customize its appearance.
- **State is like a component's memory**.
  It lets a component keep track of some information and change it in response to interactions.

A parent component will often keep some information in state (so that it can change it), and *pass it down* to child components as their props.

	Props	**State**
Can get initial value from parent Component?	Yes	Yes
Can be changed by parent Component?	Yes	No
Can set default values inside Component?	Yes	Yes
Can change inside Component?	No	Yes
Can set initial value for child Components?	Yes	Yes
Can change in child Components?	Yes	No

https://beta.reactjs.org/learn/thinking-in-react#step-4-identify-where-your-state-should-live

Context

Context provides a way to **pass data through the component tree** without having to pass props down manually at every level.

```
const MyContext = React.createContext(defaultValue)

const App = () => (
  <MyContext.Provider value={/* some value */}>
    {/* ...child components... */}
  </MyContext.Provider>
)
```

```
const MyComponent = () => {
  const value = useContext(MyContext)

  return <span>The value of MyContext is {value}</span>
}
```

Use cases: theming, current account, routing, managing state, …

https://reactjs.org/docs/context.html

useEffect

The *Effect Hook* lets you perform **side effects** (e.g., data fetching, setting up a subscription, and manually changing the DOM) in function components.

```
useEffect(
  () => {
    // Execute the side effects
    return () => {
      // Clean up the previous effect before executing the next effect
    }
  }, [
    // Fire the effect only when the values listed here have changed
    // If empty, the effect is run and clean up only once (on mount and unmount)
  ]
)
```

https://reactjs.org/docs/hooks-effect.html

useRef

Essentially, `useRef` is like a “box” that can hold a mutable value in its `.current` property.

```
const Form = () => {
    const [name, setName] = useState('')
    const handleSubmit: React.FormEventHandler = e => {
        e.preventDefault()
        alert("Your name is " + name)
    }
    return (
        <form onSubmit={handleSubmit}>
            <input onChange={e => setName(e.target.value)} value={name} />
            <button type="submit">Submit</button>
        </form>
    )
}
```

```
const Form = () => {
    const ref = useRef<HTMLInputElement>(null)
    const handleSubmit: React.FormEventHandler = e => {
        e.preventDefault()
        alert("Your name is " + ref.current?.value)
    }
    return (
        <form onSubmit={handleSubmit}>
            <input ref={ref} />
            <button type="submit">Submit</button>
        </form>
    )
}
```

Controlled Component ***vs*** **Uncontrolled Component**

https://reactjs.org/docs/hooks-reference.html#useref
https://goshacmd.com/controlled-vs-uncontrolled-inputs-react/

Error Boundaries

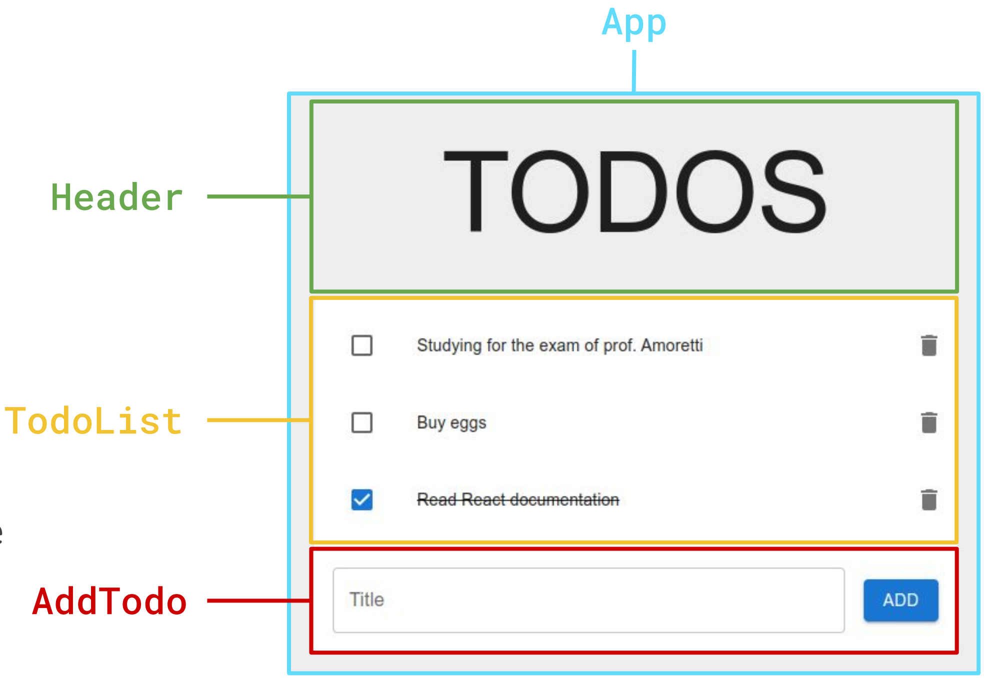
Error boundaries (React ≥ 16) are components that **catch JavaScript errors anywhere in their child component tree**, log those errors, and display a fallback UI instead of the component tree that crashed.

```
type ErrorBoundaryProps = { children?: ReactNode }
type ErrorBoundaryState = { hasError: boolean }

class ErrorBoundary extends Component<ErrorBoundaryProps, ErrorBoundaryState> {
  public state: State = { hasError: false }

  public static getDerivedStateFromError(_: Error): State {
    return { hasError: true } // Update state so the next render will show the fallback UI.
  }
  public componentDidCatch(error: Error, errorInfo: ErrorInfo) {
    console.error("Uncaught error:", error, errorInfo);
  }
  public render() {
    return this.state.hasError ? (<h1>Sorry.. there was an error</h1>) : this.props.children
  }
}
```

https://reactjs.org/docs/error-boundaries.html

Structuring an application

1. Break the UI into a **component hierarchy**
2. Build a **static version** in React
3. Find the minimal but complete representation of UI state
4. Identify where your **state** should live
5. Add **inverse data flow**


App
Header
TodoList
AddTodo
TODOS
Studying for the exam of prof. Amoretti
Buy eggs
~~Read React documentation~~
Title
ADD

https://beta.reactjs.org/learn/thinking-in-react

Install a dependency

You can install a package using:

```
npm install <package>
# Alternatively you may use yarn:
yarn add <package>
```

Your package manager will update the lock and `package.json` files accordingly.

The npm registry is a **public collection of packages** of open-source code for Node.js, front-end web apps and the JavaScript community at large.

https://docs.npmjs.com/cli/v8/commands/npm-install

https://www.npmjs.com/

Material UI

Material UI is a library of React UI components that implements Google's **Material Design**.

It includes a comprehensive **collection of prebuilt components** that are ready for use in production right out of the box.

```
npm install @mui/material @emotion/react @emotion/styled
# Robot Font
npm install @fontsource/roboto
# Material Icons
npm install @mui/icons-material
```

https://mui.com/material-ui/getting-started/installation/

https://m3.material.io/

React Router

Lightweight, fully-featured **routing** library.

Some features:

- **Client Side Routing**
- Nested Routes
- Dynamic Segments
- Ranked Route Matching
- Active Links
- Relative Links
- Data Loading
- Redirects
- ...

```
const App = () => (
  <Routes>
    <Route path="/" element={<Layout />}>
      <Route index element={<Home />} />
      <Route path="about" element={<About />} />
      <Route path="*" element={<NoMatch />} />
    </Route>
  </Routes>
)

const Menu = () => (
  <nav>
    <Link to="/">Home</Link>
    <Link to="/about">About</Link>
  </nav>
)
```

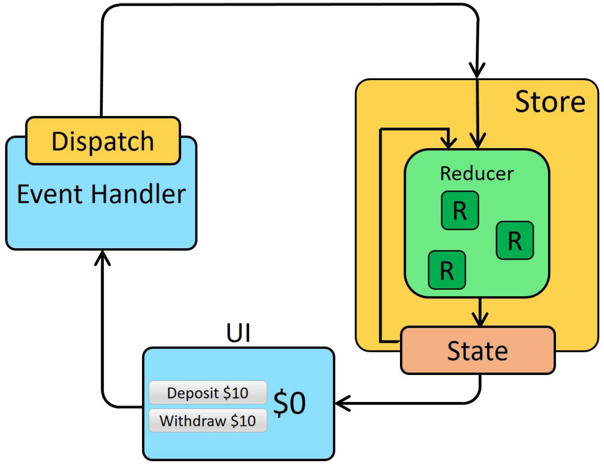
https://reactrouter.com/

Redux Toolkit

Redux is a pattern and library for managing and updating **application state**, using events called "**actions**".

It serves as a centralized store for a **global state** (used across your entire application), with rules ensuring that the state can only be updated in a predictable fashion.

The patterns and tools provided by Redux make it easier to understand **when, where, why, and how the state in your application is being updated**, and how your application logic will behave when those changes occur.



https://redux-toolkit.js.org/

React Hook Form + Yup

```
import { useForm } from 'react-hook-form'

export const App = () => {
  const { register, handleSubmit, formState: { errors } } = useForm()
  const onSubmit = data => console.log(data)
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input type="text" placeholder="Name" {...register("Name", {required: true, maxLength: 80})} />
      <input type="email" placeholder="Email" {...register("Email", {required: true, pattern: /^\S+@\S+$/i})} />
      <input type="submit" />
    </form>
  )
}
```

https://react-hook-form.com/

https://github.com/jquense/yup

react-i18next

Internationalization framework for React which is based on i18next.

```
import i18n from "i18next"
import { useTranslation, initReactI18next } from "react-i18next"

i18n
  .use(initReactI18next) // passes i18n down to react-i18next
  .init({
    // [...]
  })

export const App = () => {
  const { t } = useTranslation()
  return <h2>{t('Welcome to React')}</h2>
}
```

https://react.i18next.com/

Other noteworthy packages

- **Axios** - https://axios-http.com/
  Promise based HTTP client for the browser and node.js
- **Lodash** - https://lodash.com/
  A modern JavaScript utility library delivering modularity, performance & extras
- **Luxon** - https://moment.github.io/luxon/
  Library for dealing with dates and times in JavaScript
- **React Draggable** - https://github.com/react-grid-layout/react-draggable
  A simple component for making elements draggable
- **React Motion** - https://github.com/chenglou/react-motion
  Popular animation library

Good coding!

You can find these slides and the code of the sample application on the following GitHub repository:

https://github.com/franksacco/react-in-practice

For doubts or further information, please contact:

- Francesco Saccani - francesco.saccani@unipr.it
- Gabriele Penzotti - gabriele.penzotti@unipr.it