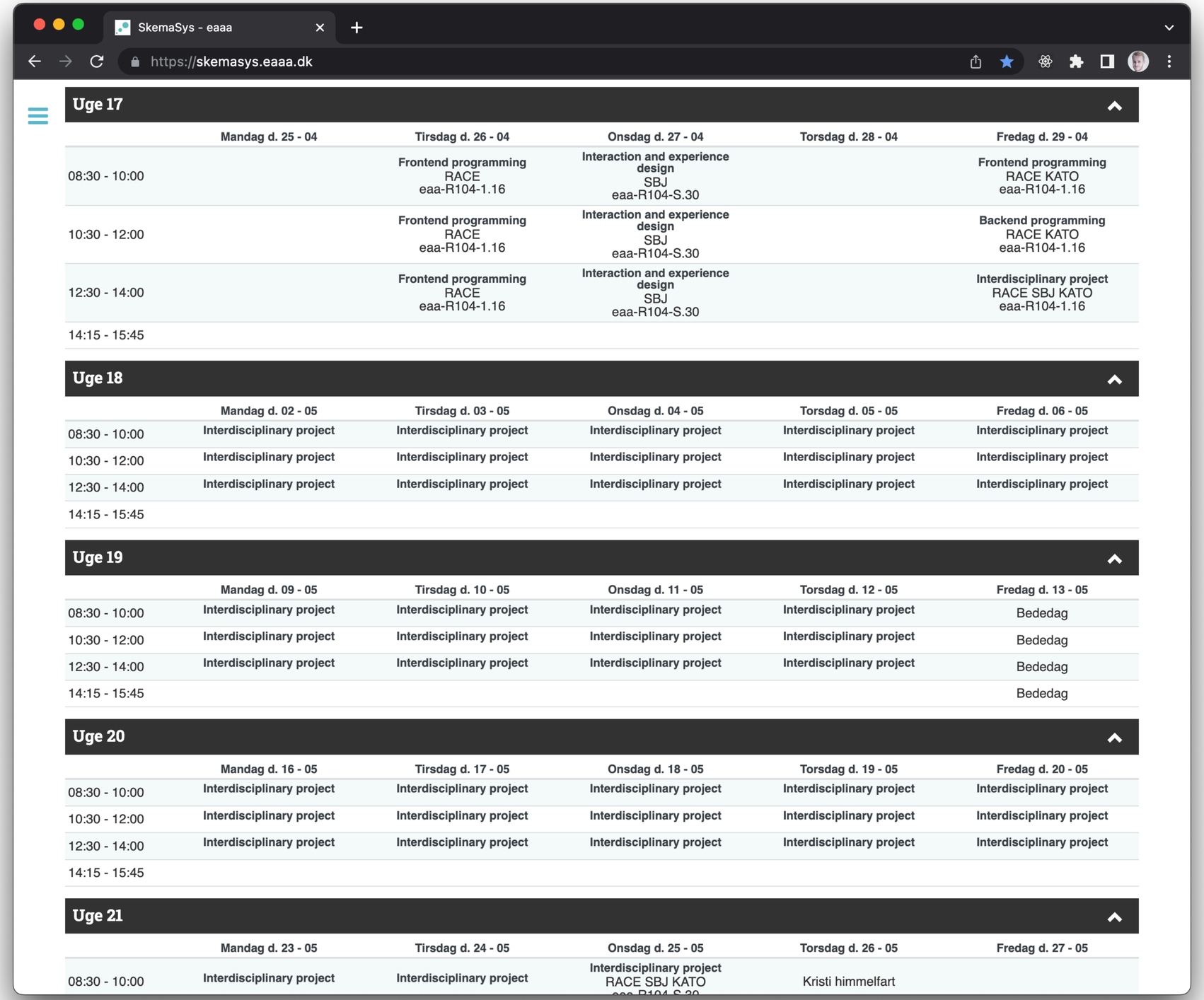
React CRUD App

Frontend Programming



Uge 17

	Mandag d. 25 - 04	Tirsdag d. 26 - 04	Onsdag d. 27 - 04	Torsdag d. 28 - 04	Fredag d. 29 - 04
08:30 - 10:00		**Frontend programming** RACE eaa-R104-1.16	**Interaction and experience design** SBJ eaa-R104-S.30		**Frontend programming** RACE KATO eaa-R104-1.16
10:30 - 12:00		**Frontend programming** RACE eaa-R104-1.16	**Interaction and experience design** SBJ eaa-R104-S.30		**Backend programming** RACE KATO eaa-R104-1.16
12:30 - 14:00		**Frontend programming** RACE eaa-R104-1.16	**Interaction and experience design** SBJ eaa-R104-S.30		**Interdisciplinary project** RACE SBJ KATO eaa-R104-1.16
14:15 - 15:45					

Uge 18

	Mandag d. 02 - 05	Tirsdag d. 03 - 05	Onsdag d. 04 - 05	Torsdag d. 05 - 05	Fredag d. 06 - 05
08:30 - 10:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**
10:30 - 12:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**
12:30 - 14:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**
14:15 - 15:45					

Uge 19

	Mandag d. 09 - 05	Tirsdag d. 10 - 05	Onsdag d. 11 - 05	Torsdag d. 12 - 05	Fredag d. 13 - 05
08:30 - 10:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	Bededag
10:30 - 12:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	Bededag
12:30 - 14:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	Bededag
14:15 - 15:45					Bededag

Uge 20

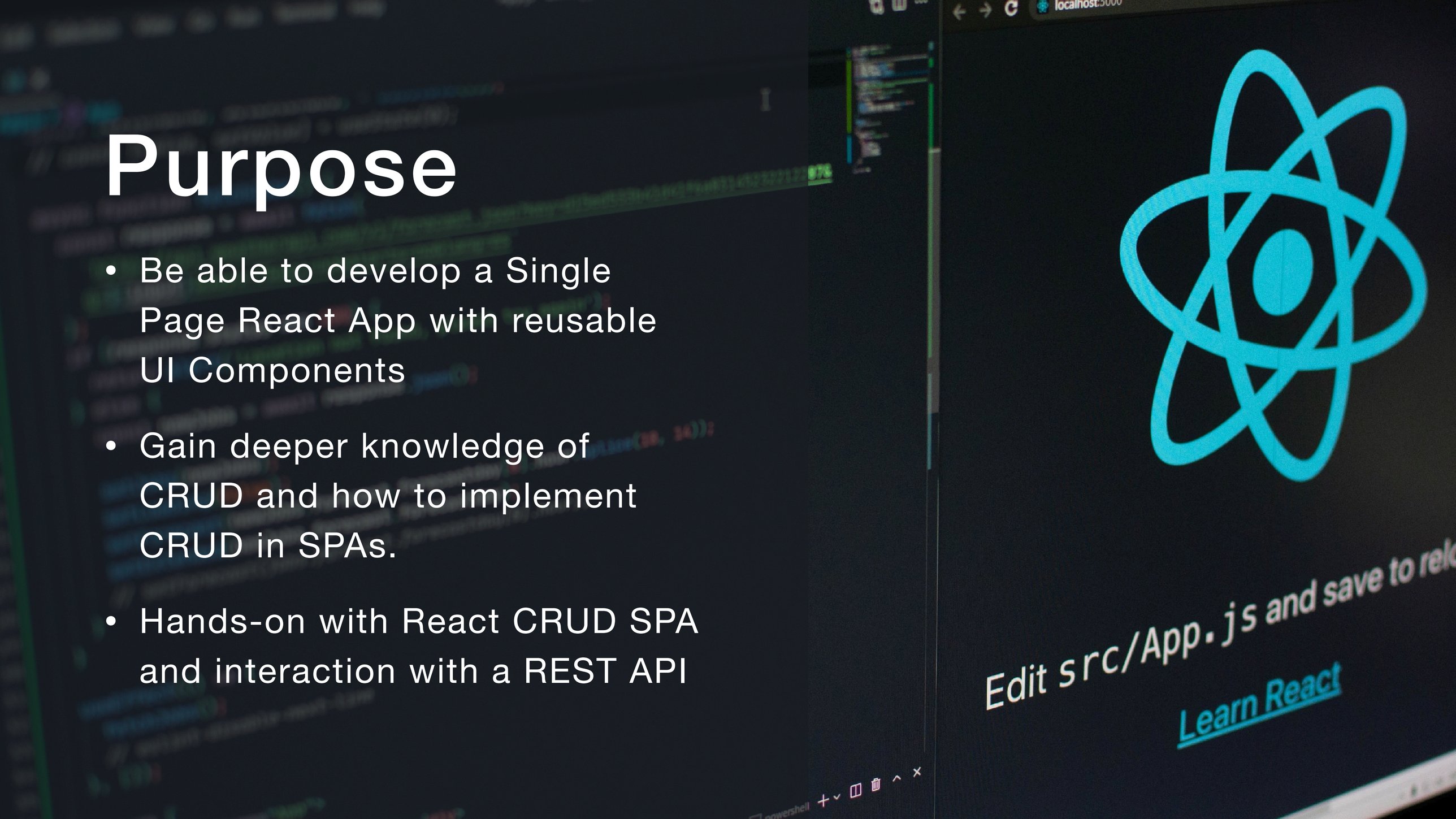
	Mandag d. 16 - 05	Tirsdag d. 17 - 05	Onsdag d. 18 - 05	Torsdag d. 19 - 05	Fredag d. 20 - 05
08:30 - 10:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**
10:30 - 12:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**
12:30 - 14:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project**
14:15 - 15:45					

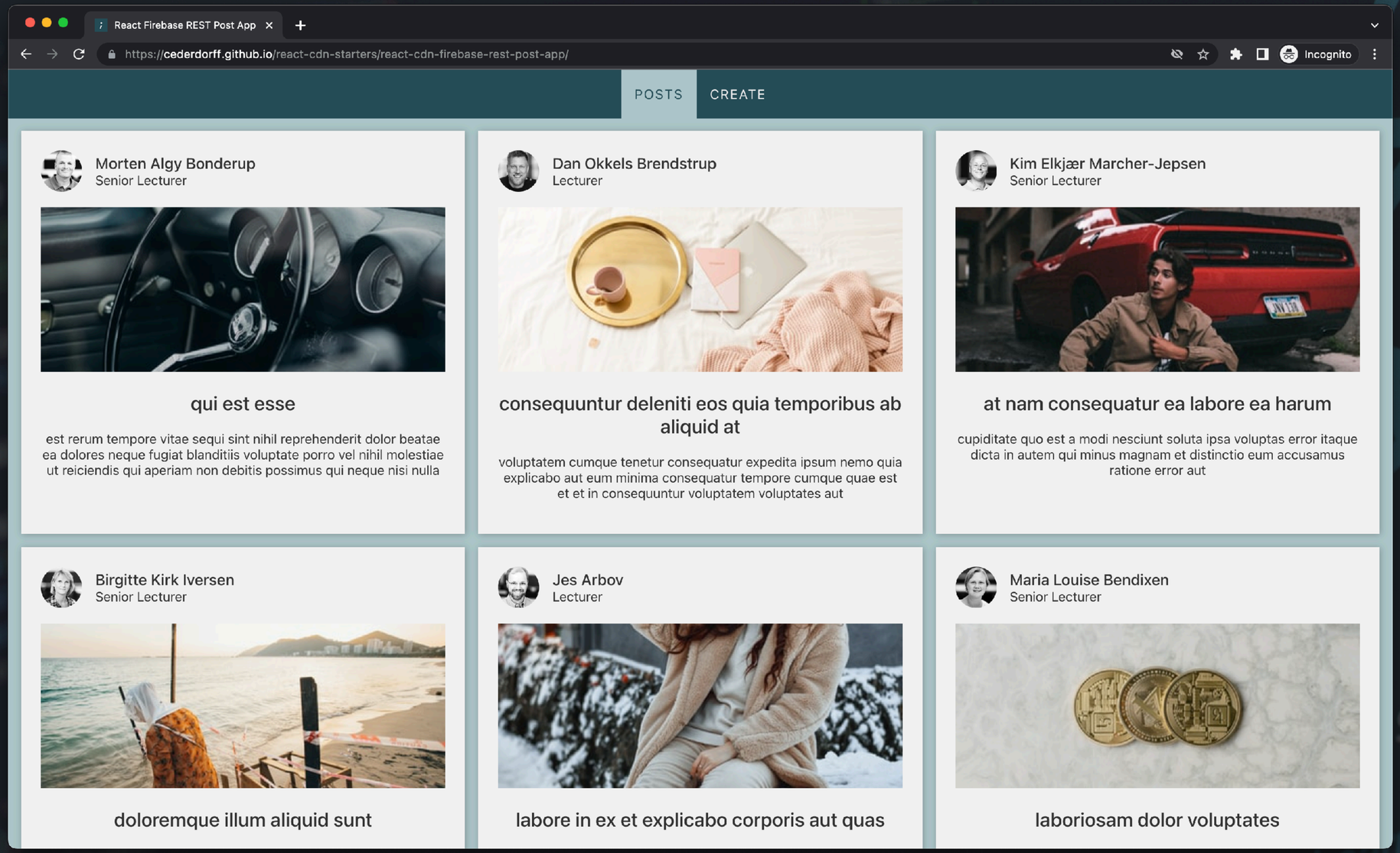
Uge 21

	Mandag d. 23 - 05	Tirsdag d. 24 - 05	Onsdag d. 25 - 05	Torsdag d. 26 - 05	Fredag d. 27 - 05
08:30 - 10:00	**Interdisciplinary project**	**Interdisciplinary project**	**Interdisciplinary project** RACE SBJ KATO	Kristi himmelfart	

Purpose

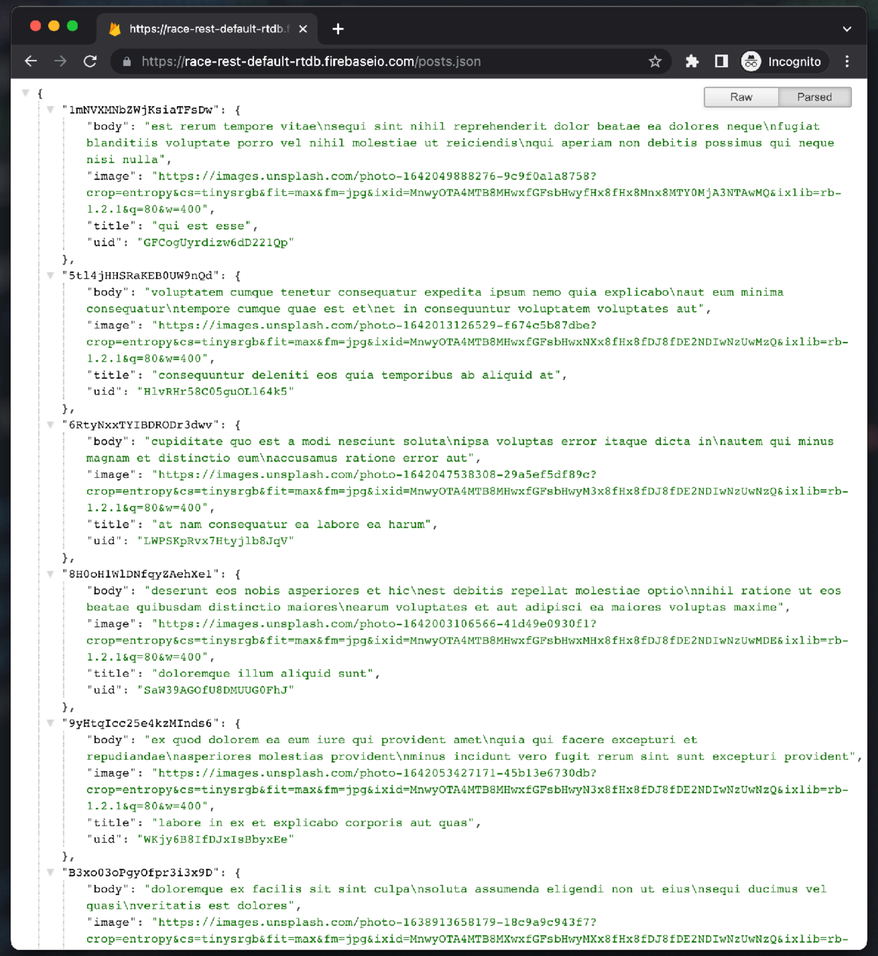
- Be able to develop a Single Page React App with reusable UI Components
- Gain deeper knowledge of CRUD and how to implement CRUD in SPAs.
- Hands-on with React CRUD SPA and interaction with a REST API



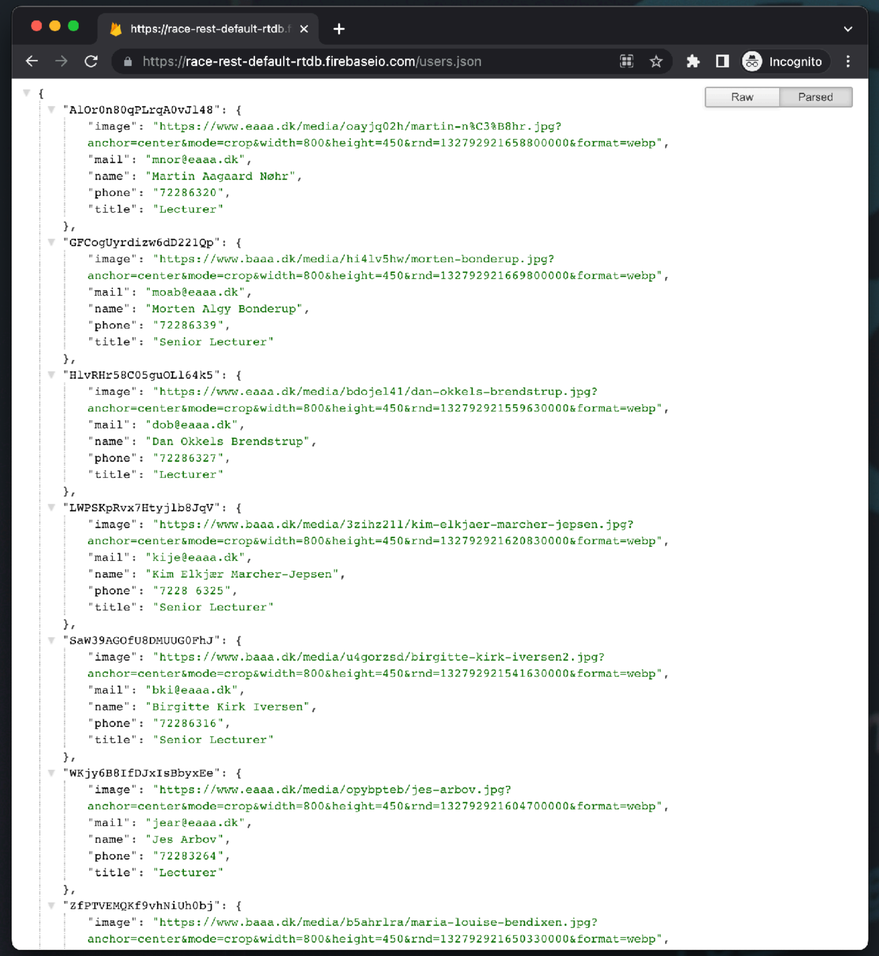


cederdorff.github.io/react-cdn-starters/react-cdn-firebase-rest-post-app/

Two data sources



race-rest-default-rtdb.firebaseio.com/posts.json



race-rest-default-rtdb.firebaseio.com/users.json

Agenda

- Introduction
  - CRUD, REST & HTTP Verbs
  - Build tools & create-react-app
  - SPA & React Router
- Build React CRUD App
  - READ —> GET
  - CREATE —> POST
  - UPDATE —> PUT
  - DELETE —> DELETE

Web Development

WEB APP

API

FETCH(...)

REQUEST

JSON DATA

RESPONSE

JSON

DATA SOURCE

Interdisciplinary project

FRONTEND

(API)

BACKEND

FETCH(...)

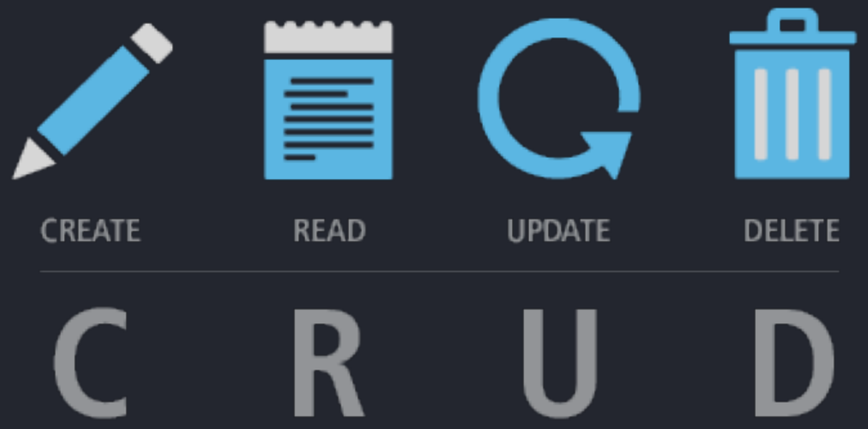
REQUEST

JSON DATA

RESPONSE

Fetch: callback (`then`) vs `async/await`

```
// fetch with callbacks
fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json")
    .then(function (response) {
        return response.json();
    })
    .then(function (data) {
        console.log(data);
    });
//
//
// or with async/await
const response = await fetch("https://cederdorff.github.io/web-frontend/canvas-users/data.json");
const data = await response.json();
console.log(data);
```



What's CRUD?

- CREATE objects like a post, user, movie, product, etc.
- READ objects like an array (or object) of objects (posts, users, movies, products, etc)
- UPDATE an object, often given by a unique id.
- DELETE an object, often given by a unique id.

codecademy.com/article/what-is-crud

GET

POST

PUT

DELETE

What’s REST?

- REpresentational State Transfer
- A standard for systems (client & server) to communicate over HTTP in order to retrieve or modify (data) resources.
- Stateless, meaning the two systems doesn’t need to know anything about the state.
- The client makes the requests using the 4 basic HTTP verbs to define the opreation.

codecademy.com/article/what-is-rest

HTTP REQUEST METHODS (verbs)

GET - POST - PUT - DELETE

HTTP (Hypertext Transfer Protocol) is the standard way to communicate between clients and servers (request-response protocol).

"HTTP defines a set of **request methods** to indicate the desired action to be performed for a given resource."

https://www.w3schools.com/tags/ref_httpmethods.asp
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

CRUD vs REST & HTTP Verbs

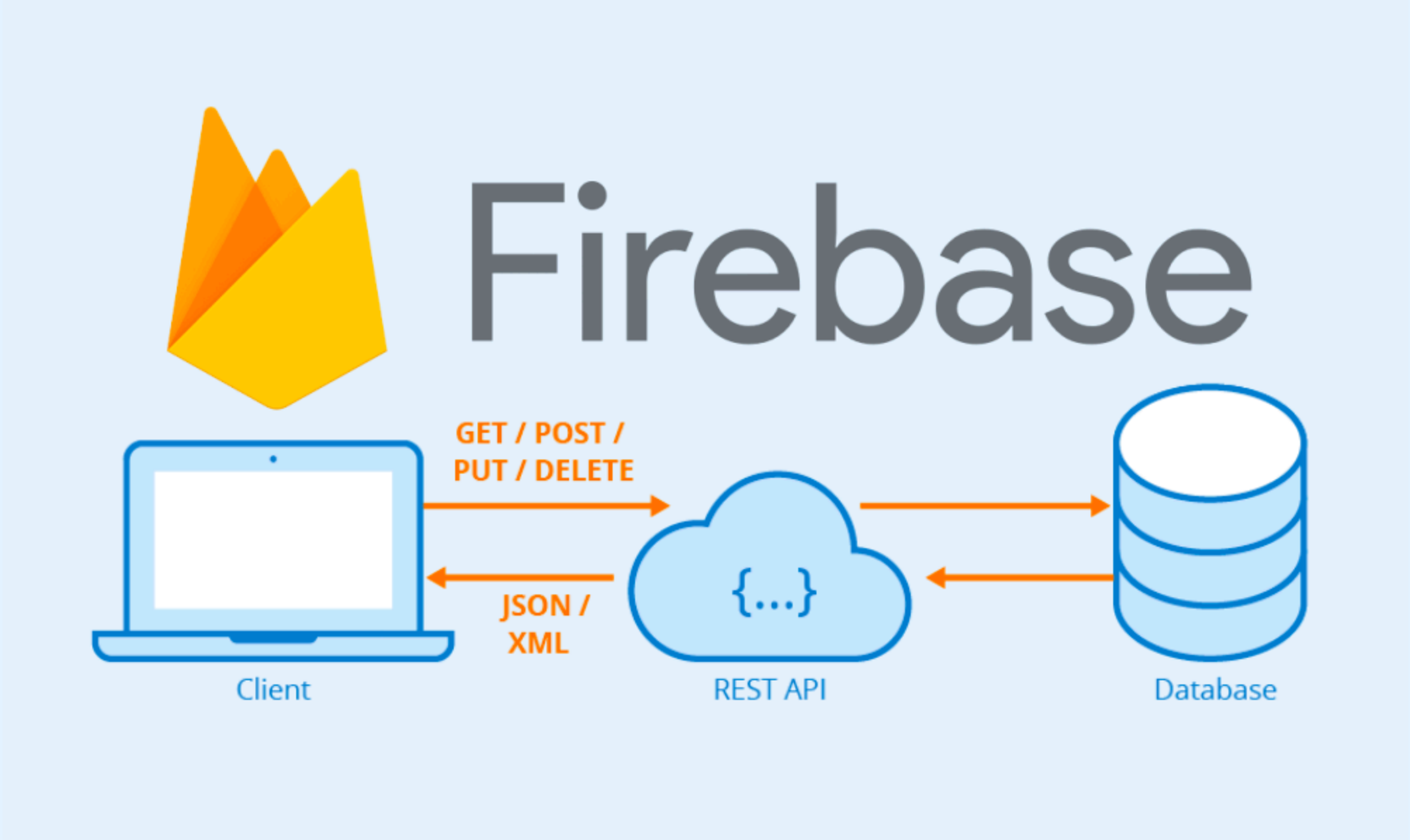
CREATE —> `POST`: create a new resource (object)

READ —> `GET`: retrieve a specific resource or a collection

UPDATE —> `PUT`: update a specific resource (by id)

DELETE —> `DELETE`: remove a specific resource by id

codecademy.com/article/what-is-rest



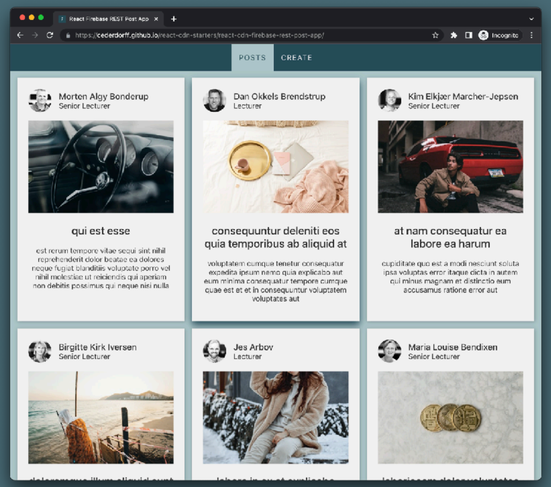
javascript.plainenglish.io/firebase-cloud-functions-tutorial-creating-a-rest-api-8cbc51479f80

Web Development

WEB APP

FIREBASE REST API

DATA SOURCE



FETCH(...)

REQUEST

JSON DATA

RESPONSE

JSON

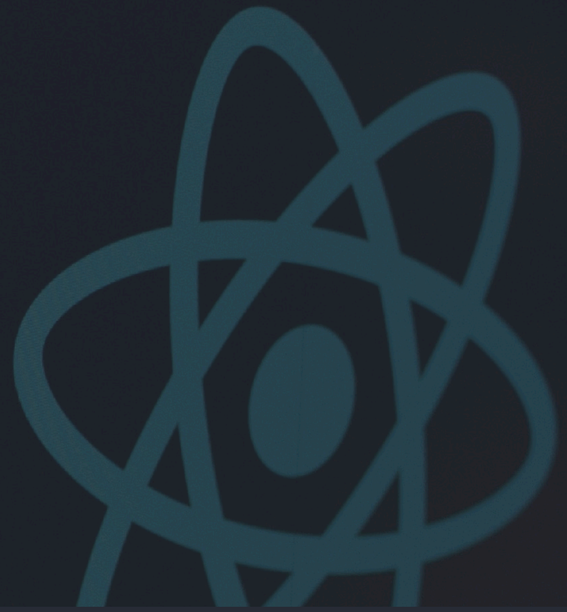
Firebase
Realtime Database

posts

users

Fetch and read posts

`GET` is the default request method for fetch.



```
async function getPosts() {
    const url = "https://race-rest-default-rtdb.firebaseio.com/posts.json";
    const response = await fetch(url);
    const data = await response.json();
    const postsArray = Object.keys(data).map(key => ({ id: key, ...data[key] })); // from object to array
    return postsArray
}
```

REQUEST headers

“[…] contain more information about the resource to be fetched, or about the client requesting the resource.”

"A request header is an HTTP header that can be used in an HTTP request to provide information about the request context, so that the server can tailor the response. For example, the Accept-* headers indicate the allowed and preferred formats of the response. Other headers can be used to supply authentication credentials (e.g. Authorization), to control caching, or to get information about the user agent or referrer, etc.”

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
https://developer.mozilla.org/en-US/docs/Glossary/Request_header

REQUEST `body`

When making HTTP requests we sometimes need to send data.

The data is wrapped inside of the request `body`.

The request `body` is one of the following:

a string (often JSON encoded string with data, object, arrays, etc.)

form data (`form/multipart`)

blob/ buffer source - binary data

URL search params (`x-www-form-urlencoded`)

https://javascript.info/fetch#post-requests

Fetch and create post

Using `POST` to create a new post object in the resource.

```
// new post object
const newPost = {
    title: "My new post",
    body: "Body description of a new post",
    image: "image url or image data string"
};

const url = "https://race-rest-default-rtdb.firebaseio.com/posts.json";
const response = await fetch(url, {
    method: "POST", // fetch request using POST
    body: JSON.stringify(newPost) // newPost object to JSON
});
```

Fetch and update post

Using `PUT` to an existing post object by given id.

```
const postId = "5tl4jHHSRaKEB0UW9nQd"; // id of the object to update
const postToUpdate = { title: "...", body: "...", image: "..." };

const url = `https://race-rest-default-rtdb.firebaseio.com/posts/${postId}.json`;
const response = await fetch(url, {
    method: "PUT", // using HTTP method PUT
    body: JSON.stringify(postToUpdate)
});
```

Fetch and delete post

Using `DELETE` to an object by given id.

```
const postId = "5tl4jHHSRaKEB0UW9nQd"; // id of the object to update
const url = `https://race-rest-default-rtdb.firebaseio.com/posts/${postId}.json`;

const response = await fetch(url, {
    method: "DELETE"
});
```

Today the slides is your documentation

Read and use them carefully

Fetch & useEffect in React

In `useEffect`, perform the sideeffect (fetch) to get data from a data source.
And save the response data in a state using `useState`.

```
function PostsPage() {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        async function getPosts() {
            const url = "https://raw.githubusercontent.com/cederdorff/web-frontend/main/data/posts.json";
            const response = await fetch(url);
            const data = await response.json();
            setPosts(data);
        }
        getPosts();
    }, []);

    return (
        <section className="page">
```

All you need to know

- Use `await` to tell JS to wait for a fetch call to finish.
- When using `await` you must tell JS that here goes some asynchronous code by wrapping it in an `async` function.

https://www.w3schools.com/js/js_async.asp

```
function PostsPage() {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        async function getPosts() {
            const url = "https://raw.githubusercontent.co
            const response = await fetch(url);
            const data = await response.json();
            setPosts(data);
        }
        getPosts();
    }, []);

    return (
        <section className="page">
            <h1>Posts</h1>
            <section className="grid-container">
                {posts.map(post => (
                    <PostItem post={post} key={post.id} /
```

Builds tools,
Package Managers,
Compilers, Transpilers,
Module Bundlers &
Command-line Interface (CLI)

It is simply just tools!

JavaScript runtime environment

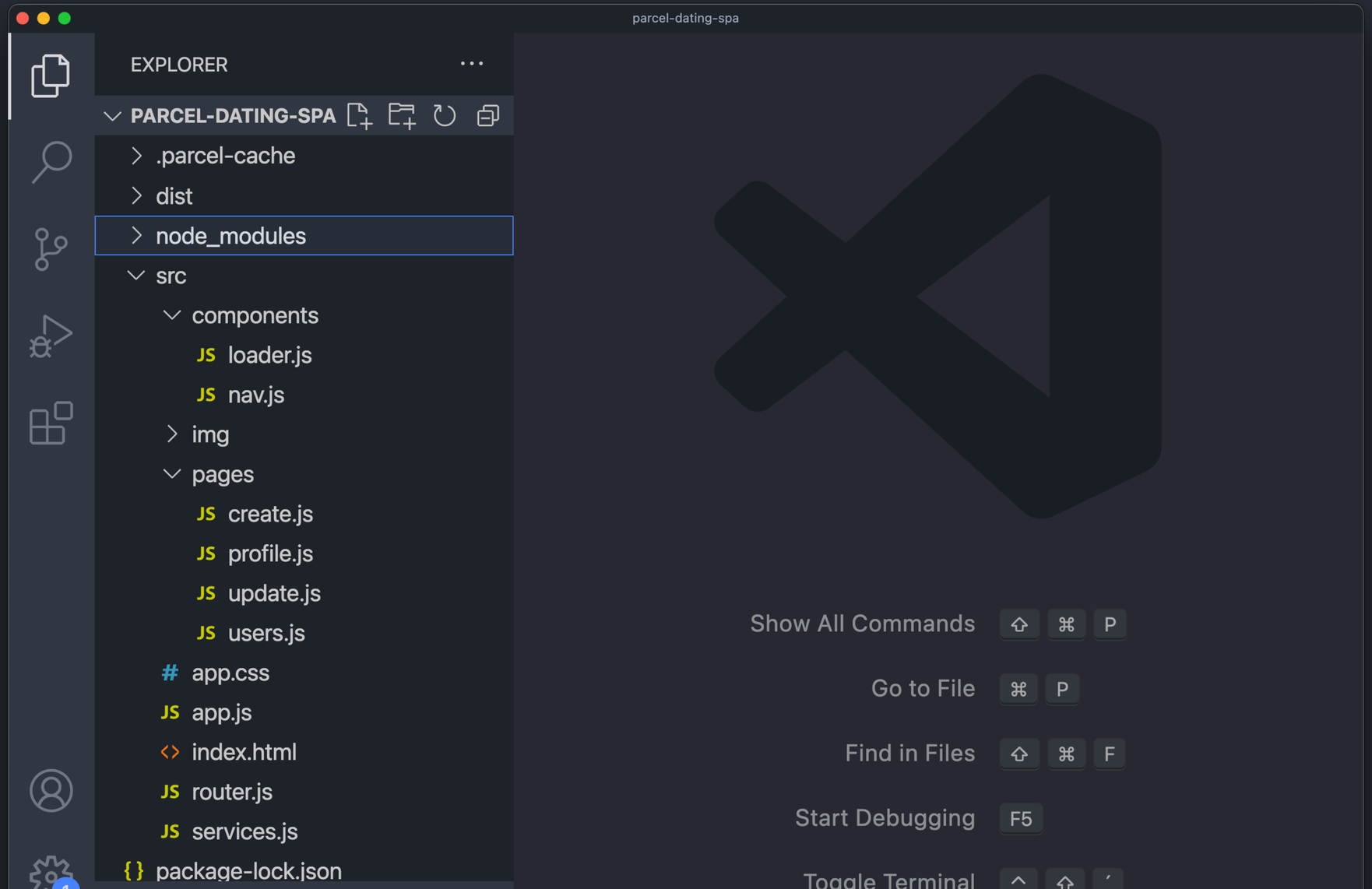
Executes JavaScript outside of the browser

Npm is a package manager for node.js packages - or modules 🤷

Node.js packages contains all the files you need for a module.

Modules are JavaScript libraries you can include in your project.

node_modules



npm consists of

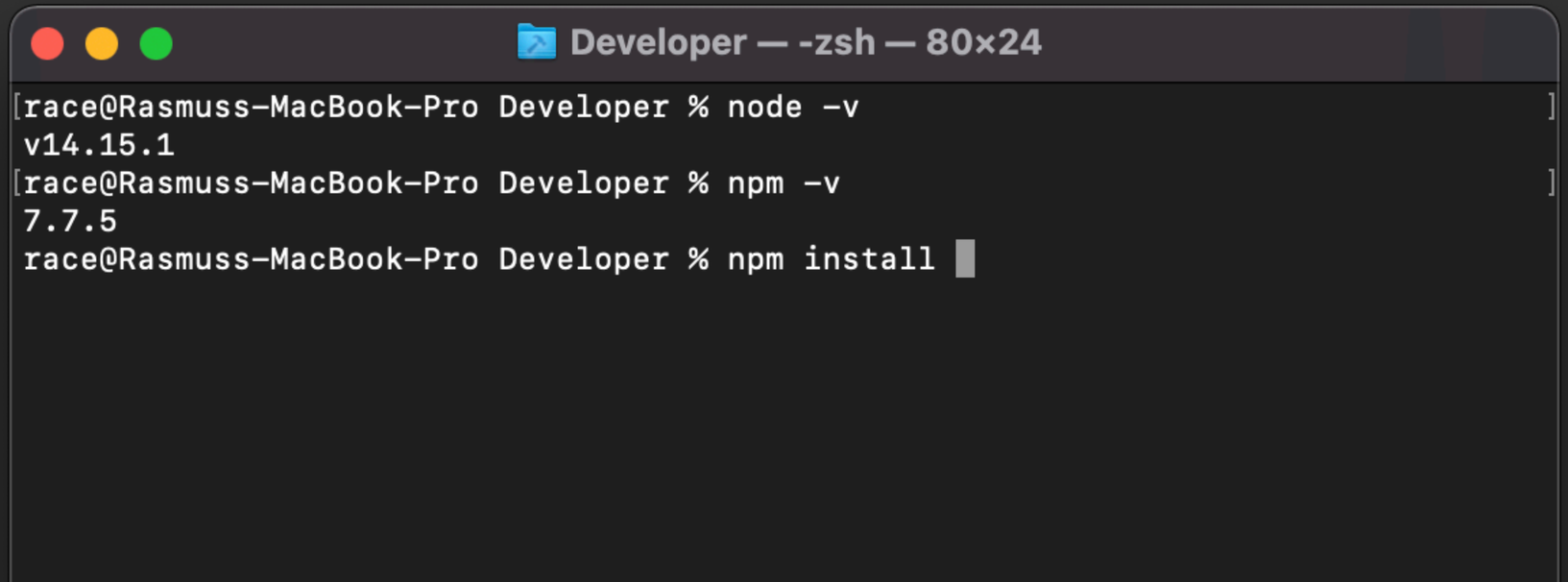
1. Website to discover packages
2. Command line interface (cli)
3. A registry - database with JS software / node modules

https://docs.npmjs.com/about-npm/

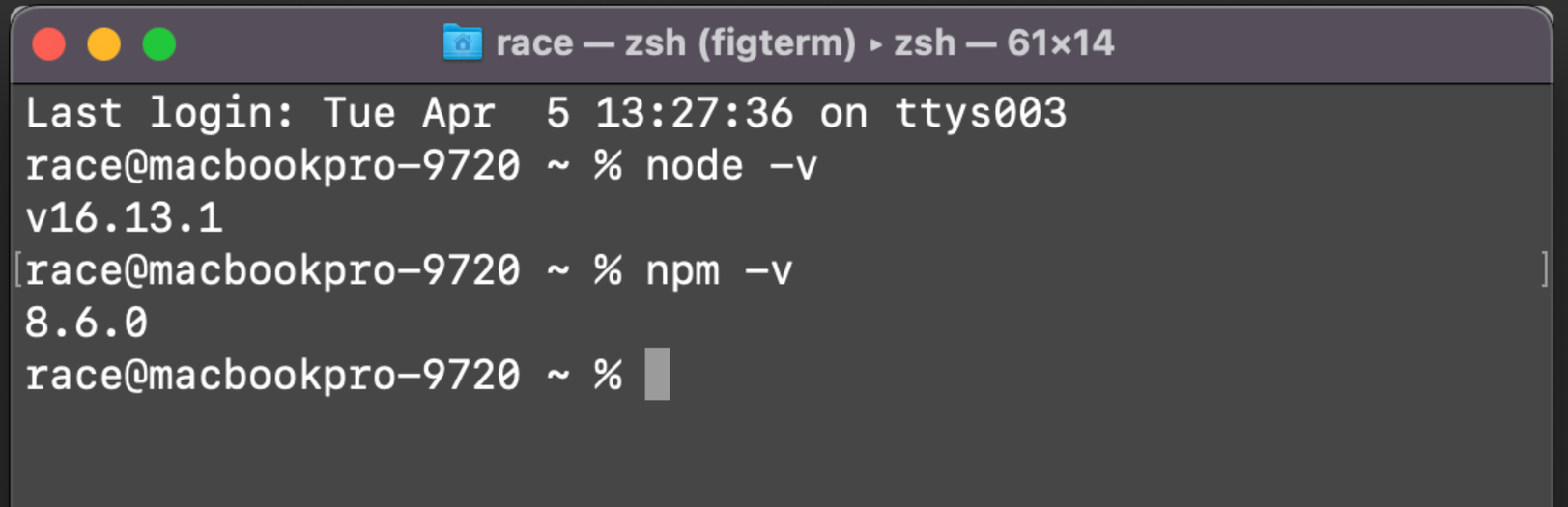
Command Line Interface

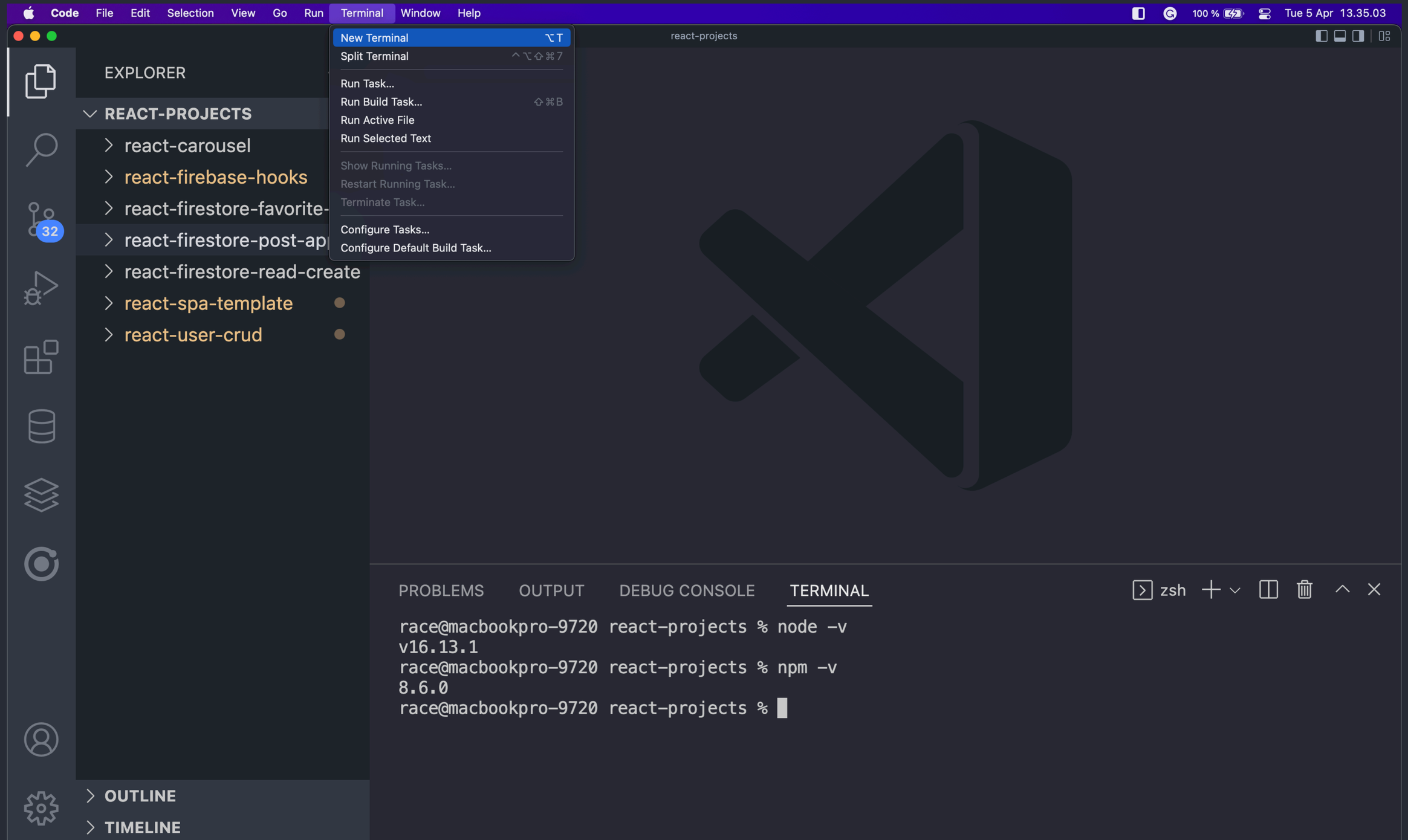
... a program that allows users to type text commands to execute functions.

npm uses CLI to install software & packages.



1. Download & install Node.js & npm: nodejs.org/en/
2. Check your version of npm & Node.js




Code File Edit Selection View Go Run Terminal Window Help
100 % Tue 5 Apr 13.35.03
New Terminal ⌥T
Split Terminal ^⌥⇧⌘7
Run Task...
Run Build Task... ⇧⌘B
Run Active File
Run Selected Text
Show Running Tasks...
Restart Running Task...
Terminate Task...
Configure Tasks...
Configure Default Build Task...
react-projects
EXPLORER
REACT-PROJECTS
react-carousel
react-firebase-hooks
react-firestore-favorite-
react-firestore-post-ap
react-firestore-read-create
react-spa-template
react-user-crud
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
zsh
race@macbookpro-9720 react-projects % node -v
v16.13.1
race@macbookpro-9720 react-projects % npm -v
8.6.0
race@macbookpro-9720 react-projects %

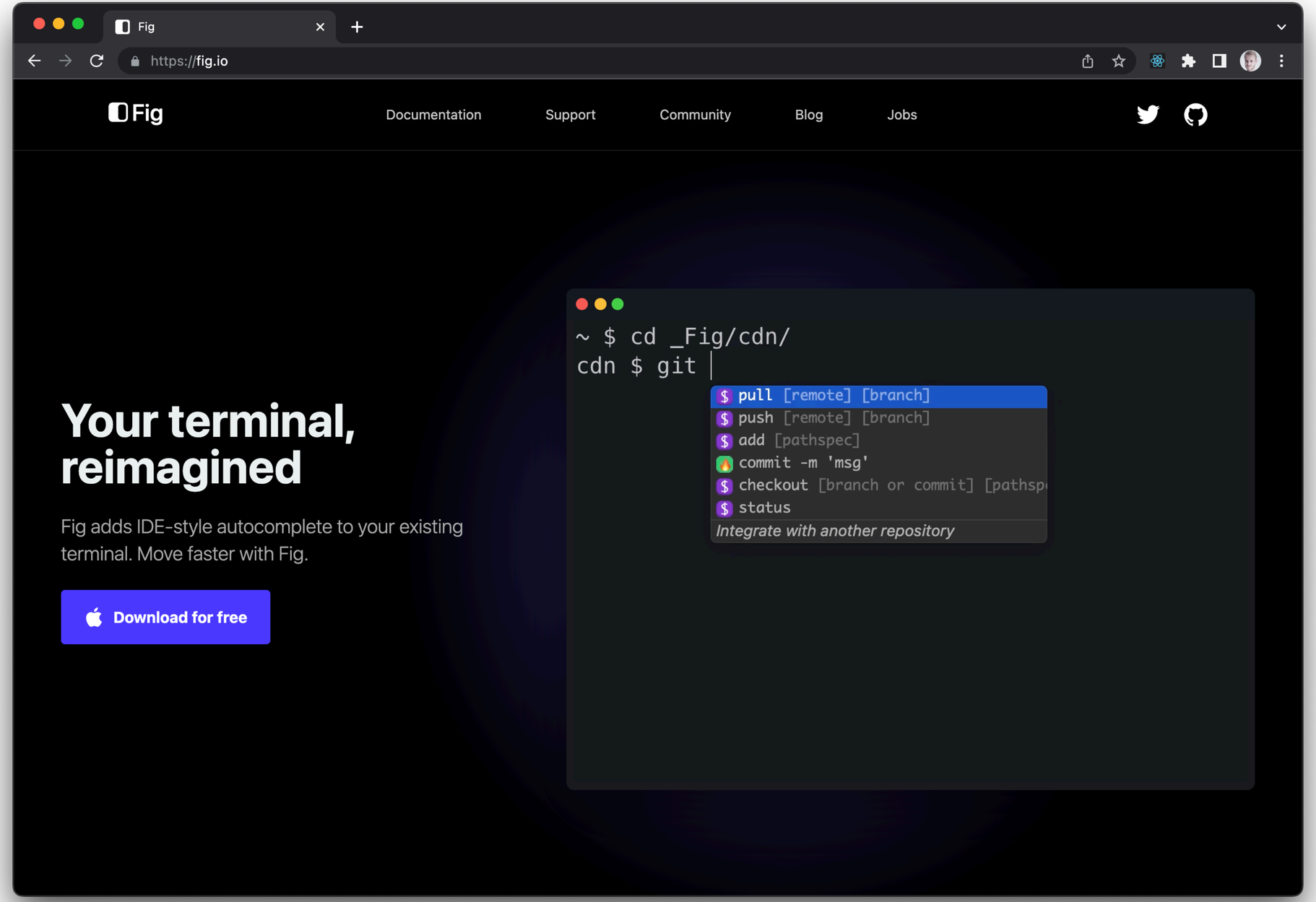
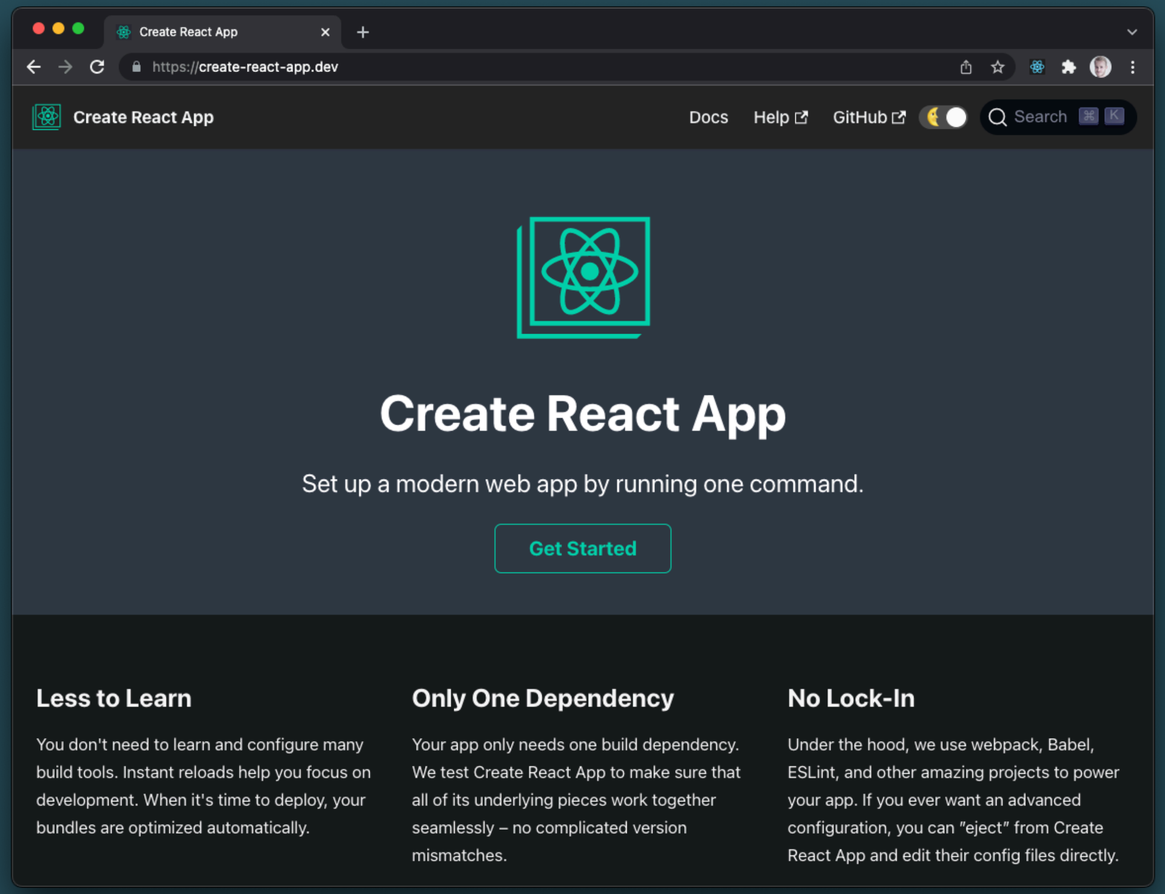


Fig
https://fig.io
Fig
Documentation
Support
Community
Blog
Jobs
Your terminal, reimagined
Fig adds IDE-style autocomplete to your existing terminal. Move faster with Fig.
Download for free
~ $ cd _Fig/cdn/
cdn $ git
pull [remote] [branch]
push [remote] [branch]
add [pathspec]
commit -m 'msg'
checkout [branch or commit] [pathsp
status
Integrate with another repository

https://fig.io/

create-react-app

Starter template for your react app

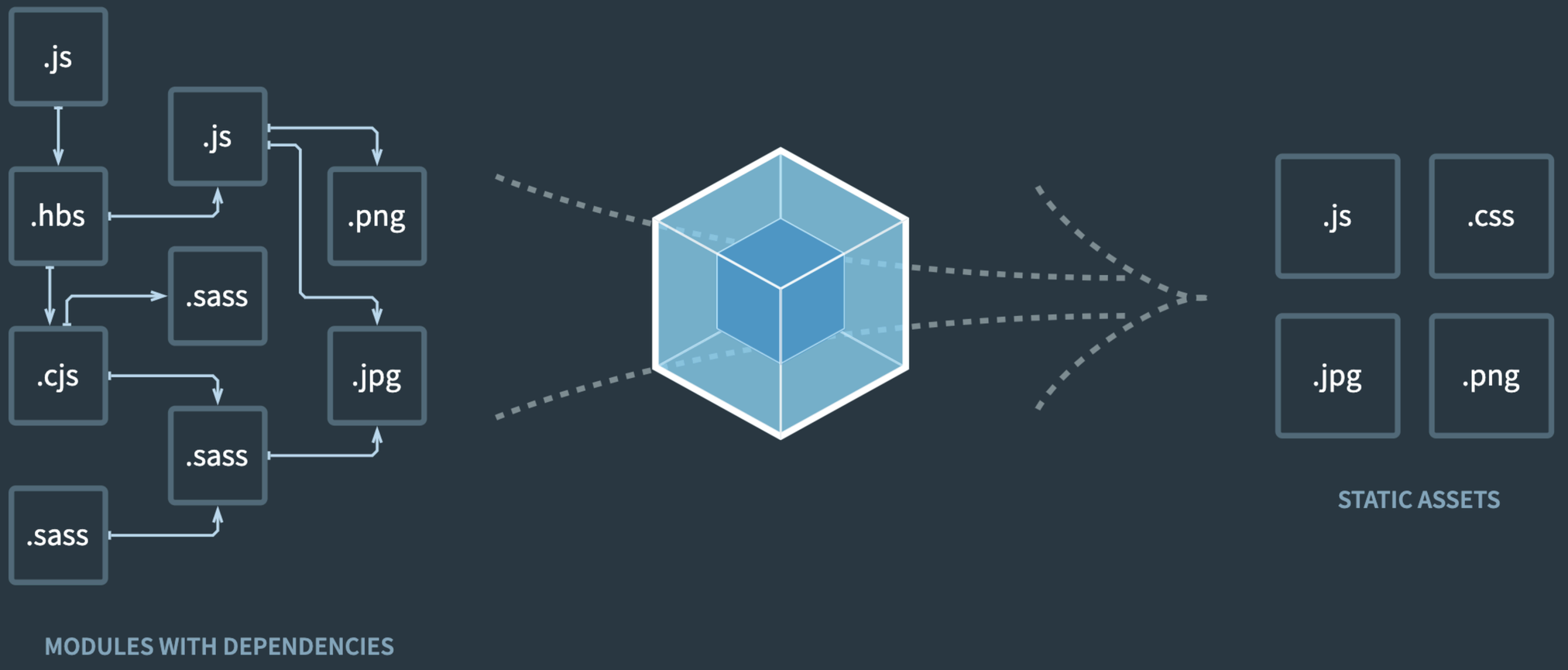


Generate create-react-app

webpack

- A Node.js module
- A module bundler/build tool
- One of many node packages
- From development to production

bundle your scripts

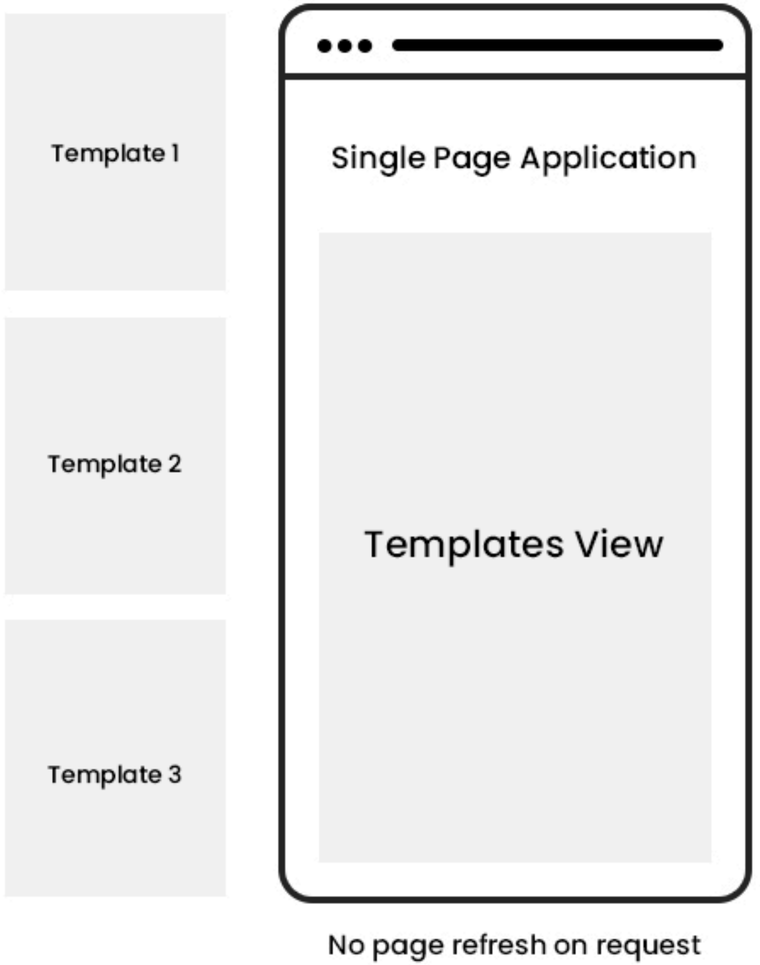


Single Page Apps

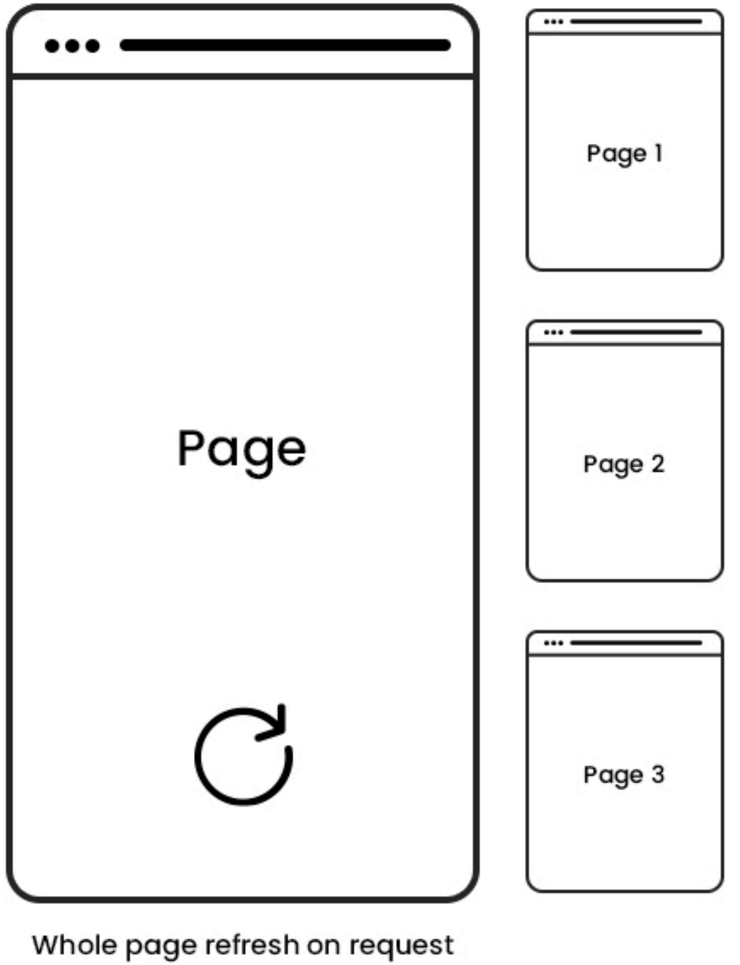
“A single-page application is an application that loads a single HTML page and all the necessary assets (such as JavaScript and CSS) required for the application to run. Any interactions with the page or subsequent pages do not require a round trip to the server which means the page is not reloaded.”

https://reactjs.org/docs/glossary.html#single-page-application

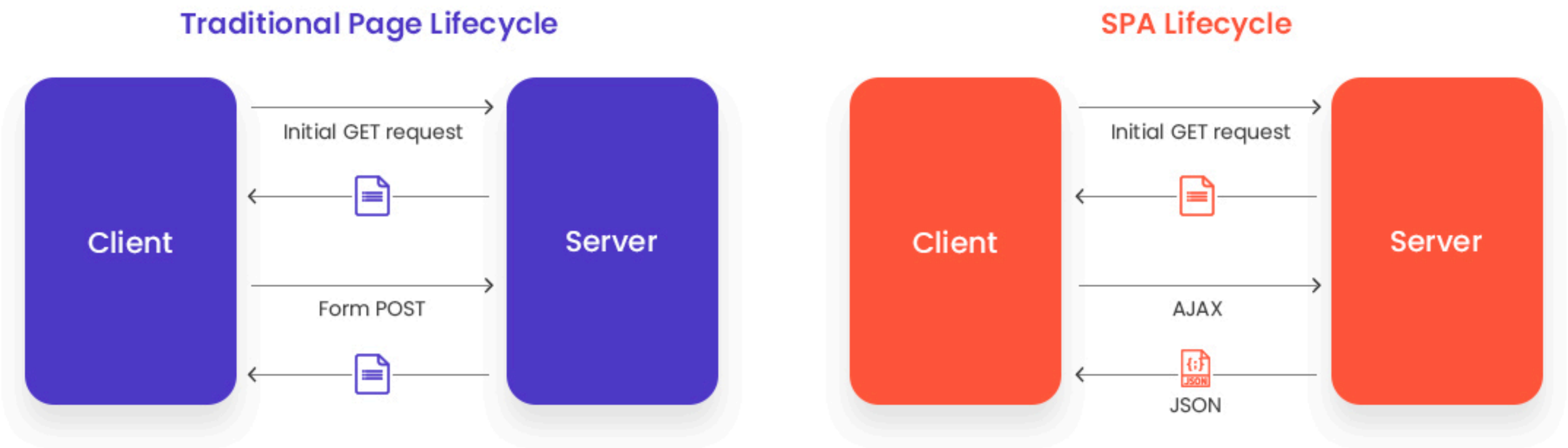
SPA



MPA



excellentwebworld.com/what-is-a-single-page-application



excellentwebworld.com/what-is-a-single-page-application

React Router

... a client & server-side routing library for React

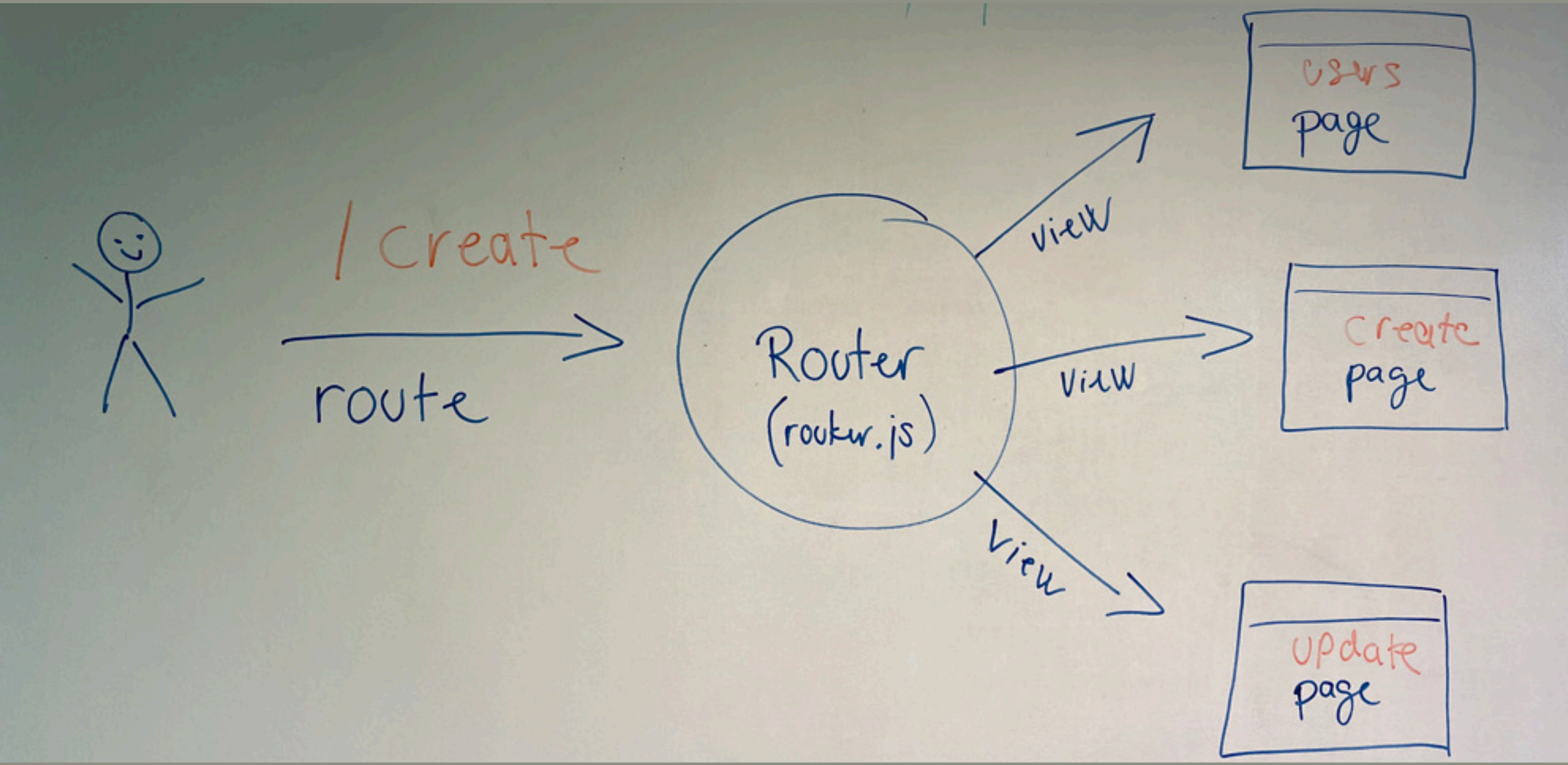
```
<Routes>
    <Route path="/" element={<HomePage />} />
    <Route path="about" element={<AboutPage />} />
    <Route path="*" element={<Navigate to="/" />} />
</Routes>
```

What’s a Router?

“*It is the piece of software in charge to organize the states of the application, switching between different views.*”

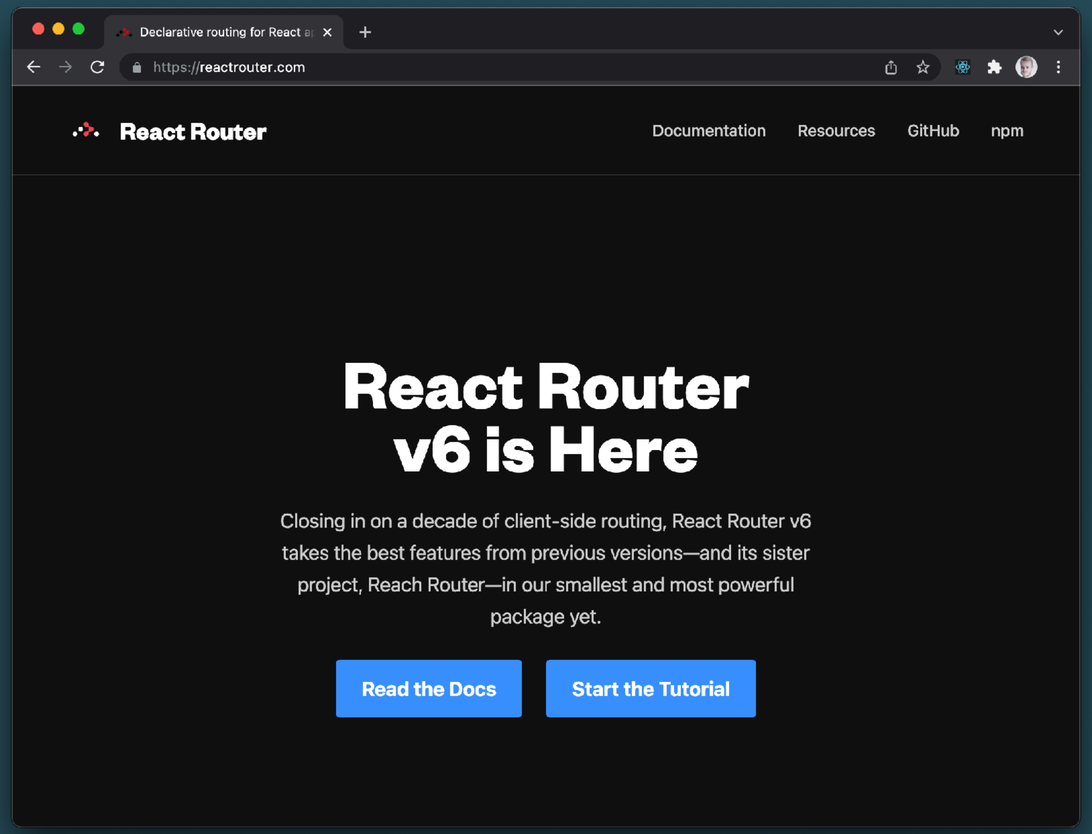
It’s a key component in Single Page Applications.

https://medium.com/@fro_g/routing-in-javascript-d552ff4d2921


/create
route
Router
(router.js)
view
view
view
users
page
create
page
update
page

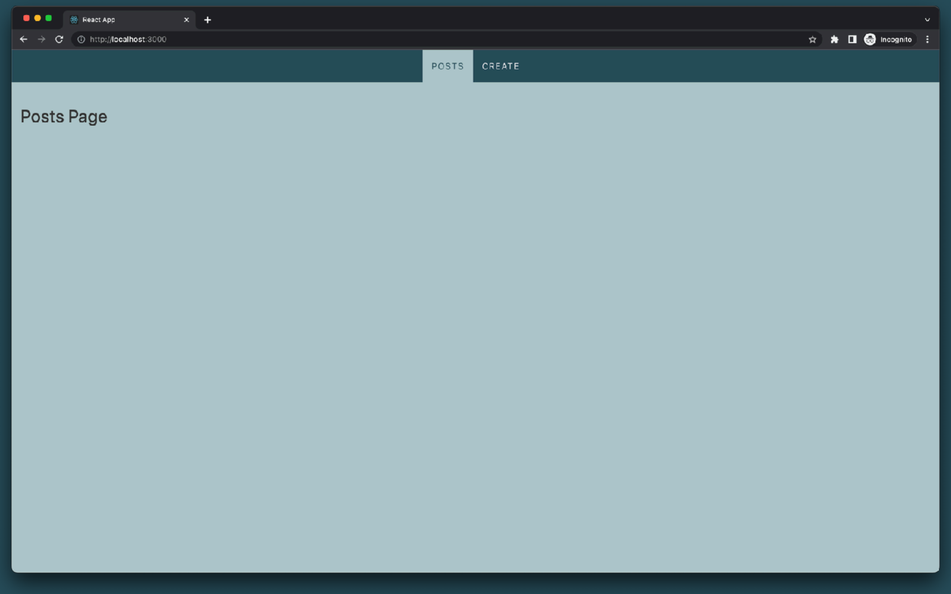
Create React SPA

With React Router



Create React SPA

Boilerplate



react-post-app-boilerplate

- Download: react-post-app-boilerplate
- Paste project folder in your local developer folder.
- Open folder in VS Code.
- Make sure you are in the root of the project.
- Open the terminal in VS Code and run `npm install`

Read

`GET` is the default request method for fetch.

```
import { useState, useEffect } from "react";
import PostCard from "../components/PostCard";

export default function HomePage() {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        async function getPosts() {
            const url = "https://race-rest-default-rtdb.firebaseio.com/posts.json";
            const response = await fetch(url);
            const data = await response.json();
            const postsArray = Object.keys(data).map(key => ({ id: key, ...data[key] }));
            setPosts(postsArray);
        }
        getPosts();
    }, []);

    return (
        <section className="page">
            <section className="grid-container">
                {posts.map(post => (
                    <PostCard post={post} key={post.id} />
                ))}
            </section>
        </section>
    );
}
```

PostCard

Use the prop post to implement the structure of the PostCard component. Skip handleClick for now.

```jsx
import { useNavigate } from "react-router-dom";

export default function PostCard({ post }) {
    const navigate = useNavigate();

    /**
     * handleClick is called when user clicks on the Article (PostCard)
     */
    function handleClick() {
        navigate(`posts/${post.id}`);
    }

    return (
        <article onClick={handleClick}>
            <img src={post.image} alt={post.title} />
            <h2>{post.title}</h2>
            <p>{post.body}</p>
        </article>
    );
}
```

PostCard

Add user details on every PostCard. Add it in PostCard or create a separate component to handle the logic. I created a separate UserAvatar component.

```jsx
import { useNavigate } from "react-router-dom";
import UserAvatar from "./UserAvatar";

export default function PostCard({ post }) {
    const navigate = useNavigate();

    /**
     * handleClick is called when user clicks on the Article (PostCard)
     */
    function handleClick() {
        navigate(`posts/${post.id}`);
    }

    return (
        <article onClick={handleClick}>
            <UserAvatar uid={post.uid} />
            <img src={post.image} alt={post.title} />
            <h2>{post.title}</h2>
            <p>{post.body}</p>
        </article>
    );
}
```

UserAvatar

Based on the prop uid, user detail is fetched from Firebase and saved in a state. Note that the user state is initialised with a default object with placeholders.

```jsx
import { useState, useEffect } from "react";
import placerholder from "../assets/img/user-placeholder.jpg";

export default function UserAvatar({ uid }) {
    const [user, setUser] = useState({
        image: placerholder,
        name: "User's Name",
        title: "User's Title"
    });
    const url = `https://race-rest-default-rtdb.firebaseio.com/users/${uid}.json`;

    useEffect(() => {
        async function getUser() {
            const response = await fetch(url);
            const data = await response.json();
            console.log(data);
            setUser(data);
        }
        getUser();
    }, [url]);

    return (
        <div className="avatar">
            <img src={user.image} alt={user.id} />
            <span>
                <h3>{user.name}</h3>
                <p>{user.title}</p>
            </span>
        </div>
    );
}
```

Create

Implement a form with a field for title, body (description) and image.
Also add an image preview.
Add states for every field including and error message state.

```
export default function CreatePage() {
    const [title, setTitle] = useState("");
    const [body, setBody] = useState("");
    const [image, setImage] = useState("");
    const [errorMessage, setErrorMessage] = useState("");


    return (
        <form onSubmit={handleSubmit}>
            <label>
                Title
                <input type="text" value={title} placeholder="Type a title" onChange={e => setTitle(e.target.value)} />
            </label>
            <label>
                Body
                <input type="text" value={body} placeholder="Type a body text" onChange={e => setBody(e.target.value)} />
            </label>
            <label>
                Image
                <input type="file" className="file-input" accept="image/*" onChange={handleImageChange} />
                <img className="image-preview" src={image} alt="Choose" onError={event => (event.target.src = imgPlaceholder)} />
            </label>
            <p className="text-error">{errorMessage}</p>
            <button type="submit">Save</button>
        </form>
    );
}
```

Create

Implement handleImageChange

```
/**
 * handleImageChange is called every time the user chooses an image in the fire system.
 * The event is fired by the input file field in the form
 */
function handleImageChange(event) {
    const file = event.target.files[0];
    if (file.size < 500000) {
        // image file size must be below 0,5MB
        const reader = new FileReader();
        reader.onload = event => {
            setImage(event.target.result);
        };
        reader.readAsDataURL(file);
        setErrorMessage(""); // reset errorMessage state
    } else {
        // if not below 0.5MB display an error message using the errorMessage state
        setErrorMessage("The image file is too big!");
    }
}
```

Create

onSubmit call handleSubmit. The function creates a new post object based on the states. Also add a hardcoded `uid = “fTs84KRoYw5pRZEWCq2Z”`
Use fetch and post to post the new post object to the data source.

```
async function handleSubmit() {
    const newPost = {
        title: title,
        body: body,
        image: image,
        uid: "fTs84KRoYw5pRZEWCq2Z" // default user id added
    };

    const url = "https://race-rest-default-rtdb.firebaseio.com/posts.json";
    const response = await fetch(url, {
        method: "POST",
        body: JSON.stringify(newPost)
    });
    const data = await response.json();
    console.log(data);
    navigate("/");
}
```

Update

Using `PUT` to an existing post object by given id.

You have two choices:

- Implement an update form similar to the create form.
- Or implement a PostForm component you can reuse for CreatePage and UpdatePage.

Update

PostForm
Component #1

```jsx
import { useEffect, useState } from "react";
import imgPlaceholder from "../assets/img/img-placeholder.jpg";

export default function PostForm({ savePost, post }) {
    const [title, setTitle] = useState("");
    const [body, setBody] = useState("");
    const [image, setImage] = useState("");
    const [errorMessage, setErrorMessage] = useState("");

    useEffect(() => {
        if (post) {
            // if post, set the states with values from the post object.
            // The post object is a prop, passed from UpdatePage
            setTitle(post.title);
            setBody(post.body);
            setImage(post.image);
        }
    }, [post]); // useEffect is called every time post changes.

    /**
     * handleImageChange is called every time the user chooses an image in the fire system.
     * The event is fired by the input file field in the form
     */
    function handleImageChange(event) {
        const file = event.target.files[0];
        if (file.size < 500000) {
            // image file size must be below 0,5MB
            const reader = new FileReader();
            reader.onload = event => {
                setImage(event.target.result);
            };
            reader.readAsDataURL(file);
            setErrorMessage(""); // reset errorMessage state
        } else {
            // if not below 0.5MB display an error message using the errorMessage state
            setErrorMessage("The image file is too big!");
        }
    }
```

Update

PostForm
Component #2

```
    function handleSubmit(event) {
        event.preventDefault();
        const formData = {
            // create a new objebt to hold the value from states / input fields
            title: title,
            image: image,
            body: body
        };

        const validForm = formData.title && formData.body && formData.image; // will return false if one of the properties doesn't hav
        if (validForm) {
            // if all fields/ properties are filled, then call savePost
            savePost(formData);
        } else {
            // if not, set errorMessage state.
            setErrorMessage("Please, fill in all fields.");
        }
    }

    return (
        <form onSubmit={handleSubmit}>
            <label>
                Title
                <input type="text" value={title} placeholder="Type a title" onChange={e => setTitle(e.target.value)} />
            </label>
            <label>
                Body
                <input type="text" value={body} placeholder="Type a body text" onChange={e => setBody(e.target.value)} />
            </label>
            <label>
                Image
                <input type="file" className="file-input" accept="image/*" onChange={handleImageChange} />
                <img className="image-preview" src={image} alt="Choose" onError={event => (event.target.src = imgPlaceholder)} />
            </label>
            <p className="text-error">{errorMessage}</p>
            <button type="submit">Save</button>
        </form>
    );
}
```

Update

CreatePage Component

```jsx
import { useNavigate } from "react-router-dom";
import PostForm from "../components/PostForm";

export default function CreatePage({ showLoader }) {
    const navigate = useNavigate();

    async function createPost(newPost) {
        newPost.uid = "fTs84KRoYw5pRZEWCq2Z"; // default user id added

        const url = "https://race-rest-default-rtdb.firebaseio.com/posts.json";
        const response = await fetch(url, {
            method: "POST",
            body: JSON.stringify(newPost)
        });
        const data = await response.json();
        console.log(data);
        navigate("/");
    }

    return (
        <section className="page">
            <h1>Create New Post</h1>
            <PostForm savePost={createPost} />
        </section>
    );
}
```

Update

UpdatePage Component

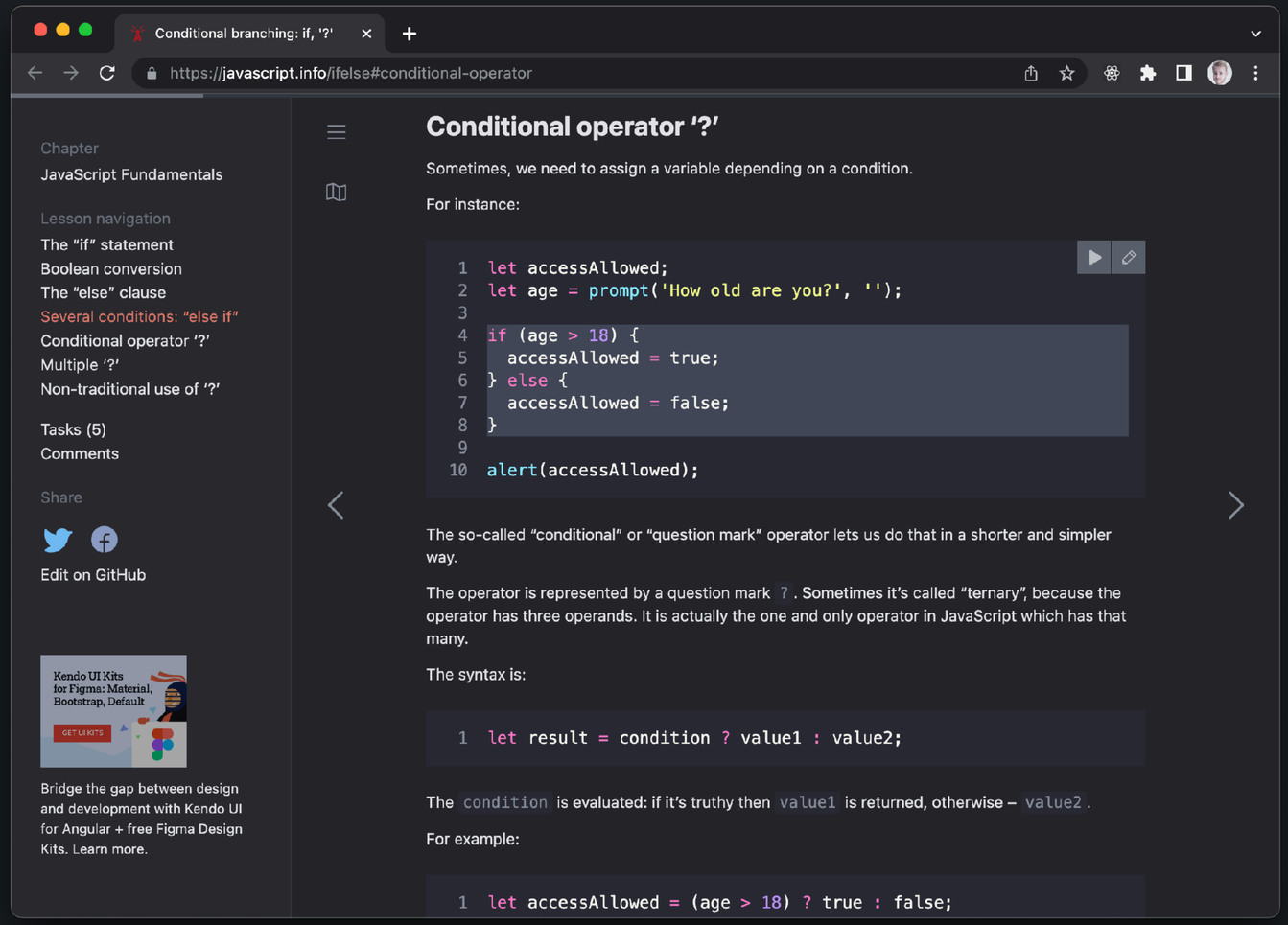
```jsx
import { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import PostForm from "../components/PostForm";

export default function UpdatePage() {
    const [post, setPost] = useState({});
    const params = useParams();
    const navigate = useNavigate();
    const url = `https://race-rest-default-rtdb.firebaseio.com/posts/${params.postId}.json`;

    useEffect(() => {
        async function getPost() {
            const response = await fetch(url);
            const data = await response.json();
            setPost(data);
        }

        getPost();
    }, [url]);

    async function savePost(postToUpdate) {
        const response = await fetch(url, {
            method: "PUT",
            body: JSON.stringify(postToUpdate)
        });
        const data = await response.json();
        console.log(data);
        navigate("/");
    }

    return (
        <section className="page">
            <h1>Update Post</h1>
            <PostForm post={post} savePost={savePost} />
        </section>
    );
}
```

Delete

Using `DELETE` to an object by a given id.
Add deletePost logic in UpdatePage component.

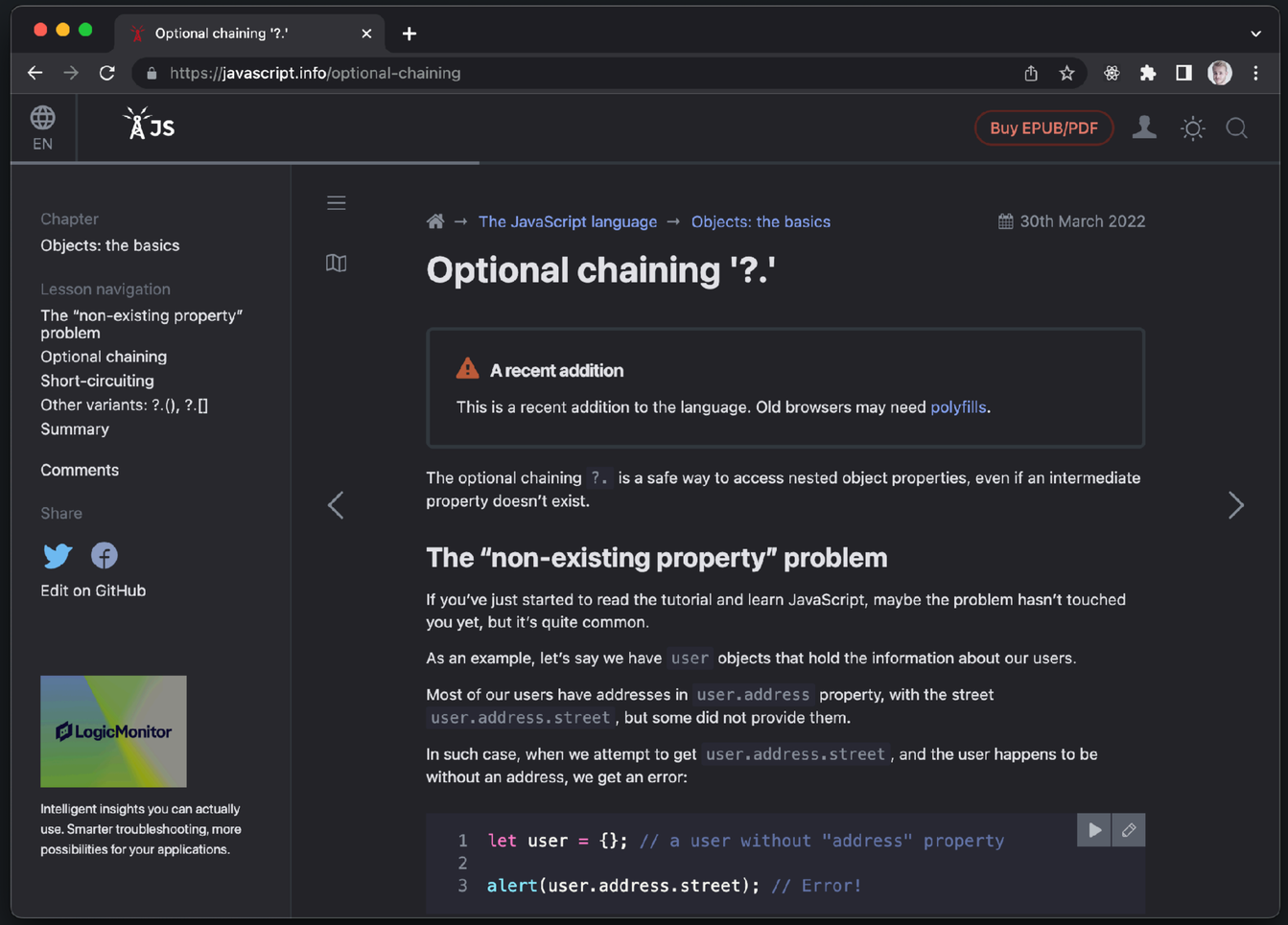
```jsx
import { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import PostForm from "../components/PostForm";

export default function UpdatePage() {
    const [post, setPost] = useState({});
    const params = useParams();
    const navigate = useNavigate();
    const url = `https://race-rest-default-rtdb.firebaseio.com/posts/${params.postId}.json`;

    useEffect(() => {…
    }, [url]);

    async function savePost(postToUpdate) {…
    }

    async function deletePost() {
        const confirmDelete = window.confirm(`Do you want to delete post, ${post.title}?`);
        if (confirmDelete) {
            const response = await fetch(url, {
                method: "DELETE"
            });
            const data = await response.json();
            console.log(data);
            navigate("/");
        }
    }

    return (
        <section className="page">
            <h1>Update Post</h1>
            <PostForm post={post} savePost={savePost} />
            <button className="btn-delete" onClick={deletePost}>
                Delete Post
            </button>
        </section>
    );
}
```

Shorter and simpler way to do `if/else`



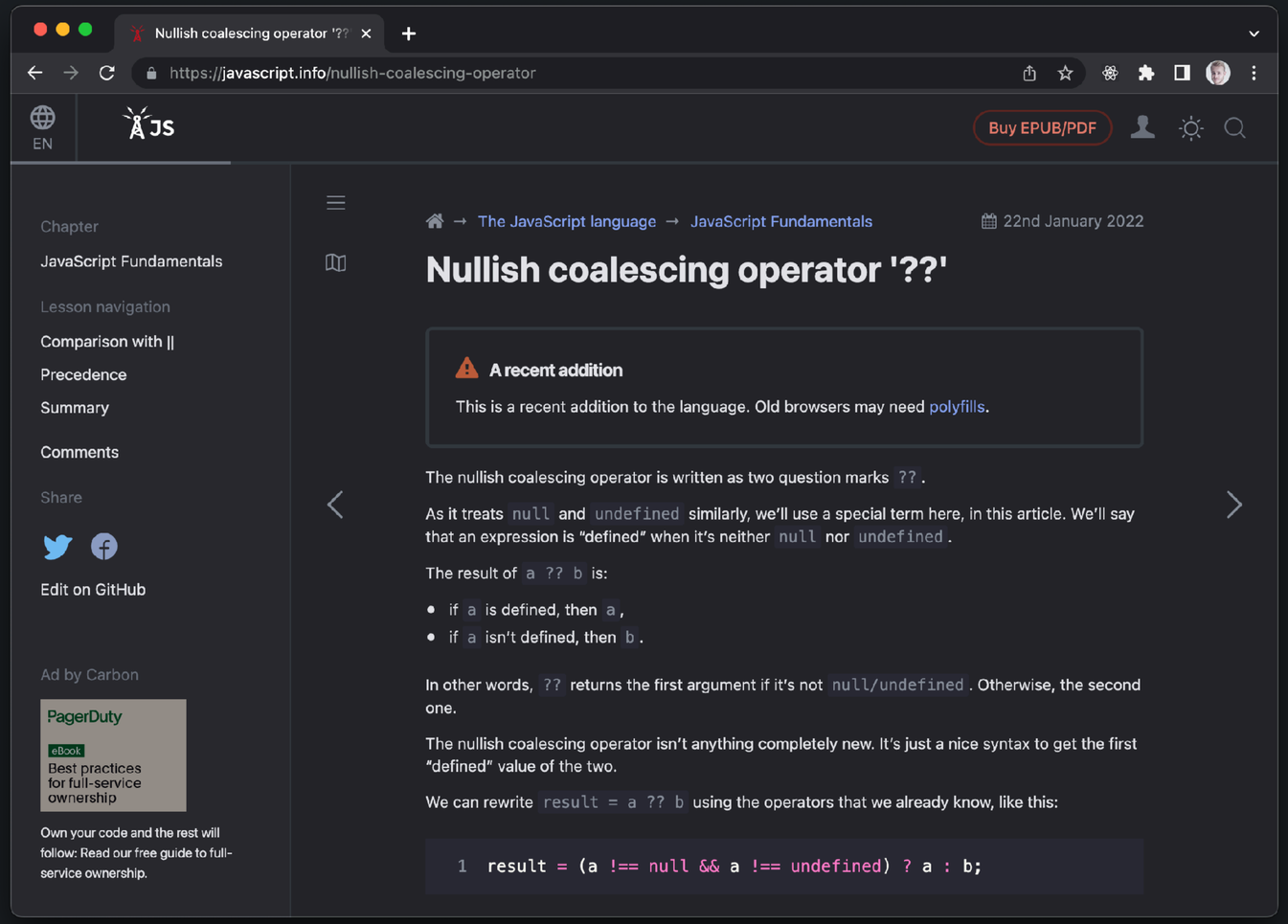
https://javascript.info/ifelse#conditional-operator
Ternary Operator

?. is a safe way to access nested object properties



https://javascript.info/optional-chaining

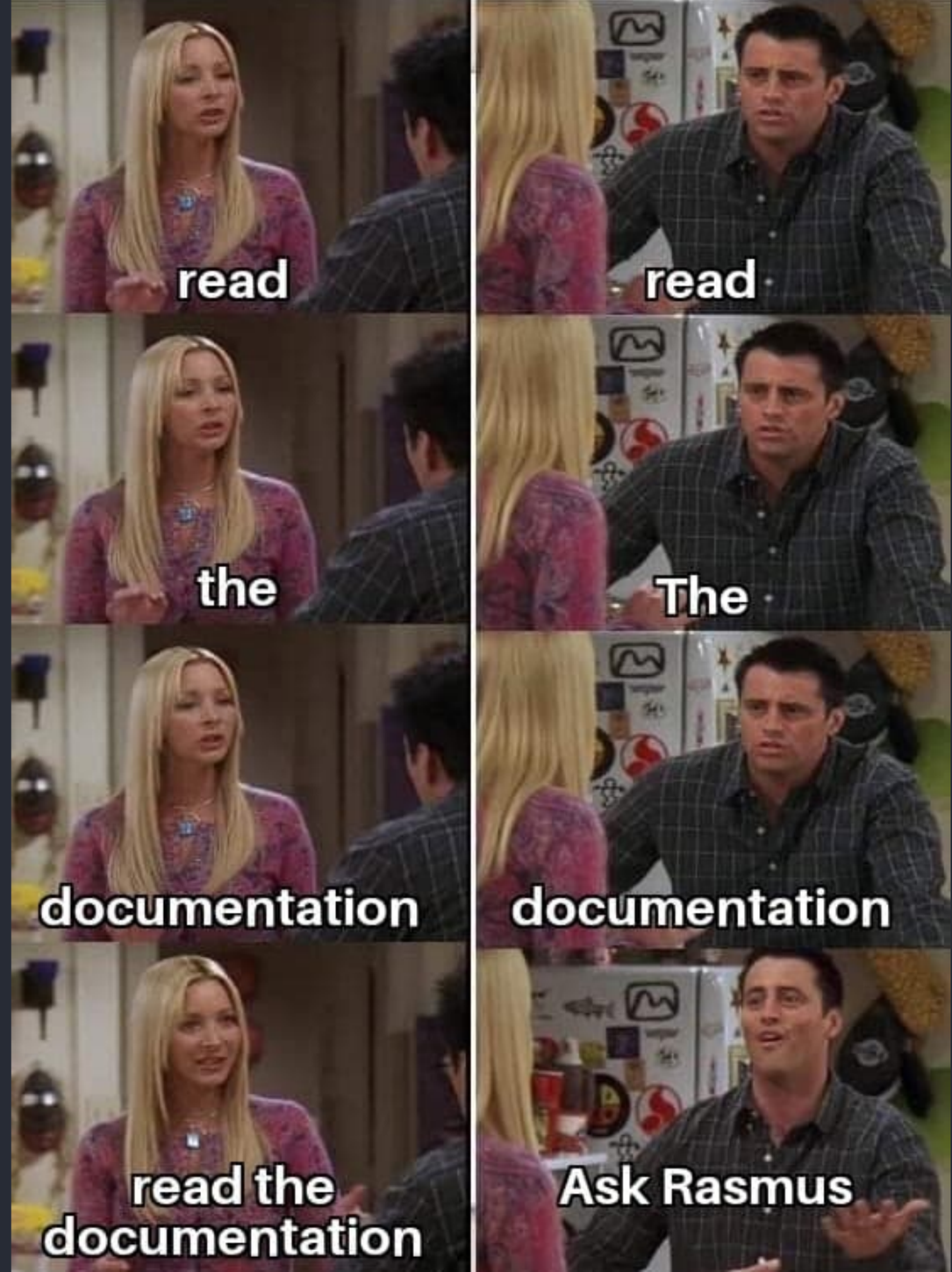
?? returns the first argument if it's not null/undefined



https://javascript.info/nullish-coalescing-operator

But RACE, how do you know all that stuff?

- React Resources
- React Self-study
- Guides & Tutorials
- From Vanilla JavaScript to React Developer



Suggested Solution

- GitHub Repo (code)
- Running solution
- Data Sources:
  - posts
  - users

Code Every Day