

BUILDING INTERACTIVE WEB APPS WITH REACT

informatica femminile Bremen, 2023



Hi, I'm Rike,

Software Developer @IAV, Berlin

♥React ♥UI/UX ♥Web3

Twitter: @rike.codes

Web: rike.dev

LinkedIn: <https://www.linkedin.com/in/rike-exner>

SOME BASIC RULES

(✓) There's much ahead! Please focus!

(✓) Use any resources you might useful on your way (and kindly share them).

(✓) There are no dumb questions.

(✓) Help each other!



<https://gitfrosh.github.io/pokeapp/>

INTRO

- Outline
 - Sat 19th August
 - 9.00-12.30 a.m. and 2.00-4.30 p.m.
- Proof of Attendance (Certificate)
- no CP

CHATTOOL / ETHERPAD

- <https://yopad.eu/p/react-course>

LET'S GET TO KNOW EACH OTHER!

Please share your name & 3-5
hashtags about you in our
Etherpad! Share a social
(LinkedIn, Instagram, Twitter,..)
too, if you want to connect!

(studies, hobbies, passions, ..)

#


#

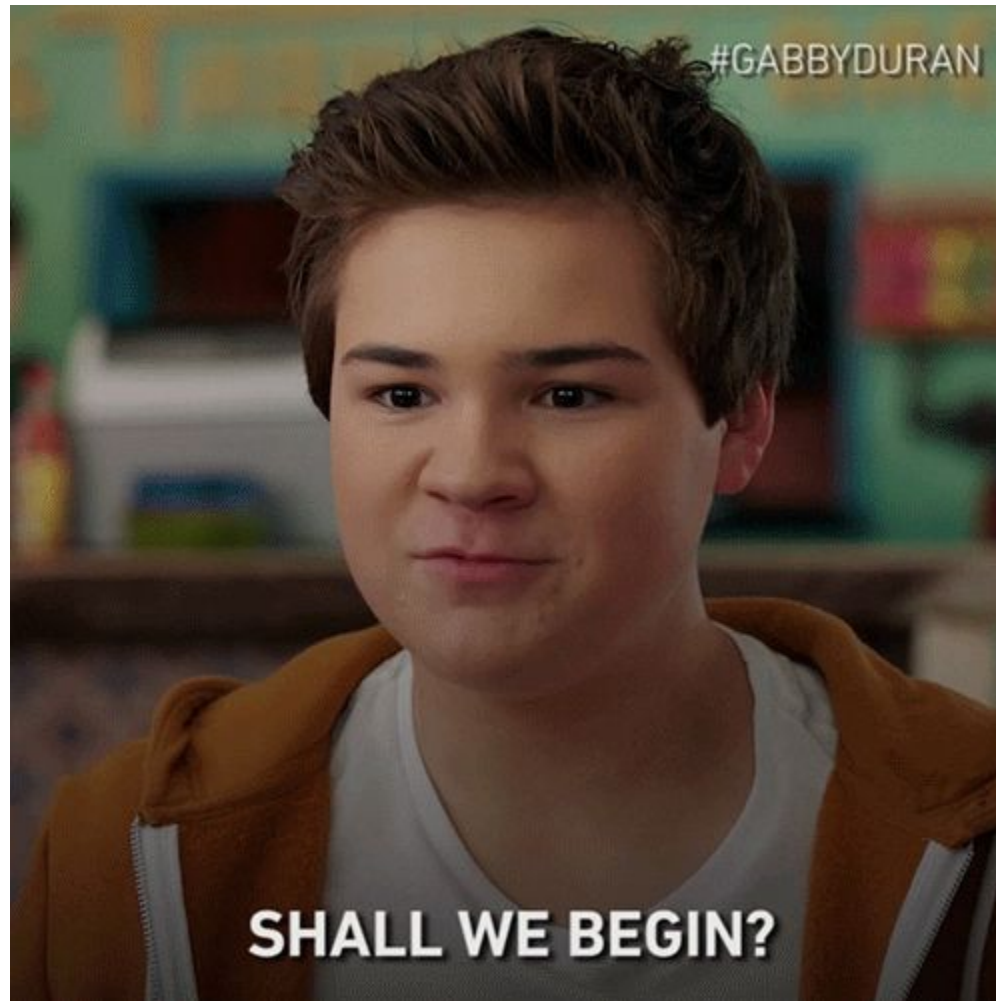
#

#

#

AGENDA

- Recap frontend dev basics
 - Frontend / Backend
 - HTML
 - CSS
 - Javascript
 - JSON
- Intro React & Setup
 - Importance & ecosystem
 - Know your dev tools
 - IDE & Installation
- React Components and JSX
 - Functional components
 - JSX in action
- Props
- Styling in React
 - Layout & Grid
- State Management
- Hooks
- Data Fetching
 - “fetch”-Library
- Advanced Topics (testing, deploying, Typescript)
- Final Setup & Wrap-up 



Questions?

The basics you need to
know about the web
development tech stack.

CLIENT-SIDE VS. SERVER-SIDE WEB DEVELOPMENT

- **Client-side languages** (HTML, CSS, JavaScript) are interpreted by browsers (= client)
(--> *Frontend-Development*)
- **Server-side languages** (PHP, Python, Ruby, C#, Go, Java, JavaScript) are interpreted by the server
(--> *Backend-Development*)
- **Fullstack-Developers:** Developers, who (think that they) can do it all 💪

WEBSITE, WEB APP, WHAT?!

- I will use the term “web app” or simply “app” for the project we build, because modern websites tend to handle so much more than just displaying information!
- .. think about Netflix, Spotify, and many more super-complex web apps!
- this course does not deal with “mobile app development”!

HTML IN 3 MINUTES

HTML acts as the base structure of any web app.

HTML Page Structure

```
<!DOCTYPE html>  ← Tells version of HTML
<html>           ← HTML Root Element

<head>           ← Used to contain page HTML metadata
  <title>Page Title</title> ← Title of HTML page
</head>

<body>           ← Hold content of HTML
  <h2>Heading Content</h2> ← HTML heading tag
  <p>Paragraph Content</p> ← HTML paragraph tag
</body>

</html>
```

```
<!DOCTYPE html>
<html>

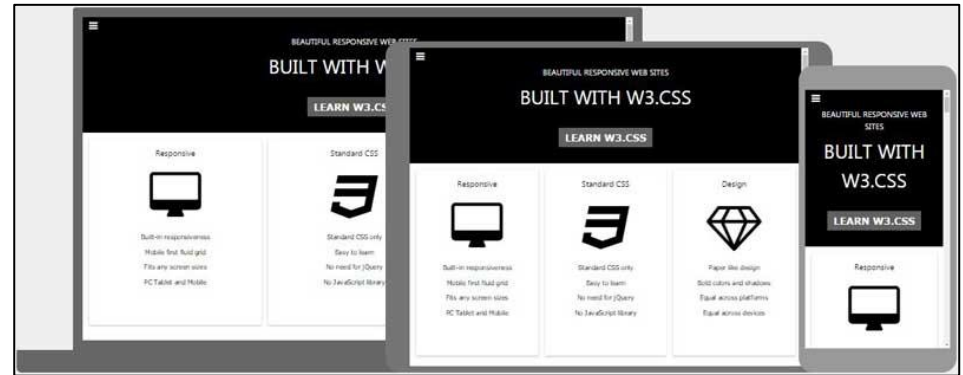
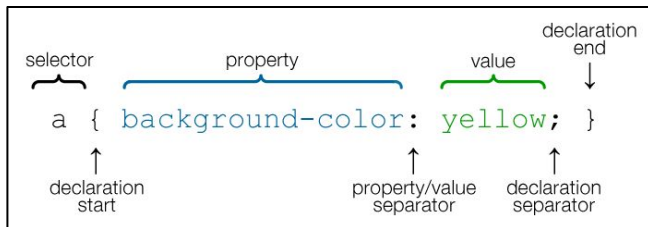
<head>
  <title>Page Title</title>
</head>

<body>
  <h2>Heading Content</h2>
  <p>Paragraph Content</p>
</body>

</html>
```

CSS IN 3 MINUTES

CSS adds (responsive) styling and formatting to any website.



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>My Cool Website</title>
5     <style>
6       body {background-color: ■aliceblue;}
7       h1 {color:■darkblue;}
8       p {color:■darkred;}
9     </style>
10  </head>
```

*optional: use of SASS or LESS
to more sophisticated
functionality to CSS*

<https://en-support.files.wordpress.com/2011/09/css-selectors-1rg.png>

JAVASCRIPT IN 10 MINUTES

Javascripts adds
interactivity and
dynamics to any web app.

```
link-js.html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>JavaScript</title>
6      <style>
7          span {
8              background-color: red;
9          }
10     </style>
11 </head>
12 <body>
13     <!-- inline mode -->
14     <h1 onclick="alert('hello from the body section')">I am headline</h1>
15
16     <!-- using script tag - internal style -->
17     <script type="text/javascript">
18         alert('hello from script tag');
19     </script>
20
21     <!-- using script tag - external style -->
22     <script type="text/javascript" src="main.js"></script>
23
24 </body>
</html>
```

[https://raw.githubusercontent.com/Luorinz/images/master/9.%20JavaScript%20Bas
ic-2019-4-13-18-38-21.png](https://raw.githubusercontent.com/Luorinz/images/master/9.%20JavaScript%20Bas%20ic-2019-4-13-18-38-21.png)

VARIABLES AND FUNCTIONS

- Declare a variable

```
let age = 25;  
age = 25 + 1;
```

```
const eyeColor = "blue";
```

```
// don't use var anymore
```

- Create a function

```
function sayMyName (name) {  
  const greeting = `Hello, ${name}!`;   
  return greeting;  
}
```

- Call a function

```
sayMyName("Anna"); // Will print "Hello, Anna!"
```

OBJECTS AND ARRAYS

- Create an object and grab it's values

```
const person = { firstname: 'John', age: 25 };  
const { firstname, age } = person;  
console.log(firstname, age); // John 25  
console.log(person.firstname); // John
```

- Create an array and grab its first value

```
const names = ['John', 'Jack', 'Julia'];  
console.log(arr[0]); // John
```

ARRAY FUNCTIONS

- functions like `map()`, `filter()`, and `forEach()`, are used frequently in React. For instance, you often use the `map()` function to render lists of components.

```
const pets = [{ name: 'Max', type: 'dog' }, { name: 'Karla', type: 'cat' }];  
pets.map(pet => {  
  return <h1>{pet.name}</h1>  
})
```

... or `filter()` to filter out items from an array that don't match a specific condition.

```
pets.filter(pet => {  
  return pet.type === "dog";  
})
```

FETCH DATA WITH ASYNC/AWAIT

- The **fetch** function in Javascript accesses resources externally. You can make HTTP requests (using GET, POST and other methods), download, and upload data.
- Async/await simplifies handling with promises.
- Without “await”, code procession would just move on and NOT wait for the server result.

```
async function fetchMovies() {  
  const response = await fetch('http://fakemoviedatabase.com/allMovies');  
  // waits until the request ("promise") completes...  
  console.log(response);  
}
```

IMPORT / EXPORT FUNCTIONS TO USE THEM ELSEWHERE

- export from myComponent.js

```
export default function MyComponent() {}
```

- import where needed

```
import MyComponent from '@components/component1';
```

- if the exported function is not a default export, you'll need to use curly brackets, e.g.

```
import { AnotherComponent } from '@components/component2';
```

WRAP-UP



HTML



HTML + CSS



HTML + CSS
+ JAVASCRIPT

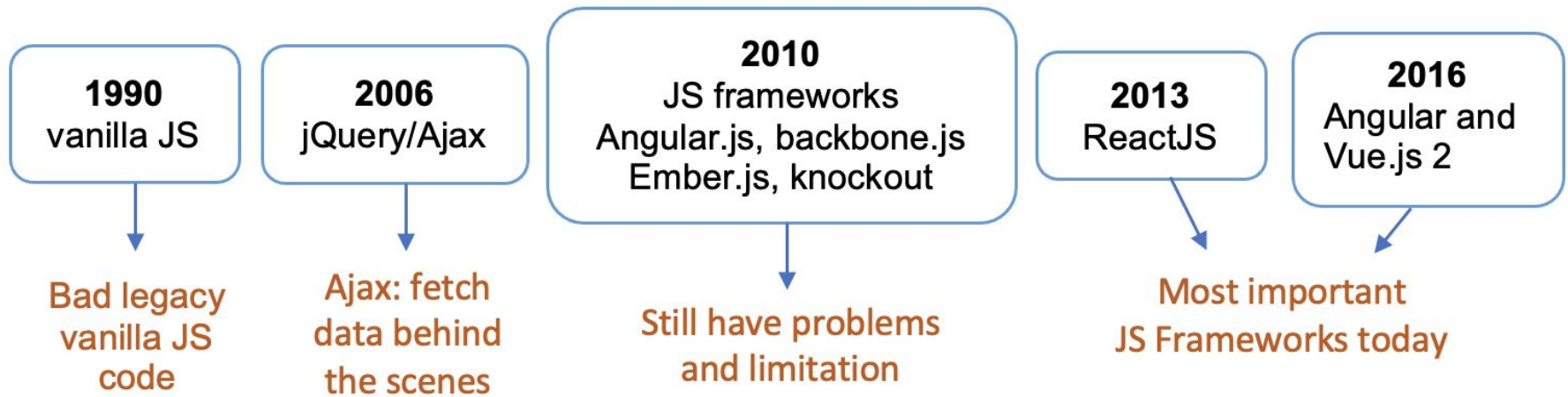
JSON

- It's a format for storing and transferring data.
- This is a JSON object, and looks very similar to a Javascript object, nice huh?
- You can convert JSON to a Javascript object easily and vice versa.
- JSON can be handled by many programming languages and it's the go-to standard to share data over the web programmatically

```
let jsObject = { name: "John", age: 30, city: "New York" };  
let jsonData = JSON.stringify(jsObject);  
console.log(jsonData); // Outputs: '{"name":"John","age":30,"city":"New York"}'
```

```
let jsonData = '{"name":"John", "age":30, "city":"New York"}';  
let jsObject = JSON.parse(jsonData);  
console.log(jsObject.name); // Outputs: "John"
```

THE EVOLUTION OF JAVASCRIPT



The “big 3” JS frameworks



**Why did web developers
make the shift towards JS
frameworks and
Client-Side-Rendered
Apps?**

WHY WOULD ONE PREFER A FRAMEWORK SUCH AS REACT OVER VANILLA JS?

- Efficiency
- Structure and Organization
- Security
- Community and Support
- Testing

React ecosystem

Why React?

@angular/core vs react vs vue

Enter an npm package...

@angular/core x

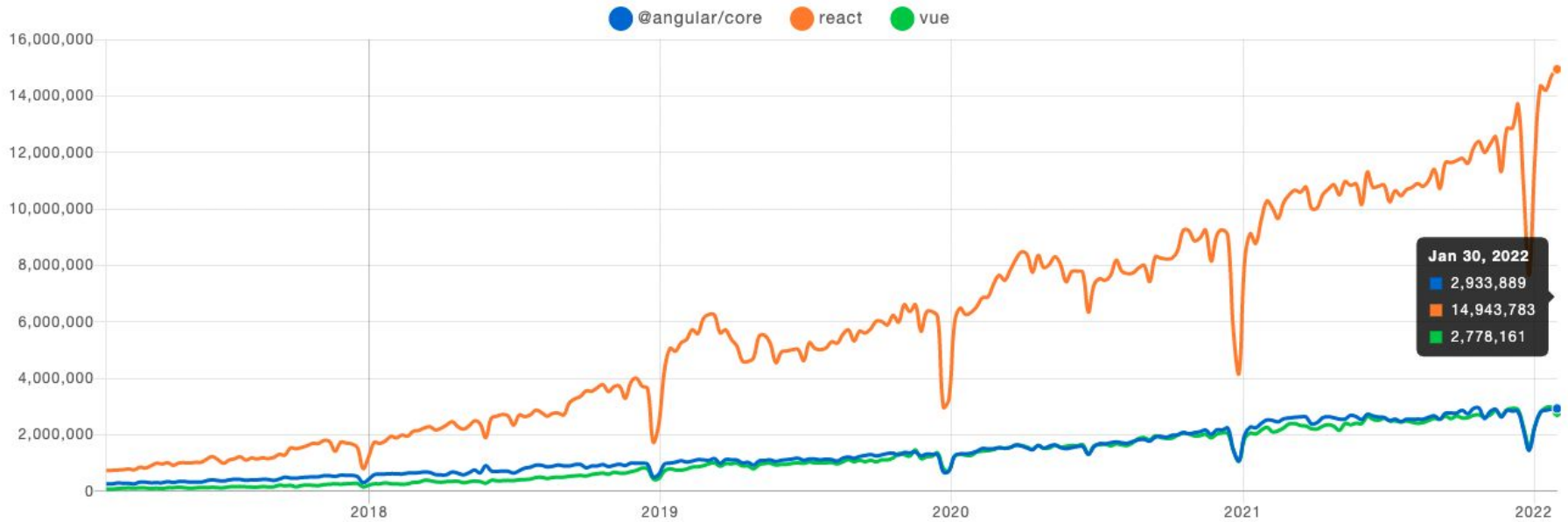
react x

vue x

+ angular

+ ember-source

Downloads in past 5 Years





Hands-on!

**Let's dive
into the React
docs first!**

LET'S START WITH A NEXT.JS PROJECT

- With your Terminal or your Explorer, move to the a folder, where you want to work in, or create a new folder (e.g. "React-Project")
- Open up VS Code and click on "Open Folder", choose "React-Project" and click on "Open"
- In the Navigation Click on Terminal -> New Terminal
- In the integrated Terminal,type ..

```
npm install npx
```

- when finished, type

```
npx create-next-app
```

... approve next question

... and choose following specifications

```
• rike@voyager-42:~/CODING/Testi-IF$ npx create-next-app
✓ What is your project named? ... pokedex
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to customize the default import alias? ... No / Yes
Creating a new Next.js app in /home/rike/CODING/Testi-IF/pokedex.
```


LET'S START WITH A NEXT.JS PROJECT

- move into “pokedex” in your Terminal with

```
cd pokedex
```

- install a nice UI Library (we use AntDesign, if you feel adventurous, try another

```
npm install antd
```

- start your development server with

```
npm run dev
```

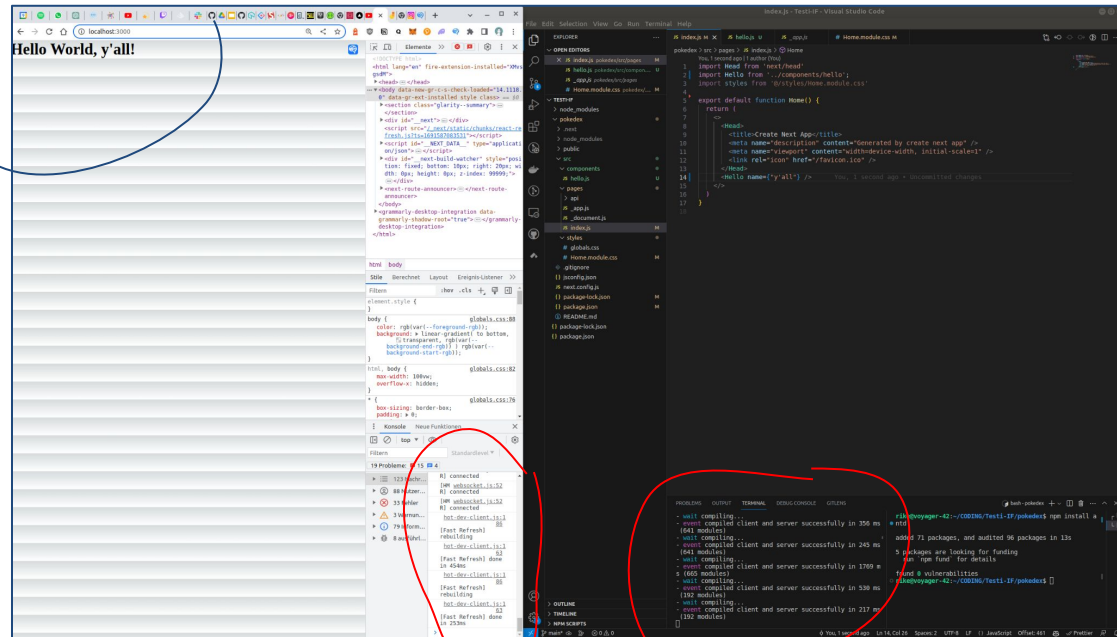
- brace yourself for coding!
 - open Chrome and navigate to <http://localhost:3000>
 - oh, Next.js set up some boilerplate code for us!

```
npm install antd
```

IMPROVE YOUR DEVELOPER SETUP

- put your IDE left, and your browser on the right hand side of your screen!
- prepare your Chrome Developer Tools!
 - look for your browser console
 - try to grab some elements to analyze their CSS styles
 - switch between browser and tablet / phone dimensions

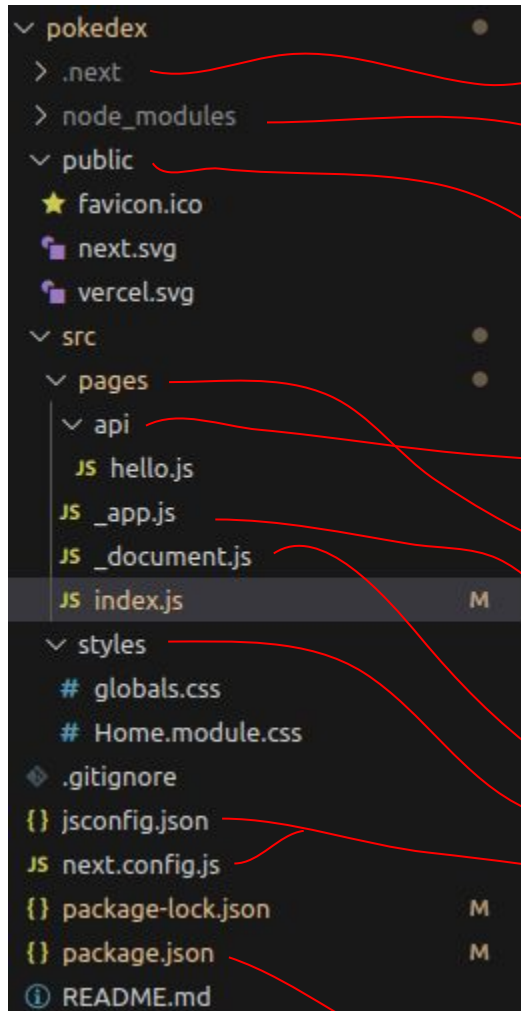
your web app will reload every time you change your code ("hot reload")



your browser console will log errors and console.log -statements from your running code

your IDE console will complain if your code has severe syntax errors, if you close down this CLI, your app and dev server will stop

GET FAMILIAR WITH A NEXT.JS PROJECT STRUCTURE



contains the compiled version of your application (important for later deployment)

libraries, we build our app upon, e.g. react, ant design and also their sub-dependencies

static files that don't need to be processed or manipulated by your code, e.g. images, fonts, favicon

if you want to build a backend with Next.js, you can do so in you api folder - we won't cover this in the course

all your pages go here (e.g. index.js: your homepage) - the most important folder!

a component that wraps around your individual pages, e.g. for layout building

file is used to augment the HTML document structure

some CSS

config files for advanced configurations

a manifest file that contains metadata about your project, e.g. scripts you can run from CLI, dependencies

JSX

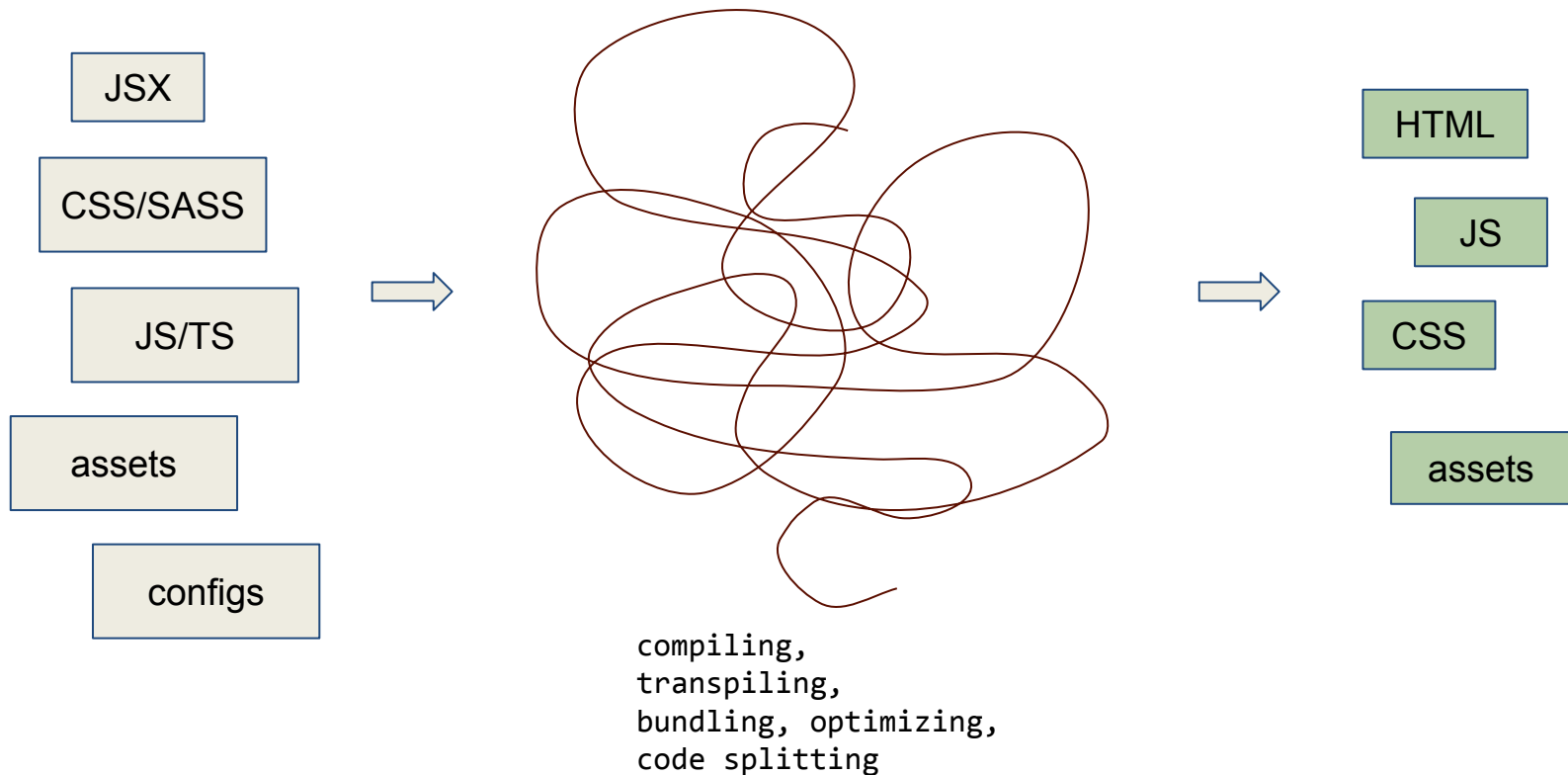
- **JSX (JavaScript XML)** is a syntax extension for JavaScript, commonly used with React to describe what the UI should look like. It allows you to write HTML-like code within your JavaScript code, making it more readable and intuitive to set up component structures. With JSX, you can also embed expressions inside `{ }` which will be evaluated to display dynamic data.

```
const element = <h1>Hello, world!</h1>;
```

```
const greeting = <h1>Hello, {name}!</h1>;
```

- JSX must be converted into standard JavaScript that browsers can understand. Don't worry, Next.js does handle this for you under the hood.

BEHIND THE SCENES OF REACT



A LITTLE CLEANUP

- in `index.js`, remove the `<main>` HTML element and all its children elements
- also remove unused font handling

LET'S EXPERIMENT ..

- in index.js, write some basic HTML just to get a feeling ..

```
import Head from 'next/head'

export default function Home() {
  return (
    <>
      <Head>
        <title>Create Next App</title>
        <meta name="description" content="Generated by create next app" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <div>
        <h1>
          Hello World!
        </h1>
      </div>
    </>
  )
}
```


React Components

SPLIT YOUR APP INTO COMPONENTS

- now create a new small component and split your app into smaller parts:
 - create a folder “components” under “src”
 - create a file “hello.js”
 - create a component named “Hello” and export it

```
export default function Hello() {  
  return (  
    <div>  
      <h1>  
        Hello World!  
      </h1>  
    </div>  
  )  
}
```

- now we can import and display this component from **everywhere else** in our app

```
import Head from 'next/head'  
import Hello from '../components/hello';  
  
export default function Home() {  
  return (  
    <>  
      <Head>  
        <title>Create Next App</title>  
        <meta name="description" content="Generated by create next app" />  
        <meta name="viewport" content="width=device-width, initial-scale=1" />  
        <link rel="icon" href="/favicon.ico" />  
      </Head>  
      <Hello />  
    </>  
  )  
}
```

Props
(=Properties)

PROPS IN REACT

- **props** (short for "properties") are a way of passing data (strings, objects, arrays, functions,..) **from parent to child** components. They act as the "inputs" for a component, allowing it to be reused in different contexts with different data.

```
import Head from 'next/head'
import Hello from '../components/hello';

export default function Home() {
  return (
    <>
      <Head>
        <title>Create Next App</title>
        <meta name="description" content="Generated by create next app" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <Hello name={"Rike"} />
    </>
  )
}
```

I'm the parent component! I can embed multiple child components and pass things as props!

*I'm the child component! I can handle things that were passed to me as props.
.. Btw. I can embed children myself ;)*

```
export default function Hello({ name }) {
  return (
    <div>
      <h1>
        Hello World, {name}!
      </h1>
    </div>
  )
}
```

Styling

STYLING IN REACT

- (un)fortunately there are several ways to include styling (CSS) into your React app
 - **The classic way:** Class-name based styling with CSS

```
1 import React from "react"
2 import './style.css'
3
4 function myComponent(){
5     return(
6
7         <p className="paragraph-text">ClassName Styled Text</p>
8     )
9 }
```

```
1 .paragraph-text{
2     font-weight: bold;
3     color: beige;
4 }
```

STYLING IN REACT

- **Quick and dirty:** Inline Styling

```
1 function MyComponent(){  
2  
3   return <div style={{ color: 'blue', lineHeight : 10, padding: 20 }}> Inline  
4  
5 }
```

STYLING IN REACT

- **What Next.js suggests:** CSS Modules

.. also works nicely with LESS or SASS modules
.. tends to work better on bigger apps

STYLING IN REACT

- **Clean & professional:** Emotion or Styled Components

```
import styled from "styled-components";

// Styled component named StyledButton
const StyledButton = styled.button`
  background-color: black;
  font-size: 32px;
  color: white;
`;

function Component() {
  // Use it like any other component.
  return <StyledButton> Login </StyledButton>;
}
```

- .. tends to work good on bigger apps
- .. probably best if you build your own Design System

STYLING IN REACT

- all these approaches can be used standalone
(but then you would have to create your own Design System :/)
- or you use a React UI Library and use one of these approaches to refine / overwrite certain styles :)
- we'll use Ant Design!



- let's dive into their documentation
- every component type you want to use, must be imported in your js file, e.g.

```
import { Layout, Space } from 'antd';
```

Other nice Libraries:

- [Bootstrap](#)
- [Material Design](#)
- [Tailwind](#)
- [Foundation](#)
- [Bulma](#)
- [Semantic UI](#)
- [Pure.CSS](#)
- [Skeleton](#)
- [Milligram](#)

YOUR BASE LAYOUT

- put it in `_app.js` because we'll need it on every subpage!

```
export default function App({ Component, pageProps }) {  
  return <Layout style={{ minHeight: '100vh', maxWidth: 1200, margin: "0 auto" }}>  
    <Header style={{ padding: 0, textAlign: 'center' }}>  
      lorem ipsum  
    </Header>  
    <Content style={{ padding: '0 50px', marginTop: '20px' }}>  
      <div style={{ padding: 24, minHeight: '100vh' }}>  
        <Component {...pageProps} />  
      </div>  
    </Content>  
    <Footer style={{ textAlign: 'center' }}>My Footer ©2023 Created by Me</Footer>  
  </Layout>  
}
```

Pages & Routing

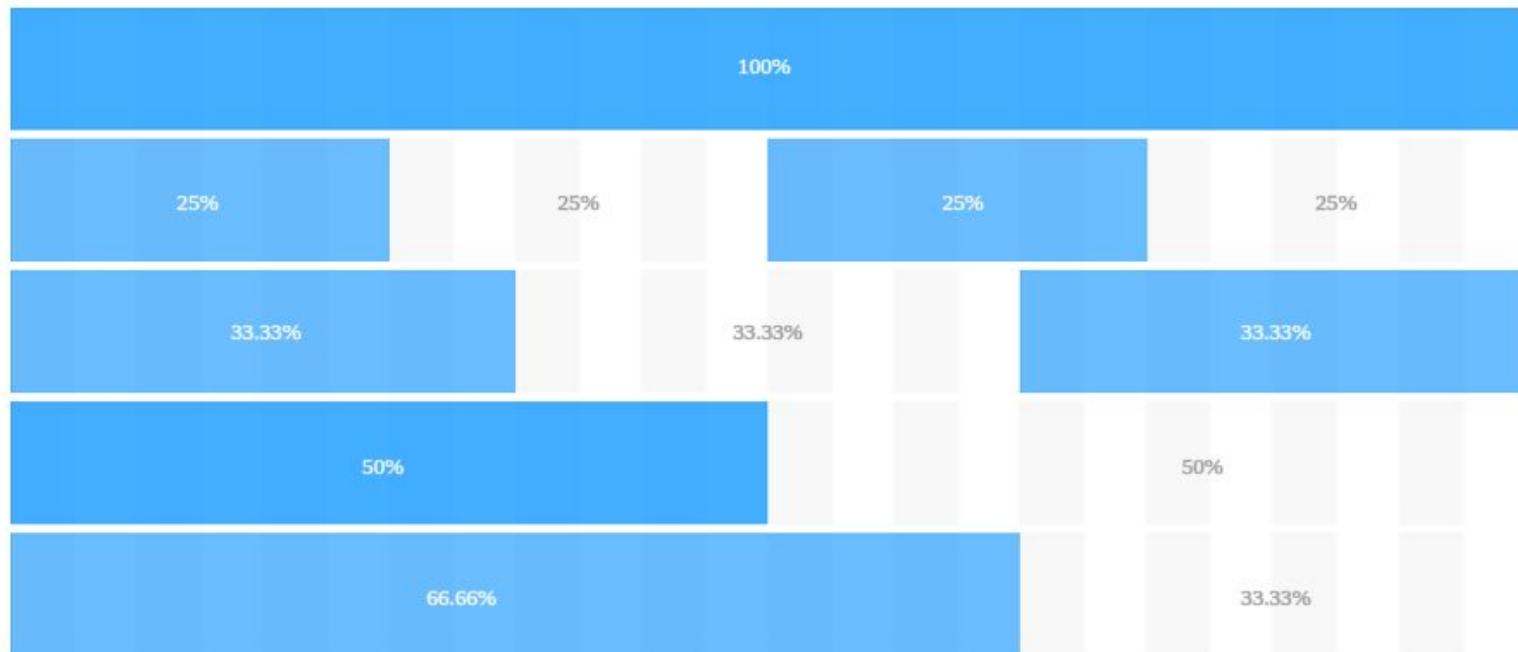
CREATING A NAVIGATION & SUB PAGES



- in the Header
create a Menu with
at least two items
“/about” and also
back to homepage
to “/”
- in the pages
folder create a
new subpage
about.js with some
blind text

LET'S BUILD A GRID SYSTEM IN INDEX.JS

Design concept



```
<Col xs={{ span: 24 }} sm={{ span: 12 }} lg={{ span: 6 }}>
```

```
  Testi
```

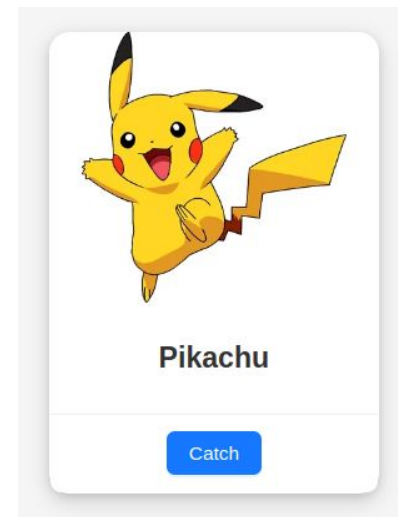
```
</Col>
```

LET'S EMBED SOME TYPICAL WEB COMPONENTS



Replace string “Testi”
with a Card-like
design, using Ant
Design components!

Move this to an own
component called
pokemon.js! (see what
we did with hello.js)



Hooks

Hooks

- Hooks are built-in functions to enhance your components, the most important ones:
 - **useState**: Lets you add state to functional components
 - **useEffect**: Lets you perform side effects in functional components

→ always introduce them **before** return-statement!

how to use useEffect:

```
useEffect(() => {  
  alert("Hello!")  
}, []);
```

how to use useState:

```
const [isVisible, toggleModal] = useState(false);
```

in JSX we'll trigger the function somewhere...

```
<Button onClick={() => toggleModal(true)}>Open Modal</Button>
```

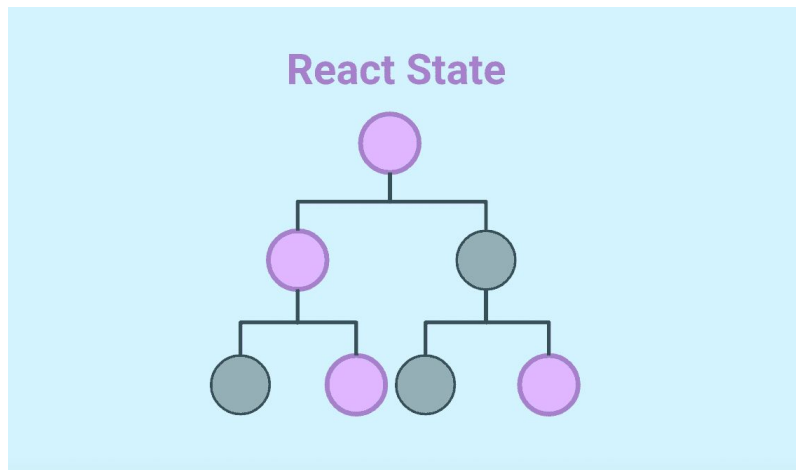
LET'S TRY IT OUT

```
initial page  
$.js) load,  
a simple Modal,  
that says "Hello!"
```

State management

STATE MANAGEMENT

- **State** is like a container that holds data or information about a component at any given point in time. Imagine it as the memory of a component. When this memory or data changes, the component updates or redraws itself to reflect those changes.
- State in a parent component can be passed down as props to its children components.



OPEN AND CLOSE A MODAL WHEN CLICKING ON "CATCH"



Embed a simple modal
in `index.js`, but only
if "Catch" was clicked

SOME THOUGHTS ON STATE..

- If you find yourself passing down state through many layers of components (known as "prop drilling")
- When multiple components, which might not be directly connected, need access to the same state
- if you have complex state transitions

..Use a state management library such as

- **zustand**
- **recoil**
- **jotai**

Do yourself a favor and don't use Redux!

Data fetching

USE A REST API

- A REST API is a set of rules and conventions for **building and interacting with web services** that allow you to create, read, update, and delete data using standard HTTP methods.
- if you're a data-heavy company/institution you might consider providing your data for others to examine or use
- and there are many fun APIs to use for free, e.g. <https://pokeapi.co>
- check out public APIs more:
<https://github.com/public-apis/public-apis>
- if you feel adventurous, use another API (that allows CORS) :)

FETCH ALL POKEMONS

- set a state that will “hold” all pokemons (initially empty)
- create function that will fetch the first 151 Pokemons from the public API and convert to a JSON
- (log your data)
- the */pokemon* endpoint only provides the names of the Pokemon, so we need to fetch detailed info “one by one” with 151 more calls

```
const getAllPokemons = async () => {  
  const res = await fetch("https://pokeapi.co/api/v2/pokemon?limit=151")  
  const data = await res.json()  
  const detailedData = await Promise.all(data.results.map(async (pokemon) => {  
    const res = await fetch(`https://pokeapi.co/api/v2/pokemon/${pokemon.name}`)  
    return await res.json();  
  }));  
  setAllPokemons(detailedData);  
}
```

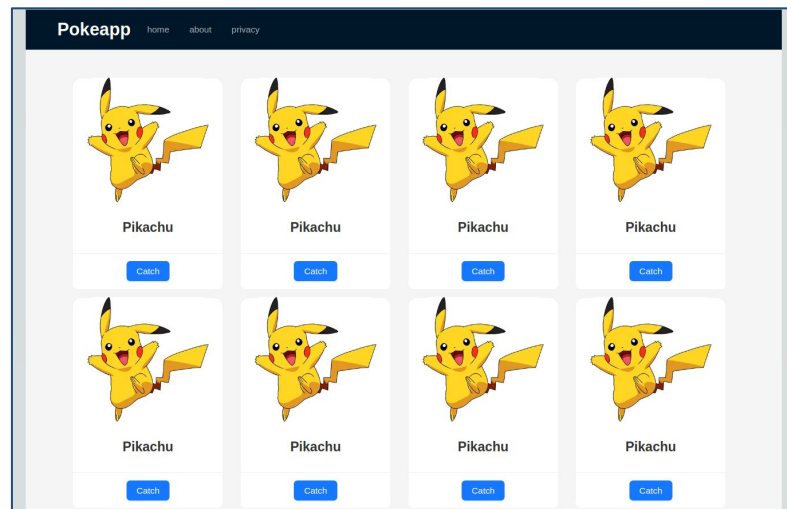
- all Pokemon details should be fetched, when index.js loads

```
useEffect(() => {  
  getAllPokemons()  
}, [])
```

DISPLAY ALL POKEMONS

- start with mapping through all pokemons and rendering a PokemonCard for every pokemon

```
<Row gutter={[12, 12]}>
  {allPokemons?.map(pokemon => <Col xs={{ span: 24 }} sm={{ span: 12 }} lg={{ span: 6 }}>
    <PokemonCard toggleModal={toggleModal} />
  </Col>)}
</Row>
```



- oh so many Pikachus! :(
- we have to tweak our PokemonCard a little bit, so it can receive dynamic Pokemon data

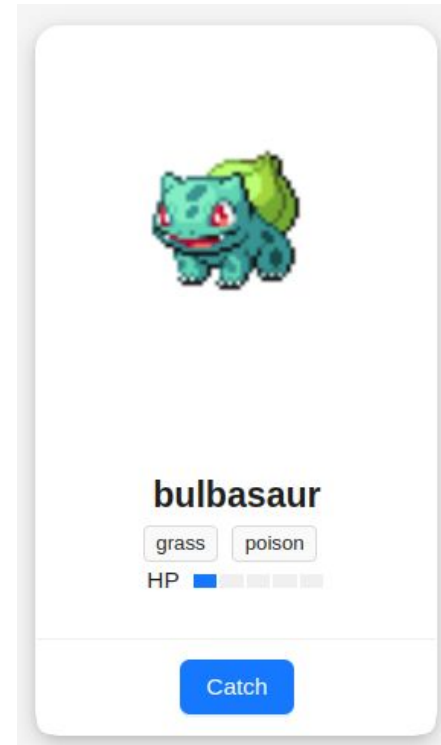
```
<PokemonCard pokemon={pokemon} toggleModal={toggleModal} />
```

ADAPT POKEMONCARD

- our PokemonCard will receive additional prop “pokemon”
- let’s replace the static string “Pikachu” with the dynamic value of key *pokemon.name* and the image url with the dynamic value of key *pokemon.sprites.front_default*



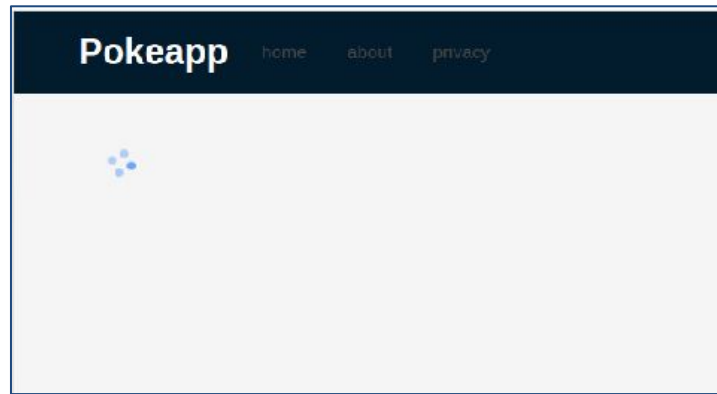
Add more info about
Pokemons as you want



A USABILITY IMPROVEMENT..



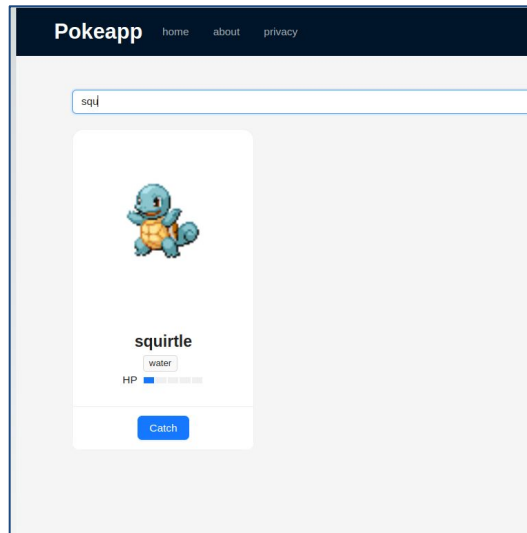
While loading the
Pokemon list, render a
Spinner



ANOTHER USABILITY IMPROVEMENT..

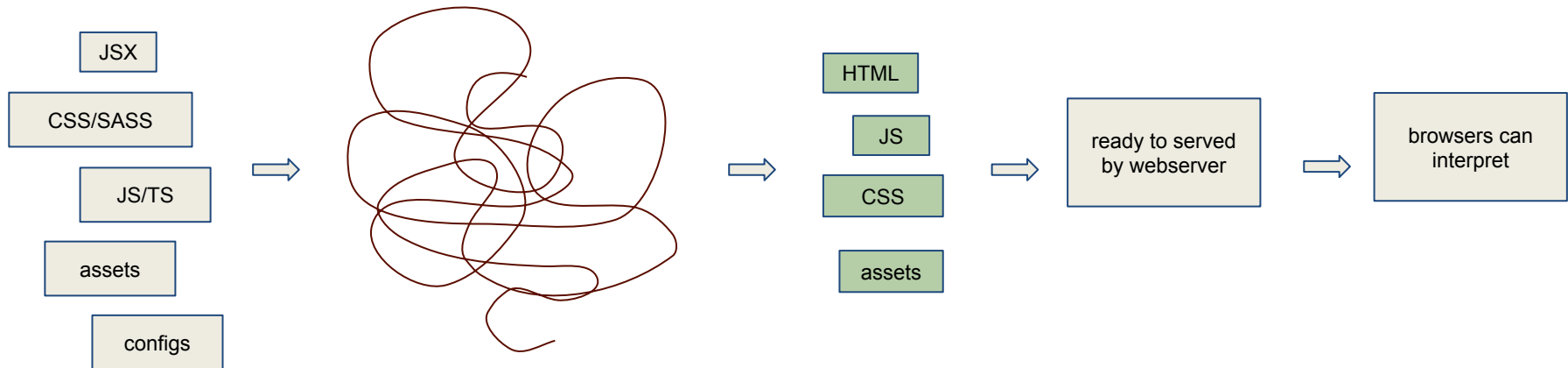


Above the Pokemon list,
implement a search bar,
that filters Pokemon, if
substring matches



HOW TO CREATE A PRODUCTION BUILD

A "**production build**" is the final version of a web application that's prepared for real users on the internet and ready to be deployed (to be put on a webserver to be requested): stripped of any tools and features only necessary for developing, optimized in performance, checked and tested, minimized and compiled to plain HTML, CSS, JS.



WHAT ABOUT UNIT TESTING?

Unit testing is key for React devs because it helps catch bugs early, ensures components work as expected, and makes refactoring safer. Plus, it gives peace of mind when adding new features!



- install needed packages

```
npm install --save-dev jest @testing-library/jest-dom @testing-library/react
```

- write a unit test (based on components works well)

A SAMPLE UNIT TEST

```
import React from 'react';
import { render } from '@testing-library/react';
import PokemonCard from "../pokemon"
import '@testing-library/jest-dom'

describe('<PokemonCard />', () => {
  const mockPokemon = {
    name: "Mew",
    sprites: {
      front_default: "path-to-image.jpg"
    },
    types: [
      { type: { name: "Electric" } }
    ],
    stats: [
      { base_stat: 100 }
    ]
  };

  it('displays the pokemon name', () => {
    const { getByText } = render(<PokemonCard pokemon={mockPokemon} />);
    expect(getByText('Mew')).toBeInTheDocument();
  });
});
```


SOME WORDS ABOUT TYPESCRIPT

- TypeScript ensures type safety, this means fewer errors related to incorrect or unexpected data types, e.g. when passing props

Without TypeScript:

```
javascript Copy code  
  
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
// Usage  
<Welcome name="React" />;
```

With TypeScript:

```
typescript Copy code  
  
type WelcomeProps = {  
  name: string;  
};  
  
function Welcome(props: WelcomeProps) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
// Usage  
<Welcome name="React" />;
```

HOW TO PROCEED..

But Rike, I've seen Wordpress and Jimdo Pages, that were more pretty and useful! :(

→ sure, there are tools that help you with popular use cases such as landing pages, webshops, blogs ..

→ but mastering frontend development, means *freedom!*

- Build more complex apps: <https://github.com/florinpop17/app-ideas>
- Build your own app idea!
- Become a frontend developer ;)

YOUR FINAL QUESTIONS!

YOU MADE IT!!

- final code base:
<https://github.com/gitfrosh/pokeapp>

RESOURCES

- <https://react.dev/> (official)
- <https://epicreact.dev/> (best)
- <https://handsonreact.com/> (best free)

UNDERSTANDING REACT

- Watch the awesome React documentary!
<https://www.youtube.com/watch?v=gmp0istg5xo>

See you soon!