Frontend workshop

Symbols explained

- Please open the link and answer the quiz question.

- Please make sure to remember the section marked with this as it will be important later as well.

Important info

- You can win a prize if you do the challenge at the end of the doc. 🙂
- 10 minutes break in each hour (can be negotiated)
- Lunch break: 12:00-13:00 Luxembourg/Amsterdam time - 13:00-14:00 Beirut/Istanbul time
- Join the discord server
    - Please ask your questions on the #sd-support channel.

General advices

- Please read explanations carefully and open the links and only skip them if you are comfortable in the given topic.
- If you don't understand something try to read the related link or google it if no link is available. If it doesn't clarify for you, try to ask your fellow students.
- If you are blocked for more than 5 minutes, reach out to the facilitators to get help.

Prerequisites

- Firefox or Chrome installed on your laptop
- Discord server

Recommended Option - We suggest doing the workshop on your own computer and using VS Code. In this case you need to do the steps in VS Code setup section.

Option B - with an easier setup is to follow Cloud9 setup section.
This can be a good option if you have a laptop with a memory < 8GB or a poor internet connection.
Another advantage is that most of the tools are pre-installed and the environment is tested by the facilitators.
The disadvantage of this approach is losing the correct syntax highlighting and some parts require extra setup.

VS Code setup

1. Install the latest Visual Studio(VS) Code
    a. Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js.
    b. VS Code intro videos.
    c. https://code.visualstudio.com/download
2. (recommended) Install NVM (Node Version Manager) or FNM (Windows)
    a. If you are using Windows please follow these steps: https://nodejs.org/en/download/package-manager
    b. `nvm` allows you to quickly install and use different versions of node via the command line.
        i. https://github.com/nvm-sh/nvm
3. Install `Node.js v20.12.2 LTS or higher version`
    a. Option 1. Using NVM:

```
nvm install v20.12.2
```

b. https://nodejs.org/en/download/package-manager
c. Option 3. Downloading node from the official website.

4. Verify Node and NPM version
    a. To verify that node and npm (Node Package Manager) are installed, run the following commands on your terminal

```
# verifies the right Node.js version is in the environment
node -v # should print `v20.12.2`

# verifies the right NPM version is in the environment
npm -v # should print `10.5.0`
```

Initialise react project

If you want to build a new app or a new website fully with React, we recommend picking one of the React-powered frameworks popular in the community. (For this tutorial we picked Next.JS)

You can use React without a framework, however we've found that most apps and sites eventually build solutions to common problems such as code-splitting, routing, data fetching, and generating HTML. These problems are common to all UI libraries, not just React.

By starting with a framework, you can get started with React quickly, and avoid essentially building your own framework later.

1. Create a new react project

You can ideally execute this command in a folder where you store other development projects. Within Amazon we usually create a workspace folder in our user folder and put their all of our projects.

Copy paste it into the terminal:

```
npx create-next-app@latest
```

This command initialises a react project with Next.js framework. Press enter to proceed

```
Need to install the following packages:
create-next-app@14.2.3
Ok to proceed? (y)
```

2. Select the following options:

Please select the ones shown as green and underlined. For the `import alias` use `@`.

```
✔ What is your project named? … pokedex-ui-workshop
✔ 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
✔ What import alias would you like configured? … @/*
```

[What are the benefits of using a static code analyzer tool like ESLint?]

Next.js lets us customize the import alias, it makes it easier and cleaner to import modules. More info.
The **App Router** is a file-system based router that uses React's latest features. More info.

3. Open the directory
    a. Open the directory of the react project that you created
    b. Learn more about cd command here. (Hints on how to use it.)

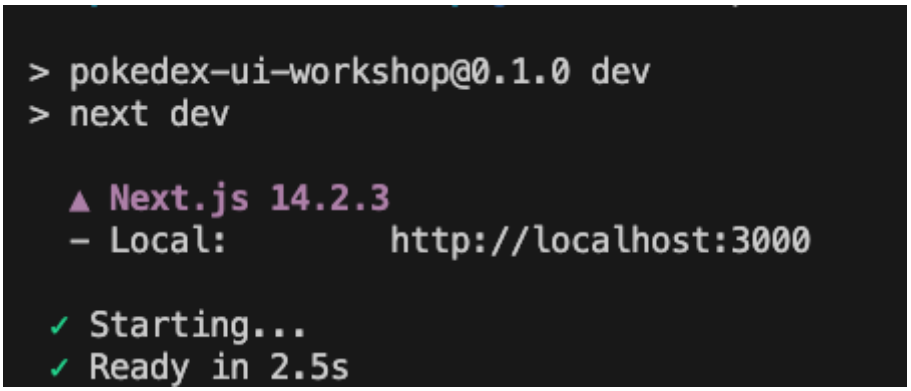
```
cd pokedex-ui-workshop // ...Or the name you gave your project...
```

4. Launch the react app

    a. 

Please do not close the terminal or kill the command as Next.js framework recompiles the project in the background whenever a change is made.

```
npm run dev
```



5. Preview the app on localhost

Open the link mentioned above (http://localhost:3000) in your favourite browser.

6. Install React Bootstrap
    a. Bootstrap is an HTML, CSS and JS library that focuses on simplifying the development of informative web pages by providing ready to use UI components.
    b. React-Bootstrap replaces the Bootstrap JavaScript. Each UI component has been built from scratch as a true React component.
    c. Open a new terminal window or terminate the current process running. Use `ctrl+c` or `cmd+c` (MAC) to terminate.
    d. Install both package:

```
npm install react-bootstrap bootstrap
```

7. Change current `css` setup
    a. Within `src/app/layout.tsx` file replace the `import "./globals.css";` with `import 'bootstrap/dist/css/bootstrap.min.css';`
    b. Expected outcome.

8. Replace the whole content of `src/app/page.tsx` with this:

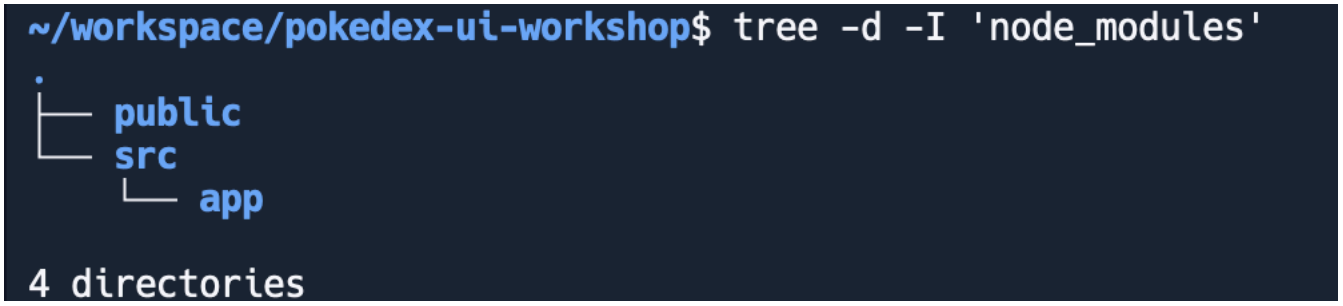
```
export default function Home() {
 return (
  <>Hello World!</>
 );
}
```

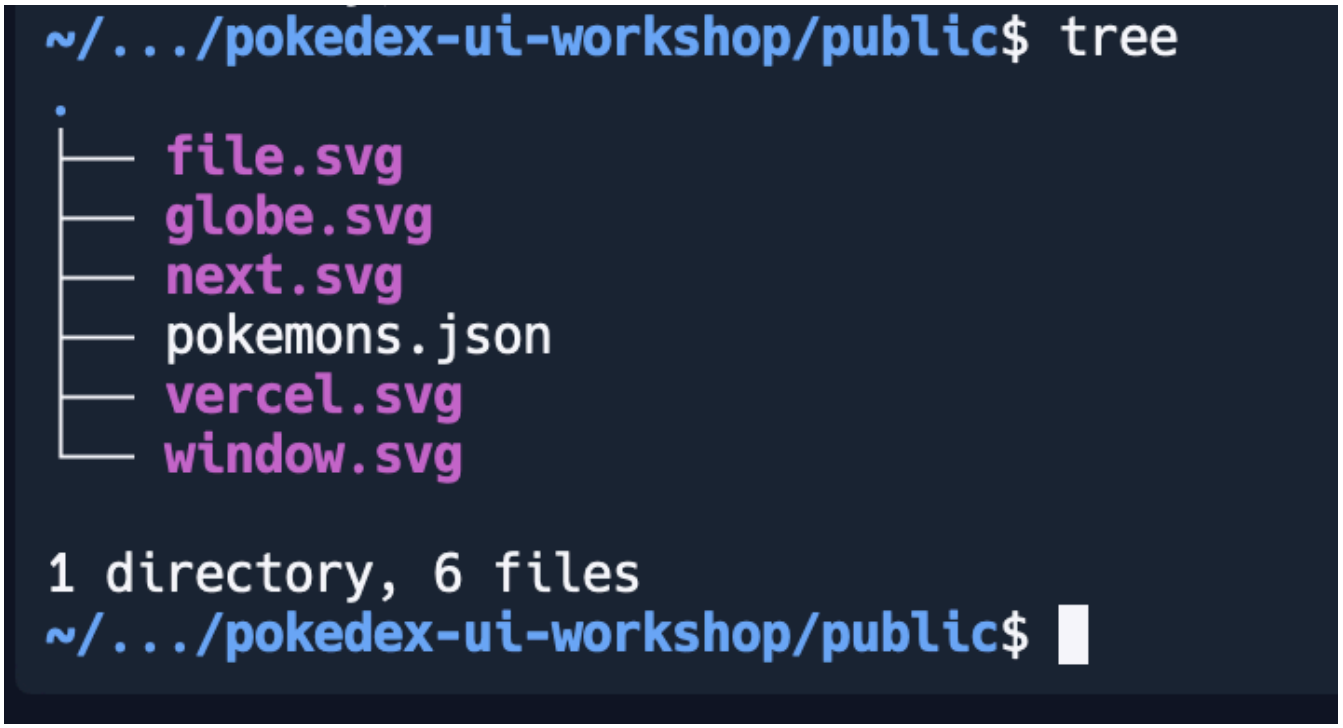
9. Download test data to the `public` folder.
    a. If you are using windows install curl
    b. Please download the following file and save it to the `public` folder as `pokemons.json`

```
# Assuming that you are currently in the pokedex-ui-workshop folder
cd public
curl -O
https://gitlab.com/university-engagement-program/pokedex-ui-workshop/-/raw/main/public/
pokemons.json
```

10. The project structure looks like this in this stage

It contains the default files plus the `pokemons.json`, that we have downloaded earlier.



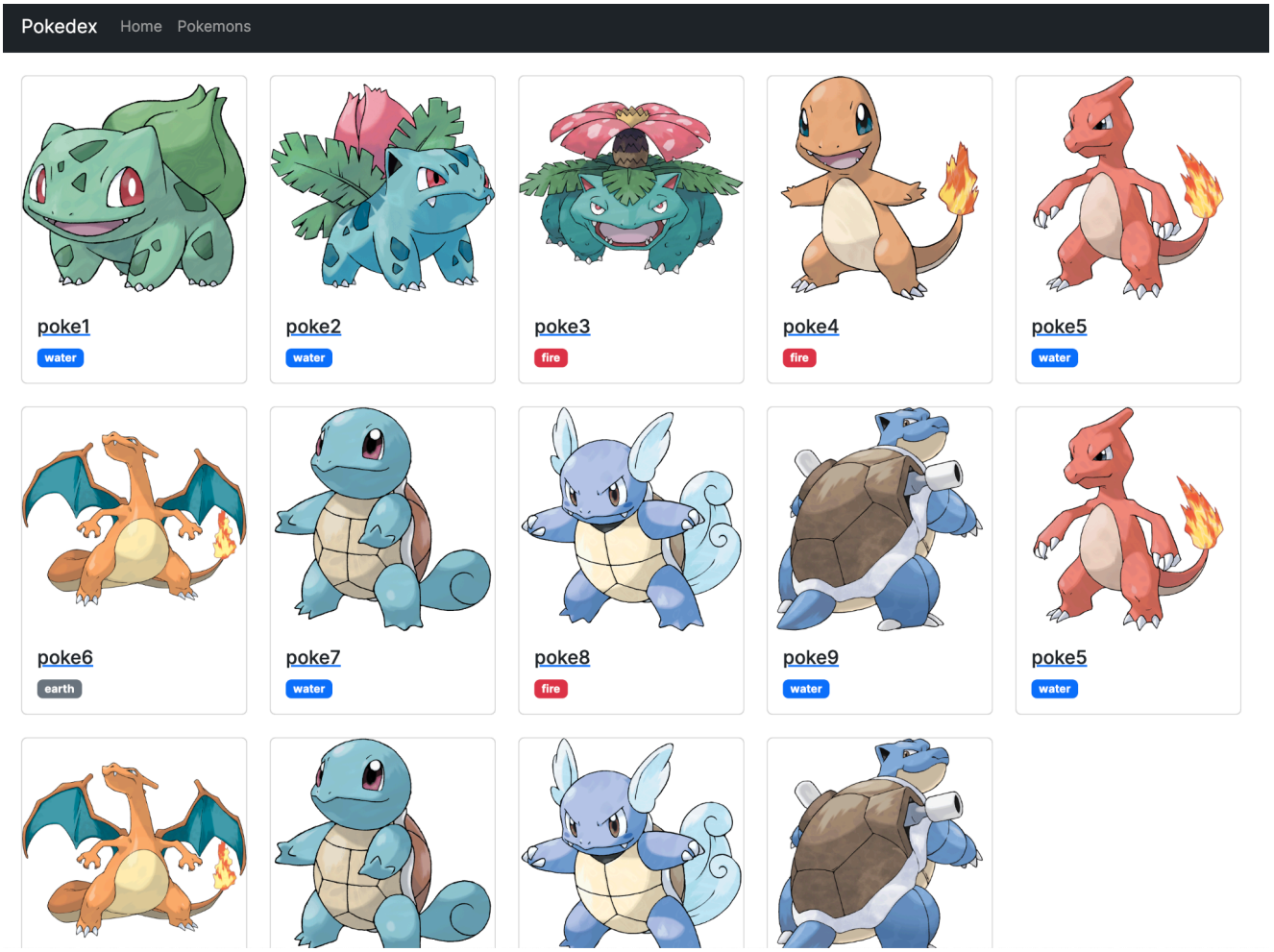


11. We are all set to kick off the development. 🎉

Create the pokemon listing page

The goal

The aim is to create a page to list all pokemons as shown in the figure



The page consists of three main components:

1. A navigation bar
2. A tiling of pokemon cards
3. A pokemon card that renders the image, title and type of the pokemon.

The Setup

Run the following commands in the command line or use file explorer or development editor.

1. Create a “components” folder under “src” to keep the client-side react components

```
# Assuming that you are currently in the pokedex-ui-workshop folder
cd src
mkdir components
cd components
```

2. Create a new “model” folder under “src” directory to model files.

```
# Assuming that you are currently in the pokedex-ui-workshop folder
cd src
mkdir model
cd model
```

3. Create a TypeScript file called pokemonCard.ts file in src/model folder to define the main data type that will be used in the website to represent pokemon data.
    a. In this case we are following the camelCase naming convention. There are other ones like snake_case. There is usually a recommended approach for a programming language, but in most of the cases it is based on personal preference. However it is strongly recommended to stick with one naming convention for the same project.
    b. `.tsx` extension is used as it is not a usual typescript file. TSX is a syntax extension for TypeScript that lets you write HTML-like markup inside a TypeScript file.
4. Use the PokemonCard interface definition below in the file.

```
export default interface PokemonCard {
   pokemonNumber: number;
   mainImage: string;
   pokemonName: string;
   pokemonType: string[];
}
```

The Navigation Bar

The navigation bar is one of the basic components in almost all websites. We will build one only for illustrative purposes. It will not be functional for this project.

1. Create a react component file under src/*components* folder with a name pokeNavBarComp.tsx

In this project we will develop only client-side components for simplicity. We use “use client”; directive at the beginning of the file to mark the components as client side.

2. We will use react bootstrap navigation components: https://react-bootstrap.netlify.app/docs/components/navbar Please navigate to this link and have a quick look at different navigation options.
3. The navbar will contain a main app link that navigates to “/” home page and other links (supposedly to other pages) in our example, it will also be called Home and navigate to “/” home page.
4. The content of the pokeNavBarComp.tsx is below, make sure to replace the current content.

```
"use client";

import Nav from 'react-bootstrap/Nav';
import Navbar from 'react-bootstrap/Navbar';
import { Container } from 'react-bootstrap';

export default function PokeNavBarComp() {
   return (
```

```
        <>
            <Navbar bg="dark" data-bs-theme="dark">
                <Container>
                    <Navbar.Brand href='/'>Pokedex</Navbar.Brand>
                    <Nav className="me-auto">
                        <Nav.Link href='/'>Home</Nav.Link>
                    </Nav>
                </Container>
            </Navbar>
        </>
    );
}
```

5. The component consists of Navbar, Container, Nav and Nav.Link react bootstrap components.
6. We don't need to go into the details, but this provides a ready made navigation bar including required styling.
7. Note that all links are relative (so that it does not navigate away to other pages).
8. Add the navigation bar to app/page.tsx and replace the content with this:

```
"use client";

import PokeNavBar from '@/components/pokeNavBarComp';

export default function Home() {
 return (
   <>
     <PokeNavBar></PokeNavBar>
   </>
 );
}
```

The output is below. You can find different styling options in https://react-bootstrap.netlify.app/docs/components/navbar

The Card Component

The card component renders the individual pokemons.

In our current design, we choose to use the bootstrap react card component, but note that you can design your own custom card with CSS and HTML.

1. Create a new react component under src/*components* folder with a name *pokemonCardComp.tsx*

2. We will use react bootstrap card component:https://react-bootstrap.netlify.app/docs/components/cards Please navigate to the link and investigate different features of Card components briefly.
3. A pokemon card will have an image, a name field and type.
4. With the bootstrap Card we define them using *Card.Img, Card.Title* and *Card.Text* components inside a Card component.
5. We also wrap each card with HTML a tag to be able to navigate to the detail page of the pokemon.
6. The content of the *pokemonCardComp.tsx* is below:

```
"use client";

import PokemonCard from "@/model/pokemonCard";
import { Card } from "react-bootstrap";

interface PokemonCardCompProps {
   pokemon: PokemonCard;
}

export default function PokemonCardComp(props: PokemonCardCompProps) {

   const pokemonUrl = `/pokemon/${props.pokemon.pokemonNumber}`;

   return (
       <a href={pokemonUrl}>
           <Card>
               <Card.Img variant="top" src={props.pokemon.mainImage} />
               <Card.Body>
                   <Card.Title>{props.pokemon.pokemonName}</Card.Title>
               </Card.Body>
           </Card>
       </a>
   );
}
```

7. The PokemonCardComp gets PokemonCardCompProps as a property. In react one well known approach to pass data to components is using component properties.
8. The PokemonCardCompProps has a single field pokemon with type PokemonCard type *(Note that we defined in src/models/pokemonCard.ts)*
9. The PokemonCardComp will create a detail page url using the pokemonNumber field.
    a. const pokemonUrl = `/pokemon/${props.pokemon.pokemonNumber}`;
10. It will use mainImage to render the image of the pokemon
11. It will use pokemonName to render the title of the pokemon.
12. To see how this component will be used, we will add a component to *app/page.tsx*
13. We first create a test pokemon data of type PokemonCard *(Note that although we did not explicitly define type Typescript automatically mapped the type as fields matches)*
14. Finally, we pass testData to pokemon props of PokemonCardComp.
15. See the latest state of app/page.tsx code:

```
import PokemonCardComp from "@/components/pokemonCardComp";
import PokeNavBar from "@/components/pokeNavBarComp";

export default function Home() {
 const testData = {
   pokemonNumber: 1,
   pokemonName:"poke1",
   pokemonType:["Water"],

mainImage:"https://raw.githubusercontent.com/HybridShivam/Pokemon/master/assets/images/001.png"
 };

 return (
   <>
     <PokeNavBar></PokeNavBar>
     <PokemonCardComp pokemon={testData}></PokemonCardComp>
   </>
 );
}
```

16. Check the output in the preview/browser window. It should look like below.

The Pokemon Type Badge Component

The pokemon types are rendered as text without any style according to our current pokemon card design. However, we want to customise the style based on the type of the pokemon. To achieve this, we will create a new component to render pokemon types.

1. Create a new react component in the src/*components* folder with a name *pokemonTypeBadgeComp.tsx*
2. We will use react bootstrap badge component: https://react-bootstrap.netlify.app/docs/components/badge Please navigate to the link and investigate different features of Badge components briefly.

3. We will choose a specific badge based on the type of the pokemon *(Note that this is not a semantically correct use of badge, but we use it for simplicity. Otherwise, we need to define our own CSS styles for each pokemon type)*
4. The content of PokemonTypeBadgeComp is below:

```
"use client";

import { Badge } from "react-bootstrap";

interface PokemonCardCompProps {
   pokemonTypes: String[];
}

export default function PokemonTypeBadgeComp(props: PokemonCardCompProps) {
   return (
       <>
           {props.pokemonTypes?.map((pokemonType, index) => {
               if (pokemonType === "Water") {
                   return <Badge key={index}
bg="primary">{pokemonType}</Badge>
               } else if (pokemonType === "Fire") {
                   return <Badge key={index} bg="danger">{pokemonType}</Badge>
               } else if (pokemonType === "Grass") {
                   return <Badge key={index}
bg="success">{pokemonType}</Badge>
               } else if (pokemonType === "Electric") {
                   return <Badge key={index}
bg="warning">{pokemonType}</Badge>
               } else {
                   return <Badge key={index}
bg="secondary">{pokemonType}</Badge>
               }
           })}
       </>
   );
}
```

5. The *PokemonTypeBadgeComp* gets a list of strings in its props.
6. For each string we render a different Badge component..
7. We define the style (bg=””) of the Badge component using the pokemonType.
8. In addition to the bg property, we need to set the key property because it is required to identify different components.
9. We will update our PokemonCarComp component to use our new PokemonTypeBadgeComp. Note that previously we were not rendering pokemonTypes.

10. We add *PokemonTypeBadgeComp* in the Card.Body component inside PokemonCardComp. It will be rendered as part of the content of the Card component.
11. The latest state of the src/components/pokemonCardComp.tsx is below:

```
"use client";

import PokemonCard from "@/model/pokemonCard";
import { Card } from "react-bootstrap";
import PokemonTypeBadgeComp from "./pokemonTypeBadgeComp";

interface PokemonCardCompProps {
   pokemon: PokemonCard;
}

export default function PokemonCardComp(props: PokemonCardCompProps) {

   const pokemonUrl = `/pokemon/${props.pokemon.pokemonNumber}`;

   return (
       <a href={pokemonUrl}>
           <Card>
               <Card.Img variant="top" src={props.pokemon.mainImage} />
               <Card.Body>
                   <Card.Title>{props.pokemon.pokemonName}</Card.Title>
                   <Card.Text>
                       <PokemonTypeBadgeComp
pokemonTypes={props.pokemon.pokemonType} />
                   </Card.Text>
               </Card.Body>
           </Card>
       </a>
   );
}
```

12. Check the output in the preview/browser window. It should look like below:

13. Test how the background of the badge changes by setting different pokemonTypes such as “Fire”.

The Pokemon Tiling Component

We will create another component to tile the pokemon cards into the page. We will use react bootstrap layout components.

1. Create a new react component in the src/*components* folder with a name *pokemonsComp.tsx*
2. We will use react bootstrap layout grid components: https://react-bootstrap.netlify.app/docs/layout/grid Please navigate to the link and investigate different features of Grid layout components briefly.
3. The Row component gets the list of Col components and tiles them. The different tiling options are controlled with Row properties such as xs, md, lg
4. The final content of pokemonsComp.tsx is below:

```
"use client";

import Container from "react-bootstrap/Container";
import PokemonCard from "@/model/pokemonCard";
import { Row, Col } from "react-bootstrap";
import PokemonCardComp from "@/components/pokemonCardComp";


interface PokemonsCompProps {
   pokemons: PokemonCard[];
}

export default function PokemonsComp(props: PokemonsCompProps) {
   return (
       <Container className="pt-4 pb-4">
           <Row xs={1} md={3} lg={5} className="g-4">
               {props.pokemons.map((pokemon) => (
                   <Col key={pokemon.pokemonNumber}>
                       <PokemonCardComp pokemon={pokemon}/>
```

```
                    </Col>
                ))}
            </Row>
        </Container>
    );
}
```

5. The *PokemonsComp* component gets the list of PokemonCard as property with field pokemons.
6. We use Container component to define the borders of the containing component.
    a. pt-4 and pb-4 are Bootstrap utility CSS classes. pt stands for padding-top, and pb stands for padding-bottom CSS rule. 4 after dash defines the size. In this case, it is defined as 1.5rem. For other Bootstrap CSS utility classes check this link: https://www.w3schools.com/bootstrap4/bootstrap_utilities.asp
7. Row component uses xs, md and lg to defines the number of columns for each screen size. xs refers to small screens such as mobile phone, md is mid size such as tablet, lg is large such as laptop.
    a. g-4 defines the spacing between each column.

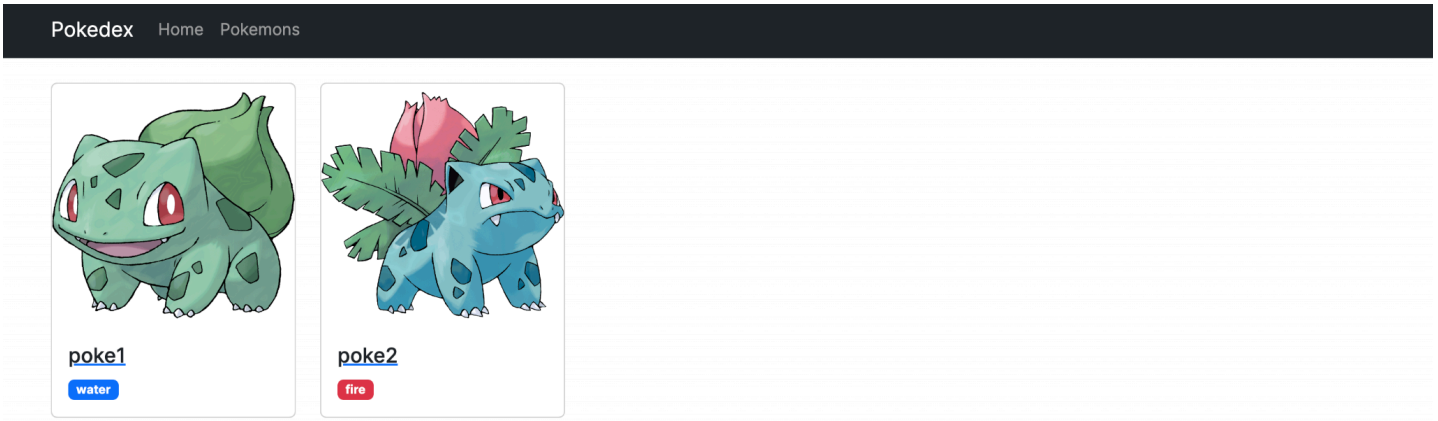
To experiment dynamic tiling based on the screen size, you can resize your browser. As the screen gets smaller, it should change to 3 column tiling, and finally 1 column tiling.

8. Note that each Col component contains a PokemonCardComp component.
9. We change home page in src/app/page.tsx to add PokemonsComp component as below:

```
import PokemonsComp from "@/components/pokemonsComp";
import PokeNavBar from "@/components/pokeNavBarComp";

export default function Home() {
 const testData = [
   {
     pokemonNumber: 1,
     pokemonName:"poke1",
     pokemonType:["Water"],
     mainImage:
"https://raw.githubusercontent.com/HybridShivam/Pokemon/master/assets/images/001.png"
   },
   {
     pokemonNumber: 2,
     pokemonName:"poke2",
     pokemonType:["Fire"],
     mainImage:
"https://raw.githubusercontent.com/HybridShivam/Pokemon/master/assets/images/002.png"
   }
 ];
```

```
  return (
    <>
      <PokeNavBar></PokeNavBar>
      <PokemonsComp pokemons={testData}></PokemonsComp>
    </>
  );
}
```

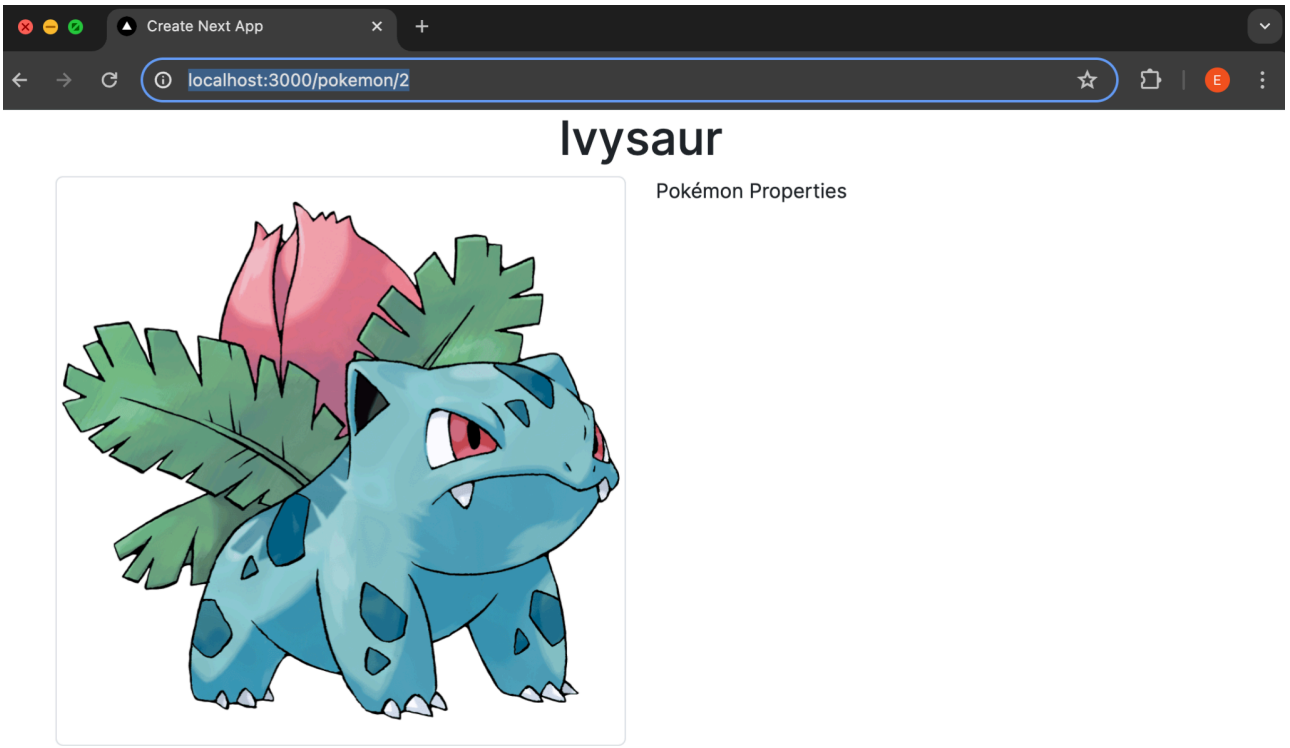
10. For testing purposes we create test data to visualise our changes (Later, we will read this data from backend)
11. To test tiling properly, you duplicate this data to test with more pokemons.
12. We replaced the PokemonCardComp component with PokemonsComp component and set pokemons to our new testData.
13. Check the output in the preview/browser window. It should look like below:



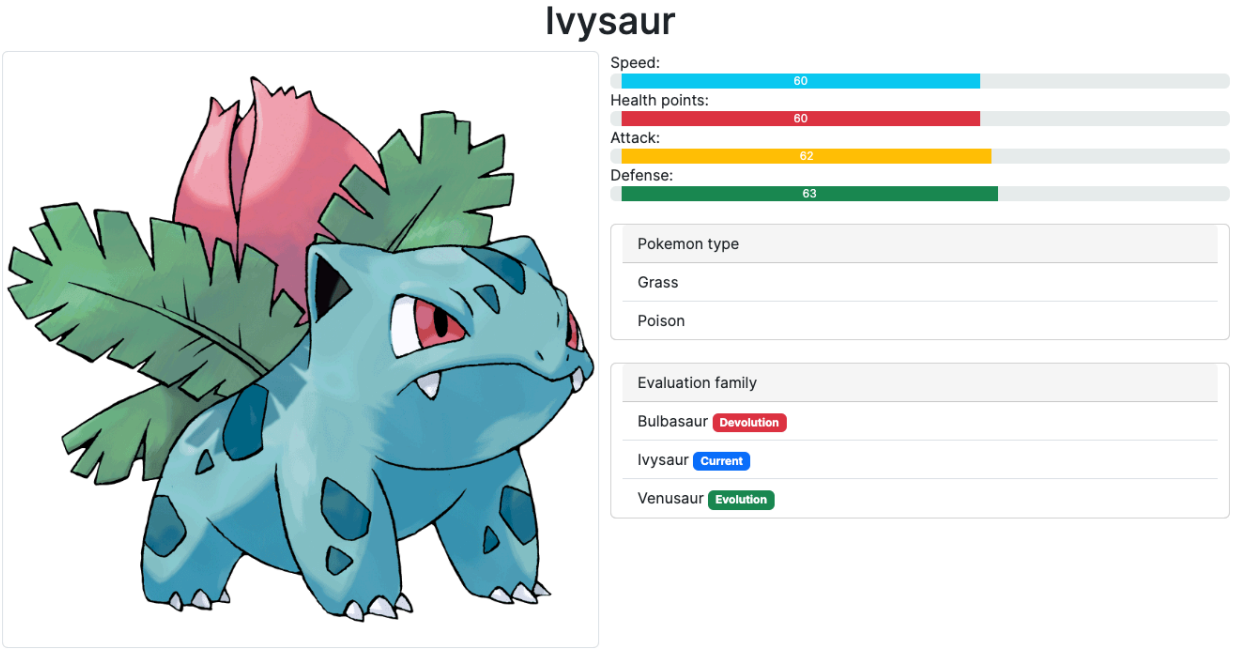
Create pokemon detail page

The Goal

In the following sections, we will create a basic Pokémon detail page.



Later if you have time you can enhance your solution and show the properties as well. More details in the Challenge section.



Basic setup

1. Creating a Pokemon interface
    a. Create a pokemon.ts file in the src/model folder.

Please create a typescript interface out of this json:

```
{
   "pokemonNumber": 7,
   "attack": 48,
   "defense": 65,
```

```
    "devolution": "",
    "evolution": "Wartortle",
    "evolutionFamily": [
        "Squirtle",
        "Wartortle",
        "Blastoise"
    ],
    "healthPoints": 44,
    "mainImage":
"https://pokemon-aub-awe-workshop.s3.eu-west-2.amazonaws.com/7/mainImage.png",
    "pokemonName": "Squirtle",
    "pokemonType": [
        "Water"
    ],
    "speed": 43
}
```


How would you do it?

2. Create the Pokemon page file
    a. Navigate to the `src/app` folder and create a folder called `pokemon`
    b. Within `pokemon` create a folder called `[pokemon_id]`
    c. Within the `src/app/pokemon/[pokemon_id]` folder create a file called `page.tsx`
    d. This folder structure serves as a router for Next.js. In short, whenever a url is invoked with the following address http://localhost:3000/pokemon/2, the Next.js framework knows that the content of `page.tsx` file needs to be loaded for the requestor.

```
# Assuming that you are currently in the src/model folder
cd ../app
mkdir pokemon
cd pokemon
mkdir "[pokemon_id]"
cd "[pokemon_id]"
```

3. Create the frame of the Pokemon page
    a. Please read the comments (green sections) carefully for explanations.

```
// Instructs React (Next.js) to run this code on the client side.
// Next.js by default would render this content on the server side where the
application is hosted.
'use client'



// This type is used to get the pokemon id from the url path
type Params = {
```

```
  params: { pokemon_id: string }
}

// Next.js passes the url parts which are defined between square brackets []
// to the function which renders the page.

// In our case http://localhost:3000/pokemon/2 is the URL.
// Where the 2 is the [pokemon_id] and passed as a parameter.
export default function PokemonPage({ params }: Params) {
  return (
      <>
          Pokémon not found
      </>
  );
}
```

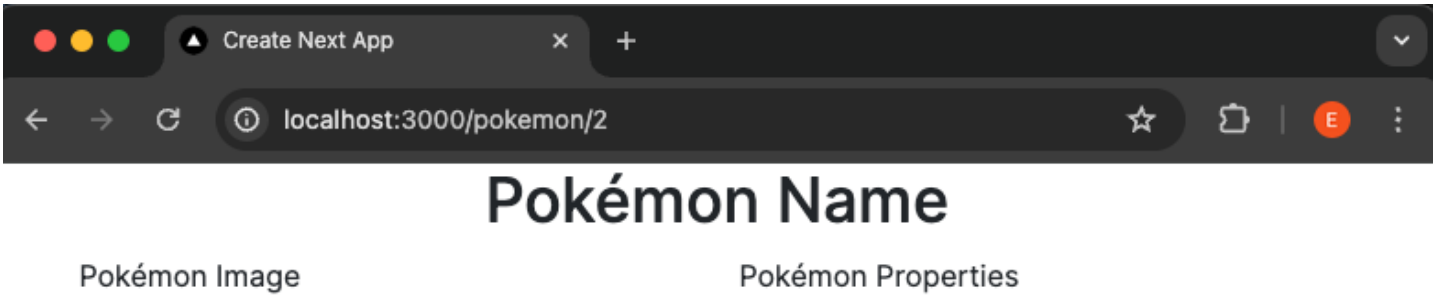
4. Setup a basic grid system from React Bootstrap
    a. A grid system is **a visual guide that is used to organize and align elements within a design**.
    b. Bootstrap's grid system uses a series of containers, rows, and columns to layout and align content. It's built with flexbox and is fully responsive.
    c. Place this code below `'use client'`

```
'use client'
import { Row, Col, Container } from 'react-bootstrap';
```

    d. Extend the PokemonPage function with the basic grid layout

```
export default function PokemonPage({ params }: Params) {
  return (
      <Container>
          <Row className="justify-content-md-center">
              <Col md="auto"><h1>Pokémon Name</h1></Col>
          </Row>
          <Row>
              <Col >
                  Pokémon Image
              </Col>
              <Col>
                  Pokémon Properties
              </Col>
          </Row>
      </Container>
  );
}
```

5. One step closer to finishing. 🙂 The page should look like this at the moment:



Adding Pokémon name and image

Extend the existing imports

a. The `Pokemon` is the interface that we have created.
b. useEffect and useState are React Hooks.
   i. useEffect lets your component connect to and synchronize with external systems. In our case we will use it to fetch the pokemons.json file and later to connect to the backend API.
   ii. useState lets a component "remember" information. In our case we will use it to store the information about the Pokémon.

```
'use client'
import Pokemon from '@/model/pokemon';
import { useEffect, useState } from 'react';
import { Row, Col, Container } from 'react-bootstrap';
```

Creating the Pokémon state

```
export default function PokemonPage({ params }: Params) {
   const pokemon_id = params.pokemon_id;
   //pokemon - A constant state variable which stores the pokemon information
and retains the data between renders.
   //setPokemon - A state setter function to update the variable and trigger
React to render the component again.
   const [pokemon, setPokemon] = useState<Pokemon>();
```

Fetching the test data

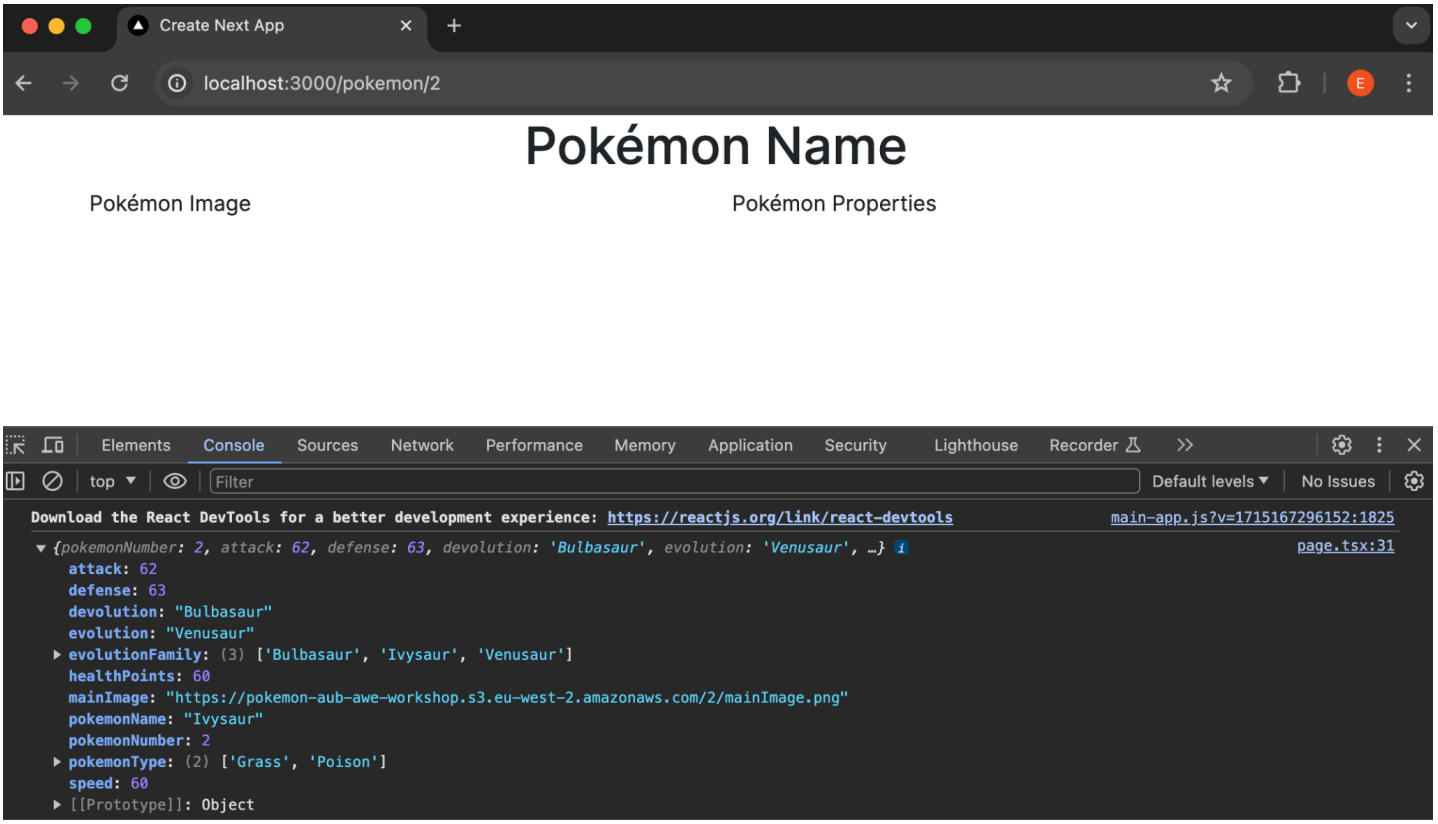
c. Previously you downloaded the required test data and put it inside the `public` folder.
d. Files within the `public` folder are accessible from the browser. E.g. the `pokemons.json` file can be requested with the following url: `http://localhost:3000/pokemons.json`
e. Within our React app we will use the fetch API to get the Pokémons.
f. Using the `fetch` within the useEffect is the right way to load this resource with React.

```
    const [pokemon, setPokemon] = useState<Pokemon>();

    useEffect(() => {
        const fetchData = async () => {
            const resp = await fetch('/pokemons.json');
            // Creating a Map out of the raw json
            const pokemons: Map<string, Pokemon> = new Map(Object.entries(await
resp.json()));
            const currentPokemon = pokemons.get(pokemon_id);
            setPokemon(currentPokemon);
            console.log(currentPokemon);
        };

        fetchData()
            // Making sure to log errors on the console
            .catch(error => {
                console.error(error);
            });
    }, []);
```

g. If you are curious why the `fetchData` function is created, please feel free to read this article.

Let's test it.

h. Open a browser and open developer tools by pressing F12 (Firefox/Chrome) and choose the `Console` tab
i. Open a browser and open the http://localhost:3000/pokemon/2 page.
j. The Pokemon should be logged at least once:



Use the pokemon variable to add name and the image.

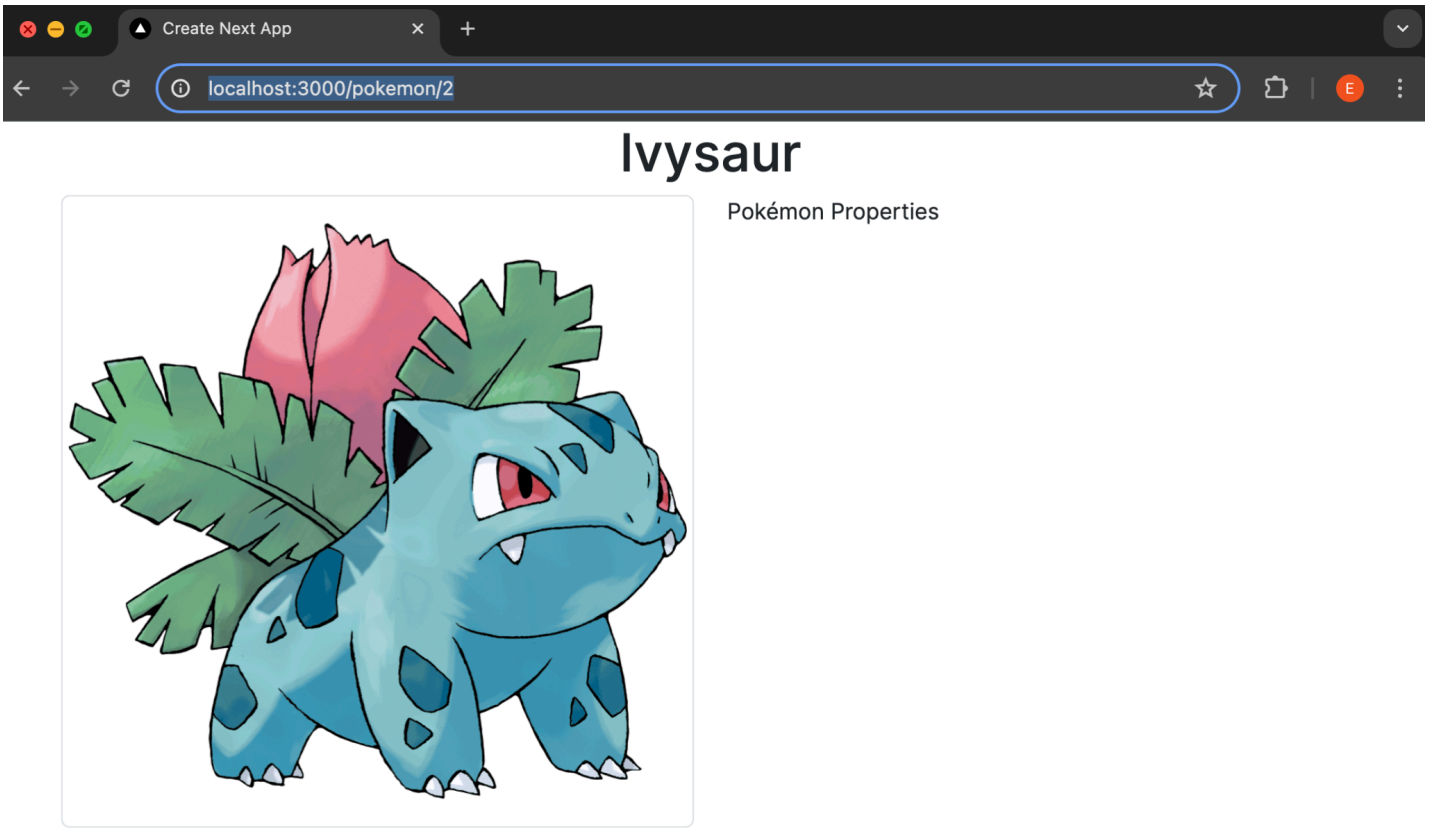
k. The React Bootstrap has an Image component which is perfect for our use-case.
l. Import this component:

```
import { Row, Col, Container, Image } from 'react-bootstrap';
```

m. Add name and image:

```
        <Container>
            <Row className="justify-content-md-center">
                <Col md="auto"><h1>{pokemon?.pokemonName}</h1></Col>
            </Row>
            <Row>
                <Col >
                    <Image src={pokemon?.mainImage} thumbnail />
                </Col>
                <Col>
                    Pokémon Properties
                </Col>
            </Row>
        </Container>
```

n. Check it out in the browser. (http://localhost:3000/pokemon/2)



Moving Pokémon to its own component and adding navbar

As our `PokemonPage` component grows bigger we can consider chopping it up and creating new components. This can help us with following software development best practices like separation of concerns and the S (Single responsibility) from the SOLID principle.

1. Create `pokemon.tsx` file within `src/app/pokemon/[pokemon_id]` folder
2. Add the content to `pokemon.tsx` file

```
import Pokemon from "@/model/pokemon";
import { Row, Col, Container, Image } from 'react-bootstrap';

type Props ={
   pokemon: Pokemon;
}

export default function PokemonComponent(props : Props) {
   const {pokemon} = props;

   return (
       <Container>
           <Row className="justify-content-md-center">
               <Col md="auto"><h1>{pokemon.pokemonName}</h1></Col>
           </Row>
           <Row>
               <Col>
```

```
                    <Image src={pokemon.mainImage} thumbnail />
                </Col>
                <Col>
                    Pokémon Properties
                </Col>
            </Row>
        </Container>
    );
}
```

3. Integrate it into the `src/app/pokemon/[pokemon_id]/page.tsx`:

```
'use client'
import Pokemon from '@/model/pokemon';
import { useEffect, useState } from 'react';
import { Container, Image, Spinner, Row } from 'react-bootstrap';
import PokemonComponent from './pokemon';
import PokeNavBar from '@/components/pokeNavBarComp'

type Params = {
  params: { pokemon_id: string }
}

export default function PokemonPage({ params }: Params) {
  const pokemon_id = params.pokemon_id;
  const [pokemon, setPokemon] = useState<Pokemon>();
  const [isPokemonLoaded, setPokemonLoaded] = useState(false);

  useEffect(() => {
      const fetchData = async () => {
          const resp = await fetch('/pokemons.json');
          const pokemons: Map<string, Pokemon> = new Map(Object.entries(await
resp.json()));
          const currentPokemon = pokemons.get(pokemon_id);
          setPokemon(currentPokemon);
          console.log(currentPokemon);
          setPokemonLoaded(true);
      };

      fetchData()
          .catch(error => {
              console.error(error);
```

```
        });
  }, []);

  return (
      <>
          <PokeNavBar></PokeNavBar>
          {
              isPokemonLoaded ?
                  pokemon ?
                      <PokemonComponent pokemon={pokemon}></PokemonComponent>
:
                      <Image className='img-fluid mx-auto d-block rounded'

src="https://cdn.dribbble.com/users/2805817/screenshots/13206178/media/6bd36939
f8a01d4480cb1e08147e20f3.png" /> :
                  <Container>
                      <Row className="justify-content-md-center p-2">
                          <Spinner className='p-2' animation='border'
role='status' />
                      </Row>
                      <Row className="justify-content-md-center p-2">
                          Loading Pokémon...
                      </Row>
                  </Container>
          }
      </>
  );
}
```

   o. Check it out in the browser. (http://localhost:3000/pokemon/2)
   p. Try with a not existing pokemon id (http://localhost:3000/pokemon/999)

What is the `isPokemonLoaded` variable and how is it used?

Kudos! You made it and already learnt a lot. 🎉
In the upcoming part we will integrate with the backend and deploy our application. If you still have time after these steps we have a challenge for you with some prizes! 😊

Integrating with the backend

1. Use Next.js to proxy the request by replacing the content of the next.config.mjs
   a. Rewrites help you to solve CORS in a hacky way. If you have control over the server, solve the CORS issue always from there. Nice to read:
      i. https://www.propelauth.com/post/avoiding-cors-issues-in-react-next-js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
   async rewrites() {
       return [
           {
               source: '/api/pokemon',
               destination:
'<PLEASE_REPLACE_IT_WITH_YOUR_BACKEND_URL>/Prod/pokemon',
           },
           {
               source: '/api/pokemon/:id',
               destination:
'<PLEASE_REPLACE_IT_WITH_YOUR_BACKEND_URL>/Prod/pokemon/:id',
           },
       ];
   }
};

export default nextConfig;
```

Pokémon listing page

Replace the content of src/app/page.tsx:

```
'use client'
import PokemonsComp from "@/components/pokemonsComp";
import PokeNavBar from "@/components/pokeNavBarComp";
import PokemonCard from "@/model/pokemonCard";
import { useEffect, useState } from "react";
import { Container, Row, Spinner } from "react-bootstrap";

export default function Home() {

 const [pokemons, setPokemons] = useState<PokemonCard[]>();

 useEffect(() => {
   const fetchData = async () => {
     const resp = await fetch('/api/pokemon');
     if (resp.ok) {
       const pokemons: PokemonCard[] = (await resp.json()).items;
       console.log(pokemons);
       setPokemons(pokemons);
     }
```

```
    };

    fetchData()
      // Making sure to log errors on the console
      .catch(error => {
        console.error(error);
      });
  }, []);

  return (
    <>
      <PokeNavBar></PokeNavBar>
      {pokemons ?
        <PokemonsComp pokemons={pokemons}></PokemonsComp> :
        <Container>
          <Row className="justify-content-md-center p-2">
            <Spinner className='p-2' animation='border' role='status' />
          </Row>
          <Row className="justify-content-md-center p-2">
            Loading Pokémons...
          </Row>
        </Container>
      }
    </>
  );
}
```

Pokémon detail page

Replace the content of src/app/pokemon/[pokemon_id]/page.tsx:

```
// Instructs React (Next.js) to run this code on the client side.
// Next.js by default would render this content on the server side where the
application is hosted.
'use client'
import Pokemon from '@/model/pokemon';
import { useEffect, useState } from 'react';
import { Container, Image, Spinner, Row } from 'react-bootstrap';
import PokemonComponent from './pokemon';
import PokeNavBar from '@/components/pokeNavBarComp';

// This type is used to get the pokemon id from the url path
type Params = {
  params: { pokemon_id: string }
```

```
}

// Next.js passes the url parts which are defined between square brackets []
// to the function which renders the page.

// In our case http://localhost:3000/pokemon/2 is the URL.
// Where the 2 is the [pokemon_id] and passed as a parameter.
export default function PokemonPage({ params }: Params) {
   const pokemon_id = params.pokemon_id;
   //pokemon - A constant state variable which stores the pokemon information
and retains the data between renders.
   //setPokemon - A state setter function to update the variable and trigger
React to render the component again.
   const [pokemon, setPokemon] = useState<Pokemon>();
   const [isPokemonLoaded, setPokemonLoaded] = useState(false);

   useEffect(() => {
       const fetchData = async () => {
           const resp = await fetch('/api/pokemon/' + pokemon_id);
           if (resp.ok) {
               const pokemon: Pokemon = await resp.json();
               console.log(pokemon);
               setPokemon(pokemon);
           }
           setPokemonLoaded(true);
       };

       fetchData()
           // Making sure to log errors on the console
           .catch(error => {
               console.error(error);
               setPokemonLoaded(true);
           });
   }, []);

   return (
       <>
           <PokeNavBar></PokeNavBar>
           {
               isPokemonLoaded ?
                   pokemon ?
```

```
                    <PokemonComponent pokemon={pokemon}></PokemonComponent>
:
                    <Image className='img-fluid mx-auto d-block rounded'

src="https://cdn.dribbble.com/users/2805817/screenshots/13206178/media/6bd36939
f8a01d4480cb1e08147e20f3.png" /> :
                <Container>
                    <Row className="justify-content-md-center p-2">
                        <Spinner className='p-2' animation='border'
role='status' />
                    </Row>
                    <Row className="justify-content-md-center p-2">
                        Loading Pokémon...
                    </Row>
                </Container>
            }
        </>
    );
}
```

Deploy using AWS Amplify

The goal

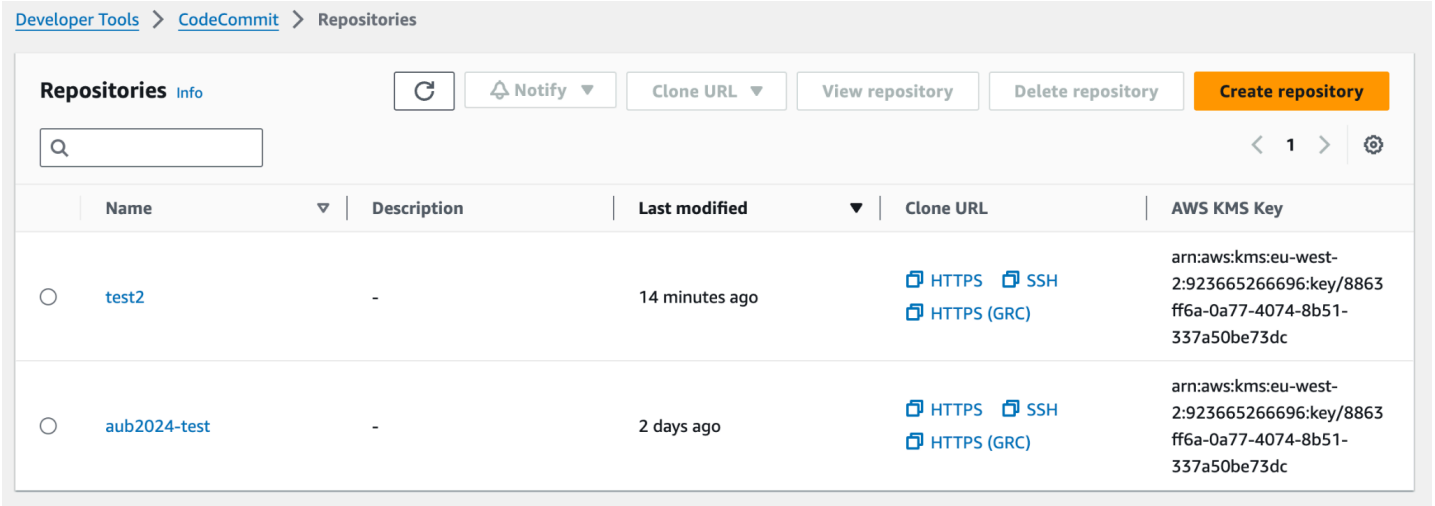
We built the app in our local development environment. We need to deploy it to make it available for our customers. AWS Amplify is one of the easiest solutions.

Git providers

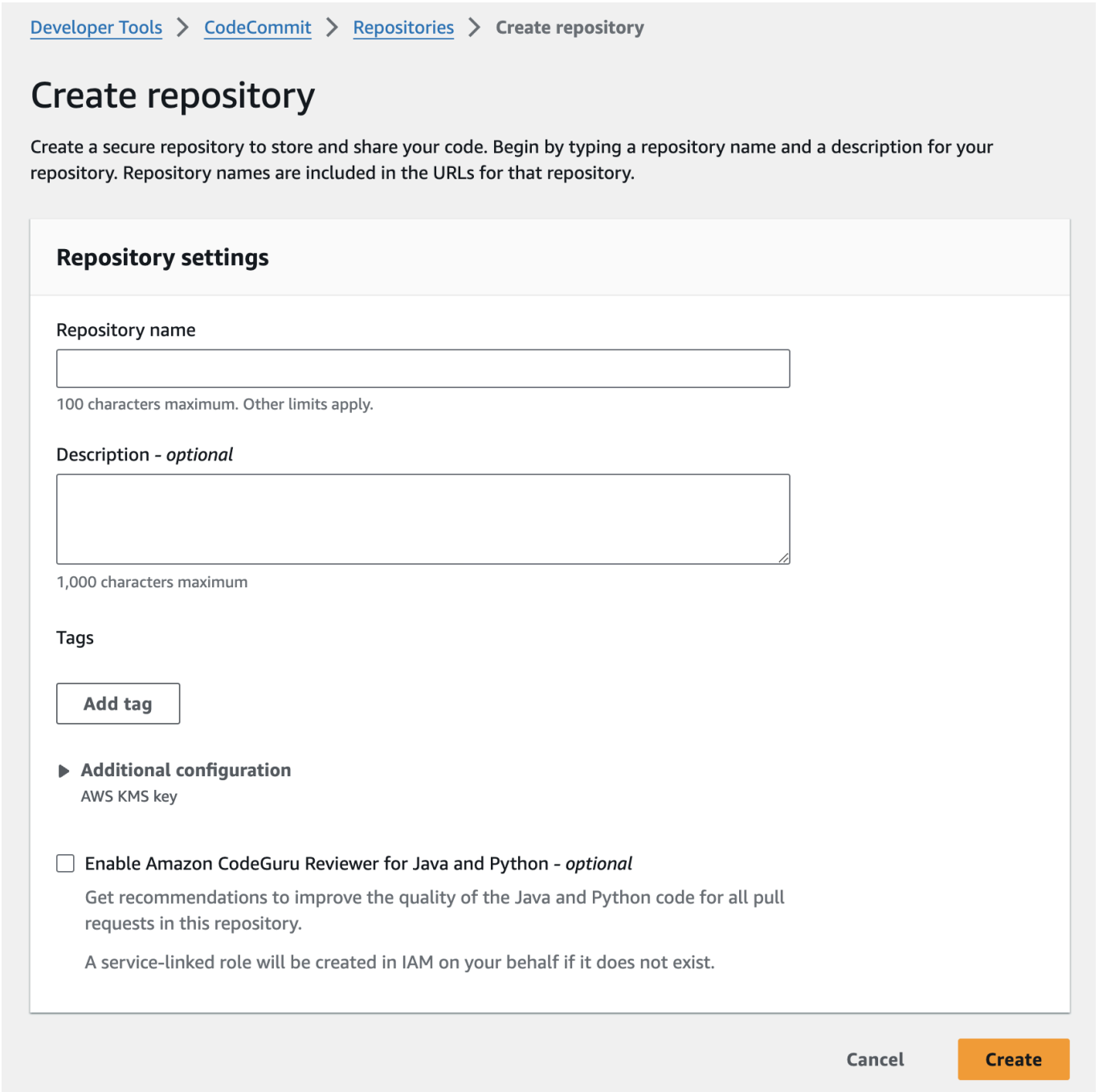
1. AWS Amplify requires a git repository to deploy the app. Some well known options: **Github, Gitlab, BitBucket, AWS CodeCommit**
2. Setup a repository in your Git provider (such as Github)
3. If you don't have an existing account in one of the Git providers you can use AWS CodeCommit create a Git repository. Follow the below steps.

[Required if not have an existing git provider] Using AWS CodeCommit to create git repository

1. Navigate to AWS CodeCommit in the AWS console using Search box.
2. Click on the "Create repository" button on the top right corner.

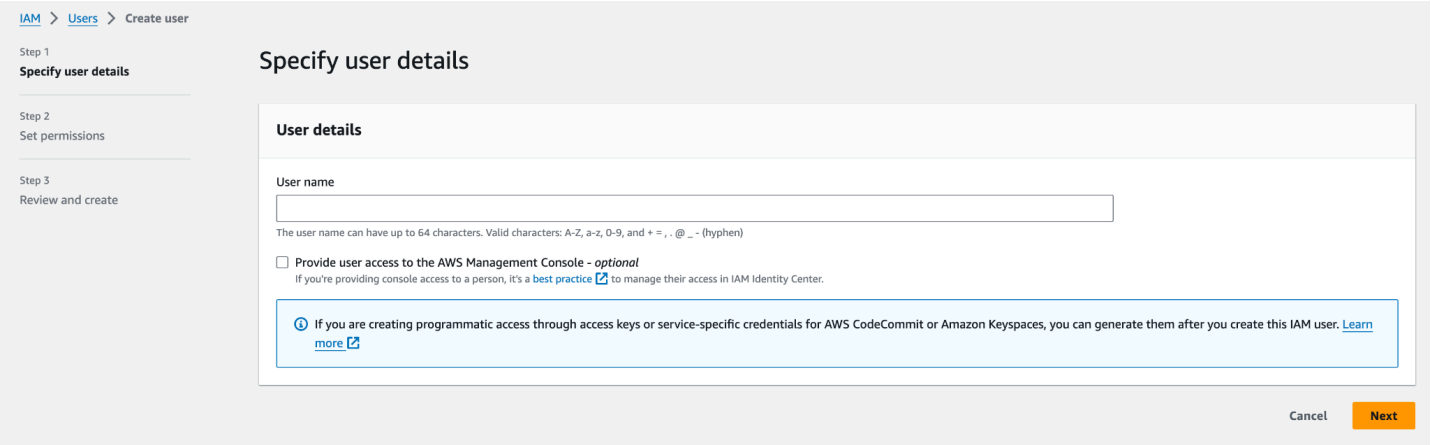


3. Fill the repository name field and Click on the “Create” button.

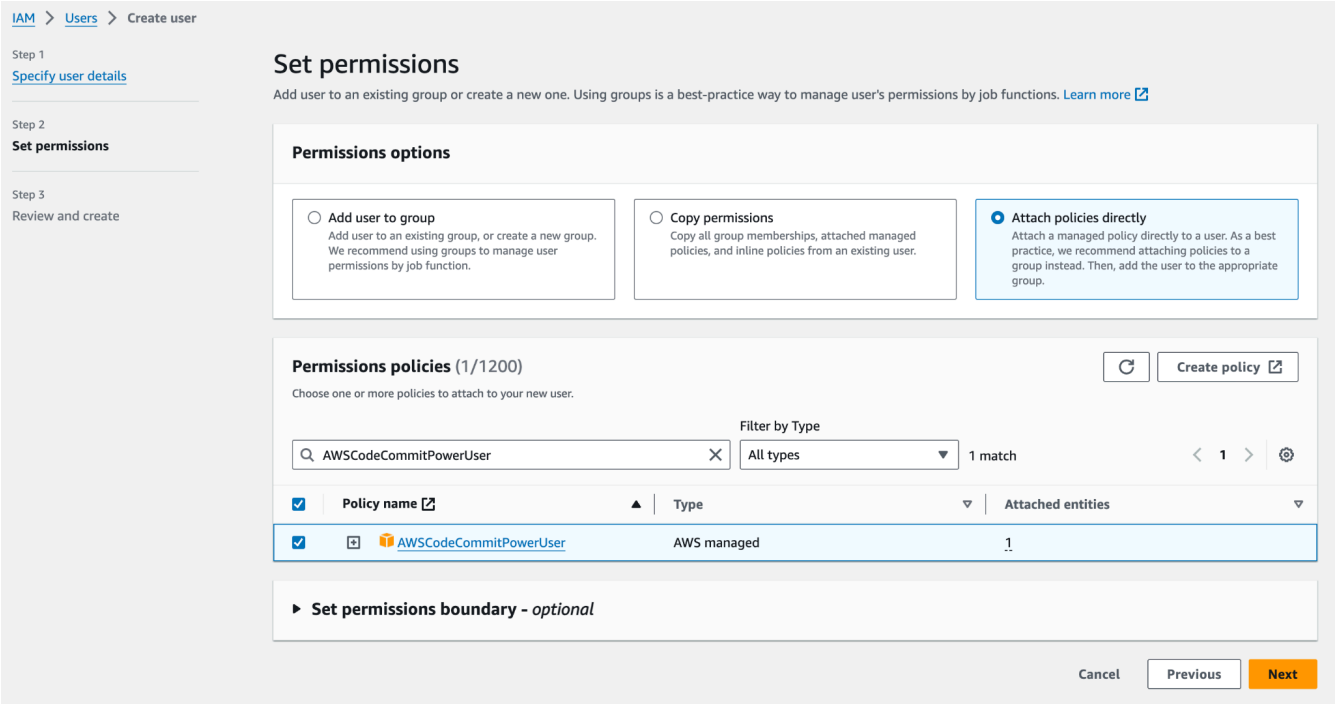


4. **[Required if not using Cloud9]** Create an IAM user that will have permissions to access to the repository. The original doc for setup: https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-gc.html?icmpid=docs_acc_console_connect_np

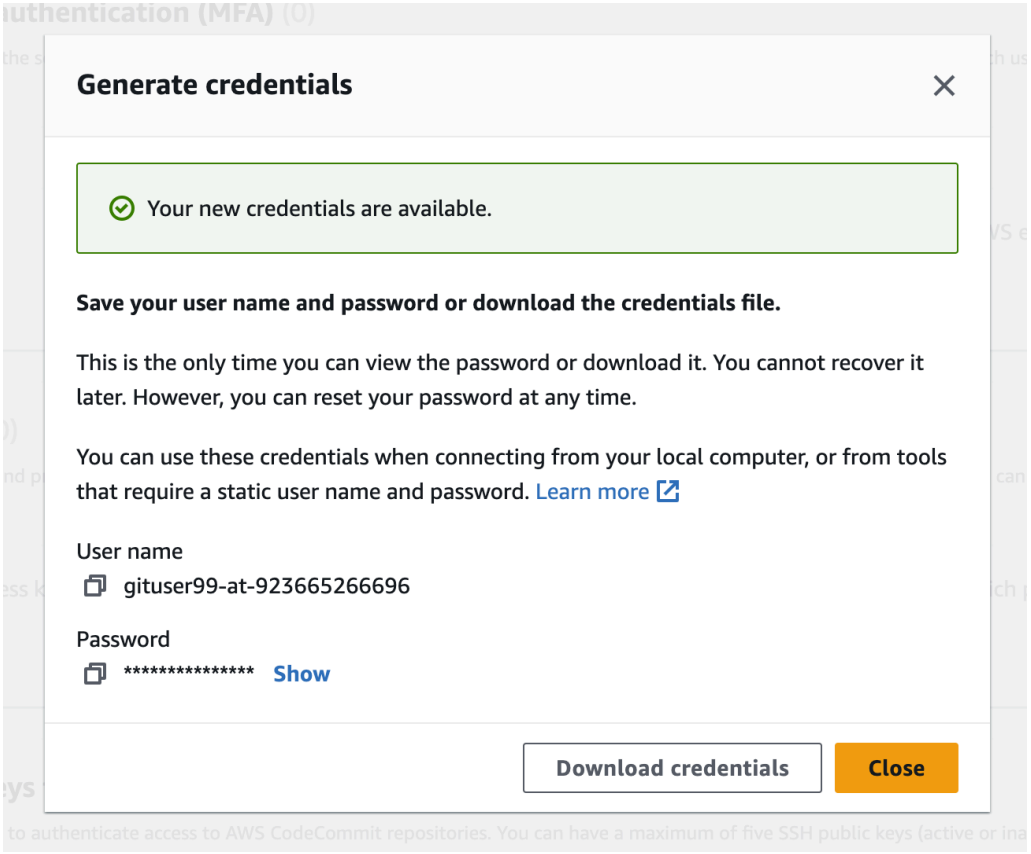
a. Navigate to AWS IAM in the AWS console using Search box.
b. On the left navigation pane, click on the “Users” link.
c. On the “IAM > Users” page, click on the “Create user” link.
d. Fill the “User name” field, give a user name and click on the “Next” button.



e. On the permission options, choose “Attach policies directly”
f. On the Permissions policies, search “AWSCodeCommitPowerUser” and choose it.



g. On the “Review and create” page, click on the “Create user” button.
h. On the “IAM > Users” page, click on the user.
i. On the “Security credentials” tab, navigate to “HTTPS Git credentials for AWS CodeCommit” section, click on the “Generate credentials”
j. Click on the download to save the username and password of your account to your computer *(Note that this is a security risk to save credentials in plain text. Other than learning context, we don’t advise you to save this file.)*



5. Copy the url of the repository from the top right corner of the repository page using the “Clone URL > Clone HTTPS” link.
6. We create the git repository and push our codes to repository
7. Run the following terminal commands to add source files and push them to our git repository.

Push the App to a Git Repository

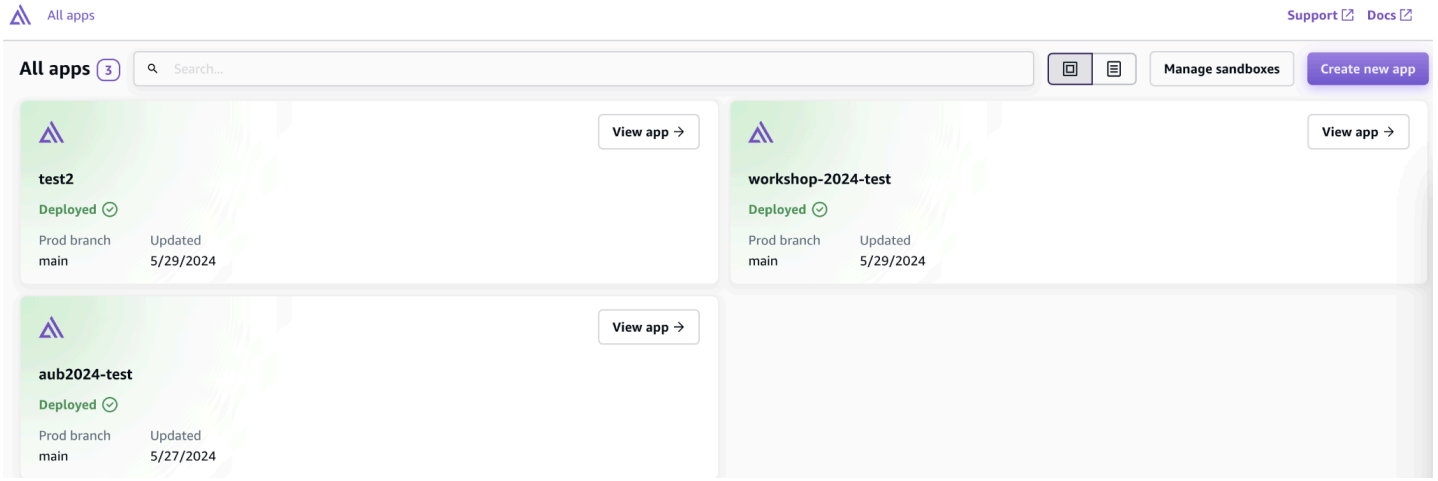
1. Since we setup a repository for the app (either existing git providers such Github, or AWS CodeCommit as shown above), we are ready to push our code to the git repository.
2. Run the following terminal commands in the project folder (If you already know Git you can use your own favourite tool to commit and push your changes)

```
git init
git add src
git add public
git add package.json package-lock.json tsconfig.json .gitignore .eslintrc.json
next.config.mjs
git commit -m "first commit"
git branch -M main
git remote add origin <repository url>
git push -u origin main
```

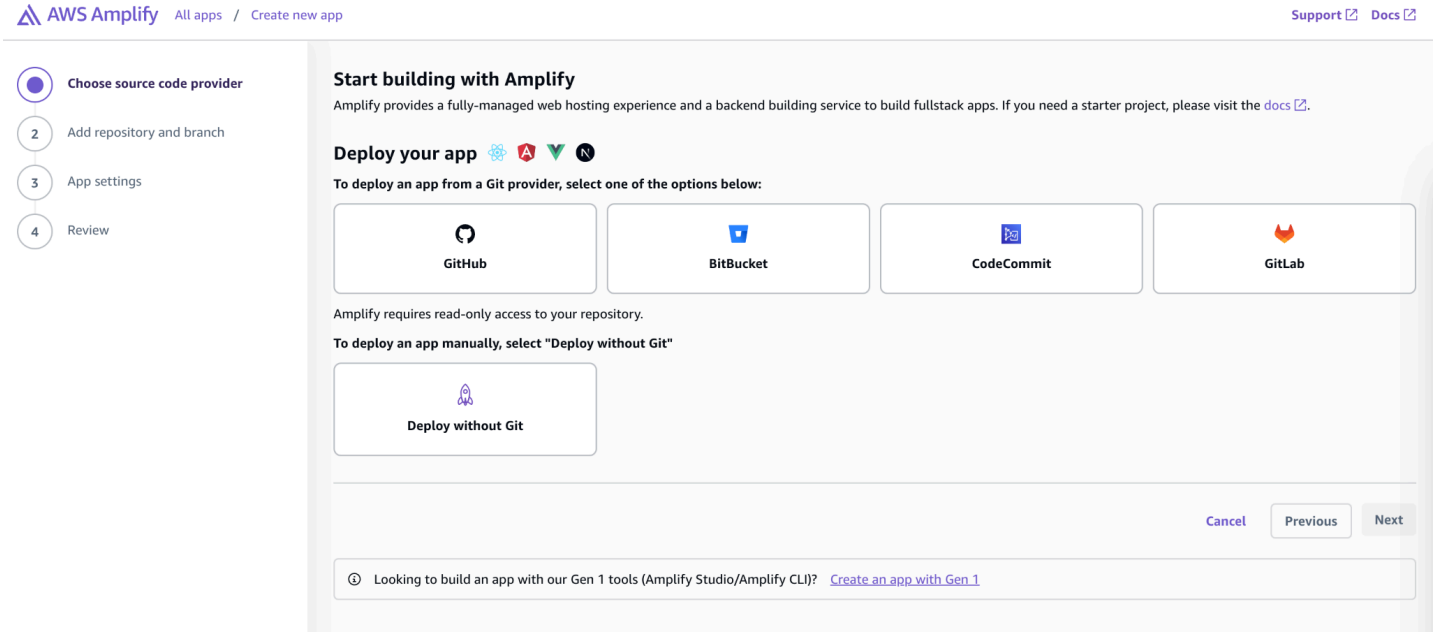
3. **[For non-Cloud9 participants using CodeCommit]** Use the username and password you saved at the end of AWS CodeCommit section.
4. Navigate to your git provider and validate that project codes are pushed to the repository.

Deploy the app using AWS Amplify

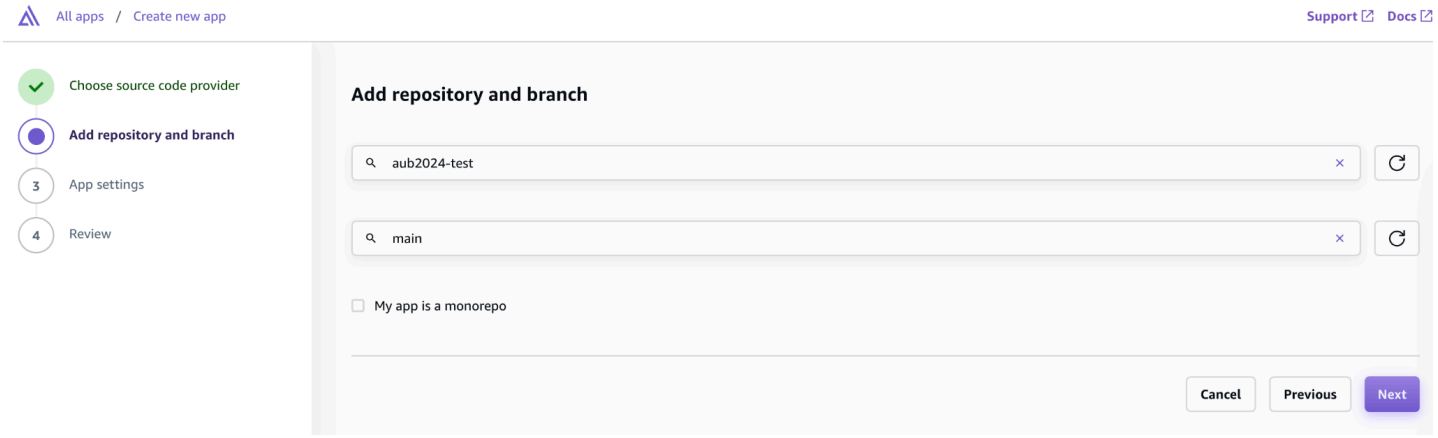
1. Click on the “Create new app” button.



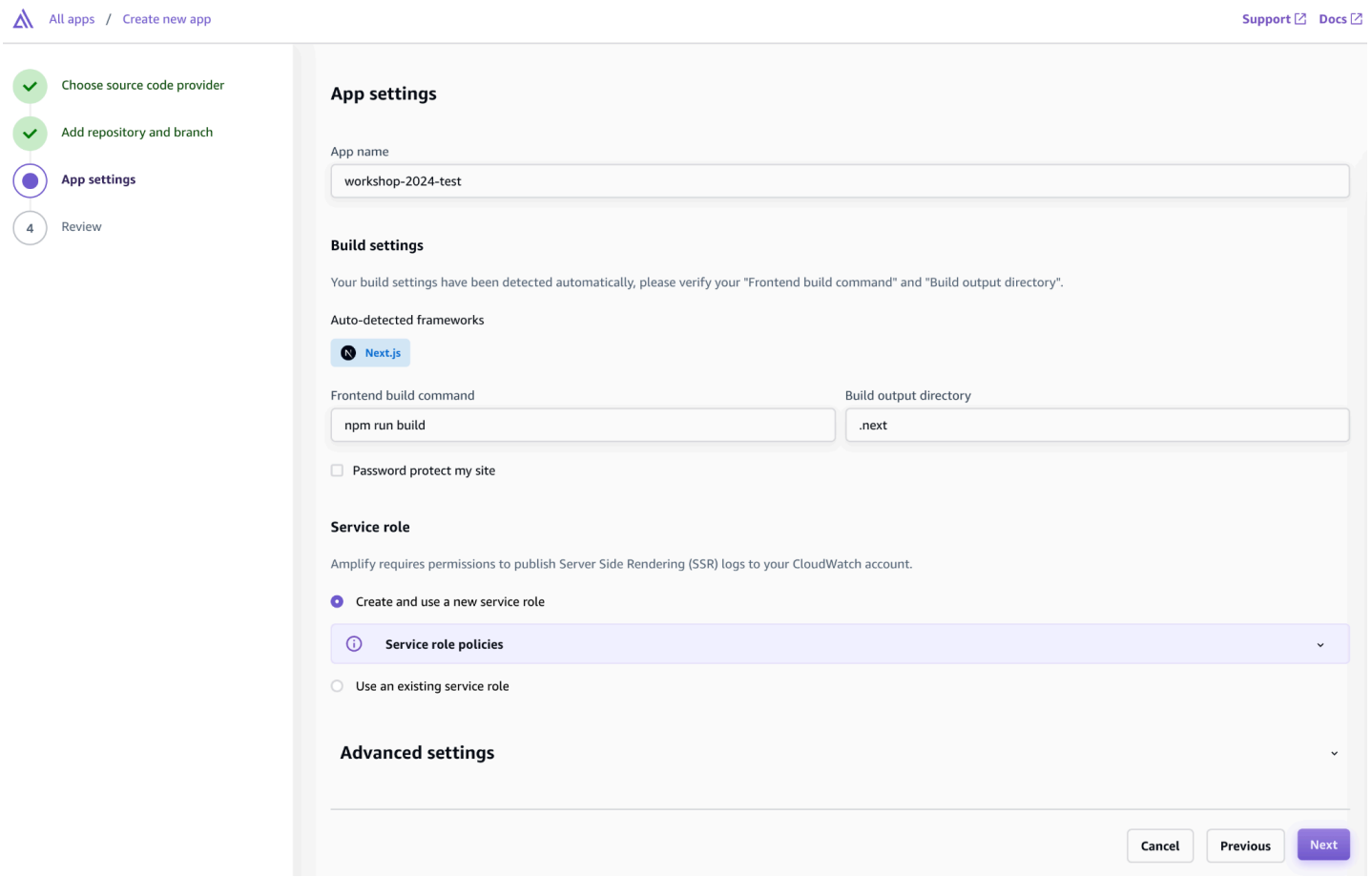
2. Choose your own git provider and click on the “Next” button.



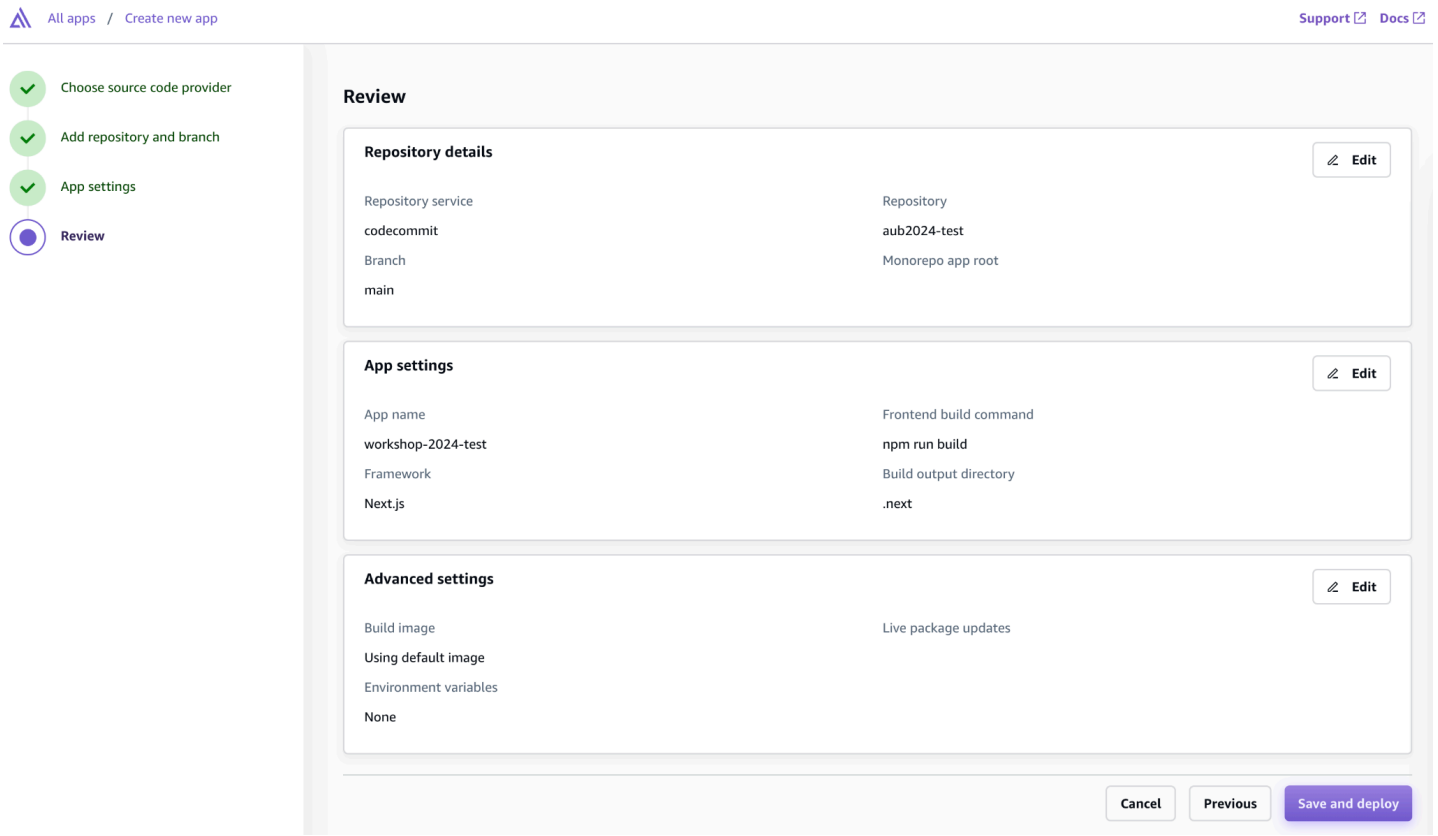
3. Choose your repository and click on “Next” button.



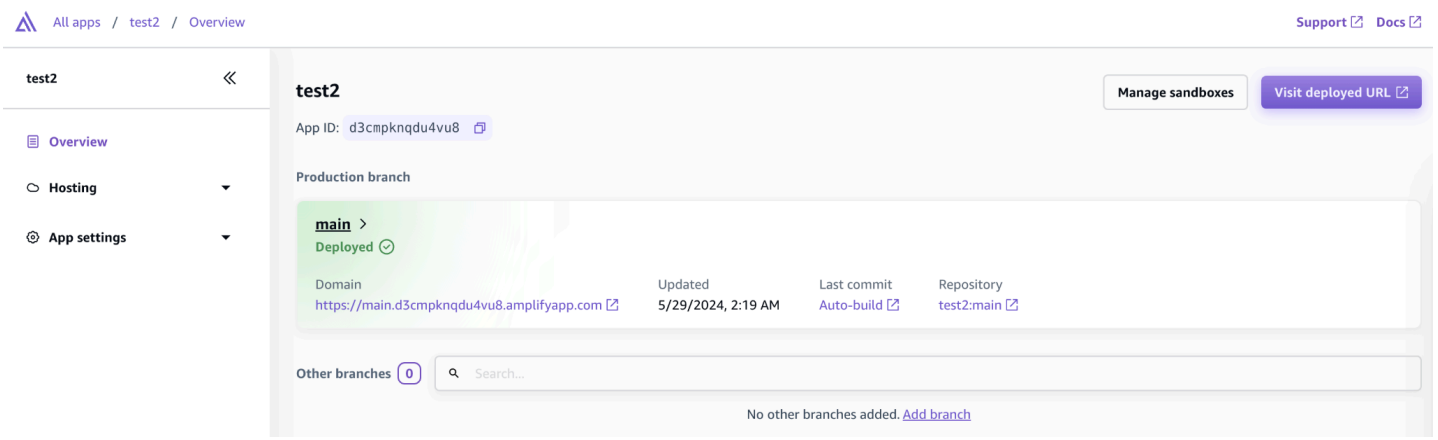
4. Keep the app settings same as loaded and click on the “Next” button.



5. Review your settings and deploy by clicking on the “Save and deploy” button.



6. Wait for deployment to be completed.
7. You can view your deployed app by clicking on the “Visit deployed URL” button.



Challenge - Best submitters get a gift

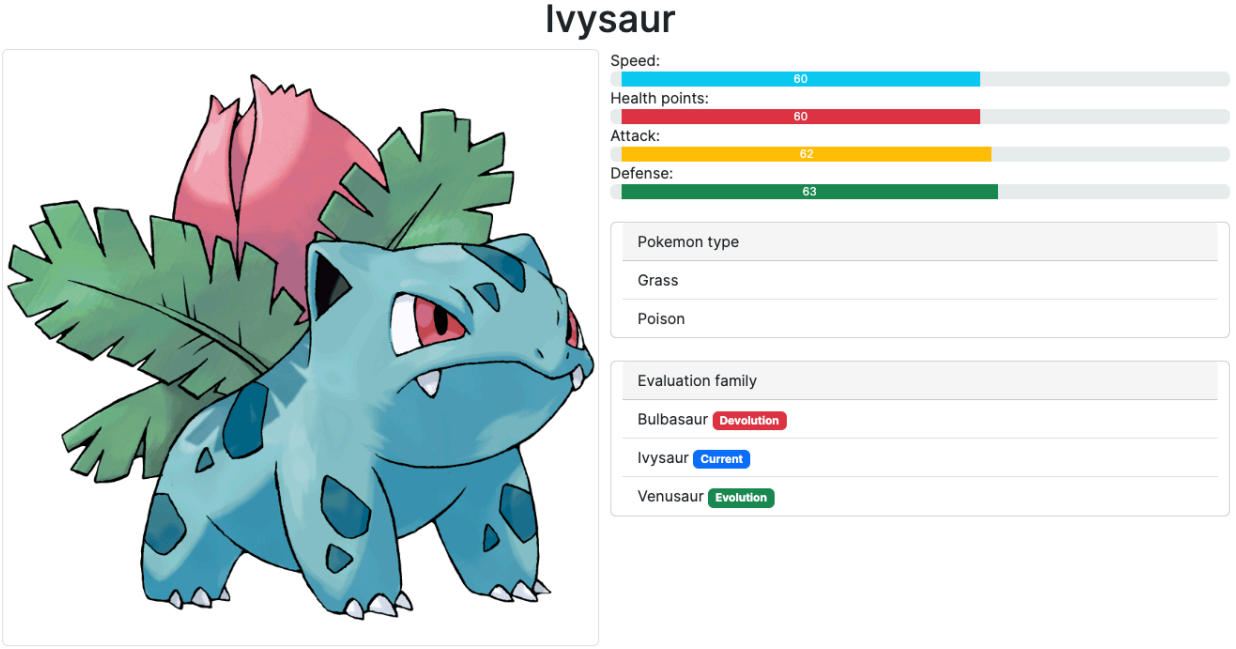
Challenge 1:

Finish the Pokémon Properties part on the Pokémon detail page and adjust the design as you want. The following pages might help:

- React Bootstrap components
- Example solution - Please feel free to be more creative ☺

How to participate?

- Send screenshots and share the code in a public git repository till **13th June** with bojti@amazon.com.



Challenge 2:

Extend the existing application purely by using Amazon Q and prompts. (Minor manual corrections on the code are allowed.) You can add new functionalities, use it to make the UI more advanced or let your imagination work and come up with your own idea.

Amazon Q installation:

- In the case of windows, start by installing WSL.
- Install Amazon Q

- Bence is working on the get access for the next 2 weeks. - Please check the discord server for updates.

Articles to read:

- https://ucddublin.pressbooks.pub/StudentResourcev1_od/chapter/the-structure-of-a-good-prompt/
- https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api

Tips and tricks:

- Always specify that the LLM needs to use Typescript, React, Bootstrap… (all the technologies)
- Make sure to ask Amazon Q to execute the `npm run build` command and try to fix the issues.

How to participate?

- Send screenshots, share the code and **prompts** in a public git repository till **13th June** with bojti@amazon.com.
- Please also explain in the email the functionalities, improvements that you have done, plus your learnings with Amazon Q.

Troubleshooting

Notes

Why to choose a React framework
https://react.dev/learn/start-a-new-react-project

Installing NVM
https://github.com/nvm-sh/nvm?tab=readme-ov-file#installing-and-updating

Handling cors with next js
https://blog.logrocket.com/using-cors-next-js-handle-cross-origin-requests/

Next js proxy:
https://blog.logrocket.com/how-to-use-proxy-next-js/

TODO (2025)

- [ ] Pokemon types are removed from the listing api.
- [x] ~~Slack worked well for helping students. - Use it again.~~
- [x] ~~All students went with VS code setup, no one used cloud9 - we can remove that part~~
- [ ] Ask the student to do npm run lint