p1-inizializzazione

1. Project initialization

> - **React for Dummies**, I.T.T. "Blaise Pascal" @ Cesena, IT
> - Written by: Nicholas Magi - nicholas.magi24[@]gmail.com

Table of contents

1.1 How to create a React + Vite project

(From https://vite.dev/guide/#scaffolding-your-first-vite-project)

> ⚠ **Warning**
>
> **Ensure your PC has installed Node.js version 18+ or 20+**! Check your installed version with `node --version` or install it from https://nodejs.org/en/download/package-manager.

Simply run `npm create vite@latest` on your terminal and follow the prompts!

Alternatively, run `npm create vite@latest <nome_progetto> -- --template react` to skip guided prompts.

Finally run the `npm install` command, to download the default dependencies needed.

1.2 What happens after project creation

To see your code changes live in the browser, run `npm run dev` on your terminal and go to the link you're given.

You will see a basic pre-defined React application with some images, texts, a Counter and a button to increase its value. Everything shown is contained in `src/App.jsx`, with the corresponding styles defined in `src/App.css`.

1.3 Before going on: Bootstrap installation

During the course we want and we will put some style to our pages and components. In order to do that, we won't use pure CSS: a shallow knowledge of the language is needed, but it's not the core of the project. We'll use **Bootstrap**, a powerful frontend toolkit that allows us to use pre-defined styles and animations. The official guide is https://getbootstrap.com/;

On your terminal, type `npm i bootstrap@5.3.3`

After the installation has been completed, we need to import its CSS and JS files to our project in order to use its classes. To do that, go to `src/main.jsx` and, below the import statements you see in the first lines of the page, add:

- `import 'bootstrap/dist/css/bootstrap.min.css'`
- `import 'bootstrap/dist/js/bootstrap.bundle.min'`

1.4 Project structure

Take a look at the files contained in `<nome_progetto>`:

```
<nome_progetto>/
|
└──node_modules/
    |   [...]
└──public/
    |   [...]
└──src/
    └──assets/
        |   [...]
    |   App.css
    |   App.jsx
    |   index.css
    |   main.jsx
|   [...]
|   README.md
|   [...]
```

What you've got is a `node_modules/` directory containing all the project **package dependencies** (*remember to '.gitignore' it, if you want to track the project with git!*), a `public/` directory (made for assets files when **deploying the application**) and a `src/` directory. Here you'll have all the important files containing **React components and the corresponding stylesheets**.

Consider that we'll be working primarily **inside the `src` directory**.

1.5 Inside `src/`

We'll define our React components and styles inside the `src/` folder, that we'll consider our main working space - as previously said. The default template has already crafted one component for us: `App.jsx`.

This gives us the chance to take a look at the typical structure of a component:

- In the very first top lines you **import** everything you need within your component - could it be a stylesheet (`App.css`), some images (`./assets/react.svg`, `./vite.svg`) or/and other components as well.
- Then there's the main component implementation - which corresponds to just a "normal" JavaScript function implementation. You declare the function *with some parameters (optional)*, write its body and its return statement. What you see returning from `App()` is not pure HTML, but it's called **JSX** (***J**ava**S**cript e**X**tension*) - **which describes the UI of the component**.

> ⚠ **Remember!**
>
> That is a crucial point to keep in mind, because many features of this language are similar to HTML, but there are some key differences that we'll cover later.

- The last line of code is

```
export default App
```

which allows the other parts of the project to see and use the component `App`. This line can be omitted if the `export default` keywords are included in the `App()` function declaration like so:

```
export default function App() {
  // [...]
}
```

Note that inside a jsx file you can write and work with other helper functions as well, but remember to **export** the function that represents the description and logic of the component!

> ✓ **Let's get started**
>
> Delete the content of `src/App.jsx`, `src/App.css` and `src/index.css` and you'll have an empty project waiting to be coded!