README.md

Lesson 11 Exercises - React Updates, TypeScript, AI

11a. We'll add document metadata (<title>, <link>) to the Chatbot Project.

- Move the `<title>` from index.html to the App component in App.jsx.
- Move the `<link rel="icon" ...>` (the favicon) to the App as well.
- Update the favicon to use `robot.png` in `src/assets/` (import it first).
- Open the project in the browser and check the tab to see if it worked.

11b. We'll set a dynamic title (it can change over time). Update the title to "{num} Messages". Insert the total number of chat messages in `{num}`.

- Note: save the title in a variable first, and then insert it like this: `<title>{title}</title>`. That's because `{num} Messages` counts as 2 pieces of text, and `<title>` may not support multiple pieces of text.
- Create some chat messages and check that the title updates.

11c. Set up the React Compiler in the Chatbot Project. After setting it up, check that the React compiler is working (just like in the lesson, use the link in the description, go to lesson 11, and follow the instructions).

Challenge Exercises

11d. Create a new React TypeScript project called chatbot-project-ts using `create-vite`. Set up the project and run it to make sure it works.

11e. Move over all the changes from chatbot-project to chatbot-project-ts (you can make a copy of chatbot-project to keep track of progress).

11f. Fix the TypeScript errors in `main.tsx` and `App.tsx` (use `!` and add `"allowJs"` setting). Run the project and check that it works.

11g. Rename the folders so chatbot-project uses TypeScript by default. Move the JavaScript version into the old-projects folder.

11h. Convert the ChatMessage component to TypeScript and fix the errors.

- Add the type for the props: `message` and `sender` are both strings.
- Save the type in a type alias called `ChatMessageProps`

11i. Convert the ChatMessages component to TypeScript and fix the errors.

- Add the type for the props: `chatMessages` is an array of objects and each object has an `id`, `message`, and `sender` property (all are strings).
- Save the type in a type alias called `ChatMessagesProps`
- There may also be an error with `containerElem` inside the `useEffect`. This happens because `containerElem = chatMessagesRef.current;` and a ref can contain any value (TypeScript cannot figure out the type).
- To tell TypeScript what type of value is inside a ref, use the `<>` syntax: `useRef<HTMLDivElement>(null);` (`HTMLDivElement` is a built-in type).

11j. Convert the ChatInput component to TypeScript and fix the errors.

- Add the type for the props: `chatMessages` has the same type as in the ChatMessages component (copy it over).
- `setChatMessages` is a function. To tell TypeScript it's a function, use an arrow function `(chatMessages: Type1) => void;` (`Type1` is the type of the first parameter. Replace it with the type of the `chatMessages` array. `void` means this function returns nothing).
- Save the type in a type alias called `ChatInputProps`

11k. Notice in the ChatInput component, that the type for `chatMessages` is repeated twice. To make the code cleaner, save the type in a type alias called `ChatMessages` and reuse it.

Solutions in description

11l. There may be a few more TypeScript errors in ChatInput.

- First, in `function saveInputText(event) { ... }` TypeScript cannot figure out the type of `event`. Set the type manually (set `event` to an object, with a property called `target`. Set `target` to an object as well, with a property called `value`, which is a string).
- At the bottom, in the `<input>`, TypeScript may want the `size` attribute to be a number instead of a string. To fix this, update it to: `size={30}`

11m. In chatbot-project, run `npm run build`. In the dist folder, notice the TypeScript code was converted to normal JavaScript. Put this project on the Internet. (If you get TypeScript errors when you `npm run build`, try to fix them. If you can't, check the solutions for this exercise).

11n. In ecommerce-project, run `npm run build` and put it on the Internet.