TypeScript

It's one thing to talk about the advantages of using TypeScript over plain JavaScript and another to implement it.

There are lots of things that are done for you if you create the project with TypeScript, things that we all take for granted. In this lab you'll add TypeScript to an existing React application and experience the challenges of doing so. Please expect lots of challenges as you do this.

Run your application as it is now. The site should be fully operational as it is but adding TypeScript will make it easier for future devs to update it without introducing type errors. In other words, refactoring to TypeScript will erase some technical debt.

Installing and setup

1. First, install the TypeScript transpiler and the types needed by React.

```
npm install --save-dev typescript
npm install @types/node @types/react @types/react-dom
npm install --save-dev @typescript-eslint/eslint-plugin
npm install --save-dev @typescript-eslint/parser
```

2. Rename main.jsx to main.tsx.

3. Edit index.html. Tell it to use main.tsx instead of main.jsx.

4. Run and test. It still works. Cool. But notice that there are linter warnings.

5. Rename vite.config.js to vite.config.ts.

6. Move for_typescript/tsconfig.json and tsconfig.node.json to the root of your project.

7. Edit package.json and change this:

```
"build": "vite build",
```

... to this:

```
"build": "tsc && vite build",
```

8. Run and test again. It should still work. You have a few linter warnings but we can work on those as we go. Keep moving.

Adding business types

(NOTE: Read through this entire section before beginning! You can save yourself a ton of time if you use a tool like https://quicktype.io or https://apidog.com.)

If we're using TypeScript, you must create types for your business objects. In this section, you'll make types from the API data. We'll give you one and have you write the others yourself.

9. Create a folder under src called types.

10. Put this file in it called cartItem.ts

```
export type CartItem = {
  id: number,
  itemId: number,
  name: string,
  category: string,
  price: number,
  imageUrl: string,
  firstName?: string,  // Diner's name
  notes?: string,  // Special instructions
}
```

11. Using Postman, cUrl, or just a browser, make a request for each of the business entities we have in the database:
    - menuItems
    - orders
    - users

    Hint: You'll find menuItems by GETting http://localhost:3008/api/menuItems. Then do the same for orders and users.

12. Make a file in the types folder for each of these. It should export a type.

You're probably going to want to create types for sub-properties. If a type has a property that is another type of object, it is using composition (a "has a" relationship) and should have a type made for it.

13. In the database each user **has a** creditCard and each order **has a** creditCard. So create a CreditCard type.

14. In the database each order **has a** menuItem. So create a MenuItem type.

You may or may not feel the need to create business types for a few entities that are not in the database but only client-side like CartItem as you created above and FoodCategory. These are totally up to you and can be added on an as-needed basis. In other words, feel free to wait on creation of these until you really need them ... or never create them at all. If you choose the latter, congratulations on embracing the sanity of compromise with strong typing. Welcome back from the dark side. 😂

Rename the source files

Here are three true things.

A) TypeScript increases the complexity of source code.
B) If a file isn't named with a .ts or .tsx extension, it's just plain JavaScript.
C) TypeScript and JavaScript can coexist in a project.

A + B + C = You don't have to make them all TypeScript files immediately. You can do it gradually.

Therefore you can do the remainder of these steps incrementally, one file at a time. Do the remaining steps for each JavaScript and JSX file, continue converting until you run out of time or are simply ready to quit. Let the instructor know that you're ready to move on as soon as that occurs.

15. Begin with the plain js files. These will be found in the data folder. After you do these, continue with the components. For them, start with App.jsx and continue in any order you like. Do all of the following bullets for each file you convert to TypeScript:

    - First, rename the file from foo.js to foo.ts or Foo.jsx to Foo.tsx.

Adding types

Most variables should have a type and it should be something other than "any".

- Go through the file and add a type to every variable declaration, using the classes defined earlier or the native JavaScript types (number, string, bool, etc.).
  Hints:
  - Arrays can be declared with generics or just square brackets. (Array<string> or string[]).
  - When you add types, you may need to cast like "const foo: string = bar as string"
  - If you're desperate, declare a variable as "any" just to get it to compile and make a TODO to come back and set it properly later.

Handling props

Each component that receives props should have those props defined as an Interface.

Here's how to do one of them ...

```
interface Props {
  addToCart: (cartItem: CartItem) => void;
}
export const Menu = ({ addToCart }: Props): ReactElement => {
 ...
}
```

- For each component that receives props, add an interface and use it like in the example above.

Adding types for state

Each component that has a useState should have the type declared. Here's an example of a single value:

```
const [user, setUser] = useState<User | undefined>({});
```

And here's an example of an array:

```
const [cart, setCart] = useState<Array<CartItem>>([]);
```

- For each component that has state, declare types for all state variables like in the above examples.

Adding style types

Some devs assert that we should always declare a type for styles. There's almost zero chance of this ever protecting against a real problem but it is excellent for intellisense help while developing.

Instead of this:

```
const styles = {
```

... you'll want this:

```
const styles: { [P: string]: CSSProperties } = {
```

- Look through each component for where a style object is defined. Add the type above to them. Don't forget to import CSSProperties from react.

Fixing miscellaneous errors

We left a few errors in the code that would have been hard to detect without TypeScript. They were easily overlooked before but if you avoided the use of 'any', they became very obvious. Here are some hints:

- There is no PAN in a credit card.
- If tip is zero, there's a nullish problem.

- Go ahead and fix any errors that you see. If you don't encounter any at this point, don't worry about it. They're very subtle. You can always squash bugs later if they pop up.