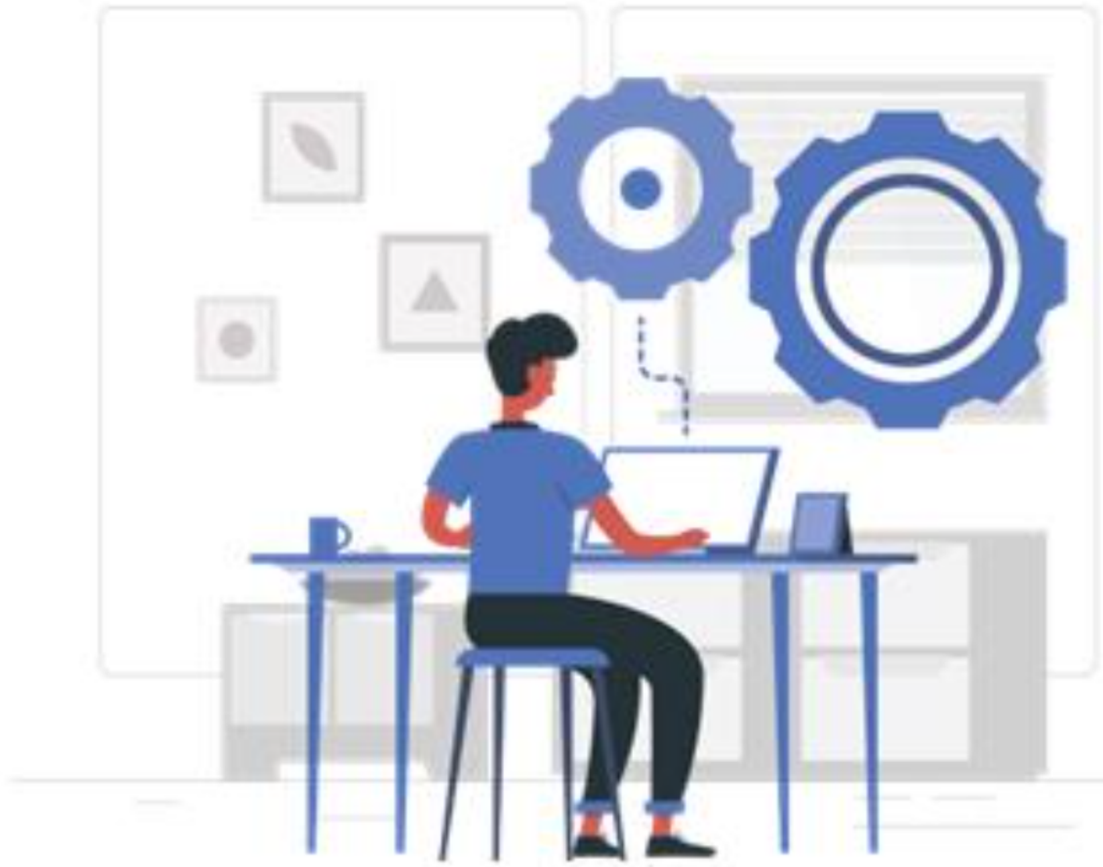

Practice

Build Interactive Web Pages With TypeScript

Practice Exercises

- Model a user object and store the user details using fetch API

Context

User registration systems are screens, forms, or profile pages that request information from a user to create a web-based account or profile. A user registration system generally asks users to create a username, password, and contact information.

Most web applications have user registration forms to record enough user information to facilitate user authentication. The user can change his profile information at any time.

In the first phase, create a simple user registration form using TypeScript where the users can fill and submit their personal information. These details are stored in a database to later be used for authentication.

Points to Remember

- Install TypeScript on your system using NPM.
- Ensure json-server is installed in your system.
- Use appropriate data types for all the data used in the solution.
- Avoid using the data type "*any*", unless you are unsure about the data type.
- Typecasting (Overriding a type) can be done using <> like the code given below:

```
let x: unknown = 'hello';
console.log((<string>x).length);
```

- Indent the TypeScript code appropriately.

Instructions for Accessing the Boilerplate

- Click here for the boilerplate.
- Please read the README.md file provided in the boilerplate for further instructions about the practice exercise.
- Fork the boilerplate into your own workspace.
- Clone the boilerplate into your local system.
- Open a command and terminal run the command **`npm install`** to install `Mocha` and `Chai` as dependencies.
- Open the folder containing the boilerplate code in VS Code.
- Provide the solution code in the files specified with the task details.

Instructions for the Practice

- The unzipped code contains **solution/index.html** which contains the partial design code of the `User Registration` web application.
- Modify the existing code to fulfil the requirements stated with the tasks.
- Write TypeScript code in the **user.ts** located inside **solution/src** folder of the boilerplate as per the requirements stated in the upcoming slides.
- Open the terminal in the root directory and run the command **`npm install`** to install the dependencies.
- Give the command **`npm run build`** to compile **`.ts`** files and verify the converted files are located inside **`solution/public/js`** folder.
- Open another terminal window and start the json-server by giving the command **`json-server solution/json/users.json`**.
- Open the **index.html** file using Live Server and test the output.
- Submit the files for evaluation.

PRACTICE

Model a User Object and Store the User Details Using Fetch API

Write a TypeScript program to create an interactive user registration page that captures the user's details and stores them in a database. The user details like username, password, contact, email, and address should be captured from the form.

Tasks

- The exercise can be performed by the steps given below:
  - Step 1: Call a function on form submit
  - Step 2: Model User class
  - Step 3: Capture form field values
  - Step 4: Persist User details

Note: Details on these steps are given in the upcoming slide.

Step 1: Call a Function on Form Submit

- Add an onsubmit event with the form element available in the **index.html** of the boilerplate code to capture form field values.
- Give the path as **`public/js/user.js`** inside the script tag of the **index.html**.

Step 2: Model User Class

- Inside `**user.ts**`, model the class User inside the user.ts file provided in the boilerplate code.
- Declare variables for the attributes identified in the class.
- Declare getter and setter property methods in the user class.

Note: Model the user object with all attributes and behaviors which can be later reused for profile change functionality.

Step 3: Capture Form Field Values

- Edit the `**submitUser()**` function inside `**user.ts**` file to capture the user form field values using the FormData object and construct the User object.
- Call `**registerUser()**` method with the constructed User object.

Step 4: Persist User Details

- Edit the **`registerUser()`** function inside **`user.ts`** file to save the user data in **`users.json`** file using fetch API.
- Once the user details are saved, it should display the text "You have successfully registered !" on the page.

Note: This text message will be used to evaluate test case and hence any mismatch between the actual and expected values will result into test case failure.