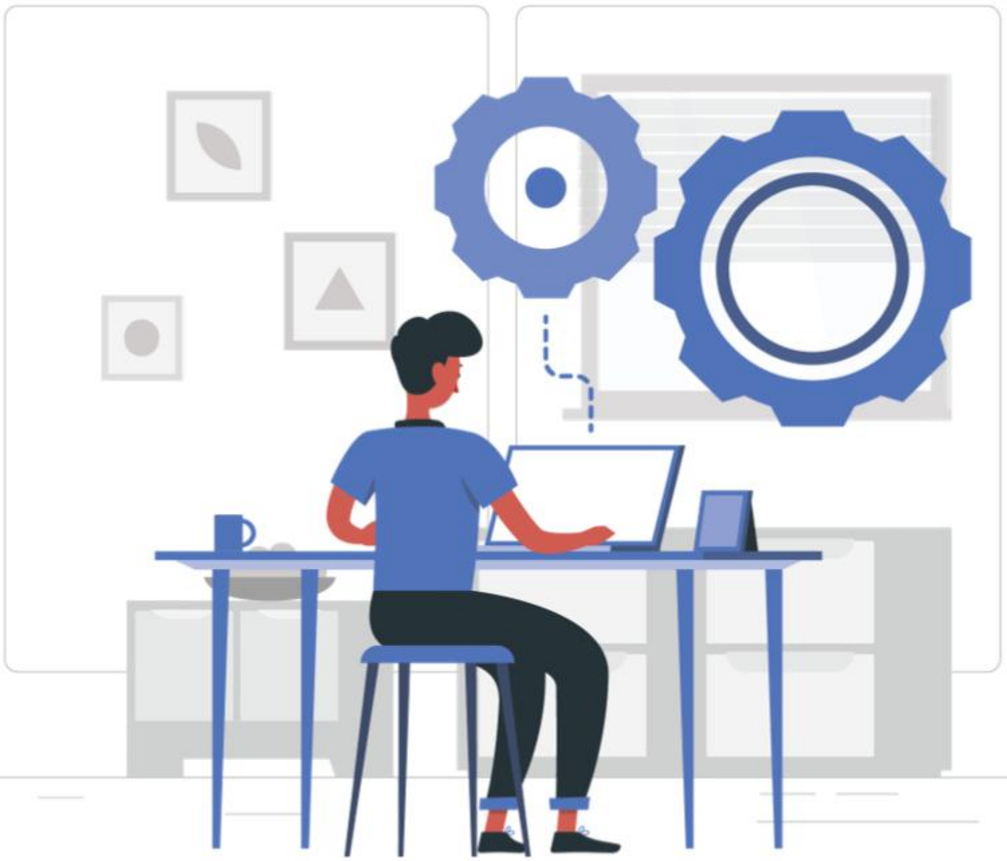

Practice **Interact With Application Servers Using HTTP Protocol**

Practices

- Practice 1: Add a new note to the To-Do list
- Practice 2: Display all notes from the To-Do list
- Practice 3: Delete a Note from the Todo List (Optional)
- Practice 4: Toggle the Notes View (Optional)

Points to Remember

- This practices of this sprint are in continuation to the practices of the previous sprint `Sprint 5 - Develop Interactive Web Pages Using DOM and DOM Events`.
- The practices require modifying the existing code and fulfilling the requirements stated with the tasks.

Context

Keep App is created by Creative Coders – a software development company. This app helps the user maintain a daily to-do list. A user can add a new note, view existing notes, and delete the selected note through this app.

For convenience, the app provides a feature that allows a user to view the notes in the form of a grid or a list.

In its current stage, the app is functional and can capture inputs from the user and display the notes added on the app's UI. However, the notes are currently stored in an array.

The developers now need to implement persistence in the app. The notes data should be saved to the server in `json` file. Whenever the page loads, the data from the server should be fetched and displayed on the app's UI.

Implementation Environment

- This practice requires the implementation of persistence using a json-server.
- The notes data should be saved in the **todo.json** file located in the **data** folder of the boilerplate.
- Ensure the `json-server` is running and serving data from the **todo.json** file.
- To access the to-do data, use `Axios` to send requests to the `json-server`.
- To use `Axios` install it using the `unpkg CDN` with the help of the `<script>` tag added in the **index.html** file:

```
<script src="https://unpkg.com/Axios/dist/Axios.min.js"></script>
```

Note: Task details are given in the upcoming slides.

Instructions for Practice

- Click here for the boilerplate.
- Read the README.md file available in the boilerplate for further instructions about the practices.
- Fork the boilerplate into your own workspace.
- Clone the boilerplate into your local system.
- Open command terminal and set the path to the folder containing the cloned boilerplate code.
- 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.

PRACTICE

Practice 1: Add a New Note to the To-Do List

Modify the existing JavaScript code in the **script.js** file to add a new note to the To-Do list.

The code should capture details inputted by the user, send the note data to the server, and display the newly added note on the page in card layout.

Steps

- Step 1: Define JavaScript function `saveNote()` in the the **script.js** file. The function `saveNote()` should:
  - Read the note details inputted by the user on the UI.
  - Using the `post()` method of the `Axios` object, send data to the server to save it in the **todo.json** file.
  - If the `post` operation is successful, the note should get stored in the notes array and should be displayed on the app's UI.
  - Raise an alert if the `post` operation fails with an appropriate error message.
- Step 2: The function `saveNote()` should get invoked when the user clicks the `Add Note` button.

Note: Make sure the `note-id` inputted should be unique.

PRACTICE

Practice 2: Display All Notes From the To-Do List

Modify the existing solution to fetch notes from the server when the page loads and display them on the UI.

Each of the notes' data should be displayed in a card layout.

Steps

- Step 1: Define the JavaScript function `displayNotes()` in the **script.js** file. The function `displayNotes()` should:
  - Using the `Axios` object's `get()` method, fetch notes data from the server.
  - If the `get` operation is successful, the notes data should be assigned to the notes array.
  - Dynamically add a div element for each note in the array.
    - The div element should contain:
      - A heading element to display the title of the note.
      - A para element to display the content of the note.
  - The div element should be applied with style properties to display the note in a card layout.
- Step 2: The function `displayNotes()` should get invoked when the page loads as well as when an existing note is deleted.

PRACTICE

Practice 3: Delete a Note from the To-Do List (Optional Practice)

Modify the JavaScript code that deletes an existing note from the note list.

The code should now delete notes from the server and update the display of notes on the UI.

Steps

- Step 1: Modify the JavaScript function `deleteNote()` to delete the selected note from the server and remove it from the notes array.
- The `deleteNote()` function should:
  - Accept the id of the note as the parameter
  - Make a `delete` request to the server with the `id` value.
  - If the note is successfully deleted from the server, delete it from the notes array.
  - Refresh the display on the UI with the updated data from the notes array.
  - If the `delete` operation fails, raise an alert to display the appropriate error message.
- Step 2: The function `deleteNote()` should get invoked when the user clicks the delete button on a note card.

PRACTICE

Practice 4 - Toggle the Notes View (Optional Practice)

The notes on the UI should by default be displayed in the grid view.

Provide an option on the UI that will allow users to toggle between the grid view and the list view.

Steps

- Step 1: In the **script.js file**, define the function `toggleView()`.
    - The `toggleView()` function should dynamically add styles/CSS classes that toggle the view between grid view and list view.
- Step 2: The function `toggleView()` should get invoked when the user clicks on the link with the text "Toggle View".

Submission Instructions

- This is a test-enabled practice; hence, the solution will have to undergo an automated evaluation process.
    - For automated evaluation, the solution should be first tested locally and then on hobbes.
- Steps to test the code locally:
    - Ensure the solution code is provided in the specified files only.
    - From the command line terminal, set the path to the folder containing cloned boilerplate code.
    - Run the command `npm install` to install the dependencies.
    - Run the command `npm run test` to test the solution locally.
    - Ensure all the test cases are passing locally, and then push the solution to git for automated testing on hobbes.