CPD (Sept 13 2023)

1. TypeScript Task List Project

   Using concepts such as `enum` and `interface` from TypeScript, this simple Web App was created.

   Enums

   - Hold the values for each of the possible task types.
   - This includes:

     ```
     enum TaskType {
         WORK = "Work",
         PERSONAL = "Personal",
         HOBBY = "Hobby",
         EXERCISE = "Exercise",
         COOKING = "Cooking"
     }
     ```

   - Components of the site used this enum such as the 'Task Type' selector which is populated only with values from the enumerated definition for `TaskType`. You can see how the selector is populated from the following code snippet:

     ```
     function populateTaskTypeDropdown(): void {
         const taskTypeDropdown = document.getElementById("taskType") as HTMLSelectElement;
         for (const taskType in TaskType) {
             if (TaskType.hasOwnProperty(taskType)) {  // Check is direct enum property
                 const option = document.createElement("option");
                 option.value = TaskType[taskType as keyof typeof TaskType]; // gets and sets the string value
                 option.textContent = TaskType[taskType as keyof typeof TaskType]; // gets and sets the string value
                 taskTypeDropdown.appendChild(option); // add to dropdown
             }
         }
     }
     ```

   - When a task is added, the representation of the value from Task Type is in its enumerated format. An example of this can be seen from the following snippet of code:

     ```
     const taskType = (document.getElementById("taskType") as HTMLSelectElement).value as TaskType;
     ```

Interfaces

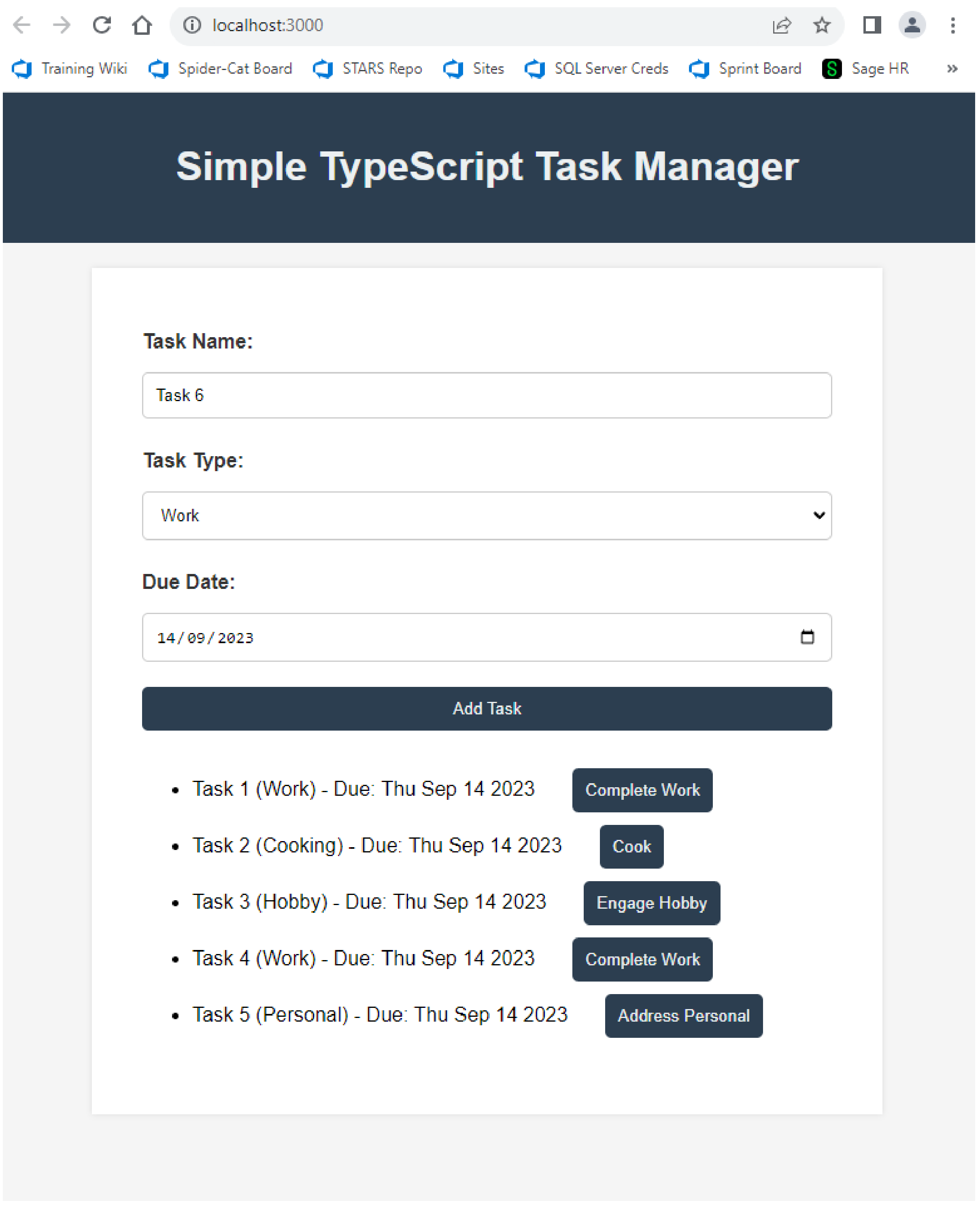
- Interfaces in TypeScript play a vital role in ensuring type safety and consistent object structures. In the provided code, the `Task` interface is utilized in various ways:
- **Type Definition**
  - The `Task` interface provides a concrete structure that any object labeled as a `Task` must adhere to:

    ```
    interface Task {
        id: number;
        name: string;
        type: TaskType;
        dueDate: Date;
    }
    ```
- **Array Type Definition**
  - The `Task` interface also defines the type for an array of tasks:

    ```
    let tasks: Task[] = [];
    ```
  - Every object in the tasks array must conform to the Task structure.
- **Function Parameter and Return Types**
  - When functions are defined, TypeScript allows type specifications for parameters and return values. The Task interface is used in this context too. For example:

    ```
    function addTask(name: string, type: TaskType, dueDate: Date):
    void {
        const newTask: Task = {
            id: Date.now(),
            name,
            type,
            dueDate
        };
        tasks.push(newTask);
    }
    ```
  - Here, newTask is explicitly typed as a Task, ensuring its conformity to the defined structure.

Web Page Screen Shot

- 

localhost:3000

Training Wiki | Spider-Cat Board | STARS Repo | Sites | SQL Server Creds | Sprint Board | Sage HR

Simple TypeScript Task Manager

Task Name: Task 6

Task Type: Work

Due Date: 14/09/2023

Add Task

- Task 1 (Work) - Due: Thu Sep 14 2023 Complete Work
- Task 2 (Cooking) - Due: Thu Sep 14 2023 Cook
- Task 3 (Hobby) - Due: Thu Sep 14 2023 Engage Hobby
- Task 4 (Work) - Due: Thu Sep 14 2023 Complete Work
- Task 5 (Personal) - Due: Thu Sep 14 2023 Address Personal