Workshop

Task

Create a simple **Single Page Web Application** (SPA) with TypeScript:

* Separate your project files into **Modules** and import them for better modularity
* You can use this fake API for testing (<https://jsonplaceholder.typicode.com/guide/>)
* How the app looks is not important, the main idea is to train working with TypeScript in a project setting.
* You will also probably need a development server to host your app to see it in the browser, you can use any you find appropriate, like live-server, http-server, vite, etc.

## **Framework**

To give you some framework to guide you for the workshop, you can try implementing the following in your project:

* CRUD services – creating, reading, updating and deleting data from a remote server (i.e. <https://jsonplaceholder.typicode.com>)
  + Create an **UsersService** that can make CRUD operations to the API’s users endpoint (<https://jsonplaceholder.typicode.com/users>)
* A simple way of displaying the information from the CRUD services – you can use basic DOM element manipulations to display data in the browser
  + **getElementById** - <https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById>
  + **innerHTML** - <https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML>
* A simple router – a way to navigate in your app, we can use the following browser APIs to create a simple router
  + **window.history** - <https://developer.mozilla.org/en-US/docs/Web/API/Window/history>
  + **window.location -** <https://developer.mozilla.org/en-US/docs/Web/API/Window/location>

**NOTE:** You don’t need to follow the above structure, it’s just there if you need direction during implementation.

Goal

The main aim is to use what we have learned from the TypeScript course in the project:

* Use as much **typization** as possible
* Create **classes** and extract **interfaces** or **type aliases** for models you work with, to provide easier readability
* Use **generics** – for example try to create a generic CRUD service that can work with different types (i.e. Users, Posts, etc)
* Use **inheritance** – You can inherit the generic CRUD service to reuse code and implement specific services like the UsersService
* Using **decorators** – you can try adding validations to User fields (like name, username, email, etc.) using decorators
* Use anything else you find appropriate, **show what you have learned :)**