MyFace MVP - TypeScript

Learning Goals

- Semantic HTML
- CSS
- The DOM

Bonus Goals

- Progressive Enhancement
- Responsive Design

This week we are going to be building our own social network - MyFace!

Our app is quite small, but it will have all the main features. You'll be able to:

- View a feed of all the posts on the site.
- View each user's profile, and see their activity
- Like or dislike any of the posts
- Create new posts and new users.

We've made a start for you, and most of the 'logic' of the app is already in place. Your task is to add a nice user interface so that people can easily interact with the application.

Set Up

Start by forking https://github.com/techswitch-learners/myface-typescript . Then clone **your fork** into your local PC.

Like with many projects, the instructions for getting it up and running can be found in the README file (also displayed on the GitHub page for the project)

Follow the instructions there to get everything up and running.

Part 0 - Getting to know the App

Let's start by taking a quick look around the codebase and seeing if we can figure out where the key things are.

Firstly, just like BusBoard, MyFace is an MVC-like app built using TypeScript and a framework called express. With MVC apps, a good place to start is *usually* with the controllers (or Routes in express), as these tell us what things the app is able to do.

In our express app, you'll find a set of routes for dealing with 'Posts' and another for dealing with 'Users'. From just these 2 files we should get a decent understanding of our app. Within each of these there are various 'actions' that provide the functionality of our application.

There are a mix if `.get()` actions which work some things out and then return some HTML to the user, and `.post` actions which update something and then redirect the user back to another page.

See if you can work out what some of these actions do (don't worry too much about the service or repo files - these cover Database interactions that we'll cover later on in the course).

Part 1 - HTML

We can get almost all of the functionality of our site from HTML alone. Today we are going to try doing just that! Don't add any CSS or JS for now - we'll add that later in the week.

Let's start by improving the page that lists all the posts. You will find the code for this in the `postRoutes.ts` file under `router.get("/")`. This router takes the HTTP, finds posts from the database, and then returns the `post_list` view. (which you can find under `Views->post_list.ejs`).

As a first step, let's extend our EJS template so that our page now returns all of the content from the posts. Try to make your HTML as descriptive as possible using semantic HTML elements.

Next, can you link to the next and previous pages? This is already done somewhere on the site!

Your site probably won't look very pretty yet, but the key functionality of this page should already be there.

When you are done, check in with your trainer to see if you can make your HTML even more semantic.

Forms

Next lets move onto the page for creating a new post.

This one is slightly different, because we want to be able to send data back to the server. For this, we are going to need a form!

One form already exists - see the `create_user.ejs` and the corresponding 'post' action in `user_routes.ts`. Can you figure out how the inputs and attributes in the form link up with our routes?

See if you can create a similar form for creating a new post.

As always - ask for help if you are unsure!

Validation Errors

When a user submits a form, we need to check if the form is 'valid' - ie it contains all the information it is meant to contain and that the information provided has the right format etc.

There are 2 places this can happen:

- client side (ie happens in the browser as you fill in the form)
- server side (ie the app checks the form after you submit it)

Client side validation is great for giving clear and quick feedback to users… but it is always possible for users to edit things and find ways around this validation.

Server side is much more reliable (the user can't get around it) but it tends to be a bit slower.

Most apps will therefore do both - client side to help users best, and server side to ensure we always have valid data.

Our server already does validation (see what happens if you try to submit a form without filling in all the fields). Our job is to add some client side validation to improve the user experience.

Simple Client Side Validation

Most browsers will do some client side validation for you:

- try adding `required` to an input and then trying to submit the form without filling it in
- on the email field for `create_user`, try adding `type="email"` and `required` and then entering something that isn't an email.

In this case, all the fields are required, so make sure this is reflected in our form.

Extensions

If you still have time, give some of these things a try.

Screen Readers

Screen readers are how many people with no or limited vision interact with a computer. There are quite a lot of screen readers to choose from, but one popular, free option is NDVA.

Download it and see what happens if you try to use it on your site.

How did it go? Are there any changes to your site that would make it easier to use?

Liking a Post

Can you add buttons for 'liking' and 'disliking' a post.

Hint: You should be able to do this with pure HTML by adding a form for each button, with a hidden field for the postId.

Fill in the other pages

Now we've got a couple of pages working, see if you can repeat it for the other pages in the site.

Part 2 - CSS

The next step is to begin using CSS to style your pages.

We are going to be using an approach called mobile first design. Try the following pattern while developing.

1. Start at the smallest screen width you'd like to support (probably 320px for a small smart phone)
2. Use CSS to make your page look great.
3. Gradually make the screen wider until your design starts to break.
4. Add a 'breakpoint' using media queries
5. Go to step 2 and repeat until you reach the largest support screen size.

Try this approach on the `post_list` page.

Hint: Try working on small sections of the page at a time - often this is easier than trying to do everything all at once.

CSS tips:

- To target an html element use something like `h1 {}`
- To target a class use `.<CLASS_NAME> {}`
- Import one css file to another using `@import "<FILE_PATH>"`
- Use media queries to control behaviour at different screen sizes e.g. `@media only screen and (max-width 375px) {}`
- Use developer tools in your browser to easily change the width of your screen and view your html/css!

Part 3 - Implementing a Design

We've been given some designs here for the user profile page.

See if you can use what we've been practicing to make a start on implementing some of the details from this design.

You probably won't be able to finish everything in just one day, so try to pick one section of the design to focus on - perhaps just the cover & profile images, or just the cards for the posts etc.

Part 4 - Adding JavaScript

HTML and CSS can do a lot, but for some more dynamic functionality you'll often need to reach for JavaScript.

Changing Styles

Add a button to the `post_list` page. When this button is clicked, we'd like it to set the background colour on each of the posts to be a random colour!

Hint: Selecting elements

The `document` object has many methods for finding elements on a page. Which is likely to be the best for us to use?

```
document.getElementById('the-id') // Gets a single element matching a
particular ID
document.getElementsByClassName('the-class') // Gets a list of all
elements matching the class
document.getElementsByTagName('div') // Gets a list of all elements
matching the tag.
```

Hint: Generating a Random Color

You can set the background style of an element to an RGB color like this:

```
myElement.style.background = "rgb(100, 100, 100)";
```

To make this a random color instead, we just to pick random numbers between 0 and 255.

```
myElement.style.background = `rgb(${Math.random() * 255}, ${Math.
random() * 255}, ${Math.random() * 255})`
```

Mobile Menu

The designs we looked at yesterday included a mobile menu that appears over the top of the existing elements. Let's see if we can build something like this (don't worry about making it pretty though).

Steps:

- Add the menu to the page, for now just make it always display, and make sure that it displays 'on top' of the rest of the page.
- Add a button to the page (later this will open and close the menu)
- We want the page to start with the menu hidden… so in the CSS add the `display: none` property to make the menu disappear
- Now create another class that will make the menu re-appear if its added. (Make sure that this has a more specific selector that the regular class, otherwise it won't have any effect)
- Finally, add some JavaScript so that clicking the button toggles adding and removing this new class. If everything is correct your menu should now work!

Extension Steps

- Make it so that clicking outside the menu also causes it to close
- Make it so that clicking the Escape button also causes the menu to close
- Add an animation to the menu appearing and disappearing. (Note: you can't animate the `display`property, so you'll need a different way initially hide the menu)

Stretch goals

Stretch goals

More Form Validation

As well as getting the browser to help validate our form inputs, we can use JS to check that the inputs are valid. Using JS is more work… but gives us full control over how we want the validation to work.

For the `create_user` form, see if you can:

- disable the submit button until all the fields are valid
- add validation for the username field (a username should be all lower case, and include no spaces)

(It's actually possible to do both of these things without JS - if you have time, see if you can figure out how!)

Liking and Disliking Posts

Add 'like' and 'dislike' buttons to the posts. Can you make it so that clicking these buttons submits your 'like' to the app, and updates the displayed liked count… but doesn't require the browser to refresh the page?

Hint: You'll need to use the fetch function to submit the form in JS and handle the response.