Towa Code Challenge Documentation

The framework choice

The code challenge requires that the slider is made up with **Web Components** / **Web Elements** together with **Typescript**. I started to research how to combine both technologies into one Carousel with all the requirements (responsive, mobile friendly, etc.). Unfortunately, I faced big difficulties as I do not have vast experience with **Web Components**. In the end, I spent a lot of hours of work which did not lead to satisfying results.

Therefore, I decided to pivot to a more familiar solution, considering the very short deadline. Then I decided to use **React Js** and create the **React Components** with **Javascript**. As I'm familiar with this technology, the work started to flow and progress was quickly made.

In addition, React automatically **minifies the CSS and JS**, one of the requirements of the project.

The Carousel choice

To create the carousel, I decided to use Swiper (https://swiperjs.com/swiper-api). This carousel can be used in many frameworks (React, Web Components, etc,) and provide a lightweight solution that includes all the features needed for this project:

- Pagination buttons with position indicator
- Arrow buttons to control the slides
- Fluid design (responsive)
- Gestures capability for touch devices
- Does not bring any dependencies

In addition, it has a very stable and fluid animation, does not break up in case of screen resize, is adapted to old browsers and does not become unstable in real mobile devices.

The CSS

The CSS was made using **SASS** which, together with React, **minifies** and adds all **vendor prefixes** to the CSS, which are requirements of the project.

The structure of the SASS folder was made taking into consideration the **ITCSS** (simplified version) and the code was written using the **OOCSS** approach. All the CSS classes were created using the **BEM** methodology.
To make the CSS code **scalable and maintainable**, the following were created:

- **Variables**: for sizes, spacing and breakpoints.
    - with the green color from the button, I used Microsoft Fluent Design to build up a color pallet.
- **Mixin**: was created to control the breakpoints

For a future development, an idea could be to make use of variables and mixing for the texts and titles in the project.

The Components

The components structure for the project was made taking into consideration the **Atomic Design** Methodology, a project requirement. Please note that due to the relatively small scope of this project, it is not possible to apply all the Methodology, however, it can be seen in the button's components. The button can be reused in other areas of the project if necessary, with a secondary version by using the class --*secondary*.

Open Issues

The **Ajax** to read the online Json available was not implemented. For some reason, I don't know yet, the method ***map()*** was not working with the URL provided (https://dummyjson.com/products). I spent around 2 hours trying different solutions but I did not find the problem. Therefore, I pivoted and decided to save and import the JSON directly from the project folder structure. If we had more time available, I would for sure debug this problem.

The **Testing Framework** was not implemented. As I do not have experience with a testing framework, it will require some extra time for implementation. React works super well with **Jest Framework** (https://jestjs.io/docs/tutorial-react) but I would need some time to read the documentation and create the tests.

The project is using **Google Fonts** as a dependency. Once I wanted to use a **@fontface** in the project, I decided to accept this dependency. In a real project, I will ask the designers for the font files and add them in the project folders structure.

Lint

For the project, I used 2 lint softwares to analyze and fix the code:

- **ESlint**: the lint software made for React projects (https://eslint.org/docs/latest/use/getting-started)
- **Stylelint**: a lint build for CSS, in order to check the SCSS files (https://stylelint.io/user-guide/get-started/)

General Information

- I added a **footer** and **header** to increase the quality of the presentation.
- As there is no access to Figma or a similar app for the layouts, this project has no **pixel perfect** approach.
- The Json brings 30 items. I limited to 4.
- The Carousel has autoplay and 5 seconds for each slide.
- The Project can be accessed online at https://645e19fb0be5e37a2bf81bd9--starlit-hummingbird-ec344b.netlify.app/
- I copied the favicon from the Towa website. 🙂

Further Development

In case this was a real life project, the following steps would be necessary for quality improvement:

- The progress pagination bar does not have the smoothest animation. It can be improved, possibly doing it directly in the JS and not with the current CSS implementation.
- The Carousel buttons were made in css. Maybe using a svg icon might be a better solution.
- Create a function to limit the number of characters from the JSON, in case the text is too big it might break the mobile layout.
- Check with the designers the size of the buttons for mobile and tablet. It might be too small for the users.
- Check with the designers a solution to the cases when an image has a white background. It makes the text not readable.
- Google Fonts: ask the designers for a package with the fonts to avoid a project dependency.
- Check with the client the quality of the images in the JSON. They are really low resolution.
- Check with the client the possibility to have images in different resolutions in the JSON, so we can use **<picture>** tag and improve the project performance.
- Change the **Stylelelint** preferences to make it more suitable for our project specifications.