**1. What issues did you think about in choosing your approach to this task?**

For this challenge, I focused on scalability, responsiveness, and user experience.

I prioritized efficient handling of large datasets, ensured the table and pagination worked well across screen sizes, and added clear visual indicators for sorting.

Tailwind helped me to achieve a clean and responsive design.

I also included dynamic features like customizable items per page adding validations and restrictions in case the user make some mistakes writing the items per page number.

**2. How can we ensure the table remains usable on smaller screen sizes such as phones?**

This combination of responsive design and dynamic features ensures the table is functional and user-friendly across all screen sizes:

**Horizontal Scrolling:** The *table-fixed* class ensures the table layout remains consistent, while the responsive design from Tailwind's *min-w-full* ensures the table spans the full width of the container. For smaller screens, users can scroll horizontally if the content exceeds the screen width.

**Pagination Flexibility:** The pagination section uses *flex-wrap*, allowing buttons to move to a new line when there are too many to fit on one row, preventing overflow issues.

**Dynamic Items per Page**: The items per page input gives users control over the number of rows displayed, letting them manage the table's height on smaller devices.

**3. What would be your next steps to make this into a generic, reusable component that can display data from an API?**

To make this a generic, reusable component that can display data from an API, my next steps would be:

* Refactor the table to accept props like columns, data, itemsPerPage, and onSort to make it flexible for different datasets and APIs.
* Use a columns prop to define column headers, field mappings, and custom render functions for specific data formats.
* Add a fetchData prop or hook that allows the parent component to control how data is loaded, enabling support for server-side pagination and sorting.
* Add built-in handling for loading spinners and error messages when fetching data from APIs.
* Expose callbacks for actions like onPageChange and onSortChange so that parent components can handle external state updates or API calls.
* Allow pagination to be customized (display style, number of buttons visible) or even disabled for datasets that don’t require it.