Responsive Tables

Alexander Kassil, Daniel Hevesy-Szettyan, Dominik Bauer, and Miloš Globočki

706.041 Information Architecture and Web Usability 3VU WS 2024/2025
Graz University of Technology

13 Dec 2024

Abstract

Responsive tables are an essential component of web design, ensuring data accessibility and usefulness across devices with different screen sizes. This survey investigates ways to develop responsive tables, focusing on responsive table patterns like stacking, flipping, and pagination, and best practice patterns for responsive tables like sort, search, persistence, and more. The tools and techniques used to implement these concepts are examined, with an emphasis on usability, performance, and flexibility. By comparing solutions like AG-Grid and Handsontable, insights are provided into how to choose the best approaches and technologies for responsive table implementation.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

With the rise of mobile device usage, the demand for responsive web design has increased dramatically. Responsive web design seeks to produce web pages that deliver an ideal user experience across a variety of devices, including smartphones, tablets, laptops, and desktops [Marcotte 2014]. This design philosophy emphasizes accessibility and usability, ensuring that the content is usable and visually appealing regardless of screen size or orientation.

Tables present distinct issues when it comes to adapting to various device formats. Tables are commonly used to convey organized data, but their traditional shape is sometimes unsuitable for smaller screens, as columns and rows may extend beyond the viewable space. This might cause usability concerns, such as asking users to scroll horizontally or zoom in on selected portions. Addressing these problems is critical to providing a consistent user experience [Mozilla 2024b].

Responsive tables address these restrictions by adopting strategies that dynamically modify their layout. These techniques include the following:

- *Container Queries*: Adjusting table dimensions based on the parent container's width and height.
- *Flexbox Layouts*: Using flexible box layouts to dynamically rearrange table elements.
- *Horizontal Scrolling*: Allow users to scroll horizontally to view wide tables without compromising data readability.

The usefulness of responsive tables goes beyond technological adaptability. Responsive tables also simplify development and maintenance by allowing a single codebase to serve numerous devices. Furthermore, it is important to make sure that responsive tables remain accessible for people with disabilities by ensuring compliance with standards such as the Web Content Accessibility Guidelines (WCAG).

This survey presents an in-depth analysis of responsive table design, highlighting the patterns and tools accessible for implementation. Practical approaches for adjusting tables to different screen sizes are explored, as well as basic good practices for table usability. The survey then compares popular tools, such as AG-Grid and Handsontable, to help developers make sensible choices.

Chapter 2

HTML Tables

HTML (HyperText Markup Language) defines the structure and semantics of web content, CSS handles the visual styling, and JavaScript handles interactivity and functionality. The word "hypertext" refers to a network of interactions that connect online pages, allowing movement inside a single site or across many sites [Mozilla 2024a].

2.1 Traditional HTML Tables

HTML tables are basic components of web design that show data organized in rows and columns. Despite their simplicity, classic HTML tables are essentially static, making them unsuitable for current responsive web design and reducing the accessibility options that the developer may want to include. As a result, adapting tables to dynamic layouts has become a top priority for developers.

The table in Listing 2.1 shows the most basic implementation of the table in HTML. This code snippet shows the basic structure that every table in HTML should have, and that it should consist of <table>, <tr>, <th>, and <td> HTML elements, with their closing tags, respectively. While this structure is effective for fixed layouts, it presents challenges in responsive design, particularly on smaller screens. Issues such as overlapping content, excessive scrolling, and loss of readability are common [W3Schools 2024].

2.2 Techniques for Responsive Tables

To address these challenges, developers use various techniques to make tables responsive. Some of them are listed below, and others are discussed in the following chapters:

- *Stacking*: Converts table rows into vertically stacked blocks for narrow screens.
- *Collapsible Rows/Columns*: Provides an interactive way to hide or reveal specific rows or columns.
- *Reflowing Content*: Rearranges table content to fit the available space dynamically.

CSS and JavaScript play an important role in the transformation of static tables into responsive components. Without JS and CSS, any table would be static and would not be able to react to the different screen sizes and devices in use. With CSS Container Queries, developers can adjust table styles based on the parent container's size, which gives them the advantage of having only one code-base for multiple devices. JavaScript Plugins add interactivity, such as sorting, filtering, and pagination, to the said tables.

```
<table>
<thead>
  <tr>
    <th>Specification</th>
    <th>Details</th>
    <th>Value</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>Engine Type</td>
    <td>Displacement</td>
    <td>3.0 L I6</td>
  </tr>
  <tr>
    <td>Power</td>
    <td>Torque</td>
    <td>335 hp / 500 Nm</td>
  </tr>
</tbody>
</table>
```

Listing 2.1: A basic HTML table without any CSS styling.

Chapter 3

Responsive Patterns for Tables

A responsive pattern is a good practice technqiue for implementing responsive behavior for tables. With these patterns, the appearance and behavior of HTML tables can be controlled depending on the device on which they are displayed. Tables ideally should provide the same usability and functionality on both mobile devices and PCs. Some examples of commonly used responsive table patterns include stacking, squishing, flipping, pagination, accordion rows, column and row toggle, horizontal scrolling, as well as more advanced techniques such as lazy loading and virtualization.

3.1 Stacking

The Stacking pattern converts each row of the table into a vertical stack, with cells in a displayed one below the other. The initial table headers are displayed as labels alongside the table data for the respective row. This maintains readability and usability even for small devices, and users can view all the content one row at a time. It is typically implemented only using CSS, without the need of additional JS code. This is another advantage because it simplifies the implementation and ensures that stacking will work in most environments, even when the execution of JS is disabled. One disadvantage is that it is difficult to compare the table content of separate rows, because in the stacked view, only one row at a time is visible. This makes the pattern impractical in situations where the focus is mainly on comparing values.

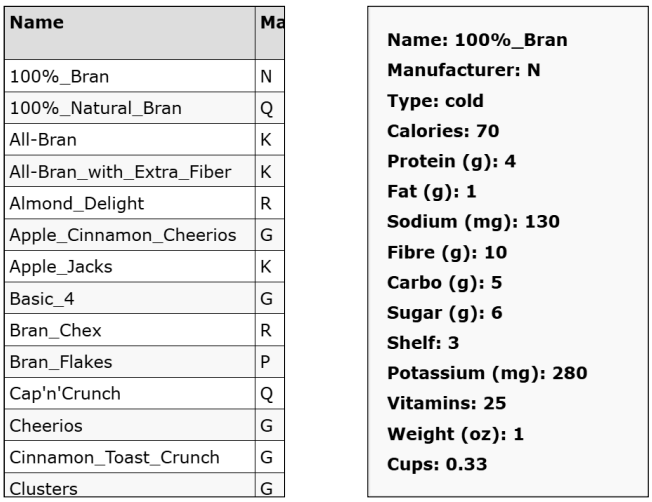
Figure 3.1 shows part of a data table with 15 columns. Due to the resolution of mobile devices, most columns would be cut off if stacking was disabled. With the enabled pattern, users can view the content of one row without truncated text.

3.2 Squishing

The simplest way to make a table responsive is to change its width. One method to achieve this is to decrease the width of each column to such an extent that the content just remains readable. The text is abbreviated, using methods such as truncation, truncation with ellipsis, compression, or line wrapping. Only a few lines of CSS are necessary to handle how the text is abbreviated, which makes the implementation of the pattern trivial. This straightforward implementation is the greatest advantage of the pattern and there is no reason to exclude it from any HTML table. Ideally, squishing allows all columns to remain visible, making it ideal for tables where maintaining the overall structure is crucial, but in practice this is often not the case and additional patterns need to be considered.

3.3 Flipping

The Flipping pattern transposes the HTML table by swapping rows and columns to better fit smaller or wider screens. When flipped, the table headers are displayed vertically, potentially making it possible for



Name	Ma
100%_Bran	N
100%_Natural_Bran	Q
All-Bran	K
All-Bran_with_Extra_Fiber	K
Almond_Delight	R
Apple_Cinnamon_Cheerios	G
Apple_Jacks	K
Basic_4	G
Bran_Chex	R
Bran_Flakes	P
Cap'n'Crunch	Q
Cheerios	G
Cinnamon_Toast_Crunch	G
Clusters	G

(a) Stacking disabled.

Name: 100%_Bran
Manufacturer: N
Type: cold
Calories: 70
Protein (g): 4
Fat (g): 1
Sodium (mg): 130
Fibre (g): 10
Carbo (g): 5
Sugar (g): 6
Shelf: 3
Potassium (mg): 280
Vitamins: 25
Weight (oz): 1
Cups: 0.33

(b) Stacking enabled.

Figure 3.1: Stacking: The same table with and without stacking enabled displayed on a mobile device. [Screenshot taken by the author of this paper.]

users on narrower screens to see the whole of a wide table. This pattern is only useful for tables with many columns and fewer rows, as shown in Figure 3.2a, otherwise the transposed table becomes too wide and cannot be viewed on one screen. To overcome this issue, it can be combined with pagination (see Section 3.4) to limits the number of rows shown. Flipping is implemented in CSS and JS because the entire table needs to be removed from the DOM and rebuilt again in the transposed form. Flipping can be automatically triggered by screen size detection through container queries, or manually activated by the user with a button.

Figure 3.2 shows the same table in its standard form and its transposed form. For mobile devices, it would be necessary to show the flipped version of the table and additionally implement the squishing pattern to abbreviate longer text in the name row.

3.4 Pagination

The Pagination pattern splits table content into multiple pages, reducing the amount of content displayed at once. It is particularly useful for large data tables that would otherwise require users to scroll to view all the data. In combination with other patterns, such as flipping and squishing, pagination highly improves the usability and viewability for both mobile and PC users. Implementing pagination using plain CSS and JS can be a more complex task. Therefore, other tools (see Chapter 5) are often considered when implementing this pattern. Figure 3.3 displays a single page of a data table containing over 70 rows, which otherwise would be challenging to present clearly.

3.5 Accordion Rows

This pattern ensures that additional information is only displayed when the user needs it. By default, one item per row is shown, typically with short key information. On mobile devices and smaller screen sizes, this saves space and guarantees better readability. Additional rows remain hidden until the user interacts by pressing or clicking the specific item. The expanded content is often displayed below the item in a stacked format with headings for clarity. The Accordion Rows pattern can be implemented in pure CSS. It ensures a clean and compact layout on smaller devices while preserving access to detailed information through user interactions.

Name	Manufacturer	Type	Calories	Protein (g)	Fat (g)	Sodium (mg)	Fibre (g)	Carbo (g)	Sugar (g)	Shelf	Potassium (mg)	Vitamins	Weight (oz)	Cups
100%_Bran	N	cold	70	4	1	130	10	5	6	3	280	25	1	0.33
100%_Natural_Bran	Q	cold	120	3	5	15	2	8	8	3	135	0	1	N/A
All-Bran	K	cold	70	4	1	260	9	7	5	3	320	25	1	0.33
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14	8	0	3	330	25	1	0.5

(a) Standard Table.

Name	100%_Bran	100%_Natural_Bran	All-Bran	All-Bran_with_Extra_Fiber
Manufacturer	N	Q	K	K
Type	cold	cold	cold	cold
Calories	70	120	70	50
Protein (g)	4	3	4	4
Fat (g)	1	5	1	0
Sodium (mg)	130	15	260	140
Fibre (g)	10	2	9	14
Carbo (g)	5	8	7	8
Sugar (g)	6	8	5	0
Shelf	3	3	3	3
Potassium (mg)	280	135	320	330
Vitamins	25	0	25	25
Weight (oz)	1	1	1	1
Cups	0.33	N/A	0.33	0.5

(b) Flipped Table.

Figure 3.2: Flipping: The same table in its standard form and its flipped form. [Screenshot taken by the author of this paper.]

Filter: Filter by name...

Name	Manufacturer	Type	Calories	Protein (g)	Fat (g)	Sodium (mg)	Fibre (g)	Carbo (g)	Sugar (g)
100%_Bran	N	cold	70	4	1	130	10	5	6
100%_Natural_Bran	Q	cold	120	3	5	15	2	8	8
All-Bran	K	cold	70	4	1	260	9	7	5
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14	8	0
Almond_Delight	R	cold	110	2	2	200	1	14	8
Apple_Cinnamon_Cheerios	G	cold	110	2	2	180	1.5	10.5	10
Apple_Jacks	K	cold	110	2	0	125	1	11	14

« 1 2 3 4 5 6 7 8 9 10 11 »

Figure 3.3: Pagination: Table with enabled pagination, sorting and searching pattern. [Screenshot taken by the author of this paper.]

3.6 Column and Row Toggle

The Column and Row Toggle pattern hides specific content. This can be done automatically when there is not enough room for the whole table or manually by users. It allows hiding less critical columns or rows and emphasizes more important content. It is crucial to provide users with a form of feedback as to which content is hidden and how it can be shown again. This is typically done with buttons or dropdown menus beside the table. The pattern can be implemented using pure CSS, which is beneficial for maintaining simplicity and avoiding dependency on JS.

3.7 Horizontal Scrolling

The Horizontal Scrolling pattern allows users to scroll horizontally to view content that does not fit the width of the screen. Typically, a horizontal scroll bar is placed somewhere in or near the table. Compared to all other listed patterns, horizontal scrolling truly preserves the full table structure and ensures that all columns remain accessible without truncating or hiding the data. It is particularly useful for tables with a large number of columns or detailed content. Similar to Squishing, only a few lines of CSS code are needed, making the pattern trivial to implement. One downside of this simple pattern is that, for touch display users, the interaction with a scroll bar can be tricky and frustrating. A different pattern should be used on these devices.

3.8 Lazy Loading and Virtualization

Lazy loading and virtualization are techniques that generally delay the loading of resources until they are needed. In the context of tables, only the visible rows are rendered initially, while additional rows are dynamically loaded when the user needs it. This approach significantly improves performance, especially when dealing with large datasets, by reducing initial load times and memory usage. It can be implemented manually with JS or, more commonly, using other tools (see Chapter 5).

The differences between the two patterns are:

- *Lazy Loading*: Lazy Loading typically makes multiple API calls that retrieve only a subset of the needed data. For example, when a user scrolls through a table, new data is requested from the server all the time, reducing the initial data load and server response size. This approach is ideal for large datasets that cannot be entirely stored in local memory.
- *Virtualization*: Virtualization only makes a single, larger API call to get all necessary data at once. The data is then stored in local memory, such as cache, and rendered dynamically based on what is visible in the table. Virtualization enhances rendering efficiency by presenting only the visible rows, which decreases DOM size and increases performance, without making repeated calls.

Chapter 4

Good Practice Patterns for Tables

A good practice pattern defines a good practice technique for implementing tables in general. Some examples include sortable columns, pinnable header, zebra striping, and natural cell alignment (text left, numbers right).

4.1 Sortable Columns

The Sortable Columns pattern does what the name says. It allows the user to sort columns either ascending or descending. Most of the time, indicated by small arrows next to the pattern that represents the sorting order. Listing 4.1 shows how this pattern might be implemented.

4.2 Filterable Rows

The Filterable Rows pattern requires the developer to add a search box that allows for quick filtering of data. This pattern is especially useful for quickly locating information in large datasets. Figure 3.3 shows an example search box in the top left corner. Listing 4.2 shows how this pattern might be implemented.

4.3 Infinite Scrolling

This pattern is used mostly for mobile devices. It replaces the pagination pattern and allows the user to scroll rather than switching pages. This is achieved by loading new rows as the user scrolls down. This proves to be much more intuitive and user-friendly than the pagination pattern on touch devices.

4.4 Show More Button

A Show More button is used to allow the user to explicitly request more data rows be loaded and displayed. This pattern is used to improve performance and improves usability, particularly on mobile devices.

4.5 Pinnable Header

While scrolling to large datasets, it can be easy to lose context, especially if the header rows (or columns) are no longer shown. The Pinnable Header pattern pins a header row (or header column) in place, as the rest of the data scrolls past. This means the header is always visible no matter how far is scrolled, allowing the user to easily identify which information belongs to which header.

```
function sortTable_hor_scr(column) {
  const headers = document.querySelectorAll('th.paginations_hor_scr');
  let sortedHeader = null;

  headers.forEach(header => {
    if (header.textContent.trim().startsWith(column)) {
      sortedHeader = header;
    } else {
      header.classList.remove('sorted-asc', 'sorted-desc');
    }
  });

  if (sortColumn_hor_scr === column) {
    sortDirection_hor_scr *= -1;
  } else {
    sortColumn_hor_scr = column;
    sortDirection_hor_scr = 1;
  }

  if (sortedHeader) {
    if (sortDirection_hor_scr === 1) {
      sortedHeader.classList.add('sorted-asc');
      sortedHeader.classList.remove('sorted-desc');
    } else {
      sortedHeader.classList.add('sorted-desc');
      sortedHeader.classList.remove('sorted-asc');
    }
  }

  filteredData_hor_scr.sort((a, b) => {
    const aValue = a[column] || 0;
    const bValue = b[column] || 0;

    if (typeof aValue === "number" && typeof bValue === "number") {
      return (aValue - bValue) * sortDirection_hor_scr;
    } else {
      return aValue.toString().localeCompare(bValue.toString()) *
    sortDirection_hor_scr;
    }
  });

  currentPage_hor_scr = 1;
  renderTable_hor_scr(currentPage_hor_scr);
  renderPagination_hor_scr();
}
```

Listing 4.1: JavaScript code to implement the Sortable Columns pattern.

```
function filterTable_hor_scr() {
  const query = document.getElementById('search-bar_hor_scr').value.toLowerCase();
  filteredData_hor_scr = data.filter(item => item.Name.toLowerCase().includes(query)
    );
  currentPage_hor_scr = 1;
  renderTable_hor_scr(currentPage_hor_scr);
  renderPagination_hor_scr();
}
```

Listing 4.2: JavaScript code to implement the Filterable Rows pattern.

4.6 Zebra Striping

The Zebra Striping pattern is used to improve readability by coloring the rows in alternating colors. It is easier for the user to distinguish between rows.

4.7 Natural Cell Alignment

By default, text should be left aligned in a cell and numbers should be right aligned. The Natural Cell Alignment pattern should make it easy to set the correct alignment.

4.8 Color Coding

The Color Coding pattern is used primarily for datasets with various categories. The pattern can help distinguish different categories or highlight important values by coloring them differently.

4.9 Persistence

The Persistence pattern ensures that any previously entered sorting or filtering is saved and restored upon revisiting or reloading.

Chapter 5

Responsive Table Libraries

A number of libraries are available to help provide responsive tables. Five such libraries were explored in the course of this survey.

5.1 Handsontable

Handsontable is a powerful JavaScript library for managing tabular data in spreadsheet-like tables [Handsoncode 2024]. Unlike other tools, it is designed for inline editing and enables spreadsheet-like behavior in data tables, so end users can interact with data directly. The library has built-in support for many responsive table patterns and good practice patterns like sorting and filtering. Handsontable is optimized for large data tables with its virtualization implementation. Figure 5.1 shows Handsontable with a dataset containing 1000 rows and 1000 columns, where the data is loaded on demand.

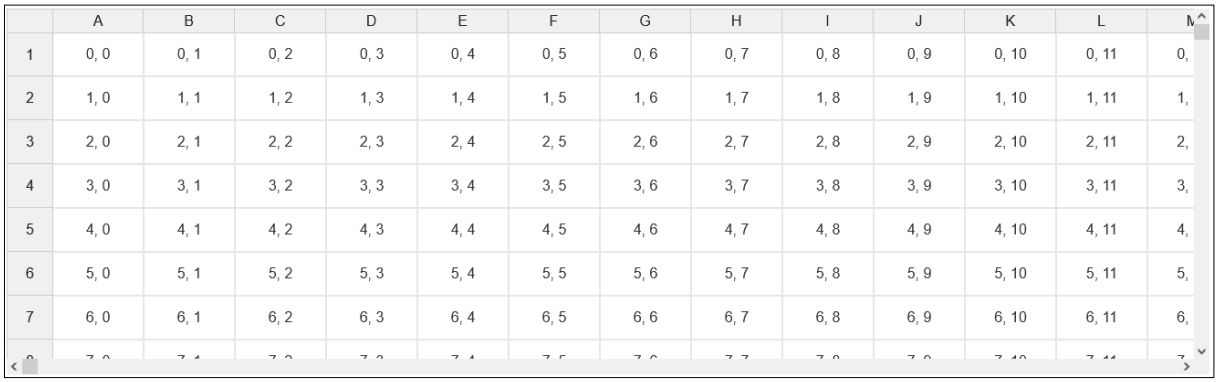
5.2 TanStack Table

TanStack Table is a lightweight, headless library for responsive data tables [TanStack 2024]. It offers developers the ability to customize every aspect of the behavior and appearance of data tables. Since it is a headless data table library, the tool does not provide predefined styles or UI components. TanStack Table supports out-of-the-box patterns such as squishing, pinnable headers, sorting, and grouping. The library itself is minimal in size, but still provides good performance with large data sets through lazy loading. Figure 5.2 shows TanStack Table with a large dataset and lazy loading.

5.3 AG Grid

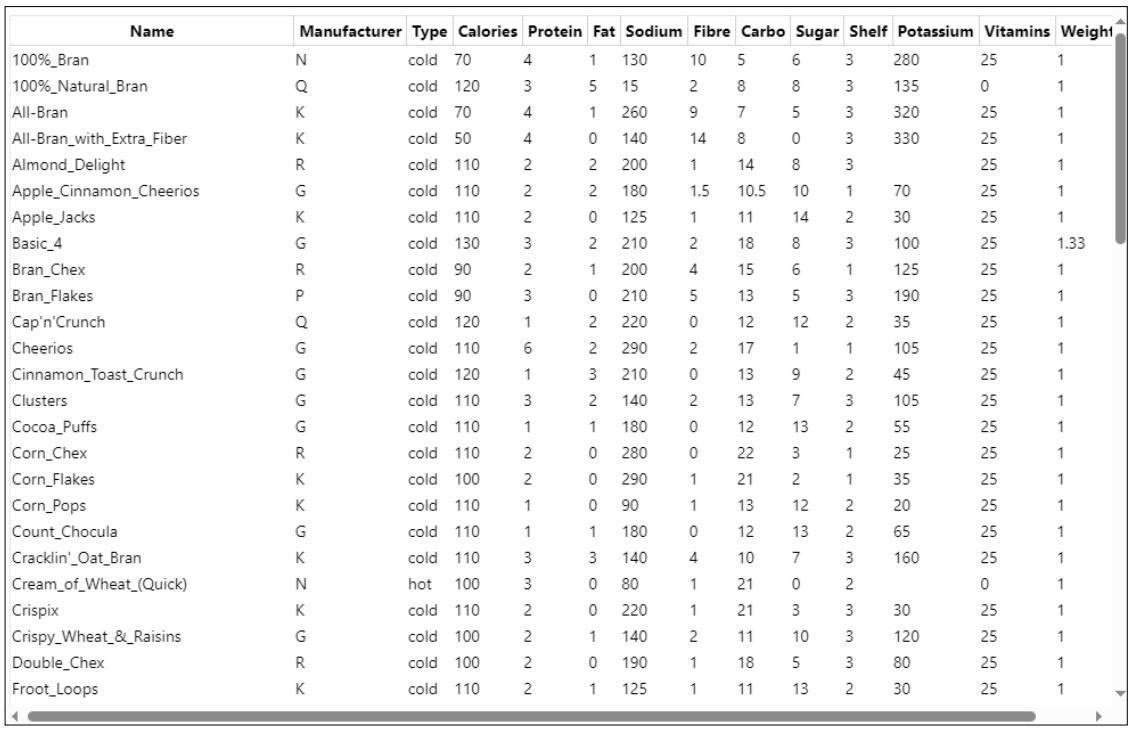
AG Grid is a fully featured grid solution tool with extensive customization options [AG Grid 2024a]. This library can handle large data tables and is optimized for high performance. In addition, AG Grid has extensive documentation and examples for each feature. Since AG Grid has a large community, it also has extensive community-driven plugins. AG Grid is available as a community edition and an enterprise edition. The Community Edition provides many responsive and good practice patterns. When using the default implementation of AG Grid, some responsive patterns such as horizontal scrolling are enabled by default.

The paid Enterprise Edition unlocks much more functionality to make large and complex data tables readable on different devices, such as pivoting and a side bar. AG Grid offers the opportunity to test these features in an non-production environment or for research purposes. Thus, companies can use a trial and only need to pay for these features when using them in production environments [AG Grid 2024b]. Figure 5.3 shows a responsive table with default configuration. It has filtering, sorting, a side-bar, and pivoting. Despite many features being available, the table does not appear overloaded.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0, 0	0, 1	0, 2	0, 3	0, 4	0, 5	0, 6	0, 7	0, 8	0, 9	0, 10	0, 11	0,
2	1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7	1, 8	1, 9	1, 10	1, 11	1,
3	2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	2, 8	2, 9	2, 10	2, 11	2,
4	3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	3, 8	3, 9	3, 10	3, 11	3,
5	4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	4, 8	4, 9	4, 10	4, 11	4,
6	5, 0	5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7	5, 8	5, 9	5, 10	5, 11	5,
7	6, 0	6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7	6, 8	6, 9	6, 10	6, 11	6,

Figure 5.1: Handsontable: Squishing, horizontal scrolling, and virtualization are enabled. [Screenshot taken by the author of this paper.]

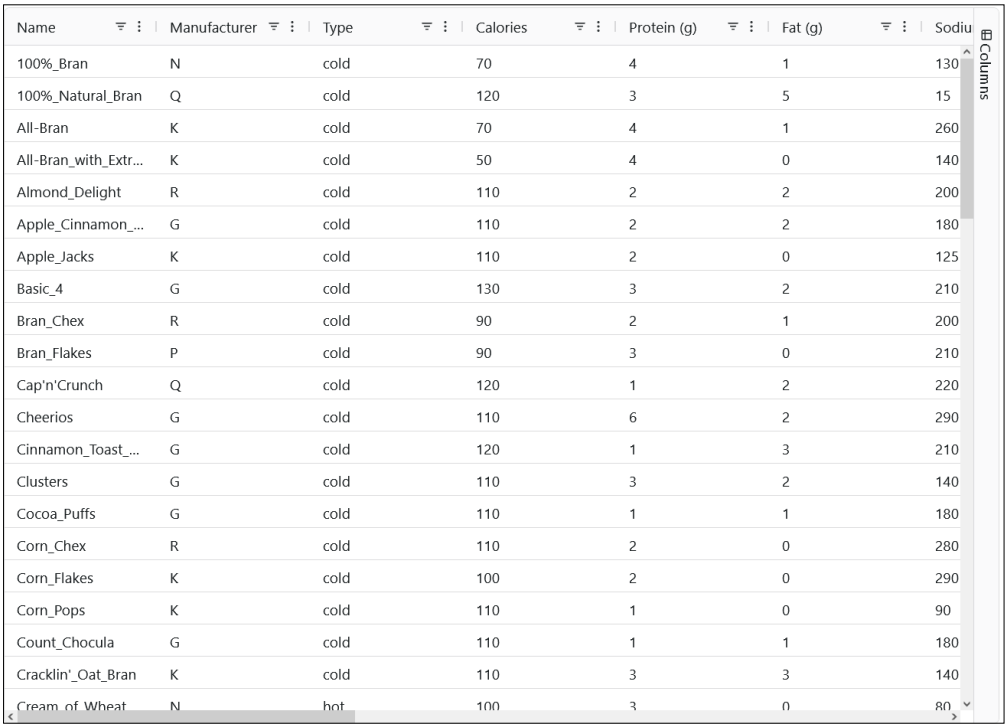


Name	Manufacturer	Type	Calories	Protein	Fat	Sodium	Fibre	Carbo	Sugar	Shelf	Potassium	Vitamins	Weight
100%_Bran	N	cold	70	4	1	130	10	5	6	3	280	25	1
100%_Natural_Bran	Q	cold	120	3	5	15	2	8	8	3	135	0	1
All-Bran	K	cold	70	4	1	260	9	7	5	3	320	25	1
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14	8	0	3	330	25	1
Almond_Delight	R	cold	110	2	2	200	1	14	8	3		25	1
Apple_Cinnamon_Cheerios	G	cold	110	2	2	180	1.5	10.5	10	1	70	25	1
Apple_Jacks	K	cold	110	2	0	125	1	11	14	2	30	25	1
Basic_4	G	cold	130	3	2	210	2	18	8	3	100	25	1.33
Bran_Chex	R	cold	90	2	1	200	4	15	6	1	125	25	1
Bran_Flakes	P	cold	90	3	0	210	5	13	5	3	190	25	1
Cap'n'Crunch	Q	cold	120	1	2	220	0	12	12	2	35	25	1
Cheerios	G	cold	110	6	2	290	2	17	1	1	105	25	1
Cinnamon_Toast_Crunch	G	cold	120	1	3	210	0	13	9	2	45	25	1
Clusters	G	cold	110	3	2	140	2	13	7	3	105	25	1
Cocoa_Puffs	G	cold	110	1	1	180	0	12	13	2	55	25	1
Corn_Chex	R	cold	110	2	0	280	0	22	3	1	25	25	1
Corn_Flakes	K	cold	100	2	0	290	1	21	2	1	35	25	1
Corn_Pops	K	cold	110	1	0	90	1	13	12	2	20	25	1
Count_Chocula	G	cold	110	1	1	180	0	12	13	2	65	25	1
Cracklin'_Oat_Bran	K	cold	110	3	3	140	4	10	7	3	160	25	1
Cream_of_Wheat_(Quick)	N	hot	100	3	0	80	1	21	0	2		0	1
Crispix	K	cold	110	2	0	220	1	21	3	3	30	25	1
Crispy_Wheat_&_Raisins	G	cold	100	2	1	140	2	11	10	3	120	25	1
Double_Chex	R	cold	100	2	0	190	1	18	5	3	80	25	1
Froot_Loops	K	cold	110	2	1	125	1	11	13	2	30	25	1

Figure 5.2: TanStack Table: Squishing, horizontal scrolling, and lazy loading are enabled. [Screenshot taken by the author of this paper.]

5.4 DataTables

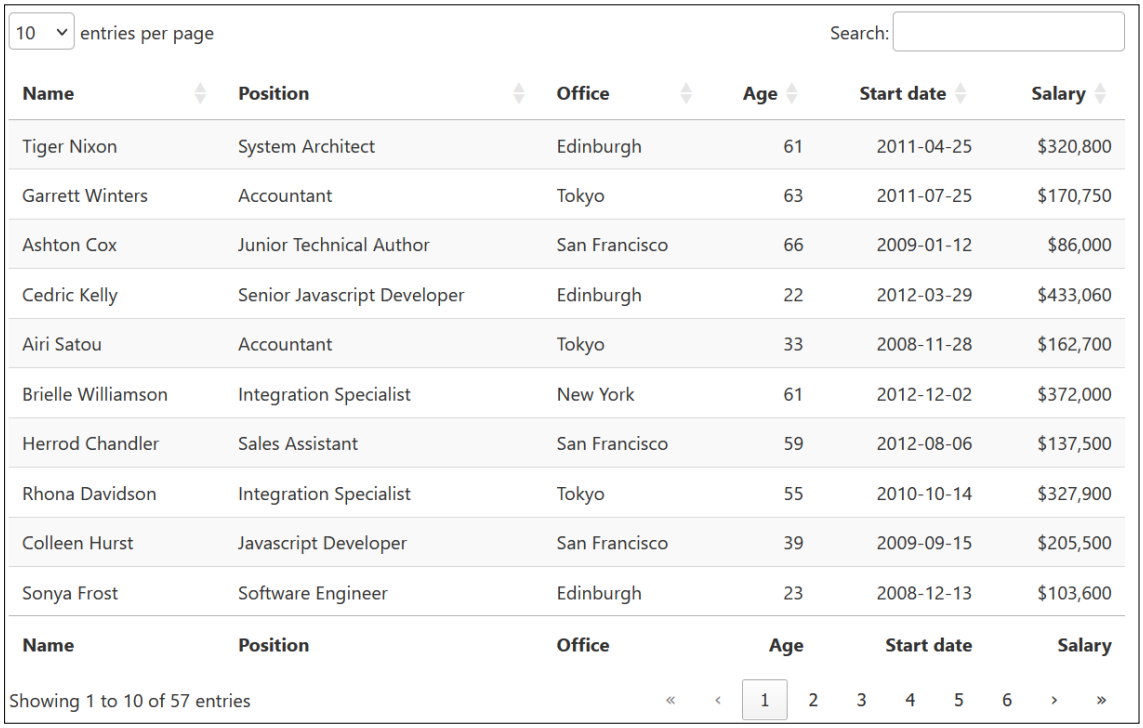
DataTables is a jQuery-based library for data tables that provides basic and advanced table functionality [SpryMedia 2024]. It is designed to improve readability of small to medium-sized HTML tables using responsive and good-practice patterns. The library is designed to work out of the box with existing HTML data tables and can be initialized with a few lines of JavaScript. Some patterns, such as pinnable headers, row grouping, and column resizing, need to be imported through extensions. Since DataTables works with jQuery, it is ideal for legacy web applications, which already use jQuery. To provide compatibility for modern frameworks, DataTables offers a variety of wrappers. Figure 5.4 shows an example from the documentation. It is an out-of-the-box configuration for a data table which has filtering, sorting, and pagination enabled by default.



Name	Manufacturer	Type	Calories	Protein (g)	Fat (g)	Sodiu
100%_Bran	N	cold	70	4	1	130
100%_Natural_Bran	Q	cold	120	3	5	15
All-Bran	K	cold	70	4	1	260
All-Bran_with_Extr...	K	cold	50	4	0	140
Almond_Delight	R	cold	110	2	2	200
Apple_Cinnamon_...	G	cold	110	2	2	180
Apple_Jacks	K	cold	110	2	0	125
Basic_4	G	cold	130	3	2	210
Bran_Chex	R	cold	90	2	1	200
Bran_Flakes	P	cold	90	3	0	210
Cap'n'Crunch	Q	cold	120	1	2	220
Cheerios	G	cold	110	6	2	290
Cinnamon_Toast_...	G	cold	120	1	3	210
Clusters	G	cold	110	3	2	140
Cocoa_Puffs	G	cold	110	1	1	180
Corn_Chex	R	cold	110	2	0	280
Corn_Flakes	K	cold	100	2	0	290
Corn_Pops	K	cold	110	1	0	90
Count_Chocula	G	cold	110	1	1	180
Cracklin'_Oat_Bran	K	cold	110	3	3	140
Cream_of_Wheat	N	hot	100	3	0	80

Columns

Figure 5.3: AG Grid: Squishing, horizontal scrolling, sorting, filtering, side bar, for a large dataset. [Screenshot taken by the author of this paper.]

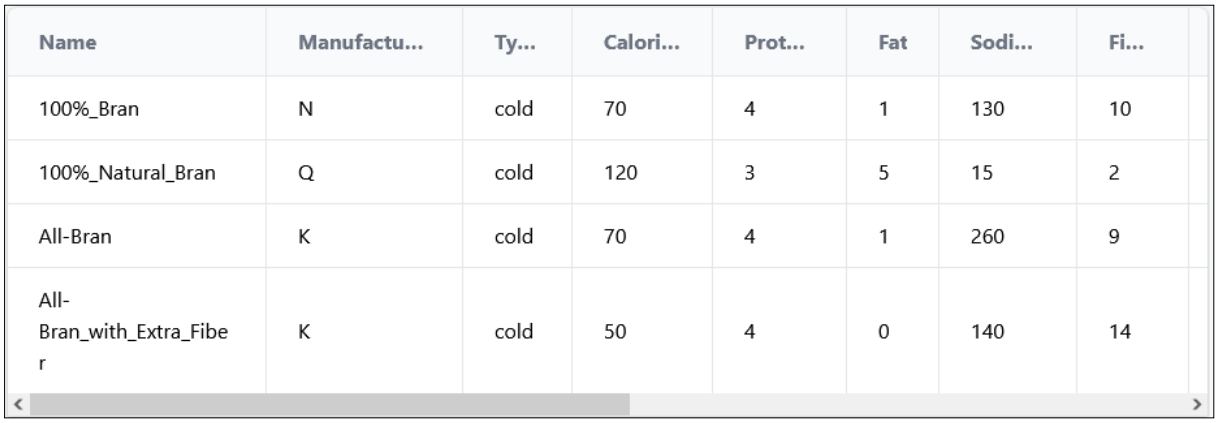


10 entries per page Search:

Name	Position	Office	Age	Start date	Salary
Tiger Nixon	System Architect	Edinburgh	61	2011-04-25	$320,800
Garrett Winters	Accountant	Tokyo	63	2011-07-25	$170,750
Ashton Cox	Junior Technical Author	San Francisco	66	2009-01-12	$86,000
Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012-03-29	$433,060
Airi Satou	Accountant	Tokyo	33	2008-11-28	$162,700
Brielle Williamson	Integration Specialist	New York	61	2012-12-02	$372,000
Herrod Chandler	Sales Assistant	San Francisco	59	2012-08-06	$137,500
Rhona Davidson	Integration Specialist	Tokyo	55	2010-10-14	$327,900
Colleen Hurst	Javascript Developer	San Francisco	39	2009-09-15	$205,500
Sonya Frost	Software Engineer	Edinburgh	23	2008-12-13	$103,600
Name	**Position**	**Office**	**Age**	**Start date**	**Salary**

Showing 1 to 10 of 57 entries « ‹ 1 2 3 4 5 6 › »

Figure 5.4: DataTables: Filtering, sorting, and pagination are enabled. [Screenshot taken by the author of this paper.]



Name	Manufactu...	Ty...	Calori...	Prot...	Fat	Sodi...	Fi...
100%_Bran	N	cold	70	4	1	130	10
100%_Natural_Bran	Q	cold	120	3	5	15	2
All-Bran	K	cold	70	4	1	260	9
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14

Figure 5.5: Grid.js: Horizontal scrolling is enabled. [Screenshot taken by the author of this paper.]

5.5 Grid.js

Grid.js is a lightweight JavaScript library to create minimalistic, responsive data tables [Usablica 2024]. It supports frameworks like React, Vue, and Angular using modern JavaScript. Grid.js only offers essential functionality for responsive tables, such as sorting, filtering, and pagination. This makes it a good choice for quickly implementing a small to medium-sized data table without requiring heavy dependencies. The setup is simple and only requires importing the library and passing data to a configuration object. Grid.js lacks more advanced functionality such as grouping, virtualization, or lazy loading, making it only recommendable for small data tables. Figure 5.5 shows an out-of-the-box configuration. Compared to other tools, not much functionality enabled by default. As shown in Figure 5.5, the width of the columns are not sized according to text length. In addition, no sorting or filtering is enabled.

Chapter 6

Discussion

The goal of responsive tables is to display data efficiently and easily on various different devices. It is often necessary to combine multiple patterns to achieve this. During this survey work, a combination of pagination with sorting, filtering, and column toggling was found to work well. However, pagination can be cumbersome to use on mobile devices and should perhaps be replaced by infinite scroll for better usability. For narrower widths, the stacking pattern works well. It is works intuitively even on very narrow devices, and can be combined with other solutions for wider screens.

AG Grid not only offers extensive functionality out of the box, but also provides detailed documentation with examples. The simple implementation and ease of use make it ideal for developers to improve the readability and usability of data tables across multiple screen sizes, with minimal time investment. It is suitable for small to medium-sized data tables as well as for large data tables. The Community Edition offers a large set of basic features under an MIT license. Similar tools often require more effort to achieve the same functionality compared to this tool. The ease of implementation, extensive documentation, and functionality provided, make it the library of choice.

Bibliography

AG Grid [2024a]. *AG Grid: The Best JavaScript Grid in the World*. 6th Dec 2024. https://ag-grid.com/ (cited on page 13).

AG Grid [2024b]. *Community vs. Enterprise Licencing*. 6th Dec 2024. https://ag-grid.com/javascript-data-grid/licensing/ (cited on page 13).

Handsoncode [2024]. *Handsontable: JavaScript Data Grid with Spreadsheet UI*. 6th Dec 2024. https://handsontable.com/ (cited on page 13).

Marcotte, Ethan [2014]. *Responsive Web Design*. 2nd Edition. A Book Apart, 2nd Dec 2014. 153 pages. ISBN 1937557189. http://abookapart.com/products/responsive-web-design (cited on page 1).

Mozilla [2024a]. *HTML: HyperText Markup Language*. 25th Sep 2024. https://developer.mozilla.org/en-US/docs/Web/HTML (cited on page 3).

Mozilla [2024b]. *Mozilla HTML Tables*. 25th Sep 2024. https://developer.mozilla.org/en-US/docs/Learn/HTML/Tables/Basics (cited on page 1).

SpryMedia [2024]. *DataTables*. 6th Dec 2024. https://datatables.net/ (cited on page 14).

TanStack [2024]. *TanStack Table*. 6th Dec 2024. https://tanstack.com/table (cited on page 13).

Usablica [2024]. *Grid.js Advanced Table Plugin*. 6th Dec 2024. https://gridjs.io/ (cited on page 16).

W3Schools [2024]. *HTML Tables*. 3rd Dec 2024. https://w3schools.com/html/html_tables.asp (cited on page 3).