RespTable

A Lightweight, Responsive HTML Table Library

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

05 Feb 2025

Abstract

This report describes RespTable, an open-source, lightweight, responsive table pattern library. It is designed to enhance the usability of tables seamlessly across various screen sizes and devices. RespTable is highly flexible and provides an easy way for developers to implement patterns in their tables. In addition, it is very easy to expand and enhance. This report introduces RespTable, provides a concise overview of HTML tables and the principles of responsive design. It discusses the responsive table patterns and best-practice table patterns implemented in RespTable and explains the library structure along with the development workflow for incorporating new features.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

These days, almost all web sites and web applications are designed to be resposnive, so that they can be used and viewed on a variety of devices and screen widths [Marcotte 2014]. Web developers have to ensure that their content is usable and readable on every possible screen size. Among the content elements in HTML, data tables can be tricky to make responsive.

In previous work, Kassil et al. [2024] surveyed responsive table patterns and libraries. Larger responsive table libraries, like Handsontable [Handsoncode 2024], AG Grid [AG Grid 2024], and DataTables [SpryMedia 2024], provide extensive features and sophisticated functionality. Smaller libraries, like TanStack Table [TanStack 2024] and Grid.js [Usablica 2024], have fewer features and are simpler to use.

The aim of this project was to build an open-source, lightweight, responsive table library, positioned somewhere between the larger and smaller libraries mentioned above and self-contained with no other dependencies. The library is called RespTable and is available on GitHub [Kassil et al. 2025].

Chapter 2

HTML Tables and Responsive Design

The essential framework for web content is HTML (HyperText Markup Language). It defines the structure and semantics of web content. CSS handles the visual display, and JavaScript runs interactivity and functionality. These three components will be used in our project to ensure responsiveness for HTML tables on every device.

2.1 Traditional HTML Tables

HTML tables are basic components of web design that show data organized in rows and columns [W3Schools 2025; MDN 2025]. 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.

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.

2.2 Responsive Tables

To address these challenges for responsive design, various techniques can be used to make tables responsive and usable on different devices. With such patterns tables can be rearranged into new shapes, important data can be filtered, table content can be highlighted, or the user interaction with the table can be enhanced.

Some of the patterns implemented in RespTable include:

- *Stacking*: Converts table rows into vertically stacked blocks for narrow screens.
- *Squishing*: Fitting a table into a smaller space by reducing its size.
- *Sorting*: Rearrange rows by clicking on a column header to sort the values in ascending or descending order.
- *Filtering*: Display only rows matching a filter criteria.

Some of the patterns, such as Stacking and Squishing, can be implemented with CSS only. Some patterns, such as Filtering and Sorting, need JavaScript to work.

```
<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 Table Patterns

As explained in previous work [Kassil et al. 2024], table patterns can be grouped into *responsive* table patterns and *good practice* table patterns. Responsive table patterns allow HTML tables to adapt their layout and functionality based on the device, ensuring that they remain usable and functional whether viewed on a mobile device, PC, or TV. The RespTable library implements four key responsive patterns: Stacking, Squishing, Flipping, and Horizontal Scrolling.

3.1 Stacking

In the Stacking pattern, each table row is rearranged into a vertical stack, displayed sequentially. The initial table headers serve as labels next to the data for each corresponding row. This approach improves readability and usability on compact devices, allowing users to access all content in a row at once without scrolling. It is usually implemented solely with CSS, eliminating the need for additional JavaScript. This provides the benefit of straightforward implementation and ensures functionality even in environments where JavaScript is disabled. However, a downside of stacking is that comparing content across different rows becomes challenging, as the stacked format displays only one row at a time, making this approach less suitable where the primary goal is value comparison.

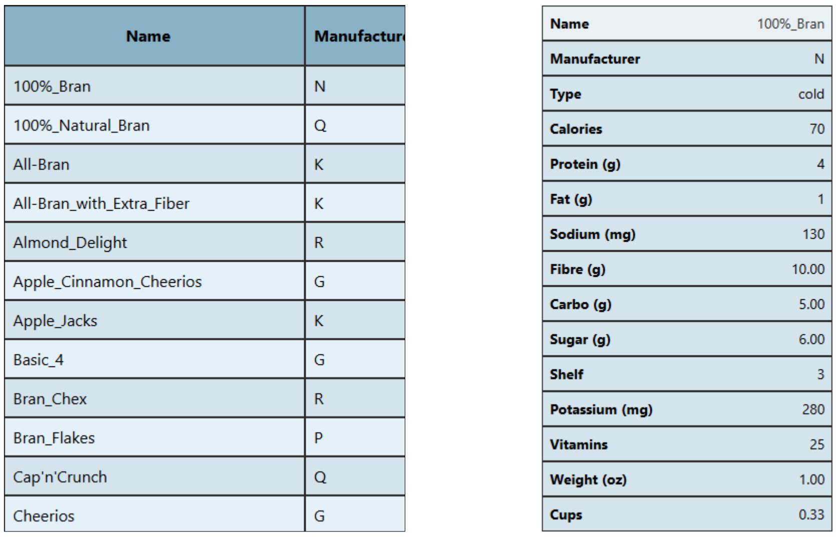
Figure 3.1 shows parts of a data table with 15 columns. On a narrow screen, most columns would be cut off if stacking were disabled. With stacking, users can view the content of one row at a time without truncated text.

3.2 Squishing

The easiest approach to make a table responsive is by reducing the width of its columns just enough to keep the content readable. In the Squishing pattern, the content of cells are truncated, compressed, or wrapped, so they take up less width. These techniques can be implemented purely in CSS with only a small amount of code. This simplicity is the primary benefit of the pattern, and there is little justification to omit it from any HTML table. Ideally, squishing allows all columns to remain visible, which is beneficial for tables where preserving the overall layout is important. However, in practice, this is often not feasible, requiring the consideration of additional patterns.

3.3 Flipping

The Flipping pattern transposes a HTML table by swapping rows and columns to better fit smaller or wider screens. When flipped, the table headers are displayed vertically, enhancing the visibility of content on mobile devices. This technique is particularly beneficial for tables with many columns but few rows. Without flipping, the table can become too wide, making it impossible to view on one screen. For tables



Name	Manufactur
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

(a) Stacking disabled.

Name	100%_Bran
Manufacturer	N
Type	cold
Calories	70
Protein (g)	4
Fat (g)	1
Sodium (mg)	130
Fibre (g)	10.00
Carbo (g)	5.00
Sugar (g)	6.00
Shelf	3
Potassium (mg)	280
Vitamins	25
Weight (oz)	1.00
Cups	0.33

(b) Stacking enabled.

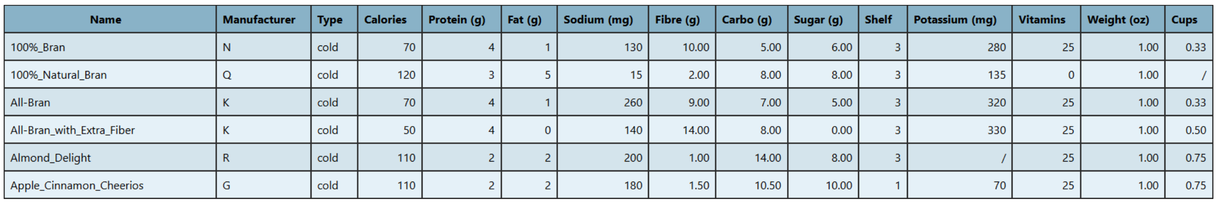
Figure 3.1: The same table with and without stacking, displayed on a narrow screen. [Screenshots taken by the author(s) of this report.]

with many rows and columns, flipping is not optimal and needs to be combined with other patterns, such as horizontal scrolling. Flipping uses CSS and JS to remove and reconstruct the entire table in its transposed form within the DOM. Activation can be automatic via container query-based screen size detection or manually through a user-initiated button.

Figure 3.2 shows the same table with 15 columns and one header row and six data rows in its basic and transposed forms. For mobile devices such as tablets, the flipped form of the table would improve readability. On other devices, especially with very wide screens, the basic form of the table is better.

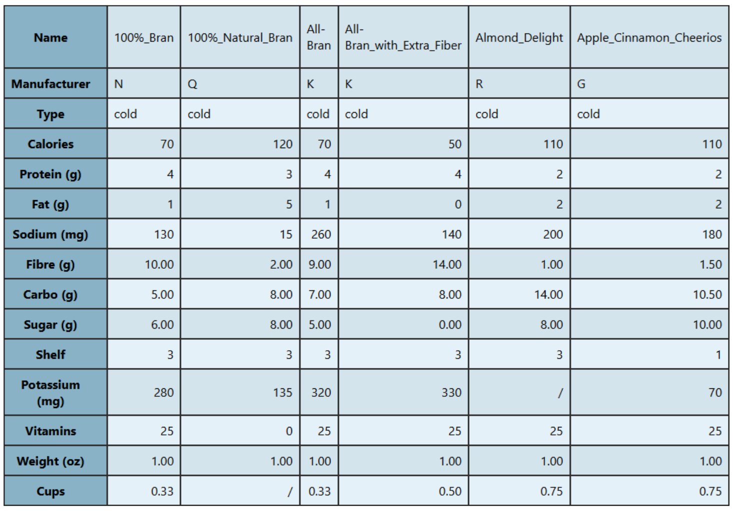
3.4 Horizontal Scrolling

The Horizontal scrolling pattern enables users to navigate tables horizontally, ensuring that all columns remain visible even when the table exceeds the width of the screen. This is achieved by adding a horizontal scroll bar, either within or next to the table. Unlike other responsive patterns, horizontal scrolling maintains the complete table structure, making it ideal for tables with numerous columns or complex data. Implementation is straightforward, often requiring just a few lines of CSS, similar to the Squishing pattern. However, one drawback is that horizontal scrolling can be cumbersome for touchscreen users, as interacting with scroll bars on such devices may lead to a less intuitive experience. For this reason, alternative patterns may be more suitable for touch-based interfaces.



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	10.00	5.00	6.00	3	280	25	1.00	0.33
100%_Natural_Bran	Q	cold	120	3	5	15	2.00	8.00	8.00	3	135	0	1.00	/
All-Bran	K	cold	70	4	1	260	9.00	7.00	5.00	3	320	25	1.00	0.33
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14.00	8.00	0.00	3	330	25	1.00	0.50
Almond_Delight	R	cold	110	2	2	200	1.00	14.00	8.00	3	/	25	1.00	0.75
Apple_Cinnamon_Cheerios	G	cold	110	2	2	180	1.50	10.50	10.00	1	70	25	1.00	0.75

(a) Standard Table.



Name	100%_Bran	100%_Natural_Bran	All-Bran	All-Bran_with_Extra_Fiber	Almond_Delight	Apple_Cinnamon_Cheerios
Manufacturer	N	Q	K	K	R	G
Type	cold	cold	cold	cold	cold	cold
Calories	70	120	70	50	110	110
Protein (g)	4	3	4	4	2	2
Fat (g)	1	5	1	0	2	2
Sodium (mg)	130	15	260	140	200	180
Fibre (g)	10.00	2.00	9.00	14.00	1.00	1.50
Carbo (g)	5.00	8.00	7.00	8.00	14.00	10.50
Sugar (g)	6.00	8.00	5.00	0.00	8.00	10.00
Shelf	3	3	3	3	3	1
Potassium (mg)	280	135	320	330	/	70
Vitamins	25	0	25	25	25	25
Weight (oz)	1.00	1.00	1.00	1.00	1.00	1.00
Cups	0.33	/	0.33	0.50	0.75	0.75

(b) Flipped Table.

Figure 3.2: The same table in its basic and flipped forms. [Screenshots taken by the author(s) of this report.]

Chapter 4

Good Practice Table Patterns

As explained in previous work [Kassil et al. 2024], table patterns can be grouped into *responsive* table patterns and *good practice* table patterns. Good practice table patterns significantly enhance the usability and readability of HTML data tables regardless of the screen width and have been widely recognized and used extensively for years. The RespTable library implements five common good practice patterns: Sorting, Filtering, Pinnable Header Row, Zebra Stripes, and Easy Column Alignment.

4.1 Sorting

The Sorting pattern enables users to sort columns in ascending or descending order. It is activated by clicking on a column header, which toggles between the two sorting directions. Small arrows, located next to the text in the header cell, indicate if the column is sorted in ascending order (upward arrow) or descending order (downward arrow). The arrows are disabled by default and only appear on a click. Additionally, the arrows appearing do not affect the column width, which prevents a sudden change in the size of the table. The table in Figure 4.1, is sorted in ascending order of Calories, indicated by the upward arrow next to the Calories header.

4.2 Filtering

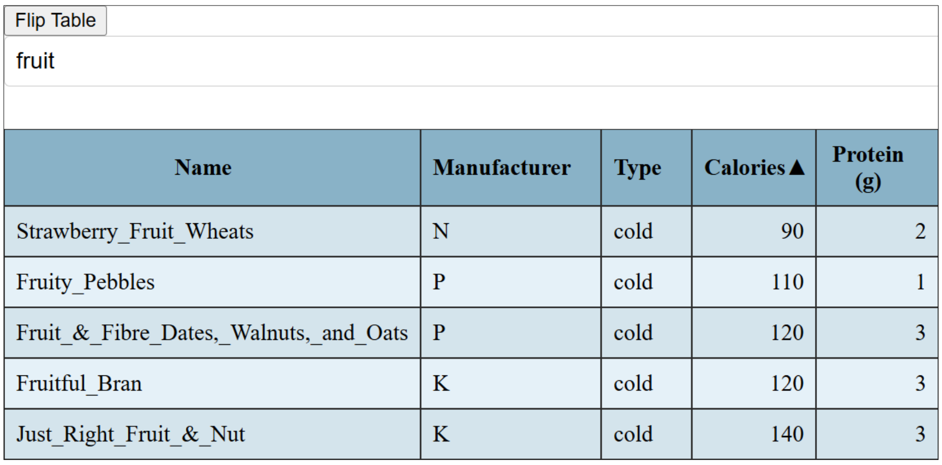
The Filtering pattern involves incorporating a filter bar into the table interface, allowing users to quickly filter data and locate specific data within large data tables. This feature is particularly useful for enhancing usability in tables with extensive information. Figure 4.1 shows a table with a specific filter query for fruits and the five results. The filter bar is located in the top left corner. Listing 4.1 shows the code to implement this feature. The code is part of the `CreateTable-function` where the main functionality of the library is implemented. When the filterable option is activated, the filter input is created, as well as an listener which monitors for text input and, upon detection, filters the data appropriately and reconstructs the table.

4.3 Pinnable Header Row

When working with large data tables, scrolling can often lead to a loss of context, particularly when header rows are no longer visible. To address this issue, the Pinnable Header Row pattern was implemented. This ensures that the table headers remain anchored to the top of the view port, staying visible at all times regardless of how far the user scrolls down. By keeping the headers in place, this pattern allows users to easily associate data with the corresponding columns, enhancing readability, and maintaining context throughout the table.

```
const container = document.getElementById('table-container');
container.innerHTML = '';

const filterInput = document.createElement('input');
if (isFilterable) {
filterInput.type = 'text';
filterInput.placeholder = 'Filter...';
filterInput.classList.add('table-filter');
container.appendChild(filterInput);
}

const tablePlaceholder = document.createElement('div');
tablePlaceholder.id = 'table-content';
container.appendChild(tablePlaceholder);

const fullTable = initializeTable(data);
tablePlaceholder.appendChild(fullTable);

if (isFilterable) {
filterInput.addEventListener('input', (event) => {
  const filterTerm = event.target.value.toLowerCase();
  const filteredData = data.filter((row) =>
    Object.values(row).some((value) =>
      value.toString().toLowerCase().includes(filterTerm)
    )
  );

  tablePlaceholder.innerHTML = '';

  if (filteredData.length === 0) {
    const noResultsMessage = document.createElement('div');
    noResultsMessage.classList.add('no-results');
    noResultsMessage.textContent = 'No results found.';
    tablePlaceholder.appendChild(noResultsMessage);
  } else {
    const filteredTable = initializeTable(filteredData);
    tablePlaceholder.appendChild(filteredTable);
    if (applyStacking) {
      ApplyStacking();
    } else if (a) {
      ApplyFlipping(filteredData, applyFlipping);
    }
  }
});
}
```

Listing 4.1: The source code to implement a filterable table.



Name	Manufacturer	Type	Calories▲	Protein (g)
Strawberry_Fruit_Wheats	N	cold	90	2
Fruity_Pebbles	P	cold	110	1
Fruit_&_Fibre_Dates,_Walnuts,_and_Oats	P	cold	120	3
Fruitful_Bran	K	cold	120	3
Just_Right_Fruit_&_Nut	K	cold	140	3

Figure 4.1: A table illustrating five table patterns simultaneously: Sorting, Filtering, Zebra Stripes, Easy Column Alignment, and Flipping (via a button). [Screenshot taken by the author(s) of this report.]

4.4 Zebra Stripes

The Zebra Stripes pattern is used to improve readability. As a table gets larger and has more rows, it can be quite challenging to match the information to the correct row. Zebra stripes solve this problem by coloring the rows in alternating colors. This pattern should probably be implemented in almost every table, due to its simplicity and large impact.

4.5 Easy Column Alignment

The Easy Column Alignment pattern ensures consistent and intuitive formatting, by automatically aligning text to the left and numbers to the right. This approach improves readability and usability, since left-aligned text enhances the flow of reading, while right-aligned numbers make it easier to compare numerical values. It was implemented via two CSS classes to specify either left-aligned text or right-aligned numbers. When a table is constructed from a CSV file, the correct classes are automatically assigned.

Chapter 5

RespTable Library

The RespTable library is built in JavaScript. It implements four responsive table patterns: Stacking, Squishing, Flipping, and Horizontal Scrolling. Additionally, it implements five good practice patterns: Sorting, Filtering, Pinnable Header Row, Zebra Stripes, and Easy Column Alignment.

A pattern can be activated by including the library files and either assigning a CSS class or calling a JavaScript function. RespTable also contains fucntionality to read a data table from a CSV file and automatically generate a corresponding HTML table. Besides documentation, three examples of using RespTable are provided: from a simple, CSS-only HTML example, through a CSS and JavaScript HTML example with multiple patterns, to a more extensive React example.

5.1 Library Structure

The structure of the RespTable library is hown in Listing 5.1. The `dist/` folder contains the built library files; it is created upon the first successful build. The `examples/` folder contains three examples of using RespTable.

To include new patterns or adapt existing ones, the developer only has to add or modify code in the file `Table.js`. Once the functionality is given, it is recommended to add it as a parameter to the `CreateTablefunction`. However, it is also possible to not do so, then it is simply required to declare the newly implemented feature as export and import it into the `index.js` file in the root folder of the library. To add new styles or adapt the current stylesheet, the developer has to work on the file `table.css`. If a newly implemented pattern needs a `.css` file, it is recommended to add them to the `styles/` folder, named after the specific pattern, and import it into file `Table.js`.

5.2 Building

After cloning the library, install its dependencies with the command:

```
npm install
```

To build the RespTable library, use the command:

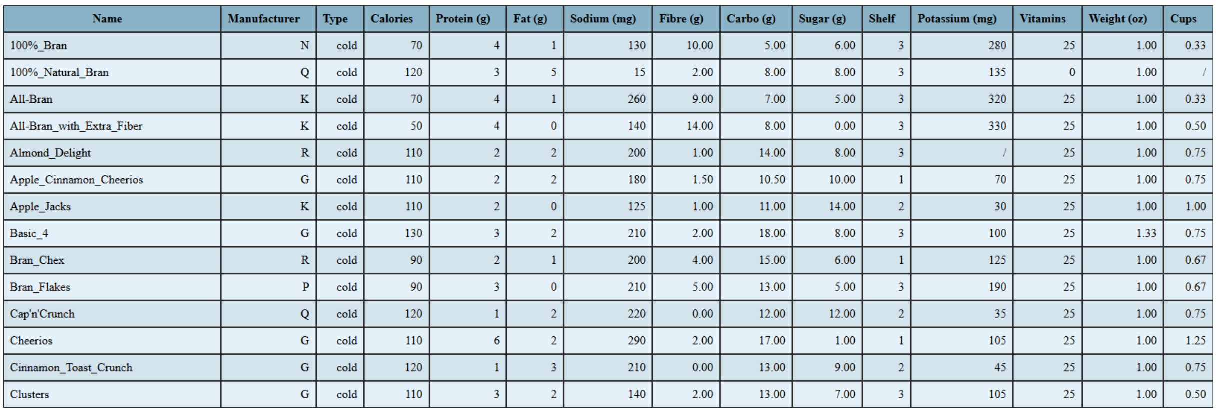
```
npx gulp build-lib
```

The three examples can be built and started with the commands:

```
npx gulp html
npx gulp html-csv
npx gulp react
```

```
RespTable/
├── dist/ (generated)
│   ├── cjs/
│   └── esm/
├── examples/
│   ├── html/
│   ├── html-with-csv/
│   └── react/
├── src/
│   ├── components/
│   │   └── Table.js
│   └── styles/
│       ├── squishing.css
│       ├── stacking.css
│       └── table.css
└──── index.js
```

Listing 5.1: The top-level file and directory structure of the RespTable project.



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	10.00	5.00	6.00	3	280	25	1.00	0.33
100%_Natural_Bran	Q	cold	120	3	5	15	2.00	8.00	8.00	3	135	0	1.00	/
All-Bran	K	cold	70	4	1	260	9.00	7.00	5.00	3	320	25	1.00	0.33
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14.00	8.00	0.00	3	330	25	1.00	0.50
Almond_Delight	R	cold	110	2	2	200	1.00	14.00	8.00	3	/	25	1.00	0.75
Apple_Cinnamon_Cheerios	G	cold	110	2	2	180	1.50	10.50	10.00	1	70	25	1.00	0.75
Apple_Jacks	K	cold	110	2	0	125	1.00	11.00	14.00	2	30	25	1.00	1.00
Basic_4	G	cold	130	3	2	210	2.00	18.00	8.00	3	100	25	1.33	0.75
Bran_Chex	R	cold	90	2	1	200	4.00	15.00	6.00	1	125	25	1.00	0.67
Bran_Flakes	P	cold	90	3	0	210	5.00	13.00	5.00	3	190	25	1.00	0.67
Cap'n'Crunch	Q	cold	120	1	2	220	0.00	12.00	12.00	2	35	25	1.00	0.75
Cheerios	G	cold	110	6	2	290	2.00	17.00	1.00	1	105	25	1.00	1.25
Cinnamon_Toast_Crunch	G	cold	120	1	3	210	0.00	13.00	9.00	2	45	25	1.00	0.75
Clusters	G	cold	110	3	2	140	2.00	13.00	7.00	3	105	25	1.00	0.50

Figure 5.1: Example of a responsive HTML table using CSS only. [Screenshot taken by the author(s) of this report.]

5.3 Simple HTML Table with CSS-Only Squishing and Stacking

The Squishing and Stacking patterns can be realised with CSS only, no JavaScript is required. Listing 5.2 shows the HTML code for a simple responsive table with CSS only; it has been shortened for brevity. The steps required are:

1. Include the RespTable CSS file in the `<head>` element:

   ```
   <link rel="stylesheet" href="respTable/RespTable.css">
   ```

2. Create a standard HTML `<table>` with hardcoded data.

3. Apply the RespTable CSS classes `responsive-table`, `squishing`, and `stacking` to the `<table>` element.

The result is shown in Figure 5.1.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title></title>

  <link rel="stylesheet" href="respTable/RespTable.css">
</head>

<body>
  <h1>Responsive Table with CSS Only</h1>

  <div id="table-container" class="table-container">
  <table class="responsive-table squishing stacking">
    <thead>
      <tr>
        <th>Name</th>
        <th>Manufacturer</th>
        <th>Type</th>
        ...
      </tr>
    </thead>
    <tbody>
      <tr>
        <td data-label="Name" class="text">100%_Bran</td>
        <td data-label="Manufacturer" class="text">N</td>
        <td data-label="Type" class="text">cold</td>
        ...
      </tr>
      ...
    </table>
  </div>
</body>
</html>
```

Listing 5.2: Creating a basic responsive HTML Table, using only CSS.

5.4 Responsive HTML Table from CSV with JavaScript

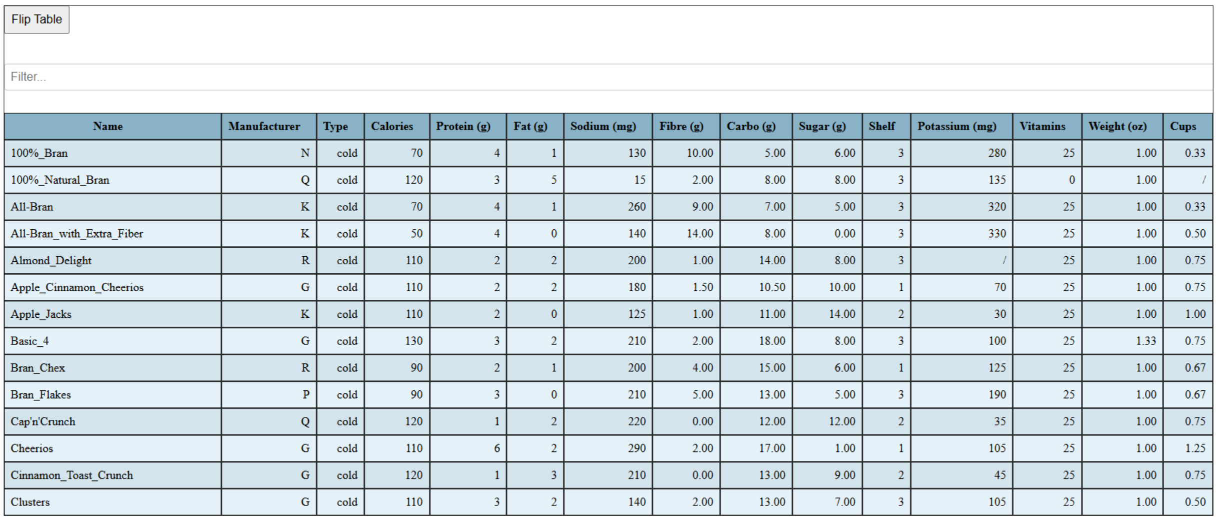
In this example, a table is dynamically loaded from a CSV file using JavaScript and then enhanced with the Flipping and Stacking patterns. Listing 5.3 shows the corresponding HTML code; it has been shortened for brevity. The steps required are:

1. Include the RespTable JavaScript and CSS files in the HTML.
2. Use JavaScript to load CSV data and generate the table directly.
3. Apply the desired patterns from JavaScript.

The result is shown in Figure 5.2.

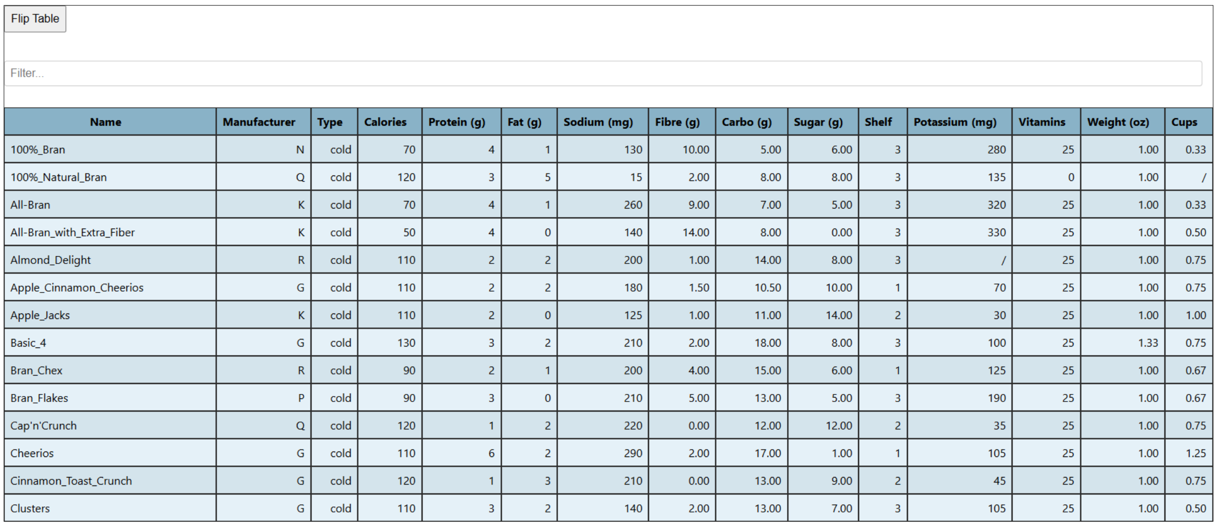
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0 />
  <title>Responsive Table from CSV with JavaScript</title>

  <link rel="stylesheet" href="respTable/RespTable.css">
</head>

<body>
<h1>Responsive Table from CSV with JavaScript</h1>

<!-- Flipping and Stacking Example -->
<button id="flip-button">Flip Table</button>
<div id="table-container"></div>

<script type="module">
  import { LoadCSV, CreateTable, ApplyFlipping, ApplyStacking }
    from './respTable/RespTable.js';

  let isFlipped = false;

  const initializeTable = async () => {
    const csvData = await LoadCSV('./cereals.csv');
    CreateTable(csvData);
    ApplyStacking();

    const flipButton = document.getElementById('flip-button');
    flipButton.addEventListener('click', () => {
      isFlipped = !isFlipped;
      ApplyFlipping(csvData, isFlipped);
    });
  };

  document.addEventListener('DOMContentLoaded', initializeTable);
</script>
</body>
</html>
```

Listing 5.3: Loading table data from a CSV file with JavaScript and enhancing with the Flipping and Stacking patterns.



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	10.00	5.00	6.00	3	280	25	1.00	0.33
100%_Natural_Bran	Q	cold	120	3	5	15	2.00	8.00	8.00	3	135	0	1.00	/
All-Bran	K	cold	70	4	1	260	9.00	7.00	5.00	3	320	25	1.00	0.33
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14.00	8.00	0.00	3	330	25	1.00	0.50
Almond_Delight	R	cold	110	2	2	200	1.00	14.00	8.00	3	/	25	1.00	0.75
Apple_Cinnamon_Cheerios	G	cold	110	2	2	180	1.50	10.50	10.00	1	70	25	1.00	0.75
Apple_Jacks	K	cold	110	2	0	125	1.00	11.00	14.00	2	30	25	1.00	1.00
Basic_4	G	cold	130	3	2	210	2.00	18.00	8.00	3	100	25	1.33	0.75
Bran_Chex	R	cold	90	2	1	200	4.00	15.00	6.00	1	125	25	1.00	0.67
Bran_Flakes	P	cold	90	3	0	210	5.00	13.00	5.00	3	190	25	1.00	0.67
Cap'n'Crunch	Q	cold	120	1	2	220	0.00	12.00	12.00	2	35	25	1.00	0.75
Cheerios	G	cold	110	6	2	290	2.00	17.00	1.00	1	105	25	1.00	1.25
Cinnamon_Toast_Crunch	G	cold	120	1	3	210	0.00	13.00	9.00	2	45	25	1.00	0.75
Clusters	G	cold	110	3	2	140	2.00	13.00	7.00	3	105	25	1.00	0.50

Figure 5.2: Example of a responsive table populated from a CSV file. [Screenshot taken by the author(s) of this report.]



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	10.00	5.00	6.00	3	280	25	1.00	0.33
100%_Natural_Bran	Q	cold	120	3	5	15	2.00	8.00	8.00	3	135	0	1.00	/
All-Bran	K	cold	70	4	1	260	9.00	7.00	5.00	3	320	25	1.00	0.33
All-Bran_with_Extra_Fiber	K	cold	50	4	0	140	14.00	8.00	0.00	3	330	25	1.00	0.50
Almond_Delight	R	cold	110	2	2	200	1.00	14.00	8.00	3	/	25	1.00	0.75
Apple_Cinnamon_Cheerios	G	cold	110	2	2	180	1.50	10.50	10.00	1	70	25	1.00	0.75
Apple_Jacks	K	cold	110	2	0	125	1.00	11.00	14.00	2	30	25	1.00	1.00
Basic_4	G	cold	130	3	2	210	2.00	18.00	8.00	3	100	25	1.33	0.75
Bran_Chex	R	cold	90	2	1	200	4.00	15.00	6.00	1	125	25	1.00	0.67
Bran_Flakes	P	cold	90	3	0	210	5.00	13.00	5.00	3	190	25	1.00	0.67
Cap'n'Crunch	Q	cold	120	1	2	220	0.00	12.00	12.00	2	35	25	1.00	0.75
Cheerios	G	cold	110	6	2	290	2.00	17.00	1.00	1	105	25	1.00	1.25
Cinnamon_Toast_Crunch	G	cold	120	1	3	210	0.00	13.00	9.00	2	45	25	1.00	0.75
Clusters	G	cold	110	3	2	140	2.00	13.00	7.00	3	105	25	1.00	0.50

Figure 5.3: Example of a React integration with RespTable. [Screenshot taken by the author(s) of this report.]

5.5 React Integration with RespTable

This example shows how to use RespTable within a React project created with `create-react-app`. Listing 5.4 shows the corresponding code; it has been shortened for brevity. The steps required are:

1. Import the table component into a React file, with Flipping and CSV loading.
2. Add `<App />` to the main component to render it.

The result is shown in Figure 5.3.

```

import React, { useState, useEffect } from 'react';
import {
  LoadCSV,
  CreateTable,
  ApplyFlipping,
} from '../src/respTable/RespTable.js';
import './respTable/RespTable.css';

const App = () => {
  const [isFlipped, setIsFlipped] = useState(false);
  const [csvData, setCsvData] = useState([]);

  useEffect(() => {
    const initializeTable = async () => {
        const data = await LoadCSV(`${process.env.PUBLIC_URL}/cereals.csv`);
        setCsvData(data);
        CreateTable(data, true, true);
    };

    initializeTable();
  }, []);

  const handleFlip = () => {
    setIsFlipped((prev) => !prev);
    ApplyFlipping(csvData, !isFlipped);
  };

  return (
    <div>
      <h1>RespTable: Showcase</h1>
      <button onClick={handleFlip}>Flip Table</button> {}
      <div id="table-container"></div>
    </div>
  );
};

export default App;
```

Listing 5.4: Importing RespTable to use within React. CSV loading and Flipping are used.

Chapter 6

Concluding Remarks

RespTable is a lightweight, open-source responsive table library with a compact package size of 24 kB. It includes basic patterns that are only dependent on plain HTML and CSS, and more advanced patterns which are also dependent on JavaScript. The library and three example usages available on GitHub [Kassil et al. 2025].

RespTable could be further improved and extended. Additional responsive patterns, such as pinnable columns for easier navigation in wide tables, pagination for managing large datasets, and column/row toggle for customizable data views, could be implemented to address a broader range of use cases. Furthermore, accessibility features such as screen reader compatibility should be added to ensure that the library meets modern web accessibility standards and caters to all users. Extensive compatibility testing could be performed with Rslidy, a responsive HTML slide deck tool [Rslidy 2025], to ensure seamless integration and optimal performance when used together. These enhancements will establish the library as a versatile solution for responsive table design.

Bibliography

AG Grid [2024]. *AG Grid: The Best JavaScript Grid in the World*. 06 Dec 2024. https://ag-grid.com/ (cited on page 1).

Handsoncode [2024]. *Handsontable: JavaScript Data Grid with Spreadsheet UI*. 06 Dec 2024. https://handsontable.com/ (cited on page 1).

Kassil, Alexander, Daniel Hevesy-Szettyan, Dominik Bauer and Miloš Globočki [2024]. *Responsive Tables*. 706.041 Information Architecture and Web Usability WS 2024 Survey Paper. Graz University of Technology, 13 Dec 2024. https://courses.isds.tugraz.at/iaweb/surveys/ws2024/iaweb-ws2024-g3-survey-responsive-tables.pdf (cited on pages 1, 5, 9).

Kassil, Alexander, Daniel Hevesy-Szettyan, Dominik Bauer and Miloš Globočki [2025]. *RespTable*. 05 Feb 2025. https://github.com/milos-globocki/RespTable (cited on pages 1, 19).

Marcotte, Ethan [2014]. *Responsive Web Design*. 2nd Edition. A Book Apart, 02 Dec 2014. 153 pages. ISBN 1937557189. https://ethanmarcotte.com/books/responsive-web-design/ (cited on page 1).

MDN [2025]. *The Table Element*. 05 Feb 2025. https://developer.mozilla.org/docs/Web/HTML/Element/table (cited on page 3).

Rslidy [2025]. *Rslidy*. 05 Feb 2025. https://github.com/tugraz-isds/rslidy (cited on page 19).

SpryMedia [2024]. *DataTables*. 06 Dec 2024. https://datatables.net/ (cited on page 1).

TanStack [2024]. *TanStack Table*. 06 Dec 2024. https://tanstack.com/table (cited on page 1).

Usablica [2024]. *Grid.js Advanced Table Plugin*. 06 Dec 2024. https://gridjs.io/ (cited on page 1).

W3Schools [2025]. *HTML Tables*. 05 Feb 2025. https://w3schools.com/html/html_tables.asp (cited on page 3).