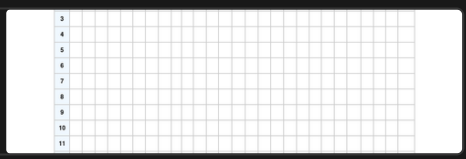
Excel Sheet - Clone

Intended for Beginners

CodeSandbox Link :

Excel-Clone

https://codesandbox.io/p/sandbox/excel-clone-2pr5g3



Basic HTML Structure

First, you need a basic HTML structure to hold the table.

```
<!DOCTYPE html> <html> <head> <title>JavaScript Sandbox</title> <meta cha
rset="UTF-8" /> </head> <body> <div id="spreadsheet"> <table id="excel-ta
ble"> <thead> <tr> <!-- Column Headers --> </tr> </thead> <tbody> <!-- Ap
pend Child Nodes which represent -- Rows having Cells --> </tbody> </tabl
e> </div> <script src="./index.js" type="module"></script> </body> </html
>
```

CSS for Styling

Add CSS to style the table to look like an Excel sheet.

```
body { font-family: sans-serif; display: flex; } #spreadsheet { width: 8
0%; margin: 0 auto; overflow: auto; } table { width: 100%; border-collaps
e: collapse; } th { border: 2px solid #ccc; padding: 10px; text-align: ce
nter; background-color: aliceblue; } td { border: 2px solid #ccc; paddin
g: 10px; text-align: right; }
```

Notice that Excel generates column headers using single letters (A, B, C, etc.), and also handles cases where you have more than 26 columns, which would require labels like AA, AB, and so on.

To handle this, we create a function to generate column labels beyond 'Z'.

JavaScript Code

```
import "./styles.css"; /** * Problem Statement * Create Excel/Google Shee
ts Table Layout/Cells * Front-End * Optional to add other properties such
as select, editable formulas, etc. * * **/ const table = document.getElem
entById("excel-table").querySelector("tbody"); const columnHeaders = docu
ment .getElementById("excel-table") .querySelector("thead tr"); const num
Rows = 30; const numCols = 3000; // Created 1st Column Header; which shou
ld be empty as its filled with Row Headers const emptyTh = document.creat
eElement("th"); emptyTh.innerText = " "; columnHeaders.appendChild(emptyT
h); // Create Remaining Column Headers for (let col = 1; col <= numCols;
col++) { const th = document.createElement("th"); th.innerText = getColum
nName(col - 1); columnHeaders.appendChild(th); } // Function to generate
headers from Column Number Received function getColumnName(index) { let c
olumnName = ""; // "" while (index >= 0) { columnName = String.fromCharCo
de((index % 26) + 65) + columnName; index = Math.floor(index / 26) - 1; }
return columnName; } // Create Rows and Cells for (let row = 1; row <= nu
mRows; row++) { const tr = document.createElement("tr"); // Row headers /
sl.no. const th = document.createElement("th"); th.innerText = row; tr.ap
pendChild(th); // Each Row // Input Cells for (let col = 0; col < numCol
s; col++) { const td = document.createElement("td"); // Cell // const inp
ut = document.createElement("input"); // input Field wrapped inside cell
// input.type = "text"; // td.appendChild(input); td.setAttribute("conten
tEditable", true); // Makes the whole cell editable and behave like actua
l sheets tr.appendChild(td); } table.appendChild(tr); // Adding the row t
o our table } // Event handling to enable Cell Input table.addEventListen
er("input", (event) => { // Handle Cell Input - Try out yourselves! });
```

How It Works

- `getColumnName(index)`: This function converts a zero-based index to a column label like "A", "B", ..., "Z", "AA", "AB", ..., "AZ", "BA", and so forth.
  - The function uses a loop to repeatedly divide the index by 26, appending the appropriate character to `columnName` for each iteration.

Example

If you set `numCols = 30`, the first few column headers will be `A, B, C, ..., Z, AA, AB, AC, ..., AD`.

This approach ensures that your spreadsheet can have as many columns as needed, and the column names will scale correctly beyond "Z".

The code will also handle column labels correctly even after "ZZ". It will generate labels like "AAA", "AAB", etc., as needed.

Detailed Explanation:

The function `getColumnName(index)` works as follows:

1. **Index Calculation**:
   - The index is treated as a base-26 number (with letters A-Z corresponding to 0-25).
   - The loop continues until the index becomes negative, which means all digits (letters) have been processed.
2. **Character Mapping**:
   - `String.fromCharCode((index % 26) + 65)` converts the remainder (from `index % 26`) to the corresponding letter.
   - `index = Math.floor(index / 26) - 1` reduces the index for the next iteration.

Example Beyond "ZZ":

For instance, if the index reaches 702 (corresponding to "ZZ"):

- 703 (Index 702):
  - First loop: `index % 26 = 0`, so 'A' is added, and `index` is updated to 26.
  - Second loop: `index % 26 = 0`, so another 'A' is added, and `index` is updated to 0.
  - Third loop: `index = -1` ends the loop.
  - Result: "AAA"
- 704 (Index 703):
  - First loop: `index % 26 = 1`, so 'B' is added, and `index` is updated to 26.
  - Second loop: `index % 26 = 0`, so 'A' is added, and `index` is updated to 0.
  - Third loop: `index = -1` ends the loop.
  - Result: "AAB"