The background is a green grid with various hand-drawn business diagrams and sketches. These include flowcharts with boxes labeled 'PLAN', 'PROGRESS', 'MANAGEMENT', and 'NEXT'. There are also line graphs, bar charts, pie charts, and circular diagrams. Some sketches include human figures representing people in an organization. The word 'IDEA' is written near a lightbulb icon, and 'MAX' appears in several places. The word 'OPTIONS' is written in an oval. The overall theme is business strategy and management.

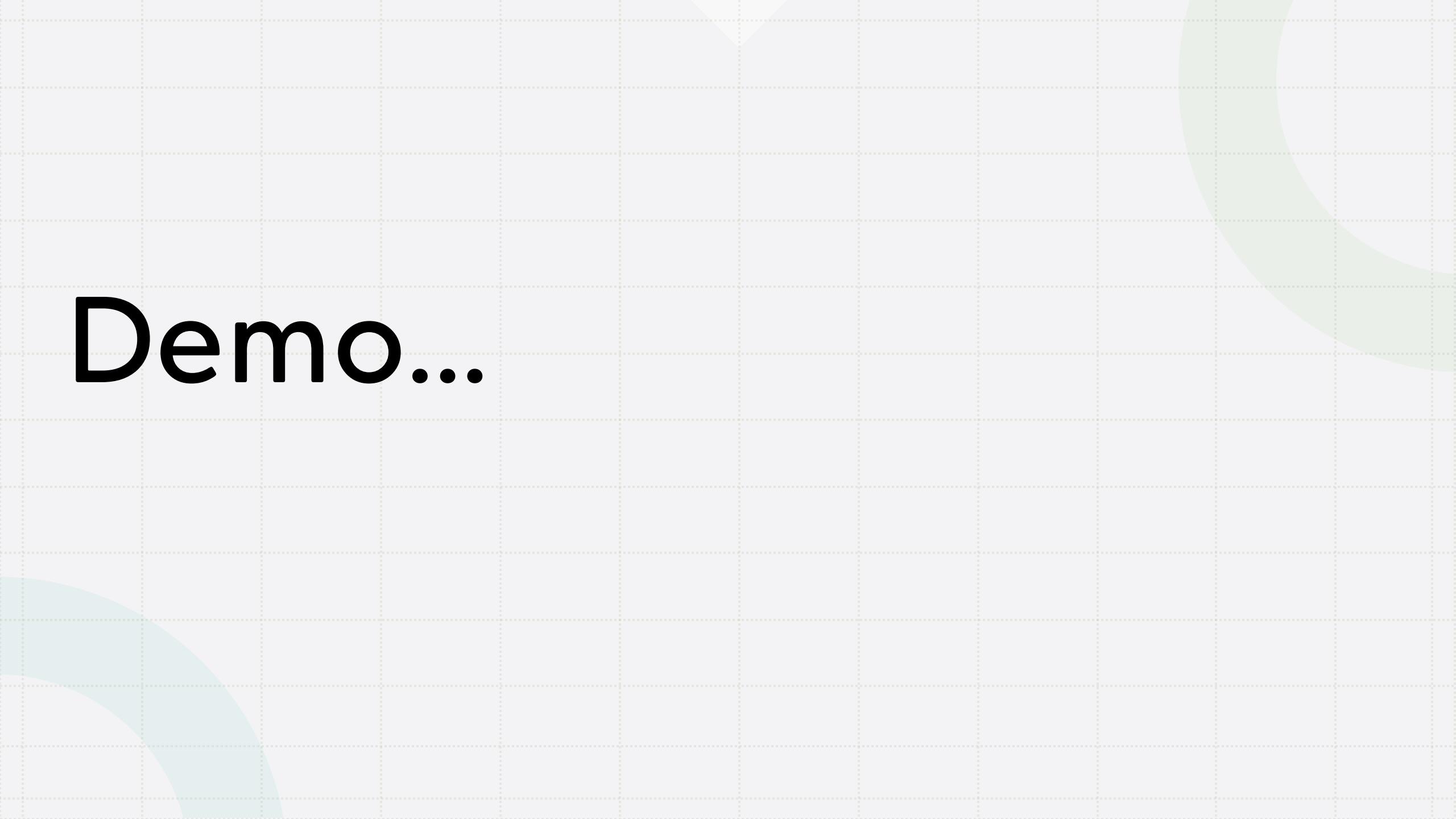
Our Spreadsheet Application

Group 507

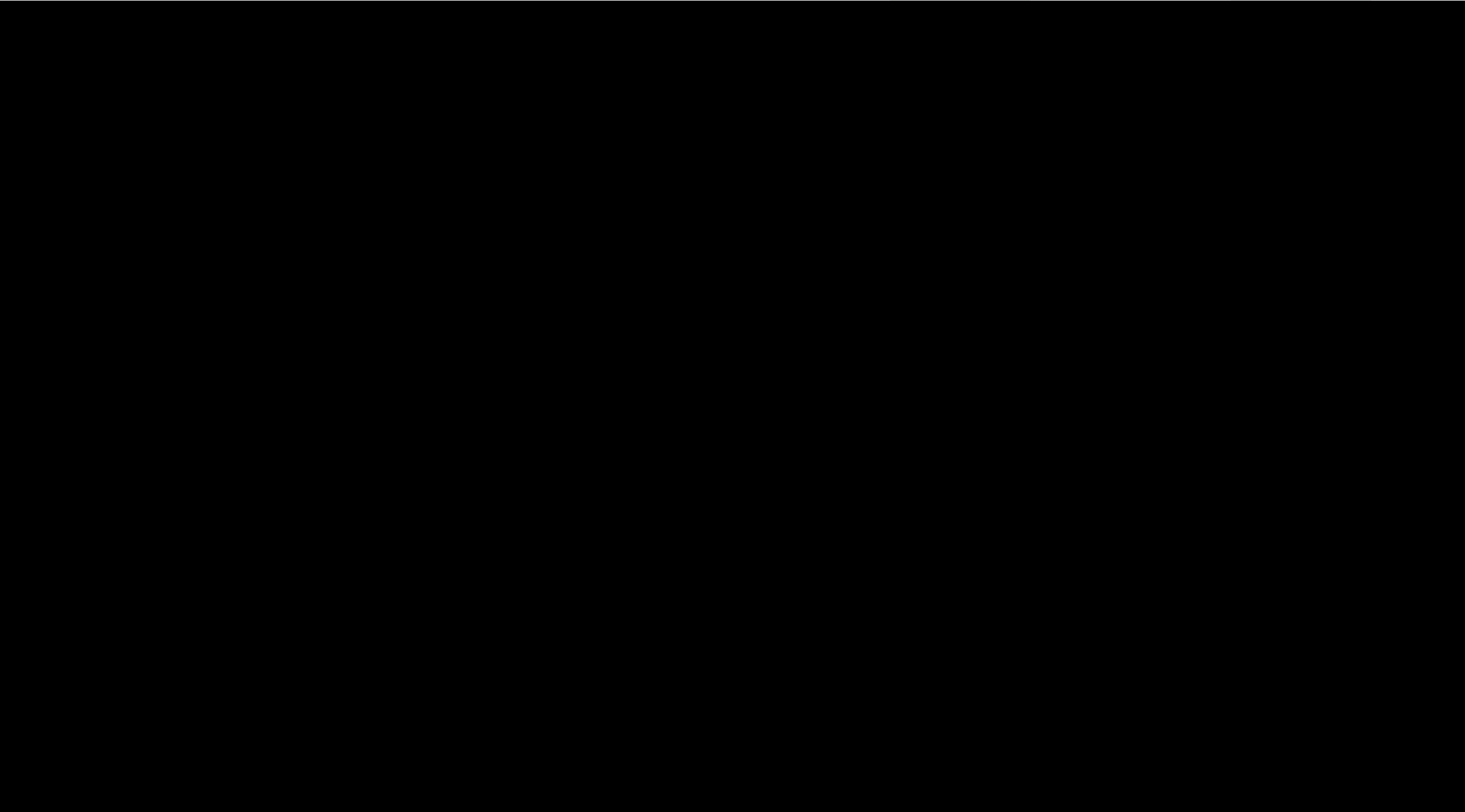


Team members

- Amaiya Brickhouse
- Bianca Anne-Marie Ciorobea
- Suhani Singhvi



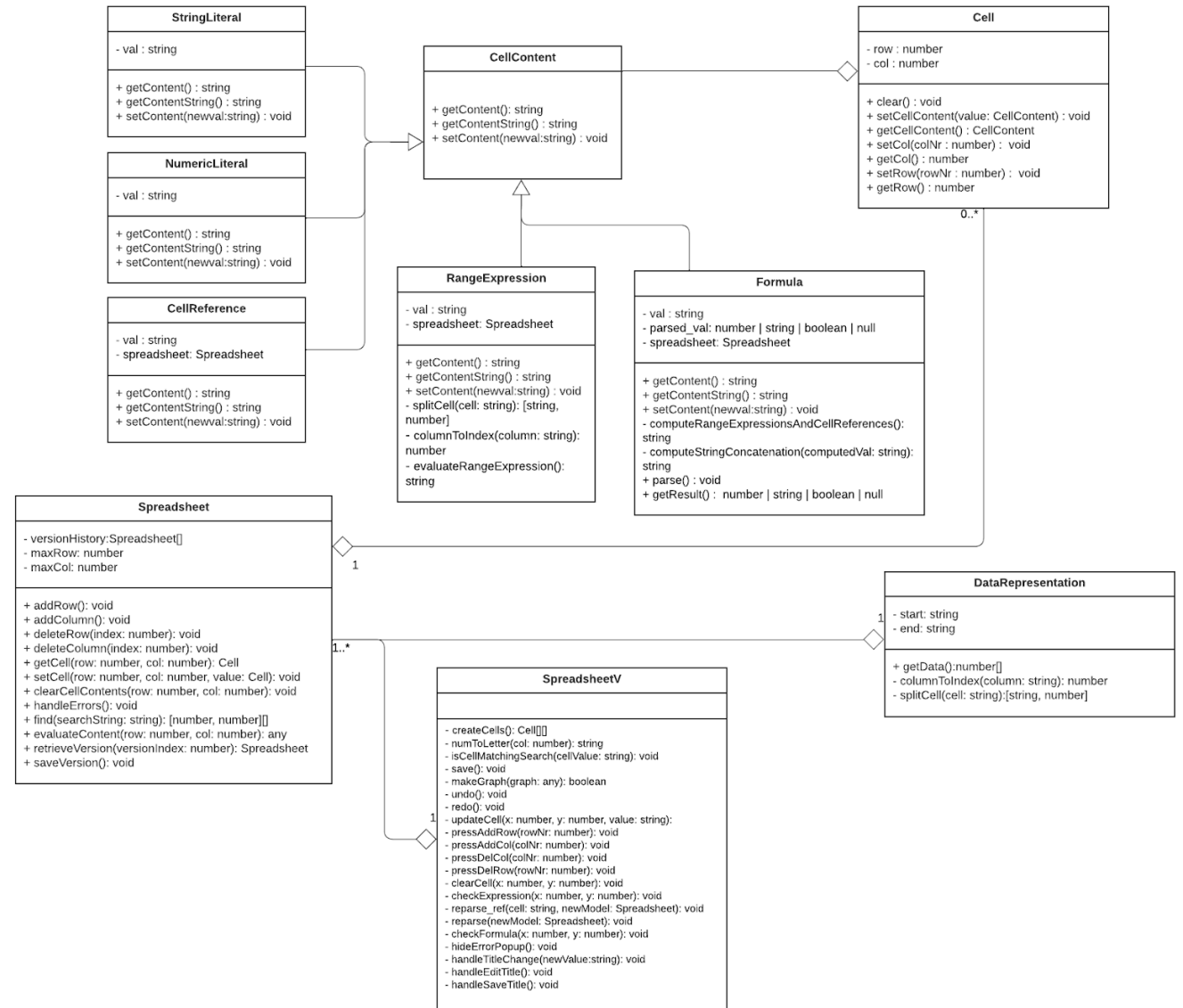
Demo...



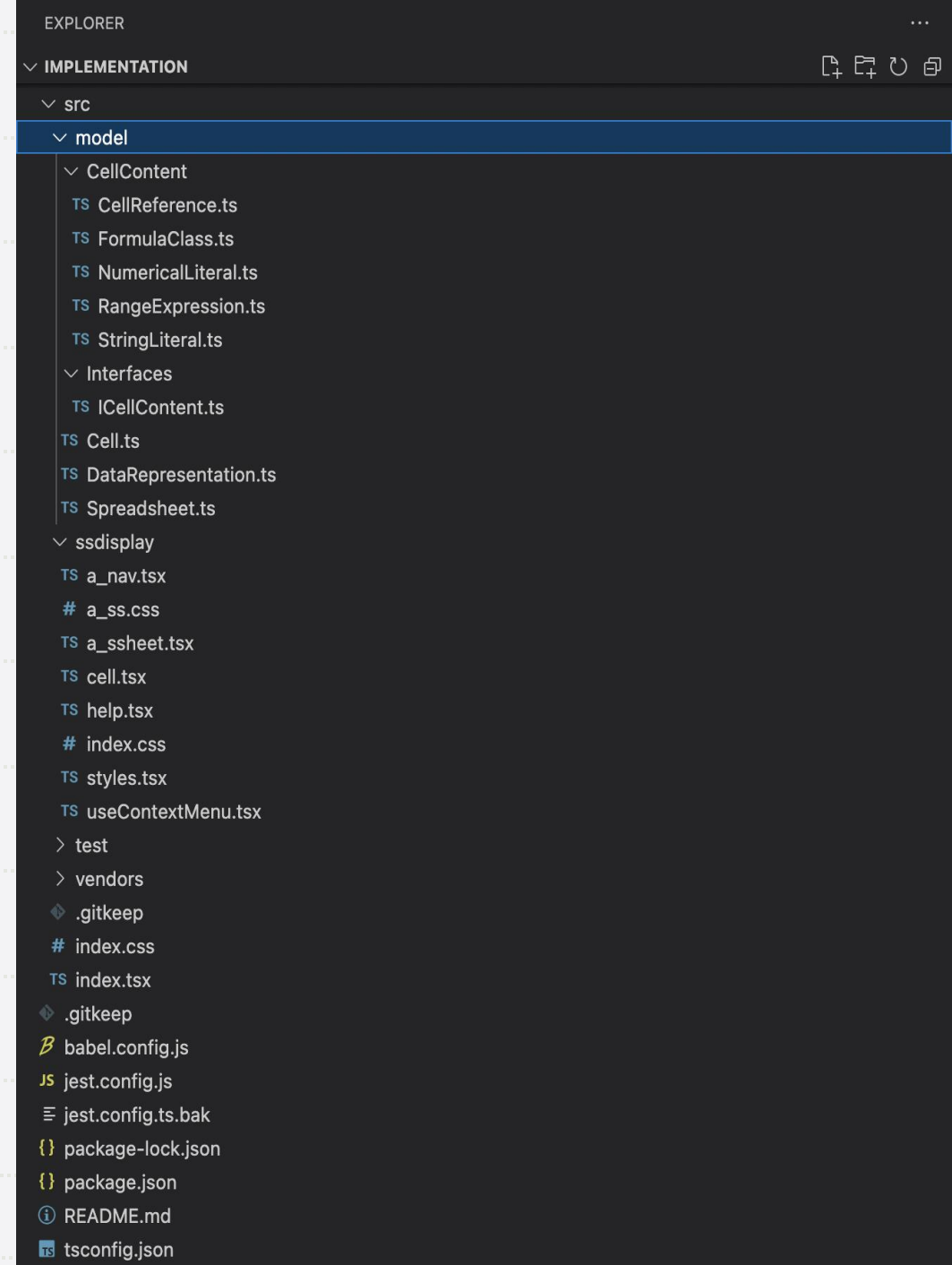


Overall Architecture of our system

UML Diagram



Structure of our code



1. Model Layer

Cell Management (Cell.ts):

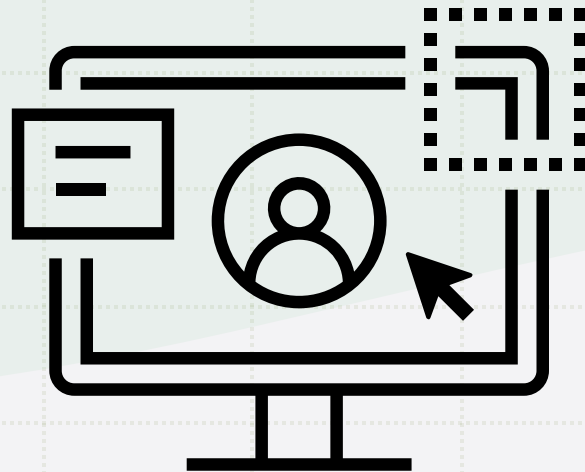
- ❑ defines the individual cells, including their content and position (row and column).

Data Representation (DataRepresentation.ts):

- ❑ handles the organization of cell data, particularly for graphical representations (charts).

Spreadsheet Core (Spreadsheet.ts):

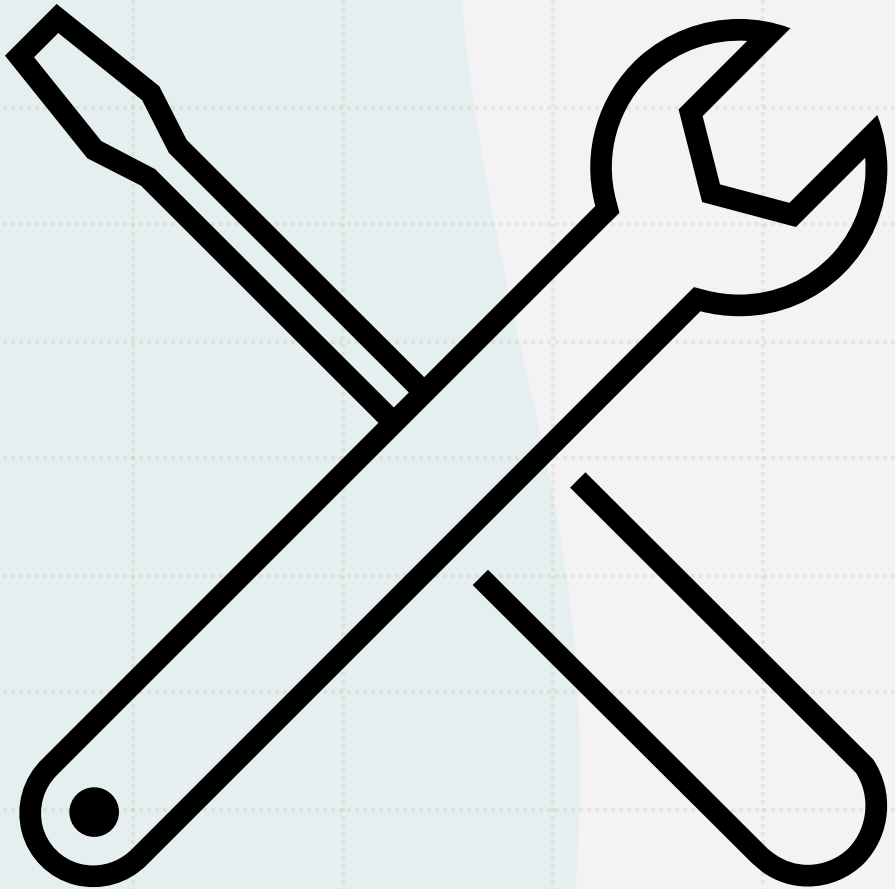
- ❑ central to the system
- ❑ it manages the 2D array of cells and handles operations including adding, deleting, and modifying cells.



2. Interface and Interaction Layer

Display (ssdisplay directory):

- ❑ handles the presentation of the spreadsheet to the user
- ❑ includes updating cells according to user input, recomputing formulas, highlighting searched values, the display of any errors or messages, and many other frontend features



3. Tools and Testing

Configuration Files

- ❑ Includes **webpack.config.js**, **babel.config.js**, **tsconfig.json**

Testing ('test' Directory)

- ❑ Indicates the presence of a testing framework.
- ❑ Ensures correctness of spreadsheet functionalities

4. Abstracted Components

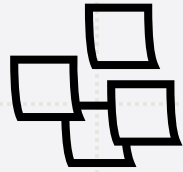
Interfaces (Interfaces in 'model')

- ❑ Enhances modularity and ease of maintenance.

Range Expressions, Cell References and Content Types (concrete classes that implement CellContent)

- ❑ Used for handling complex functionalities like formulas, cell references, range expressions, etc.

5. Help and Documentation



Documentation (README.md file)

- ☐ Includes all the instructions necessary to use to features of the application

Interesting features

```
function renderTable() {
  const table = document.querySelector('table');
  table.innerHTML = `
    <table class="table table-striped table-bordered function-keys py-1">
      <thead>
        <th class="text-center py-2 border"></th>
        {model.cells[0].map((x, i) => (
          <th class="text-center border">{numToLetter(i)}</th>
        ))}
      </thead>
      <tbody>
        {model.cells.map((val, x) => {
          return (
            <tr>
              <th class="text-center pb-0">{x + 1}</th>
              {val.map((cell, y) => {
                try {
                  const cellValue = cell.getCellContent()?.getContentString();
                  const cellString = cell.getCellContent()?.getContent();
                  const highlightStyle = isCellMatchingSearch(cellString as string)
                    ? { backgroundColor: "lightblue" } // apply highlighting style
                    : {};
                } catch {
                  // ignore error
                }
                return <td class="text-center">{cellValue}
              ))}
            </td>
          )
        })}
      </tbody>
    </table>
  `;
  table.appendChild(document.createTextNode(''));
}
```

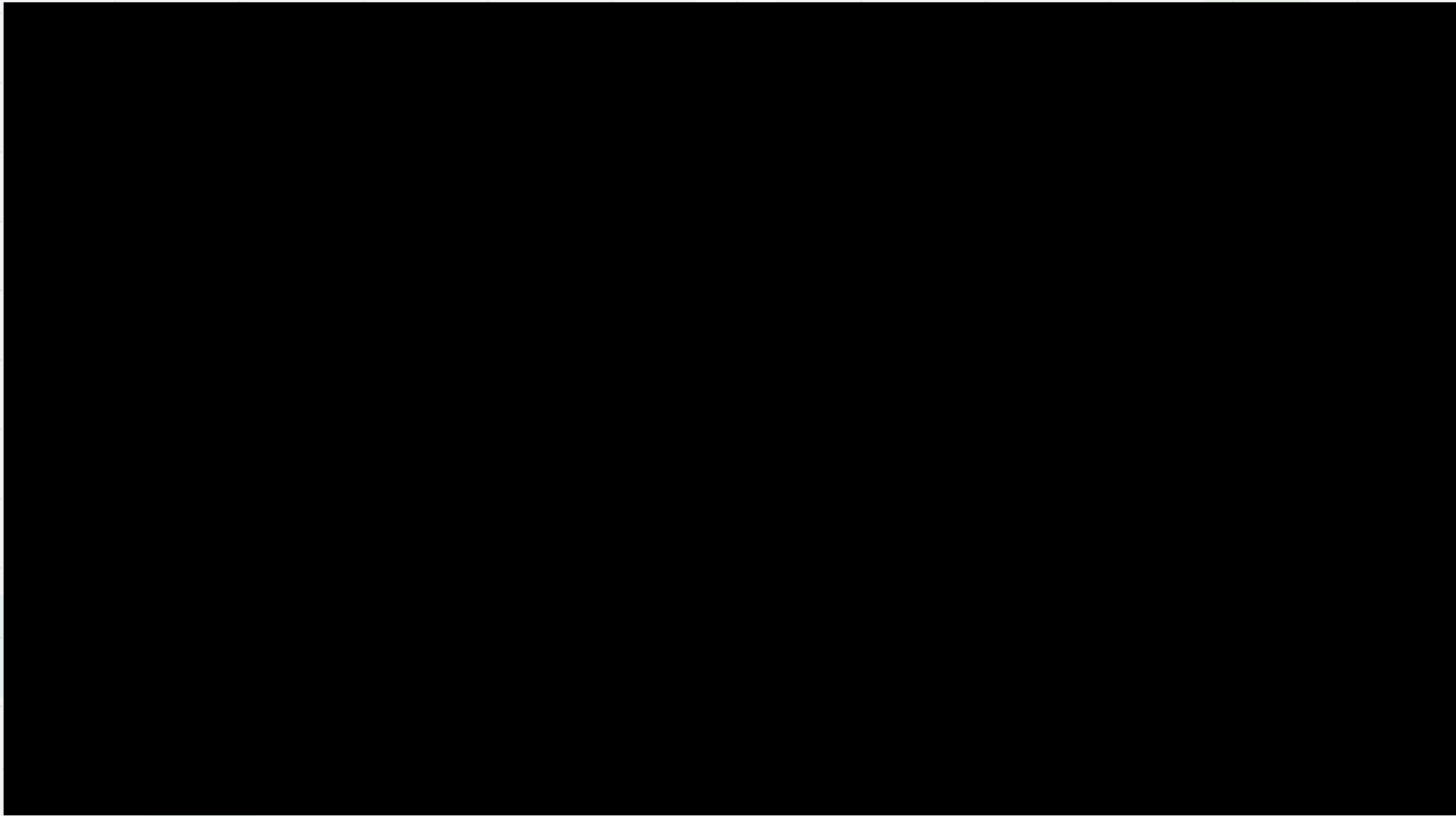
```
// function to undo latest change
function undo() {
  if (historyActiveIndex > 0) {
    // calculate the new index for the previous history element
    const newIndex = historyActiveIndex - 1;

    // reassess the spreadsheet using one of the versions within the history array
    // (the one at the computed index)
    reassess(new Spreadsheet(history[newIndex].cells, 300, 52));

    // update the history active index
    setHistoryActiveIndex(newIndex);
  }
}

// function to redo latest change
function redo() {
  if (historyActiveIndex < history.length - 1) {
    // calculate the new index for the next history element
    const newIndex = historyActiveIndex + 1;

    // reassess the spreadsheet using one of the versions within the history array
    reassess(new Spreadsheet(history[newIndex].cells, 300, 52));
  }
}
```



What went well?

- Frequent meetings
- Good communication
- Pair programming made completing tasks easier
- Helpful documentation and clean code
- Adequate version control
- Code reviews among team members

Challenges

Coordination in earlier parts of the project, caused by conflicting schedules

Misattribution of story points (underestimating and overestimating)

Issues with Github

Merge conflicts leading to loss of functionality in code sometimes

Lessons

- The overall design of the project will most probably not coincide with the initial one
- Starting early in order to prepare for possible future complications
- Communicate with teammates often to stay on the same page

Search for ...

[illegible]