Spreadsheet Project CS4530

Phase B

Group 507

Amaiya Brickhouse

Bianca Anne-Marie Ciorobea

James Peterson

Suhani Singhvi

User Stories

1.

TITLE: Allow Numeric Constants in Spreadsheet Cells

PRIORITY: high

ESTIMATE: 1 story point

AS A User

I WANT TO enter numeric constants in the spreadsheet cells

SO THAT I can insert numerical data easily when necessary.

ACCEPTANCE CRITERIA:

- Numeric constants can be entered in cells within the spreadsheet.
- The constants that were entered are accurately displayed in the selected cells.

2.

TITLE: Allow String Constants in Cells

PRIORITY: high

ESTIMATE: 1 story point

AS A User

I WANT TO enter string constants in the cells

SO THAT I can input text data easily when needed.

ACCEPTANCE CRITERIA:

- String constants can be entered in cells within the spreadsheet.
- The constants that were entered are accurately displayed in the selected cells.

3.

TITLE: Allow Cell References

PRIORITY: high
ESTIMATE: 2 story points

AS A User
I WANT TO use cell references to refer to values of other cells within the spreadsheet

SO THAT I can create relationships between certain cells.

ACCEPTANCE CRITERIA:

- Cell references can be used to represent a value within another cell, by writing REF(<cell>).
- Cell references get the correct value from the referenced cell and add it to the right location within the spreadsheet.
- If a row or column is inserted or deleted within the spreadsheet, the cell references update automatically with the new positions of the cells (the result of the cell references stay the same and the reference changes according to the new position of the old value).

4.

TITLE: Allow Range Expressions

PRIORITY: high
ESTIMATE: 2 story points

AS A User
I WANT TO use range expressions to calculate the SUM or AVERAGE of a range of cells

SO THAT I can easily compute the total sum or average of multiple cell values.

ACCEPTANCE CRITERIA:

- Range expressions for SUM and AVERAGE can be used in cells, by writing SUM(<range of cells>) or AVERAGE(<range of cells>).

- Range expressions within the spreadsheet correctly calculate the sum or average of the values within the desired ranges of cells.

- If a row or column is inserted or deleted within the spreadsheet, the range expressions update automatically with the new positions of the cells (the result of the range expressions stay the same and the range changes according to the new positions of the old values).

5.

TITLE: Allow Cells to Contain Formulas

PRIORITY: high
ESTIMATE: 3 story points

AS A User
I WANT TO use formulas in cells when needed
SO THAT I can create and evaluate expressions constructed using arithmetic operators, such as addition, subtraction, multiplication, etc., and be able to use the concatenation operator on string values.

ACCEPTANCE CRITERIA:

- Formulas with arithmetic operators can be used in cells.

- Formulas with arithmetic operators can include parentheses that control the order of the operations.

- The arithmetic operations within the formulas are correctly executed, and the spreadsheet cell will contain the final result.

- The arithmetic operations, within a formula that contains parentheses, are evaluated based on the order of the parentheses.

- Users can create formulas that represent string concatenation (using the + operator with strings).

- Formulas that use the concatenation operator on string values are correctly executed, and the spreadsheet cell will contain the resulting string.

6.

TITLE: Insert Rows and Columns

PRIORITY: high

ESTIMATE: 2 story points

AS A User

I WANT TO insert a new row or column

SO THAT I can extend the spreadsheet easily when needed.

ACCEPTANCE CRITERIA:

- A new row or column can be inserted anywhere within the spreadsheet, by right clicking on a specific cell and selecting the appropriate option.

- The data within the existing cells are appropriately moved to fit the addition.

- If a row is inserted above the current selected cell/row (e.g. if the user clicks on “Insert row above”), the currently selected row and every row below the new one will have its row index incremented by 1.

- If a row is inserted below the current selected cell/row (e.g. if the user clicks on “Insert row below”), every row below the new one will have its row index incremented by 1 with the new row taking on the position of “currentRowPos + 1” (where currentRowPos is the index of the current row).

- If a column is inserted to the right of the currently selected cell/column (e.g. if the user clicks on “Insert column right”), every column to the right of the current column will have its column index incremented by 1.

- If a column is inserted to the left of the currently selected cell/column (e.g. if the user clicks on “Insert column right”), the currently selected column and every column to the right of the current column will have its column index incremented by 1.

7.

TITLE: Delete Rows and Columns

PRIORITY: high

ESTIMATE: 3 story points

AS A User

I WANT TO delete a row or column from the spreadsheet

SO THAT I can easily remove any data that I find unnecessary.

ACCEPTANCE CRITERIA:

- A row or column from anywhere within the spreadsheet can be deleted, by right clicking on a specific cell and selecting the appropriate option (e.g. press on “Delete row” or “Delete column”).

- If a row is deleted, then the existing data below the removed row is shifted up.

- If a column is deleted, then the existing data to the right of the removed column is shifted to the left.

8.

TITLE: Clear Cell Contents

PRIORITY: high

ESTIMATE: 1 story point

AS A User

I WANT TO remove the contents of a cell

SO THAT I can easily reset data from the spreadsheet when needed.

ACCEPTANCE CRITERIA:

- The contents of any cell within the spreadsheet can be cleared, by right clicking on a cell and selecting the appropriate option (e.g click on “Clear cell”).

- If the cleared cell is used in a formula within another cell, its value will be automatically substituted for a default value (if it is a numeric operation, the value of the cell will become 0 in the context of the formula, and if it is a string concatenation, it will become “”).

- If the cleared cell is not used within a formula, then the value of the cell will be set to Null.

9.

TITLE: Display Current Cell Values

PRIORITY: high

ESTIMATE: 1 story point

AS A User

I WANT TO see the current value of each cell

SO THAT I can easily find data within the spreadsheet.

ACCEPTANCE CRITERIA:

- The current value of each cell is displayed in the user interface.
- The values that are displayed in the UI are updated as data changes.

10.

TITLE: Edit Formulas or Cell Contents

PRIORITY: high

ESTIMATE: 2 story points

AS A User

I WANT TO select a cell and view or edit its associated formula or content

SO THAT I can easily customize the operations and data within the spreadsheet.

ACCEPTANCE CRITERIA:

- Users can click on any cell, view the current formula associated with it and update it according to their preference.
- The value of the new formula is displayed in the cell after the user updates it.
- Users can modify the values, strings, cell references and range expressions as desired, by clicking on a cell and entering the new data. All dependent cells are also updated according to the new data.

11.

TITLE: Automatic Update of Dependent Cells

PRIORITY: high
ESTIMATE: 3 story points

AS A User
I WANT TO automatically recompute all dependent cells, if I change a formula within a specific cell
SO THAT I can easily update the spreadsheet and minimize errors.

ACCEPTANCE CRITERIA:

- If a formula is edited, the cell that contains the formula, as well as its dependent cells (the cells that reference it or the ones that use it within a range expression), are automatically updated.
- The values that have been recomputed are accurate and they reflect the new formula.

12.

TITLE: Error Handling for Data Types

PRIORITY: high
ESTIMATE: 1 story point

AS A User
I WANT TO receive accurate errors and clear feedback in the case that I introduce the wrong data type within the spreadsheet
SO THAT I can easily find the issue and modify the data type accordingly.

ACCEPTANCE CRITERIA:

- If the user adds a type that is not a string or numeric value (such as a symbol), the spreadsheet will show an informative error message indicating that the data type used is not valid.

13.

TITLE: Error Handling for Cell References and Range Expressions

PRIORITY: high

ESTIMATE: 1 story point

AS A User

I WANT TO receive accurate errors and clear feedback in the case that I refer to a cell that does not exist (or a cell that is clear) or I create a range expression using cells that do not exist (or cells that are clear)

SO THAT I can easily identify the issue and modify the cell reference or range expression accordingly.

ACCEPTANCE CRITERIA:

- If the user tries to create a cell reference using a cell that does not exist (or a cell that is clear), the spreadsheet will show an informative error message indicating the issue.
- If the user tries to create a range expression using cells that do not exist (or cells that are clear), the spreadsheet will show an informative error message indicating the issue.
- Circular references (e.g., A1 referring to B1 and B1 referring back to A1) will also be detected to avoid infinite loops.

14.

TITLE: Error Handling for Formulas

PRIORITY: high

ESTIMATE: 2 story points

AS A User

I WANT TO receive accurate errors and clear feedback in the case that I define a formula that cannot be computed, in a specific cell

SO THAT I can easily identify the issue and can fix the formula in cause as desired.

ACCEPTANCE CRITERIA:

- If the user tries to divide a number by 0 within a specific formula, the spreadsheet will show an error message indicating that this operation is not possible.
- If the user tries to add symbols (other than the ones that represent the arithmetic operators) to their formulas, the spreadsheet will show an error message indicating that it is forbidden.

- If the user uses parentheses within their formula that do not have a proper syntax (for example if they open a parenthesis, but never close it), the spreadsheet will show an error message indicating that the syntax is not correct.

15.

TITLE: Error Handling for Deleting Rows and Columns

PRIORITY: high

ESTIMATE: 1 story point

AS A User

I WANT TO receive accurate errors and clear feedback in the case that I try to delete a row or column that contains referenced values

SO THAT I can easily identify the issue and fix the cell reference in cause as desired.

ACCEPTANCE CRITERIA:

- If the user tries to delete a row or a column that contains a cell that is being referenced somewhere else within the spreadsheet, it will show an error message indicating that the values cannot be deleted until the cell reference is updated (it should reference a cell that is not within that row or column).

16.

TITLE: Error Handling for Clearing Cells

PRIORITY: high

ESTIMATE: 1 story point

AS A User

I WANT TO receive accurate errors and clear feedback in the case that I clear a cell that is referenced somewhere within the spreadsheet

SO THAT I can easily identify the issue and update the cell reference in cause appropriately.

ACCEPTANCE CRITERIA:

- If the user tries to clear a cell that is being referenced somewhere else within the spreadsheet, it will

show an error message indicating that the value of the cell cannot be deleted until the cell reference is updated (it should reference a different cell).

Additional features

17.

TITLE: Search bar

PRIORITY: low

ESTIMATE: 3 story points

AS A User

I WANT TO search for specific keywords or values

SO THAT I can easily identify data within the spreadsheet when needed.

ACCEPTANCE CRITERIA:

- A search bar is present at the top of the spreadsheet.
- Users can enter specific keywords and values in the search bar.
- The keywords or values searched are accurately identified within the spreadsheet.
- The matching cells that contain the desired value or word are highlighted.
- If the word or value is not found within the spreadsheet, then no cell will be highlighted.

18.

TITLE: Creating Charts

PRIORITY: low

ESTIMATE: 5 story points

AS A User

I WANT TO create charts based on the values within the spreadsheet

SO THAT I can easily visualize and present the data.

ACCEPTANCE CRITERIA:

- Users can select ranges of cells to create charts based on that data.
- The chart values, style and location can be easily customized by the user through a menu.
- The charts are accurately displayed within the spreadsheet.
- The charts update automatically when the cells they are based on are updated
- If the range of cells used for the chart contains types that are not numeric values, the spreadsheet will show an error message indicating that the types used in the chart are not valid.

19.

TITLE: Undo/redo and version history

PRIORITY: low

ESTIMATE: 5 story points

AS A User
I WANT TO save versions of the spreadsheet while I am editing

SO THAT I can easily retrieve any previous work if I make a mistake or want to go back to an older version.

ACCEPTANCE CRITERIA:

- Allows users to explicitly save the current spreadsheet state, by clicking on the save button.
- Users can retrieve any of the last 5 versions that they have saved.
- Spreadsheet cannot be saved again if no changes have been made since the last save state. The save button will be disabled if the spreadsheet was not changed.
- Clicking the undo button will set the spreadsheet to the previous version and allow the user to press the redo button to take the spreadsheet back to the latest change

Typescript Classes

CellContent Interface

```
/*
* CellContent represents the contents of a cell with its value stored in string form.
*/
export interface CellContent {
   /*
   * Returns the content of the cell in string form.
   */
   getContent(): string;

   /*
   * Changes the value of the cell to the given newval argument.
   */
   setContent(newval: string): void;
}
```

NumericLiteral class

```
/*
* CellContent that represents a numeric literal.
*/
export class NumericLiteral implements CellContent {

   private val: string; // the value of the numeric literal in string format

   /**
    * Constructor to initialize the numeric literal with the given value.
    */
   public constructor(val: string) {
       this.val = val;
   }

   /*
   * Returns the content of the cell in string form.
   */
   public getContent(): string {
       throw new Error('Method not implemented.');
   }

   /*
```

```
   * Changes the value of the cell to the given newval argument.
   */
   public setContent(newval: string): void {
       // ...
   }

}
```

StringLiteral class

```
/*
* CellContent that represents a string literal.
*/
export class StringLiteral implements CellContent {

   private val: string; // the content of the string literal

   /**
    * Constructor to initialize the string literal with the given value.
    */
   public constructor(val: string) {
       this.val = val;
   }

   /*
   * Returns the content of the cell in string form.
   */
   public getContent(): string {
       throw new Error('Method not implemented.');
   }

   /*
   * Changes the value of the cell to the given newval argument.
   */
   public setContent(newval: string): void {
       // ...
   }

}
```

RangeExpression class

```
/*
* CellContent that represents a range expression.
*/
export class RangeExpression implements CellContent {

   private val: string; // the range expression in string format (e.g. "SUM(<range of
cells>)")

   /**
    * Constructor to initialize the string literal with the given value.
    */
   public constructor(val: string) {
       // placeholder, later on we will update the constructor or getContent
       // to actually evaluate the Range Expression
       this.val = val;
   }

   /*
   * Returns the content of the cell in string form.
   */
   public getContent(): string {
       throw new Error("Method not implemented.");
   }

   /*
   * Changes the value of the cell to the given newval argument.
   */
   public setContent(newval: string): void {
       // ...
   }
}
```

CellReference class

```
/*
* CellContent that represents a cell reference.
*/
export class CellReference implements CellContent {

   private val: string; // the string that represents the cell reference (e.g.
"REF(<cell>)")
```

```
    /**
     * Constructor to initialize the cell reference with the given value.
     */
    public constructor(val: string) {
        // placeholder, later we will update the constructor or getContent
        // to return the id of the referenced cell
        this.val = val;
    }

    /*
    * Returns the content of the cell in string form.
    */
    public getContent(): string {
        throw new Error("Method not implemented.");
    }

    /*
    * Changes the value of the cell to the given newval argument.
    */
    public setContent(newval: string): void {
        // ...
    }
}
```

Formula class

```
/*
* CellContent that represents a formula.
*/
export class Formula implements CellContent {

    private val: string; // the formula in a string format

    /**
     * Constructor to initialize the formula with the given value.
     */
    public constructor(val: string) {
        // placeholder, later we will update the constructor or getContent
        // to store the result of the formula
        this.val = val;
    }
```

```
    /*
    * Returns the content of the cell in string form.
    */
    public getContent(): string {
        throw new Error("Method not implemented.");
    }

    /*
    * Changes the value of the cell to the given newval argument.
    */
    public setContent(newval: string): void {
        // ...
    }

}
```

Cell class

```
/**
* Cell represents a cell from the spreadsheet. Each Cell object contains a cell
content,
* a row number and a column number.
*/
export class Cell {
    private content: CellContent;  // the content of the cell
    private row: number;  // the row number of the cell
    private col: number;  // the column number of the cell

    /**
     * Constructor to initialize the cell with the content, row number and column
number.
     */
    constructor(content: CellContent, row: number, col: number) {
        this.content = content;
        this.row = row;
        this.col = col;
    }

    /**
     * Method to clear or reset the cell content.
     */
```

```
public clear(): void {
    // …
}

/**
* Method to set the value of the cell content.
*/
public setCellContent(value: CellContent): void {
    // …
}

/**
 * Method to get the value of the cell content.
 */
public getCellContent(): CellContent {
    // …
    throw new Error('Method not implemented.');
}

/**
 * Method to set the column number to a given value.
 */
public setCol(colNr: number): void {
    // …
}

/**
 * Method to get the column number of the cell.
 */
public getCol(): number {
    // …
    throw new Error('Method not implemented.');
}

/**
 * Method to set the row number to a given value.
 */
public setRow(rowNr: number): void {
    // …
}

/**
```

```
    * Method to get the row number of a cell.
    */
   public getRow(): number {
       // …
       throw new Error('Method not implemented.');
   }

   /**
    * Method that handles various errors for a cell instance.
    */
   handleErrors(): void {
       // …
   }
}
```

Spreadsheet class

```
/**
* Spreadsheet class manages a 2D array of cells and provides functionalities
* like adding/deleting rows/columns, getting/setting/clearing cell contents,
* handling errors, finding content, evaluating content, and managing version history.
*/
export class Spreadsheet {
   private cells: Cell[][];  // 2D array to hold the cells of the spreadsheet
   private versionHistory: Spreadsheet[];  // array to hold the version history of the
spreadsheet
   private maxRow: number;  // maximum allowed rows in the spreadsheet
   private maxCol: number;  // maximum allowed columns in the spreadsheet


   /**
    * Constructor to initialize the spreadsheet with cells, version history, and
optionally set max rows and columns.
    */
   private constructor(cells: Cell[][], versionHistory: Spreadsheet[], maxRow: number
= 1000, maxCol: number = 52) {
       this.cells = cells;
       this.versionHistory = versionHistory;
       this.maxRow = maxRow;
       this.maxCol = maxCol;
   }
```

```
/**
 * Method to add a new row to the spreadsheet.
 */
public addRow(): void {
    // ...
}

/**
 * Method to add a new column to the spreadsheet.
 */
public addColumn(): void {
    // ...
}

/**
 * Method to delete a row from the spreadsheet based on the provided index.
 */
public deleteRow(index: number): void {
    // ...
}

/**
 * Method to delete a column from the spreadsheet based on the provided index.
 */
public deleteColumn(index: number): void {
    // ...
}

/**
 * Method to retrieve the content of a cell at the specified row and column.
 */
public getCell(row: number, col: number): Cell {
    throw new Error("Method not implemented.");
}

/**
 * Method to set the content of a cell at the specified row and column.
 */
public setCell(row: number, col: number, value: Cell): void {
    // ...
}
```

```
    /**
     * Method to clear the content of a cell at the specified row and column.
     */
    public clearCellContents(row: number, col: number): void {
        // ...
    }

    /**
     * Method to handle errors and perform necessary actions or logging.
     */
    public handleErrors(): void {
        // ...
    }

    /**
     * Method to find all occurrences of a string in the spreadsheet and return their
positions.
     */
    public find(searchString: string): [number, number][] {
        throw new Error("Method not implemented.");
    }



    /**
     * Method to evaluate the content of a cell at the specified row and column.
     */
    public evaluateContent(row: number, col: number): any {
        throw new Error("Method not implemented.");
    }



    /**
     * Method to retrieve a specific version of the spreadsheet based on the provided
index.
     */
    public retrieveVersion(versionIndex: number): Spreadsheet {
        throw new Error("Method not implemented.");
    }



    /**
     * Method to save the current state of the spreadsheet to the version history.
```

```
    */
   public saveVersion(): void {
       // ...
   }
}
```

Controller Class

```
/**
* Controller class that connects UI and Spreadsheet model implementation.
*/
export class Controller {
   private spreadsheet: Spreadsheet;  // the spreadsheet object that contains all of
the data inputted by the user
   private userInput: string;  // the user input received from the frontend part of
the application

   /**
    * Constructor to initialize the controller with a spreadsheet object and a string
that represents
    * the user input.
    */
   public constructor(spreadsheet: Spreadsheet, userInput: string) {
       this.spreadsheet = spreadsheet;
       this.userInput = userInput;
   }

   /**
    * Method that communicates with the visual representation and spreadsheet model in
response to the user
    * interaction. It updates the spreadsheet according to the activity of the user.
    */
   public runSpreadsheet(): void {
       // ...
   }

   /**
    * Method that parses the user input to determine what their intended change is.
    */
   public parseUserInput(): void {
       // ...
   }
```

```
}
```

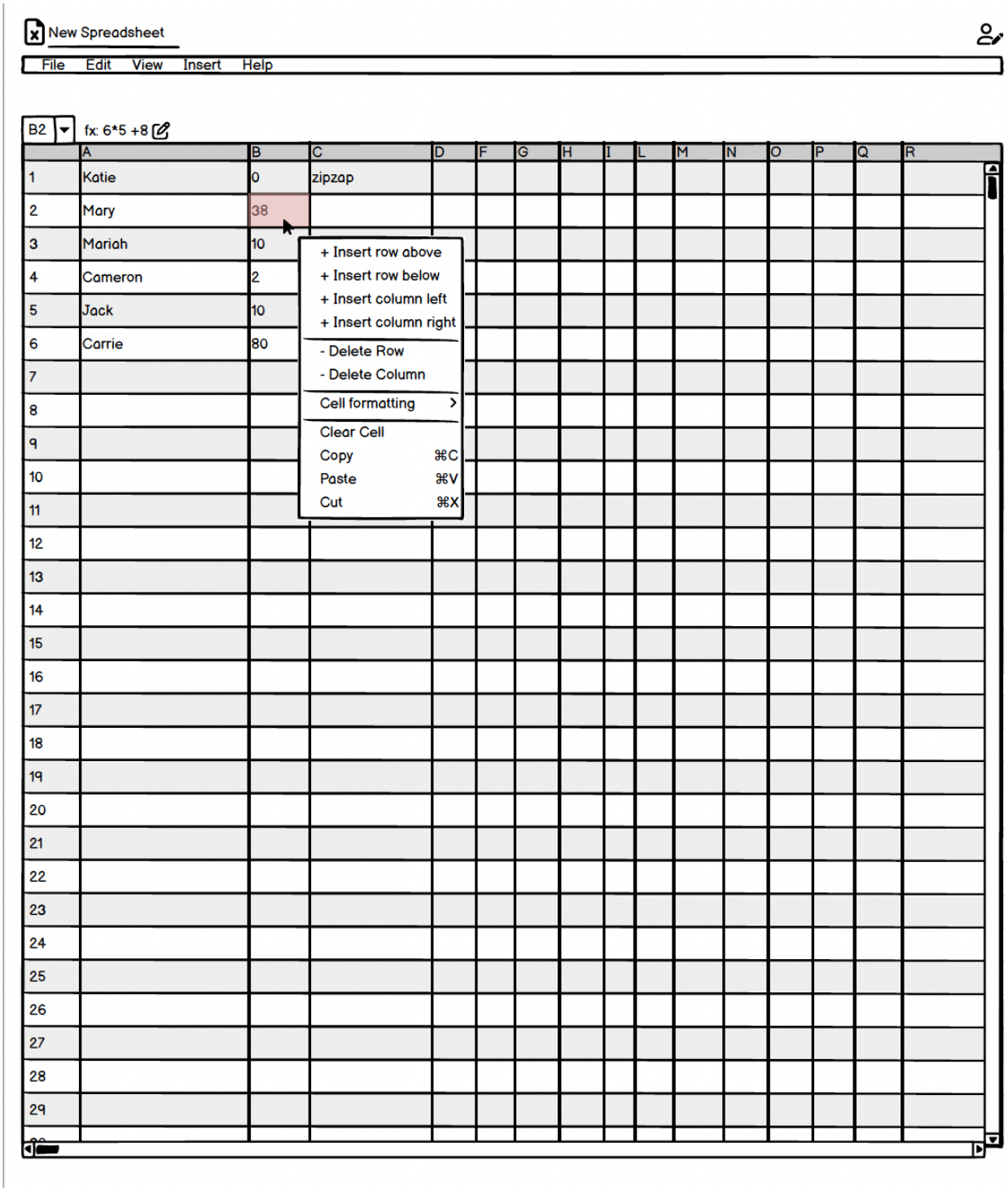
ADataRepresentation class

```
/*
* ADataRepresentation class that gathers and organizes the data of all cells chosen to
* be made into a graph. The will serve as an abstract class that can be extended for
* different types of graphs we decide to represent, which will use the same
constructor
* but not necessarily the same visualize() function (e.g. scatter, bar, line, pie
chart)
*/
export class ADataRepresentation {
   private cellData: Array<Cell>; // the cell data used in the data representation

   /**
    * Constructor that parses string for cell references, locates the correct cells in
the spreadsheet
    * and stores them for reference so that the graph can be updated with the related
cells.
   */
   public constructor(rangeOfCells: string, spreadsheet: Spreadsheet) {
       // ...
   }

   /**
    * Method to organize data for specific display purposes.
    */
   public visualize(): void {}
}
```

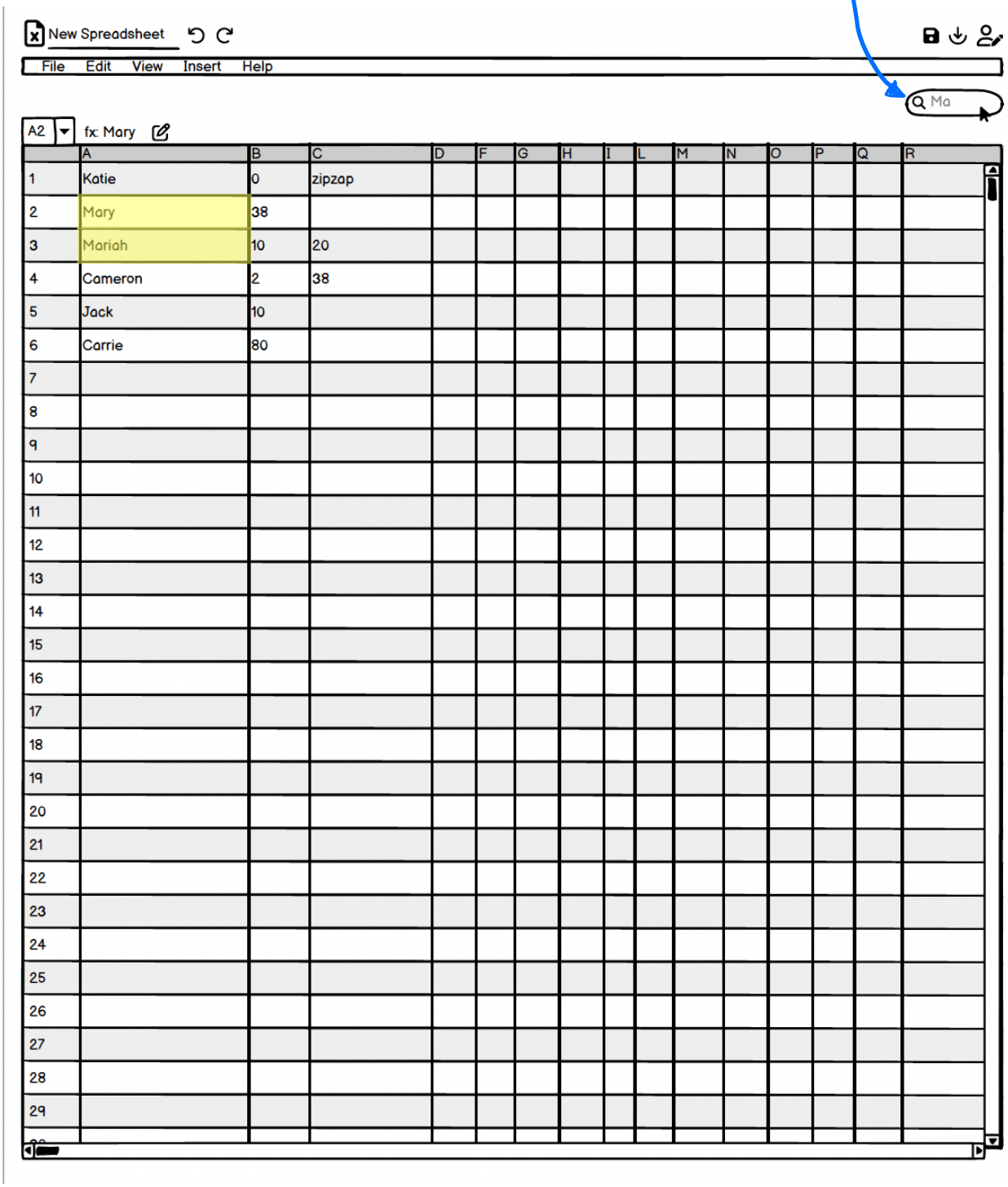
UI Prototypes



	A	B	C	D	F	G	H	I	L	M	N	O	P	Q	R
1	Katie	0	zipzap												
2	Mary	38													
3	Mariah	10													
4	Cameron	2													
5	Jack	10													
6	Carrie	80													
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															
27															
28															
29															

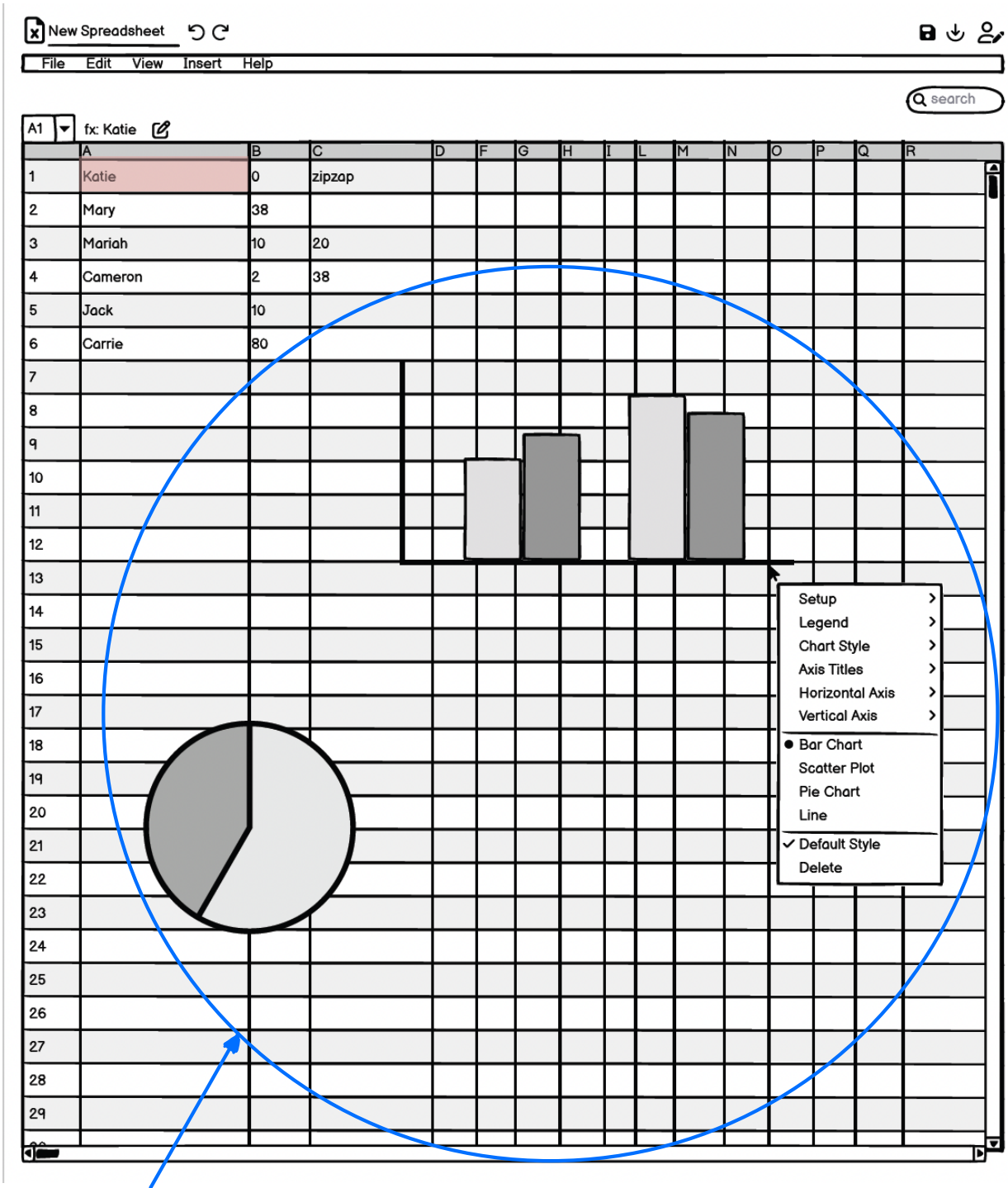
Users can search for specific keywords or values within the spreadsheet. The value or string entered will be highlighted within the spreadsheet, if found.

Additional Feature 1 - Search Bar



	A	B	C	D	F	G	H	I	L	M	N	O	P	Q	R
1	Katie	0	zipzap												
2	Mary	38													
3	Mariah	10	20												
4	Cameron	2	38												
5	Jack	10													
6	Carrie	80													
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															
27															
28															
29															

Additional Feature 2 - Data Representation



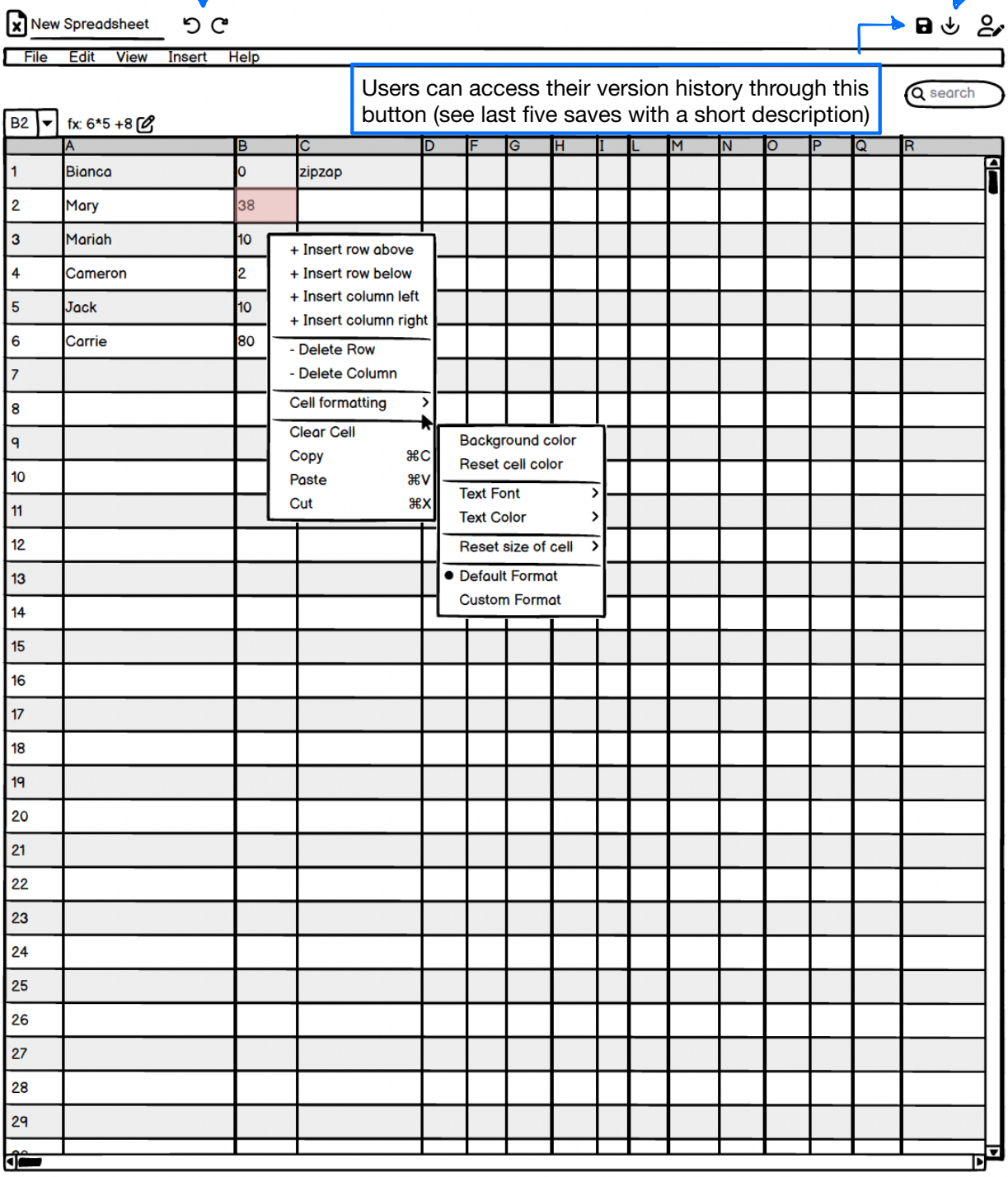
	A	B	C
1	Katie	0	zipzap
2	Mary	38	
3	Mariah	10	20
4	Cameron	2	38
5	Jack	10	
6	Carrie	80	

Users can create charts based off of the values within their spreadsheet. Moreover, if a user right clicks on the chart, they can find a menu with multiple options, through which they can customize the chart however they want.

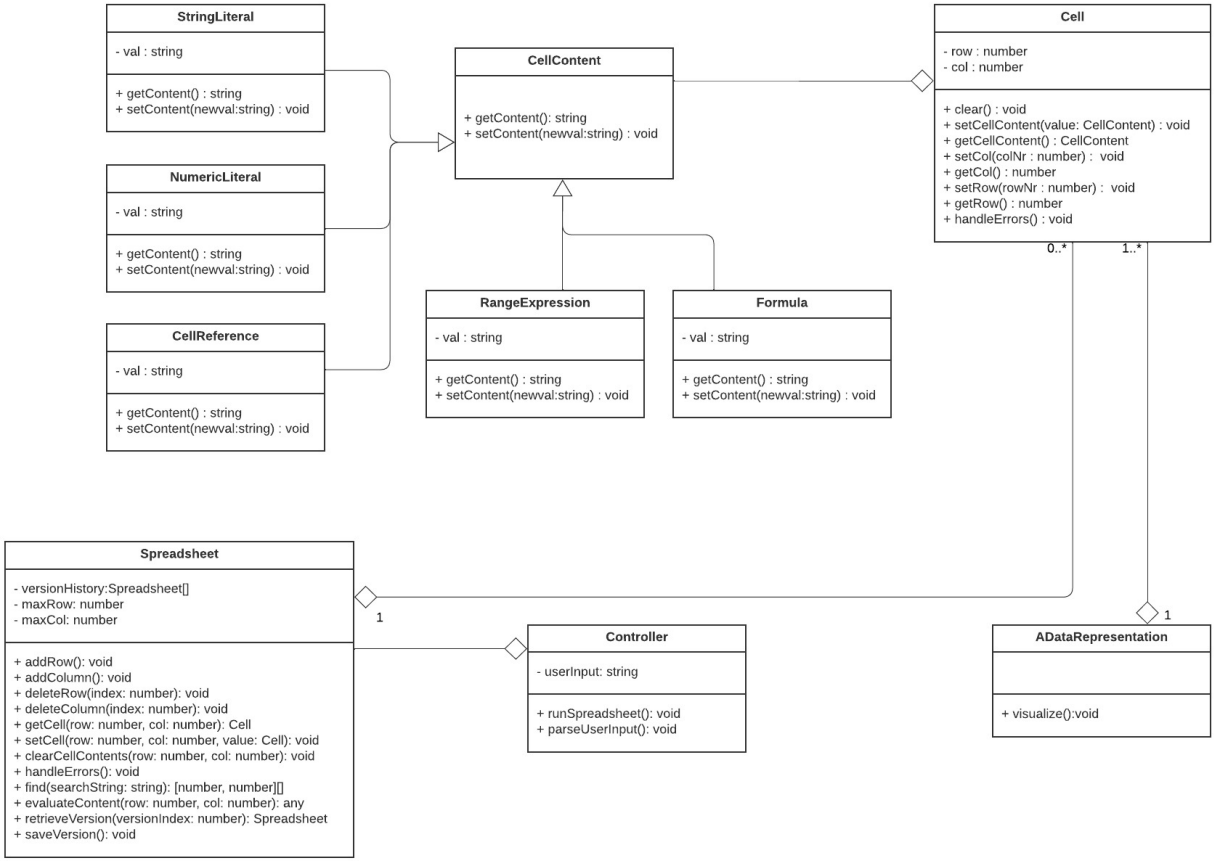
Through the undo button, users can retrieve the previous version saved. Through the redo button, users can return to their latest changes, which were visible before the undo action.

By pressing on this, users can save their current version of the spreadsheet.

Additional Feature 3 - Version History



UML Diagram


StringLiteral
- val : string
+ getContent() : string
+ setContent(newval:string) : void
NumericLiteral
- val : string
+ getContent() : string
+ setContent(newval:string) : void
CellReference
- val : string
+ getContent() : string
+ setContent(newval:string) : void
CellContent
+ getContent(): string
+ setContent(newval:string) : void
RangeExpression
- val : string
+ getContent() : string
+ setContent(newval:string) : void
Formula
- val : string
+ getContent() : string
+ setContent(newval:string) : void
Cell
- row : number
- col : number
+ clear() : void
+ setCellContent(value: CellContent) : void
+ getCellContent() : CellContent
+ setCol(colNr : number) : void
+ getCol() : number
+ setRow(rowNr : number) : void
+ getRow() : number
+ handleErrors() : void
0..*
1..*
Spreadsheet
- versionHistory:Spreadsheet[]
- maxRow: number
- maxCol: number
+ addRow(): void
+ addColumn(): void
+ deleteRow(index: number): void
+ deleteColumn(index: number): void
+ getCell(row: number, col: number): Cell
+ setCell(row: number, col: number, value: Cell): void
+ clearCellContents(row: number, col: number): void
+ handleErrors(): void
+ find(searchString: string): [number, number][]
+ evaluateContent(row: number, col: number): any
+ retrieveVersion(versionIndex: number): Spreadsheet
+ saveVersion(): void
1
Controller
- userInput: string
+ runSpreadsheet(): void
+ parseUserInput(): void
1
ADataRepresentation
+ visualize():void