Spreadsheet

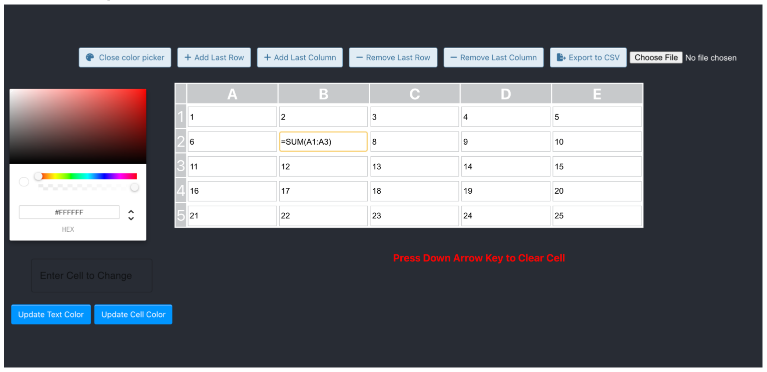
Team-109

Hector Padilla, Dylan Mccann, Maddie Lebiedzinski, Ousman Jobe

Summary of your system's functionality, focusing particularly on the three additional features. Please consider illustrating this using a few small screenshots (no more than 1 page of screenshots please).

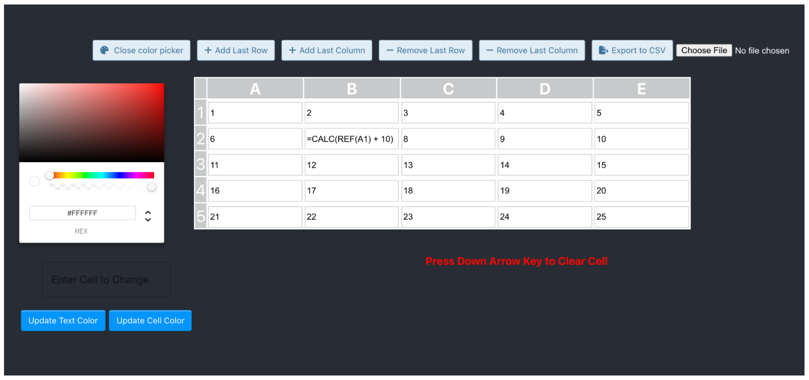
Our application successfully implements basic spreadsheet functionality such as allowing cells to contain numeric constants, string constants, cell references, and formulas. The spreadsheet allows users to insert rows and columns, as well as delete rows and columns. Also, users can clear contents of a cell instantaneously upon click of the down arrow key. Additional features include being able to change the color of an individual cell as well as the color of the cell's text. Users can select a color using the color picker widget, specify the cell to be changed, and then select whether to change the full cell or text. Another additional feature allows for exporting the spreadsheet as a csv. Upon clicking the export button in the UI, it automatically creates a download of the spreadsheet's contents. Finally, users can import a csv file containing contents to be shown in the spreadsheet.

Sum expression :



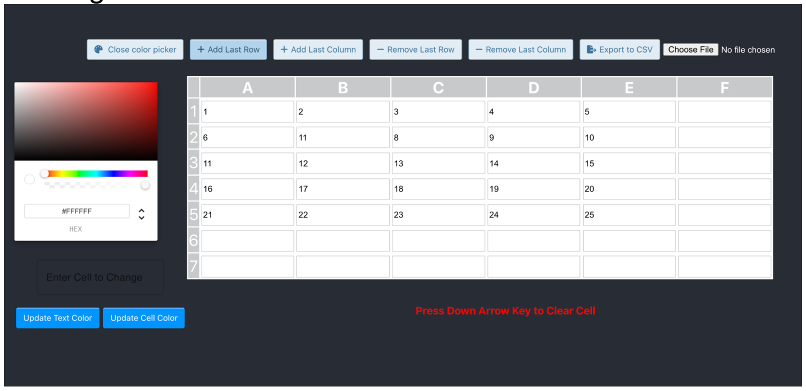
	A	B	C	D	E
1	1	2	3	4	5
2	6	=SUM(A1:A3)	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20
5	21	22	23	24	25

Formula and cell reference :



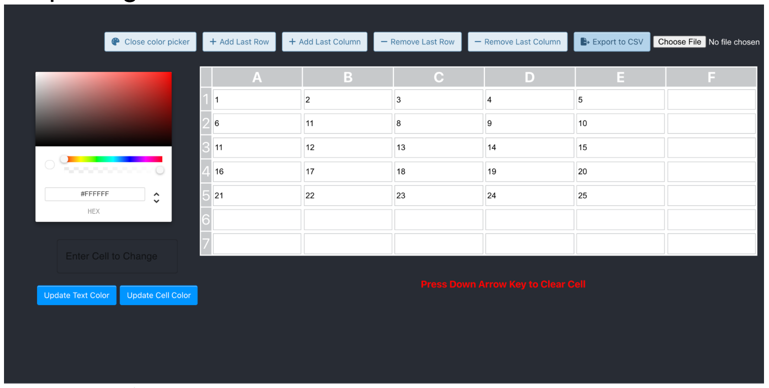
	A	B	C	D	E
1	1	2	3	4	5
2	6	=CALC(REF(A1) + 10)	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20
5	21	22	23	24	25

Adding a Row/Column :



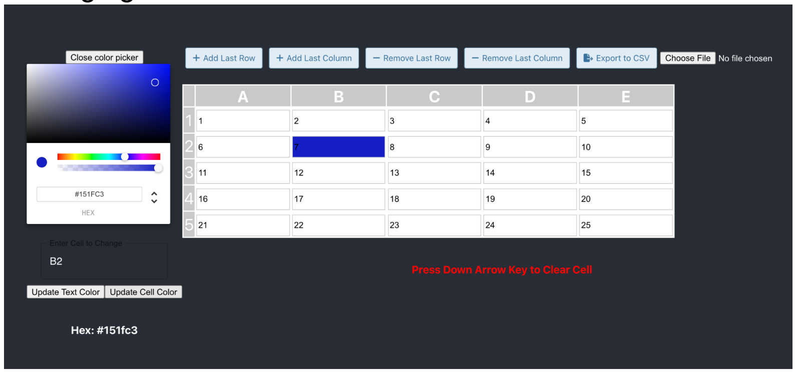
	A	B	C	D	E	F
1	1	2	3	4	5	
2	6	11	8	9	10	
3	11	12	13	14	15	
4	16	17	18	19	20	
5	21	22	23	24	25	
6						
7						

Exporting to CSV :



	A	B	C	D	E	F
1	1	2	3	4	5	
2	6	11	8	9	10	
3	11	12	13	14	15	
4	16	17	18	19	20	
5	21	22	23	24	25	
6						
7						

Changing cell color :

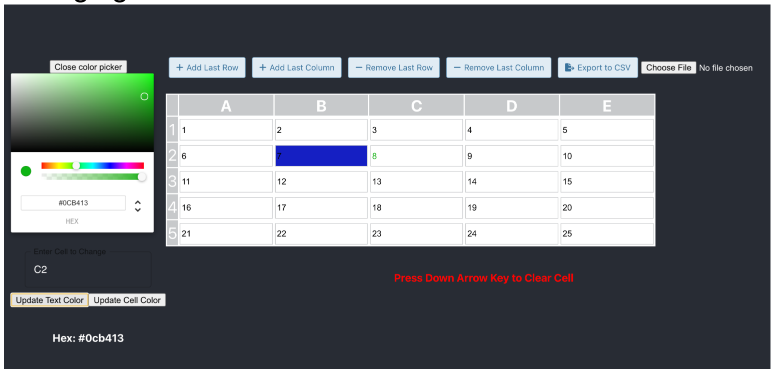


B2

Hex: #151fc3

	A	B	C	D	E
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20
5	21	22	23	24	25

Changing Text Color :

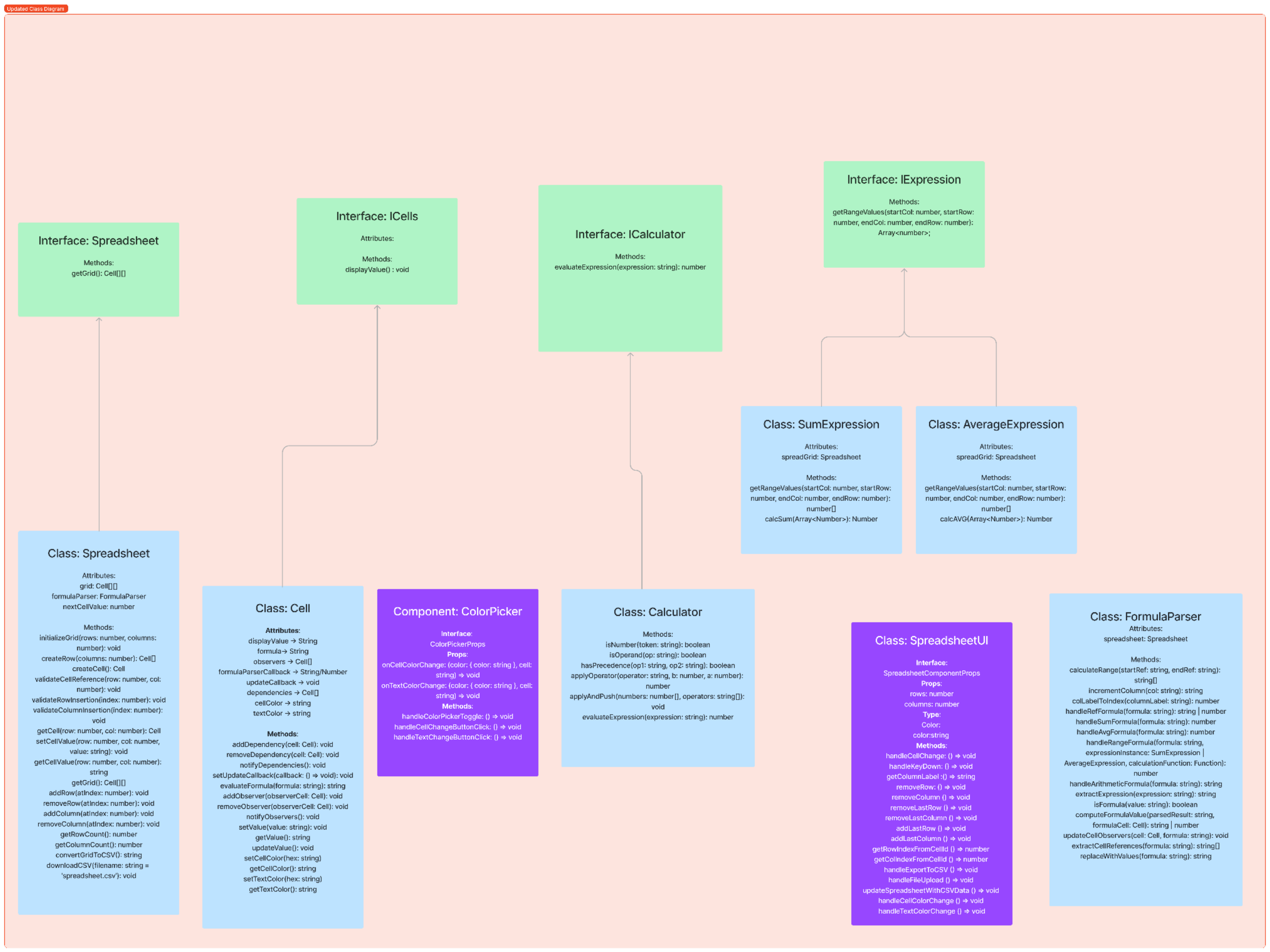


C2

Hex: #0cb413

	A	B	C	D	E
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20
5	21	22	23	24	25

1. Description of the high-level architecture of your system, focusing on what are the major components, and how they communicate with each other. Please consider visualizing your high-level design using a diagram (e.g., a UML Class Diagram), but please limit yourself to no more than 1 page of diagrams.


Updated Class Diagram
Interface: Spreadsheet
Methods:
getGrid(): Cell[][]
Interface: ICells
Attributes:
Methods:
displayValue() : void
Interface: ICalculator
Methods:
evaluateExpression(expression: string): number
Interface: IExpression
Methods:
getRangeValues(startCol: number, startRow: number, endCol: number, endRow: number): Array<number>;
Class: SumExpression
Attributes:
spreadGrid: Spreadsheet
Methods:
getRangeValues(startCol: number, startRow: number, endCol: number, endRow: number): number[]
calcSum(Array<Number>): Number
Class: AverageExpression
Attributes:
spreadGrid: Spreadsheet
Methods:
getRangeValues(startCol: number, startRow: number, endCol: number, endRow: number): number[]
calcAVG(Array<Number>): Number
Class: Spreadsheet
Attributes:
grid: Cell[][]
formulaParser: FormulaParser
nextCellValue: number
Methods:
initializeGrid(rows: number, columns: number): void
createRow(columns: number): Cell[]
createCell(): Cell
validateCellReference(row: number, col: number): void
validateRowInsertion(index: number): void
validateColumnInsertion(index: number): void
getCell(row: number, col: number): Cell
setCellValue(row: number, col: number, value: string): void
getCellValue(row: number, col: number): string
getGrid(): Cell[][]
addRow(atIndex: number): void
removeRow(atIndex: number): void
addColumn(atIndex: number): void
removeColumn(atIndex: number): void
getRowCount(): number
getColumnCount(): number
convertGridToCSV(): string
downloadCSV(filename: string = 'spreadsheet.csv'): void
Class: Cell
Attributes:
displayValue → String
formula→ String
observers → Cell[]
formulaParserCallback → String/Number
updateCallback → void
dependencies → Cell[]
cellColor → string
textColor → string
Methods:
addDependency(cell: Cell): void
removeDependency(cell: Cell): void
notifyDependencies(): void
setUpdateCallback(callback: () ⇒ void): void
evaluateFormula(formula: string): string
addObserver(observerCell: Cell): void
removeObserver(observerCell: Cell): void
notifyObservers(): void
setValue(value: string): void
getValue(): string
updateValue(): void
setCellColor(hex: string)
getCellColor(): string
setTextColor(hex: string)
getTextColor(): string
Component: ColorPicker
Interface:
ColorPickerProps
Props:
onCellColorChange: (color: { color: string }, cell: string) ⇒ void
onTextColorChange: (color: { color: string }, cell: string) ⇒ void
Methods:
handleColorPickerToggle: () ⇒ void
handleCellChangeButtonClick: () ⇒ void
handleTextChangeButtonClick: () ⇒ void
Class: Calculator
Methods:
isNumber(token: string): boolean
isOperand(op: string): boolean
hasPrecedence(op1: string, op2: string): boolean
applyOperator(operator: string, b: number, a: number): number
applyAndPush(numbers: number[], operators: string[]): void
evaluateExpression(expression: string): number
Class: SpreadsheetUI
Interface:
SpreadsheetComponentProps
Props:
rows: number
columns: number
Type:
Color:
color:string
Methods:
handleCellChange: () ⇒ void
handleKeyDown: () ⇒ void
getColumnLabel :() ⇒ string
removeRow: () ⇒ void
removeColumn () ⇒ void
removeLastRow () ⇒ void
removeLastColumn () ⇒ void
addLastRow () ⇒ void
addLastColumn () ⇒ void
getRowIndexFromCellId () ⇒ number
getColIndexFromCellId () ⇒ number
handleExportToCSV () ⇒ void
handleFileUpload () ⇒ void
updateSpreadsheetWithCSVData () ⇒ void
handleCellColorChange () ⇒ void
handleTextColorChange () ⇒ void
Class: FormulaParser
Attributes:
spreadsheet: Spreadsheet
Methods:
calculateRange(startRef: string, endRef: string): string[]
incrementColumn(col: string): string
colLabelToIndex(columnLabel: string): number
handleRefFormula(formula: string): string | number
handleSumFormula(formula: string): number
handleAvgFormula(formula: string): number
handleRangeFormula(formula: string, expressionInstance: SumExpression | AverageExpression, calculationFunction: Function): number
handleArithmeticFormula(formula: string): string
extractExpression(expression: string): string
isFormula(value: string): boolean
computeFormulaValue(parsedResult: string, formulaCell: Cell): string | number
updateCellObservers(cell: Cell, formula: string): void
extractCellReferences(formula: string): string[]
replaceWithValues(formula: string): string

The Calculator class implements the ICalculator interface to evaluate arithmetic operations. The Spreadsheet class implements the ISpreadsheet interface and integrates with the FormulaParser class to manage a grid of cell objects. The Cell class implements the ICells interface to create a cell object for storing data and integrates with the ColorPicker class and helps handle cell dependencies. The SumExpression and AverageExpression classes implement the IExpression interface to calculate the sums and averages in a range of cells/cell references. The SpreadsheetUI class manages the user controls for row & column addition and deletion as well as importing and exporting. The FormulaParser class handles the parsing of formulas in the spreadsheet and recognizes formulas starting with the equal sign such as SUM, AVG, CALC, and REF(reference). It manages cell dependency based on the formula, and replaces the cell reference with their respective cells' value(s).

2. Description of the approach that you used to track dependencies between spreadsheet cells.

Observer Pattern:

For tracking cell dependencies we used the observer design pattern. Each cell acts as both a subject (observable) and an observer. As a subject, it can notify other cells (observers) when its value changes. As an observer, it can be notified by other cells (subjects) it depends on.

Cell Class:

This class represents a cell in the application. It maintains a list of dependent cells (observers) and cells it depends on (dependencies).

Observer Management:

AddObserver(observerCell: Cell) -> This adds a cell to the observers list, meaning this cell will be notified when the current cell's value changes

RemoveObserver(observerCell: Cell) -> This removes a cell from the observers list

notifyObservers() -> This notifies all observer cells to update their value, typically called when the current cell's value changes

Dependency Tracking:

addDependency(cell: Cell) -> This adds a cell to the dependency list, indicating that the current cell's value depends on the added cell.

removeDependency(cell: Cell) -> This removes a cell from the dependency list.

notifyDependencies() -> This notifies all cells in the dependency list, typically used to update dependent cells when a cell they depend on changes.

Value Setting and Formula Evaluation:

setValue(value: string) -> This sets the cell's value. If the value is a formula (starts with '='), it evaluates the formula using `evaluateFormula()` and then notifies both observers and dependencies.

evaluateFormula(formula: string) -> This uses `formulaParserCallback` to evaluate the formula.

Formula Parsing and Computation:

The `FormulaParser` class handles formula evaluation and cell reference extraction. It parses and computes formulas like SUM, AVG, REF, and arithmetic expressions.

computeFormulaValue(parsedResult: string, formulaCell: Cell) -> This computes the value of a formula and updates cell observers based on the formula's dependencies.

Observer-Dependency Linking:

When a cell's formula is set or changed, the `FormulaParser` identifies all cells referenced in the formula. These referenced cells are added as dependencies to the cell with the formula, and the cell with the formula is added as an observer to the referenced cells. This ensures that any change in the value of the referenced cells triggers an update in the cell with the formula.

This approach enabled dynamic and efficient tracking of cell dependencies and updates.

3. Discussion of how the design and functionality of your system evolved since the phase B plan.

Since phase B most of our design and functionality has remained consistent. Slight changes include making the down key arrow delegate cell clearing. The design of our UI has changed a bit based on what was best suited for our functionality. We also implemented error handling, which was not initially included in our diagrams.

4. Discussion of the development process that your team followed, including techniques, tools, methods used, libraries used (e.g. for parsing), division of work between the team members.

We followed agile methodologies. We would give ourselves user stories for the week, and complete each of them individually. Sometimes we would do pair programming either in person or through zoom. User stories and feature implementations were successfully divided so that progress was made in a timely fashion. We implemented a color picker, called Chrome, from react-color for the coloring feature. We used Material UI for features such as the TextField within the ColorPicker component.

5. Reflection on your development experience, problems encountered, lessons learned.

Our development experience was marked by several successes and challenges. The team effectively managed work scheduling during our sprints, maintaining a steady pace that allowed us to consistently implement user stories and complete tasks in a timely manner. Teamwork and communication were strong, fostering a collaborative environment where everyone contributed in various ways. Our design successfully aligned with our UML diagram from the planning phase of the project, we implemented required features using practices touched on in class, and we referenced calculator implementation, refactoring it to be incorporated into our project. However, some challenges arose when transitioning Typescript files, after initially working in Javascript files, leading to unnecessary conversion work. The initial approach to cell dependencies presented difficulties, requiring refactoring to ensure accurate updates of dependent cell values. Looking back, we recognize the potential for improvement by exploring third-party parsers to enhance parsing scope and performance. Additionally, alterations in our plan for selecting cells were made, deviating from the traditional approach used in mainstream spreadsheets. These experiences have emphasized the importance of careful project setup and the need for continuous evaluation and adaptation of implementation strategies. Overall, the process provided valuable insights into effective team dynamics, strategic decision-making, and the ongoing pursuit of optimized development practices.

6. Installation instructions and user guide (you may also include this with your code instead of in the report)

Installation:

Clone the repo and cd into spreadsheet-app

To install dependencies, run “npm install”.

For running tests in the terminal run, “npm run test:coverage”. After that runs in it will show all our tests and our code coverage for all branches. We have 85% branch coverage and 93% statement coverage for this project.

To run the program locally, run “npm start” in the terminal, which will open up a localhost:3000 tab leading you to the spreadsheet.

Using the Spreadsheet:

To use cell references, type =REF(CR) where C = the letter of the column and R = the number of the row.

For example =REF(A1) would create a cell reference to cell A1.

Similarly, for using formulas:

- =AVG(C_1R_1:C_2R_2) will compute the average of the range of cells.
- =SUM(C_1R_1:C_2R_2) will compute the sum of the range of cells.
- =CALC(expression) will compute the result of an expression using arithmetic operations, for example =CALC(3 * (2 + 1)) will result in the cell displaying 9.

To clear the contents of a cell, select the cell and use the down arrow key.