Final Report

Part 1 - Name of project, group number, team members

Name of project: Spreadsheet Project CS4530
Group number: 507
Name of team members:

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

Part 2 - Summary of System's Functionality

Our system can perform all of the required functionality mentioned in the project requirements, as well as five additional features that we have come up with in order to make our application more efficient and user-friendly. Our code handles editing cells, adding numeric and string values to cells, executing formulas, adding references to other cells within the spreadsheet and computing range expressions. Aside from the basic functionality presented, we have also added a search bar, undo/redo buttons, charts, a help page, functionality for editing the title of the spreadsheet and also a save button that saves the spreadsheet as a CSV file.

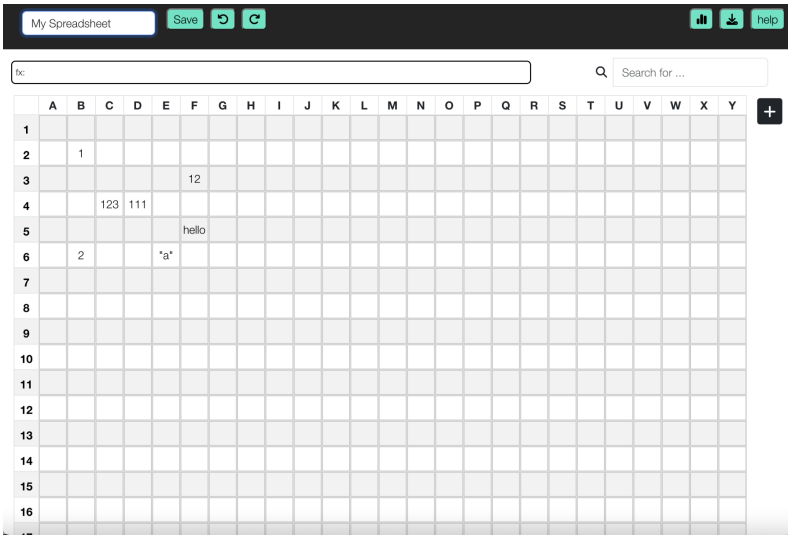
Basic functionality:

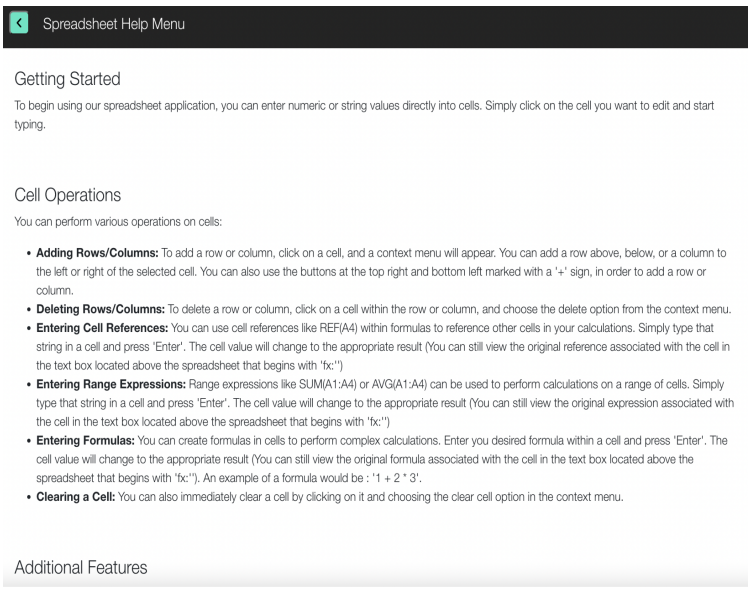
- Adding Rows/Columns: The user can add a row above, below, or a column to the left or right of the selected cell.
- Deleting Rows/Columns: The user can delete any row or column, by clicking on a cell within the row or column, and choosing the delete option from the associated context menu.
- Entering Cell References: The user can use cell references like "REF(A4)" within the spreadsheet to reference other cells. The cell references are automatically recomputed if the value of the referenced cell changes.
- Entering Range Expressions: Range expressions like "SUM(A1:A4)" or "AVG(A1:A4)" can also be used within the spreadsheet to perform calculations on a range of cells. The range expressions are automatically recomputed if the value of any of the referenced cells changes.
- Entering Formulas: The user can create formulas in cells to perform complex calculations. The formulas can contain cell references and range expressions, and they are automatically recomputed if any changes are made to the referenced cells. Our spreadsheet also supports string concatenation (e.g. "a" + "b").
- Clearing a Cell: The user can also immediately clear a cell by clicking on it and choosing the clear cell option in the context menu.

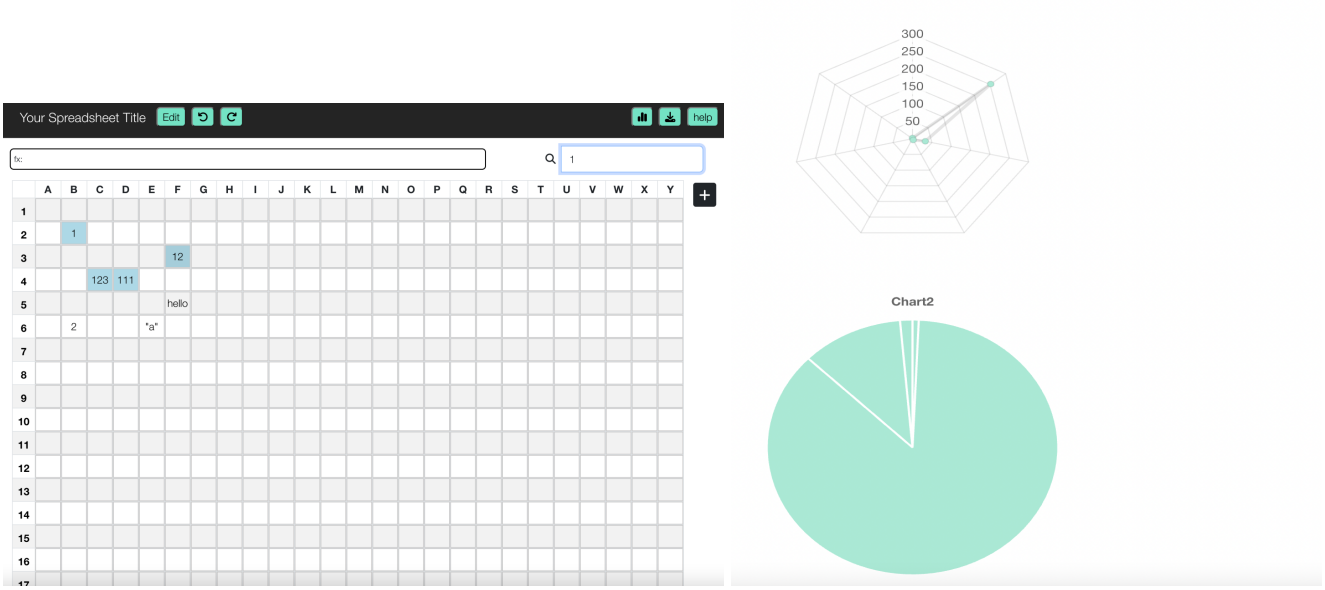
Our additional features:

- Undo/Redo: Our application provides undo and redo functionality to revert or reapply changes made to the spreadsheet. The user can simply press on the undo button in order to undo the last change they made. Similarly, the user can press on the redo button to go back to the change that was just reverted.
- Search: The user can utilize the search bar to quickly find specific content within the spreadsheet. They can click on the search bar and type the string or value that they would like to find. The cells that contain that string or value will be highlighted within the spreadsheet.
- Edit Spreadsheet Title: The user can edit the title of the spreadsheet by clicking on the 'Edit' button in the navigation menu and changing the string inside the text box.
- Creating Charts: The user can create charts based on the data contained in the spreadsheet. The charts can be customized according to the user's desires, using a menu located in the navigation bar. The graphs will be displayed under the spreadsheet.
- Save as CSV: The user can save the spreadsheet as a CSV file for external use or sharing, by clicking on the save button in the navigation menu.
- Help menu: The user can access a help menu, that includes instructions on how to use the features of our project, by clicking on the 'help' button in the navigation menu.

There is more information about how to utilize our features in Part 8 of the report which includes all of the instructions necessary to run our application.

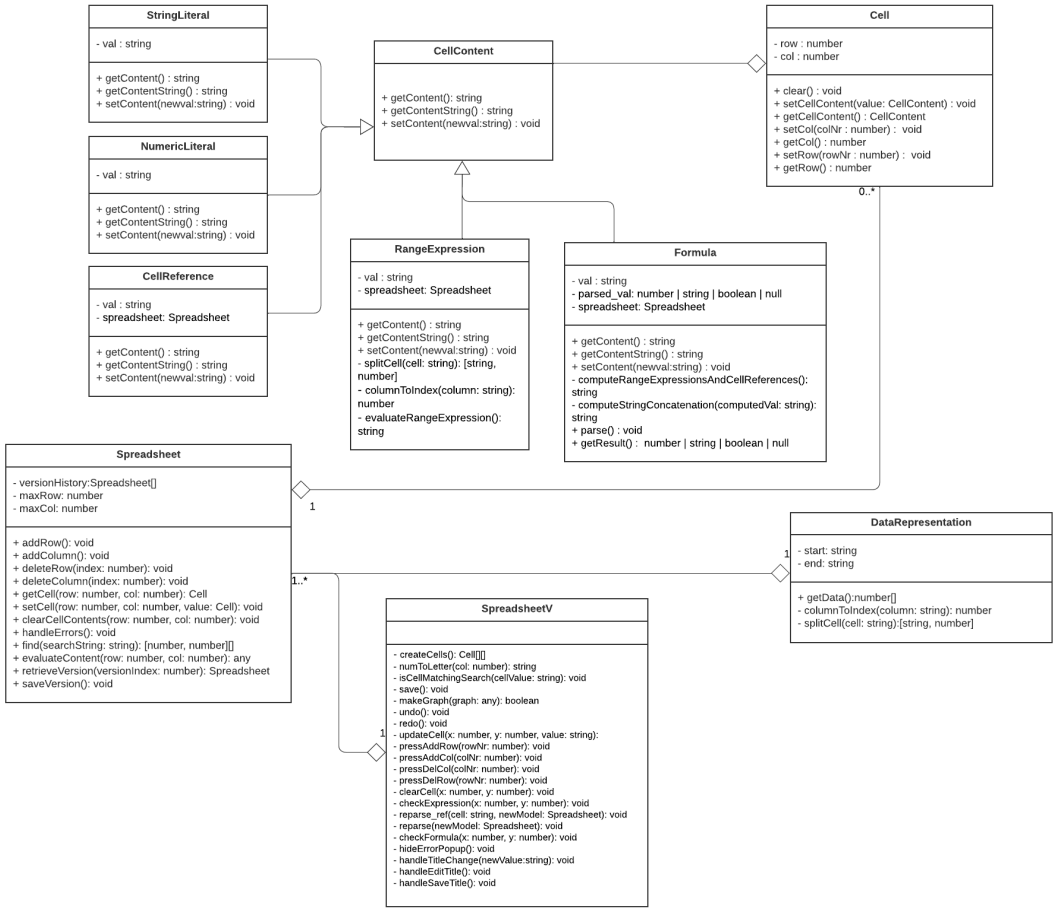






Part 3 - Infrastructure of Project

UML diagram



High Level Infrastructure:

Our code is separated into two parts: the frontend and the backend portions. The backend section is represented by the ‘model’ directory that holds the classes responsible for computing the logic of the spreadsheet. The frontend part of our project is contained within the ‘ssdisplay’ directory, that contains all of the components that are used to build the UI through which the user can interact with our project. We will first present the backend section of the application, which is represented by the contents of the ‘model’ directory.

The centerpiece of the system is the Spreadsheet class, which functions as a container for the data and offers functionalities such as adding or removing rows and columns, as well as accessing individual cells. Cells within the spreadsheet are represented by the Cell class. The Cell Class represents a single cell within the spreadsheet, holding information about its row and column, and it can contain different types of content.

The ICellContent acts as a superclass for different types of content that a cell can hold. It defines the interface for getting and setting content. StringLiteral, NumericLiteral, CellReference, RangeExpression and Formula are subclasses of CellContent, representing different types of data that a cell can contain, such as strings, numbers, and formulas, etc. This follows the design principle of inheritance meaning they inherit its methods and properties and can be used interchangeably where a CellContent is expected.

The Formula class allows for dynamic computation based on the data stored in other cells (an example of a formula could be ‘1 + REF(A1) * 2 - 9’). Additionally, the RangeExpression class deals with a range of cells within the spreadsheet and can evaluate expressions over these ranges.

The frontend part of our code communicates with the backend, utilizing the logic implemented within the methods of our classes, to update the state of the model. The ssdisplay directory contains multiple components that represent parts of the UI that the user interacts with.

SpreadsheetV (inside a_ssheet.tsx file) represents the user interface component that renders the spreadsheet and captures user interactions. This class interacts with the Spreadsheet class (from our backend portion) to reflect the changes made by the user, like updating cell values or changing the selected cell. It also listens to changes in the Spreadsheet to update the view accordingly.

The DataRepresentation class represents the spreadsheet data in various visual formats, such as charts. It retrieves data from the Spreadsheet through a method call and then processes it into a visual format.

Overall, the communication between these components is a clear example of the Model-View-Controller (MVC) architectural pattern, where Spreadsheet acts as the Model, SpreadsheetView is the View, and the user interactions that lead to data manipulations represent the Controller. The design ensures that each component has a specific role and communicates with other parts of the system in a controlled and predictable manner, enhancing the system's maintainability and scalability. This architecture separates the

concerns of data management and user interaction ensuring a scalable and maintainable system design.

Part 4 - Handling Dependencies Between Spreadsheet Cells

There are three different cases in which we needed to account for dependencies between cells: cell references, range expressions and formulas.

We use the updateCell function in a_ssheet.tsx to handle most of the dependency issues. The function handles changes made to the cells, including updates resulting from user input. When a cell is updated, updateCell checks if there are any other cells that depend on this cell (i.e., cells that reference it directly or through a formula). For these dependent cells, especially those with formulas, updateCell triggers a recalculation to reflect the new values.

Cell References

The mechanism followed for our cell references is that cells store references to other cells. When a cell's value is based on the value of another cell, it creates a dependency. Moreover, whenever the value of a referenced cell changes, all cells that depend on it need to be updated. This ensures that the data displayed is always consistent with the source cell. Within the CellReference class our getContent retrieves the value of the referenced cell from the spreadsheet and returns the cell value. Since this function is called every time we render our spreadsheet, inside of a_ssheet.tsx (within the return statement) the cell references will always be recomputed and updated accordingly.

Range Expressions

The RangeExpression class is crucial for tracking dependencies between cells because it needs to update its results when the content of any cell within its range changes. When rows or columns are added or deleted, the methods updateRangeExpressionsForRowChange and updateRangeExpressionsForColumnChange within the Spreadsheet Class are invoked, respectively. These methods iterate through all cells, identify those containing RangeExpression objects, and adjust their cell references to align with the new structure of the spreadsheet. This is achieved by the updateRangeExpression method within the Spreadsheet Class, which parses the range within the expression and recalibrates it based on the type of change—addition or deletion of rows or columns. Furthermore, the renumber method updates the cell indices post-modification to ensure consistency in the spreadsheet's addressing system. Within the RangeExpression class our getContent retrieves the value of the referenced cell from the spreadsheet and returns the cell value. Since this function is called every time we render our spreadsheet, inside of a_ssheet.tsx (within the return statement) the range expression will always be recomputed and updated accordingly.

Formulas

The formulas used within our spreadsheet can contain cell references and range expressions as well as other arithmetic operations. Therefore, we had to handle automatic recomputation every time referenced cells within formulas are updated. We did this using a helper function, reparse_ref located within a_ssheet.tsx.

reparse_ref is a helper function that aids in re-evaluating cell references, particularly useful in cases where cells that are referenced within the formula change. It takes in a cell number (e.g. "A1") and a model, and it re-parses the formulas that contain that string (that reference cell A1) within the given model. This function is called within updateCell, so that we reparse all formulas that reference the changed cell, everytime the user tries to update it. This ensures that every formula that is dependent on the modified cell will be immediately recomputed once the new value was entered.

Moreover, whenever we add/delete a row or a column, our formula will remain the same, it will keep referencing the same cell as it did before, and the value of the referenced cell will be updated. We do this by calling a reparse function, that re-parses all formulas in the spreadsheet, within the functions that handle adding/deleting rows and columns.
For example, if we have a formula "REF(A1) + 3", and cell A1 contains value 4, after pressing Enter the display value will be 7. If we add a row above cell A1, our display value for the cell that contains the formula will now be 3, as A1 will be empty.

Part 5 - Evolution Since Phase B

Our system and design have evolved in several ways since our planning in phase B. We had originally planned to follow the MVC design but, after structuring our model, once we began to build out the front end, we realized that the front end could act as our view and controller, utilizing functions within the components themselves. Additionally, with our implementation, we changed how one might initialize and represent a spreadsheet. We ultimately designed our spreadsheet to hold a 2D array of Cells; a Cell has its own row, column, and CellContent; and a CellContent is either a String, Number, Cell Reference, Range Expression, or Formula. The overall structure of our system stayed largely the same.

Additionally, at the start of the process, it was not immediately clear to our team how we would implement charts for our application. In our initial design, we stayed broad because we knew that the details would become more clear as we implemented the feature. For the chart feature, we worked backward, finding a react library that could visualize data, studying how it worked, and then figuring out how to get the correct data to the chart components. Our front-end design changed from our initial idea both because of small details and capabilities in react but also because of what we determined worked best for our system. For example, for chart generation, our initial design was based on other spreadsheet technologies we have seen before but we ended up using a modal to get the appropriate information from the user since it is not possible to select multiple cells at once in our spreadsheet. Through the chart modal, the user can quickly and easily view all of the requirements and customizations that are available for chart generation.

Moreover, initially, we thought the search bar's functionality would primarily be developed in the backend of our project. But as we began working on the frontend of our spreadsheet, we realized the backend implementation was unnecessary. We found a simpler method: executing the search entirely within the frontend code. Therefore, we decided against using a find function in our Spreadsheet class. Instead, we opted to develop the search functionality in our a_ssheet.tsx file. Thus, we are scanning the current state of the model (spreadsheet) to see if any cells contain the input value. When we find the desired string, we then highlight those specific cells.

During phase B of our project, we also discussed keeping track of all of the versions of our spreadsheet and also allowing the user to undo and redo certain changes. After completing the required features for our project, and started developing the undo/redo functionality, we decided that version history would require too much refactoring for our current implementation. Furthermore, as for the search bar feature, we believed that the undo/redo functionality would be primarily implemented in the backend portion of our project. However, as we explored the frontend of our spreadsheet more, we understood that these features do not need to be implemented within the backend. Thus, we handled undo/redo by keeping track of the previous states of the spreadsheet using an array, and updating the current state according to the buttons clicked by the user.

We also added new features where we felt they made sense and would not be too costly to include such as a help screen with simple documentation for all of the features that our spreadsheet has, an editable title for the main spreadsheet page, and a save feature that downloads the spreadsheet as a CSV file.

Part 6 - Reflection on Software Engineering Processes

We utilized an agile development process for our project which helped us to adapt designs quickly over the course of this short-term project timeline. Our team met frequently and discussed smaller details through messaging as needed over the course of each sprint. When developing our system we first decided to tackle the small pieces that would allow us to achieve the quickest prototype of our application; this included designing cells and how they would hold simple data such as strings and numbers and then designing a simple spreadsheet that knows what cells it holds and its dimensions. After developing a simplified model we were able to simultaneously build out more features from the back end like range expressions, cell references, and formulas, and begin designing the front end of the application and connecting it to our model. On the frontend we used bootstrap and font-awesome to amplify and unify the appearance of our application. For our chart creation functionality, we utilized the react-chartjs-2 library in order to help us create many types of charts quickly. Additionally, we incorporated the hot-formula-parser within our Formula class to manage parsing formulas in our system.

We all committed to one repo, made pull requests when necessary, and used a group chat to keep each other updated with what was changing in the system and any concerns we

had. One team member had trouble pushing to the repository and, as we tried to fix the issue, we communicated during our meetings and over text messages about updating the code with the appropriate changes. As we all worked together very closely and communicated frequently over the course of the project, we were lenient on code reviews within the GitHub site. We did, however, make pull requests when necessary to document large and impactful changes with accompanying descriptions and sometimes videos for future reference. As our design was constantly evolving and changed quickly in the beginning as we all worked on small modular parts of the larger system, we did not prioritize maintaining extensive tests. Tests, instead, became a larger focus in the later parts of our process as the system and its design were more solidified. For our spreadsheet class, the structure of a spreadsheet implicates several other classes by nature (Cell, CellContent, String, Number, etc) so the tests for this class also doubled as integration tests, ensuring that all of the parts worked together cohesively. On the front end, much of our code was tested visually throughout the process as we had a rapidly evolving UI. Towards the very end of our project, we were able to implement a few end-to-end (e2e) tests to wrap up our testing.

Part 7: Reflection on your development experience, problems encountered, lessons learned

Reflecting on the development experience of our recent project, several aspects contributed positively to our workflow and the quality of the final product. We met often, which made it easy to work together and keep everyone up to date. This ensured a collaborative environment where information was shared promptly, ensuring that all team members were aware of project updates and changes. Collaborating with each other during the development process helped us to address issues quickly and efficiently, reducing the likelihood of misunderstandings.

We pair-programmed a lot of the functionality in our system. Working in pairs allowed for immediate feedback and idea sharing and expedited the development process. We benefited from well-maintained documentation and clean code practices. This helped other team members to understand the codebase and eased the process of building code on top of the already implemented code. Moreover, our approach to version control was quite robust. We were careful about keeping track of changes in our code, and checking each other's work made sure our code was the best it could be. Code reviews among team members helped ensure code quality, consistency, and the sharing of knowledge across the team.

Despite these strengths, we encountered several challenges as well. At the start, it was hard to get everyone together at the same time because of our different schedules. We also had some trouble guessing how long tasks would take while discussing future sprints, which sometimes made our planning inaccurate. Not being able to accurately assign story points led to occasional underestimation or overestimation of the effort required for certain tasks. This meant that we needed a better understanding of the project's complexity.

Additionally, issues with GitHub like merge conflicts sometimes resulted in a loss of functionality. These conflicts were often a result of multiple team members working on the same files or from branches that had diverged significantly from the main codebase. However, we managed to overcome this issue by discussing extensively prior to creating pull requests or making significant modifications to code that other members were working on at the same time.

From these experiences, we learned that the initial design of a project is likely to evolve significantly. Therefore, being adaptable and open to change is crucial as new requirements and challenges arise. Furthermore, starting early on tasks was another key takeaway that we identified. It provided us with the buffer time needed to address unexpected complications.

Constant communication helped us in maintaining a unified vision and prevented the project from straying off course. Overall, the project was a learning curve that highlighted the importance of adaptability, proactive planning, and continuous communication within the team.

Part 8 - Installation Instructions

Prerequisites

- Node.js and npm: Ensure you have Node.js and npm installed. They are essential for managing the dependencies and running the project.
- Git: Required for cloning the repository.

Steps

1. Clone the Repository:
   - git clone https://github.com/neu-cs4530-fall2023/team507-project.git
2. Open the implementation folder located in the repository in Visual Studio Code.
3. Install Dependencies:
   - Open a new terminal and run “npm install” (this command installs all the necessary dependencies defined in the package.json file)
4. Run the Application:
   - Run “npm start” in order to launch our application (this command will compile the TypeScript code and start the React application, usually opening it in your default web browser)

User Guide

Starting the Application

After following the installation steps, the application should open in your web browser. If it doesn't, you can manually navigate to http://localhost:3000 (or the port specified in the terminal).

Getting Started

To begin using our spreadsheet application, you can enter numeric or string values directly into cells. Simply click on the cell you want to edit and start typing. To save the data in the cell, press Enter or click away from the cell.

Cell Operations (Basic Functionality)

You can perform various operations on cells:

- Adding Rows/Columns: To add a row or column, click on a cell, and a context menu will appear. You can add a row above, below, or a column to the left or right of the selected cell. You can also use the buttons at the top right and bottom left marked with a '+' sign to add a row or column.
- Deleting Rows/Columns: To delete a row or column, click on a cell within the row or column, and choose the delete option from the context menu.
- Entering Cell References: You can use cell references like “REF(A4)” within formulas to reference other cells in your calculations. Simply type that string in a cell and press “Enter”. The cell value will change to the appropriate result (You can still view the original reference associated with the cell in the text box located above the spreadsheet that begins with “fx:”).
- Entering Range Expressions: Range expressions like “SUM(A1:A4)” or “AVG(A1:A4)” can be used to perform calculations on a range of cells. Simply type that string in a cell and press “Enter”. The cell value will change to the appropriate result (You can still view the original expression associated with the cell in the text box located above the spreadsheet that begins with “fx:”).
- Entering Formulas: You can create formulas in cells to perform complex calculations. Enter your desired formula within a cell and press “Enter”. The cell value will change to the appropriate result (You can still view the original formula associated with the cell in the text box located above the spreadsheet that begins with “fx:”). An example of a formula would be: “1 + 2 * 3”.
- Clearing a Cell: You can also immediately clear a cell by clicking on it and choosing the clear cell option in the context menu.

Additional Features

- Undo/Redo: Our application provides undo and redo functionality to revert or reapply changes made to your spreadsheet. You can simply press on the undo button in order to undo the last change you made. Similarly, you can press on the redo button to go back to the change that was just reverted. The buttons are located in the navigation menu, right after the title of the spreadsheet (first undo, then redo). If you undo a change, and then modify the spreadsheet in any way, you will not be able to redo again.
- Search: Use the search bar to quickly find specific content within your spreadsheet. Click on the search bar (located above the spreadsheet, on the right) and type the string or value you would like to find. The cells that contain that string or value will be highlighted within the spreadsheet.

- Edit Spreadsheet Title: You can edit the title of your spreadsheet by clicking on the 'Edit' button in the navigation menu (in the top left of the screen) and changing the string inside the text box. When you are satisfied with your changes, simply click on the save button, in order to update the title.
- Creating Charts: Create charts by clicking on the chart button in the navigation menu (the first button on the top right corner of the screen). Enter the reference for the starting cell and ending cell you desire. Cells will be displayed as a graph of your choosing in which all data points are displayed in order (you can customize the graph using the buttons in the chart menu). The graphs will be displayed under the spreadsheet.
- Save as CSV: Save your spreadsheet as a CSV file for external use or sharing, by clicking the save button in the navigation menu (located in the top right corner of the screen).