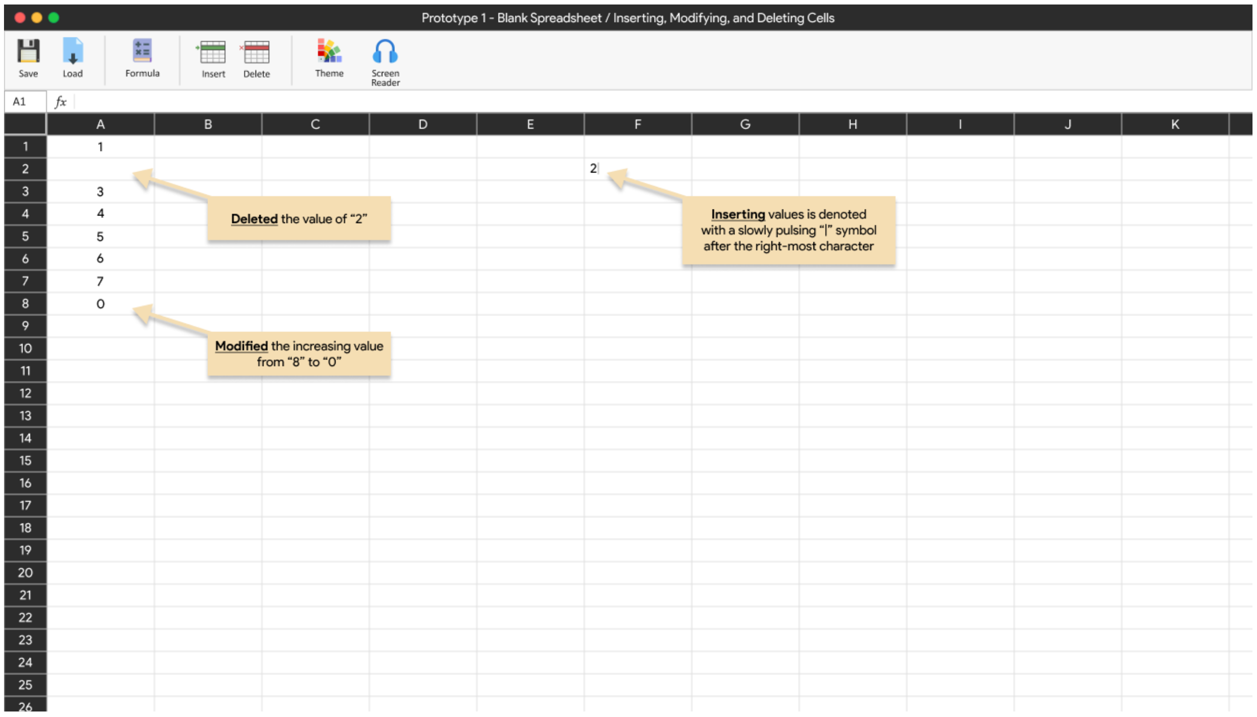
Mateus Aurelio, Evan Gurry, Palak Tyagi, A (Anthony) Vo
20 October 2023

User Stories and UI Sketches for Spreadsheet Application (Phase B)

Prototype	User Stories
Prototype 1 – Basic Spreadsheet and Inserting, Modifying, and Deleting Cells	User Stories 1 and 3 - Creating a Blank Spreadsheet - Inserting, Modifying, and Deleting Cells
Prototype 2 – Inserting and Deleting Rows	User Story 2 - Inserting and Deleting Rows and Columns
Prototype 3 – Referencing a Single and Multiple Cells	User Story 4 and 5 - Referencing a Single Other Cell - Referencing Multiple Other Cells
Prototype 4 – Using Formulas	User Story 6 - Using Formulas
Prototype 5 – Error Handling	User Story 7 - Error Handling Throughout Spreadsheet and Formulas
Prototype 6 – Saving and Loading Files	User Story 8 - Saving and Loading Files
Prototype 7 – Theming	User Story 9 - Dark Mode, High Contrast, and Theming
Prototype 8 – Speech Reader (Text-to-Speech)	User Story 10 - Text-to-Speech Interaction with Spreadsheet

Prototype 1 – Basic Spreadsheet and Inserting, Modifying, and Deleting Cells (WIP)


Prototype 1 - Blank Spreadsheet / Inserting, Modifying, and Deleting Cells
Save
Load
Formula
Insert
Delete
Theme
Screen Reader
A1
fx

Deleted the value of "2"

Inserting values is denoted with a slowly pulsing "|" symbol after the right-most character

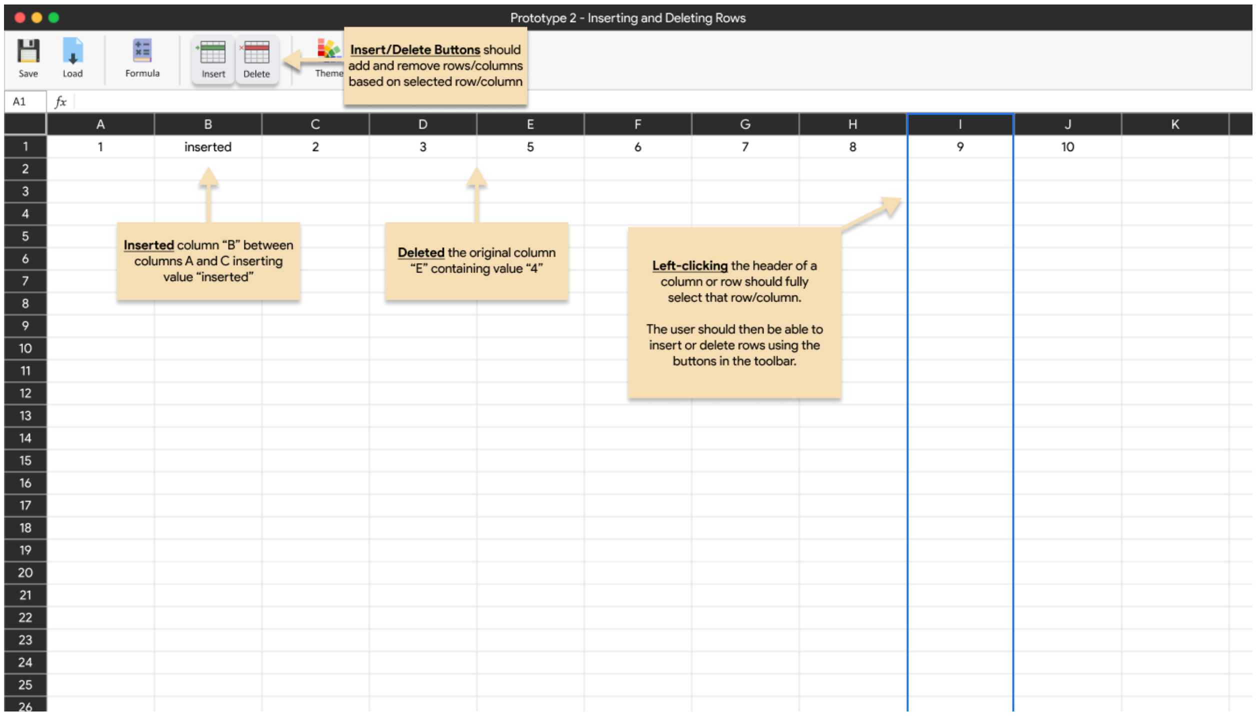
Modified the increasing value from "8" to "0"

User Story 1
Title: Creating a Blank Spreadsheet
Priority: High
Estimate: 13 points

As a user, I want to be able to create a blank spreadsheet so that I can start entering and storing data.

Acceptance Criteria:
- Initial View
    - Upon opening the application, a blank table of grid cells should be displayed.
- UI Elements
    - The spreadsheet application consists of the Options Pane at the top, a formula bar below it, and the cell data.
    - The Options Pane shows functionality for Saving and Loading files, invoking formulas, inserting and deleting rows, selecting an application theme, and starting a screen reader application.
- Cell Interaction
    - Users can click on cells to select them.
    - Users can click and drag to select/highlight multiple cells.
- Cell Naming and Structure
    - Cells are named A-Z and 1-99 (row and columns).

Prototype 2 – Inserting and Deleting Rows



Insert/Delete Buttons should add and remove rows/columns based on selected row/column

A	B	C	D	E	F	G	H	I	J	K
1	inserted	2	3	5	6	7	8	9	10	

Inserted column “B” between columns A and C inserting value “inserted”

Deleted the original column “E” containing value “4”

Left-clicking the header of a column or row should fully select that row/column.

The user should then be able to insert or delete rows using the buttons in the toolbar.

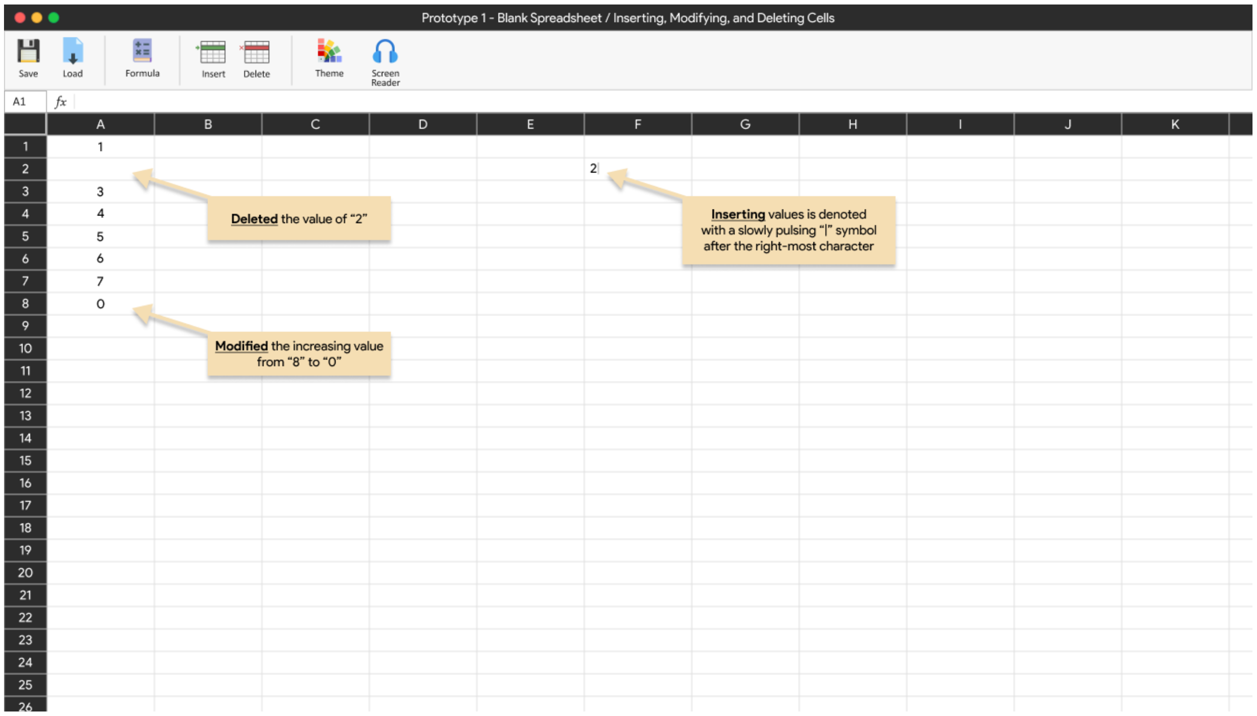
User Story 2
Title: Inserting and Deleting Rows and Columns
Priority: Medium
Estimate: 5 points

As a user, I want to be able to insert and delete rows and columns so that I can easily format my spreadsheet how I want it to look.

Acceptance Criteria:

- A user may left-click on a row or column header to select it.
- When selecting a row or column header, users can select the Insert or Delete buttons in the Options Pane (toolbar at top).
- When the user selects the “Insert” option from the toolbar:
  - If a row header was selected, a new row is inserted directly above the selected row. The index of the selected row and all subsequent rows is increased by 1.
  - If a column header was selected, a new column is inserted directly to the left of the selected column. The index of the selected column and all subsequent columns is increased by 1.
- When the user selects the “Delete” option from the toolbar:
  - If a row header was selected, the entire row is deleted, and the index of all subsequent rows is decreased by 1 (shifted left).
  - If a column header was selected, the entire column is deleted, and the index of all subsequent columns is decreased by 1 (shifted up).

Prototype 1 (Repeat) – Basic Spreadsheet and Inserting, Modifying, and Deleting Cells


Prototype 1 - Blank Spreadsheet / Inserting, Modifying, and Deleting Cells
Save
Load
Formula
Insert
Delete
Theme
Screen Reader
A1
fx

Deleted the value of “2”

Inserting values is denoted with a slowly pulsing “|” symbol after the right-most character

Modified the increasing value from “8” to “0”

User Story 3

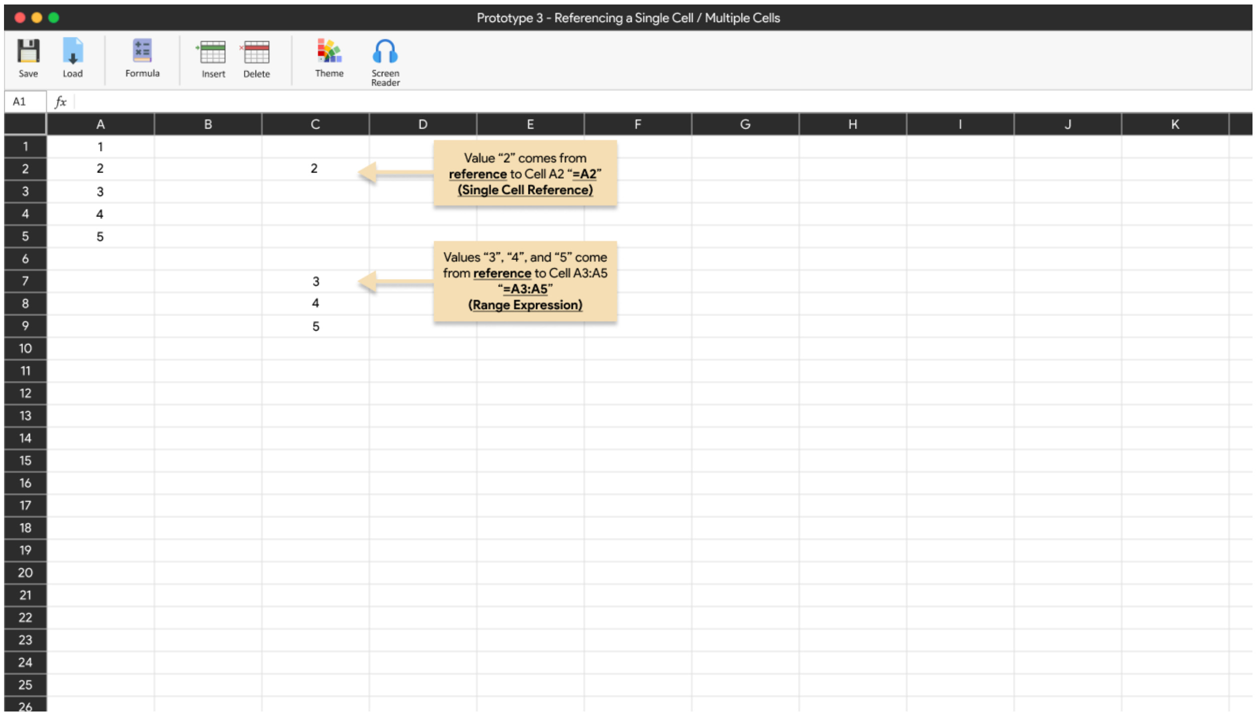
Title: Inserting, Modifying, and Deleting Cells
Priority: High
Estimate: 5 points

As a user, I want to be able to insert, modify, and delete the contents of a cell (including its formula if used) so that I can enter, edit, and store data in my spreadsheet.

Acceptance Criteria:

- Insertion
  - When a user selects a blank cell, an input cursor should appear and pulsate, informing and allowing the user to type values (or formulas) directly into the cell.
- Modification
  - When a user double-clicks a cell already containing data or a formula, the system should open the cell in edit mode, allowing the user to modify its contents as if it were a text box.
  - In cell edit mode, the user may move their cursor freely in the text box.
- Deletion
  - When a user selects one or more cells and presses the “Delete” key, all data (values and formulas) within the selected cells should be removed, leaving them blank.

Prototype 3 – Referencing a Single and Multiple Cells


Prototype 3 - Referencing a Single Cell / Multiple Cells
Save
Load
Formula
Insert
Delete
Theme
Screen Reader
A1
fx
Value "2" comes from **reference** to Cell A2 "**=A2**" **(Single Cell Reference)**
Values "3", "4", and "5" come from **reference** to Cell A3:A5 "**=A3:A5**" **(Range Expression)**

	A	B	C
1	1		
2	2		2
3	3		
4	4		
5	5		
6			
7			3
8			4
9			5

User Story 4
Title: Referencing a Single Other Cell
Priority: Low
Estimate: 3 points

As a data analyst, I want to be able to reference a single other cell and retrieve its value so that I can avoid manual data duplication and maintain data integrity.

Acceptance Criteria:

- Starting a Formula
    - When the user selects a cell, they should be able to invoke a formula by entering the equals sign ("=") keyword as the first character of the cell.
    - The formula bar above the cells in the spreadsheet application should display the formula as typed, while the value displayed in the cell should display the calculated (referenced) value.
- Referencing a Cell
    - After invoking a formula with "=", the user can type the address of another cell (such as "A1") to reference that cell.
    - Once the cell address is typed and entered, the cell should display the value of the referenced cell.
- Visual Feedback
    - When a user edits a cell that references another cell, that cell should be highlighted with a blue outline to show what is being referenced.

- Error Handling
    - If a user attempts to reference a non-existent cell (such as cell "ZZZ10000"), an error message "#REF!" should be displayed in the cell, indicating an invalid reference.
    - A cell should not be able to reference itself, or itself in a loop. For example, if cell A1 references cell B1, cell B1 cannot reference cell A1. If this happens, the message "#CYCLE!" should appear.

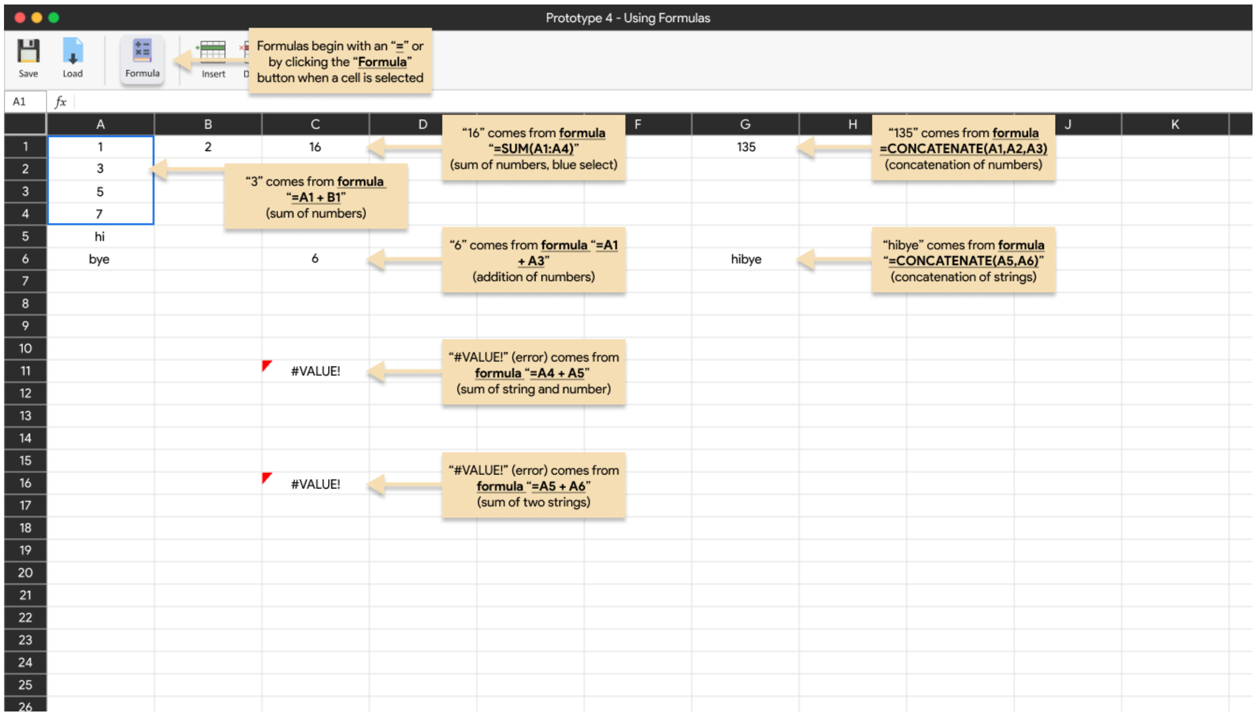
User Story 5
Title: Referencing Multiple Other Cells
Priority: Low
Estimate: 3 points

As a data analyst, I want to be able to use range expansion (potentially in combination with formulas) to reference multiple cells so that I can quickly reference a lot of data at once.

Acceptance Criteria:

- Starting a Formula
    - When the user selects a cell, they should be able to invoke a formula by entering the equals sign ("=") keyword as the first character of the cell.
    - The formula bar above the cells in the spreadsheet application should display the formula as typed, while the value displayed in the cell should display the calculated (referenced) value.
- Referencing a Range of Cells
    - After invoking a formula with "=", the user can type the addresses of a range of cells by specifying the top-left and bottom-right of the cell selection, split with a colon. For example, "=A1:B10" would reference cells A1 to B10.
    - Once the range expression is typed and entered, the cell should display the values of those cells starting from this cell.
        - This cell will be treated as the top-left cell in the range (in "=A1:B10", this would be "A1"). Afterwards, all other cells below or to the right of this cell are overwritten with the values received from the range expression.
    - If range expression is used in a formula, the result of the formula is outputted to this cell. The formula uses the range as input.
- Visual Feedback
    - When a user edits a cell that references a range expression, those cells should be highlighted with a blue outline to show what is being referenced.
- Error Handling
    - If a user attempts to reference non-existent cell(s)/ranges (such as "=A1:ZZZ10000"), an error message "#REF!" should be displayed in the cell, indicating an invalid reference.
    - A cell should not be able to reference itself, or itself in a loop. For example, if cell A1 references cell B1, cell B1 cannot reference cell A1. If this happens, the message "#CYCLE!" should appear.

Prototype 4 – Using Formulas



Formulas begin with an "=" or by clicking the "**Formula**" button when a cell is selected

"16" comes from **formula** "**=SUM(A1:A4)**" (sum of numbers, blue select)

"3" comes from **formula** "**=A1 + B1**" (sum of numbers)

"135" comes from **formula** **=CONCATENATE(A1,A2,A3)** (concatenation of numbers)

"6" comes from **formula** "**=A1 + A3**" (addition of numbers)

"hibye" comes from **formula** "**=CONCATENATE(A5,A6)**" (concatenation of strings)

"#VALUE!" (error) comes from **formula** "**=A4 + A5**" (sum of string and number)

"#VALUE!" (error) comes from **formula** "**=A5 + A6**" (sum of two strings)

User Story 6

Title: Using Formulas
Priority: Medium
Estimate: 8 points

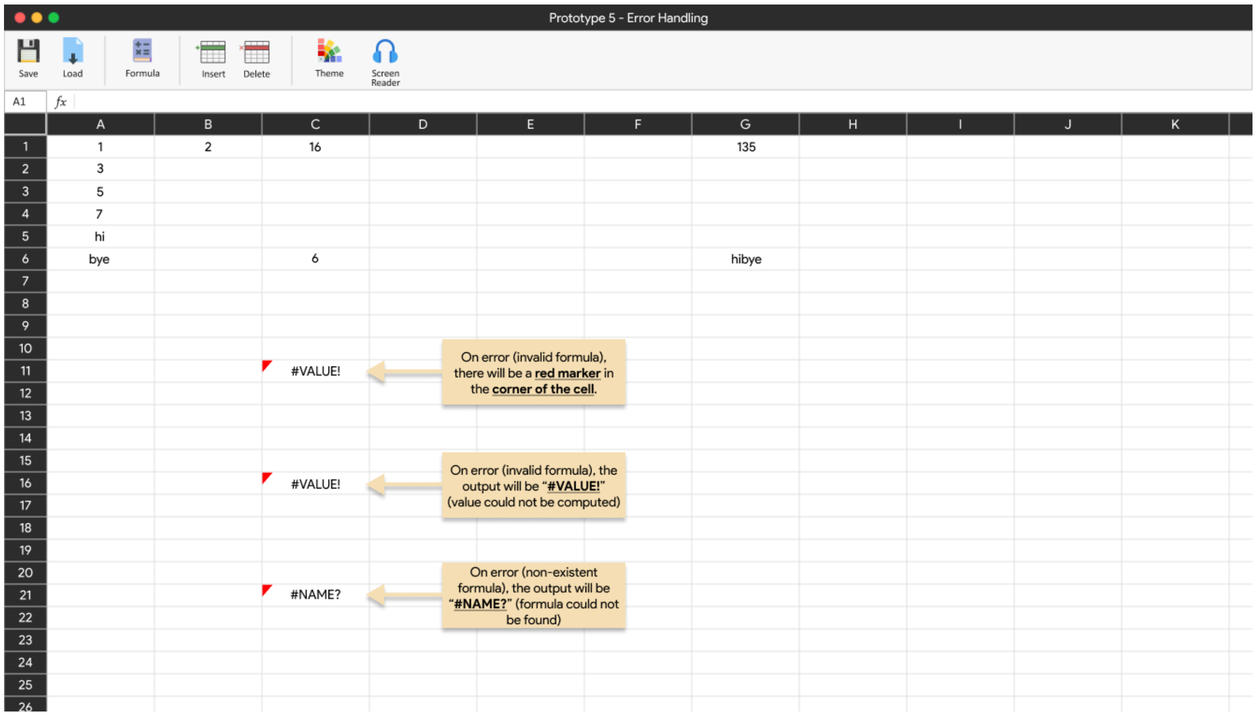
As a data analyst, I want to be able to use formulas that can calculate values using data stored in other cell(s) (this includes the concatenation and arithmetic operators) so that I can quickly get analyze the data I have.

Acceptance Criteria:

- Starting a Formula
  - When the user selects a cell, they should be able to invoke a formula by entering the equals sign ("=") keyword as the first character of the cell.
  - The formula bar above the cells in the spreadsheet application should display the formula as typed, while the value displayed in the cell should display the calculated (referenced) value.
- Basic Functions
  - The user should be able to perform basic arithmetic operations by using operations like SUM(), SUBTRACT(), PRODUCT(), and DIVIDE() on one to a range of cells (range expansion).
    - For example, "=SUM(A1:A4)" should sum all the values of cells from A1 to A4.

        - These operations can be performed using keyword notation, such as "+", "-", "*", and "/".
    - The user should be able to perform additional core spreadsheet functions typically used by most spreadsheet users like COUNT(), IF(), AVERAGE(), COUNTIF(), etc.
        - COUNT() – return the number of cells in the range
        - IF() – return a value if condition is true
        - AVERAGE() – calculate mean of values in range
        - COUNTIF() – return number of cells in range meeting condition
    - The user should be able to perform concatenation functions on cells.
        - CONCATENATE() – combine the value of two cells as long as they are both string datatypes.
- Editing Cells
    - The user may double-click to enter edit mode on a cell.
        - In edit mode, the original formula is shown, which can be modified.
    - The user may single-click a cell to select it and view it's formula in the formula bar above the cells.
        - The user may edit the formula from the formula bar directly.
- Displaying the Results
    - Once a formula is entered and edit mode has been exited, the cell value should display the output of the formula.
    - If a cell that a formula references is modified, then the formula value should update immediately.
- Error Handling
    - If a formula value results in an error (such as division by zero, or an invalid reference), an error message is displayed, such as: "#DIV/0!" or "REF!".
    - A cell should not be able to reference itself, or itself in a loop. For example, if cell A1 references cell B1, cell B1 cannot reference cell A1. If this happens, the message "#CYCLE!" should appear.

Prototype 5 – Error Handling



	A	B	C	D	E	F	G	H	I	J	K
1	1	2	16				135				
2	3										
3	5										
4	7										
5	hi										
6	bye		6				hibye				
7											
8											
9											
10											
11			#VALUE!								
12											
13											
14											
15											
16			#VALUE!								
17											
18											
19											
20											
21			#NAME?								
22											
23											
24											
25											
26											

On error (invalid formula), there will be a **red marker** in the **corner of the cell**.

On error (invalid formula), the output will be "**#VALUE!**" (value could not be computed)

On error (non-existent formula), the output will be "**#NAME?**" (formula could not be found)

User Story 7

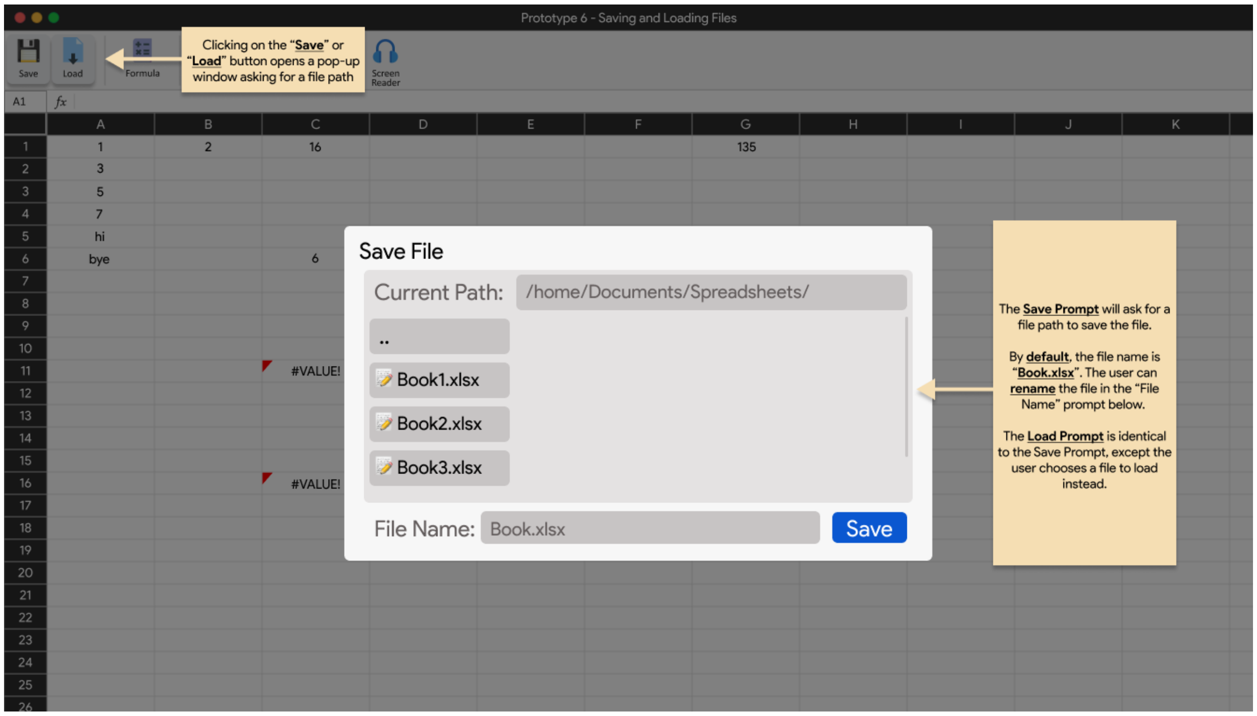
Title: Error Handling Throughout Spreadsheet and Formulas
Priority: High
Estimate: 8 points

As a user, I want to be informed when I make an illegal action (error handling) and tell me what I did wrong so that I can fix it without much hassle.

Acceptance Criteria:

- Displaying an Error
  - When an illegal formula is invoked, the program should tell the user that the formula is invalid by displaying an error message in the cell.
    - The error message will always return with "#" as the first character. For example, "#VALUE!", or "#REF!".
- Formula Errors
  - Given an incomplete range (ex. "=A1:A"), the output should be "#NAME?"
  - Given a non-existent formula (ex. "=BAD()"), the output should be "#NAME?"
- Reference Errors
  - Given an invalid reference (ex. "=ZZZ10000"), the output should be "#REF!"
- Cyclic Errors
  - Given a reference that would infinitely loop, the output should be "#CYCLE!"
- Division by Zero Error
  - Given a division by zero (ex. "=A1/0"), the output should be "#DIV/0!"

Prototype 6 – Saving and Loading Files



Clicking on the "**Save**" or "**Load**" button opens a pop-up window asking for a file path

Current Path: /home/Documents/Spreadsheets/

File Name: Book.xlsx

The **Save Prompt** will ask for a file path to save the file.

By **default**, the file name is "**Book.xlsx**". The user can **rename** the file in the "File Name" prompt below.

The **Load Prompt** is identical to the Save Prompt, except the user chooses a file to load instead.

	A	B	C	D	E	F	G	H	I	J	K
1	1	2	16				135				
2	3										
3	5										
4	7										
5	hi										
6	bye		6								
11			#VALUE!								
16			#VALUE!								

User Story 8
Title: Saving and Loading Files
Priority: High
Estimate: 5 points

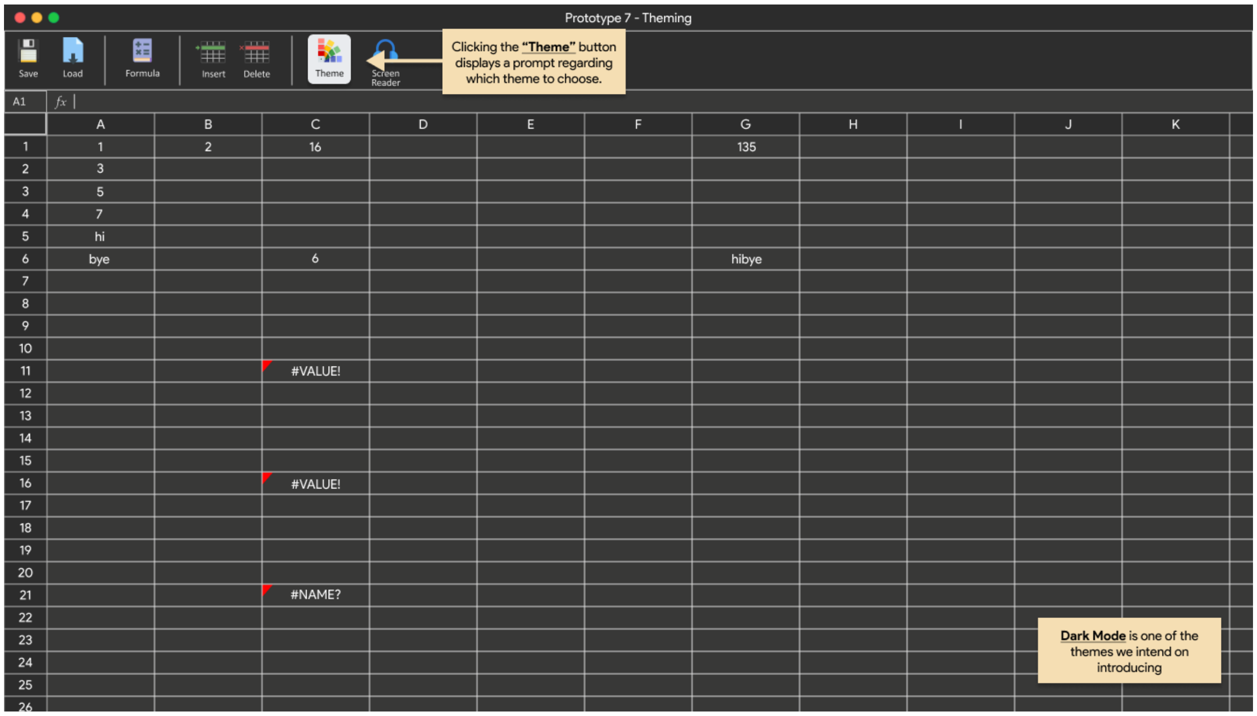
As a user, I want to be able to save and open my spreadsheet so that I can continue editing it later with all the data intact.

Acceptance Criteria:

- Saving a Spreadsheet
  - A spreadsheet file should be able to be stored locally using the "Save" button from the Options Pane, prompting the user for a path to store the file.
  - The Spreadsheet should be stored in a .csv file format (or an equivalent open-source format).
  - When saving a file, the state of the current spreadsheet will be exported into the file. This state stores cell data and formulas.
- Loading a Spreadsheet
  - A saved file (stored as a .csv or equivalent open-source format) should be able to be reloaded using the "Load" button from the Options Pane.
  - The program should verify the integrity of the file.
    - If the file is corrupt, it should inform the user with a pop-up in the middle of the screen.

    - Upon loading a valid file, the state of the current Spreadsheet application is overwritten by the file.
- File System Permissions
    - The spreadsheet program should have some file storage access to the device’s local file system.

Prototype 7 – Theming



Clicking the **"Theme"** button displays a prompt regarding which theme to choose.

	A	B	C	D	E	F	G	H	I	J	K
1	1	2	16				135				
2	3										
3	5										
4	7										
5	hi										
6	bye		6				hibye				
7											
8											
9											
10											
11			#VALUE!								
12											
13											
14											
15											
16			#VALUE!								
17											
18											
19											
20											
21			#NAME?								
22											
23											
24											
25											
26											

Dark Mode is one of the themes we intend on introducing

User Story 9
Title: Dark Mode, High Contrast, and Theming
Priority: Low
Estimate: 5 points

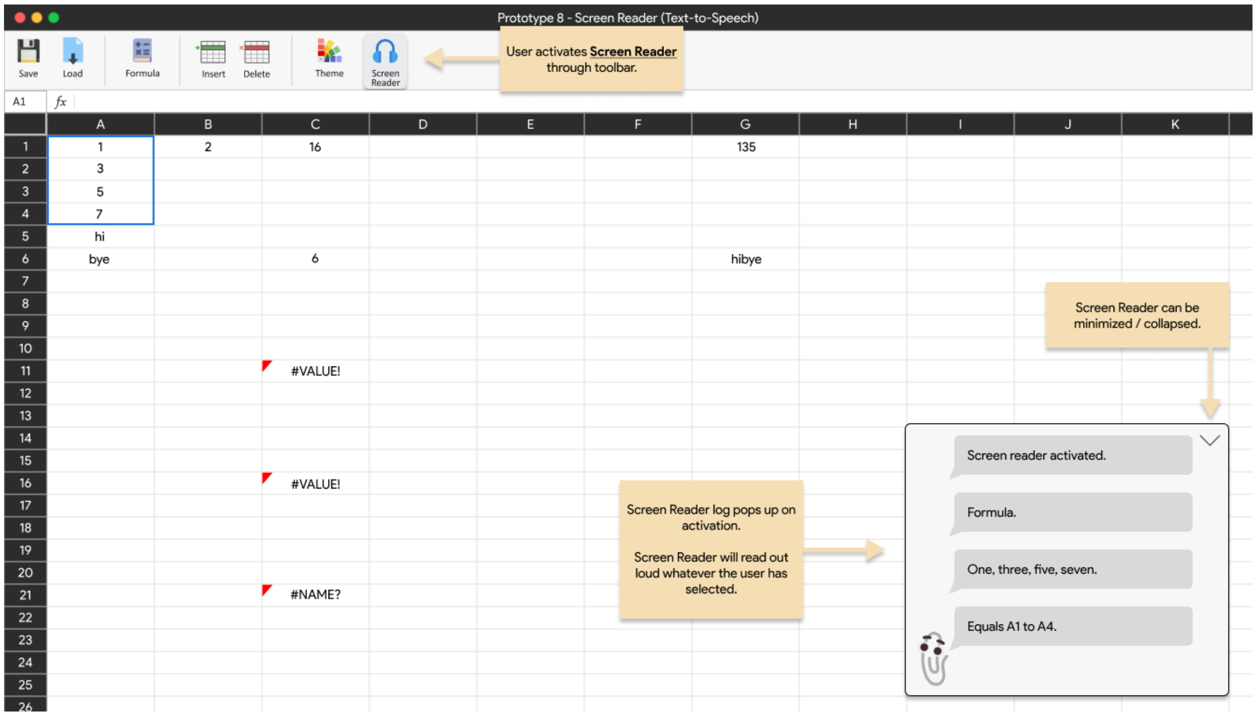
As a user, I want the flexibility to customize the visual appearance of my spreadsheet application through themes, including a dark mode and high contrast option, so that I can work comfortably in different lighting conditions and personalize my user experience.

Acceptance Criteria:

- Theme Selection
  - In the Options Pane, there is a "Theme" button.
  - On clicking the "Themes" button, a pop-up will display prompting the user for different types of theme options.
    - The options are: Dark Mode, High Contrast, "New Custom Theme", and any stored Custom Themes previously created.
- Dark Mode:
  - When the dark mode theme is selected, the application should switch to a dark color (dark gray).
  - Text and UI elements adjust to lighter colors to ensure visibility and contrast against the dark background.
- High Contrast Mode:

    - When the high contrast theme is selected, the application should maximize the amount of contrast between the foreground and background elements to make things more distinguishable.
    - Colors should be bold and distinct between the background, icons, and cells to ensure easier readability.
- Custom Theming:
    - When selecting “New Custom Theme”, users will be able to create a new custom theme.
        - A custom theme has a:
            - Name (string)
            - Background Color (string – HEX)
            - Cell Background Color (string – HEX)
            - Cell Text Color (string – HEX)
            - Cell Border Lines Color (string – HEX)
        - Custom themes can be saved and deleted from the Theme selection pop-up menu.
        - Custom themes are stored in the spreadsheet’s configuration.

Prototype 8 – Speech Reader (Text-to-Speech)



User activates **Screen Reader** through toolbar.

	A	B	C	D	E	F	G	H	I	J	K
1	1	2	16				135				
2	3										
3	5										
4	7										
5	hi										
6	bye		6				hibye				
11			#VALUE!								
16			#VALUE!								
21			#NAME?								

Screen Reader can be minimized / collapsed.

Screen Reader log pops up on activation.

Screen Reader will read out loud whatever the user has selected.

Screen reader activated.

Formula.

One, three, five, seven.

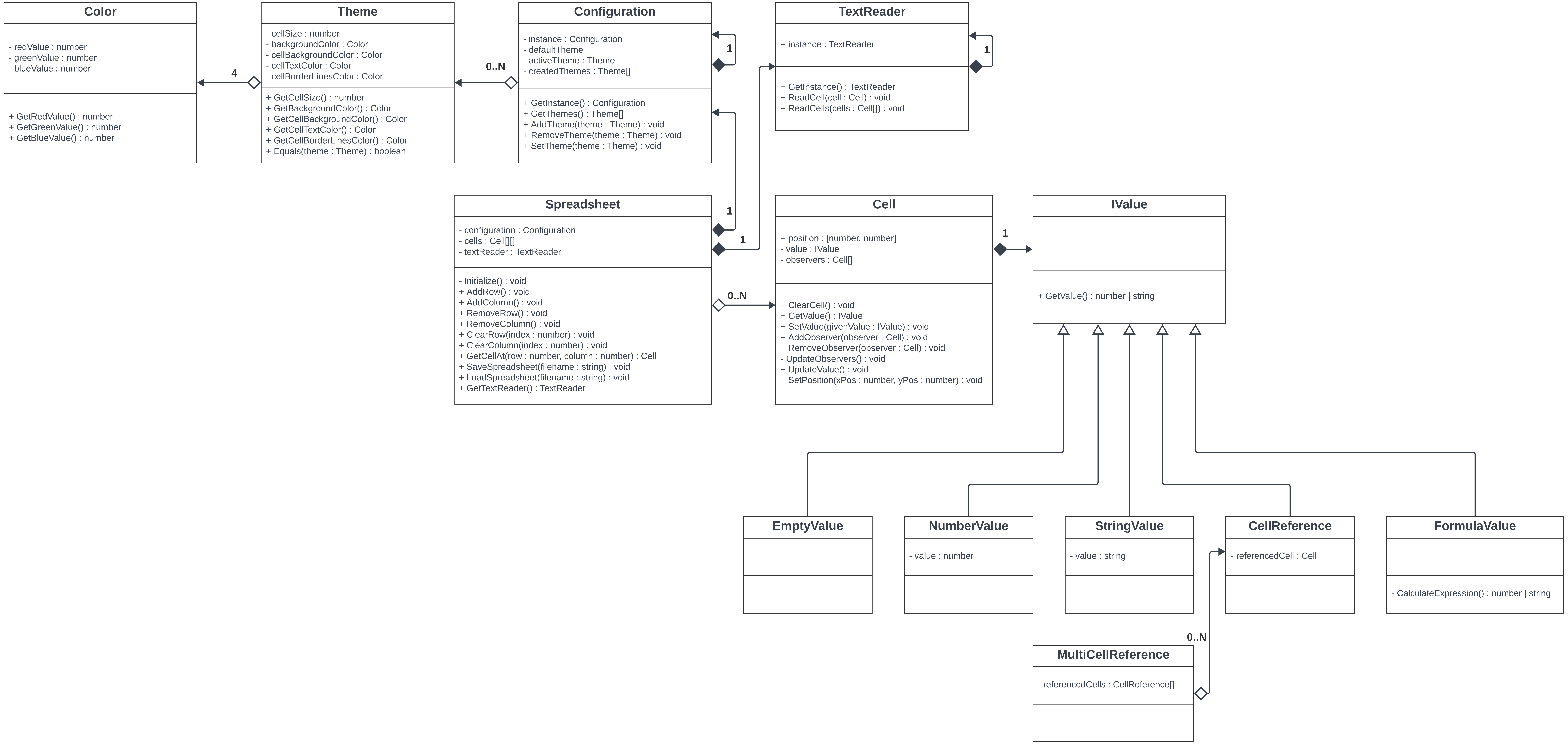
Equals A1 to A4.

User Story 10
Title: Text-to-Speech Interaction with Spreadsheet
Priority: Low
Estimate: 8 points

As a visual impaired user, I want to be able to use this application using a screen reader application so that I am still able to use the software despite being visually impaired.

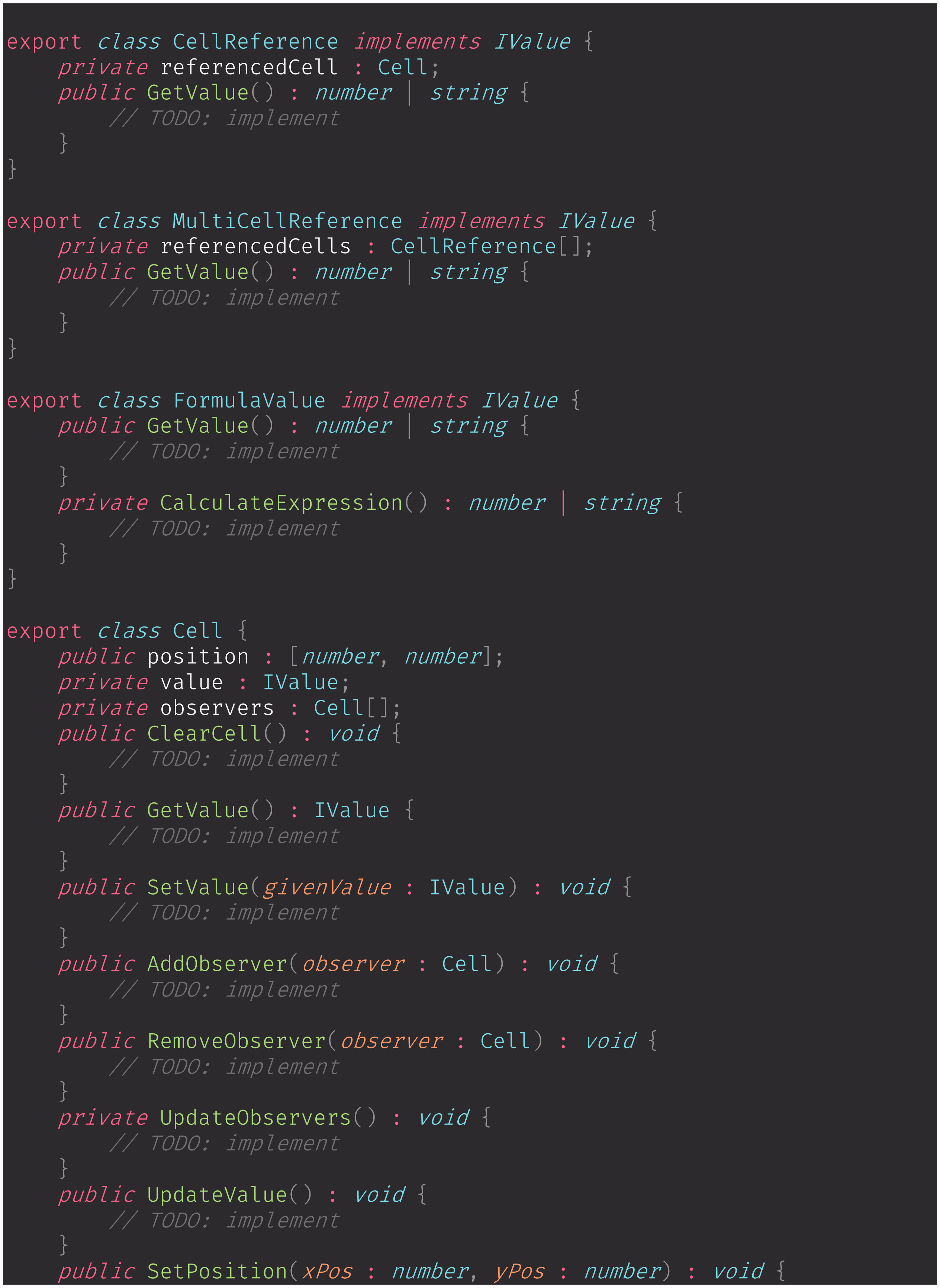
Acceptance Criteria:

- Activation
  - When the “Screen Reader” option in the Options Pane is toggled, a text-to-speech (TTS) tool should be invoked that reads out the contents of a cell to the user through their speakers.
- Basic Cell Integration
  - When clicking on a cell, the TTS will read the contents of the cell.
  - When double-clicking on a cell, if it is a formula, it will read out the contents of the formula.
  - When there is an error, the TTS will inform the user there is an error in the output.
- UI Integration
  - When a UI element is hovered, it should be read out loud.
  - For example, hovering over the “Save” button in the Options Pane will output “Save” via TTS.


Color
- redValue : number
- greenValue : number
- blueValue : number
+ GetRedValue() : number
+ GetGreenValue() : number
+ GetBlueValue() : number
4
Theme
- cellSize : number
- backgroundColor : Color
- cellBackgroundColor : Color
- cellTextColor : Color
- cellBorderLinesColor : Color
+ GetCellSize() : number
+ GetBackgroundColor() : Color
+ GetCellBackgroundColor() : Color
+ GetCellTextColor() : Color
+ GetCellBorderLinesColor() : Color
+ Equals(theme : Theme) : boolean
0..N
Configuration
- instance : Configuration
- defaultTheme
- activeTheme : Theme
- createdThemes : Theme[]
+ GetInstance() : Configuration
+ GetThemes() : Theme[]
+ AddTheme(theme : Theme) : void
+ RemoveTheme(theme : Theme) : void
+ SetTheme(theme : Theme) : void
1
TextReader
+ instance : TextReader
+ GetInstance() : TextReader
+ ReadCell(cell : Cell) : void
+ ReadCells(cells : Cell[]) : void
1
Spreadsheet
- configuration : Configuration
- cells : Cell[][]
- textReader : TextReader
- Initialize() : void
+ AddRow() : void
+ AddColumn() : void
+ RemoveRow() : void
+ RemoveColumn() : void
+ ClearRow(index : number) : void
+ ClearColumn(index : number) : void
+ GetCellAt(row : number, column : number) : Cell
+ SaveSpreadsheet(filename : string) : void
+ LoadSpreadsheet(filename : string) : void
+ GetTextReader() : TextReader
1
1
0..N
Cell
+ position : [number, number]
- value : IValue
- observers : Cell[]
+ ClearCell() : void
+ GetValue() : IValue
+ SetValue(givenValue : IValue) : void
+ AddObserver(observer : Cell) : void
+ RemoveObserver(observer : Cell) : void
- UpdateObservers() : void
+ UpdateValue() : void
+ SetPosition(xPos : number, yPos : number) : void
1
IValue
+ GetValue() : number | string
EmptyValue
NumberValue
- value : number
StringValue
- value : string
CellReference
- referencedCell : Cell
FormulaValue
- CalculateExpression() : number | string
MultiCellReference
- referencedCells : CellReference[]
0..N

TypeScript Classes and Interfaces

```
export class Color {
    private redValue : number;
    private greenValue : number;
    private blueValue : number;
    public GetRedValue() : number {
        // TODO: implement
    }
    public GetGreenValue() : number {
        // TODO: implement
    }
    public GetBlueValue() : number {
        // TODO: implement
    }
}

export class Theme {
    private cellSize : number;
    private backgroundColor : Color;
    private cellBackgroundColor : Color;
    private cellTextColor : Color;
    private cellBorderLinesColor : Color;
    public GetCellSize() : number {
        // TODO: implement
    }
    public GetBackgroundColor() : Color {
        // TODO: implement
    }
    public GetCellBackgroundColor() : Color {
        // TODO: implement
    }
    public GetCellTextColor() : Color {
        // TODO: implement
    }
    public GetCellBorderLinesColor() : Color {
        // TODO: implement
    }
    public Equals(theme : Theme) : boolean {
        // TODO: implement
    }
}

export class Configuration {
    private instance : Configuration;
    private defaultTheme;
    private activeTheme : Theme;
    private createdThemes : Theme[];
    public GetInstance() : Configuration {
        // TODO: implement
    }
    public GetThemes() : Theme[] {
        // TODO: implement
    }
    public AddTheme(theme : Theme) : void {
        // TODO: implement
    }
    public RemoveTheme(theme : Theme) : void {
        // TODO: implement
    }
    public SetTheme(theme : Theme) : void {
        // TODO: implement
    }
}

export class TextReader {
    public instance : TextReader;
    public GetInstance() : TextReader {
        // TODO: implement
    }
    public ReadCell(cell : Cell) : void {
        // TODO: implement
    }
    public ReadCells(cells : Cell[]) : void {
        // TODO: implement
    }
}

export interface IValue {
    public GetValue() : number | string;
}

export class EmptyValue implements IValue {
    public GetValue() : number | string {
        // TODO: implement
    }
}

export class NumberValue implements IValue {
    private value : number;
    public GetValue() : number | string {
        // TODO: implement
    }
}

export class StringValue implements IValue {
    private value : string;
    public GetValue() : number | string {
        // TODO: implement
    }
}
```



```
export class CellReference implements IValue {
    private referencedCell : Cell;
    public GetValue() : number | string {
        // TODO: implement
    }
}

export class MultiCellReference implements IValue {
    private referencedCells : CellReference[];
    public GetValue() : number | string {
        // TODO: implement
    }
}

export class FormulaValue implements IValue {
    public GetValue() : number | string {
        // TODO: implement
    }
    private CalculateExpression() : number | string {
        // TODO: implement
    }
}

export class Cell {
    public position : [number, number];
    private value : IValue;
    private observers : Cell[];
    public ClearCell() : void {
        // TODO: implement
    }
    public GetValue() : IValue {
        // TODO: implement
    }
    public SetValue(givenValue : IValue) : void {
        // TODO: implement
    }
    public AddObserver(observer : Cell) : void {
        // TODO: implement
    }
    public RemoveObserver(observer : Cell) : void {
        // TODO: implement
    }
    private UpdateObservers() : void {
        // TODO: implement
    }
    public UpdateValue() : void {
        // TODO: implement
    }
    public SetPosition(xPos : number, yPos : number) : void {
        // TODO: implement
    }
}

export class Spreadsheet {
    private configuration : Configuration;
    private cells : Cell[][];
    private textReader : TextReader;
    private Initialize() : void {
        // TODO: implement
    }
    public AddRow() : void {
        // TODO: implement
    }
    public AddColumn() : void {
        // TODO: implement
    }
    public RemoveRow() : void {
        // TODO: implement
    }
    public RemoveColumn() : void {
        // TODO: implement
    }
    public ClearRow(index : number) : void {
        // TODO: implement
    }
    public ClearColumn(index : number) : void {
        // TODO: implement
    }
    public GetCellAt(row : number, column : number) : Cell {
        // TODO: implement
    }
    public SaveSpreadsheet(filename : string) : void {
        // TODO: implement
    }
    public LoadSpreadsheet(filename : string) : void {
        // TODO: implement
    }
    public GetTextReader() : TextReader {
        // TODO: implement
    }
}
```