Statement of Work

#	Item	Priority	Value
Minimum Viable Product			
1	Implementation of a user interface displaying a grid of cells with columns labeled in shortlex order over the alphabet of capital letters and rows numbered.	VERY HIGH	Required for MVP - must be able to see sheet and locate data in the sheet
2	Functionality allowing users to select, edit, and delete the contents of a cell.	VERY HIGH	Required for MVP - Main functionality for all users
3	Support for various data types (texts, numbers, formulas).	VERY HIGH	Required for MVP -
4	Support for Main mathematical functions: addition, subtraction, multiplication, division, and summation across rows and columns.	HIGH	Required for MVP but not initial prototypes -
5	Support for sheets up to 100 by 100 (10000 cells)	VERY HIGH	Required for MVP - This is the bare minimum spreadsheet size we want to support
6	Support for multiple users editing sheets in parallel.	VERY HIGH	Required for MVP - Users need to be able to edit in parallel
7	Server responds within 10 seconds.	HIGH	Critical for ensuring a responsive and seamless user experience; delays can significantly disrupt user workflows.
8	Ability to register new users. (REST API)	VERY HIGH	Essential for expanding user base and controlling access; allows the system to secure user data and customize experiences.
9	Ability to retrieve publisher data. (REST API)	VERY HIGH	Vital for administrative and user operations; enables users and admins to manage or interact with data publishers effectively.
10	Ability to create new spreadsheets. (REST API)	VERY	Fundamental for user

#	Item	Priority	Value
		HIGH	productivity; central to the core functionalities of the application.
11	Ability to retrieve available spreadsheets. (REST API)	VERY HIGH	Crucial for user navigation and access; supports efficient data management and retrieval by users.
12	Ability to delete spreadsheets. (REST API)	VERY HIGH	Important for maintaining data hygiene and managing storage; provides users control over their data.
13	Ability to get updates for subscriptions. (REST API)	VERY HIGH	Necessary for maintaining data integrity and timeliness; ensures users receive the latest data changes they are subscribed to.
14	Ability to get updates for published sheets. (REST API)	VERY HIGH	Key for collaborative environments; allows publishers to track changes and interactions with their shared data.
15	Ability to publish updates to sheets. (REST API)	VERY HIGH	Essential for real-time collaboration; supports dynamic data sharing and ensures all users view the most current data.
16	Ability to subscribe to updates for sheets. (REST API)	VERY HIGH	Critical for users who rely on up-to-date information; facilitates proactive notifications and updates to enhance user engagement and data accuracy.
Desirables			
17	Confirmation to the user when updating the server.	LOW	Reassures the user that their input
18	Copy/Cut and paste for single cells.	HIGH	Not required for MVP, but is a standard and important convenience feature
19	Main cell formatting options: font styles, text alignment, background colors, and borders.	LOW	Nice to be able to change formatting, but does not affect functionality

#	Item	Priority	Value
20	Error handling mechanisms for invalid inputs, formula errors, and other issues.	HIGH	Notify the user about the format error
21	Resizable rows and columns.	HIGH	Non MVP function but important to work in the sheet.
22	Select the whole row/column.	LOW	Formatting can be applied to an entire row/column, and new cells in that column will get the formatting
23	Select the whole region.	MEDIUM	With left click and drag can select as many cells as present
24	Copy/Cut and paste for regions.	MEDIUM	Moving whole regions of data is very useful for organizing a sheet
25	Shift-click and control-click to add or remove from the selected region.	MEDIUM	Sometimes it is useful to copy-paste multiple sections of data at once.
26 27	Insert a new row or column above/below or left/right of the current row or column.	MEDIUM	Users may need to add more data to the middle of a table or other already filled cells.
28	See which sheets are being edited by another user.	LOW	It is nice to know if you are working on the same sheet as someone else.
29	Support for larger sheets up to 1 million cells	MID	Users who have large amounts of data need sheets that can support that data
30	Sheets auto-save when edited	VERY HIGH	Required for MVP - Users need their changes to be saved, and we want to save after every change
31	Resolve conflicts with multiple users editing the same cells	HIGH	Collaborators may try to edit the same cells at the same time, and this conflict should be automatically resolved
Bonus Features - VERY LOW			
32	Delete row/column.		
33	Highlighting the entire row when being accessed.		

#	Item	Priority	Value
34	Smart copying of formulas where pasting updates relative cell numbers.		
35	Dragging a cell to copy/extend (for example dragging on the number 1 to get numbers in order).		
36	Support for multiple file formats: CSV, JSON, etc.		
37	Advanced formatting options: conditional formatting, cell merging, etc.		
38	Importing and exporting data from external sources: Excel files, Google Sheets, and other databases.		
39	Version history for documents and the ability to roll back to a previous version.		
40	Diff and conflict system so multiple people can check out the same sheet.		
41	Support for pulling data from external databases in formulas.		
42	Charting and graphing capabilities to visualize data.		
43	Different number formatting options (currency/time/number of decimal places).		
44	Support for concurrent Google Sheets style collaboration.		
45	Iterative calculation for recursive formulas.		
46	Formulas are recalculated when their dependency cells are updated.		
47	Mutually recursive formulas for not crashing the program.		

MVP User Stories and Use Cases

1. **Implementation of a user interface displaying a grid of cells with columns labeled in shortlex order over the alphabet of capital letters and rows numbered.**
    - **User Story:**
        - As a user, I want to view a spreadsheet interface so that I can interact with data organized in a grid of cells.
    - **Use Case:**
        - **Name:** Display Spreadsheet Grid
        - **Actor:** User
        - **Preconditions:** User is logged into the system.
        - **Postconditions:** The spreadsheet grid is displayed to the user.
        - **Main Flow:**

1. User opens the application.
2. System displays a grid of cells with columns labeled in shortlex order over the alphabet of capital letters and rows numbered.

- **Alternative Flows:** None.

2. **Functionality allowing users to select, edit, and delete the contents of a cell.**
    - **User Story:**
        - As a user, I want to select, edit, and delete the contents of a cell to easily modify data within the spreadsheet.
    - **Use Case:**
        - **Name:** Manipulate Cell Contents
        - **Actor:** User
        - **Preconditions:** Spreadsheet grid is displayed.
        - **Postconditions:** Cell contents are updated as per user actions.
        - **Main Flow:**
            1. User clicks on a cell to select it.
            2. User enters edit mode either through a keyboard shortcut or by double-clicking the cell.
            3. User modifies the content and presses Enter to update or presses Delete to clear the content.
        - **Alternative Flows:**
            1. User presses Escape to cancel editing.
3. **Support for various data types (texts, numbers, formulas).**
    - **User Story:**
        - As a user, I want the spreadsheet to support different data types including text, numbers, and formulas so that I can use it for diverse data management tasks.
    - **Use Case:**
        - **Name:** Enter Various Data Types
        - **Actor:** User
        - **Preconditions:** User has selected a cell.
        - **Postconditions:** The cell correctly stores and displays the data type entered.
        - **Main Flow:**
            1. User selects a cell.
            2. User inputs a value (text, number, or formula).
            3. System processes and displays the entered data according to its type.
4. **Support for Main mathematical functions: addition, subtraction, multiplication, division, and summation across rows and columns.**
    - **User Story:**
        - As a user, I want to perform Main mathematical functions on the data to manage and analyze information effectively.
    - **Use Case:**
        - **Name:** Perform Mathematical Calculations

        - **Actor:** User
        - **Preconditions:** User has entered numerical data in relevant cells.
        - **Postconditions:** The calculated result is displayed in the specified cell.
        - **Main Flow:**
            1. User inputs a formula in a cell (e.g., =A1+A2).
            2. System calculates and displays the result in the cell.
        - **Alternative Flows:**
            1. User inputs an incorrect formula syntax and receives an error prompt.
5. **Support for sheets up to 100 by 100 (10000 cells).**
    - **User Story:**
        - As a user, I want the spreadsheet application to support sheets up to 100 by 100 cells so that I can manage large datasets without needing to segment data unnecessarily.
    - **Use Case:**
        - **Name:** Handle Large Spreadsheets
        - **Actor:** User
        - **Preconditions:** User begins to create or open a spreadsheet.
        - **Postconditions:** The spreadsheet can correctly display and manage up to 10,000 cells without performance issues.
        - **Main Flow:**
            1. User creates or opens a spreadsheet.
            2. User adds data up to the application's maximum limit of 100 rows by 100 columns.
            3. The system ensures that the spreadsheet operates smoothly, handling data entry, updates, and calculations even at full capacity.
        - **Alternative Flows:**
            1. If user attempts to exceed the cell limit, the system warns and prevents further data addition beyond the limit.
6. **Support for multiple users editing sheets in parallel.**
    - **User Story:** As a user, I want to be able to work on spreadsheets simultaneously with other users so that we can collaborate efficiently without waiting for others to finish their edits.
    - **Use Case:**
        - **Name:** Concurrent Spreadsheet Editing
        - **Actor:** Users
        - **Preconditions:** Multiple users have access to the same spreadsheet.
        - **Postconditions:** Users can make changes in parallel without data conflicts.
        - **Main Flow:**
            1. User attempts to change a cell.
            2. Server notifies the client that they were updating an out of date version of the cell and rejects the change.

3. Client displays the updated cell to the user and allows them to make changes.
4. Server accepts the changes on the up-to-date cell.

- **Alternative Flow:**
  1. User attempts to change a cell.
  2. Server notifies the client that they were updating an out of date version of the cell and rejects the change.
  3. Client displays the updated cell to the user and the user accepts the updated cell.

7. **Server can respond within 10 seconds.**
   - **User Story:** As a user, I need the server to respond within a defined time frame of 10 seconds to ensure smooth operation and good user experience.
   - **Use Case:**
     - **Name:** Ensure Server Response Efficiency
     - **Actor:** Server
     - **Preconditions:** A request is made to the server.
     - **Postconditions:** Server responds within an acceptable time frame of 10 seconds.
     - **Main Flow:**
       1. User performs an action that requires server interaction.
       2. The server processes the request and returns a response within the defined time frame.
     - **Performance Requirement:** The server must respond within 10 seconds for all user actions.
8. **Ability to register new users. (REST API)**
   - **User Story:** As an administrator, I want to register new users to ensure they can securely access the system.
   - **Use Case:**
     - **Name:** Register New Users
     - **Actor:** Server
     - **Preconditions:** Administrator is authenticated on the server.
     - **Postconditions:** A new user is registered and can access the system.
     - **Main Flow:**
       1. Administrator inputs user registration details.
       2. Server processes and registers the new user.
       3. Server confirms registration success.
9. **Ability to retrieve publisher data. (REST API)**
   - **User Story:** As a user, I want to retrieve a list of all publishers to manage or interact with their data.
   - **Use Case:**
     - **Name:** Retrieve Publisher Data
     - **Actor:** Server
     - **Preconditions:** User is authenticated and has necessary permissions.
     - **Postconditions:** A list of publishers is displayed to the user.

        - **Main Flow:**
            1. User requests a list of publishers.
            2. Server retrieves and provides the list.

10. Ability to create new spreadsheets. (REST API)

- **User Story:** As a user, I want to create new spreadsheets to manage various data sets.
- **Use Case:**
    - **Name:** Create New Spreadsheets
    - **Actor:** Server
    - **Preconditions:** User is authenticated and has permission to create spreadsheets.
    - **Postconditions:** A new spreadsheet is available for use.
    - **Main Flow:**
        1. User sends a request to create a new spreadsheet.
        2. Server processes the request and creates the spreadsheet.
        3. Server confirms the creation to the user.

11. Ability to retrieve available spreadsheets. (REST API)

- **User Story:** As a user, I need to view all spreadsheets available to me for selection and further action.
- **Use Case:**
    - **Name:** Retrieve Available Spreadsheets
    - **Actor:** Server
    - **Preconditions:** User is logged in and has the necessary rights.
    - **Postconditions:** User sees a list of all available spreadsheets.
    - **Main Flow:**
        1. User requests to see available spreadsheets.
        2. Server retrieves and displays the list of spreadsheets.

12. Ability to delete spreadsheets. (REST API)

- **User Story:** As a user, I want to delete unnecessary spreadsheets to maintain an organized workplace.
- **Use Case:**
    - **Name:** Delete Spreadsheets
    - **Actor:** Server
    - **Preconditions:** User selects a spreadsheet and has deletion rights.
    - **Postconditions:** The spreadsheet is deleted from the server.
    - **Main Flow:**
        1. User requests to delete a specific spreadsheet.
        2. Server processes the request and deletes the spreadsheet.
        3. Server confirms deletion to the user.

13. Ability to get updates for subscriptions. (REST API)

- **User Story:** As a user, I want to receive updates for the sheets I am subscribed to, so I can stay informed of any changes.
- **Use Case:**
    - **Name:** Retrieve Updates for Subscribed Sheets

        - **Actor:** Server
        - **Preconditions:** User is authenticated and has subscriptions.
        - **Postconditions:** User receives the latest updates for the subscribed sheets.
        - **Main Flow:**
            1. User requests updates for a subscribed sheet.
            2. Server checks for new updates since the last sync.
            3. Server sends the updates to the user, if available.
        - **Alternative Flows:**
            1. If no updates are available, the user is informed that the sheets are up-to-date.

14. Ability to get updates for published sheets. (REST API)

- **User Story:** As a publisher, I want to receive feedback or updates made by others to the sheets I have published, to monitor engagement and contributions.
- **Use Case:**
    - **Name:** Retrieve Updates for Published Sheets
    - **Actor:** Server
    - **Preconditions:** User is authenticated and has published sheets.
    - **Postconditions:** Publisher receives updates or changes made by other users.
    - **Main Flow:**
        1. Publisher requests updates for a published sheet.
        2. Server checks for new interactions or updates since the last check.
        3. Server sends the relevant updates to the publisher.
    - **Alternative Flows:**
        1. If no updates are available, the publisher is informed that there have been no changes.

15. Ability to publish updates to sheets. (REST API)

- **User Story:** As a user, I want to publish updates to the sheets I own, so that subscribers can see the latest changes.
- **Use Case:**
    - **Name:** Publish Updates to Sheets
    - **Actor:** Server
    - **Preconditions:** User is authenticated and owns or has edit rights to the sheet.
    - **Postconditions:** Updates are published and accessible to subscribers.
    - **Main Flow:**
        1. User finalizes changes and sends a request to publish updates.
        2. Server processes the request and updates the sheet.
        3. Server notifies subscribers of the new updates.
    - **Alternative Flows:**
        1. If the update cannot be processed, the user is notified of the failure.

16. Ability to subscribe to updates for sheets. (REST API)

- **User Story:** As a user, I want to subscribe to updates for specific sheets, to automatically receive notifications of any changes.
- **Use Case:**
  - **Name:** Subscribe to Sheet Updates
  - **Actor:** Server
  - **Preconditions:** User is authenticated and has identified sheets of interest.
  - **Postconditions:** User is subscribed to updates for the chosen sheet.
  - **Main Flow:**
    1. User selects sheets to subscribe to and sends a subscription request.
    2. Server processes the subscription and adds the user to the update list.
    3. Server confirms subscription success to the user.
  - **Alternative Flows:**
    1. If the subscription request fails, the user is notified of the issue.

Desirables User Stories and Use Cases

17. Confirmation to User When Updating the Server

- **User Story:** I want to receive confirmation when data is updated on the server, So that I am sure my changes have been saved.
- **Use Case:**
  - **Name:** Confirm Server Updates
  - **Actor:** User
  - **Preconditions:** User has made changes to the data in the spreadsheet.
  - **Postconditions:** User receives confirmation that the changes have been saved on the server.
  - **Main Flow:**
    1. User makes changes to the spreadsheet.
    2. User initiates the save process.
    3. System updates the server with the changes.
    4. System displays a confirmation message to the user that the data has been successfully saved.
  - **Alternative Flows:**
    1. Connection Issues: If there is a server connectivity issue, the system notifies the user of the failure to save changes.

18. Copy/Cut and paste for single cells.

- **User Story:** As a user, I want the ability to copy/cut and paste data from single cells, so that I can easily move or duplicate content within the spreadsheet.
- **Use Case:**
  - **Name:** Manage Single Cell Data
  - **Actor:** User

        - **Preconditions:** User selects a cell containing data.
        - **Postconditions:** Data from the selected cell is copied/cut and pasted successfully.
        - **Main Flow:**
            1. User selects a cell and chooses to copy or cut the content.
            2. User selects another cell and pastes the content.
            3. System updates the cell with the new data.

        - **Alternative Flows:**
            1. Invalid Target: If the paste operation is attempted on an invalid cell, the system alerts the user and cancels the paste operation.

19. **Main Cell Formatting Options: font styles, text alignment, background colors, and borders.**
    - **User Story:** As a user, I want to access Main cell formatting options such as font styles, text alignment, background colors, and borders, so that I can enhance the readability and presentation of my spreadsheet.
    - **Use Case:**
        - **Name:** Format Spreadsheet Cells
        - **Actor:** User
        - **Preconditions:** User selects one or more cells for formatting..
        - **Postconditions:** The selected cells are formatted according to the user's preferences.
        - **Main Flow:**
            1. User selects cells and accesses the formatting options.
            2. User applies desired formatting settings like font style, text alignment, background color, and borders.
            3. System applies the formatting to the selected cells.
        - **Alternative Flows:**
            1. Formatting Errors: If there are conflicts or errors in formatting settings, the system alerts the user.
20. **Error Handling Mechanisms: invalid inputs, formula errors, and other issues.**
    - **User Story:** As a user, I want the system to handle errors gracefully, so that I can correct invalid inputs and formula errors efficiently.
    - **Use Case:**
        - **Name:** Handle Spreadsheet Errors
        - **Actor:** User
        - **Preconditions:** User interacts with the spreadsheet.
        - **Postconditions:** User is informed of any errors and can take corrective actions.
        - **Main Flow:**
            1. User enters data or formulas that could result in errors.
            2. System checks for errors and identifies any issues.
            3. System notifies the user of the error with a descriptive message.

        - **Alternative Flows:**
            1. Recovery Options: System offers suggestions or tools to help the user correct the error.

21. Resizable Rows and Columns

- **User Story:** As a user, I want to resize rows and columns, so that I can customize the layout according to my data needs.
- **Use Case:**
    - **Name:** Resize Spreadsheet Rows and Columns
    - **Actor:** User
    - **Preconditions:** User accesses a spreadsheet with adjustable layout features.
    - **Postconditions:** Rows and columns are resized according to user specifications.
    - **Main Flow:**
        1. User selects a row or column.
        2. User adjusts the size of the selected row or column.
        3. System updates the layout to reflect the new size.

22. Select Whole Row/Column

- **User Story:** As a user, I want to select whole rows or columns, so that I can quickly apply operations to entire rows or columns at once.
- **Use Case:**
    - **Name:** Select Entire Rows or Columns
    - **Actor:** User
    - **Preconditions:** User is viewing a spreadsheet.
    - **Postconditions:** Entire row or column is selected.
    - **Main Flow:**
        1. User clicks on the row or column header.
        2. System highlights and selects the entire row or column.

23. Select Whole Region

- **User Story:** As a user, I want to select a whole region of cells, so that I can efficiently manage larger sections of data.
- **Use Case:**
    - **Name:** Select Large Data Regions
    - **Actor:** User
    - **Preconditions:** User is viewing a spreadsheet.
    - **Postconditions:** A region of cells is selected.
    - **Main Flow:**
        1. User drags the mouse over a group of cells to define the region.
        2. System highlights and selects the defined region.

24. Copy/Cut and paste for regions.

- **User Story:**
    - As a user, I want to copy, cut and paste for selected regions.
- **Use Case:**
    - **Name:** Copy, cut and paste region

        - **Actor:** User
        - **Preconditions:** Spreadsheet must be created.
        - **Postconditions:** A region which has either been copied, cut, or pasted into.
        - **Main Flow:**
            1. User selects a region in the spreadsheet.
            2. They use control-c to copy the highlighted region, control-v to paste into the highlighted region, or ‘delete’ to delete the highlighted region.
        - **Alternative Flow:**
            1. User cancels the edit by pressing ‘Escape.’

25. Shift-click and control-click to add or remove from the selected region.

- **User Story:** As a user, I want to Shift-click and control-click to add or remove from the selected region.
- **Use Case:**
    - **Name:** Add or remove from selected region
    - **Actor:** User
    - **Preconditions:** Spreadsheet must be created
    - **Postconditions:** The spreadsheet will either have been added to or deleted from.
    - **Main Flow:**
        1. User presses the 'shift’ button and ‘clicks’ on the region they wish to add to.
        2. Users press the ‘control’ button and ‘clicks’ on the region they wish to delete.
    - **Alternative Flow:**
        1. User presses ‘Escape’ to quit editing

26. Insert a new row above/below the current row.

- **User Story:** As a user, I want to add a new row above/below the current row.
- **Use Case:**
    - **Name:** Add row
    - **Actor:** User
    - **Preconditions:** Spreadsheet must be created
    - **Postconditions:** A new row will be added to the spreadsheet
    - **Main Flow:**
        1. User highlights the row they wish to add above or below.
        2. User is prompted whether they wish to add a row above or below.
        3. Row is added.
    - **Alternative Flow:**
    - User presses cancels by pressing ‘Escape’

27. Add a new column right/left of the current column.

- **User Story:** As a user, I want to be able to add a new column to the right/left of the current column.
- **Use Case:**

        - **Name:** Add column
        - **Actor:** User
        - **Preconditions:** Spreadsheet must be created
        - Postconditions: A new column is added to the spreadsheet
        - **Main Flow:**
            1. User highlights the column they wish to add to the right or left of the current column.
            2. User is prompted whether they want to add a column to the right/left of the highlighted column.
            3. Column is added.
        - **Alternative Flow:**
            1. User cancels edit by pressing 'Escape.'

28. See which sheets are being edited by another user.

- **User Story:** As a user, I want to be able to see which sheets are being edited by another user.
- **Use case:**
    - **Name:** Sheets being edited
    - **Actor:** User
    - **Preconditions:** User must be logged in.
    - **Postconditions:** User will learn which sheets another user is editing.
    - **Main Flow:**
        1. User examines edit lo.
    - **Alternative Flow:**
        1. None.

29. Support for larger sheets up to 1 million cells.

- **User Story:** As a user, I want to be able to create a spreadsheet with up to 1 million cells.
- **Use Case:**
    - **Name:** Max sheet size
    - **Actor:** User
    - **Preconditions:** User must be logged in.
    - **Postconditions:** User will have a spreadsheet with 1 million or less cells.
    - **Main Flow:**
        1. User requests to change the size of the spreadsheet.
        2. The user gives measurements for the new spreadsheet size.
        3. The spreadsheet refreshes to the new size.
    - **Alternative Flow:**
        1. Error, the system is giving a number that is not in the proper range.

30. Sheets auto-save when edited.

- **User Story:** As a user, I want to be able to auto-save my spreadsheet.
- **Use Case:**
    - **Name:** Auto-save
    - **Actor:** User

        - **Preconditions:** Spreadsheet must be made.
        - **Postconditions:** Spreadsheet will autosave to users
        - **Main Flow:**
            1. The system sends the data to the server.
            2. The server confirms that the spreadsheet has been saved.
            3. User received confirmation of the save.
        - **Alternative Flow:**
            1. Connection error, unable to save, file name already exists.

31. Resolve conflicts with multiple users editing the same cells.

- **User Story:** As a user, I want the system to automatically handle conflicts when multiple users edit the same cells, so that changes are synchronized without overwriting important data, ensuring a smooth collaborative experience.
- **Use Case:**
    - **Name:** Manage Concurrent Cell Edits
    - **Actor:** Server
    - **Preconditions:**
        1. Multiple users have access to and are editing the same spreadsheet.
        2. Users are authenticated and authorized to make edits to the spreadsheet.
    - **Postconditions:**
        1. Changes by all users are incorporated without data loss.
        2. Users are informed of any conflicts and the resolutions applied.
    - **Main Flow:**
        1. User A and User B simultaneously submit changes to the same cell in a spreadsheet.
        2. The server receives both requests and detects a conflict due to simultaneous edits.
        3. The server applies a conflict resolution strategy, such as Last Writer Wins or merging changes based on certain rules.
        4. The server updates the cell with the resolved content.
        5. The server broadcasts the updated cell information to all active users viewing that spreadsheet.
        6. Users receive a notification about the update and any conflict resolution that occurred.
    - **Alternative Flows:**
        1. **Conflict Notification:** If the conflict resolution results in one user's input being overridden, the affected user receives a detailed notification explaining why their changes were not applied and suggesting further actions.
        2. **Manual Resolution Option:** For critical conflicts, users are prompted to manually resolve the conflict by choosing which version to keep.

Summary: Minimum Viable Product

Minimum set of features required for Main functionality:

1. Implementation of a user interface displaying a grid of cells with columns labeled in shortlex order over the alphabet of capital letters and rows numbered.
2. Functionality allowing users to select, edit, and delete the contents of a cell.
3. Support for various data types (texts, numbers, formulas).
4. Support for Main mathematical functions: addition, subtraction, multiplication, division, and summation across rows and columns.
5. Support for sheets up to 100 by 100 (10000 cells)
6. Support for multiple users editing sheets in parallel.
7. Server responds within 10 seconds.
8. Ability to register new users. (REST API)
9. Ability to retrieve publisher data. (REST API)
10. Ability to create new spreadsheets. (REST API)
11. Ability to retrieve available spreadsheets. (REST API)
12. Ability to delete spreadsheets. (REST API)
13. Ability to get updates for subscriptions. (REST API)
14. Ability to get updates for published sheets. (REST API)
15. Ability to publish updates to sheets. (REST API)
16. Ability to subscribe to updates for sheets. (REST API)

Summary: Desirables

More complex set of features for enhancing project functionality (*Don't include features we are uncertain about to avoid losing points*):

17. Confirmation to the user when updating the server.
18. Copy/Cut and paste for single cells.
19. Main cell formatting options: font styles, text alignment, background colors, and borders.
20. Error handling mechanisms for invalid inputs, formula errors, and other issues.
21. Resizable rows and columns.
22. Select the whole row/column.
23. Select the whole region.
24. Copy/Cut and paste for regions.
25. Shift-click and control-click to add or remove from the selected region.
26. Insert a new row or column above/below of the current row or column.
27. Insert a new row or column left/right of the current row or column.

28. See which sheets are being edited by another user.
29. Support for larger sheets up to 1 million cells
30. Sheets auto-save when edited.
31. Resolve conflicts with multiple users editing the same cells

Summary: Bonus Features

Additional features for extra credit:

32. Delete row/column.
33. Highlighting the entire row when being accessed.
34. Smart copying of formulas where pasting updates relative cell numbers.
35. Dragging a cell to copy/extend (for example dragging on the number 1 to get numbers in order).
36. Support for multiple file formats: CSV, JSON, etc.
37. Advanced formatting options: conditional formatting, cell merging, etc.
38. Importing and exporting data from external sources: Excel files, Google Sheets, and other databases.
39. Version history for documents and the ability to roll back to a previous version.
40. Diff and conflict system so multiple people can check out the same sheet.
41. Support for pulling data from external databases in formulas.
42. Charting and graphing capabilities to visualize data.
43. Different number formatting options (currency/time/number of decimal places).
44. Support for concurrent Google Sheets style collaboration.
45. Iterative calculation for recursive formulas.
46. Formulas are recalculated when their dependency cells are updated.
47. Mutually recursive formulas for not crashing the program.

TA Name: Shubh
TA GitHub: shubhdesai1611
TA Email: desai.shu@northeastern.edu
Breakout Room: 4

Components

- GUI (JavaFX)
    - Sheet Display
        - Grid layout for displaying cell data with resizable columns and rows.
        - Highlight active cell and selection range for clarity.
        - Real-time updates to cell values and formulas.
    - Menu for Choosing Sheet
        - Dropdown or sidebar menu to list available sheets.
        - Options to create new sheets or delete existing ones.
        - Functionality to rename sheets directly from the menu.
    - Menus for Choosing Formatting Options
        - Toolbar with buttons for font style, size, bold, italic, underline.
        - Color picker for text and background color.
        - Tools for cell alignment (left, center, right), text wrapping, and borders.
- Client
    - Data Handling
        - Manage user sessions and interactions with the spreadsheet.
        - Send requests to the server for saving, retrieving, and updating sheets.
    - User Authentication
        - Implement user login, registration, and session management.
        - Ensure secure transmission of credentials and use tokens for session validation.
    - Error Handling
        - Provide user-friendly error messages and logging for debugging.
        - Implement retry mechanisms for failed requests due to network issues.
- Server

- API Endpoints
  - ‘POST /sheets’ to create a new sheet.
  - ‘GET /sheets’ to retrieve a list of sheets.
  - ‘PUT /sheets/{id}’ to update an existing sheet.
  - ‘DELETE /sheets/{id}’ to remove a sheet.
  - Additional endpoints for specific actions like updating cell values or batch operations.
- Data Storage
  - Use a database to store sheet data, user information, and session logs.
- Business Logic
  - Implement the core logic for how users interact with sheets.
  - Ensure consistency and integrity of data through transactional controls and validations.
  - Optimization strategies for handling large sheets and concurrent users.

Use Case

A use-case is a methodology used in system analysis to identify, clarify, and organize system requirements

© 2023 Released under the CC BY-SA license

Use Case

An *actor* is something that can act – a person, a system, or an organization

A *scenario* is a specific sequence of actions and interactions between actors (where at least one actor is a system)

A *use case* is a collection of related scenarios – successful and failing ones

Useful for *clients* as well as for *developers*

© 2023 Released under the CC BY-SA license

Actors and Goals

What are the *boundaries* of the system? Is it the software, hardware and software, also the user, or a whole organization?

Who are the *primary actors* – i.e., the stakeholders?

What are the *goals* of these actors?

Describe how the system fulfills these goals (including all exceptions)

© 2023 Released under the CC BY-SA license

Example

Use-Case Name: Place Order

Actor: Customer

Goal: The customer wants to purchase selected items and have them delivered.

Preconditions: The customer has items in their shopping cart.

Postconditions: The inventory is updated to reflect the purchased items.

Basic Flow:

- Start Order: The customer clicks on the "Checkout" button in the shopping cart.
- Provide Shipping Information: Display a form to enter shipping information.
- Enter Payment Details: The customer enters their payment details

© 2023 Released under the CC BY-SA license

Example

- Review Order: The system displays a summary of the order; The customer reviews all the details.
- Confirm Order: The customer clicks the "Place Order" button to confirm the details.
- Process Payment: The system processes the payment; The system updates the inventory
- Display Confirmation: The system displays an order confirmation page with an order number

Alternate Flows:

- Invalid Payment: If the payment is declined, the customer is prompted to enter a new payment
- Edit Order: At the review stage, the customer chooses to modify the order

Exception Flows:

- System Error During Payment: the customer is informed of the error and asked to retry later.

© 2023 Released under the CC BY-SA license

User Stories

User Stories are requirements from a user's point of view

© 2023 Released under the CC BY-SA license

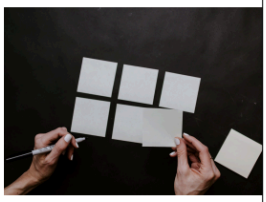
User Stories

Specifying what should happen, for whom, and why

As a *role* I can *capability*, so that *receive benefit*

Conditions of Satisfaction:

Given *interaction with software, state of environment*,
I expect *behavior and side effects*



© 2023 Released under the CC BY-SA license

Who is a User?

A *user* is an end-user of the software, but could be any stakeholder:

- As an *operator*, I want to look at the network traffic so that I can improve customer response time.
- As a *manager*, I want to see the distribution of customer response times so I can improve customer satisfaction

© 2023 Released under the CC BY-SA license

User Stories in Context

A *user story* puts end users at the center of the conversation.

- User stories are grouped into *epics* (a set of related user stories)
- User stories are amplified by *acceptance criteria* (conditions used to confirm when a story is completed)

These stories use non-technical language to provide context for the development team and their efforts.

After reading a user story, the team knows why they are building, what they're building, and what value it creates. Stories are not software requirements.

https://www.atlassian.com/agile/project-management/user-stories

© 2023 Released under the CC BY-SA license

Acceptance Criteria

User story: *As a passenger, I want several available drivers to be displayed so that I can choose the most suitable option for me.*

Possible acceptance criteria for the app:

- …show drivers online within last 20 minutes without an ongoing ride.
- …show the 5 drivers closest to the user.
- …lets browse profiles of these drivers, including photos and reviews.

© 2023 Released under the CC BY-SA license

Writing User Stories: INVEST

Independent
Negotiable
Valuable
Estimatable
Small
Testable

As a <*role*>
I can <*capability*>,
so that <*receive benefit*>

© 2023 Released under the CC BY-SA license

User Story Example

User stories tell what users want to do, and why

As a College Administrator, I want a database to keep track of students, the courses they have taken, and the grades they received in those courses, so that I can advise them on their studies.

© 2023 Released under the CC BY-SA license

Satisfaction Conditions

Satisfaction Conditions list capabilities the user expects, in the user's words

My database should allow me to:
- Add a new student
- Add a new student with same name as an existing one
- Retrieve a student's transcript
- Delete a student
- Add a new grade for an existing student
- Get grade a student got in a course

© 2023 Released under the CC BY-SA license

User Stories may be Prioritized

Priorities:

- *Essential* means the project is useless without it
- *Desirable* means the project is less usable without it, but is still usable
- *Extension* a desirable story that may not be achievable within project scope

Minimum Viable Product (MVP) is the smallest, simplest system one can build to meet satisfy all essential conditions

© 2023 Released under the CC BY-SA license

Requirements: Which to pick?

There are four knobs you can adjust when negotiating requirements:

- *Project scope*
- *Project duration*
- *Project quality*
- *Project cost*

Usually cost is most constrained: you have limited budget and headcount

Determining feasibility is the subject of much software engineering research

© 2023 Released under the CC BY-SA license

Use Cases vs User Stories

user stories

- high level
- good for quickly capturing ideas and desired features

use cases:

- low level
- describes system responses to user actions

© 2023 Released under the CC BY-SA license