5/14/24

Statement of Work - Group 7

Eduardo Ruiz De Garay, Rishav Sarma, Laurence Ehrhardt, Hunter Brodie, Kristoffer Amerman

Item	Priority	Value
Minimum Viable Product		
Login interface	High	Required for MVP – Username, password form with submit button
Sheet management interface	High	Required for MVP - Create-new-sheet button, list of user's sheets (to edit), delete sheet
Sheet editing interface	High	Required for MVP – Cells, save button
Auth flow	High	Required for MVP – Find user/authenticate, persist current user session, redirect to sheet management UI on login
Sheets table	High	Required for MVP – keep track of master sheets
Users table	High	Required for MVP – keep track of users for auth
Subscriptions table	High	Required for MVP – keep track of sheet subscriptions
Updates table	High	Required for MVP – keep track of sheet updates
`register` endpoint communicates with server upon entry	High	Required for MVP – Handles auth, input error
`getSheets` endpoint returns list of sheets for given publisher	High	Required for MVP – Handle incoming `getSheets` request to send sheets associated with this publisher to the client
`createSheet` endpoint	High	Required for MVP – Handle incoming `createSheet` request and create an empty sheet in the DB
`deleteSheet` endpoint	High	Required for MVP – Handle incoming `deleteSheet` request and remove sheet from the DB
`getUpdatesForSubscription` endpoint returns a list of updates for a given publisher	High	Required for MVP – Handle incoming `getUpdatesForSubscription` request to send updates for the sheet to the client
`getUpdatesForPublisher` endpoint returns a list of updates for a given publisher	High	Required for MVP – Handle incoming `getUpdatesForPublisher` request to send update requests for the sheet to the client that owns this sheet
`updatePublished` endpoint updates a published sheet	High	Required for MVP – Handle incoming `updatePublished` request and update sheet to reflect changes
`updateSubscription`	High	Required for MVP – Handle incoming

endpoint updates a subscribed sheet		`updateSubscription` request and update a subscribed sheet
Setup Docker image	High	Required for MVP – Setup mongoDB dockerfile with react/express stack and pull from github with deploy key
JSON encoding and decoding	High	Required for MVP – Encodes the cells info into JSON and decodes into spreadsheet
Add functionality for entering a formula into a cell	Medium	Required for MVP – User should be able to enter formulas and perform computations with cell values
Desirable Features		
Add interactivity for individual cell	High	User should be able to enter data
Header interface	Medium	Logout button, display current user
Create Test Suites for `createSheet` endpoint	Medium	Test for adding new sheet to users list of sheets, test adding multiple new sheets, test for DB update
Create Test Suites for `deleteSheet` endpoint	Medium	Test for deleting sheet from users list of sheets, test to not delete when there are 0 sheets, test for DB update
Create Test Suites for `register` endpoint	Medium	Test for fetching user information with correct UN/PW, test to error with incorrect UN/PW, test to create new user with correct UN/PW
Create Test Suites for `getPublishers` endpoint	Medium	Test for fetching list of all known publishers, test to return 0 publishers with no publishers in server
Create Test Suites for `getSheets` endpoint	Medium	Test for fetching list of all known sheets for given publisher, test to return 0 sheets if given publisher has none, test to return nothing if given publisher doesn't exist
Create Test Suites for `getUpdatesForPublisher` endpoint	Medium	Test for return correct payload for all updates after given id for a published sheet, test to return no updates if id is latest, test to return no updates if incorrect input
Create Test Suites for `getUpdatesForSubscription` endpoint	Medium	Test for return correct payload for all updates after given id for a subscribed sheet, test to return no updates if id is latest, test to return no updates if incorrect input
Create Test Suites for `updatePublished` endpoint	Medium	Test to see updates reflected in DB server, test to see if no update shows with empty payload
Create Test Suites for `updateSubscription` endpoint	Medium	Test to see updates reflected in DB server, test to see if no update shows with empty payload
Bonus Features		
Dynamic size input	Low	Add option to increase size of sheet on creation or during editing
Setup persistent login	Low	Cookie for user sessions

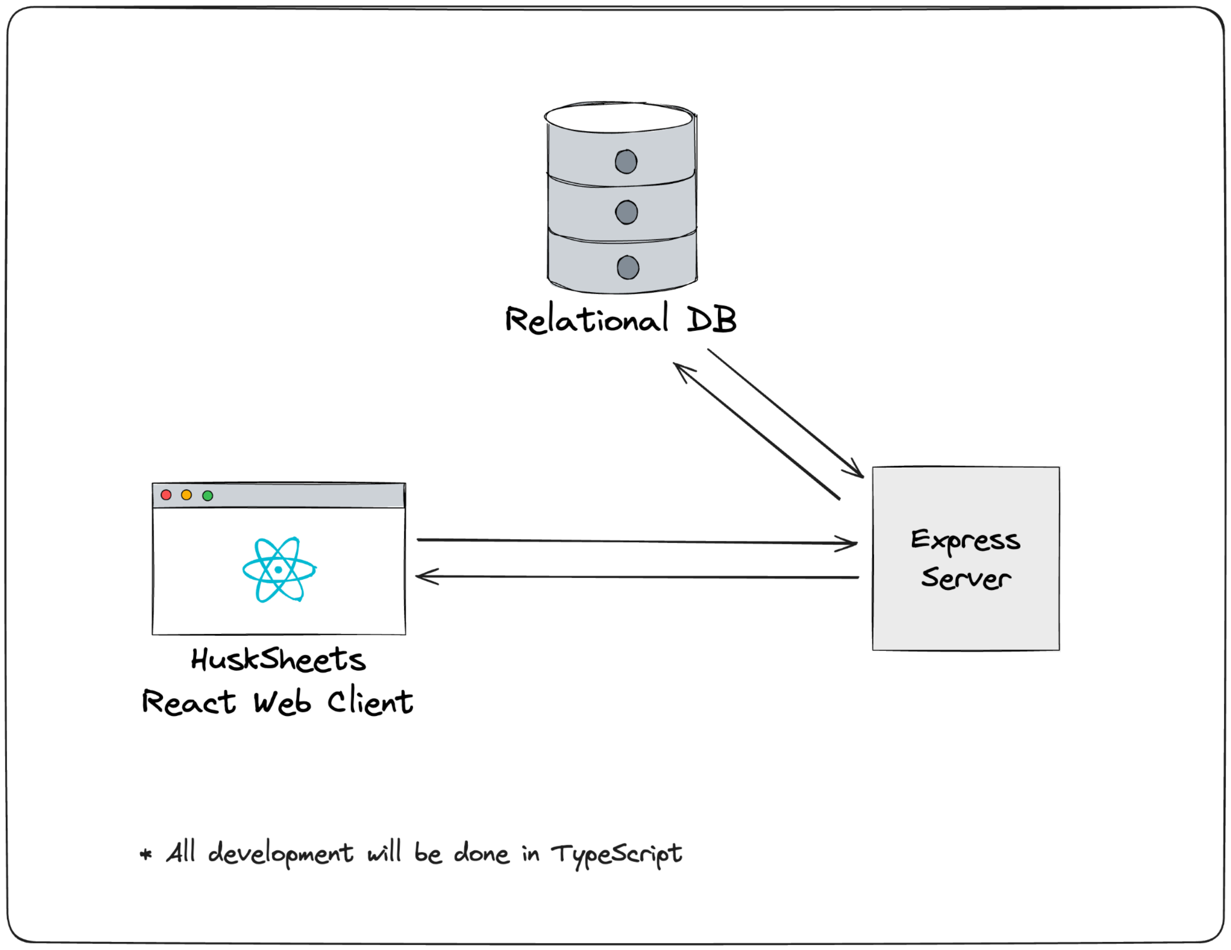
Public/Private Sheets	Low	A user might not be able to subscribe to a sheet if they are set to private
Save to multiple Locations	Low	Save sheet to disk, save sheet as csv
Multiple Sheets Open at once	Low	Tabs for different sheets a publisher can edit
Select multi cells	Low	Be able to cut, copy, paste
Styling for sheets page	Low	Be able to customize fonts, colors
More sophisticated auth	Low	Be able to have account recovery/2fa authorization
Automatic Updates	Low	If someone saves the sheet while I am editing it, I want to be notified and allowed to update my sheet right away

Description

HuskSheets is a collaborative spreadsheet software that allows users to work on the same spreadsheets over the internet. Users can make changes locally and then push those changes to a server. Others working on the same sheet can request those changes form the server, keeping the spreadsheet up-to-date across all users. The goal for this project is to create a seamless and easy to use end product that allows users to persist and collaborate on spreadsheets. Our team is all familiar with our proposed tech stack.

Architecture

- React frontend
- Express backend
- Cucumber testing framework (TS version)
- SQLite persistent store


Relational DB
Express Server
HuskSheets React Web Client
* All development will be done in TypeScript

Minimum Viable Product (MVP)

A basic spreadsheet application that displays a user interface to (1) create a new sheet, (2) access an existing sheet, (3) edit an existing sheet, and (4) push changes to a persistent store. Does not support formulas. Support for one publisher (i.e., no users).

Create a new spreadsheet – *See table for further breakdown*

Stakeholders and Interests:

User: Wants to load a new sheet locally.

System: Must create the sheet in the DB with the given name.

Preconditions: The user has navigated to the Husksheets UI landing page.

Postconditions: A new sheet is created in the DB and the user is redirected to edit the sheet.

Basic flow:

1. User navigates to HuskSheets UI. dashboard
2. User enters a name and clicks the “Create new sheet” button.
3. Client makes a request to the server (`createSheet`).
4. Server creates an empty sheet object in the DB.
5. Server sends a success/failure response to the client.
6. Client reports a success/failure on the UI and redirects the user to the editing view on success.

Extensions:

On error, the program tells the user what the error is and remains on the landing page.

Requirements:

If a sheet with the given name already exists, ask the user to try again with a different name.

User is automatically redirected on successful creation.

User input cannot break the client (i.e., escapes, quotes, input limitations, etc.).

Open issues:

N/A

Access an existing sheet *– See table for further breakdown*

Stakeholders and Interests:

User: Wants to access an existing sheet.

System: Must retrieve and display the sheet to the user in a timely manner.

Preconditions: The user has navigated to the Husksheets UI landing page.

Postconditions: User can view and edit the requested sheet with the most up-to-date changes while waiting no longer than 3 seconds for any sized sheet.

Basic flow:

1. User navigates to Husksheets UI.
2. Client makes a request to the server to retrieve existing sheet names (`getSheets`).
3. Server fetches the existing sheet names (for the sole publisher) from the DB.
4. Server returns the sheet names to the client.
5. Client displays the sheet names as an interactive list on the UI.
6. User selects a sheet name from the list of sheet names.
7. Client makes a request to the server with the sheet name (`getUpdatesForPublished`).
8. Server fetches the sheet from the DB.
9. Server returns the sheet payload to the client.
10. Client redirects to the editing view with sheet data.

Extensions:

On error, the program tells the user what the error is and remains on the landing page.

Requirements:

User is automatically redirected on sheet load.

User can recover if a sheet is not found.

The application should handle retrieving large sheets.

Open issues:

N/A

Edit and save changes *– See table for further breakdown*

Stakeholders and Interests:

User: Wants to make and save changes to the sheet.

System: Must record and store updates in the DB for future reference.

Preconditions: A sheet is open in the UI.

Postconditions: Changes are made and reflected in the DB.

Basic flow:

1. User makes some changes to the sheet.
2. User clicks on the “Save” button.
3. Client makes a request to the server with the updates (`updatePublished`).
4. Server generates an ID for the updates and stores them in the DB.
5. Server sends a success/failure response to the client.
6. Client informs the user of a success/failure via the UI.

Extensions:

If saving fails due to an error (e.g., no space, no permission), the system alerts the user and suggests possible fixes. The system does not overwrite any previously saved state until the issue is resolved.

Requirements:

System ensures atomicity of updates.
New pushes overwrite old pushes.

Open issues:

N/A

Formula support *– See table for further breakdown*

As a user, I would like to enter basic arithmetic, logic, or bitwise formulas between 1 to *n* number of cells and have it executed correctly.

Acceptance Criteria:

1. Upon entering proper grammar for a formula that is logically correct in a ref cell, once I execute, the ref value has the correct solution for the formula.
2. I can execute the following formulas: 'IF' | 'SUM' | 'MIN' | 'AVG' | 'MAX' | 'CONCAT' | 'DEBUG' with n number of cells
3. If a cell that is included in the formula has no value, it is ignored.
4. If the grammar for the formula is incorrect, I expect the execution to not proceed and an error message to be shown.

Login page *– See table for further breakdown*

As a user, I want to be able to log into the application to access my sheets.

Acceptance Criteria:

1. Simple, navigable login page with a username and password entry section
2. I am notified if my credentials are incorrect
3. I am able to create a new account

Desirable Features

Extending the basic spreadsheet functionality to support (1) multiple users and (2) complex functionality.

Scalability

As a user, I want this program to support 50 users each with 50 unique sheets.

Acceptance Criteria:

1. Under these criteria, there should be now slow down past the predefined 3 second maximum delay
2. No changes I make should be lost, all changes will be stored in the history of each sheet

Logout

As a user, I want to be able to log out to protect my account and/or sign into a different account.

Acceptance Criteria:

1. I can logout at any time
2. I am redirected back to login after logout

Dynamic size

As a user, I would expect to open a sheet that does not have a fixed size, and would expand to fit the work I add into it.

Acceptance Criteria:

1. On opening a new sheet, size is fixed to a default size of 50x50
2. I should be able to add more rows or columns up to 1000 in each direction
3. I should be able to scroll to view all rows and columns

Automatic updates

As a user, I want my sheet to periodically update with changes from the server and to be able to request new changes manually, so that my sheet stays up-to-date.

Acceptance Criteria:

1. I expect the client to periodically poll the server for new changes
2. If new changes are discovered, I will be given a notification with an option to reload the sheet to receive the latest changes
3. If I have unsaved changes, I will be warned that my changes will be lost before reloading the page
4. If I save without accepting the most recent changes, I will be warned that my changes could potentially overwrite others' changes

Bonus Features

Nice-to-haves.

Create new account

As a new user, I want to be able to create a new account to create and manage sheets.

Acceptance Criteria:

3. Simple form to create a new account
4. Form should include username, password, and password-verification fields
5. I am notified if an account with these credentials already exists
6. I am not able to overwrite other users' accounts
7. I am redirected back to login after successful account creation

Auto-saving

As a user, I want my sheet to automatically save my recent changes so that I don't risk losing my work.

Acceptance Criteria:

1. I will not be able to auto-save unless my sheet is up-to-date
2. I will be notified if I do not have the latest version of the sheet and that I will not be able to auto-save until I accept the most recent changes or manually save to overwrite the incoming changes
3. The client will periodically make requests to save my changes
4. The client will check for incoming updates before auto-saving

Password-protected sheets

As a user, I want a way to selectively share access to sheets so that I can protect sensitive data.

Acceptance Criteria:

1. If I do not "own" a sheet, a password will be required to view and modify the sheet
2. If I provide the appropriate password, I will be able to manage the sheet like the owner
3. If I do not provide the appropriate password, I will not be able to view or modify the sheet

Multiple tabs/sheets open

As a user, I would like to have tabs within my spreadsheet and be able to reference cells on one tab in another tab similar to the Excel and Google Sheets implementations.

Acceptance Criteria:

1. I should be able to create and delete tabs of sheets on each individual spreadsheet
2. I should be able to reference cells on one tab from another

Styling

As a user, I would like to be able to format my sheet for customizability.

Acceptance Criteria:

1. I should be able to change the coloring, font, etc. of cells

Alternative save locations/export as CSV/ import as CSV

As a user, I would like to be able to export my sheet in a csv and other popular formats in order to export the sheet into a different application.

Acceptance Criteria:

1. I should be able export a sheet as a properly formatted csv or other popular file type on disk

Selecting multiple cells

As a user, I would like to be able to highlight multiple cells to perform bulk operations.

Acceptance Criteria:

1. I should have the ability to select multiple cells at a time
2. I should then be able to either copy or cut those cells and paste them in a different area of the sheets

Share editing rights with another user

As a user, I would like to be able to give access to specific users so that they can easily access a sheet indefinitely.

Acceptance Criteria:

1. I should be able to share editing rights with another user, either via a link or some internal operation

More sophisticated authentication/authorization

As a user, I would like some form of account recovery and/or 2FA to manage my account.

Acceptance Criteria:

1. I should be able to provide an email and/or third-party auth provider

Automatically reconcile edit conflicts

As a user, I would like to avoid conflicts with other users editing a sheet to prevent lost work.

Acceptance Criteria:

1. I should only have to reconcile conflicts if they cannot be automatically resolved (i.e., changes made to the same refs)
2. Any non-conflicting edits should be handled without issue