OK we are going to make a radical change.

MOTIVATION

THE ENTIRE BEAUTY OF brokering the page input through LLM is that we can generate the UI part ‘on demand’ – depending on what we find in the page.

((1)) eliminate the ‘form input’ in sidebar.html with ONE LARGE MAIN text box that we will use to display key=value pairs. We already have a function function formatStateAsKeyValuePairs()

that captures the right idea – no matter what STATE we are in – we format it as key=value pairs and display it in the display\_win. THE DISPLAY WINDOW IS ALSO EDITABLE OF COURSE

(((2))))

We need a DB layer that is pluggable

the DB\_layer.js should be an interface just like parser.js

and be implemented by DB\_local\_prisma\_sqlite.js for now – and later by a cloud solution

The DB\_layer subclass should:

1. take a state object as input
2. create Client and Booking objects
3. call toJSON on each (with overlap – the same name for instance)
4. save the results to the DB using appropriate endpoints

(((3)))

Extension UI will evolve from sidebar.html:

HEADER and HAMBURGER

status bar shows single line status report i.e. ‘Gmail thread detected’

and RELOAD button currently there

MAIN WINDOW

display\_win

shows whatever key=value pairs are found in current state object

CANCEL BUTTON (clears all fields and resets the state)

and

SAVE BUTTON (save appropriate STATE to Booking or Client object in DB layer)

FOOTER with the grass – just like we have now – showing all the logging and debug output

WHAT WE WANT!!!!

\*In the browser user goes to Gmail.

\*user has a long Gmail conversation going in a thread with several collapsed panes.

\*user clicks the extension to open it.

\*The extension detects a Gmail page, updates status in UI.

\*the extension loads gmail\_parser.js

\*gmail\_parser configured to use LLM at url

\*\*url corresponding to LM\_Studio running port 1234 with a slm loaded and a meta \*\*prompt designed for this purpose

\*LLM returns a structured JSON containing all the fields it can find

\*parser creates a state object

\* UI renders state object to display win as key=value pairs

\*USER HAS OPTION TO CHANGE KEY=VALUE PAIRS

\* user presses ‘Save’ – sidebar sends state object to db\_layer

\* db\_layer is configured to use DB\_local\_prisma\_sqlite.js

\* db\_layer creates Client and Booking objects

\*\* add toJSON() methods to Client and Booking ojects

\* db\_layer saves Client and Booking using calls to subclass

\* state object sent to rendering layer

\* render\_layer is interface subclassed by pdf\_render

\* pdf\_render will generate a save a pdf using a configuration we will create later

\*

--------------- EXTRACT EMAIL ENTIRE BLOB -----------------

function extractEmails() { const emailThread = document.querySelector('.ii.gt'); if (!emailThread) { alert("Could not find the email thread. Make sure you are in a Gmail conversation view."); return; } // Find and click all "show quoted text" elements to expand collapsed messages const collapsedElements = document.querySelectorAll('.ajy'); collapsedElements.forEach(el => { el.click(); }); let fullText = ''; // Use a more specific selector to get all individual email bodies const emailBodies = emailThread.querySelectorAll('.a3s.aiL'); emailBodies.forEach(emailBody => { fullText += emailBody.innerText + '\n\n---\n\n'; }); // Send the extracted text back to the extension's background script chrome.runtime.sendMessage({ fullThread: fullText }); }