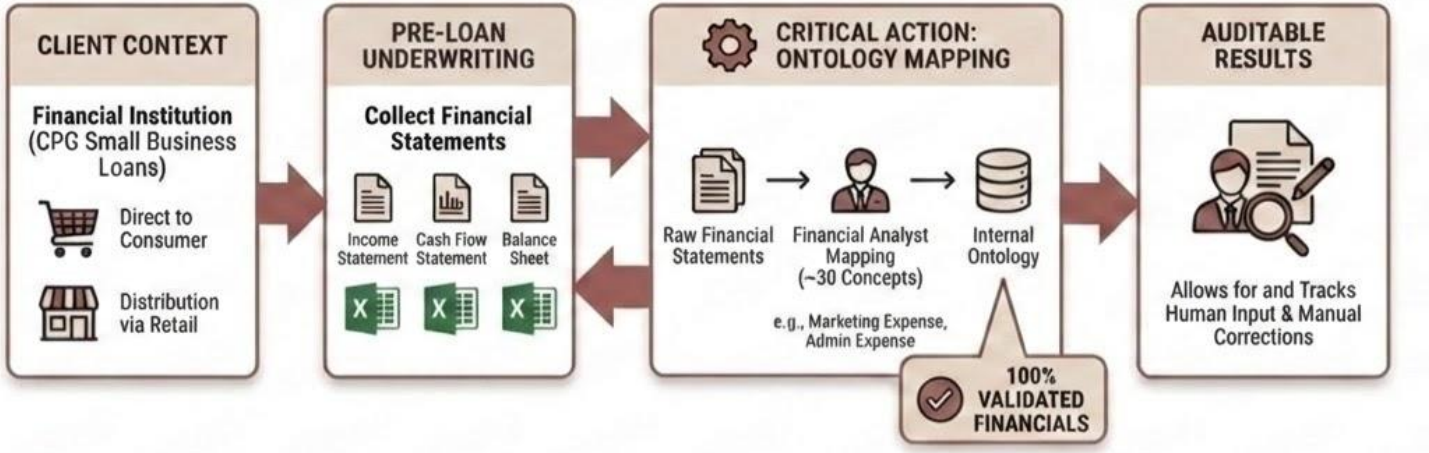
Exgent - A Finance Workflow

- Exgent is designed to solve the problem of evaluating a company's financials before a loan is made.
- As companies have different reporting formats, the accounts must be standardized so the internal loan evaluation system can process it.
- Exgent provides an Agent solution to a task that is currently 100% manual



Our Goal

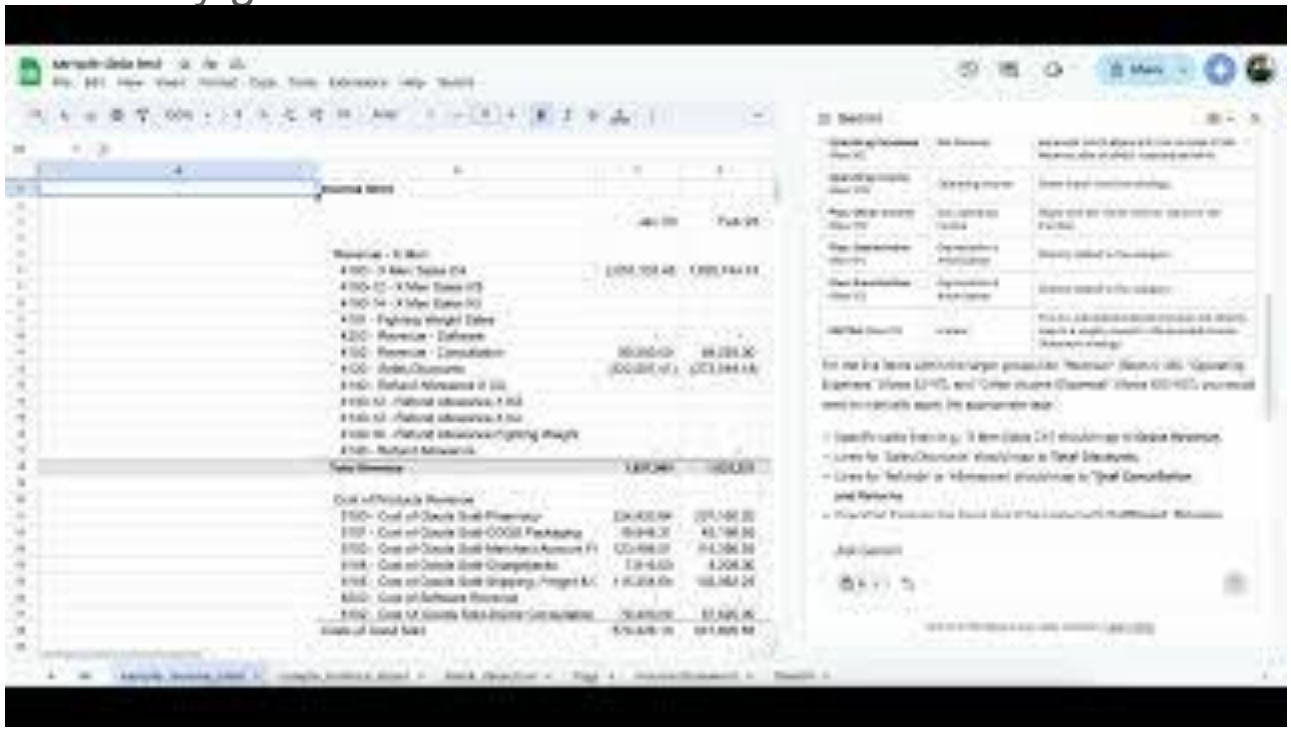
Turn excel files into standardized reports. 30+ standard tags need to be associated

	A	B	C
1	**Income Stmt**		
2			
3		Jan '23	Feb '23
4			
5	Revenue - X Men		
6	4100 - X Men Sales CA	2,051,332.45	1,820,744.74
7	4100-12 - X Men Sales KS		
8	4100-14 - X Men Sales NJ		
9	4101 - Fighting Weight Sales		
10	4203 - Revenue - Software	-	-
11	4102 - Revenue - Consultation	99,250.00	84,525.00
12	4120 - Sales Discounts	(322,587.41)	(273,049.18)
13	4140 - Refund Allowance X CA		
14	4140-12 - Refund Allowance X KS		
15	4140-14 - Refund Allowance X NJ		
16	4140-16 - Refund Allowance Fighting Weight		
17	4140 - Refund Allowance	-	-
18	**Total Revenue**	**1,827,995**	**1,632,221**
19			
20	Cost of Products Revenue		
21	5100 - Cost of Goods Sold-Pharmacy	224,420.64	207,160.20
22	5101 - Cost of Goods Sold-COGS Packaging	19,848.31	45,198.55
23	5103 - Cost of Goods Sold-Merchant Account Fe	123,486.51	115,356.55
24	5104 - Cost of Goods Sold-Chargebacks	7,816.00	4,208.00

Standard tags	**Income Stmt**	
		Jan '23
	Revenue - X Men	
Gross Revenue	4100 - X Men Sales CA	2,051,332.45
Gross Revenue	4100-12 - X Men Sales KS	
Gross Revenue	4100-14 - X Men Sales NJ	
Gross Revenue	4101 - Fighting Weight Sales	
Gross Revenue	4203 - Revenue - Software	-
Gross Revenue	4102 - Revenue - Consultation	99,250.00
Total Discounts	4120 - Sales Discounts	(322,587.41)
Total Cancellation and Returns	4140 - Refund Allowance X CA	
Total Cancellation and Returns	4140-12 - Refund Allowance X KS	
Total Cancellation and Returns	4140-14 - Refund Allowance X NJ	
Total Cancellation and Returns	4140-16 - Refund Allowance Fighting Weight	
Total Cancellation and Returns	4140 - Refund Allowance	-
Net Revenue	**Total Revenue**	**1,827,995**
	Cost of Products Revenue	
Gross Product Cost	5100 - Cost of Goods Sold-Pharmacy	224,420.64
Gross Product Cost	5101 - Cost of Goods Sold-COGS Packaging	19,848.31
Shipping & Fulfillment Cost	5103 - Cost of Goods Sold-Merchant Account Fe	123,486.51
Shipping & Fulfillment Cost	5104 - Cost of Goods Sold-Chargebacks	7,816.00

Start point - Dec 26th 2025

Lets try gemini - How hard can it be?



https://youtu.be/aau53DzZsdE

Maybe just

- Open sheets
- Add prompt
- Ask gemini to insert ontology tags

*** Recorded on Jan3rd, with prompt from demo**

Settling on a no-compromise design - Dec 30th

- Users **upload excel files** into a React application
- Users must be able to i**nteract with the excel sheet** (Similar to canvas in modern AIs)
- Make the process interactive so the analyst remains involved
    - **Remove the boring parts of validating financials** “such as checking the totals for each date”.

- **Divide and conquer the problem** that is too hard to solve in single step
    - Agent to understand **sheet structure**
    - Another agent to take logical part of sheet **and tag them**
    - Testing on sample data in gemini web app with prompt engineering has been promising..

- Must maintain **Audit trail** and be able to review all changes.

I choose with Python Fast API and React 13 for the frontend, Google ADK for the agent framework.

Github: https://github.com/sanjayvenkat2000/exgent

Demo

https://youtu.be/WT1LHVl8Qgc

Learning - What the demo achieves

- Models are much better at dealing with csv or markdown than excel files.
  - Matches their training data more closely
- Context engineering is key
  - Small problems with **clean context** and **clear prompt instructions** proved very effective.
  - Asking the **model to rewrite its prompt** for clarity help as always.
- In order to build trust an interactive experience is ideal.
  - Show the user the end result
  - Show the user the steps
  - Allow the user to change the output of the model
  - Maintain an audit trail

Architecture highlights - Server

Storage

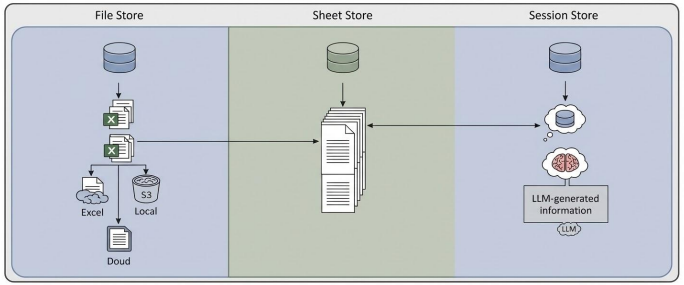
- **File Store**: Stores uploaded files and access control information
- **Sheet Store**: Stores metadata and validation information extracted from the excel files. This is a **versioned store**
- **Session Store**: Stores the LLM chat outputs that are synchronize with the Sheet Store

Agents

- **Sheet Structure Agent**: Purely responsible for understand the sheet structure
- **Tag Agent**: Responsible for tagging
- **Verify Agent**: Uses tools to validate the sheet numbers

Server

- REST inspired endpoints with streaming chat endpoint (all endpoints shown)

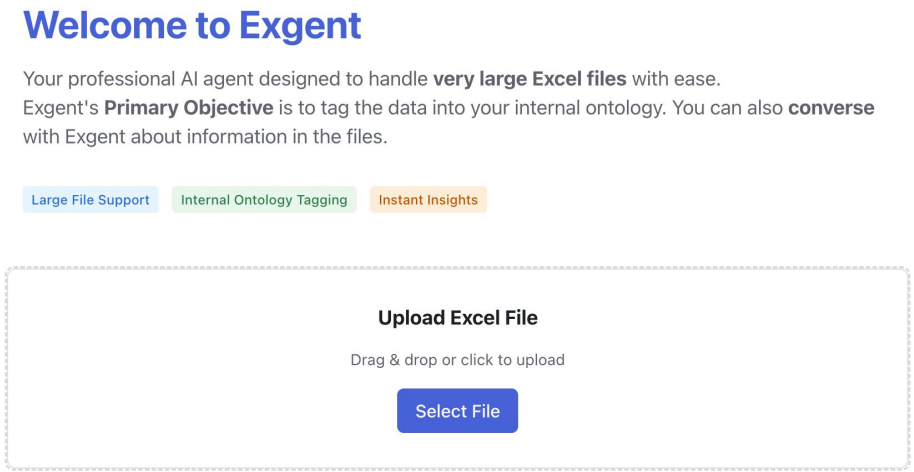


GET	/	Read Root
GET	/files	List Files
DELETE	/files/{file_id}	Delete File
GET	/files/{file_id}	Get File Details
GET	/sheets/{file_id}	Get Sheet Names
GET	/sheetdata/{file_id}/{sheet_idx}	Get Sheet Data By Index
GET	/sheetinfo/{file_id}/{sheet_idx}	Get Sheet Info By Index
POST	/sheetinfo/{file_id}/{sheet_idx}	Update Sheet Info
POST	/sheetchat/{file_id}/{sheet_idx}	Sheet Chat Stream
GET	/sheetchat/history/{file_id}/{sheet_idx}	Get Sheet Chat History
POST	/upload	Upload File

Architecture highlights - Client

Not Much: “Good react app. Supports streaming for agent”

- **Use best of class libraries:** React apps can break down quick if good design is not followed.
  - Tanstack useQuery is used here to borrow the great work they do.
- Abstract all service calls to a single file, we need to implement authentication etc. (one file to change later).
- Use good state management using providers
  - Providers allow good access without passing props all over the place (prop tunneling)
- Do not use **useEffect(..)**, where possible, as this is an anti-pattern and causes issues very fast that are hard to debug.



Architecture highlights - Client

Synchronizing UI state

- Ag-UI and A2UI are emerging standards. Needs a few more months to flush out.
- **Current idea: Use a custom LLM message to tell the UI “what to update”.**
- Should migrate to standards to get UI reuse.