I extensively interacted with AI particularly I used Claude, ChatGPT, TRAE (an LLM integrated IDE), and Lovable (No-code platform).

In general I think blind vibe coding is bad. Vibe coding while using file context and checking what the AI is putting (as if you are a senior engineer and the LLM is a junior). That is the future.

Once you create a working version on Lovable, you can connect it to github and then use that source code to develop further. For the frontend of the app itself I fed in a PRD prompt.

I watched this Youtube video on how to "vibe code" without having to write too many prompts: https://www.youtube.com/watch?v=r2TUTOHoYGA&t=2948s and one of the advices they gave was to feed in "Product Requirements Documents" to add specificity for the agent.

I use this to create a full-stack app to iterate. Because Lovable interacts solely in React, Typescript and doesn't use FastAPI or sophisticated backend. I just created the interface where you can feed in the email blurb and it creates a basic JSON that just extracts the first 10 words. This functionality is minimal and only written in the frontend. Later I would exchange it and replace it with a Backend infra on FastAPI to test whether FastAPI is working. After that I would iterate it and add more complexity in python. So build it step by step.

https://github.com/rohitk2/email-blurb-morph/commit/8f7245bf669200630930c86f7d05612f979d8055

Here is the PRD fed into Lovable. Note I used chatgpt to help me construct a thorough one: https://github.com/rohitk2/email-blurb-morph/blob/main/PRD_Prompt.tx

After just 3-4 prompts I created a full app that I was happy with and now moved to the FastAPI

Started with research:

have something very similar to this:......<code snippet> Where I connect the main.py using FastAPI. Make sure the CORSMiddleware is correct.

Converting frontend implementation to backend FastAPI:

actually what I am trying to do is instead of implementing all this in the frontend itself.

- ingesting the typed email text blurb

- converting the email text into an extracted JSON

I'm trying to implement all that in the python main.py

Can you do that for me?

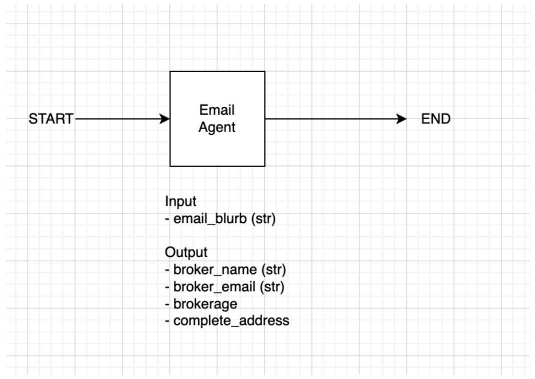
Use the code previously mentioned as reference
Another good practice I did is to ensure that we don't have random pip issues so I made sure we had a requirements.txt and the requirements.txt was extensible. This simplifies the process and eliminates package issues:

no instead of just doing pip install fast API or whatever modules I need here main.py 1-6 can you create a requirements.txt that I can then use for pip install
—-----------—-----------—----------—----------—-----------—-----------—-----------—----------

.env 1-8 now that everything is setup in place main.py 1-7 email_parser_agent.py 1-9 all the imports I need can you put them in my requirements.txt

Now I added an agent. I had code from another project I did that used langgraph so I referenced it in the prompt and also had added a system design drawing:

<langgraph code>...
Given similar code what I want to create is this agent that takes in a text blurb and outputs 4 fields



Modified to groq:

email_parser_agent.py 9-10 I want to enable it such that I use my groq API key so add a .env file.

Also add all the necessary imports needed for groq email_parser_agent.py 1-7

Created entire functions with prompts and a lot of file context:

First I want to modify this function main.py 46-61 to take just this as input main.py 12-12 from here main.py 11-12

I want it to run the agent email_parser_agent.py 1-142

input corresponds with email_parser_agent.py 14-15

output email_parser_agent.py 18-22 email_parser_agent.py 102-107 is then put as the output of this function main.py 46-61 here main.py 57-61

Also modify the imports up here main.py 1-6 to ensure that it properly retrieves the agent

Created the Mongo caching functions using a prompt. Wanted to make them modular so I had a separate file for it. I put everything I needed into the .env and had a main so that you can test it separately before importing it.

add_sample_entry.py 1-1 given this code snippet:

<code snippet>

Also note I have the uri saved:

.env 10-10 write a sample script that all it does is insert 1 entry:

{field: test} into the table

Added another function and gave LLM step by step logic, function declaration, parameters, file context:

create another function which references the same database as the insert like this mongo_caching.py 25-43

but instead tries to retrieve an entry

def cache_hit(email_blurb) -->

RETURNS

{

"broker_name": broker_name,

"broker_email": broker_email,

"brokerage": brokerage,

"complete_address": complete_address,

"broker_name_confidence": broker_name_confidence,

"broker_email_confidence": broker_email_confidence,

"brokerage_confidence": brokerage_confidence,

"complete_address_confidence": complete_address_confidence,

}

IF THERE

OR

False if not there

After testing the cache operations. Cache hit + cache insert. I integrated those 2 functions using a prompt. The prompt had file context, step-by-step logic. Another thing was sometimes TRAE would forget imports so I added that in:

main.py 55-91 I want to change the flow of this function

Original Flow

(1) runs agent

main.py 61-61

(2) inserts to mongo

main.py 65-75

(3) main.py 82-91

New Flow

(1) Checks to see if there is a cache hit mongo_caching.py 7-48

(2A) YES CACHE HIT ===> return same stuff main.py 82-91 but from the cache function mongo_caching.py 39-48

(2B) FALSE NO CACHE HIT ===> Does the original flow

----

(1) runs agent

main.py 61-61

(2) inserts to mongo

main.py 65-75

(3) main.py 82-91

Created gitignore:

.env 1-10 can we put a git ignore here Terminal 53 - 53 in this directory that doesn't add the .env

Made many incremental fixes to functions. Here is an example:

main.py 57-115 now add latency to the return type

basically just calculate the time_start at start of function main.py 57-57

time_end when there is a return

Here if cache works properly

main.py 69-80

Here if cache miss

main.py 105-115

Using previous files if there was a lot of overlap, I would use them as reference to duplicate files:

mongo_caching.py 49-101 mongo_caching.py 1-6

Use this mongo_caching as reference

And create a function inside this file mongo_metrics.py 1-1

def insert_tracing(tokens_used, latency)

instead of using the cache collection, use metrics mongo_caching.py 85-85

Use the

----

mongo_metrics.py 1-109 I want to duplicate this file.

Here are the functions I want to add

def insert_log(request_id, source_hash, cache_hit, latency) :

- request_id: str

- source_hash: str

- cache_hit: boolean

- latence: float

==> inserts into mongo table however it is not

mongo_metrics.py 38-39

It is MailMorph.Logging

...but otherwise do the same

def get_logging():

mongo_metrics.py 50-92 does same as this except the item type is different not mongo_metrics.py 86-90 but this

{ - request_id: str

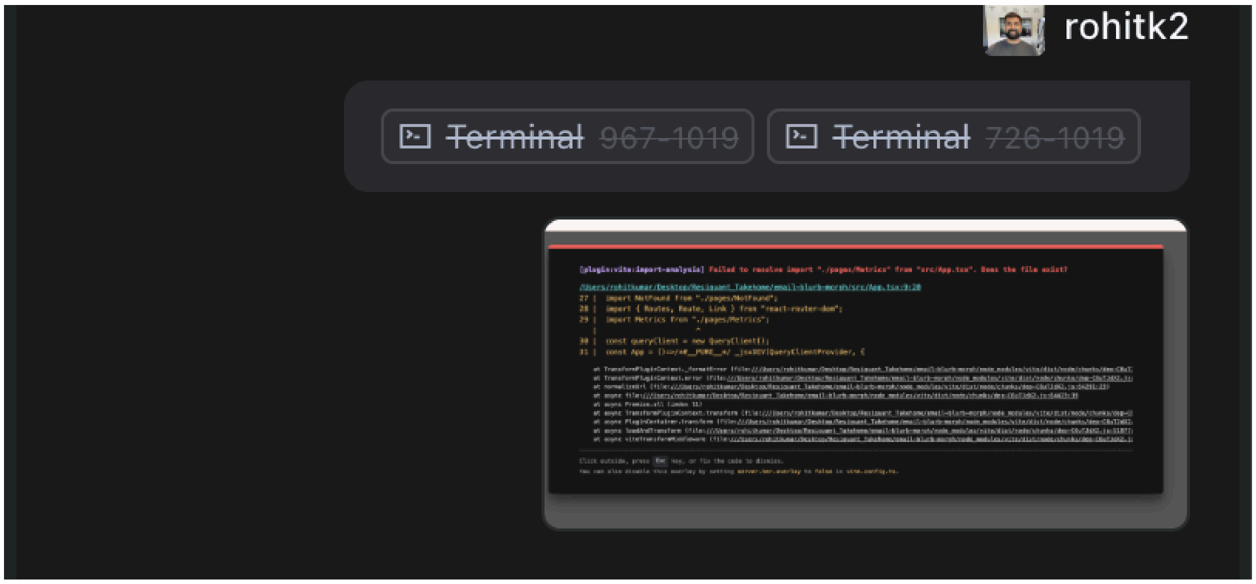
- source_hash: str

- cache_hit: boolean

- latence: float }

Do a main to test both functions

Sometimes I would even screenshot logs



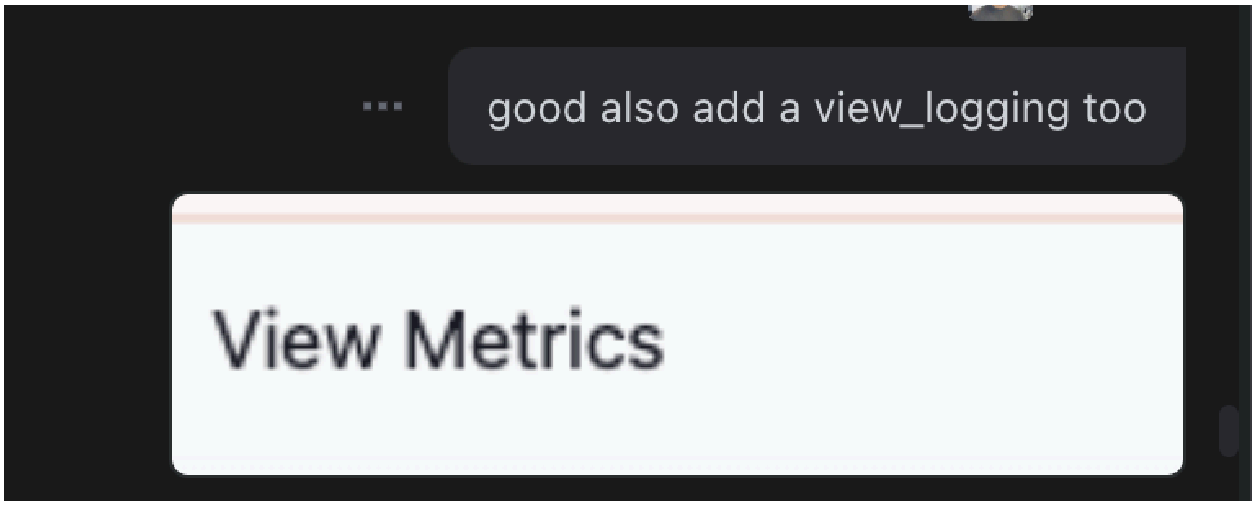
```
[plugin:vite:import-analysis] Failed to resolve import "./pages/Metrics" from "src/App.tsx". Does the file exist?

/Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/src/App.tsx:9:20
27 |  import NotFound from "./pages/NotFound";
28 |  import { Routes, Route, Link } from "react-router-dom";
29 |  import Metrics from "./pages/Metrics";
   |                      ^
30 |  const queryClient = new QueryClient();
31 |  const App = ()=>/*#__PURE__*/ _jsxDEV(QueryClientProvider, {

    at TransformPluginContext._formatError (file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]
    at TransformPluginContext.error (file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]
    at normalizeUrl (file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]
    at async file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]
    at async Promise.all (index 11)
    at async TransformPluginContext.transform (file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]
    at async PluginContainer.transform (file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]
    at async loadAndTransform (file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]
    at async viteTransformMiddleware (file:///Users/rohitkumar/Desktop/Resiquant_Takehome/email-blurb-morph/node_modules/vite/dist/node/chunks/dep-[illegible]

Click outside, press Esc key, or fix the code to dismiss.
You can also disable this overlay by setting server.hmr.overlay to false in vite.config.ts.
```

Use human touches to talk to the frontend like I was the product team:



good also add a view_logging too

Integrating regex fallback:

I want to do a regex fallback for the broker_email and complete_address using this function

regex_fallback.py 86-86

Do what you gotta do with imports

main.py 1-14

Now what I want is 2 things in 2 places:

LOCATION 1 CACHED:

(1) if cached["broker_email_confidence"] < 0.8 :

replace

main.py 84-84 with the fallback email regex_fallback.py 106-106

(2) if cached["complete_address_confidence"] < 0.8 :

replace main.py 86-86 with the fallback regex_fallback.py 107-107

LOCATION 2 UNCACHED:

(1) if res.broker_email_confidence < 0.8

replace main.py 119-119 with the fallback email regex_fallback.py 106-106

(2) if res.complete_address_confidence < 0.8 :

replace

main.py 121-121 with the fallback email regex_fallback.py 106-106

Integrating hashing:

So depending on this .env 12-12 variable being true or false I want it to perform certain behaviors

Basically all PII data must be hashed and unhashed when performing Mongo operations

Do what you gotta do to import these functions email_blurb_hashing.py 1-54 in mongo_caching.py 1-5 mongo_logging.py 1-8 mongo_metrics.py 1-7

Now whenever you insert any PII data hash

unhash is only needed in certain situations

Let's begin with mongo_cache:

(1) Hash these before inserting

mongo_caching.py 89-89 mongo_caching.py 90-90 mongo_caching.py 91-91 mongo_caching.py 92-92 mongo_caching.py 92-92

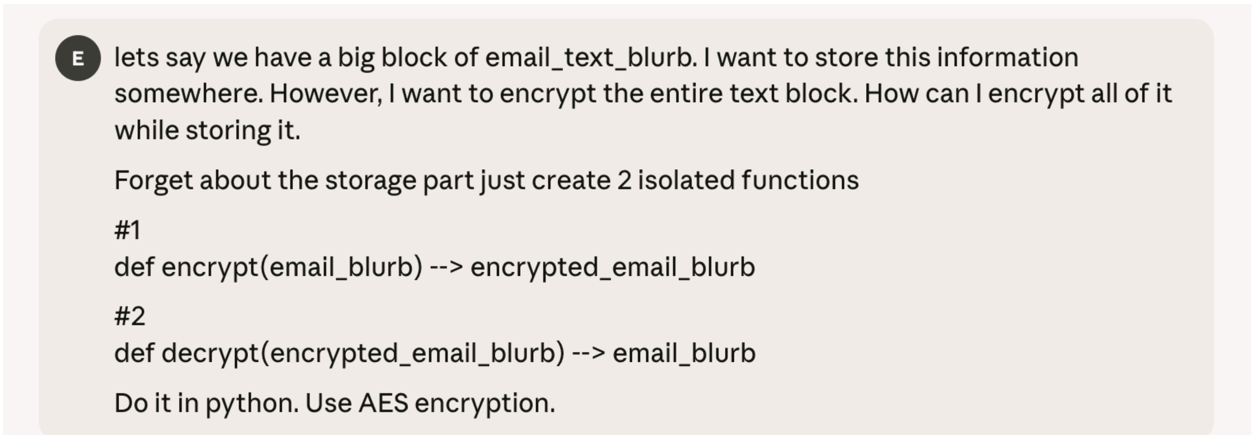
(2) Hash this before querying mongo_caching.py 34-34

(3) Unhash these mongo_caching.py 39-42 before returning

only perform these 3 operations if .env 12-12 is true

Otherwise leave like the original

Sometimes if the function was disconnected I used the claude terminal which is separate from TRAE:



lets say we have a big block of email_text_blurb. I want to store this information somewhere. However, I want to encrypt the entire text block. How can I encrypt all of it while storing it.

Forget about the storage part just create 2 isolated functions

#1
def encrypt(email_blurb) --> encrypted_email_blurb

#2
def decrypt(encrypted_email_blurb) --> email_blurb

Do it in python. Use AES encryption.

If I want to do more of a research and idea exploration.