this is Varun logging at 1547 so the project title is WELLNESS COMPANION, AI

LET ME JUST EXPLAIN YOU THE COMPLETE PROJECT

The idea is to typically separate the whole company in air workflow in six different layers

Being

1. desktop layer

2. Core back and layer

3. Aiml orchestration layer

4. Data layer

5. Infrastructure and deployment layer

6. CD pipelines and logging

So now let’s talk about the whole project workflow. The ideation is that the app is installed on desktop through DMG or EXE extension and also if possible further extension into Linux environment. So when the app is downloading, the user will have a mandatory single LLM there are multiple Ls but he should at least select one some LLM to process in the backend and hopefully to also add multiple different different vector Dbs like chroma db, QDrant and FAISS

Now once the installation is completely done, so he gets all of the packages that are required to download. I think through EXE or DMG. Once it is done, he would be shown at desktop application that is completely based on PyQt6 plus QSS interface.

There’s going to be a health check if the app is working or not a simple before getting the dashboard is simple notification pop hopefully to add it to check out healthy and points if it’s completely working or not just so that if it’s buggy we could report it in that fashion. So once this is checked up so this is cheque boxed it’s going to proceed into the next session which is Authentication and in this desktop itself, we would have a briefcase for packaging. We have pie cut plus Q for front-end and entry CI point so this is where the whole entry goes on so this is where we have the core orchestrator where it triggers okay what to happen where it is the front-end trigger point for the backend API, the core backend, fast API

So now in this interface, he has to login so this login is verified and authenticated by using Google Oauth or AWS cognitive and this is triggered by fast API. So the core backend layer is where the fast API so this fast API also checks for the health health and points this fast API is what triggers the Lang chain if there is a query or not so there is going to be in the whole this part so whatever UIU axis there’s going to be multiple sessions like okay you can pick what kind of LLM you want you can pick what kind of documents you wanna upload so there’s going to be document session where you can ingest documents so once the document everything is like you know triggered by fast API and this fast API is completely in the core back layer which is triggered by the front end core orchestrator, the orchestrator only deals with inputs so okay what to be shown what not to be shown so it it’s like a two way where this orchestrator picks up Jason response gives out CLL entry point so it is what is triggering the back in main API

So in this fast API itself we get if the query is present it’s going to trigger the next layer

Which is Aiml orchestration layer. So this line this API is triggering Lang chain and this Lang chain is triggering rag or texture. Which is the main connecting point for the Aiml orchestration layer so where will understand the vector search so based on Cosign similarity or some other function where you realise okay if the data that we have in our Q or in our FA ISS or Croma DB that yes is this reliable or not if the confidence score is less if it would trigger travel API and then generate a response and trigger back to fast API which is going to trigger back again. The core orchestra and the core Orchester is going to trigger the QT interface. If the confidence is more and good enough. This is known through the next layer which is data layer that typically handles all the vectors DB. in this data itself, we have one more supplier which is called embedding pipeline. So this pipeline triggers all the document upload. Once the document is uploaded it’s going to break down into chunks. This Chungs are embedded once embedded stored all the stored meta data is put into post. This post is triggered into the next infrastructure layer, AWSRDS while this chunks that are being stored in meta data or also stored in S3bucket

In between the document upload and text printer there is a key main component that triggers a text splitter. What this does this it just it is called embed or Store where you’d understand. Oh wait is it a document that already exist or is it a document that is something new do I have to do the work again update the vector space so if yes, it would trigger splitter. If no, it would generate a response saying no it’s already there.

So once this data is you know the red is the QDRANT. The Postgres are properly orchid properly done we would get into the next phase which is infrastructure and deployment clear. This is one of the keys because the security protocol the docker base debt the RS End AW Ac two which is nothing but an virtual machine. Instance in the cloud it’s like a server and then we would also have the NGI NX API gateway protected with HTTP and SSL which is basically nothing but a protocol for Security and will also have AWS ELASTICACHE which is basically a direct logger from data layer, rais and also the core backend layer API gateway/NGI NX. We also have Dr which triggers the core back in layer authentication which also takes out information from the Aiml orchestration layer. Ola Ma. The local all of these are provided into the NGI NX/API gateway and their properly secured, and one secured their pushed into AWS EC2

So now comes the final year that is CIC D and logging. So here we have prometheus get her actions which trigger briefcase and also we have P installer which is where the whole file is being into EXE or DMG or something.

So one of the key things to note in the whole project are important. Things like

having briefcase and “signing in desktop app layer

having rate rate limiting and health points in core back end layer

Having response synthesiser in Aiml orchestration layer, which typically takes a yes from the vector search result found or not. And also Uma in the Aiml oration layer itself which is like you know local LLM

In the data layer, one of the key things is embedding pipeline and all the databases and the one of the conditionality thing in it that typically triggers okay. Is it a yes or no is nothing but embed plus Store.

And in the infrastructure deployment, we have technologies like AWS RDS EC2 ELA,STICACHE doer NGINX/API, Gateway

And in the final CID pipeline, plus logging, we have briefcase get up action prometheus

sentry Pyinstallers

Son of the idea is to develop the whole project one by one and the approach that I would follow. Is let’s say we go

So first, I’m going to note down all the API points or all the links so I’m gonna make a big diagram. I think I should be making a big diagram where it basically at least I should understand all the folders okay firstly let’s go with folders where let’s not go with API, but rather let’s go with all the folders so okay if this layer I’m gonna need needing this folder maybe like let’s take in the Aiml we might need a rag orchestration folder. We might need a tab search folder. We might need a response synthesiser folder which can have their own files respectively but at least each of them to be having multiple files. Once this is done all of the files are duly noted, just make a little bit of brief on all the APS that I might need in this whole big project because there are around 20 technologies that I am using and they might need what like hundred API or maybe so I’ll have to at least understand the basic API how everything goes the trigger points in between and all so and then I’ll start working on the Core logic which is nothing but very basic one document document injection and Dividing it into into chunks and then dividing it into or providing it into a vector space and once the vector space is done the vector DBH. I’m doing it. I’m gonna pull it back and make a local search once the local search is done. I would also work with Anil so now how I want this to happen is to start working on the folders. Once the folders are done since there is a document upload part in the UI UX so I’ll be writing the code in UI UX while I’ll be importing the data and everything into the Lang and Orchestra from Core. Back and layer will trigger. Basically it’s going to trigger not only the core back and layer Lang chain but also the Core orchestrator whatever the whole function is it’s going to trigger the fast API and the fast API is going to do the document uploaded in embedding pipeline, so now what I’m gonna do is I’m gonna do the embedding pipeline that’s it. I’m gonna store it in cute, not gonna log it but make the whole core pipeline properly working get everything everything everything done properly and then write the APS once it is properly working okay this is this I need to get this API. This is here. I need to get that API once this is done including the MB plus store option Then maybe I would go for the next thing let’s say connecting to a database proper database. So all of these API is I need to connect it to some of the other API. So once this is done every stage I’ll have to push it so once this is done I’ll get back to data basis Once the database is verified, I would try to apply the login the login ID where I am gonna trigger API gateway in Koback and layer and also I would that would trigger infrastructure layer so what I am planning to do in this whole work is, I’ll be dividing it in two different different different Folders across the folders. So each layer is going to be its own working functionalities that are cross dependent on each other so I just wanted to be that way. I’m not sure if I should put the documentation layer all by itself in the Aiml orchestration layer. I think I should also have an instance of it or at least trigger it maybe through API, if not API directly import it Because I also want to provide an off-line thing so even if you don’t have you know since we are using Ola Ma and it can do off-line search maybe at least for the documents that one person is uploading to maybe there to help them to learn so I’ll be working on having it in a proper proper way so that you know it basically triggers everything at least off-line searches happening And once this authentication is done, I would go for UI UX the fourth phase once the fourth phase UI UX I work with it before before doing this in Bai phase 3 at authentication I would complete all of the API’s okay what are this? What are that based on that API I would start working on the UI UX part And once UI UX part is done I would then go for the infrastructure layer where I’ll be tagging all of them into AWS and everything everything maybe even before you are UX would work on AWS completely. Once this is done I would stick I would stick my stick in everything till then and once everything is done then I will be starting with the final layer that is CD and pipe lining so that is my option so this is my initial plan and I am yet to confirm I’m at least for sure. Sure on to work on the Core. Once the Core is working then provide with an API and then start working on before doing this at least get all the folders start working on get of all the API is then start working on the Core and then start working on Maybe the logging and then once logging is done before logging itself I plan to do the storage into all of the data basis. Once the databases store then logging and once the logging is done then UI UX. Once the UI UX is done and then I would go for the final phase that is CiCD pipeline.

The UI ux is proper animation styling if necessary, and along with it a floating icon system tray but very efficient this is just add on to what is required a simple overlay over screen that is very minimised almost like how sliderAi works always ready to summarize same a little pop soo

Desktop layer

- PQT6 + QSS

- NC, Pt/CLI

- Core orchestrator

- Briefcase with “signing

- Web search handler

- Theme engine

- O llama llm

Core backend layer

- Fast API

- API, gateway/NGI NX

- Redis rate limiting

- Health points

- Authentication ( Google Oauth +cognito)

- LLAMA index/Lang chain

Aiml orchestration layer

- Rag orchestrator

- vector search result

- OLLAMA(local LLMz)

- Response synthesiser

- Tavily

Data layer

- Embedding pipeline(document, upload embedded text, splitter Store)

- S3

- PostgresQL

- QDRANT

- REDIS

Infrastructure and deployment layer

- Aws rds and ec2 and aws elsticache

- DockerNgnix/API gateway

CICD pipeline and logging

- Get hub

- Sentry

- prometheus

- Py installer

- Briefcase

Let me tell you the core features once again

- Local document processing: upload and index PDF FILES and documents

- Semantic search: vector base, similarity, search through personal knowledge base

- Multimodal AI: support for text and document analysis

- Web search fall back: When confidence score is low triggers web search

- Cloud synchronisation: option, cloud, backup, and multi device sync

- Privacy, First: local processing with optional, cloud enhancement

Key benefits

- Speed: LLM, for instant responses

- Privacy: sensitive data stays local by default

- Intelligence: combines personal knowledge with web intelligence

- Reliability: fallback mechanisms ensure system availability

Technologies stack

Front-end technologies

- Desktop framework: PYQT6

- System styling: QSS plus Q dark style

- State management: python classes

- Local storage: SQlite

Back end technologies

- Web frame work: fast API

- Language: Python, 3.11

- A SGI server: unicorn

- Test Queue : celery, plus Redis

- Cashing: REDIS

AIML technologies

- LLM framework: OLLAMA

- RAG framework: lllama index/lang chain

- Vector database: QDRANT

- Web search: TAVILYAPI

- Embedding : sentence transformers

Infrastructure technologies

- Cloud provider: AWS

- Container platform: docker plus ECS

- Database: POSTGRESQL plus RDS

- File storage: S3

- API gateway: AWS API, Gateway

Let me just give you the

user query processing flow

Primary flow

user submit query through desktop application, query, pre-processing and intent analysis, vector similarity search in QDRANT database context retrieval from match documents. Local LLM generates response using retrieved content, response, formatting and delivery to user interface

Fall backflow

When local search returns, low confidence results, 0.7 similarity or less system triggers web search fallback mechanism that uses tali API performs web search with optimised query. Web results are processed and filters for relevance. The combined text, local plus Web sent to LLM enhanced response generation with source attribution response delivery with source indicators.

Document injection flow

Document upload process

User select documents through desktop, file dialogue and client. Validate Us file types and sizes. Locally file uploaded to the backend via chunk, HTTP transfer, backend, virus, scanning and content, validation document, passing using appropriate processes like PDF,DOCX and txt

Text extraction and pre-processing I.cleaning and chunking are done and embedding generation using sentence transformers. This data is stored in vector storage in QDRANT with meta data indexing original file storage in S3 with encryption and the user notification of successful processing.

Authentication flow

Initial authentication

User click sign in on desktop system. Opens browser for Google flow, user authorise application permissions, Au call back written Us authorisation code back in exchanges code for Axis tokens, user profile creation and update and PostgresQL JW T token generation for API, axis, token storage and desktop application keyring

Automatic, token, refresh handling

Real-time communication flow

Web socket, connection, Management

Desktop, client establishes web socket, connection, connection, authentication, using JWT, token channel, subscription for users, specific updates, real time, message, routing in delivery, delivery connection, heartbeat and reconnect, logic, graceful degradation to HTTP bowling if needed

Few of the possible API’s

- Authentication APIsuch as login, call back, refresh, logout and profileDocument API

- Document management API such as upload list ID process search

- Chat in API such as conversations, conversation, conversations post message get message post regenerate, post models get status

- Search and knowledge API Semantic, web, hybrid status rebuild

- System and admin API such as Health metric, users, usage, maintenance

One of the key features of the whole project idea is to have multiple models that they want to search. Preferably there are some few models that are good at court generation that are good at generation. So maybe provide multiple models and provide multiple vector spaces depending upon the file size to trigger one of the other thing and along with it. I also plan to work on let’s say How do I tell you that a bubble so just like how there is a pop-up on desktop so constant pop-up or much more like the Apple assistance that is there on iPhones the dot also have it in on in the tray also have the application running in the back down in the tree and at the same time have a little bit of pop-up always showing up on the side of the screen which went triggered. Open the search web basic search search bar that you can directly query and within few seconds you can get a quick answer. So this is to typically trigger web Searchor depending upon what they want either one of them either web search or this or maybe both it should like a new what do we call it conversation without triggering the whole app but also to have a little bit of pop-up. It’s just like you’re gonna there’s gonna be a certain bar that’s showing up that’s like triggering and then you text you send your text in it and then it’s gonna give the answer with a slide. You know a slider AI desktop slider

### 

| **all-MiniLM-L12-v2** | 384 | 87% | Fast | 120MB |
| --- | --- | --- | --- | --- |

### 

| **all-MiniLM-L6-v2** | 384 | 85% | Fast | 90MB |
| --- | --- | --- | --- | --- |

### 

| **LLaMA2:7b** ⭐ | 3.8GB | 8GB | 85% | Medium |
| --- | --- | --- | --- | --- |
| **Gemma:3b** 🎯 | 2GB | 4GB | 82% | Fast |
| Mistral:7b | 4.1GB | 8GB | 87% | Medium |

### 

| **Database** | **Performance** | **Setup** | **Scalability** | **Features** |
| --- | --- | --- | --- | --- |
| **Qdrant** ⭐ | Excellent | Easy | High | Full-featured |
| **FAISS** 🎯 | Excellent | Medium | High | **Required option** |
| **ChromaDB** 🎯 | Good | Easy | Medium | **Required option** |

### 

| **Multi-Model Embedding** | Single model → Ensemble | +5% accuracy |
| --- | --- | --- |
| **Reranking System** | Simple similarity → Neural rerank | +8% relevance |
| **Query Enhancement** | Direct query → Expanded query | +10% recall |

### 

| **Feature** | **Description** | **Complexity** |
| --- | --- | --- |
| **Multi-Modal RAG** | Images + Text processing | High |

### 

| **Quantization** | FP32 → INT8 models | 4x faster, 4x less memory |
| --- | --- | --- |
| **Batch Processing** | Single → Batch embeddings | 3x faster indexing |
| **Async Pipeline** | Sync → Async processing | 2x throughput |
| **GPU Acceleration** | CPU → GPU inference | 10x faster |

also I forgot to cythonize properly the core files, so that it is increasing efficiency, idk where to add this so add it

and I am to make sure the core is cythonized for faster production and processing, and along with I,t another task is to make this into a production level, i.e., a deployable instantly working not in one but anyone laptop so need to dockerise this soo remember this that this is a production level soo