**Call with dhawan, Siddharth-20250814\_143900-Meeting Recording**

August 14, 2025, 9:09AM

37m 36s

 **dhawan, Siddharth** 0:04  
So English.  
I don't know.

 **Chauhan, Ankur** 0:07  
Perhaps speaking English.

 **dhawan, Siddharth** 0:10  
So function app is our entry point into our back end extraction service. So for now there is just one STTP trigger as of now like if user uploads a PDF and click on extract button.

 **Chauhan, Ankur** 0:16  
OK.

 **dhawan, Siddharth** 0:25  
And like API hits this function so HTTP trigger is called. After that we get the name name of the file which we have to run. So here we are just resolving the name and the body.

 **Chauhan, Ankur** 0:29  
OK.  
OK.  
OK.  
This name we are getting from the UI.

 **dhawan, Siddharth** 0:45  
So.  
UI yes, the PDF file name.

 **Chauhan, Ankur** 0:48  
OK.  
Similarly, this MGG 24 product key and all that OK.

 **dhawan, Siddharth** 0:53  
Yes body. This is the request body for that Jason like Jason can structure. I can show you that also. So this is the request body which we are sending from UI. This is the name of the file MDG product name and the keys and the interest information.

 **Chauhan, Ankur** 1:04  
Oh.  
Take.

 **dhawan, Siddharth** 1:12  
So this is extracted here and stored in these video tools.

 **Chauhan, Ankur** 1:13  
OK.  
OK.

 **dhawan, Siddharth** 1:20  
Then.  
This we are checking it is not needed for our end and after that our main function is process extraction in which we are passing each and everything getting from UI. Now in this process extraction we are importing PDF search engine.

 **Chauhan, Ankur** 1:34  
Yeah, OK.

 **dhawan, Siddharth** 1:40  
PDF search engine will have like all the functions for PDF loading like we have just called it and we know it. What it will do it will extract all the text from the PDF using libraries like PDF minor or PDF this library and.  
I think another library also.  
So this is a class to search the keywords in the ADF basically.

 **Chauhan, Ankur** 2:07  
OK.

 **dhawan, Siddharth** 2:08  
Here you can see how.

 **Chauhan, Ankur** 2:08  
So no. So here we're passing prompts also with the keywords. No, sorry, only keywords we're passing.

 **dhawan, Siddharth** 2:16  
We are first of all in this PDF search engine we are passing keywords with the its corresponding text. So this load load PDF function as I show shown you what it will do it will see it is appending the text of the page.

 **Chauhan, Ankur** 2:25  
OK.

 **dhawan, Siddharth** 2:35  
Pages in the PDF.

 **Chauhan, Ankur** 2:36  
Mhm.

 **dhawan, Siddharth** 2:37  
So this is whole function which will read all the pages and then read the text and append it in a one document.

 **Chauhan, Ankur** 2:38  
OK.  
OK.  
OK.

 **dhawan, Siddharth** 2:47  
Still.  
Here the load PDF is called. Here we are getting just the text and it will be stored like in this variable self dot document where it is. Yes it will be stored in self dot document text so when.

 **Chauhan, Ankur** 3:07  
So here we're just passing the like taking the text from the PDF and passing it. That's it. Nothing. No, no keyword currently, currently.

 **dhawan, Siddharth** 3:08  
Bose.  
Yes, yes, that currently no keyword. This line will just extract the text.

 **Chauhan, Ankur** 3:15  
OK.  
That's it. OK.

 **dhawan, Siddharth** 3:20  
And then we are creating one main function in which we are also giving all the information which we got from UI and we are passing the object instance of Indian.

 **Chauhan, Ankur** 3:28  
Mhm.  
OK.

 **dhawan, Siddharth** 3:34  
So this engine will have those extracted pages because it's the same object which we created and pass the name of the PDF.

 **Chauhan, Ankur** 3:41  
OK.

 **dhawan, Siddharth** 3:43  
And then we move to main. First step of the main is to get an instance of our GPT, so this instance will get loaded.

 **Chauhan, Ankur** 3:57  
Fine.

 **dhawan, Siddharth** 3:57  
Promise your.

 **Chauhan, Ankur** 3:59  
Mhm.

 **dhawan, Siddharth** 4:01  
And we have stored this variable in LLM variable. Then we are creating that prompt dictionary. This prompt dictionary is from this prompt manager file. This will have all the keywords, sorry all the keywords, queries and like basically.  
In these keywords, queries and the pedantic class.

 **Chauhan, Ankur** 4:26  
OK, we're passing IN, engine also there.

 **dhawan, Siddharth** 4:27  
So Pandit engine we are passing because we are using the search method from that start sorry from that that searcher in which our PDF classes made.

 **Chauhan, Ankur** 4:35  
OK.  
OK.

 **dhawan, Siddharth** 4:46  
So as I told you before that self dot document text will be appended when we call this function as the last line is this only this one. So if we pass object of that.

 **Chauhan, Ankur** 4:51  
Oh.  
Mhm, mhm.

 **dhawan, Siddharth** 5:01  
Engine in this dictionary it will contain all the text for that current PDF.

 **Chauhan, Ankur** 5:08  
Fine.

 **dhawan, Siddharth** 5:09  
So in this engine calling we are using search method. The search method it will extract the date keyword along with the context context words.

 **Chauhan, Ankur** 5:22  
OK.

 **dhawan, Siddharth** 5:23  
So 10 words to the left, 10 words to the right.  
So this is how context words work and after that this logic I'll explain later like it is for the extraction. Currently we do not need it.

 **Chauhan, Ankur** 5:30  
Oh, fine.  
OK, fine.

 **dhawan, Siddharth** 5:43  
So now second thing we have gotten this dictionary. This is the full dictionary. Then we have.

 **Chauhan, Ankur** 5:49  
Uh, OK.

 **dhawan, Siddharth** 5:53  
Got this dictionary here and then we will be processing the prompts. Now this processing prompts is present in startpy. This processing prompts is taking prompt dictionary and LLM as the arguments.  
So what here we are doing? We are creating empty list so that we can append each and every prompt for batch processing.

 **Chauhan, Ankur** 6:10  
Hmm.  
OK.

 **dhawan, Siddharth** 6:20  
So here the batching is happening. First we are iterating our prompts and or I in a dictionary means we are passing the key. I is key here so dictionary. If we do dictionary key we will get all these.

 **Chauhan, Ankur** 6:32  
OK.

 **dhawan, Siddharth** 6:38  
Uh, these uh, three things.

 **Chauhan, Ankur** 6:41  
OK.

 **dhawan, Siddharth** 6:43  
When I pass this 99, I'll get uh a list of this and.

 **Chauhan, Ankur** 6:47  
OK.

 **dhawan, Siddharth** 6:51  
And prompt info will have list of these three things. So we have written these three things, parenting class, prompt function, field data.

 **Chauhan, Ankur** 6:55  
OK.  
OK.

 **dhawan, Siddharth** 7:00  
So now in this three list we are appending it and this is a batching logic. So currently in this code batch size is the maximum. Maximum means it is equal to the length of the parameter dictionary.

 **Chauhan, Ankur** 7:08  
Bye.

 **dhawan, Siddharth** 7:19  
So what it will do it will process all the prompt at once. So if you have to change or configure like only 10 prompts should run in parallel so we can just customize this if like we can add percentage 10.

 **Chauhan, Ankur** 7:34  
So is there any any?

 **dhawan, Siddharth** 7:36  
Um.

 **Chauhan, Ankur** 7:40  
Upper limit on the number of forms IN, the batches. Is there any you encounter?

 **dhawan, Siddharth** 7:46  
Upper limit it will be set on the Azure side. When we create service there we can set a rate limit option and token per minute option like how many Max tokens we have to allow and Azure has some of their own limits as well if we have not set it.  
So current.

 **Chauhan, Ankur** 8:06  
Earlier I heard that it was like 20 prompts. We were you started the 20 prompts badges, right?

 **dhawan, Siddharth** 8:11  
Yes, that time I started with 20 prompts. That was due to like we were processing 10 PDFs parallely. So if 10 PDFs and if we have 100 prompts and 10 PDFs, 1000 prompts will hit at once.

 **Chauhan, Ankur** 8:19  
OK.  
OK.

 **dhawan, Siddharth** 8:26  
So it can cause rate limit. So if we solve this issue concurrency like we can create multiple APIs and separate the load so we can just use all the dictionary at once. It will be it will complete around 30 seconds.

 **Chauhan, Ankur** 8:28  
OK.  
Um.  
So that batch processing Some, of the code which might be written here later, right? It is the IT is the point where we are actually sending it and here is like whatever manual code we might have to create for the different APIs. Here we need to the maximum edit, right?

 **dhawan, Siddharth** 9:00  
He.  
I think yes, because this will be processed by this extract and process data. So in this the LLM is being called so here.

 **Chauhan, Ankur** 9:11  
OK.  
That means OK.

 **dhawan, Siddharth** 9:18  
So here you will write the logic if you have to send 20 prompts to DLM, 40 prompts or all the prompts. So here you can decide it. So if you want to decide and on which API it should hit, I think like.

 **Chauhan, Ankur** 9:24  
Mm-hmm. OK.  
Mhm.

 **dhawan, Siddharth** 9:33  
It should be added in this extract data function.

 **Chauhan, Ankur** 9:38  
Uh, OK.

 **dhawan, Siddharth** 9:40  
Because here we are using this prompt template and then LLM dot batch we are calling the prompts. So so basically I don't think here we should edit. We should be editing in the function app itself.

 **Chauhan, Ankur** 9:48  
OK.  
Mhm.

 **dhawan, Siddharth** 10:07  
Create a list of LLM. So we have 10 GPT 4 parallel running. So here this process prompts.

 **Chauhan, Ankur** 10:09  
Huh.  
Yeah, that all we have to pass in the in this process prompt and there we need to do the round Robin scheduling or whatever required.

 **dhawan, Siddharth** 10:20  
Uh.  
Yes, whatever we can. I think that will work.  
Yes.

 **Chauhan, Ankur** 10:26  
OK.

 **dhawan, Siddharth** 10:30  
So extract data where we will we will start this function.  
So this is just the batching logic. If we have multiple instances I my suggestion would be take all the prompter at once and hit just at once only.

 **Chauhan, Ankur** 10:46  
Hmm.  
Mm-hmm. OK.

 **dhawan, Siddharth** 10:49  
Oh.

 **Chauhan, Ankur** 10:51  
OK, after that we go.

 **dhawan, Siddharth** 10:53  
So after that I think I should explain this extraction process data. So extraction process data is just calling this function and we are moving into this function. Now this is the main kind of function which will actually call the LLM.

 **Chauhan, Ankur** 10:55  
Hello.  
Extract and process, yes.

 **dhawan, Siddharth** 11:11  
With the context data, so here we are getting the list.

 **Chauhan, Ankur** 11:17  
OK, fine.

 **dhawan, Siddharth** 11:18  
Each list like if there were 20 prompts, it will contain 20 prompts only and if all then all will be present. Now what? Yes this is the batch and with the LLM instance we initiated.

 **Chauhan, Ankur** 11:21  
Mm-hmm.  
This is the badge.  
OK, OK.

 **dhawan, Siddharth** 11:34  
So after this we are doing a for loop of zipping these three list and then we are IN, like using this piedentic output parser as we have written the code IN, our piedentic class.

 **Chauhan, Ankur** 11:49  
Mhm.

 **dhawan, Siddharth** 11:49  
To extract the corresponding values. So we are using this padentic output parser and we are passing the padentic class.

 **Chauhan, Ankur** 11:59  
OK.

 **dhawan, Siddharth** 12:00  
So.  
Wait, what is the? So we are using for loop here so one by one, but this will be initiated.  
So and one by one this prompt template will be initiated. Why we are doing one by one? Because we have different contents data for different different prompts, right? So this prompt function, this prompt function we have to call for.

 **Chauhan, Ankur** 12:15  
Oh, OK.  
Oh yeah.

 **dhawan, Siddharth** 12:35  
Each and everyone.

 **Chauhan, Ankur** 12:35  
OK.

 **dhawan, Siddharth** 12:37  
So we are creating prompt for each and every 100 prompt. If you say so hundred of these variables will be created and this format. Now here the context data from the PDF will be passed to the prompt.

 **Chauhan, Ankur** 12:53  
OK, field tracks is equal to field data from where the field data is coming.

 **dhawan, Siddharth** 12:54  
Did.  
Field data is coming IN, this for loop. We have extracted this field data from our main this where is it prompt manager.

 **Chauhan, Ankur** 13:00  
OK.  
Oh, OK.  
Got it.

 **dhawan, Siddharth** 13:10  
So this is our field data.

 **Chauhan, Ankur** 13:13  
OK.

 **dhawan, Siddharth** 13:14  
And we are appending it here.

 **Chauhan, Ankur** 13:17  
Oh, thank God. Thank you. Thank you.

 **dhawan, Siddharth** 13:17  
Text.  
We are appending it here and we are appending the instruction also for instruction we are passing it here and for the text we are passing it here.

 **Chauhan, Ankur** 13:28  
Uh.  
Oh, got it, got it.

 **dhawan, Siddharth** 13:38  
So this uh token length also been calculated here.  
Uh.

 **Chauhan, Ankur** 13:44  
But if you're using the for loop prompt by prompt, how come it is batching them?

 **dhawan, Siddharth** 13:46  
Uh huh.  
Yes, the I'll explain. We are not calling LLM right now, we are just prepare pre processing the data. We are this completing this query as we have two variables, text and format instruction.

 **Chauhan, Ankur** 13:56  
OK, OK. OK.  
Oker.  
What we are passing into the LLM. We are preparing that now, OK?

 **dhawan, Siddharth** 14:08  
Yes, we are preparing that C input prompt dot append. We are doing so we just need prompts for the LLM. LLM doesn't need anything. So IN, prompt we are appending our context data and appending our format instruction. Format instructions are for Jason output, structured output.

 **Chauhan, Ankur** 14:12  
Mhm.  
Bye.  
Fine.

 **dhawan, Siddharth** 14:27  
So now I have the problems.

 **Chauhan, Ankur** 14:27  
Bye.  
Mm-hmm. Got it.

 **dhawan, Siddharth** 14:31  
So it will be equal to the prompts you have passed from the start file. If 20 then this will have 20 prompts. Yes, if all prompts will be there then 100 of 100 prompts will be there. Then we are calling it batch wise. So this occurs outside the for loop.

 **Chauhan, Ankur** 14:37  
It will be 20, yeah.  
OK.  
Bye.  
Uh, that should.

 **dhawan, Siddharth** 14:51  
Yes, so LLM dot batch. This LLM is the instance of langchain. So langchain has already implemented batching. So if you have to implement batching on your own then we had to do that async kind of things or parallel processing but as.

 **Chauhan, Ankur** 14:58  
Hmm.  
OK.  
Yeah, yeah, yes.

 **dhawan, Siddharth** 15:11  
Since they have already done it, we do not need to. If we do it, it will cause an error.

 **Chauhan, Ankur** 15:16  
OK.

 **dhawan, Siddharth** 15:17  
So yes.  
This all the outputs we have extracted at one go and now again we are using for loop. Why for loop? Because we have we will be having 20 or 30 outputs according to our batch size. Then we have to parse those output. Parsing means because LLM will answer.

 **Chauhan, Ankur** 15:24  
Bye, bye.  
Mm.  
Yeah.

 **dhawan, Siddharth** 15:40  
With Some, text and adjacent LLM won't be just giving us the Jason, it can give us anything but IN, text it will return adjacent format also. So what we are doing, we are just parsing the output from this pedantic output parser.

 **Chauhan, Ankur** 15:43  
Hmm.  
Yeah.

 **dhawan, Siddharth** 16:00  
And this parse content, we are appending it IN, a list.

 **Chauhan, Ankur** 16:08  
OK.

 **dhawan, Siddharth** 16:08  
So this best output dot append we are appending IN, it and returning it. So we are returning this extract data from like LLM has given this best output and we have passed it and we are giving it to our start.

 **Chauhan, Ankur** 16:26  
OK.

 **dhawan, Siddharth** 16:28  
This one and now this one will be called.  
So we will get this data here.

 **Chauhan, Ankur** 16:35  
OK.

 **dhawan, Siddharth** 16:35  
So for patching at last we are extending it here extend. When we extend IN, a list we can just like if I have 20 browns 20 + 20. If we have to do we have to extend it. If I use append it then it will create a.  
List of list, but we need just a linear list, so that's why we are using extend here. When we have to append 20 elements at once IN, an existing list, we have to use extend. So this is part of the batching logic to the start and this is clear right?

 **Chauhan, Ankur** 16:57  
OK.  
OK.

 **dhawan, Siddharth** 17:14  
Now we have all the prompt outputs with us.

 **Chauhan, Ankur** 17:18  
Mm-hmm.

 **dhawan, Siddharth** 17:20  
No, we will. We are going with the retrieve data. Retrieve data is basically we are using this pass output as well as all the information came from UI and this retrieve data.

 **Chauhan, Ankur** 17:21  
OK.

 **dhawan, Siddharth** 17:37  
We are doing neuro uh.  
These will be directly assigning uh the values of this file name, product name. This is the column names. This is their corresponding values.

 **Chauhan, Ankur** 17:48  
We're we're creating the intermediate file now.

 **dhawan, Siddharth** 17:51  
Yes, intermediate LLM, uh, intermediate file, right?

 **Chauhan, Ankur** 17:53  
OK.  
Yes, OK.

 **dhawan, Siddharth** 17:57  
Then what we are doing for object IN, parse data. Now we will be having our dictionaries IN, it dictionary because pydentic class gives us dictionary right?

 **Chauhan, Ankur** 18:12  
Hmm.

 **dhawan, Siddharth** 18:12  
So this dictionary we are using vars object items. What it will do it will our data will be IN, the form of this payment condition and answer auto call condition and answer.

 **Chauhan, Ankur** 18:29  
OK.

 **dhawan, Siddharth** 18:29  
So while doing VARS on object like this, it will take the field value and value. So what it will do it will extract this information and this corresponding answer which LLMM extracted.

 **Chauhan, Ankur** 18:44  
OK.

 **dhawan, Siddharth** 18:46  
So it will just append the field value.  
And new row will append the value. Why we appended out? Because field will be our name and name will be same as these pidantic classes IN, each and everything. These will be our name of our columns.

 **Chauhan, Ankur** 18:50  
OK.  
OK.  
Yeah, yes, OK.  
Hm.  
OK.

 **dhawan, Siddharth** 19:07  
And its corresponding values. So we are creating just a one row thing like we are like number of these these many things will be equal to our columns.

 **Chauhan, Ankur** 19:20  
Huh. OK.

 **dhawan, Siddharth** 19:23  
And its corresponding values. So here we have parsed everything. Sometimes parsing fails. LLM fails to give us Jason.

 **Chauhan, Ankur** 19:25  
Although.  
OK.

 **dhawan, Siddharth** 19:33  
So this is for error handling. For Some, fields it might fail.

 **Chauhan, Ankur** 19:40  
OK.

 **dhawan, Siddharth** 19:40  
And then we are creating the data frame with the new row and this column output columns.  
So here we will be getting our intermediate DF.

 **Chauhan, Ankur** 19:55  
OK.  
Bye.

 **dhawan, Siddharth** 19:57  
And after that retrieve data is finished.

 **Chauhan, Ankur** 20:26  
OK.  
OK.

 **dhawan, Siddharth** 20:26  
All the data extracted from LLM that intermediate file we call is this DF. Now we have to compute some functions uh to process this data.

 **Chauhan, Ankur** 20:31  
Mhm.  
Plan.  
Hmm.

 **dhawan, Siddharth** 20:40  
So we are using this convert to ME. This convert to ME file we have imported from ME formatter. For this refactor project we have created ME formatter to a new formatting output thing.

 **Chauhan, Ankur** 20:44  
OK.

 **dhawan, Siddharth** 20:56  
Which I'm going to explain now.  
So this convert to MEV need our intermediate output.

 **Chauhan, Ankur** 21:06  
Yeah.

 **dhawan, Siddharth** 21:08  
And we need this ME data frame. This ME data frame is our UI thing. In UI we need these things. These all fields are required. So what we are doing, we are first creating a data frame out of it.

 **Chauhan, Ankur** 21:16  
Mm-hmm.  
OK, OK.

 **dhawan, Siddharth** 21:24  
And returning this defined data frame.  
So here we are getting intermediate file and empty data frame currently.

 **Chauhan, Ankur** 21:35  
Oh, yeah.  
You order.

 **dhawan, Siddharth** 21:41  
So now what we are doing, we are creating an ME formatter class and we are passing these two things ME formatter class then. So we have created a class and if you need GLOBAL, variables you can just define IN, IN, it so that each and every function can use these variables.

 **Chauhan, Ankur** 21:49  
OK.  
Mhm.

 **dhawan, Siddharth** 22:01  
And we have written this Some, logic of file extraction and these issuer GK and each and everything. So this I don't have to explain right each and MGG functions are written.

 **Chauhan, Ankur** 22:14  
Yeah, yeah, that's OK.  
No, no.

 **dhawan, Siddharth** 22:19  
Oh.  
So after this convert to AMI we have finally gotten RDF. So suppose we have to create another service like UI chain service.

 **Chauhan, Ankur** 22:31  
OK.  
Mm-hmm.

 **dhawan, Siddharth** 22:39  
So you IN, UI change service this is the core logic. But one thing we need is UI change list. What if someone change changes a value? Suppose this column GKT?

 **Chauhan, Ankur** 22:54  
Mhm.

 **dhawan, Siddharth** 22:56  
We need exact column name which is present IN, this ME data frame.

 **Chauhan, Ankur** 23:02  
OK.

 **dhawan, Siddharth** 23:04  
This one.

 **Chauhan, Ankur** 23:05  
Mhm.

 **dhawan, Siddharth** 23:06  
And we have created a map.  
Function map IN, which this is mapped to the corresponding function.

 **Chauhan, Ankur** 23:16  
OK.

 **dhawan, Siddharth** 23:17  
Present IN, ME formatter. These right side functions are all present IN, ME formatter and left side left side functions are IN, UI as well as IN, data frame files that I showed you earlier this one.

 **Chauhan, Ankur** 23:19  
OK.  
Hmm.

 **dhawan, Siddharth** 23:33  
And after getting a list of changes, what we are doing, we are.

 **Chauhan, Ankur** 23:37  
But where? Well, just one more thing where we are like mentioning that this value has been changed to this.

 **dhawan, Siddharth** 23:45  
Yes. So this we are not mentioning actually we have thought of a process to identify it. What we will be doing, we will be saving our Jason when we are running it for first time, right this one and this loop won't be present here, it will be a new.

 **Chauhan, Ankur** 23:56  
Yeah, OK.  
Mhm.

 **dhawan, Siddharth** 24:05  
New URL. So after this returning the Jason we this Jason will go to UI and what we are trying to do we will compare those both Jason.

 **Chauhan, Ankur** 24:06  
Yeah, OK.  
Mhm.  
OK.

 **dhawan, Siddharth** 24:19  
So one saved old Jason and new Jason and we will get our result. So from there we can get this or other case can be we can tell UI guide to send us these things if if it is sent from UI.

 **Chauhan, Ankur** 24:20  
Mm-hmm.  
OK.  
Mhm.

 **dhawan, Siddharth** 24:37  
Then it won't be a big problem to get us like we need to save. We don't need to save anything then.

 **Chauhan, Ankur** 24:43  
I got it that I got my. Actually my question was that the value change but value change will automatically reflect in the Jason which is coming back.

 **dhawan, Siddharth** 24:47  
Uh huh.  
Yes.  
Yes, yes, it will be reflected because it will saved IN, UI.

 **Chauhan, Ankur** 24:56  
OK, but here higher how you implemented it.

 **dhawan, Siddharth** 24:58  
So.  
So here what what I implemented like this one.

 **Chauhan, Ankur** 25:05  
OK, this this I this I think this is the 95th line where you're mentioning the.

 **dhawan, Siddharth** 25:09  
Yes, yes this is the change change to be done IN, UI. So if UI person tells us this thing then we do not need to save anything.

 **Chauhan, Ankur** 25:11  
Oh, OK.  
OK.  
Hmm, OK.

 **dhawan, Siddharth** 25:24  
And if we do not get thing, we need to save 2 Jason's old Jason and new Jason and then compare and detect this change.  
And run the new Jason through this file.

 **Chauhan, Ankur** 25:35  
Mhm.

 **dhawan, Siddharth** 25:38  
So before that we need to create a mapping. Also like if this function is changed, we need to execute Some, other function because it can be dependent on other fields. So this mapping DF we are using this field name mapping excel.

 **Chauhan, Ankur** 25:38  
OK, OK.

 **dhawan, Siddharth** 25:56  
This field name mapping Excel what it will do it will go to this function. After this function we are we are creating an execution list. Execution list will create these a list of these function if I have done two or four 2-3 changes.

 **Chauhan, Ankur** 26:08  
OK.

 **dhawan, Siddharth** 26:16  
Maturity date also then it would be also be appended. So this execution list will be created. After that we are loading a dependency map. Dependency map is from function under score map.

 **Chauhan, Ankur** 26:21  
OK.

 **dhawan, Siddharth** 26:33  
So here we will be running those functions like GKG as I shown earlier this function. If this function is executed, we need this, this and this function correspondingly because.

 **Chauhan, Ankur** 26:47  
OK.

 **dhawan, Siddharth** 26:49  
In detail sub classification GK key is used. If I change that so it it has to be updated and this scheme also has to be updated and this GKUS this is US case. GKUS field has to be called.

 **Chauhan, Ankur** 27:05  
Yeah.

 **dhawan, Siddharth** 27:06  
So this is just for US. That's why I mentioned IN, US IN, the function. So suppose maturity rate was also called then instrument short name has to be changed, suffix has to be changed, taxation and report. So basically we are loading this map.  
And we are passing DF and DFME IN, the ME formatter function. We are creating again this ME formatter fully.

 **Chauhan, Ankur** 27:35  
Mhm.

 **dhawan, Siddharth** 27:39  
So after that.  
OK.

 **Chauhan, Ankur** 27:46  
It's the best OK NP map then OK army formatter.

 **dhawan, Siddharth** 27:47  
Mm.  
So what it will do, it will create this corresponding class and initialize with these values only. It will not run fully because we have not called functions yet.

 **Chauhan, Ankur** 27:56  
Mhm.  
OK.  
Mm-hmm. Yeah. OK.

 **dhawan, Siddharth** 28:07  
So functions will be called here.  
And first of all we need field name to function mapping. What this function is doing it is returning as the function from the me formatter.

 **Chauhan, Ankur** 28:14  
Oh.  
OK.

 **dhawan, Siddharth** 28:26  
It will.  
Use this name.  
Get attribute meformatter name and it will same save it as a dictionary name of the function and the function object.

 **Chauhan, Ankur** 28:36  
Hmm.  
OK.

 **dhawan, Siddharth** 28:46  
And how it will do? It will iterate over the directory DI. When we do a DIR on a class, it will list down all the functions present IN, and it will check if it is a function. Callable returns a function and it will also check attribute.

 **Chauhan, Ankur** 28:58  
OK.

 **dhawan, Siddharth** 29:06  
Uh, this get the named attribute from object.

 **Chauhan, Ankur** 29:09  
OK.

 **dhawan, Siddharth** 29:10  
So what it will do it will take name and me formatter and we are excluding IN, it and private methods. So if a method is started with double under score it won't get like get IN, this field name to function mapping.

 **Chauhan, Ankur** 29:21  
OK.

 **dhawan, Siddharth** 29:30  
Like if user wants to like private its function like make the function private like init kind of things and other things so it won't become here coming here. So this will return a dictionary of.

 **Chauhan, Ankur** 29:30  
OK.  
OK, OK.

 **dhawan, Siddharth** 29:46  
This maturity date and its corresponding object from this.

 **Chauhan, Ankur** 29:53  
OK.

 **dhawan, Siddharth** 29:54  
Sure.  
So this function will be stored along with its string name.

 **Chauhan, Ankur** 30:06  
Hmm.  
OK.

 **dhawan, Siddharth** 30:13  
So IN, this we have now got gotten a dictionary dictionary of functions which we need to update our TFME file.

 **Chauhan, Ankur** 30:24  
OK.  
Oh.

 **dhawan, Siddharth** 30:35  
And execution list and field names. This will contain the functions itself. This will contain only the names which we have to execute from field name and this will be having the dependencies which is present in this field name. Sorry.

 **Chauhan, Ankur** 30:35  
Oh.

 **dhawan, Siddharth** 30:53  
Uh, sorry, function mapping.

 **Chauhan, Ankur** 30:56  
OK.

 **dhawan, Siddharth** 30:59  
Now we are moving into our queue. In this queue NP map value dispatcher and reextract. Now we do not need reextract so we just need three values NP map value. First of all value is being used IN, execution queue we are appending.

 **Chauhan, Ankur** 31:12  
Mm-hmm.

 **dhawan, Siddharth** 31:19  
Appending it IN, it. So we are appending our basically execution queue. So we cannot randomly call ABCD. We have to 1st resolve the dependencies. So we are putting IN, execution queue and we are using.

 **Chauhan, Ankur** 31:27  
Hmm.  
Mhm.

 **dhawan, Siddharth** 31:39  
The Q method for graph dependency resolution.

 **Chauhan, Ankur** 31:45  
OK.

 **dhawan, Siddharth** 31:45  
So here is the logic for that while execution queue and then we are popping out the first element.  
And.  
Sorry rare element we are popping out and so pop zero will I think it will pop first element only right?

 **Chauhan, Ankur** 32:04  
That's OK. It's we'll leave the logic which is IN, the end from the queue, we'll get the processing queue, execution queue IN, the end which have to execute right. That is it.

 **dhawan, Siddharth** 32:06  
Uh.  
Uh.  
Yes, we we are. We are executing the function here.  
Dispatcher Q function and this parenthesis. So here the function is called.

 **Chauhan, Ankur** 32:18  
OK, we are not OK.  
So we are executing it as the function has been added to the queue. We wait for the end of the recursion to execute everything.

 **dhawan, Siddharth** 32:30  
No, no, no. We are just executing as we are getting the function.

 **Chauhan, Ankur** 32:34  
OK, OK, fine.  
But then the visited also been if you're executing it on the go then it will not affect the like it should not go into the loop right?

 **dhawan, Siddharth** 32:44  
Hmm.  
No, it it won't be going into the loop because we are using this widget set.

 **Chauhan, Ankur** 32:55  
OK, fine.

 **dhawan, Siddharth** 32:56  
Should.  
We are if you function not IN, visited then only this it will go into if and then only it will execute.

 **Chauhan, Ankur** 33:03  
Hmm.  
OK, fine, got it.

 **dhawan, Siddharth** 33:12  
And then after this we are just adding visited add Q function and as it is a set it cannot come twice or it cannot be get it cannot get stuck IN, bloom.

 **Chauhan, Ankur** 33:20  
OK.

 **dhawan, Siddharth** 33:28  
Due to this visited.

 **Chauhan, Ankur** 33:31  
OK.

 **dhawan, Siddharth** 33:32  
And then this queue is implemented and as we have passed this what we can say this execution list with field name, this field name.  
Was the what we can say this is this was the object of our class, right?

 **Chauhan, Ankur** 33:53  
Mm-hmm.

 **dhawan, Siddharth** 33:54  
If I am using the object of the class and running the functions using those objects, this this is the kind of object of each and every function of that class we have instantiate like we have initiated it.

 **Chauhan, Ankur** 34:02  
Mhm.  
Hm.  
OK.

 **dhawan, Siddharth** 34:11  
So we do not need to return anything as the function. It will as you can see our ME formatter. As you can see our ME formatter, it doesn't return a thing, it just saves IN, this GLOBAL, variable DFME.

 **Chauhan, Ankur** 34:29  
Yeah.

 **dhawan, Siddharth** 34:29  
Which is present IN, IN, it. So we can just directly take object dot DFME it's variable and save it IN, DFME.

 **Chauhan, Ankur** 34:32  
OK.  
OK.

 **dhawan, Siddharth** 34:41  
So our changes will be updated here and now this DFME will be converted to a group data.

 **Chauhan, Ankur** 34:52  
OK.

 **dhawan, Siddharth** 34:53  
This group data what it will do it will create the group wise MG like 21 all the data will be here then 111 then data will be here. Although the current thing is quite like different now because we have included tabs and everything so this logic has.  
To be changed.  
So this group logic has, uh, totally changed.  
We need to import from the new code when we are doing any change here.  
And then we are converting the types. Sometimes an error was coming like it cannot be converted to Jason because of Some, numpy numbers or Some, issues. So for the safer side we will be removing after the migration.  
This convert to data types because sometimes while implementing these functions we like save sometime numpy objects or Some, objects which are not converted to Jason. That's why.

 **Chauhan, Ankur** 36:04  
Mhm.

 **dhawan, Siddharth** 36:05  
It is being made for now. We can just test it.

 **Chauhan, Ankur** 36:11  
OK.

 **dhawan, Siddharth** 36:13  
And then we are just converting to Jason and dumping it.

 **Chauhan, Ankur** 36:19  
Return and where we are returning it.

 **dhawan, Siddharth** 36:22  
We are just returning process extraction. So process extraction is our API and Jason data. This STTP trigger this API. It will send out the Jason to the to our back end API back end.

 **Chauhan, Ankur** 36:23  
To which?  
OK.  
OK.

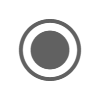
 **dhawan, Siddharth** 36:39  
And other thing, anything other than it, I don't think so.

 **Chauhan, Ankur** 36:44  
No, this is enough actually. And one question I have OK if I'm using multiple alarm instances I will have to also put Some, IDO tag right on that.

 **dhawan, Siddharth** 36:46  
The rest everything is same.  
Mm.  
Hmm.

 **Chauhan, Ankur** 37:01  
Request I'm sending from which user it is going and then I have to resolve it back when it is coming back.

 **dhawan, Siddharth** 37:03  
Yes.  
Uh huh.  
I think yes, maybe it will be required like basically IN, the new function app they are giving us the username, container ID also everything they are giving.

 **Chauhan, Ankur** stopped transcription