The frontend is probably the source of most bugs at this point.

Throughout the javascript, the functions get and set the global state value, which likely works alright, HOWEVER, when the state value is set, there is also the option to make it sync with the backend as well. This may or may not be happening everywhere where it should be happening.

When the page loads the global state frontend object needs to be initialized with values that are fetched from the backend. There may be places where this is called erroneously, or where it is not called when it should.  
  
There are a couple bits of redundant code that should be refactored to reduce (indicated in the CODE\_REVIEW document).  
  
There is also dead code on the frontend having to do with fetching transcripts, console messages, and the like. They try to run but cannot. This should not actually be causing any problems elsewhere, but is just something to be aware of.  
Basically, what we need to do is update the backend so that things like x\_poll transcripts, auxiliary duo transcripts, and whatever other tool outputs we want to expose on the frontend actually need to not only get saved to a database, but should specifically get saved to some attribute associated with the USER model.

* So the user model needs to be updated for that, then the various tools whose outputs we want to store there need to call some new methods that will do that storing (this likely means that we also need to refactor most of the backend logic minorly to literally pass the USER object through the entire chain of chat logic and function calls, NOT just the client initialized with their api key as it is now)
* And then we need to update those frontend functions that are trying to fetch those transcripts or console messages to actually point at new endpoints which then check the user model for new stuff, and return it. (the transcripts are things that get stored in that side window, and the console messages are actually things that when retrieved, interrupt your chat, and send the console message to the assistant you are talking to, and then displays both console message and assistants reply to it in the chat window once all received… so idk maybe that needs to change)

As already mentioned, the backend logic i think should be updated so that when the USER objects are instanced from db (which is first retrieved inside of chat\_services functions using the token from the request data and the app.config passed in from the route) they are actually passed in directly to the giint service and subsequent run\_giint and then any and all function calls that precipitate from that. Currently, the user object is only used to then extract the api key, and that api key is then only used to initialize an instance of the openai client, and then that open ai client becomes an attribute of the giint service, and then is passed through all of the logic, so that anytime you need the client again (either for a direct call, or to set up a new giint service) you always have a client that has been united with the user’s api key…  
HOWEVER… this si not good enough, and we need to pass the entire user through everything i think because!!!...

What we want to be able to do is save the giint\_threads and giint\_agents SPECIFICALLY to some attribute on the user object db model.   
At present the giint\_threads and giint\_agents are just saved to a json. And while it is technically linked to a user, because the user has a state attribute, and that state attribute should have a ‘sessionId’ key, which should then have as values all of the giint\_id (corresponding to giint\_threads) that teh user has caused the creation of in the past. And furthermore, those giint\_threads will have as attributes agent\_ids of various orders that are themselves saved in json.  
So you can trace it all the way through, but TBH i don’t think that is right. I think each user’s giint threads and giint\_agents should be saved according to user (they can be saved generally TOO), which would then allow us to search through the threads that they have, but more importantly through the agents that they have, which of course have assistant\_ids as attributes that ARE specific to that user’s open ai account.  
  
The reason for all that is because currently the way the giint\_agents get made it recreates a new agent for you in every new giint\_thread, so when you start a new chat on front, and start talking, the new giint\_id will be for a new giint\_thread, and i DO NOT have logic to reuse a previously made agent for that. So instead a new agent gets made, and that then makes a new assistant on your account.  
So then your account explodes with assistants after use.  
This is going to be very easy to fix, but it is a core failing of the current system.

NOTE: one possible other solution would actually be to make it so that agents, when not being used, actually just save all the relevant data, including specialized non-templated instructions that get made for them, but that the actual assistant ALWAYS gets deleted from the user’s openai account. So that way when the agent wakes back up again for use, it re-instances the necessary assistant on open ai, and assigned that id to itself, and uses it for the chat logic, but that whenever it is done, the actual assistant would be cleared way, and only the data needed to recreate it is saved.  
I am honestly a fan of this idea, cause it just simplifies the whole thing in my mind, and we don’t ever get ourselves into a situation where we need to “check” the assistants actually present on someones account to make sure it is there, or is/is not the same as one used before, or has or doesn’t have what we need etc.  
Also it allows us to update the agent classes, and add tools and whatever, and then never have to think about making sure that the associated assistants also get updated. Cause the assistant is just abstracted into the agent.  
And this would still be able to handle any pre-existing assistants that a user wanted to use, but could have a method to immediately ingest the assistant data into the saved agent schema when used for the first time, so that you could then delete or change the assistant, and would actually still have the agent version that awoke as that original assistant the way you had it the first time.  
Anyway, it’s not like we have to do it this way, but i really can think of a million reasons why i think it would be easier…