

SetupAI Solution Architecture

## Summary

This document summarizes the solution architecture of SetupAI, including objects, custom metadata types, Apex and Lightning Web Components.

## Model

### Objects

* **Chat**
  + Represents a chat thread between the end user and the AI
  + Title is only field used(some unused/deprecated fields)
* **Message**
  + Represents an individual message in a chat thread
  + Fields
    - Thread: Lookup to Chat.
    - Role: Role of entity that sent the message(can be either ‘user’ for end user, ‘assistant’ for AI generated response, ‘function’ for AI generated function call, or ‘error’)
    - Content: Body of message
    - Function Name: Name of function being called by AI(only for function messages)
    - Function Arguments: Arguments to pass into function as JSON object(only for function messages)

### Platform Events

* **Message Notice**
  + To be published whenever an AI generated response is received
  + User interface is subscribed to this in order to reload message thread for end user
* **Async Error**
  + To be published whenever an exception is throw in an async job
  + User interface is subscribed to this in order to display exception and not stay stuck on loading screen

### Custom Metadata Types

* **API Secret**
  + Represents an OpenAI API Key generated and provided by SetupAI
  + Specifies SetupAI preferred GPT model
  + Protected type
* **Endpoint**
  + Represents an endpoint in an API call
  + Protected type
* **Function**
  + Represents a function that can be called by gpt-4 to gather information on Salesforce org.
  + Protected type
* **Function Property**
  + Represents a parameter that can be passed into a given function
  + Protected type
* **System Prompt**
  + Represents a system message, giving an AI chat all the info it needs to interact with a user and have knowledge about SetupAI’s functionality
  + Protected type
* **URL**
  + Represents a URL endpoint in Salesforce so that accurate URLs can be displayed to the end user
  + Protected type

### Custom Settings

* **Token Usage**
  + Protected Hierarchy setting used to save token settings around monthly allowance usage and whether or not a backup token should be used
* **GPT Settings**
  + Public Hierarchy setting that customers can use to supply their own API key(this allows them to bypass the token allowance) and set their own gpt model.

### Feature Parameters

* **Input Tokens**
  + Subscriber to LMO
  + Number of Input Tokens used this month
  + For reporting purposes
* **Input Token Price**
  + LMO to Subscriber
  + Price of Input Tokens(per 1000 tokens, number of ten thousandth cents)
  + Used to dynamically set price if updated by OpenAI
  + Default 100
* **Output Tokens**
  + Subscriber to LMO
  + Number of Output Tokens used this month
  + For reporting purposes
* **Output Token Price**
  + LMO to Subscriber
  + Price of Output Tokens(per 1000 tokens, number of ten thousandth cents)
  + Used to dynamically set price if updated by OpenAI
  + Default 300
* **Token Spend Limit(in USD)**
  + LMO to Subscriber
  + Monthly Token Allowance
  + Default 40
* **Is Using Custom API Key**
  + Subscriber to LMO
  + Set to true when customer is using their own API key
  + For reporting/budgeting purposes

### Remote Site Setting

https://api.openai.com

## Controller

*(For more info on OpenAI api and how chat messages are managed and generated, visit https://platform.openai.com/docs/guides/text-generation/chat-completions-api)*

### Apex

* **MessageTrigger**
  + After Insert
  + Enqueues *MessageTriggerHandler* queueable job(limits permitting) for each record
* **MessageTriggerHandler**
  + Queueable class that is fired on Message insert. Flow of logic is determined by Role
    - Role is user(message is submitted by user in UI) or function(results of function call are saved)
      * Appends message to end of existing chat thread
      * Submits thread to OpenAI chat API(via ChatInterface class)
      * Saves response as new Message record, which will subsequently enqueue new job
    - Role is assistant(AI response saved) or error(error message was submitted by app)
      * If assistant is calling function, name, arguments and chat Id are passed into Function.execute() method
      * If not function, Message\_Notice\_\_e platform event is published, which refreshes UI
  + If exception is thrown, it is caught and published to Async\_Error\_\_e platform event
* **ChatInterface**
  + Used to interface other classes with AI chat functionality. Expects array of ChatRequest.ChatMessage instances and returns single generated ChatRequest.ChatMessage response.
  + Calls OpenAI.chat() which returns a ChatResponse wrapper instance. Used tokens are updated in FeatureParameters. If a valid message was returned in the ChatResponse it’s returned in this method, otherwise error is handled/thrown.
* **ChatRequest**
  + JSON wrapper for submitting Chat requests to OpenAI.chat() method
  + Contains all variables and wrapper methods to conform messages to expected API input
* **ChatResponse**
  + JSON wrapper for receiving responses from OpenAI.chat() method
* **OpenAI**
  + Interface for interacting with OpenAI API
  + Verifies max tokens were not exceeded before submitting request(otherwise throw exception)
  + Submitting ChatRequest will submit the request to OpenAI by retrieving the API key from the CalloutSecurity class, and executing callout in Callout class. Response is then deserialized to ChatResponse
  + If primary provided API key is compromised/turned off, this class contains logic to update Token Usage custom settings to use backup API key for future callouts.
* **CalloutSecurity**
  + Contains method to retrieve API key for calling OpenAI api
  + Checks for custom customer provided key. If one doesn’t exist, SetupAI provided key is returned.
  + Contains logic to return primary or backup provided API key based on Token Usage settings
* **Controller**
  + Contains methods called by LWC user interface.
  + Methods
    - *submitMessage - public*
      * Takes in user submitted message content and Id of chat thread.
      * If it’s a new thread, the initialize method is called. Otherwise a new Message record is immediately inserted.
    - *Intialize - private*
      * Calls generateChat method to get new Chat record
      * Inserts new message into thread
    - *generateChat - private*
      * Executes one off chat through ChatInterface class to generate the title of chat.
      * Inserts chat record
    - *getMessages - public*
      * Takes in Chat Id
      * Returns all messages in thread in creation order, wrapped in MessageWrapper subclass. Subclass contains CSS classes to display messages correctly based on role
    - *getChatSummary - public*
      * Return name/title of chat
* **CRUD**
  + Contains *createChat* and *createMessage* methods which checks running user security levels on respective objects and associated fields all in one method call
* **FeatureManager**
  + Methods to read and set feature parameter values
* **Function**
  + Contains all functions/methods called by AI assistant to read Salesforce information
  + Public *execute* method is called by MessageTriggerHandler. Function Name determines subsequently called function. Response is saved to new Message record, firing the queueable job
    - soql: Executes standard SOQL query. Calls SOQLModifier class to find/replace semi-common mistakes and limit records returned to prevent token limits being quickly hit.
    - toolingQuery: Executes SOQL query on Tooling API. Utilizes ToolingAPI class
    - getToolingObjects: Returns Names of all Tooling API objects.
    - getFields: Return list of all fields on a given SObject
    - getToolingFields: Return list of all fields on a given Tooling API SObject
    - metadataDrilldown: Used to drill down into Metadata when returned in a Tooling query. Expects query as well as JSON path to value to view in Metadata
    - getEndpoints: Gets all URL\_\_mdt custom metadata records saved so accurate endpoints are returned to end user
    - getOrgLimits: Returns org limits
* **FunctionParam**
  + Contains subclasses for each function that easily deserialize JSON formatted params generated by AI
* **JSONHelper**
  + Contains getJSONValue method that can take in a dot notated path(ex. level1.level2.level3) and returns value in JSON body at that path
* **MessageConvertor**
  + Converts ChatRequest.ChatMessage class instances to Message SObject records and vice versa
* **PostInstall**
  + Runs on package installation to set default Custom Setting Hierarchy values
* **SOQLModifier**
  + Modifies query for simplification and token limit avoidance
    - Automatically adds ‘IsOwnedByProfile’ and ‘ProfileId’ fields to PermissionSet/PermissionSetAssignment queries for added context for the AI
    - Limits all query results to 50 records if limit not otherwise specified by AI
* **SystemPromptHelper**
  + Returns system prompt saved in custom metadata for use in chat
* **Tokenizer**
  + Used to read and update token usage
  + Called every time response is generated by AI
  + Automatically resets every month
  + MaxExceeded method returns true if token allowance is exceeded AND a custom key is not used.
* **ToolingAPI**
  + Interface for all Tooling API methods
    - Get Sobject Names
    - Get SObject Fields(describe Tooling SObject and return field names in describe info)
    - Tooling SOQL Query
    - Get Org Limits
* **Utils**
  + All other helper methods
  + isPrimitive
  + isList
  + getModel
  + sanitizeHTML

## View

### App

SetupAI is a lightning console app that contains 1 tab for the Chat custom object. This is a standard Custom Object tab that displays the list view for existing Chat records and the option to create new ones. Multiple chats can be open at the same time as lightning console tabs.

### Aura Component

One component called *newChatOverrideAura* exists. This component overrides the Chat Sobject New, View and Edit screens. This component is simply a wrapper for the *newChatOverrideLWC.* It passes the record Id to the child component

### Lightning Web Component

* **newChatOverrideLWC**
  + Primary interface through which all main user interaction occurs.
  + All messages are displayed in creation order, with the newest messages on the bottom. User messages on the right, assistant messages on the left.
  + Function messages are hidden from the interface. Only user, assistant and error messages are displayed.
  + On component load
    - If recordId is populated, existing messages on this thread are retrieved via Controller.getMessages(). The title for the chat is retrieved via Controller.getChatSummary().
    - Subscriptions to Message Notice and Async Error are setup
    - Token usage is checked in Tokenizer class. If exceeded, input chat box is disabled
  + On message submit
    - Call Controller.submitMessage() method passing in message contents and thread Id.
    - Display loading wheel
  + On Message Notice received
    - Call Controller.getMessages()
    - Hide Loading wheel
  + On Async Error received
    - Display error message in thread as chat message
    - Hide Loading wheel

## Permissions

SetupAI is the only permission set in this app. It grants access to the Chat object and fields, Message object and fields, and the SetupAI app.

End User Documentation

## Summary

*SetupAI* is a Salesforce app that utilizes Artificial Intelligence to assist you with your org configurations and troubleshooting. You can ask *SetupAI* questions about your Salesforce data and settings, as well as ask for recommendations for improvements to your org.

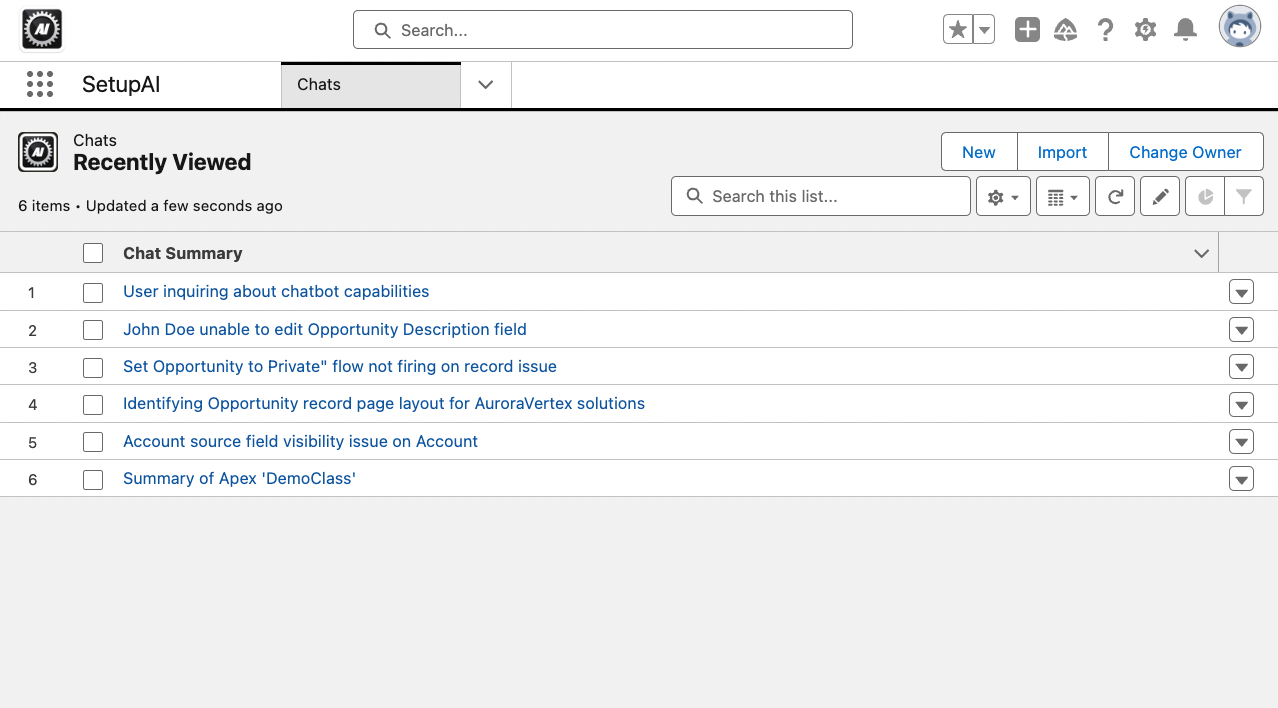
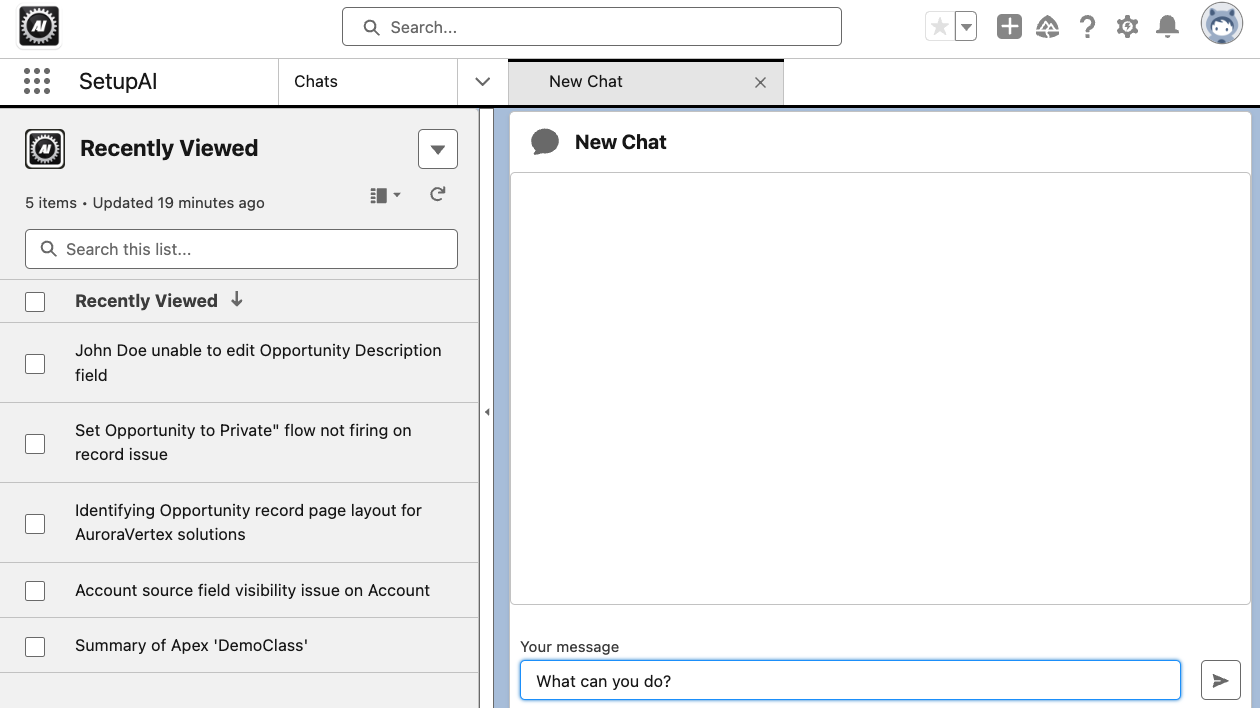
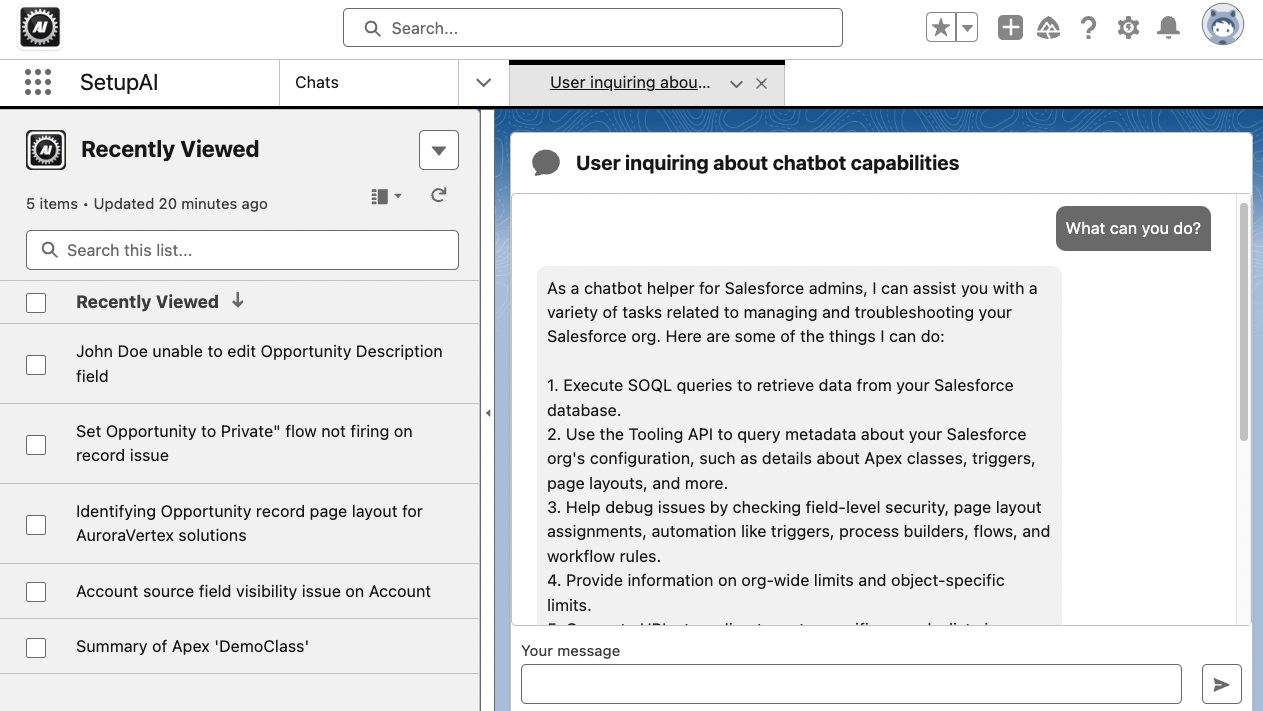
## Setup

After installing the *SetupAI* package, assign the SetupAI Permission Set to yourself, by following these steps:

1. Navigate to **Setup > Users > Permission Sets**.
2. Click the **SetupAI** Permission Set.
3. Click **Manage Assignments**.
4. Click **Add Assignments**.
5. Select your user account, and click **Next**.

## Navigation and How to Use SetupAI

Once you have assigned the SetupAI permission set to your user account, you can now start using SetupAI. To launch SetupAI:

1. Select **SetupAI** from the *App Launcher* to open the *SetupAI* app. You will be taken to the **Chats** tab, which is the only tab in the app.
2. Clicking **New** opens a blank chat window.
3. Enter your prompt in the text box labeled *Your message* and then either click the **Send** icon or press **Enter** to send your prompt to *SetupAI*.
4. After a brief loading screen, you will see the AI generated response. Once you’ve received a response, you can continue to ask follow-up questions in your chat to clarify or gain further information.  
   
5. The Chat List View on the left will display all your existing Chat history. You can open one of these records to resume a chat you’ve left previously.

## Example Prompts

Here is a list of example prompts you could use to start getting familiar with *SetupAI.*

**NOTE:** *Please note that some of these examples reference specific API names of custom fields and objects that may not be in your organization, therefore they will not be found by SetupAI. Feel free to substitute these examples with fields and objects that are in your org.*

* How much file storage space do I have left in my org?
* How many Salesforce licenses do I have available?
* Take me to the *My Open Cases* list view on Cases.
* How close am I to hitting the limit for *Validation Rules* on *Case*?
* Take me to the *General Motors* Account record.
* I can't see the **Options\_\_c** field on the **Category\_\_c** object when I should be able to.
* The **Is\_Primary\_\_c** checkbox on Contact is reverting to unchecked every time I check it.
* Summarize the *CaseTriggerHandler* class.
* Summarize the triggers on the *Opportunity* object.
* Walk me through how to convert the workflow rule *Lead Automated Email* to a Flow.

## Tips

* *SetupAI* runs on GPT-4. If you’re familiar with ChatGPT, and have experience writing prompts for it, this can inform how you write your prompts in *SetupAI*. (Unlike *ChatGPT, SetupAI* has been trained to search your Salesforce org and provide advice tailored to you and your org specifically. But having some knowledge of how to write *ChatGPT* prompts can still help you receive the best possible output.)
* Be as detailed as possible with all of your requests to ensure the best possible output.
* Include API names and exact names of component types, when referencing setup configurations, whenever possible (if you use object labels, the AI will attempt to find the object’s API name, but it may not always be successful.).
* When referencing records, try to include record IDs whenever possible.
* For complex issues with multiple referenced components and records, try walking the AI step-by-step through the issue with multiple prompts and follow up requests.
* Since chat history is saved to records in the Chat tab, you can pick up a chat where you left off and ask more questions without having to re-enter information that a particular chat is already trained on. (This can also save the AI time from having to re-search your org if it’s already found that info in a previous chat)
* AI response times can take anywhere from a few seconds to 1-2 minutes, depending on the complexity of your request and the number of component dependencies. The clearer your requests, the more accurately (and potentially more quickly) your responses will be generated.