TL;DR: We're building an AI-driven conversational assistant inside the Carton Caps app to give users easy access to personalized support. The assistant will be supported by the existing REST API endpoint and will use a retrieval-augmented LLM that integrates user context and existing Carton Caps informational content.

Problem

Carton Caps is an app that empowers consumers to raise money for the schools they care about while buying the everyday products they love. Throughout their user journey, Carton Caps users have various questions about the app–they need help with the referral process, recommendations on products, and more. Current mechanisms for users to find this information are limited to a FAQ page and customer support line. While these have proved sufficient until now, both are relatively high-friction methods of information seeking and recent technological advances present opportunities for improvement.

Proposal

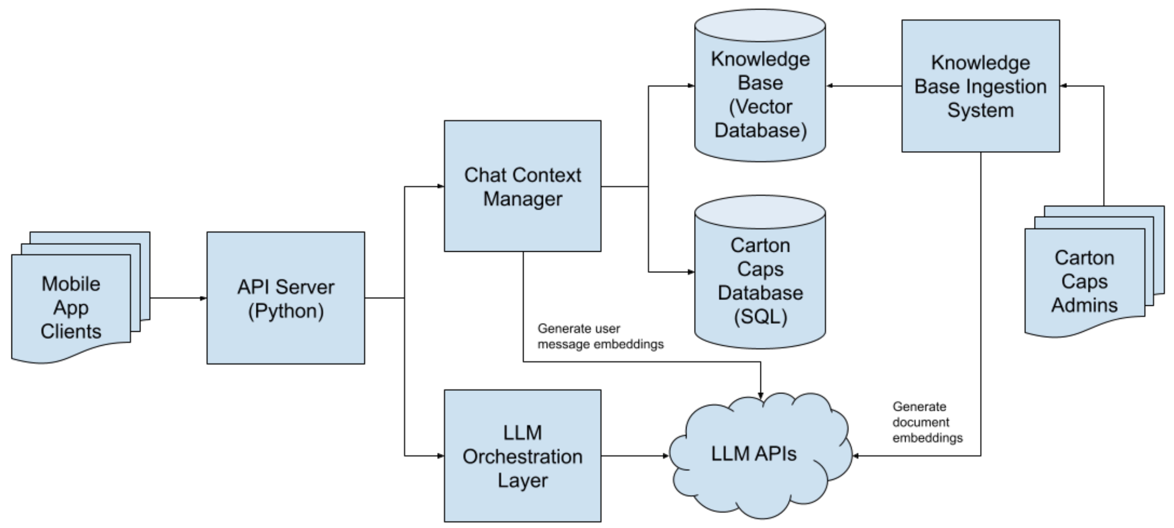
We propose an AI-driven conversational assistant for the Carton Caps app. This assistant will support chats with users through the app to answer their questions about the app. The assistant will maintain context about the user, their current chat, and relevant domain-specific informational content (such as the Carton Caps FAQ) and use this context to provide relevant and helpful information. This initial prototype will be limited in scope and focus on answering text-based queries around Carton Caps app functionality, but future iterations may include multi-modal chats (image upload, voice input) or deeper integration into other user flows such as onboarding or product search.

Requirements

- Users are identified via existing authentication tokens used by the Carton Caps mobile app.
- Users can create a chat with the chatbot.
- Users can send/receive messages in their current chat.
- The chatbot will only answer questions related to the Carton Caps app.
- The chatbot will use a large language model (LLM) with retrieval-augment generation (RAG) from a Carton Caps knowledge base to provide accurate, grounded responses and minimize hallucinations.
- Information provided to the LLM will abide by the Carton Caps user privacy policies.
- The chatbot will leverage relevant and permitted user context such as school affiliation in its responses.
- The Carton Caps knowledge base will support continual updates.

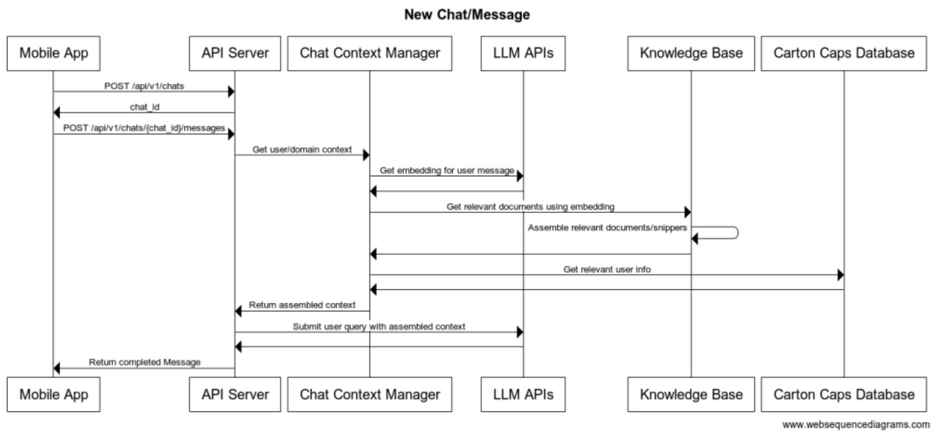
Diagrams

System model

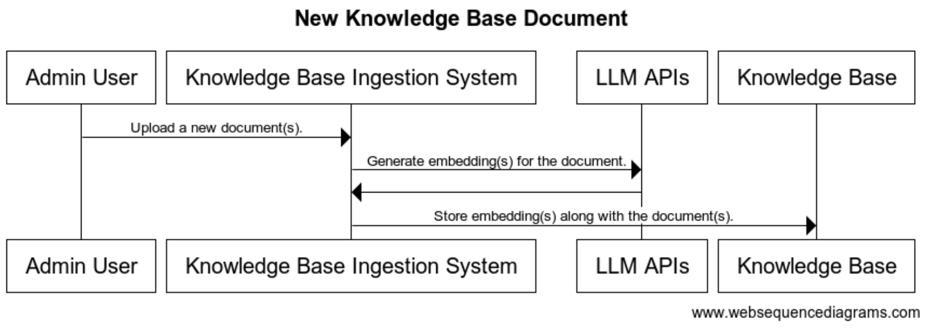

Mobile App Clients
API Server (Python)
Chat Context Manager
LLM Orchestration Layer
Knowledge Base (Vector Database)
Carton Caps Database (SQL)
Knowledge Base Ingestion System
Carton Caps Admins
LLM APIs
Generate user message embeddings
Generate document embeddings

Sequence diagrams

Sending a new message



Ingesting a document to the Knowledge Base


New Knowledge Base Document
Admin User
Knowledge Base Ingestion System
LLM APIs
Knowledge Base
Upload a new document(s).
Generate embedding(s) for the document.
Store embedding(s) along with the document(s).
Admin User
Knowledge Base Ingestion System
LLM APIs
Knowledge Base
www.websequencediagrams.com

API Spec

Models

Chat

Field	Type	Nullable	Description/Notes

id	UUID	No	The ID of the newly created chat.
is_active	Boolean	No	Whether the chat is completed. Always True for a newly created chat.
created_at	Timestamp	No	The creation timestamp of the chat.

Message

Field	**Type**	**Nullable**	**Description/Notes**
id	UUID	No	The ID of the message.
chat_id	UUID	No	The ID of the chat that contains the message.
user_text	String	No	The text of the message.
response_text	String	Yes	The response from the chatbot.
response_citations	List<Citation>	No	RAG sources used by the chatbot in generating a response.
moderation_code	Enum (IRRELEVANT/SAFETY)	Yes	Reason for a moderation flag, can be used by clients to render more detailed messaging. Null if no moderation was triggered.
is_terminal	Boolean	No	True when the chatbot has ended the chat with their response.
created_at	Timestamp	No	The creation timestamp of the message.

Citation

Field	**Type**	**Nullable**	**Description/Notes**
id	UUID	No	The ID of the citation.
name	String	No	A user-facing name for the citation source (e.g.

			Content Caps FAQ, About Us page, etc).
url	String	Yes	The URL where the user can view the content that the citation was derived from. Null if no relevant URL exists.

Endpoints

All calls require an authenticated user, and should include the user's current token in requests using the Authorization header.

Example:

```
Authorization: Bearer {jwt_token}
```

POST /v1/chats

- This endpoint automatically ends any currently active chat and starts a new one.

Request Field	**Type**	**Description/Notes**
N/A	N/A	N/A

Response Field	**Type**	**Description/Notes**
chat	Chat	The newly created chat.

GET /v1/chats/current

Request Field	**Type**	**Description/Notes**
N/A	N/A	N/A

Response Field	**Type**	**Description/Notes**

chat	Chat	The currently active chat, if it exists.

POST /v1/chats/{id}/messages

- This is a synchronous call that blocks on a response from the chatbot.

Request Field	**Type**	**Description/Notes**
text	String	The text of the message.

Response Field	**Type**	**Description/Notes**
message	Message	The newly created message with a response.

GET /v1/chats/{id}/messages

Request Field	**Type**	**Description/Notes**
limit	Integer	The number of messages to get. Defaults to 50. Max of 100.
offset	Integer	Pagination offset, i.e. 50 if the first request fetched 50 messages.
sort_field	ENUM (created_at)	What field to sort the messages on. Defaults to created_at.
sort_direction	ENUM (ascending, descending)	The direction to sort the messages. Defaults to descending.

Response Field	**Type**	**Description/Notes**
messages	List<Message>	The messages in the chat.

Integration

The expected mobile app integration pattern is as follows:

User Starts a Chat

- Create a new chat (POST /chats) and store the resulting chat_id.

User Sends a Message

- Using the stored chat_id, create a new message (POST /chats/{chat_id}/messages).
- If the response includes a moderation code, indicate this clearly to the user.
- If the response has is_terminal set to true:
    - Show a chat termination message after the response.
    - Block further user input for this chat.
    - Allow the user to start a new chat.

Page Reload/Reconnect

- Check for an existing chat (GET /chats/current). If one exists, fetch the messages (GET /chats/{chat_id}/messages and render the chat window.

LLM Considerations

Model

This will likely be determined by business constraints of some kind, but any mainstream multi-modal LLM should be sufficient. Multi-modal support is important to support future work that could include image uploads, app state screenshots, etc.

If enterprise-hosted models are acceptable, we will use gpt-4o as a starting point. Different constraints would point us in different directions but below are some basic recommendations:

- Too slow or expensive → o4-mini.
- Reasoning power seems insufficient → Claude Opus 4.
- Can't use enterprise-hosted models → Look into self-hosting a Mistral or Llama model.

Alignment

System prompting will be used to make sure the chatbot responses stay within the realm of its purpose, i.e. helping users navigate various aspects of the Carton Caps product.

Prompt Engineering will be needed to find the optimal system prompt but a good starting point could be:

```
Carton Caps is an app that empowers consumers to raise money for the
schools they care about while buying the everyday products they love.
```

`Features of the app include assistance finding products that can support the user’s local school and a referral program that allows you to invite your friends to join the Carton Caps app.`

`You are a helpful chatbot that users will interact with for help in navigating the Carton Caps app. Your responses should be narrowly constrained to be relevant to this purpose, and any irrelevant questions can be answered with “I’m here to help with Carton Caps—try asking about referrals or products.”`

Grounding/Hallucinations

It’s critical that we keep our chatbot responses grounded in factual data about the Carton Caps app and avoid any LLM hallucinations. To do this we will employ two primary strategies:

- Low temperature (0.0-0.5). Experimentation should be done to find the right value to balance between consistency and expressiveness, but a low temperature will make responses more focused and consistent, limiting hallucinations.
- The Knowledge Base will be used to fetch relevant contextual information related to the user's question. This context should be included in the prompt sent to the LLM with instructions to constrain the LLM response to that context. Prompt Engineering will be needed but a starting point is: `Provided below is relevant informational content found in the Carton Caps database. Limit the scope of your response to this context. If there’s no relevant information or the question can’t be answered confidently, respond with: I’m not totally sure, please try another question.`

Privacy

A detailed privacy policy will require a deeper dive into Carton Caps customer privacy policies. The general mechanism for handling privacy considerations in chatbot conversations will be an explicit layer in the implementation–the Chat Context Manager (found in the system model diagram). This manager will maintain strict logic for fetching customer data in a way that aligns with our privacy policies and anonymizing it to remove any identifiable information before passing the context back to the caller (which will then pass it on to our LLM integration).

Trade offs

The primary trade-offs made here are around modality and synchronous responses. The motivation for these trade-offs is primary time constraints surrounding the prototype nature of the design.

A future evolution of this design would likely do the following:

- Model message content as a list of content blocks. This would help support multi-modal use cases as well as multi-block responses (as seen in the mock UX).
- Model user messages and chatbot responses separately. The current 1:1 nature of questions/responses is suitable for an MVP but limits functionality–it makes it difficult to accomplish features like introductory messages in a chat coming from the assistant, asynchronous responses, or streaming responses as they generate.
- Move to an asynchronous response approach. This increases complexity throughout the frontend/backend but protects against slow LLM response times and allows our API server to handle higher load during peak traffic.