NOVA IMS

PROJECT CAPSTONE

Second Delivery

Group 1:
Dinis Fernandes 20221848
Dinis Gaspar 20221869
Inês Santos 20221916
Luis Davila 20221949
Sara Ferrer 20221947

Index

Data Sources

The DB_and_Synthetic_TabularData_embedding.ipynb notebook contains all the code used to create our database with its constraints. It also contains the code for synthetically generating the information to populate it. Furthermore, this notebook also has the steps to insert activity information into the vector database (pinecone index).

The PDF files with information about the company were created by us.

Database Design

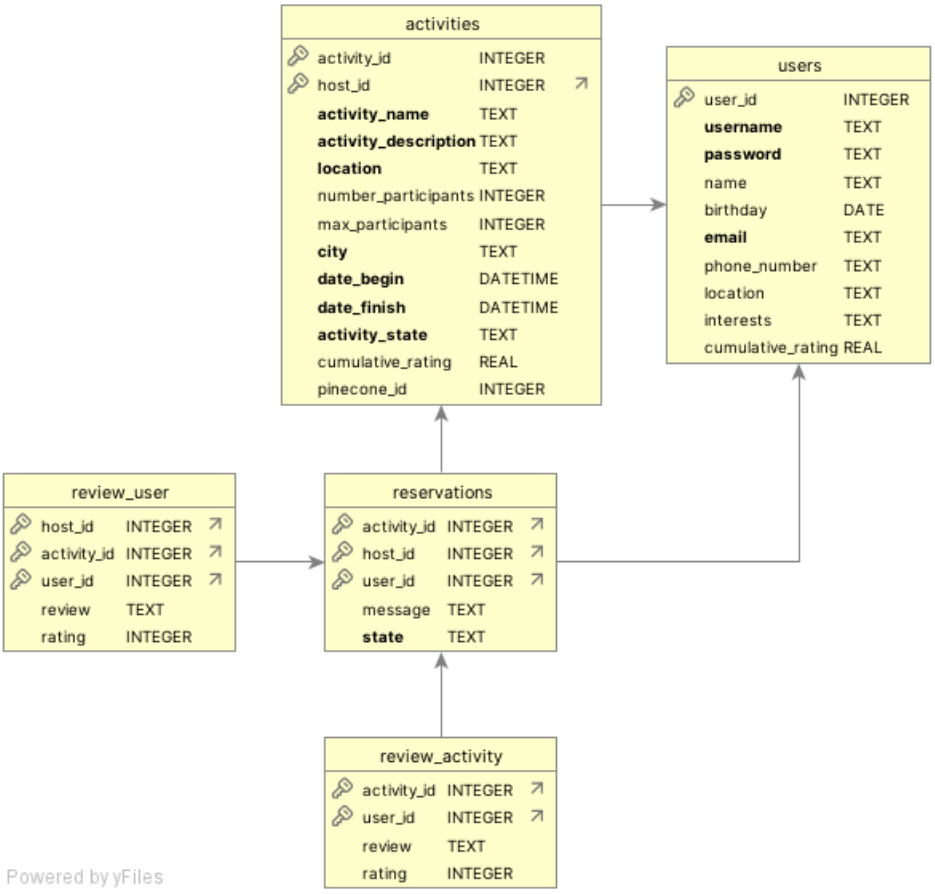


Figure 1 - Database Schema.

The database schema for this application consists of five tables:

- **users:** Maintaining user data is crucial for activity searching to allow for truly personalized recommendations, also to maintain contact points, in case there is a need to contact the user. This table only has one primary key, **user_id**. The **username** field needs to be

unique, not null and has to have less than ten characters. The **password** field has the same constraints as the username, but it doesn't need to be unique. The **email** field needs to be unique and not null. The **cumulative_rating** field has a default value of zero.

- **activities:** Ensures that information regarding all activities can be easily shown to users when making recommendations. It also gives the activities a well-structured format. This table has two primary keys, **activity_id** and **host_id**, which also is a foreign key that refers to **user_id** in the **users** table. The **activity_name**, **activity_description**, **location**, **city**, **date_begin** and **date_finish** fields cannot be null values. The **activity_state** field needs to be one of the following options: **open**, **full** or **finished**, and therefore it can't be a null value. The **cumulative_rating** field has a default value of zero.

- **reservations:** The center point of the platform. It allows users to send requests to participate in activities and gives hosts the opportunity to filter the participants of their activities, as well as keeping track of activity attendance. It works as the ultimate link between users and activities. Additionally, this table is necessary to allow users to review activities and hosts to review users, because it will ensure the consistency of reviews, and guarantee that reviews are only accepted if they are made by or about users who attended the specific activity. This table has three primary keys and all three are foreign keys: **activity_id** refers to **activity_id** in the **activities** table, **host_id** refers to **host_id** in the **activities** table and **user_id** refers to **user_id** in the **users** table. The **state** field has to be one of the following options: **confirmed** or **pending**, and therefore it can't be a null value.

- **review_user:** Allows hosts to write reviews and leave ratings about the participants of their activities, helping future hosts make more informed decisions. This table has three primary keys and all three are foreign keys: **activity_id** refers to **activity_id** in the **reservations** table, **host_id** refers to **host_id** in the **reservations** table and **user_id** refers to **user_id** in the **reservations** table. The values inserted in the **rating** field need to be between one and five.

- **review_activity:** Allows users to leave anonymous feedback for hosts about their activities, this can lead to improved activities and ultimately higher user satisfaction. This table has two primary keys, which also are foreign keys: **activity_id** refers to **activity_id** in the **reservations** table and **user_id** refers to **user_id** in the **reservations** table. The values inserted in the **rating** field need to be between one and five.

Activity Vector Database:

The **Activity Vector Database** is the heart of the activity recommendation platform, as it will leverage embedding technology to try and find activities which are truly meaningful and relevant for users based on locations, interests and the activity description. The database will store the activity description (name, location, number of people, etc) as an embedding to be queried afterwards.

Company Information Vector Database:
The **Company Information Vector Database** also leverages embedding technology, in this case to quickly and accurately reply to customer questions about company information and or chatbot/platform functionality.

Chatbot Architecture

Note: The user input will first pass into a router that will redirect it, given the value and intention of the input, to the appropriate chain. Also, a login system will be implemented to validate and extract user data for the different functionalities.

Feature: Company and Chatbot Information

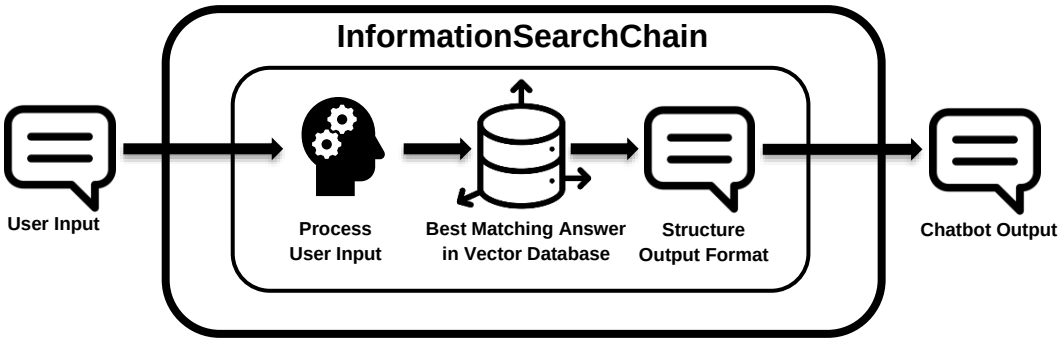
User story:

"As a user,
I want to be able to know about the company and chatbot,
So that I can understand and use the value that it provides"

Architecture Diagram:

Goal: Allow users to obtain information about the company and chatbot functionalities.

Implementation: We will implement a chain that will take the question from the customer input. Afterwards, it will find in the vector database (pinecone index, using RAG) the best answer regarding the company or chatbot functionality. Finally, the chain will return a structured output with the relevant information in clear format.



Feature: Update User Data

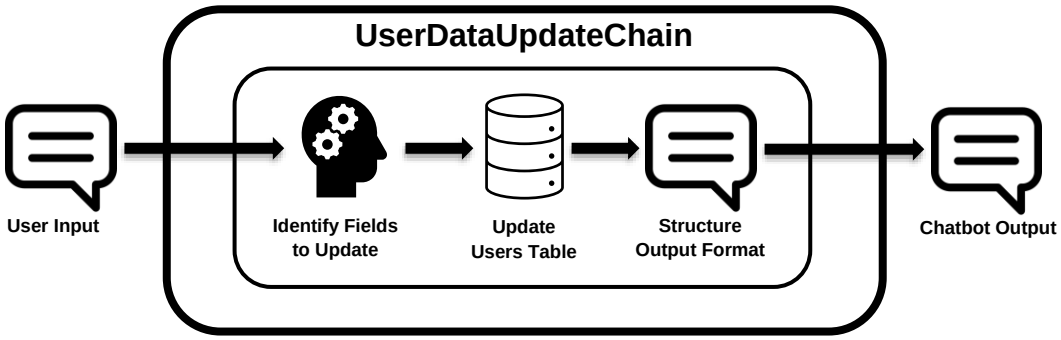
User story:

"As a user,
I want to be able to update my personal data,
So that I can find meaningful activities and people can contact me correctly"

Architecture Diagram:

Goal: Allow users to make updates to their personal information.

Implementation: We will build a chain that will infer the fields the user wants to update and the new values the user wants to input. It will then access and update the users table in the database with the new information. At the end of the chain a chatbot confirmation message will be returned if the update is successful and an error message will be returned if not.



Feature: Create an Activity

User story:

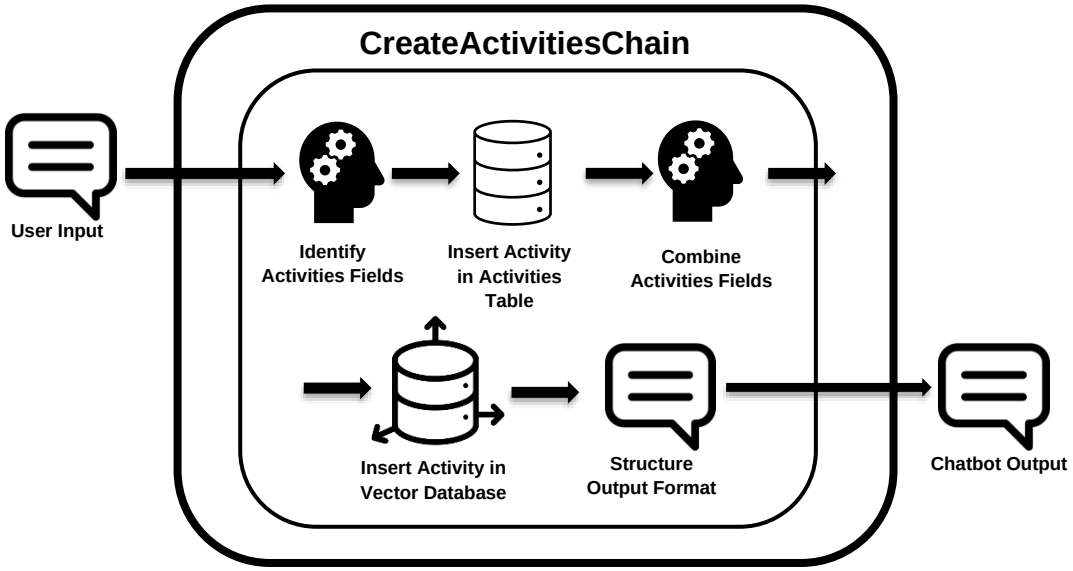
"As a host,
I want to be able to create activities,
So that I can do something special with others"

Architecture Diagram:

Goal: Allow hosts to create personalized activities to do.

Implementation: We will implement a chain that will take the customer input with the activity description and the remaining appropriate fields, it will insert the fields in the activities table (to store the activity), then combine these elements to store into the vector database (pinecone index,

to use in RAG). Finally, the chain returns a chatbot message confirming that the process has succeeded.



Feature: Searching for an Activity

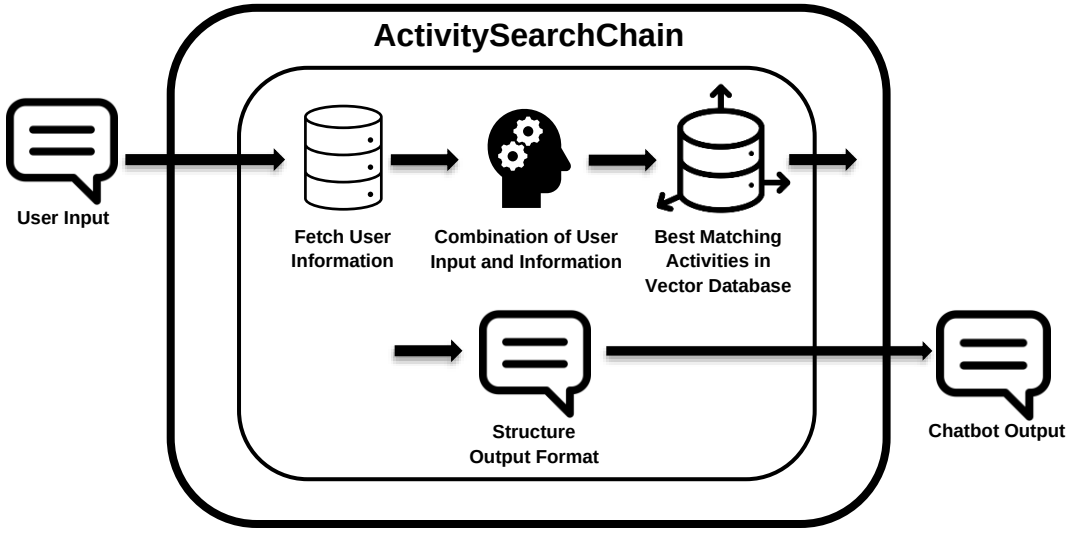
User story:

"As a user,
I want to be able to search for activities that are in line with my interests,
So that I can have fun and meaningful experiences"

Architecture Diagram:

Goal: Allow users to find activities that are best suited to them.

Implementation: We will implement a chain that will take the customer input, fetch user information such as location and interests from the users table of the database, it will then combine these elements. Afterwards, it will find in the vector database (pinecone index, using RAG) the best matching activities. Finally, the chain will return the relevant activity information for those activities in a chatbot message in a clear format.



Feature: Making a Reservation for an Activity

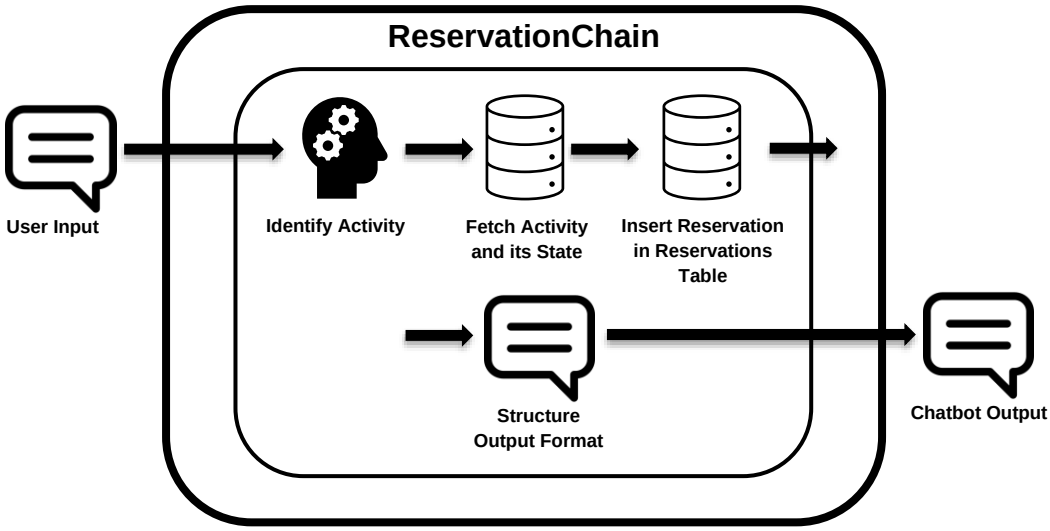
User story:

"As a user,
I want to be able to make reservations for activities I want to take part in,
So that I can guarantee my spot"

Architecture Diagram:

Goal: Allow users to make reservations for activities.

Implementation: We will create a chain that will take from the input the activity that the user is trying to reserve, it will then fetch the activity status from the activities table. If the activity is still open, it will insert into the reservations table a reservation with status "pending" for that user and activity and a message for the host. If the process is successful, the chain will return a chatbot confirmation message, and if it is unsuccessful, an error message is returned.



Feature: Check Activity Reservations

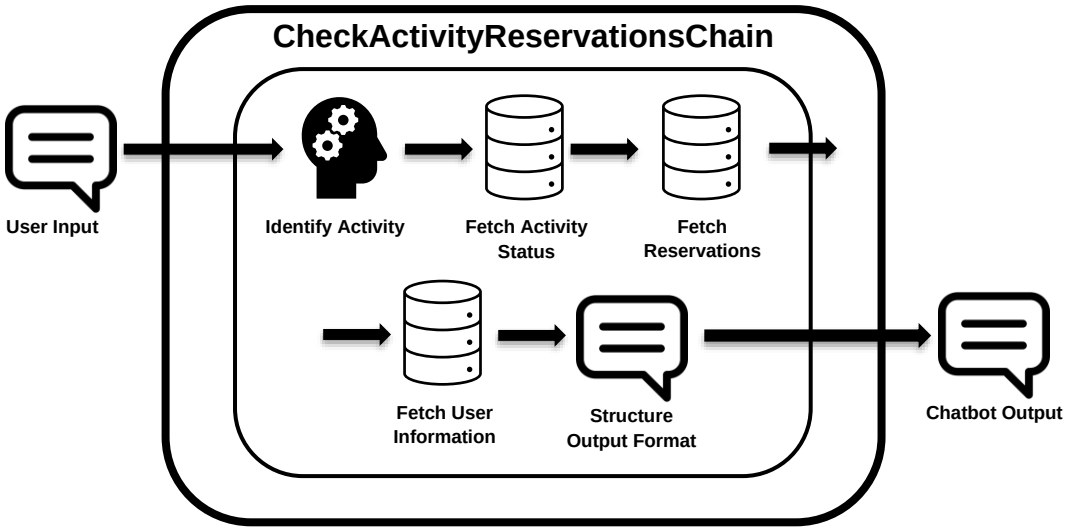
User story:

“As a host,
I want to be able to see all the reservations to my activities,
So that I can then manage them if needed”

Architecture Diagram:

Goal: Allow hosts to see a list of users who have a reservation for an activity and some relevant information about the users.

Implementation: We will implement a chain that will infer the activity the user wants to check and fetch its status, it will then fetch all reservations corresponding to that activity, the ratings, messages and contact information of the respective users. The chain will return a chatbot output containing the activity, its status, reservations (username), the status of the reservations, the user rating, messages and contact information.



Feature: Accept an Activity Reservation

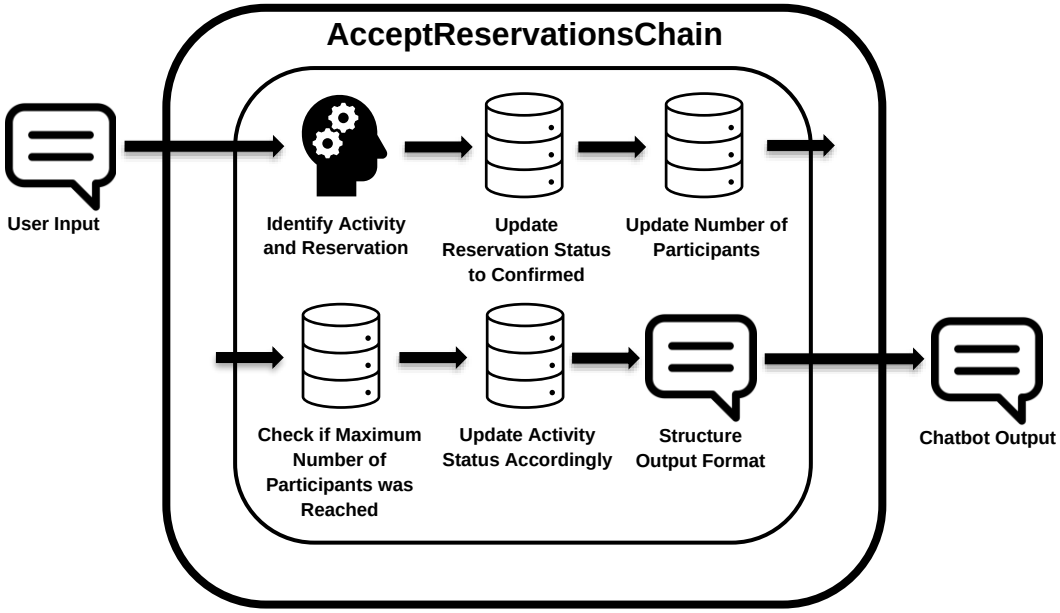
User story:

“As a host,
I want to be able to accept the reservations of users who want to participate in my activity,
So that I can manage who takes part in my activities.”

Architecture Diagram:

Goal: Allow hosts to accept reservations of users to their activities.

Implementation: We will implement a chain that will infer which activity and reservation the host wants to accept, from the input. It will then update the status of that reservation to “confirmed”. After this, the number of participants in the activity will be updated and it will be checked if the maximum number of participants is reached with the activity status being updated accordingly. Afterwards, the chain returns a chatbot message confirming that the reservation has been accepted.



Feature: Reject an Activity Reservation

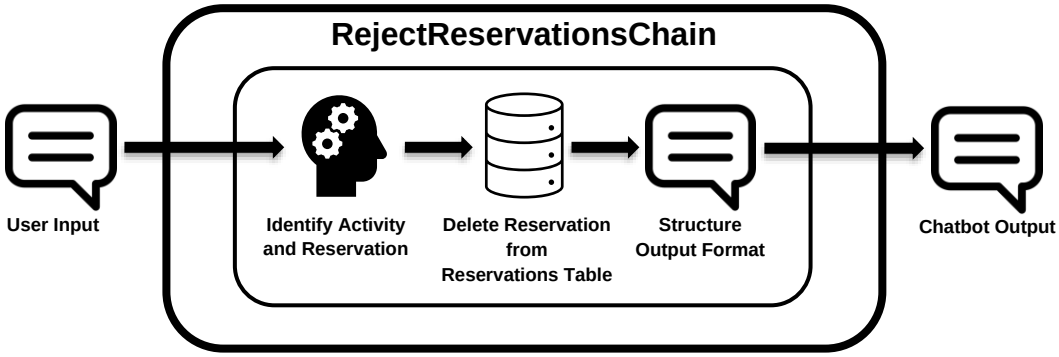
User story:

“As a host,
I want to be able to reject the reservations of users who want to participate in my activity,
So that I can manage who takes part in my activities.”

Architecture Diagram:

Goal: Allow hosts to reject reservations of users to their activities.

Implementation: We will implement a chain that will infer which activity and reservation the host wants to reject, from the input. It will then delete this reservation from the database. Afterwards, the chain returns a chatbot message confirming that the process has succeeded.



Feature: Check Number of Participants

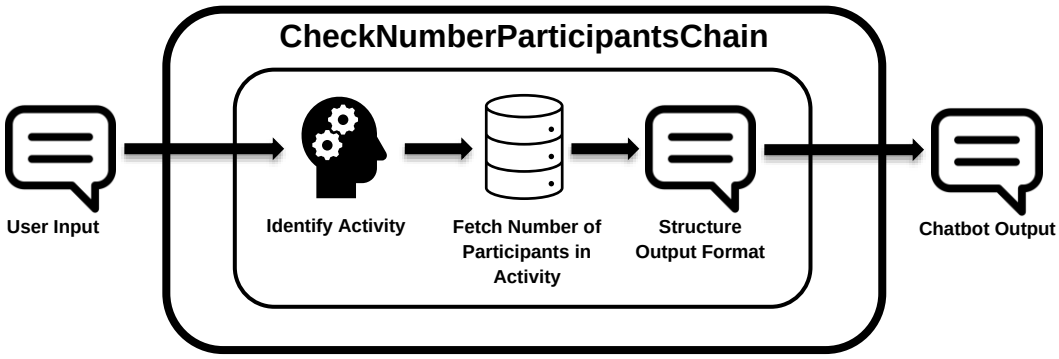
User story:

"As a host,
I want to be able to know the number of people confirmed in my activity,
So that I can know how many people I can still invite"

Architecture Diagram:

Goal: Allow hosts to see the number of people confirmed in an activity.

Implementation: We will implement a chain that will infer the activity the user wants to check. it will then fetch the number of users that are confirmed and the max number of participants. The chain will return a structure output.



Feature: Delete Activities

User story:

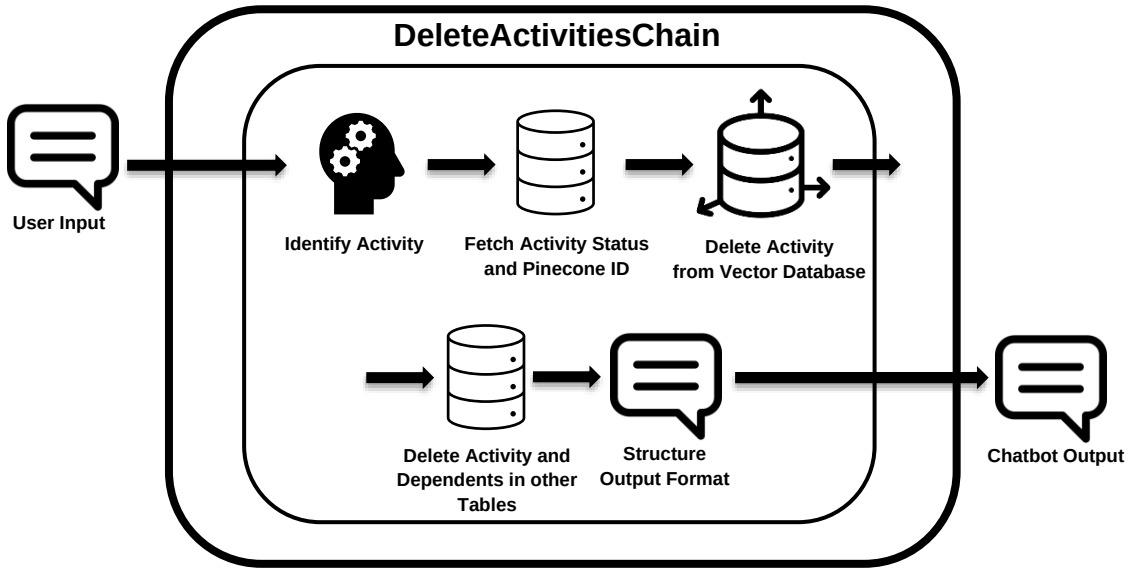
"As a host,
I want to be able to delete my activities,

So that I can remove activities which have been cancelled”

Architecture Diagram:

Goal: Allow hosts to cancel their activities before they’ve occurred.

Implementation: We will implement a chain that will infer which activity the user is attempting to delete, it will fetch its status and pinecone_id from the database. If the activity hasn’t finished, it will be deleted from the database (along with its dependents in other tables), additionally it will be deleted from the pinecone index. Afterwards, a confirmation chatbot message is returned if the deletion is successful and an error message is returned if the deletion can’t be completed (because the activity has finished, for example).



Feature: Review Activities

User story:

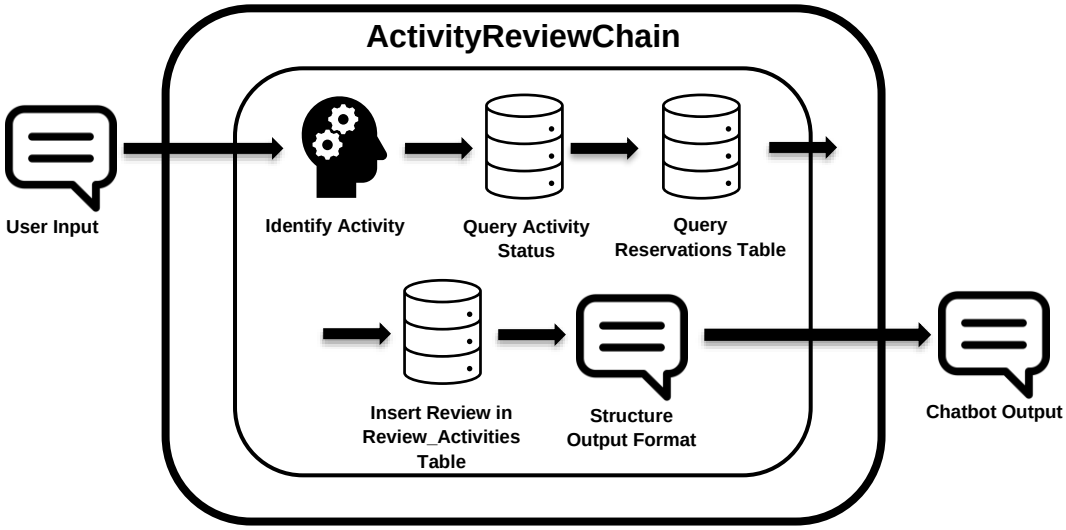
“As a user,
I want to be able to review activities I have participated in,
so that I can give hosts feedback and express my opinion”

Architecture Diagram:

Goal: Allow users to review activities they have taken part in.

Implementation: We will implement a chain that will infer the activity the user wants to review and fetch its status. If the activity has finished, we will query the reservations database to ensure

the user attended the activity they want to review. If so, the review and rating is inserted into the review_activities table in the database. If the process is successful, the chain will return a chatbot confirmation message, and if it is unsuccessful, an error message is returned.



Feature: Check Activity Participants

User story:

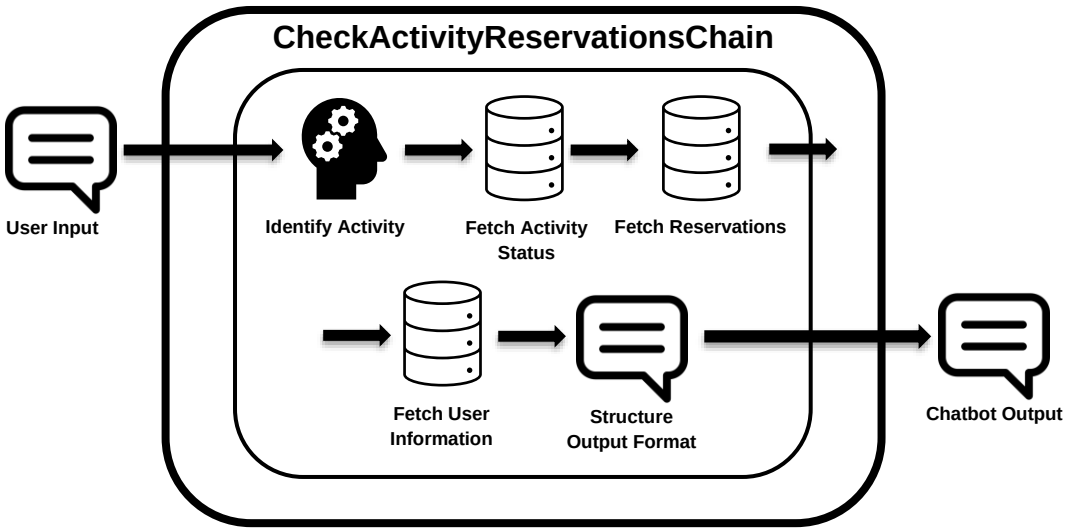
"As a host,
I want to be able to see who took part in my activity,
So that I can review them, for example"

Architecture Diagram:

Goal: Allow hosts to see a list of users who took part for a given activity.

Note: As checking activity participants is essentially just checking the reservations of an activity that has finished, we will use the same chain that we defined for checking activity reservations as that chain already returns the status of the activity and the required information for the host.

Implementation: We will implement a chain that will infer the activity the user wants to check and fetch its status, it will then fetch all reservations corresponding to that activity and the ratings and usernames of the respective users. The chain will return a chatbot output containing the activity, its status, reservations (username) and the user rating.



Feature: Check Activity Reviews

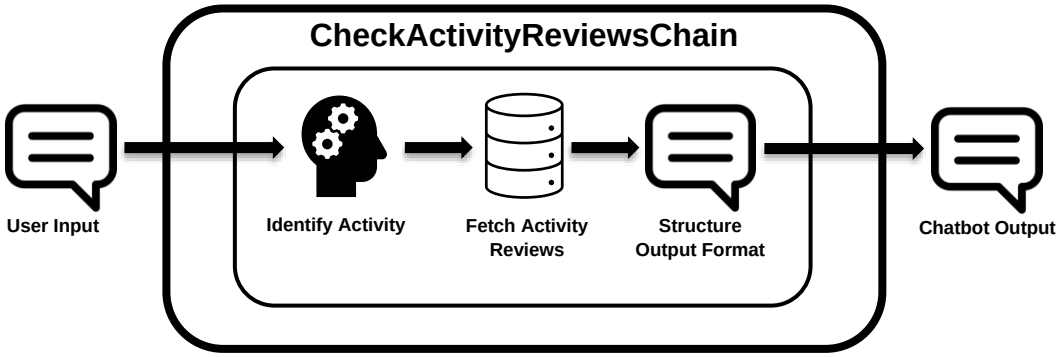
User story:

"As a host,
I want to be able to see all the reviews of my activities,
So that I can improve for next time"

Architecture Diagram:

Goal: Allow hosts to see a list of the reviews and ratings left for an activity that has concluded.

Implementation: We will implement a chain that will infer the activity the user wants to check. it will then fetch all reviews corresponding to that activity from the review_activity table. The chain will return a chatbot output containing the reviews (text review and rating, no information regarding the user who made the review).



Feature: Review Users

User story:

“As a host,
I want to be able to review users that participated in my activity,
So that I can evaluate their behaviour, and help other hosts make more informed decisions.”

Architecture Diagram:

Goal: Allow hosts to review users who took part in their activity.

Implementation: We will implement a chain that will infer the activity and username that the host wants to review, fetch the activity status from the activities table and the user_id from the users table. If the activity has finished, we will query the reservations database to ensure the user attended the activity. If so, the review is inserted into the review_user table in the database. If the process is successful, the chain will return a chatbot confirmation message, and if it is unsuccessful, an error message is returned.