Title:	Detailed development document for Music Concert Recommender System (MCRS)
Contents	1. Use case diagram. 2. Interaction diagram. 3. Class diagram. 4. Activity diagram. 5. ER diagram. 6. Inter-Agent messages
Team1 members:	- Raha Moradi Shahmiri. - Jagrit Acharya. - RAHEEL AFSHARMAZAYEJANI. - Khawla Naji Shnaikat.

In this document, we provide a detailed development overview covering the following key points:

1. VenueAgent Addition: To facilitate our system's core functionality of recommending upcoming music concerts based on user preferences, we have introduced a new agent known as the "VenueAgent." This agent acts as a service provider, possessing knowledge about upcoming concerts. The VenueAgent is responsible for retrieving information about these concerts and inserting this data into our database. The data insertion process is carried out by the DataManager agent, which is tasked with the responsibility of acquiring this concert information and inserting it into the concerts table.
2. Admin Agent for User Management: Our system relies on the Admin Agent to manage user-related operations. This agent is responsible for user registration in the user’s table. Additionally, it performs data validation, addressing any errors in the provided data fields. Furthermore, the Admin Agent verifies whether the registered user is new to the system or an existing user.
3. Concert Seeker Agent (User GUI Representation): We have introduced the Concert Seeker Agent, which serves as the graphical user interface (GUI) representation for users. This agent empowers users to request concert recommendations by providing essential parameters, including email, location preferences, ticket price, and genre.
4. Recommender Agent: The Recommender Agent plays a pivotal role in our system's recommendation process. It receives a message containing the user's email, location preference, ticket price, and genre from the Concert Seeker Agent. Subsequently, it communicates with the Admin Agent to validate whether the provided email exists within the system. If the email exists, the RecommenderAgent:
   - Inserts the user's preferences into the preferences table.
   - Proceeds with the task of finding a matching concert recommendation based on the user's input.
   - If the email does not exist, the user is prompted to complete the registration process.
5. Invitation Request Handling: Invitation requests are initiated via the ConcertSeeker GUI. Following the receipt of a concert recommendation by the Concert Seeker, a "Find Friends" button becomes available. When the user clicks this button, the preferences of the Concert Seeker are transmitted to the Invitation Agent. The Invitation Agent, in

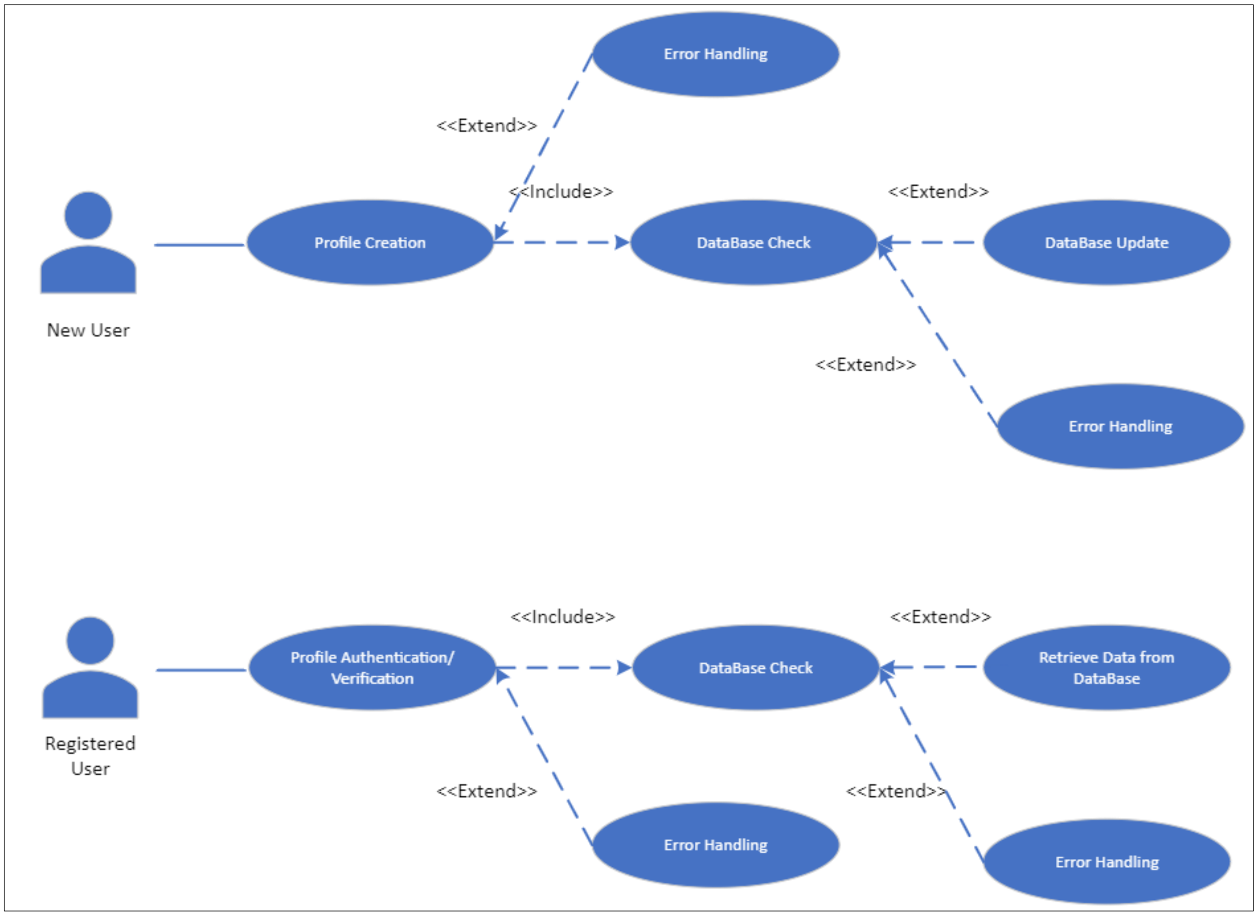
response, identifies potential friends (names and emails) who share similar preferences. This information is subsequently utilized to update the friends table accordingly.

In the next pages, we will start to provide the detailed documentation for MCRS contents.

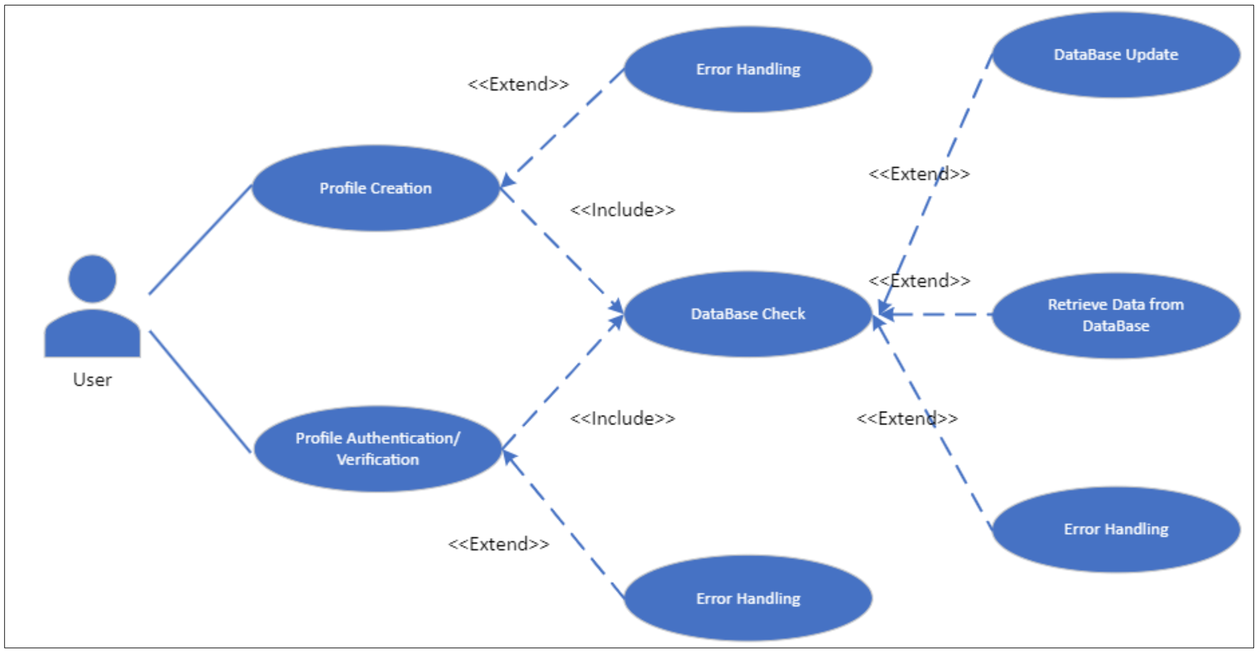
1 Use Case Diagram of Agents:

1.1 Use Case Diagram for Admin Agent

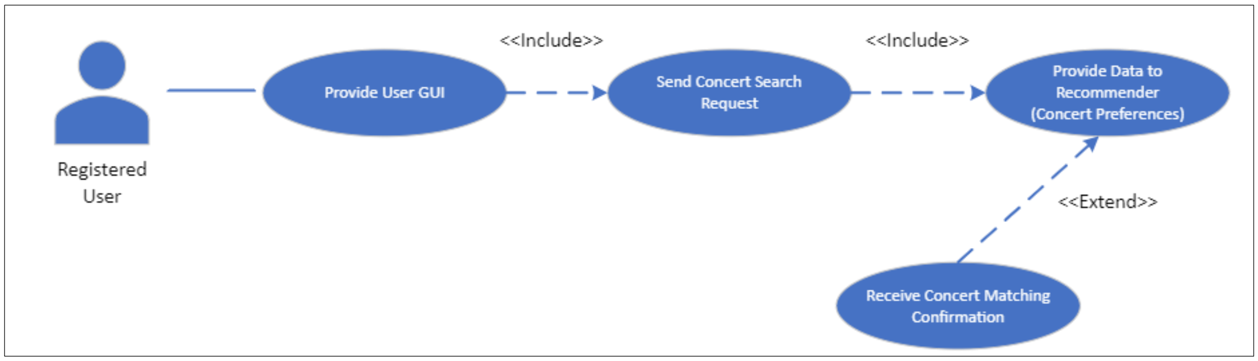
(User = "Registered User" or “New User")



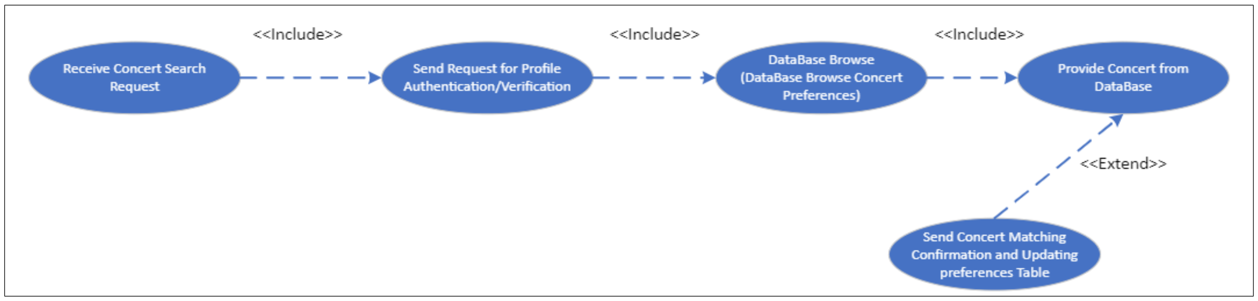
Or:



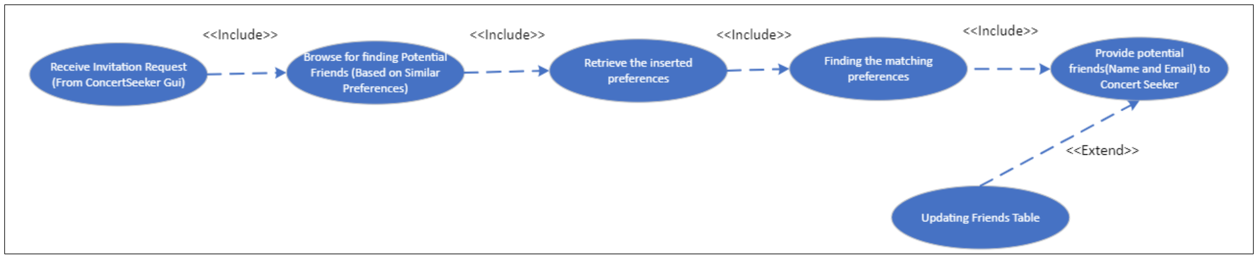
1.2 Use Case Diagram for Concert Seeker Agent


Registered User
Provide User GUI
<<Include>>
Send Concert Search Request
<<Include>>
Provide Data to Recommender (Concert Preferences)
<<Extend>>
Receive Concert Matching Confirmation

1.3 Use Case Diagram for Recommender Agent:

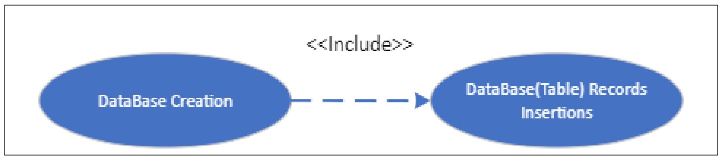

Receive Concert Search Request
<<Include>>
Send Request for Profile Authentication/Verification
<<Include>>
DataBase Browse (DataBase Browse Concert Preferences)
<<Include>>
Provide Concert from DataBase
<<Extend>>
Send Concert Matching Confirmation and Updating preferences Table

1.4 Use Case Diagram for Invitation Agent

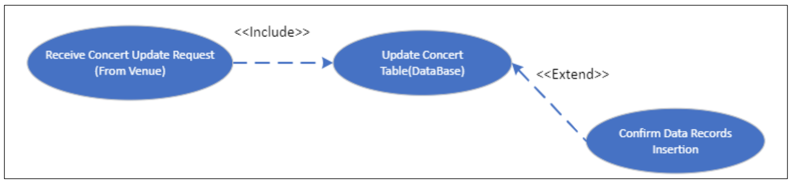

Receive Invitation Request (From ConcertSeeker Gui)
<<Include>>
Browse for finding Potential Friends (Based on Similar Preferences)
<<Include>>
Retrieve the inserted preferences
<<Include>>
Finding the matching preferences
<<Include>>
Provide potential friends(Name and Email) to Concert Seeker
<<Extend>>
Updating Friends Table

1.5 Use Case Diagram for Venue Agent

Venue agent acts as an external service that provides information related to concert venues, ticket prices, and music genre type.



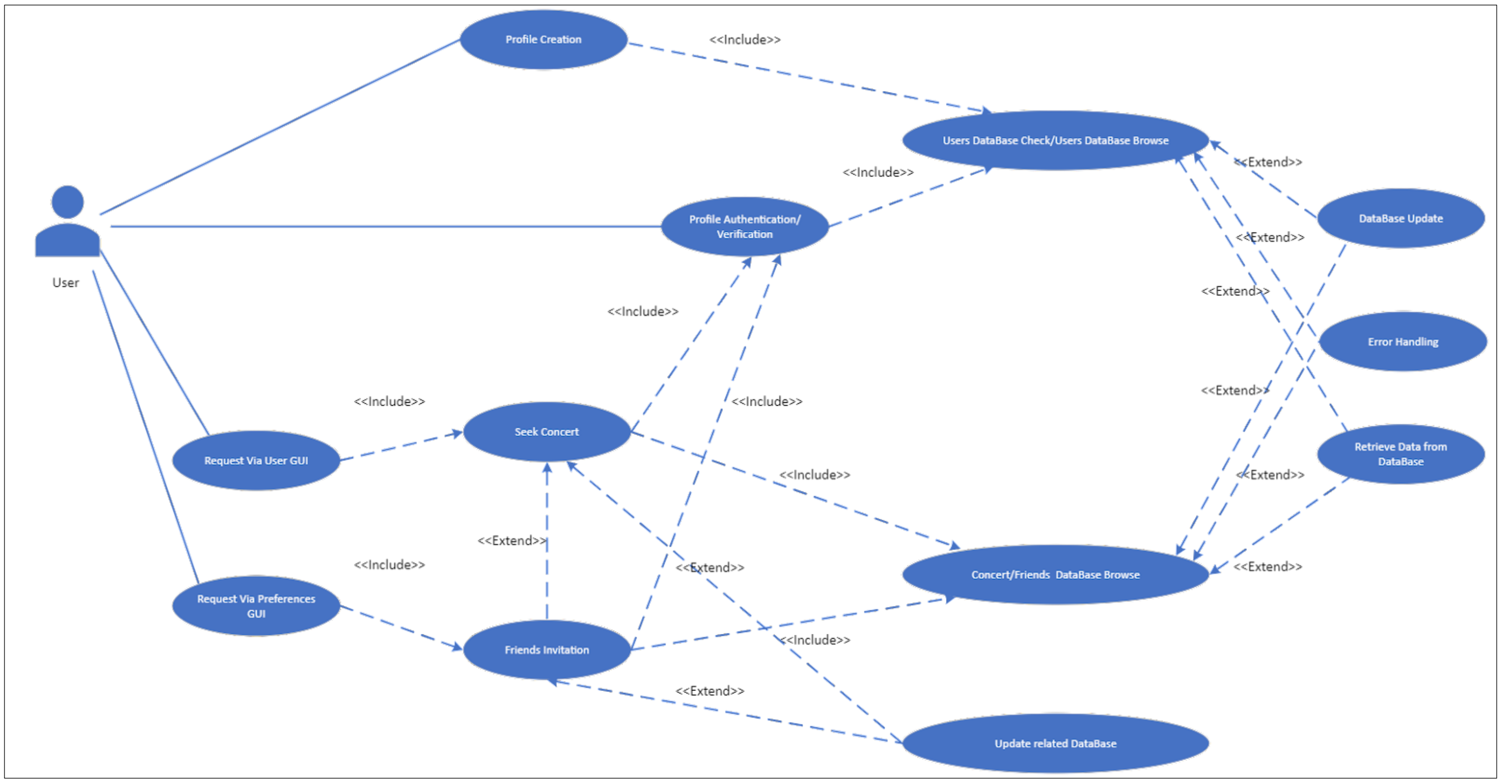
1.6 Use Case Diagram for DataManager Agent


Receive Concert Update Request (From Venue)
<<Include>>
Update Concert Table(DataBase)
<<Extend>>
Confirm Data Records Insertion

1.7 Use Case Diagram for Music Concert Recommending System (whole system)

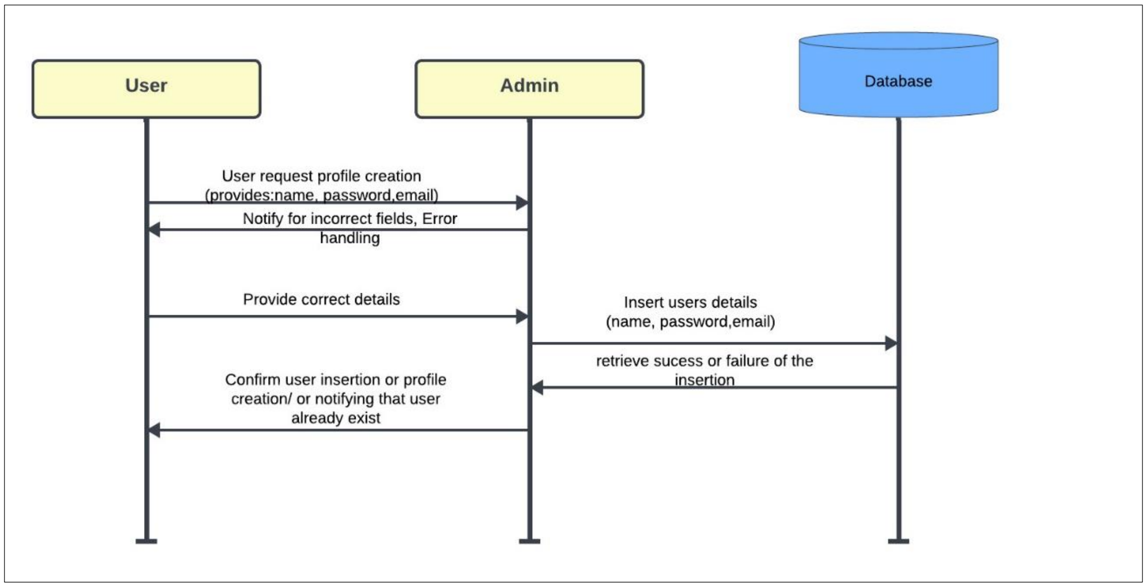
For the sake of clarity, we added detailed information in previous parts to each agent's Use Case diagram. And for the whole diagram, we tried to mention the main parts of the system.

(User = "Registered User" or “New User")

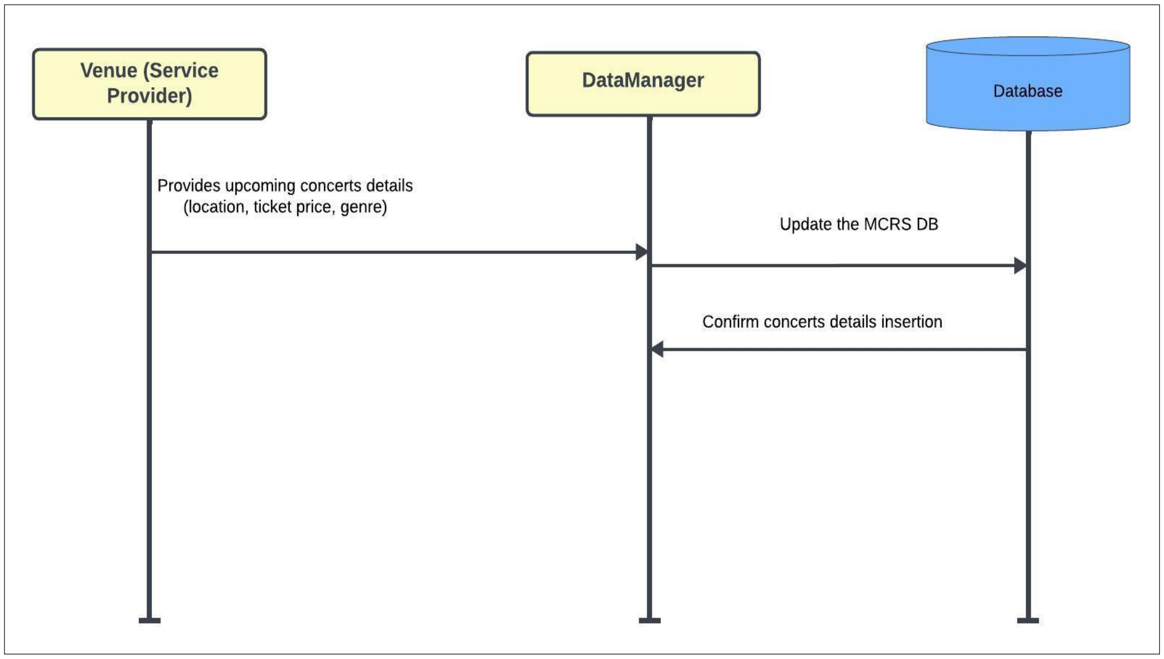


2. Interaction diagram: type of diagram that focuses on illustrating how objects or components within a system interact and communicate with each other to achieve a specific behavior or functionality.

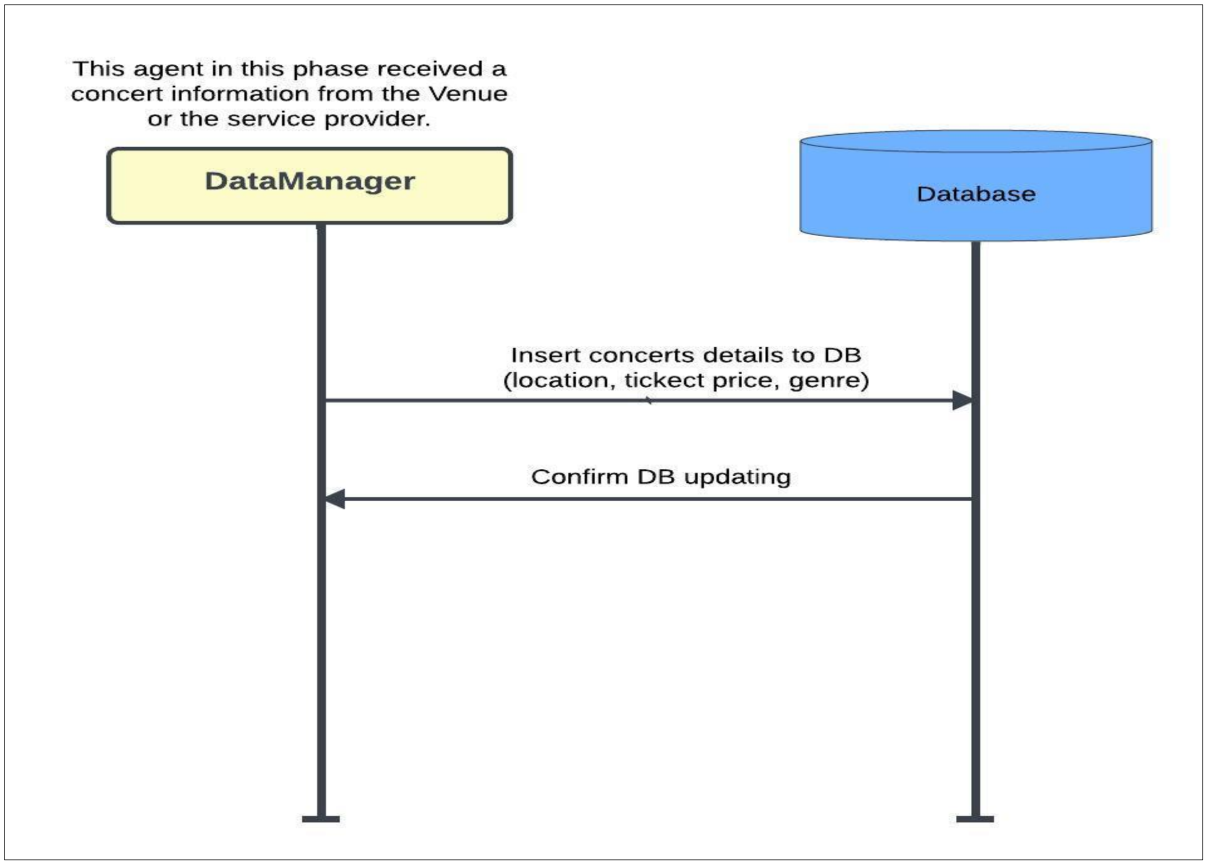
2.1 Interaction diagram for Admin Agent:


User
Admin
Database
User request profile creation (provides:name, password,email)
Notify for incorrect fields, Error handling
Provide correct details
Insert users details (name, password,email)
retrieve sucess or failure of the insertion
Confirm user insertion or profile creation/ or notifying that user already exist

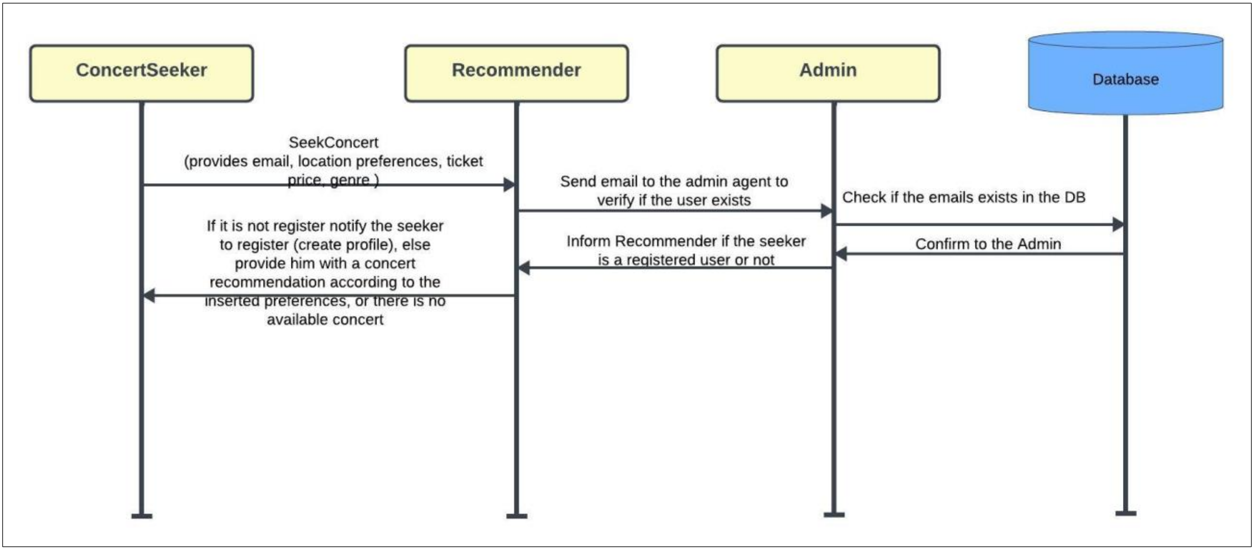
2.2 Interaction diagram for Venue Agent:


Venue (Service Provider)
DataManager
Database
Provides upcoming concerts details (location, ticket price, genre)
Update the MCRS DB
Confirm concerts details insertion

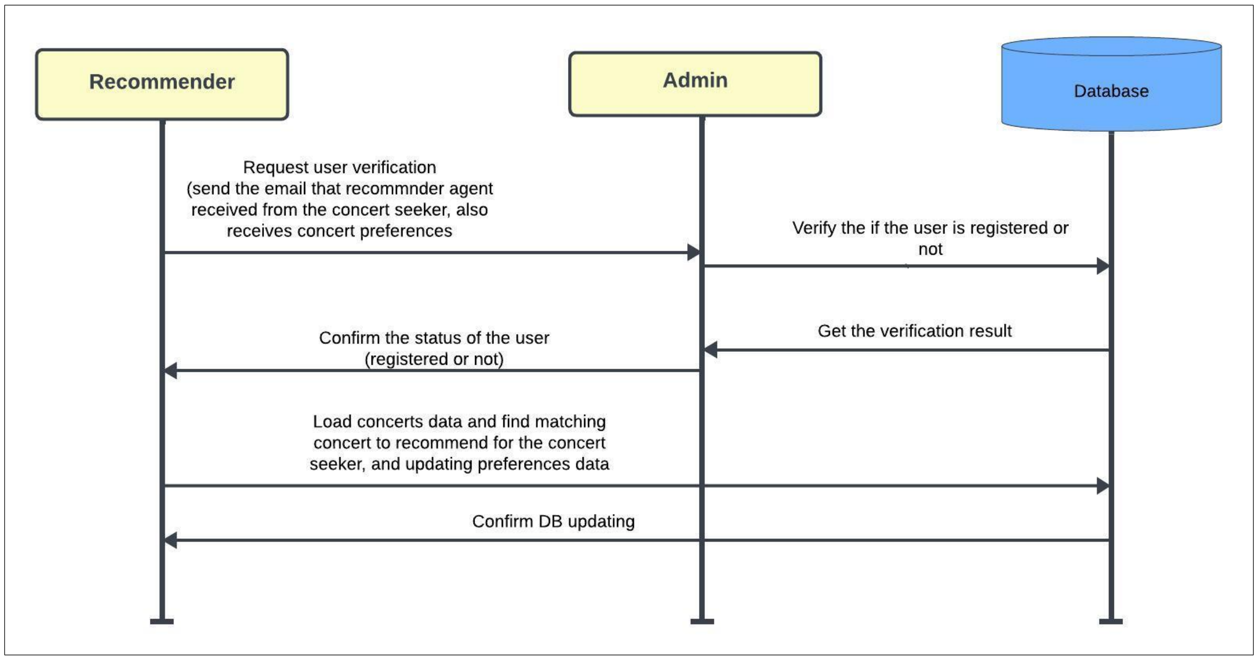
2.3 Interaction diagram for DataManager:


This agent in this phase received a concert information from the Venue or the service provider.
DataManager
Database
Insert concerts details to DB (location, tickect price, genre)
Confirm DB updating

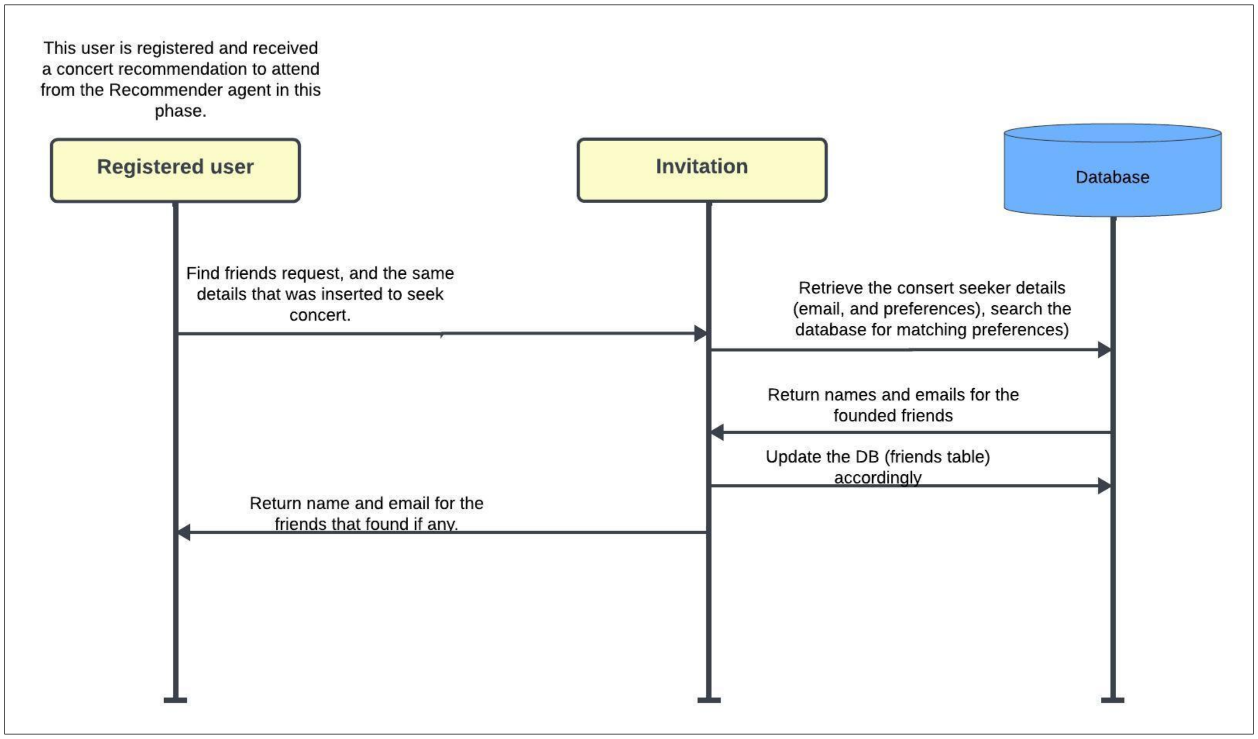
2.4 Interaction Diagram for ConcertSeeker Agent:


ConcertSeeker
Recommender
Admin
Database
SeekConcert (provides email, location preferences, ticket price, genre)
Send email to the admin agent to verify if the user exists
Check if the emails exists in the DB
Confirm to the Admin
Inform Recommender if the seeker is a registered user or not
If it is not register notify the seeker to register (create profile), else provide him with a concert recommendation according to the inserted preferences, or there is no available concert

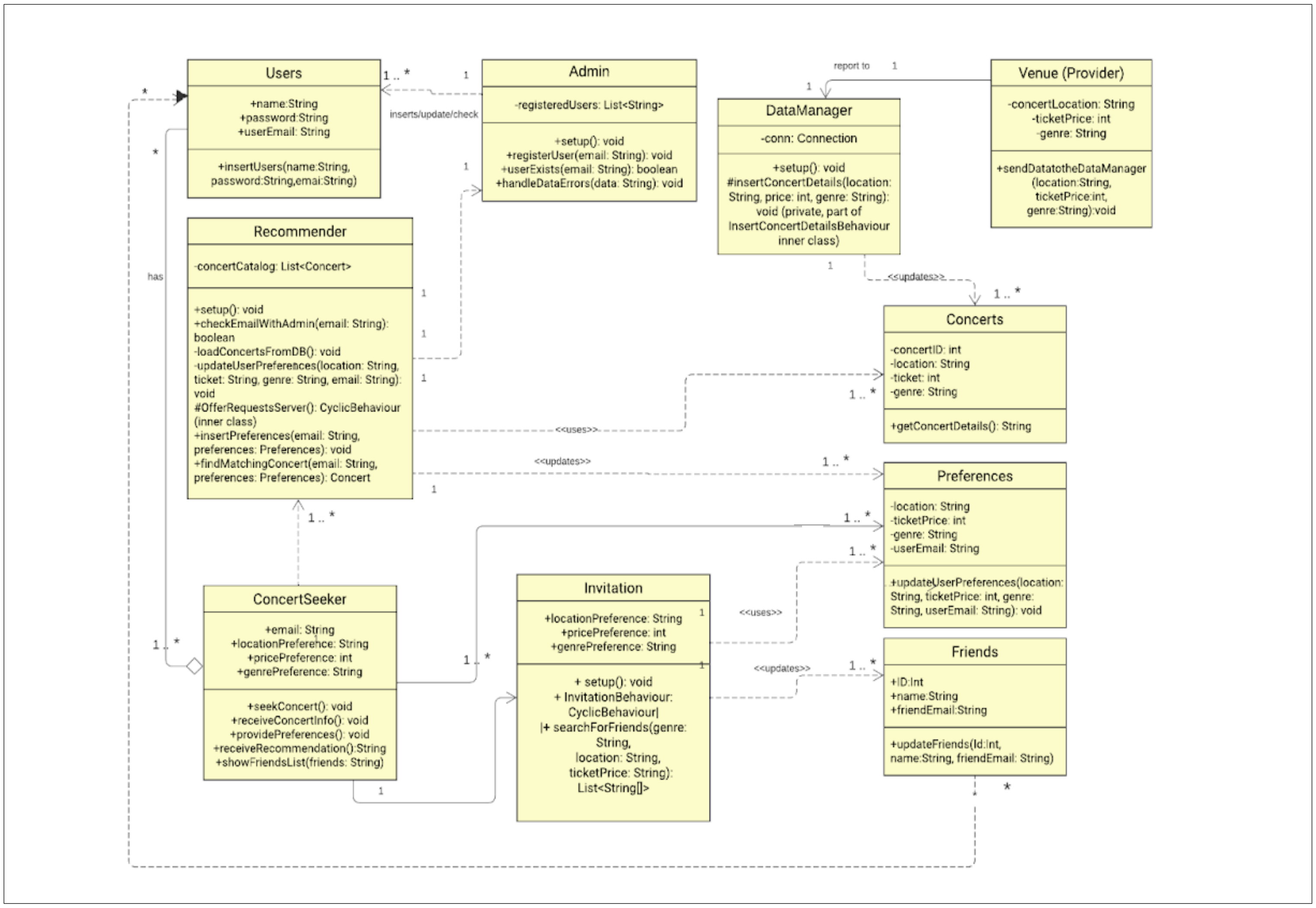
2.5 Interaction diagram for Recommender Agent:


Recommender
Admin
Database
Request user verification (send the email that recommnder agent received from the concert seeker, also receives concert preferences
Verify the if the user is registered or not
Get the verification result
Confirm the status of the user (registered or not)
Load concerts data and find matching concert to recommend for the concert seeker, and updating preferences data
Confirm DB updating

2.6 Interaction diagram for Invitation Agent:

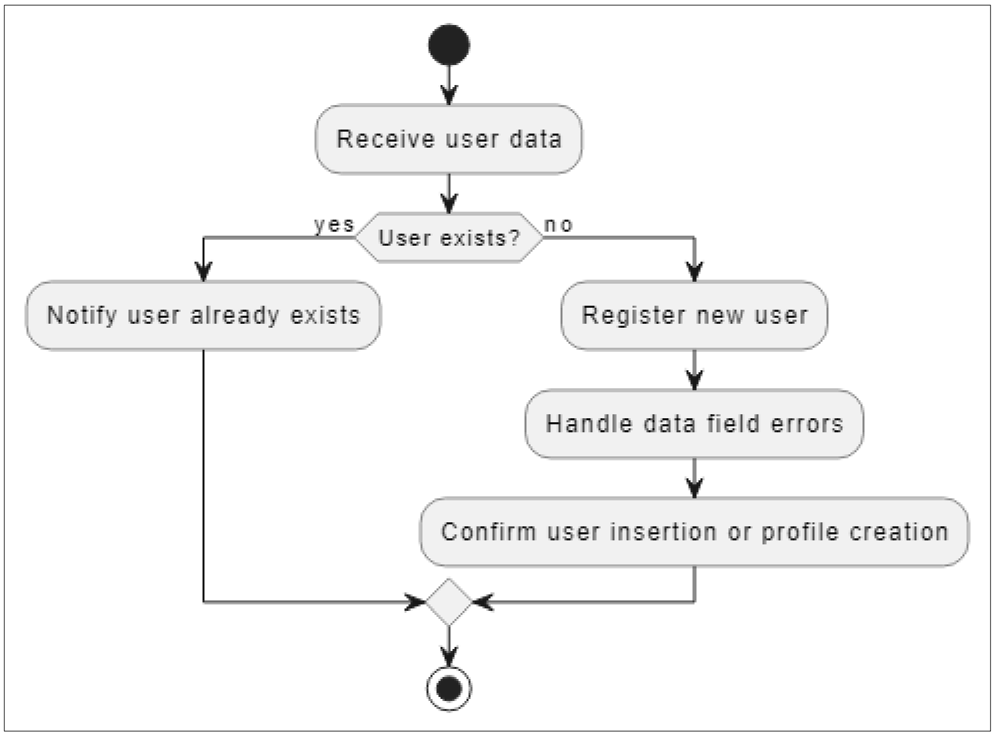

This user is registered and received a concert recommendation to attend from the Recommender agent in this phase.
Registered user
Invitation
Database
Find friends request, and the same details that was inserted to seek concert.
Retrieve the consert seeker details (email, and preferences), search the database for matching preferences)
Return names and emails for the founded friends
Update the DB (friends table) accordingly
Return name and email for the friends that found if any.

3. The class diagram: A UML class diagram depicts the essential component blocks of the system, including classes, their characteristics (attributes), functions (operations), and how they are interconnected. The next figure represents the class diagram for MCRS:

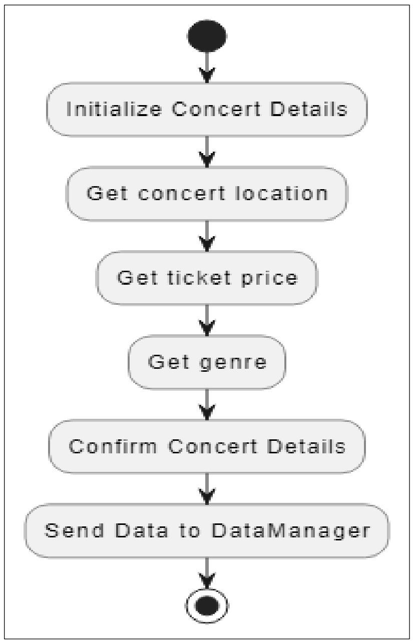

Users
+name:String
+password:String
+userEmail: String
+insertUsers(name:String, password:String,emai:String)
Admin
-registeredUsers: List<String>
+setup(): void
+registerUser(email: String): void
+userExists(email: String): boolean
+handleDataErrors(data: String): void
inserts/update/check
DataManager
-conn: Connection
+setup(): void
#insertConcertDetails(location: String, price: int, genre: String): void (private, part of InsertConcertDetailsBehaviour inner class)
report to
Venue (Provider)
-concertLocation: String
-ticketPrice: int
-genre: String
+sendDatatotheDataManager (location:String, ticketPrice:int, genre:String):void
Recommender
-concertCatalog: List<Concert>
+setup(): void
+checkEmailWithAdmin(email: String): boolean
-loadConcertsFromDB(): void
-updateUserPreferences(location: String, ticket: String, genre: String, email: String): void
#OfferRequestsServer(): CyclicBehaviour (inner class)
+insertPreferences(email: String, preferences: Preferences): void
+findMatchingConcert(email: String, preferences: Preferences): Concert
<<updates>>
Concerts
-concertID: int
-location: String
-ticket: int
-genre: String
+getConcertDetails(): String
<<uses>>
<<updates>>
Preferences
-location: String
-ticketPrice: int
-genre: String
-userEmail: String
+updateUserPreferences(location: String, ticketPrice: int, genre: String, userEmail: String): void
has
ConcertSeeker
+email: String
+locationPreference: String
+pricePreference: int
+genrePreference: String
+seekConcert(): void
+receiveConcertInfo(): void
+providePreferences(): void
+receiveRecommendation():String
+showFriendsList(friends: String)
Invitation
+locationPreference: String
+pricePreference: int
+genrePreference: String
+ setup(): void
+ InvitationBehaviour: CyclicBehaviour|
|+ searchForFriends(genre: String, location: String, ticketPrice: String): List<String[]>
<<uses>>
<<updates>>
Friends
+ID:Int
+name:String
+friendEmail:String
+updateFriends(Id:Int, name:String, friendEmail: String)
1 .. *
1
*

4. The Activity diagram: Activity diagrams are for visualizing and specifying the workflow and behavior of a system, especially for processes that involve multiple steps, decisions, and parallel activities.

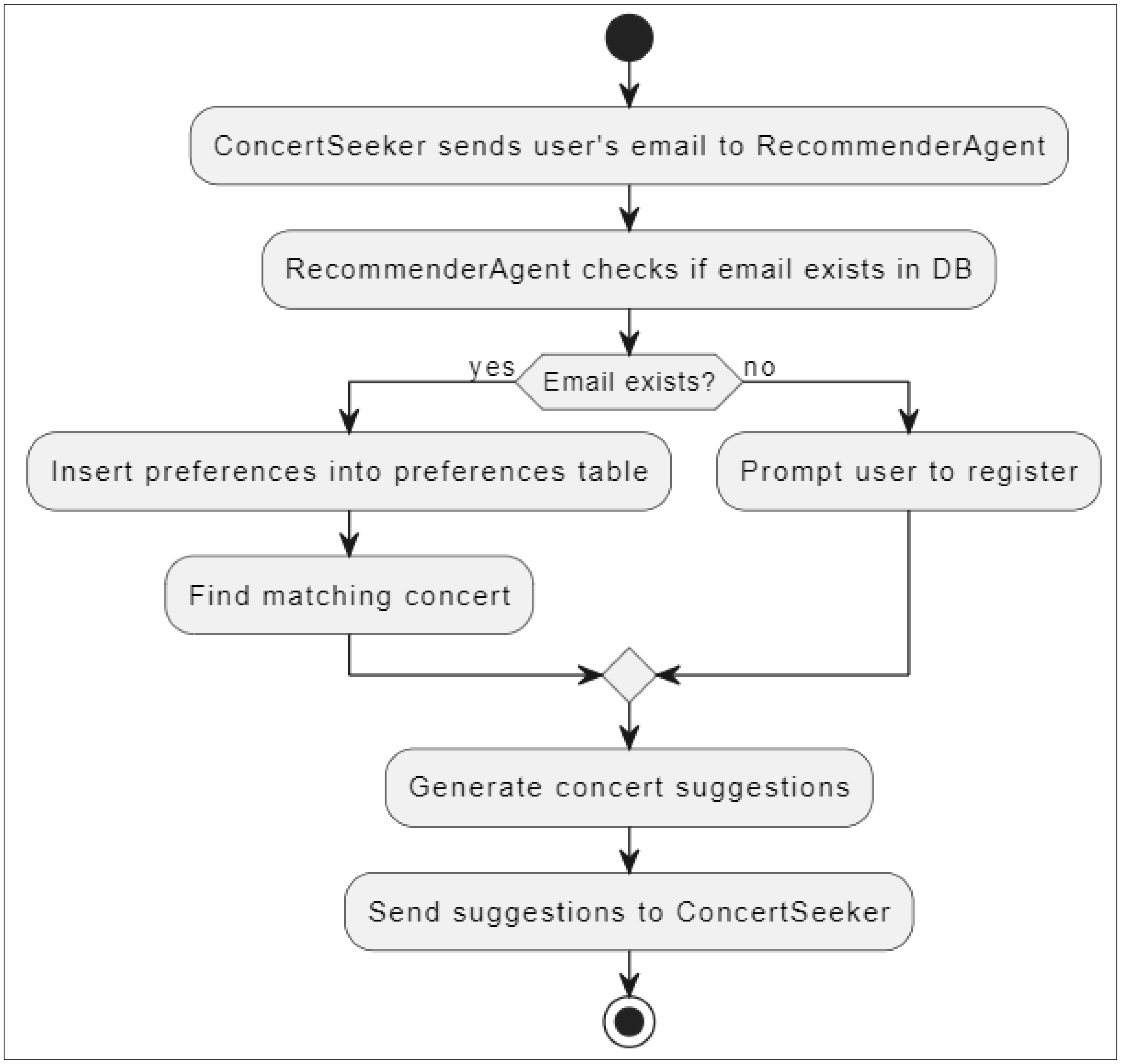
4.1 Admin Agent:



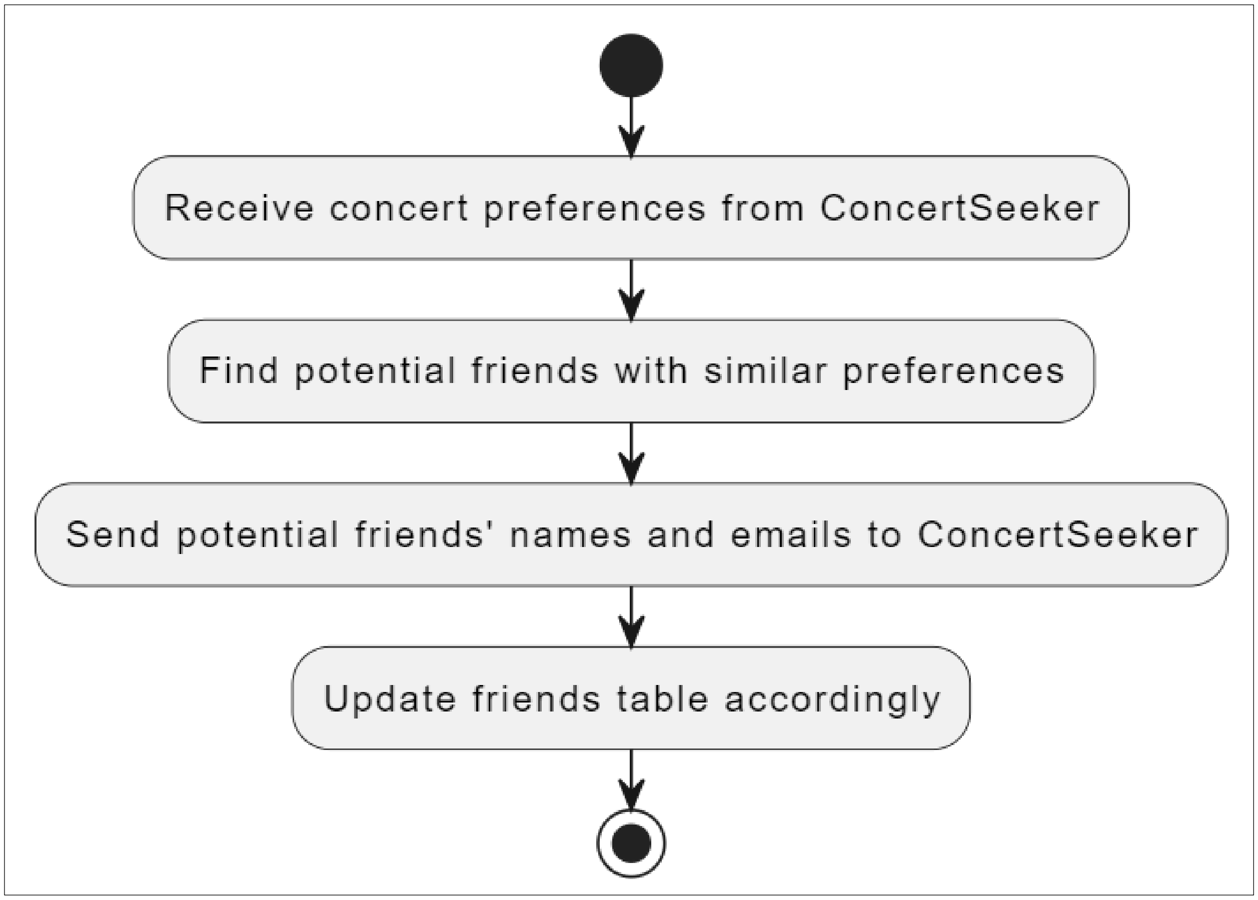
4.2 Venue Agent:



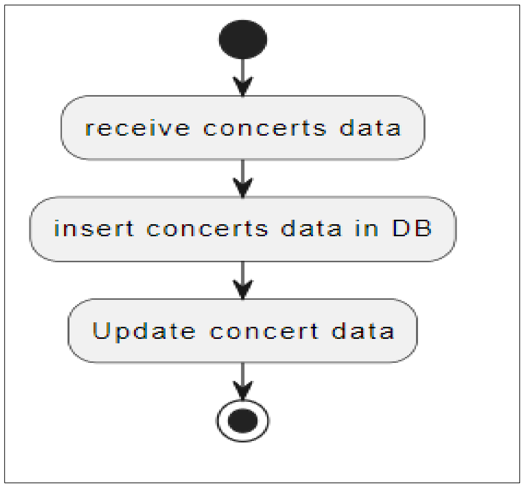
4.2 Recommender Agent:


ConcertSeeker sends user's email to RecommenderAgent
RecommenderAgent checks if email exists in DB
Email exists?
yes
no
Insert preferences into preferences table
Prompt user to register
Find matching concert
Generate concert suggestions
Send suggestions to ConcertSeeker

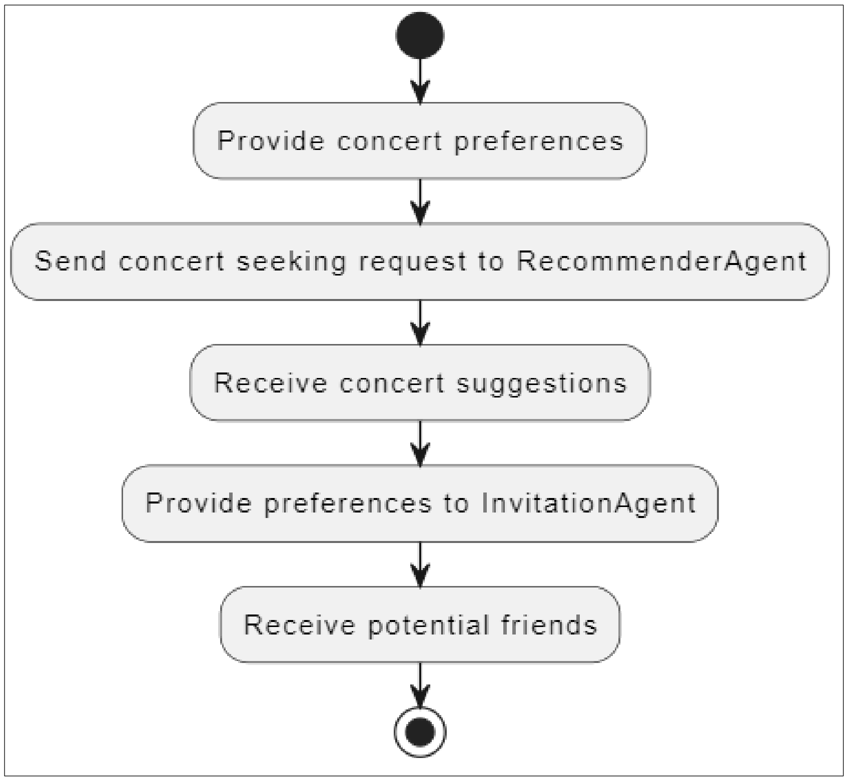
4.3 Invitation Agent**:**


Receive concert preferences from ConcertSeeker
Find potential friends with similar preferences
Send potential friends' names and emails to ConcertSeeker
Update friends table accordingly

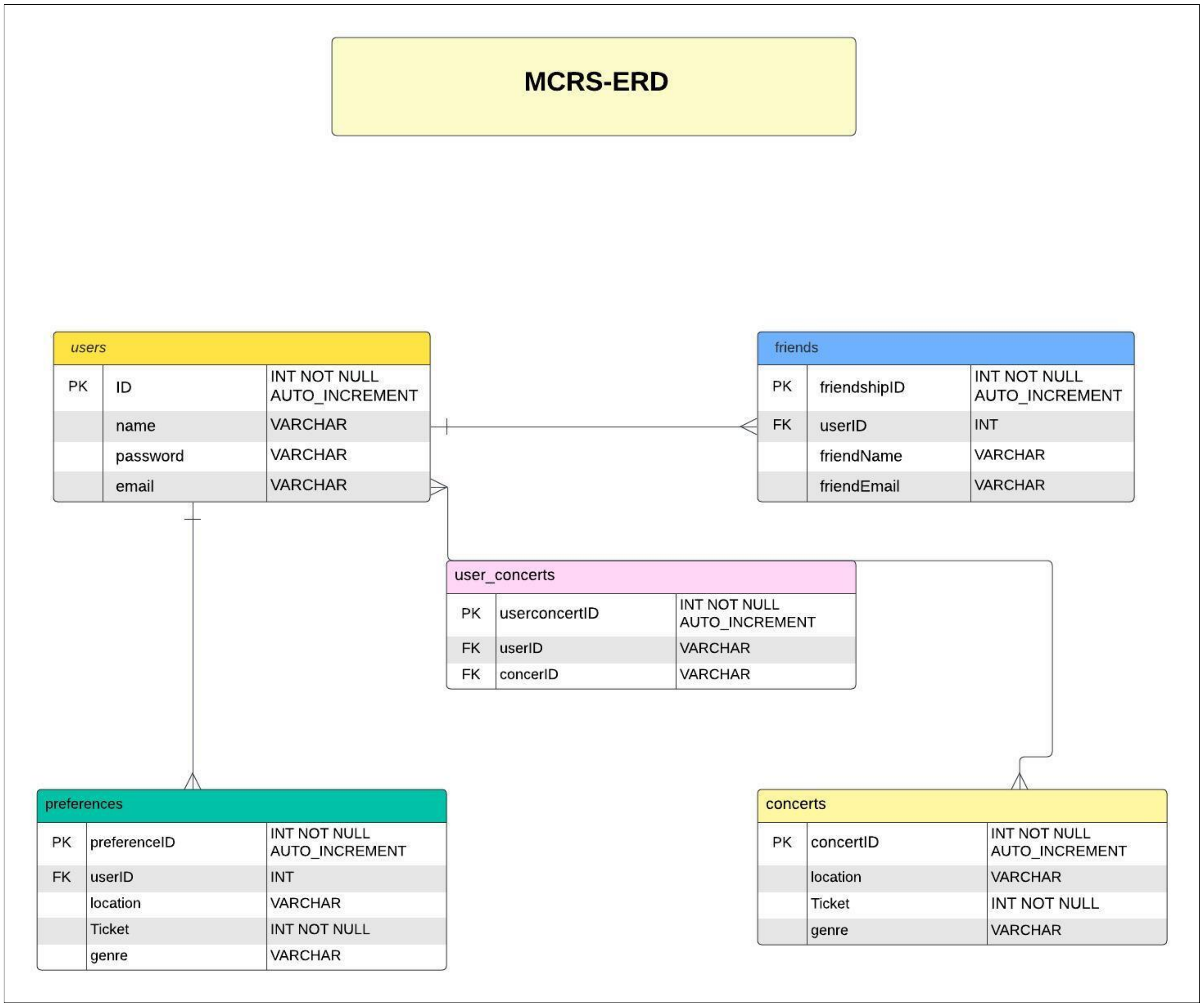
4.4 DataManager Agent:



4.5 ConcertSeeker Agent


Provide concert preferences
Send concert seeking request to RecommenderAgent
Receive concert suggestions
Provide preferences to InvitationAgent
Receive potential friends

5. E-R diagram: we are providing the knowledge sharing within our system through the E-R diagram, which stands for Entity Relation. This diagram shows the database tables and the relation between them. We also providing the SQL code that we will use to create this DB.



Relationships illustration:

- users and preferences: One-to-Many. One user can have many preferences, but each preference is linked to one user.
- users and friends: One-to-Many. One user can have many friends in the system, but each friend is linked to one user in the friends' table.
- We created the user-concerts table to create a relation between users and concerts table. The concerts table represents the concerts that the provider has in its DB, and our recommender agent will search it to find a matching concert to the user preferences.
- Within the user-concerts, we have the following:

- Each record represents a relationship between a user and a concert.
- The FOREIGN KEY constraints ensure referential integrity, meaning you can't have a record in user_concerts that refers to a non-existent user or concert.
- The UNIQUE constraint ensures that there aren't duplicate entries for the same user-concert relationship, i.e., you can't have two records indicating that the same user is attending the same concert.

MCRS-DB tables SQL code: (we are providing this section so you can check the constraints over the table fields accordingly)

Table name	**SQL code**
users	CREATE TABLE users (ID INT NOT NULL AUTO_INCREMENT, name VARCHAR(255) NOT NULL, password VARCHAR(255) NOT NULL, email VARCHAR(255) UNIQUE NOT NULL, PRIMARY KEY (ID));
preferences	CREATE TABLE preferences (preferenceID INT NOT NULL AUTO_INCREMENT, location VARCHAR(255) NOT NULL, Ticket INT NOT NULL, genre VARCHAR(255) NOT NULL, userID INT, PRIMARY KEY (preferenceID), FOREIGN KEY (userID) REFERENCES users(ID));
concerts	CREATE TABLE concerts (concertID INT NOT NULL AUTO_INCREMENT, location VARCHAR(255) NOT NULL, Ticket INT NOT NULL, genre VARCHAR(255) NOT NULL, PRIMARY KEY (concertID));
user_concerts	CREATE TABLE user_concerts (userConcertID INT NOT NULL AUTO_INCREMENT, userID INT NOT NULL, concertID INT NOT NULL, PRIMARY KEY (userConcertID), FOREIGN KEY (userID) REFERENCES users(ID),

	FOREIGN KEY (concertID) REFERENCES concerts(concertID), UNIQUE (userID, concertID));
friends	CREATE TABLE friends (friendshipID INT NOT NULL AUTO_INCREMENT, userID INT, friendName VARCHAR(255) NOT NULL, friendEmail VARCHAR(255) NOT NULL, PRIMARY KEY (friendshipID), FOREIGN KEY (userID) REFERENCES users(ID));

6. Inter-agent messages:

- Admin Agent:

Input	Output
<AdminAgentMessages> <input> <message> <performative>REQUEST</performative> <content action="CHECK_USER"> <userEmail>String</userEmail> </content> </message> </input> </AdminAgentMessages>	<AdminAgentMessages> <output> message> <performative>INFORM</performative> <content action="USER_EXISTS"> <userID>Int</userID> </content> </message> <!—Output when user is not found --> <message> <performative>FAILURE</performative> <content action="USER_NOT_FOUND"/> </message> <!—Output if DB error occurs --> <message> <performative>ACCEPT_PROPOSAL</performative> <content action="DATABASE_ERROR"/> </message> </output> </AdminAgentMessages>

- Venue Agent

Input	Output
<VenueAgentMessages>	<VenueAgentMessages>
<input>	<output>
<message>	<message>
<performative>REQUEST</performative>	<performative>INFORM</performative>
<content>	<receiver>DataManagerAgent</receiver>
<concertLocation>String</concertLocation>	<content>
<ticketPrice>Integer</ticketPrice>	<concertLocation>String</concertLocation>
<genre>String</genre>	<ticketPrice>Integer</ticketPrice>
</content>	<genre>String</genre>
</message>	</content>
</input>	</message>
</VenueAgentMessages>	</output>
	</VenueAgentMessages>

- DataMAnager Agent:

Input	Output
<dataManagerAgentMessages>	<dataManagerAgentMessages>
<input>	<output>
<message>	<message>
<performative>REQUEST</performative>	<performative>CONFIRM</performative>
<sender>VenueAgent</sender>	<receiver>VenueAgent</receiver>
<content>	<content>
<location>String</location>	<confirmation>Concert details inserted successfully</confirmation>
<price>Integer</price>	</content>
<genre>String</genre>	</message>
</content>	<!-- Include additional messages for different scenarios such as errors -->
</message>	</output>
</input>	</dataManagerAgentMessages>
</dataManagerAgentMessages>	

- Concert Seeker:

Input	Output
<concertSeekerAgentMessages>	<concertSeekerAgentMessages>
<input>	<output>
<!-- Response from Recommender agent with a concert	<message>
recommendation -->	<performative>CFP</performative>
<message>	<receiver>Recommender</receiver>
<performative>PROPOSE</performative>	<content>
<sender>Recommender</sender>	<email>User's email</email>
<content>	<location>Preferred concert location</location>
<concertDetails>String</concertDetails>	<price>Maximum ticket price</price>
</content>	<genre>Preferred genre</genre>
</message>	</content>
<message>	</message>
<performative>REFUSE</performative>	<message>
<sender>Recommender</sender>	<performative>REQUEST</performative>
<content>	<receiver>InvitationAgent</receiver>
<reason>String</reason>	<content>
</content>	<preferences>String</preferences>
</message>	</content>
<!-- Inform message from InvitationAgent with friends	</message>
list -->	</output>
<message>	</concertSeekerAgentMessages>
<performative>INFORM</performative>	
<sender>InvitationAgent</sender>	
<content>	
<friendsList>String</friendsList>	
</content>	
</message>	
<!-- Failure message from InvitationAgent when no	
friends found -->	
<message>	
<performative>FAILURE</performative>	
<sender>InvitationAgent</sender>	
<content>	
<reason>String</reason>	
</content>	
</message>	
</input>	
</concertSeekerAgentMessages>	

-Recommender:

Input	Output
<RecommenderAgentMessages>	<RecommenderAgentMessages>
<input>	<output>
<message>	<message>
<performative>CFP</performative>	<performative>PROPOSE</performative>
<sender>ConcertSeekerAgent</sender>	<receiver>ConcertSeekerAgent</receiver>
<content>	<content>
<email>String</email>	<concertID>Integer</concertID>
<location>String</location>	<location>String</location>
<ticketPrice>Integer</ticketPrice>	<ticketPrice>Integer</ticketPrice>
<genre>String</genre>	<genre>String</genre>
</content>	</content>
</message>	</message>
</input>	<!-- Refuse message when no matching concert is found -->
</RecommenderAgentMessages>	<message>
	<performative>REFUSE</performative>
	<receiver>ConcertSeekerAgent</receiver>
	<content>String</content>
	</message>
	<!-- Failure message when user check fails -->
	<message>
	<performative>FAILURE</performative>
	<receiver>ConcertSeekerAgent</receiver>
	<content>String</content>
	</message>
	</output>
	</RecommenderAgentMessages>

- Invotation Agent:

Input	Output
<invitationAgentMessages> <input> <message> <performative>REQUEST</performative> <sender>ConcertSeekerAgent</sender> <content> <genre>String</genre> <location>String</location> <ticketPrice>Integer</ticketPrice> <seekerEmail>String</seekerEmail> </content> </message> </input> </invitationAgentMessages>	<invitationAgentMessages> <output> <!-- Inform message with friends list --> <message> <performative>INFORM</performative> <receiver>ConcertSeekerAgent</receiver> <content> <friendsList> <friend> <name>String</name> <email>String</email> </friend> <!-- Repeat the <friend> element for each friend found --> <friend> <name>String</name> <email>String</email> </friend> . n-times . </friendsList> </content> </message> <!-- Failure message when no friends are found --> <message> <performative>FAILURE</performative> <receiver>ConcertSeekerAgent</receiver> <content>String</content> </message> </output> </invitationAgentMessages>