TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

GameConnect Web API

Project Report

Alya Barcous

BA/IT student

IT-325 Web Services

Sunday 21st January, 2024

Abstract

The Gaming Sessions Management System is a web-based application designed to enhance the gaming experience for Tunisian gamers by providing a centralized platform for joining gaming sessions across various titles. The project aims to create a fun and interactive space where gamers can easily discover, join, and participate in gaming events while connecting with fellow users who share similar interests.

Key Features

1. **Connecting Tunisian Gamers:**
   - The primary purpose of this project is to foster a sense of community among Tunisian gamers, it serves as a hub where gamers can come together to share their passion for gaming.
2. **Diverse Gaming Sessions:**
   - The system supports gaming sessions for a variety of games, offering a diverse range of options for users to explore and participate in.
3. **User-Friendly Interaction:**
   - The user interface is designed to be intuitive and user-friendly, ensuring a seamless experience for both new and experienced gamers.
4. **Enhanced Gaming Event Management:**
   - Administrators have robust tools for managing gaming sessions, allowing them to update, and remove events, gamers and reviews effortlessly..
5. **Reviews and Recommendations:**
   - Gamers can share their experiences and insights through reviews, providing valuable feedback to the gaming community.
   - The review system allows users to make informed decisions about joining specific gaming sessions.
6. **Like System for Popular Sessions:**
   - The inclusion of a like system allows users to express their interest in particular gaming sessions.
   - Popular sessions receive recognition, creating a dynamic environment where users can discover trending and highly-rated events.
7. **Security and User Privacy:**
   - The project prioritizes user security by implementing secure authentication mechanisms and password hashing.
   - User privacy is respected, creating a trustworthy environment for gamers to engage.

Contents

Chapter 1

Introduction

This report presents a gaming web API that was developed using **Python** and the **FastAPI** framework. The API utilizes a **Postgres SQL** database to store and manage gamer information and gaming session data. One of the main features of the API is user authentication, which ensures that only authorized users can access the gaming platform. Gamers have the ability to create, update, and delete gaming sessions. They can also set the maximum number of participants for a session and choose to make it unavailable even if the maximum number of attendees hasn't been reached. Additionally, gamers can join a gaming session and have the option to leave it. These features are currently not available in any app in the Tunisian market, making the API a unique and more enjoyable way for Tunisian gamers to engage. Another important feature is the ability for users to like and review the sessions they have attended. Overall, the API aims to enhance the gaming experience and provide a new means of communication for Tunisian gamers.

Chapter 2

Conceptual Design

The main actors in this system are the "Gamer" and the "Admin". The Gamer can either be a game session creator or someone who wants to join a gaming session. The Admin is responsible for managing the system.

2.1 Gamer

- Create an account and a Gaming Session
- Can manage gaming sessions by deleting, updating, and seeing details of his sessions only.
- Can see all the available sessions
- Can join or leave a gaming session
- Can turn off session availability if he does not want to receive more participants even if there are places.

2.2 Admin

- See all the user account details.
- See all the gaming sessions
- See all reviews
- Can delete any gamer, session, or review.

2.3 UML Use Case Diagram

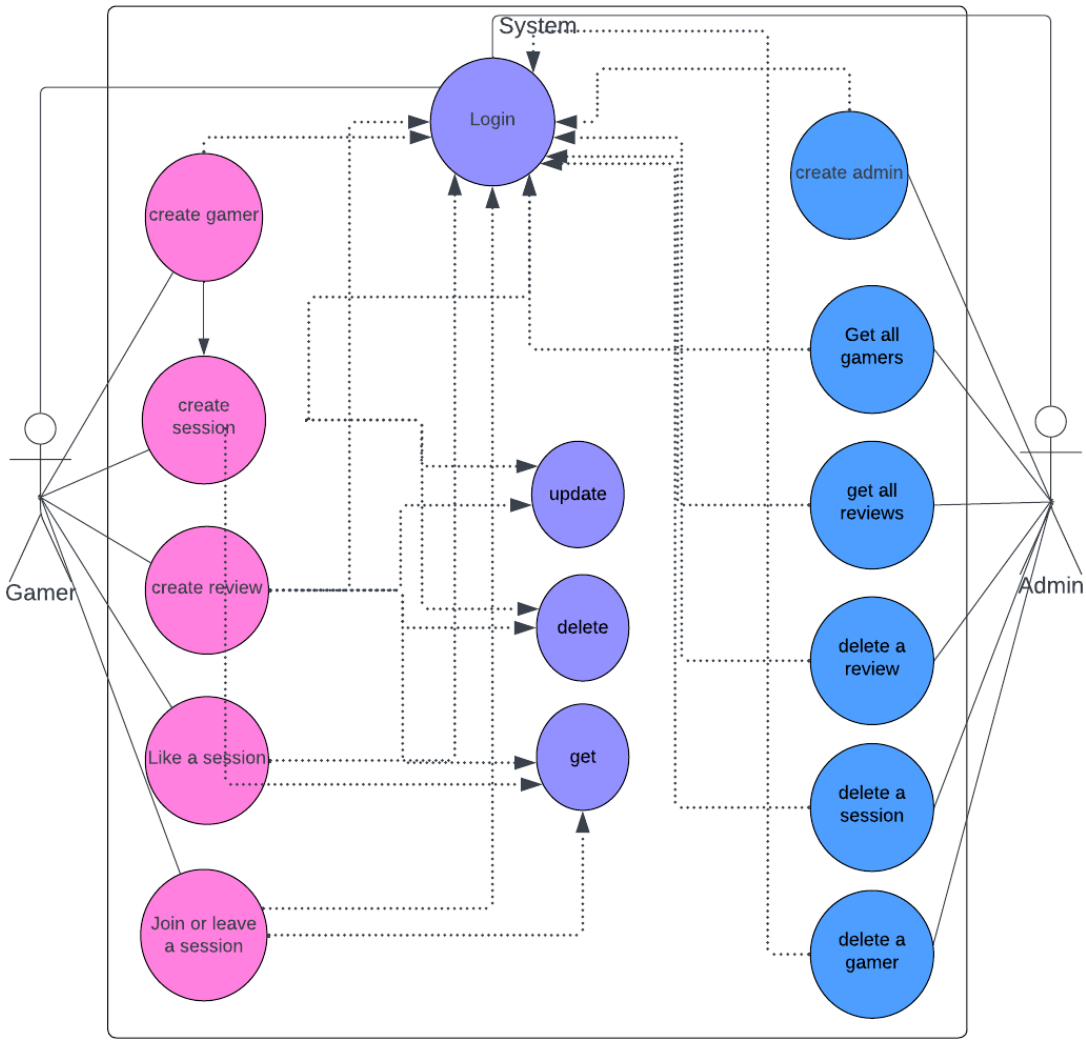


Figure 2.1: UML Use Case Diagram

Chapter 3

Implementation and Contribution

We introduce the creation and deployment of a gaming Web API using **FastAPI**, **Python**, **SQLAlchemy**, and **PostgreSQL**. The selection of these technologies was driven by various considerations.

To begin with, **FastAPI** stands out as a contemporary, swift, and effective framework designed for API development. Leveraging the **Python** standard library, it ensures ease of use for developers familiar with **Python**. Moreover, it is constructed upon **Starlette**, an asynchronous web framework, enabling high performance and scalability.

Python, being a widely adopted and versatile programming language, is well-suited for a broad spectrum of applications, including web development. Its extensive ecosystem of libraries and frameworks makes it an excellent choice for crafting intricate and feature-rich applications.

SQLAlchemy, a renowned Object-Relational Mapping (ORM) library for Python, simplifies database interactions, offering developers a more convenient and efficient approach to working with data. By abstracting the complexities of database interactions, it provides a straightforward, **Pythonic API**.

Lastly, **PostgreSQL**, an open-source relational database management system, was chosen for its robustness, scalability, and performance. Its reliability and feature-rich nature make it an apt selection for managing and storing data in the context of the Gaming API project.

Below is a list of most important packages and frameworks used for the project:

- fastapi 0.88.0
- fastapi-mail 1.1.4
- psycopg2 2.9.5
- pydantic 1.10.2
- SQLAlchemy 1.4.45

- starlette 0.22.0
- uvicorn 0.20.0
- alembic 1.9.1

3.1 OAuth2 Contribution

Implemented to facilitate secure third-party authentication, enhancing user login options and overall account security.

```
oauth2_scheme = OAuth2PasswordBearer(tokenUrl='login' )
```

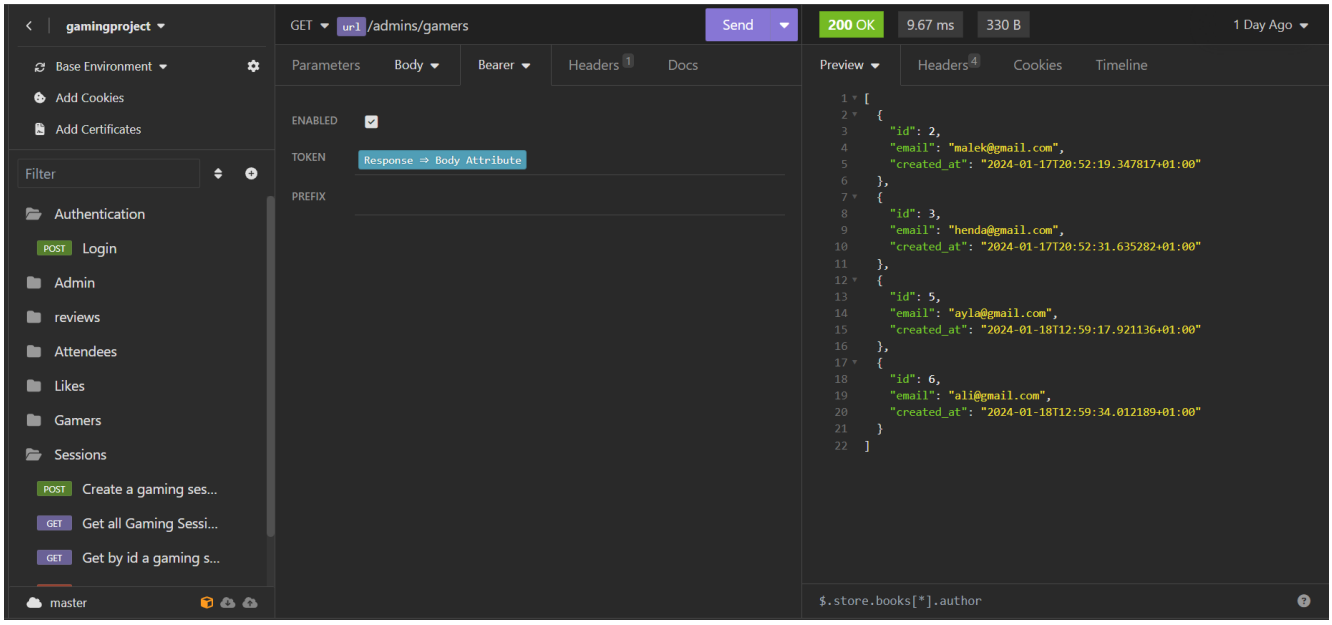
3.2 HTTP Methods Contribution

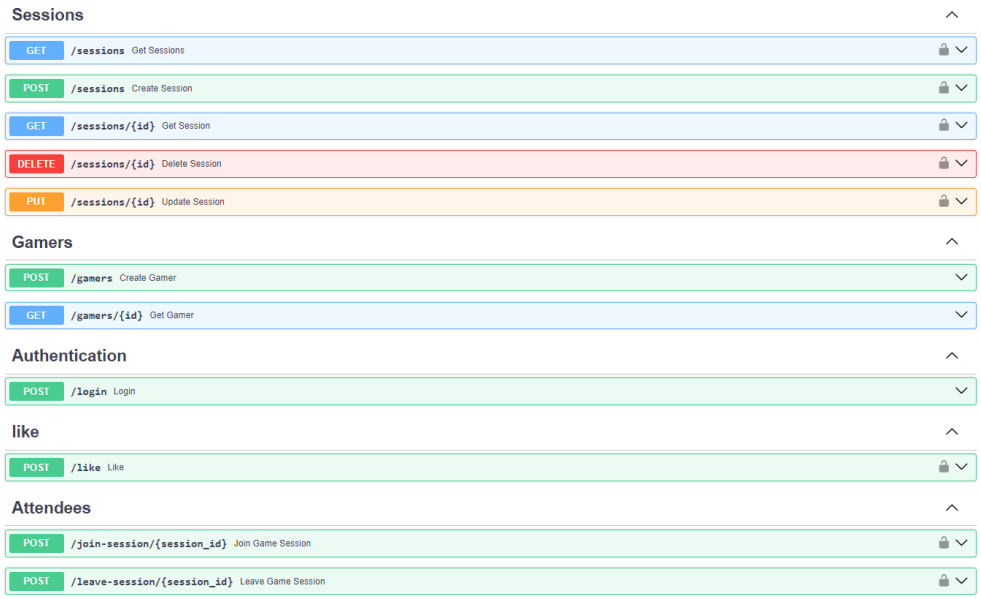
Leveraged various HTTP methods (GET, POST, PUT, DELETE) to enable standard RESTful operations for managing gaming sessions, providing a comprehensive API.

```
@router.post("/gamers", status_code=status.HTTP_201_CREATED, response_model=schemas.GamerOut)
@router.get('/gamers/{id}', response_model=schemas.GamerOut)
@router.put("/sessions/{id}", response_model=schemas.Session)
@router.delete("/sessions/{id}", status_code=status.HTTP_204_NO_CONTENT)
```

3.3 Insomnia and Swagger Contribution

Employed for API testing and documentation, streamlining the development process and ensuring clear communication of endpoints and functionalities.



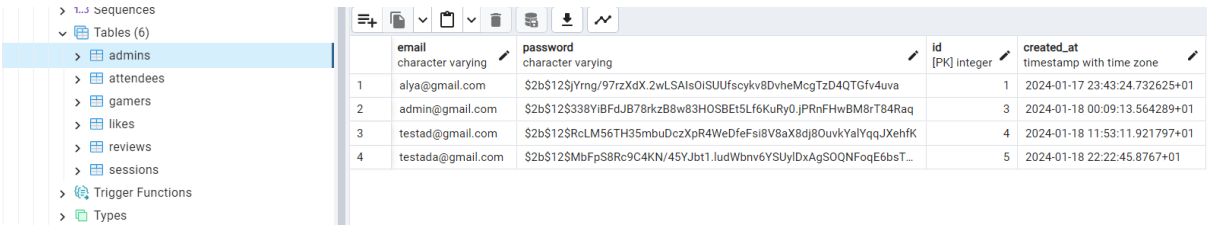


3.4 Git Contribution

Utilized for version control, enabling collaborative development, tracking changes, and providing a structured history of the project.

3.5 PostgreSQL Contribution

Chosen as the relational database management system to store and manage gaming session data efficiently.



Chapter 4

Database Design

The database is stored using PostgreSQL and has the following tables:

- **Table Admins:**
  - Stores the information of the Admins.
- **Table Gamers:**
  - Stores the information of the Gamers.
- **Table Sessions:**
  - Stores information of the gaming sessions. It has organizer_id as a foreign key from table Gamers.
- **Table Attendees:**
  - Has gamer_id as a foreign key from table Gamers and session_id as a foreign key from table Sessions.
- **Table Reviews:**
  - Stores the reviews written by Gamers. It has gamer_id as a foreign key from table Gamers and session_id as a foreign key from table Sessions.
- **Table Likes:**
  - Has gamer_id as a foreign key from table Gamers and session_id as a foreign key from table Sessions.

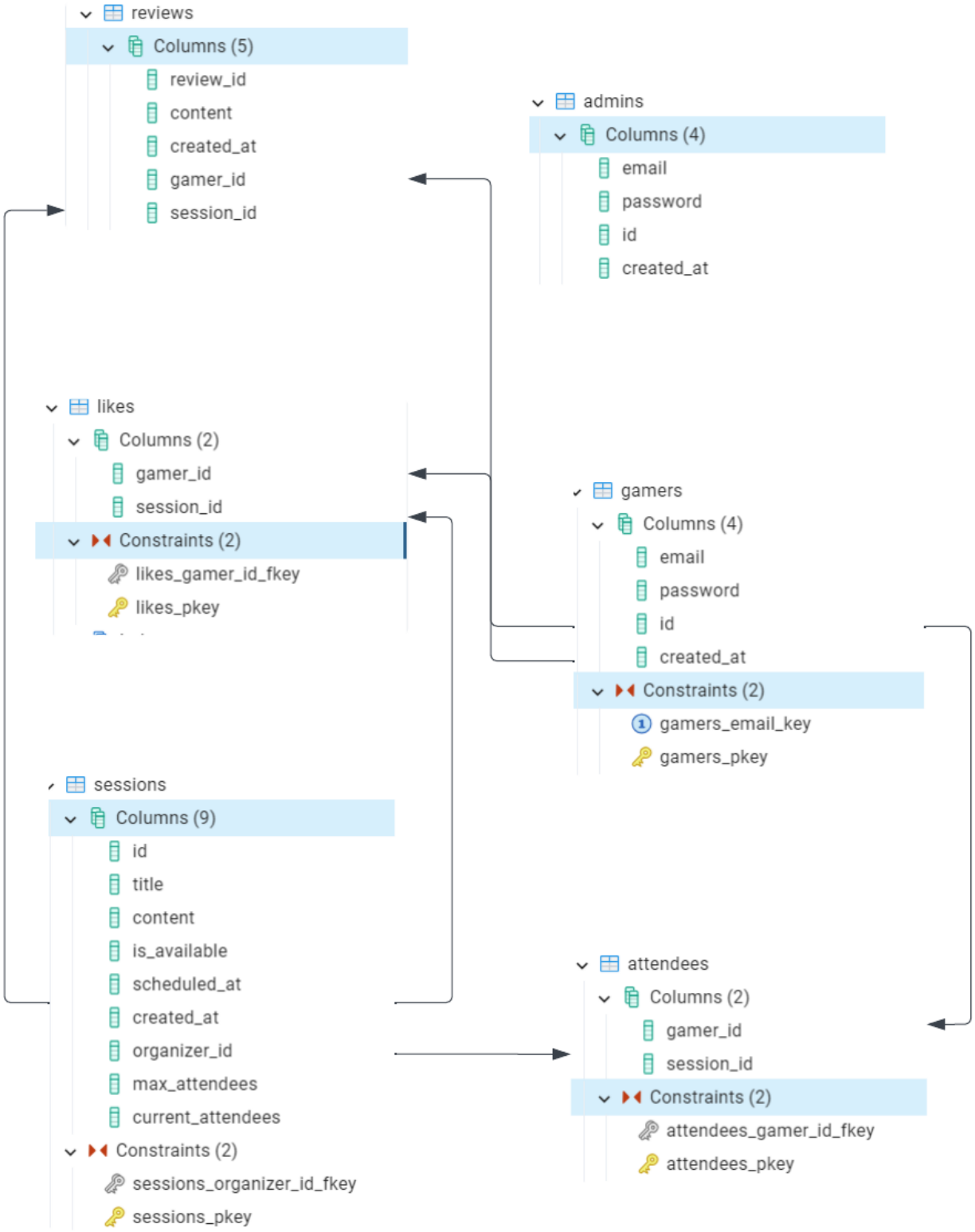


Figure 4.1: Database Design

Chapter 5

API Endpoints

5.0.1 Admin Endpoints

All the endpoints are protected with an authorization policy except /admin

- **/admin:** Admins can create a new account using this endpoint by providing an admin object that contains their information. The endpoint will check if the provided email already exists in the system. If it does, it will return a message: "Admin with this email already exists" and status code 400. If the provided information is valid, the endpoint will hash the provided password and create a new admin. Upon successful creation, it will return a message indicating the successful creation of the admin, along with its email and access-token.

- **/admins/gamers:** The admin can use this endpoint to get a list of all registered gamers. When the request is successful, it will return a list of all gamers along with a status code of 200. However, if no gamers are found, it will return a status code of 404 to indicate that no gamers were found.

- **/admins/pools:** This endpoint allows an admin to retrieve all available pools. It returns a list of all pools and a status code of 200. If no pools are found, it returns a message indicating that no pools were found and a status code 404.

- **/admins/gamers/id:** This endpoint allows an admin to delete a specific gamer by providing the gamer's id. It returns a status code 202 indicating that the gamer has been deleted successfully. If the gamer is not found, it returns a message and a status code 404 indicating that the gamer with that specific id was not found.

- **/admins/reviews:** The admin can use this endpoint to retrieve all the reviews. When the request is successful, it will return a list of all reviews made along with a status code of 200. However, if no reviews are found, it will return a message stating that no reviews were found, along with a status code of 404.

- **/admins/sessions/id:** By using this endpoint, the admin can delete a specific gaming session by providing the session's id. After successful deletion, it will return a status code of 202. However, if the session is not found, it will return a message indicating that the session was not found, along with a status code of 404.

- **/admins/reviews/review_id:** The admin can use this endpoint to delete a review by providing the review's id. If the deletion is successful, it will return a status

code of 202. However, if the review is not found, it will return a message indicating that the review was not found, along with a status code of 404.

5.0.2 Authentication Endpoints

- **/login:** This endpoint handles POST requests and verifies the provided credentials as either admin or gamer. If the validation is successful, it issues a JSON web token to the user. The token determines the user's access permissions, granting entry to specific endpoints based on whether it provides privileges exclusively to gamers, admins, or both.

5.0.3 Gamer Endpoints

All the endpoints are protected with authorization policy except /gamers.

- **/gamers:** Gamers can create a new account using this endpoint by providing a gamer object that contains their information. The endpoint will check if the provided email already exists in the system. If it does, it will return a message: "Gamer with this email already exists" and status code 400. If the provided information is valid, the endpoint will hash the provided password and create a new gamer. Upon successful creation, it will return a message indicating the successful creation of the gamer, along with their email and access-token.

- **/gamers/id:** This endpoint allows gamers to retrieve their account information by providing their id. It checks if the provided id exists in the system. If not, it returns an appropriate message and status code. If the gamer is found, it returns their account information along with an appropriate message and a status code 200 OK.

5.0.4 Attendee Endpoints

All the endpoints are protected with an authorization policy.

- **/join-session/session_id:** This endpoint allows gamers to join a gaming session by submitting a POST request with the session ID. The endpoint first checks the existence and availability of the specified gaming session. If the session is not found or is no longer available, it raises a 404 or 400 HTTPException with a relevant detail message. Additionally, it verifies whether the gamer is already attending the session or if the session is already full, raising appropriate HTTPExceptions in such cases. Upon successful validation, the endpoint creates a new attendee record, increments the current attendees count, and commits the changes to the database. The response includes a message indicating successful participation in the gaming session with a status code of 200 (OK).

- **/leave-session/session_id:** This endpoint allows gamers to exit a gaming session by submitting a POST request with the session ID. The endpoint checks whether the specified gaming session exists and if the current gamer is attending. If the session is not found or the gamer is not attending, it raises a 404 or 400 HTTPException with an appropriate detail message. Upon successful validation, the endpoint removes the attendee record associated with the current gamer, decrements the current

attendees count, and commits the changes to the database. The response includes a message indicating successful departure from the gaming session with a status code of 200 (OK).

5.0.5 Sessions Endpoints

All the endpoints are protected with an authorization policy.

- **/sessions:** This endpoint allows users to schedule gaming events by submitting a POST request with session details. The parameters, including game title, date, and maximum attendees, are validated, with a specific check ensuring that the maximum attendees are specified and greater than zero. Upon validation, the gaming session is created, with the organizer identified as the logged-in user and initial attendees set to zero. The endpoint returns the newly created session with a status code of 201 (Created).

- **/sessions/id:** This endpoint allows users to retrieve details of a specific gaming session by sending a GET request with the session ID. The endpoint queries the database, including a count of likes associated with the session, and returns the session details. If the specified session ID is not found, it raises a status code of 404 (Not Found) and an appropriate error message.

- **/sessions/delete/id:** This endpoint allows users to remove a gaming session by sending a DELETE request with the session ID specified in the URL parameters. The endpoint validates the existence of the session and ensures that it is owned by the currently logged-in user. If the session is not found or the user is not the organizer, it raises the appropriate status code of 404 (Not Found) or 403 (Forbidden), along with an informative detail message. Upon successful validation, the endpoint deletes the session from the database, commits the changes, and returns a response with a status code of 204 (No Content).

- **/sessions/update/id:** This endpoint allows users to modify a gaming session by sending a PUT request with the session ID and updated details. The endpoint validates the existence of the session and checks if the logged-in user is the organizer. If the session is not found or the user is not authorized, it raises the corresponding error along with a descriptive detail message. Upon successful validation, the endpoint updates the session with the provided details in the database, commits the changes, and returns the updated session information.

5.0.6 Reviews Endpoints

All the endpoints are protected with an authorization policy.

- **/review/session_id:** This endpoint allows gamers to submit reviews for a gaming session by sending a POST request with the session ID and review content. The endpoint checks whether the specified gaming session exists and if the current gamer has already reviewed it. If the session is not found or the gamer has already submitted a review, it raises a 404 or 400 HTTPException with relevant detail messages. Upon successful validation, the endpoint creates a new review with the provided content, associates it with the current gamer and session, and commits

the changes to the database. The response includes a message indicating successful review creation with a status code of 201 (Created).

- **/review/delete/session_id:** This endpoint allows gamers to remove their previously submitted reviews for a gaming session by sending a DELETE request with the session ID. The endpoint checks whether the specified gaming session exists and if the current gamer has previously reviewed it. If the session is not found or the gamer has not submitted a review, it raises a 404 HTTPException with relevant detail messages. Upon successful validation, the endpoint deletes the existing review associated with the current gamer and session, committing the changes to the database. The response includes a message indicating successful review deletion with a status code of 204 (No Content).

- **/review/update/session_id:** This endpoint allows gamers to modify their previously submitted reviews for a gaming session by sending a PUT request with the session ID and updated review content. The endpoint checks whether the specified gaming session exists and if the current gamer has previously reviewed it. If the session is not found or the gamer has not submitted a review, it raises a 404 HTTPException with relevant detail messages. Upon successful validation, the endpoint updates the content of the existing review associated with the current gamer and session, committing the changes to the database. The response includes a message indicating successful review update with a status code of 200 (OK).

5.0.7 Likes

All the endpoints are protected with an authorization policy.

- **/like/session_id/dir:** By submitting a POST request with like details, the endpoint first checks whether the specified session exists. If the session is not found, it raises a 404 (Not Found) HTTPException with a relevant detail message. It then queries the database to determine if the current gamer has already liked the session. If the gamer has not liked the session and the like direction is positive (dir == 1), a new like is created, and the changes are committed to the database. If the like direction is negative (dir != 1), the endpoint checks if the gamer has previously liked the session. If a like is found, it is deleted, and the changes are committed. The response includes a message indicating successful like addition or deletion with a status code of 201 (Created). If the gamer has already liked the session, a 409 (Conflict) HTTPException is raised with a relevant detail message. If attempting to delete a nonexistent like, a 404 (Not Found) HTTPException is raised with an appropriate detail message.

Chapter 6

Discord Bot Implementation

6.0.1 Functionality Overview

The Discord bot, integrated into the project using the Discord.py library, introduces interactive and entertaining features for users within the Discord platform. The bot responds to specific commands, providing a variety of functionalities designed to enhance user engagement.

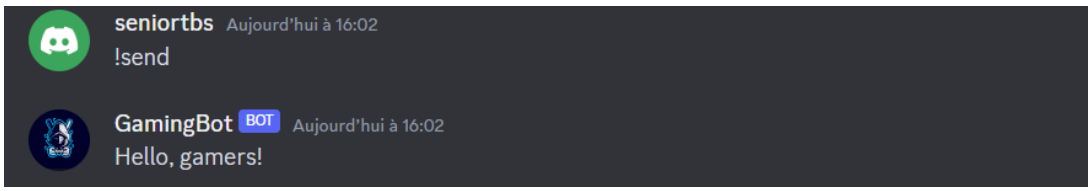
6.0.2 Integration with Existing Components

This Discord bot seamlessly integrates with different components of the project:

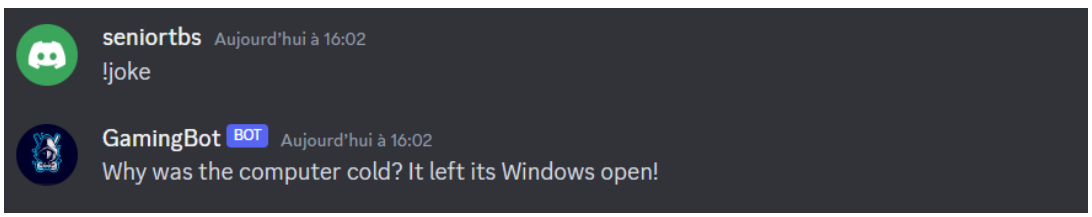
- **Command Handling:** Users can trigger commands such as `!send`, `!joke`, `!countdown`, `!random`, and `!motivate` to perform various actions directly from the Discord interface.
- **Real-time Interaction:** The bot responds to user commands in real-time, creating an interactive and dynamic experience for Discord users.

6.0.3 Commands and Interactions

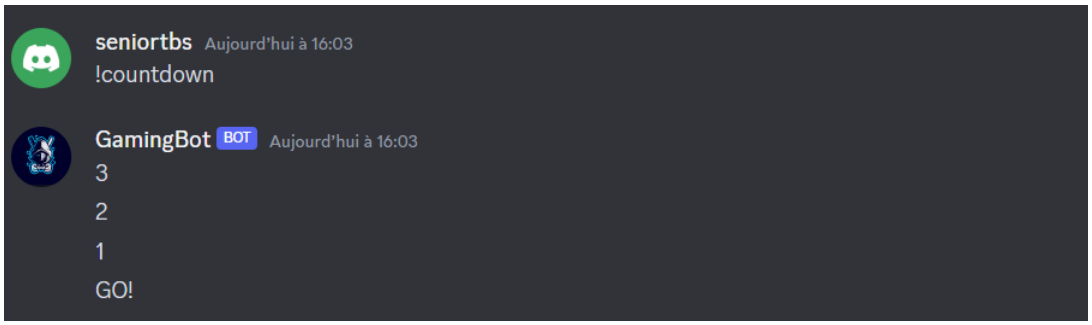
- `!send`: Sends a friendly greeting message to users, fostering a positive and welcoming environment.



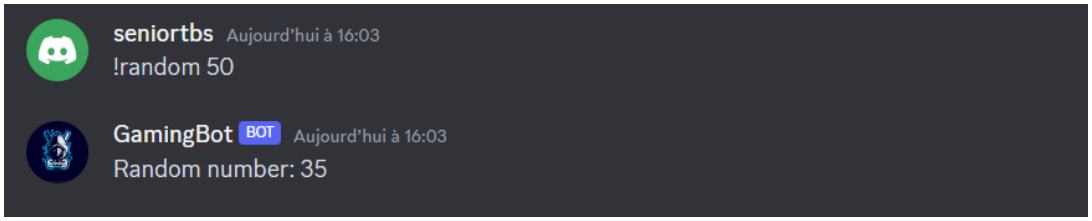
- `!joke`: Shares humorous gaming-related jokes to lighten the mood and entertain users.



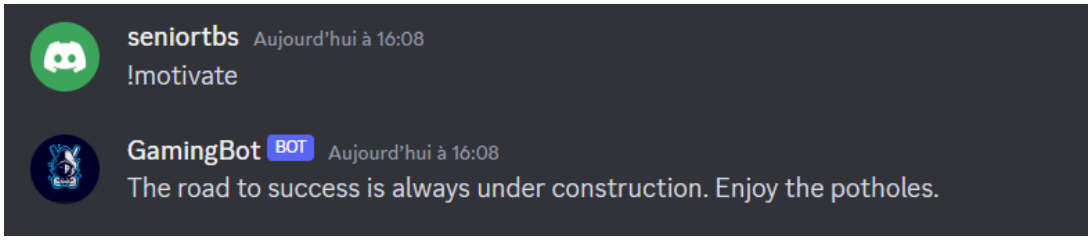
- `!countdown`: Initiates a countdown, engaging users with anticipation before declaring "GO!"



- `!random`: Generates a random number within a specified range, adding an element of unpredictability.



- `!motivate`: Provides sarcastic motivational quotes, injecting humor into traditional motivational messaging.



6.0.4 Error Handling and Logging

The Discord bot code includes basic error handling mechanisms to ensure a smooth user experience. Additionally, logs can be implemented to track user interactions and identify potential issues for future improvements.

6.0.5 Future Enhancements

Potential enhancements for the Discord bot include:

- **Dynamic Responses:** Implement responses that adapt based on user context or server-specific information.
- **Logging and Analytics:** Enhance error tracking and user interaction logs for comprehensive analysis.
- **User Session Management:** Allow users to join and leave Discord sessions, providing a more dynamic and inclusive environment for collaborative activities.

Chapter 7

Security

In our application's security architecture, we prioritize creating a clear separation between gamer and admin roles. While both users and admins use the same login path, a crucial distinction is made during account creation. A user can be designated as either a *gamer* or an *admin*, but not both. However, during the login process, the authentication system dynamically issues distinct tokens. These tokens grant access to endpoints tailored for *gamers* only, *admins* only, or both, depending on the assigned role.

To implement this, we utilize the **OAuth2PasswordBearer** authentication scheme and configure the `tokenUrl` as '*login.*' This ensures that the application issues role-specific tokens. With this approach, only authenticated users or *admins* possessing the appropriate token can access designated routes. By controlling access to specific functionalities based on the user's assigned role, we enhance security in our application.

Pydantic models:
In this gaming project, Pydantic models play a crucial role in maintaining data accuracy and integrity. The `GameSessionBase` model captures essential details for gaming sessions, including the title, description, scheduled time, and maximum attendees. Models like `GamerOut` and `GameSessionOut` encapsulate user and session information, while `GamerCreate` and `GamerLogin` handle user registration and login data. Tokens are represented by the `Token` and `TokenData` models, ensuring secure authentication. Additionally, models like `Like` and `AttendeeOut` manage likes and attendance.

These **Pydantic models** define the structure and validation rules for the data, ensuring reliable and consistent handling throughout the gaming application. By utilizing these models, we can enforce data integrity, validate inputs, and provide a standardized way of working with data across different parts of the application.

Chapter 8

Conclusion

In conclusion, FastAPI-based gaming project provides a platform for users to create, manage, and participate in gaming sessions. Using SQLAlchemy for database management and Pydantic models for data validation, the application ensures secure user registration and login. The RESTful architecture enhances communication, offering a seamless experience for gamers. The project focuses on data integrity and a user-friendly interface, enhancing the gaming community's engagement.

8.0.1 Possible Additional Implementations and Improvements

- **User Profiles and Social Features:** Introduce user profiles with additional information, such as user bio, profile picture, and social features like friend requests and messaging.
- **Gamification Elements:** Introduce gamification elements, such as leaderboards, achievements, or rewards, to make the platform more engaging for users.
- **Recommendation System:** Develop an AI-driven recommendation system that suggests gaming sessions based on a user's preferences, past attendance, and gaming history.