´

### Department of Eletronics, Telecomunications and Computer Engineering

**Practical Work (1st Phase)**

Grup 6:

Pedro Diz, nº47718

Vasco Branco, nº48259

João Pereira, nº48264

Teachers: Paulo Pereira and Pedro Pereira

**Índice**

[Department of Eletronics, Telecomunications and Computer Engineering 1](#_Toc162046136)

[**Introduction** 3](#_Toc162046137)

Physical Model……………………………………………………………………………………………………………………………5

Open-API Specifications…………………………………………………………………………………………………………….5

Details of an API request……………………………………………………………………………………………………….5

Connection Management…………………………………………………………………………………………………….6

Data Access…………………………………………………………………………………………………………………………6

Error Handling…………………………………………………………………………………………………………………6

Critical Evaluation……………………………………………………………………………………………………………………….6

# **Introduction**

The aim of this work is to implement na information system that manages video game sessions.

The application domain consists of these entities:

* Player: A Player is characterised by having a unique number, a name and an email address.
* Game: A Game is characterised by having a unique number, a unique name, a developer and set of associated genres.
* Session: A Session is characterised by having a unique number, the number of players involved in the session, the session start date, the game and the associated players.

**Conceptual Model**

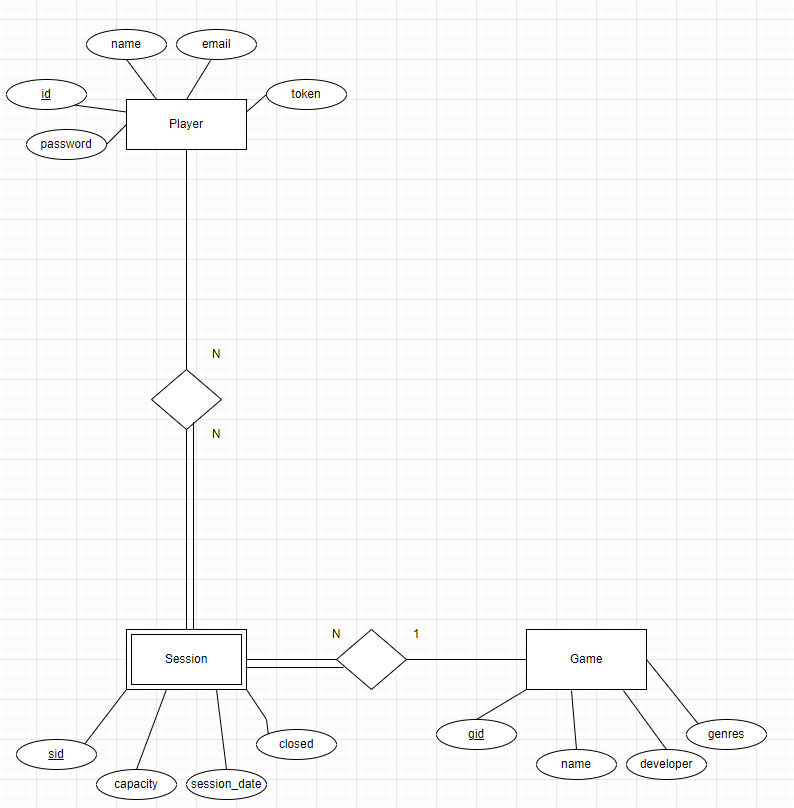
****

Figure 1 – Conceptual Model

The model contains the three entities mentioned. Note that Session is not the same as Player and Game because, if there are no players or game, there is no session.

Integrity Restrictions:

* Email has the following format: [a@a.a](mailto:a@a.a)
* Email and Name are Player candidate keys
* The unique Player token
* Session\_date has the following format: YYYY-MM-DD HH:MM:SS
* Name is Game’s candidate key

**Physical model**

The physical model can be found [here](https://github.com/isel-leic-ls/2324-2-LEIC42D-G06/blob/main/Backend/sql/createTables.sql).

Al the tables are based on the conceptual model.

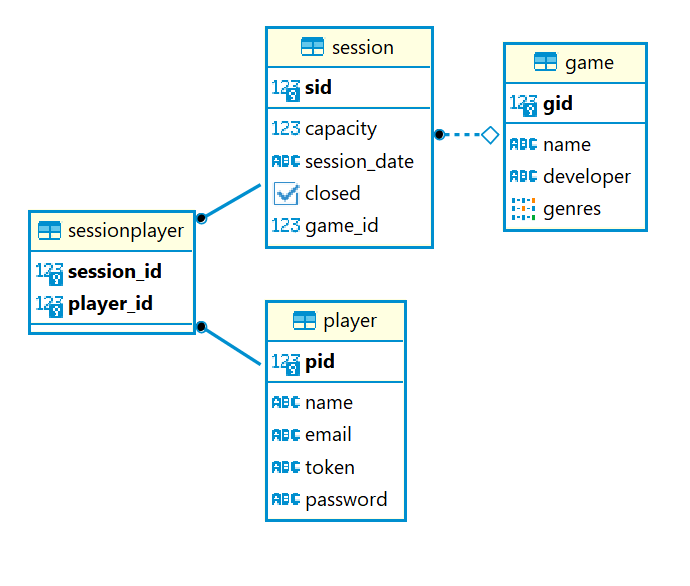


Figure 2 – Physical model

**Open-API specification**

TODO

**Details of an API request**

The request reaches the server, the server then roues the request to the appropriate handler.

After this happens, the following steps are carried out:

* The handler executes a function called errorAwareScope that executes the code needed to fulfil the request. In the event of na error, this function handles the error via its exceptionHandler.
* Within the function mentioned above, the first thing to do is extract the parameters present in the URI. The token is also extracted if a request requires authentication.
* Next, the JSON is deserialised if the request has a body.
* The associated service is called which validates the parameters passed in. Within this service, the repository iss called to persist, read and change data.

**Connection Management**

Whenever we interact with the DBMS, we fetch a connection via the getConnection() function and utilize the use{} function, which automatically closes the connection after running the code in its scope. As we never set autoCommit to false in the repository functions, all the code runs in a single transaction.

**Data Access**

Our data access has been implemented with three interfaces. There is one interface for managing players, another for managing games and another for managing sessions. There are two implementations of these interfaces. One of them is for performing CRUD operations in memory, whose names normally act in accordance with the following structure: Mem{entity} Repo, and the other is for performing CRUD operations on a postgres database, where the names of the repositories follow the structure: Jdbc{Entity}Repo. The JDBC repositories receive as a parameter the postgres datasource with which they would interact.

**Error Handling**

Backend exceptions are generated in all the main modules. For exemple:

* Repository can’t find session with id = x.
* Service fails a validation.
* Api can’t extract the route parameter.

All these exceptions are then handled by the exceptionHandler used in the exceptionAwareScope{} function. Witthin this exceptionHandler, the exception is converted to exceptionAwareScope{} function. Within this exceptionHandler, the exception is converted to a status code using na associative map. A response is then created with a description of the problem.

**Critical evaluation**

All