TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

Web Services

Tunis Business School - Fall 2023

API Tradey Project

Laroui Hamed

Professor
Montassar Ben Messaoud

Abstract

This project is the idea of developing a Restful API of an online trading platform for video games (at least for Version 1). The project has the name of Tradey, developed using different technologies for this Web Services course.

Keywords

API; Tradey API; logged in user

Contents

1 Introduction

1.1 Problematic

Gaming has grown from niches to mainstream for the last number of years now. It is still developing and emerging more day after day to become a huge sector of its own. We are referring to the video games industry; and more people are integrating in the industry for investment and its scalability. However, we address one of the very simple, although common and frustrating issues in this field. Many players after buying their game, enjoying its experience until achieving the very ultimate objective set by the game's developers, are stuck with a completed useless game franchise. If they want to benefit from it, they need to look for a possible buyer of that specific game, post online, ask friends... but in the end, most of the gamers just end up leaving it with them.

1.2 Solution

This where our Tradey API shows up. What if it was an online platform facilitating much more the process of getting rid of a used game. Not only that, but also acquiring a new unplayed game according to the user preference. In fact, Tradey is an online trading platform for video games used by gamers to post their used games for trade, browse for other posted games, and making contact with the owner of the desired game for a possible game trade between them within a safe and trustworthy trading environment.

1.3 Objective

The objective of this online API is to solve the problematic explained above. Not only helping users reinvest in their used games, but also acquiring a different new game for that used one and it is a win win situation between both parties involved. Moreover, using Tradey helps in one to prevent game wastage and recycle them instead in its own way.

2 Security

2.1 Identification

First of all, and before accessing the Tradey API, the user needs to create an account. Each account has a full name, a unique username, a password, a unique email address and is of a verified email address structure and the location of the user. This process is done through the "/users" endpoint.

2.2 Authentication

After Identification, the user is authenticated. Right after the creation of the account with valid credentials, the API informs the user that an email has been sent to the address provided with a verification code for login. The login is done through the "/login" endpoint where the user has to type his chosen username and password along with the verification code sent by email. When all these attributes are valid, the user is authenticated.

2.3 Authorization

Right after the user sends his valid credentials to login and is authenticated, an access token is generated and granted to the user. An example of an access token:
access_token =`"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmF..."`
Each user has a unique access token.
This access token is a JWT token, signed by our API app, where stored the username and has a lifespan of 3600 seconds. This token permits to the user access to specific protected endpoints of that specific user through the stored username in the JWT.

2.4 JWT Token

A JWT is a signed JSON object with a specific structure, signed by our Flask app's secret key, to prove their origin. The JWT is generated when a user logs in (through "/login" endpoint). In the JWT, is stored the username. The client then stores the JWT and can use it to send protected endpoints (@jwt_required). Since our app generated JWT can be proven through its signature, and the username is received with the JWT in every request, we can be sure that these users are logged in. Moreover, with that access token, the user is authorized to access protected endpoints but only related to that specific user. For example, a user is authorized only to view his account information through the "/users/<username>" endpoint and is not granted to view a different username specified in the path.

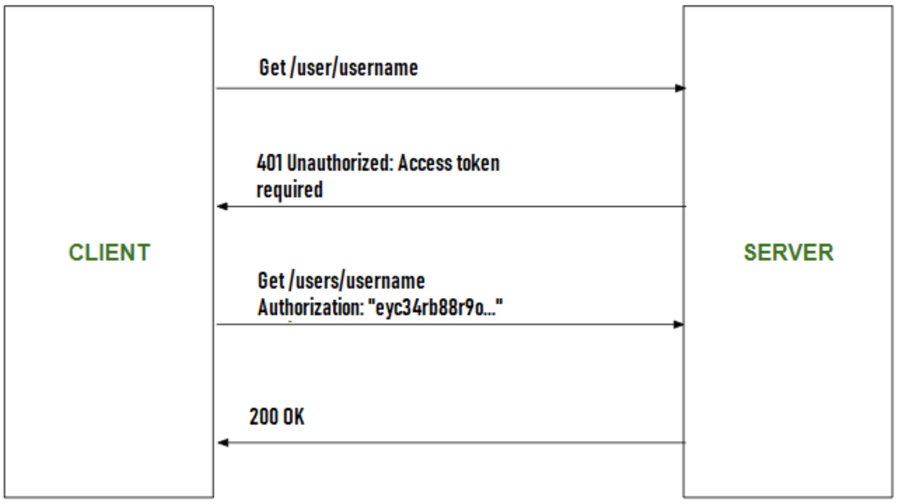

CLIENT
SERVER
Get /user/username
401 Unauthorized: Access token required
Get /users/username
Authorization: "eyc34rb88r9o..."
200 OK

Figure 1: Simple graph of the utility of a JWT Token

3 HTTP Requests

3.1 Operations on users' accounts

3.1.1 POST Methods

"/users" To work on the Tradey API, a user must create an account. This is done through a POST method on the "/users" path and providing the valid attributes: full_name, a unique username (that will be used for all endpoints), a password, a valid email address structure and the user's location (useful to trade games between different users in the real world). If the attributes are all valid, the API returns a 201 HTTP Code with the user credentials and a randomly generated user_id (for security, the password is not present in the response and is stored encrypted), also a default message is shown informing the user to login using the verification code sent by email. If a n anomaly exists, an error message is raised informing the user of it.

"/login" After identification, the user must login using the "/login" path by specifying his username, password and the verification code sent by email. If everything matches, a 200 HTTP Code is returned with an access token (JWT Token) and the concerned username. This access token is used for authorization on protected endpoints.

3.1.2 GET Methods

"/users" This endpoint path returns a 200 HTTP Code with the list of all users present in the system, otherwise returns 404 HTTP Code if there is none.

"/users/<username>" With this endpoint, only the logged in user can view his account's information by providing his username in the HTTP method path. This is done by protecting endpoints using @jwt_required and checking the access token provided by the user. If the access token is valid and generated for that user, a 200 HTTP Code is returned with the user's information, otherwise an error message is returned according to the anomaly (user does not exist, user unauthorized to view another user account, missing access token, expired token...)

3.1.3 DELETE Method: "/users/<username>"

This is a protected endpoint, permitting the logged in user to delete his account if he wants to. A 200 HTTP Code and a message is returned if the process is successful.

3.2 Operations on games

3.2.1 POST Method: "/games"

This protected endpoint asks a user to enter his username, the title of the game to add to his account for trade, the platform, the year of purchase of that game and its condition. A valid and appropriate access token for that user needs to be provided for a successful operation, otherwise the exception error is shown. The assigned user_id and an auto generated game_id are added to the response of the API. This is one example of a response:

```
{
"condition": "like new",
"game_id": "48b3ffa6f4b54778ab9a56774819835b",
"platform": "PS4",
"title": "Mario Kart",
"user_id": "69d6dfbde8db4373b32bc7d692bc5b9b",
"username": "Mohamed",
"year_of_purchase": 2018
}
```

3.2.2 GET Methods

"/games" This endpoint path is similar to the "/users" path. It shows all games listed in the system.

"/games/<username>" A protected endpoint permitting to the logged in authorized user to view his list of games.

"/game/<game_id>" This a public endpoint used by all users to retrieve a specific game to check using its game_id. If the id is undefined, a 404 message is raised.

NB: Note that the path used here is "/game" so it won't make a conflict with the previous endpoint path.

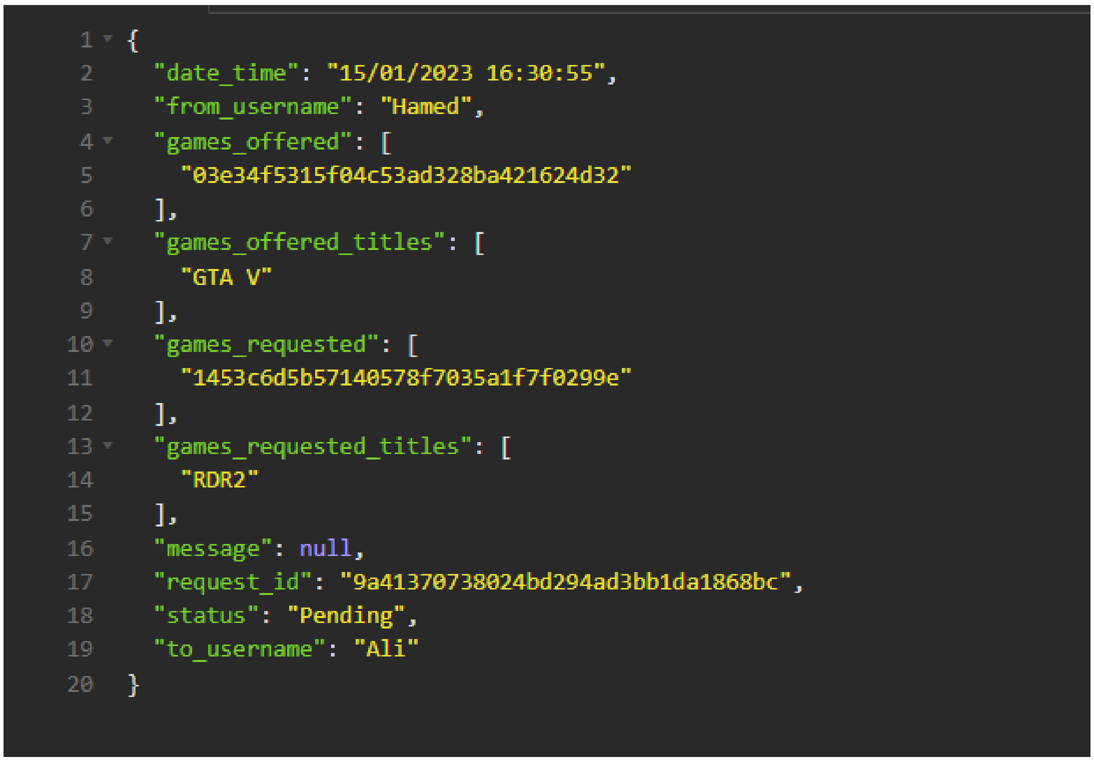
3.2.3 DELETE Methods: "/game/username/game_id"

A protected JWT endpoint, used by a logged in user to delete one of his games.

3.3 Operations on trade requests

3.3.1 POST Method: "/traderequests"

To post a trade request, the user needs to enter his username, the username of the owner of the game he wants to acquire, one or many game_id of his own and of the requested user's. He can attach a custom message to send to the requested user. If all credentials are valid and the user is logged in properly, the API returns a 201 HTTP Code with all input data, generated request_id and the title of the games requested. An email will be sent to the requested user informing him of the notification as well as his trade_requests_notif is updated in his account.



```
{
  "date_time": "15/01/2023 16:30:55",
  "from_username": "Hamed",
  "games_offered": [
    "03e34f5315f04c53ad328ba421624d32"
  ],
  "games_offered_titles": [
    "GTA V"
  ],
  "games_requested": [
    "1453c6d5b57140578f7035a1f7f0299e"
  ],
  "games_requested_titles": [
    "RDR2"
  ],
  "message": null,
  "request_id": "9a41370738024bd294ad3bb1da1868bc",
  "status": "Pending",
  "to_username": "Ali"
}
```

Figure 2: Example of a trade request post response

3.3.2 GET Methods

"/traderequests" Retrieving all traderequests in the system.

"/traderequests/<username>/sent" This protected endpoint returns the list of all trade requests sent by the provided user.

"/traderequests/<username>/received" Returns all trade requests received for this logged in user with the specified username.

"/traderequests/<username>/received/<request_id>" This a helpful protected endpoint to get directly a specific received trade request to examine it and respond to it. The request_id is sent to the requested user by email and can copy it from there directly and examine it. Once opened by the user, its status changes from pending to seen.

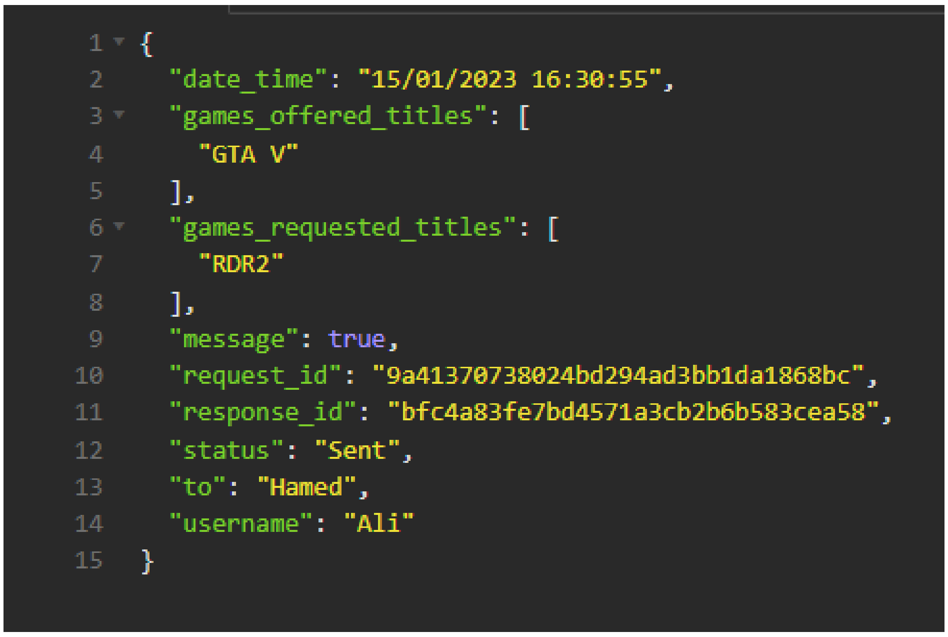
3.3.3 DELETE Method: "/traderequests/username/request_id"

A protected endpoint to delete a specific trade request sent by that logged in user.

3.4 Operations on trade responses

3.4.1 POST Method: "/traderesponses"

This a JWT protected endpoint where a user can respond to a received trade request. He specifies his username, the request_id and his answer: 1 if he accepts the transaction and 0 if not. The response is sent and email will be delivered to the user getting the response. His trade_responses_notif is also updated. This is an example of the API response of a trade response:



```
{
  "date_time": "15/01/2023 16:30:55",
  "games_offered_titles": [
    "GTA V"
  ],
  "games_requested_titles": [
    "RDR2"
  ],
  "message": true,
  "request_id": "9a41370738024bd294ad3bb1da1868bc",
  "response_id": "bfc4a83fe7bd4571a3cb2b6b583cea58",
  "status": "Sent",
  "to": "Hamed",
  "username": "Ali"
}
```

Figure 3: Example of a trade response post response

3.4.2 GET Methods

"/traderesponses" Returns all traderesponses in the system.

"/traderesponses/<username>/sent" A protected endpoint returns all trade responses sent by a user.

"/traderesponses/<username>/received" List of trade responses received of a single user.

"/traderesponses/<username>/received/<response_id>" Returns a single trade response received to a specific user (useful when the id is present in email)

3.5 Operations on ratings

3.5.1 POST Method: "/ratings"

After sending and getting requests and responses back and fourth, users can rate these flows of transaction. This is done through this endpoint by entering one of the users username involved in the transaction, the response_id, a value between 0 and 5 to rate the process and an optional feedback. The rating is stored in the system.

3.5.2 GET Methods: "/ratings"

Gets all ratings in the system.

3.5.3 POST Method: "/ratings/<rating_id>"

This returns a specific rating using rating_id in the path.

4 Project Structure

4.1 UML Diagram

This is the UML Diagram representing the classes used in the project and explained in the HTTP Requests section.

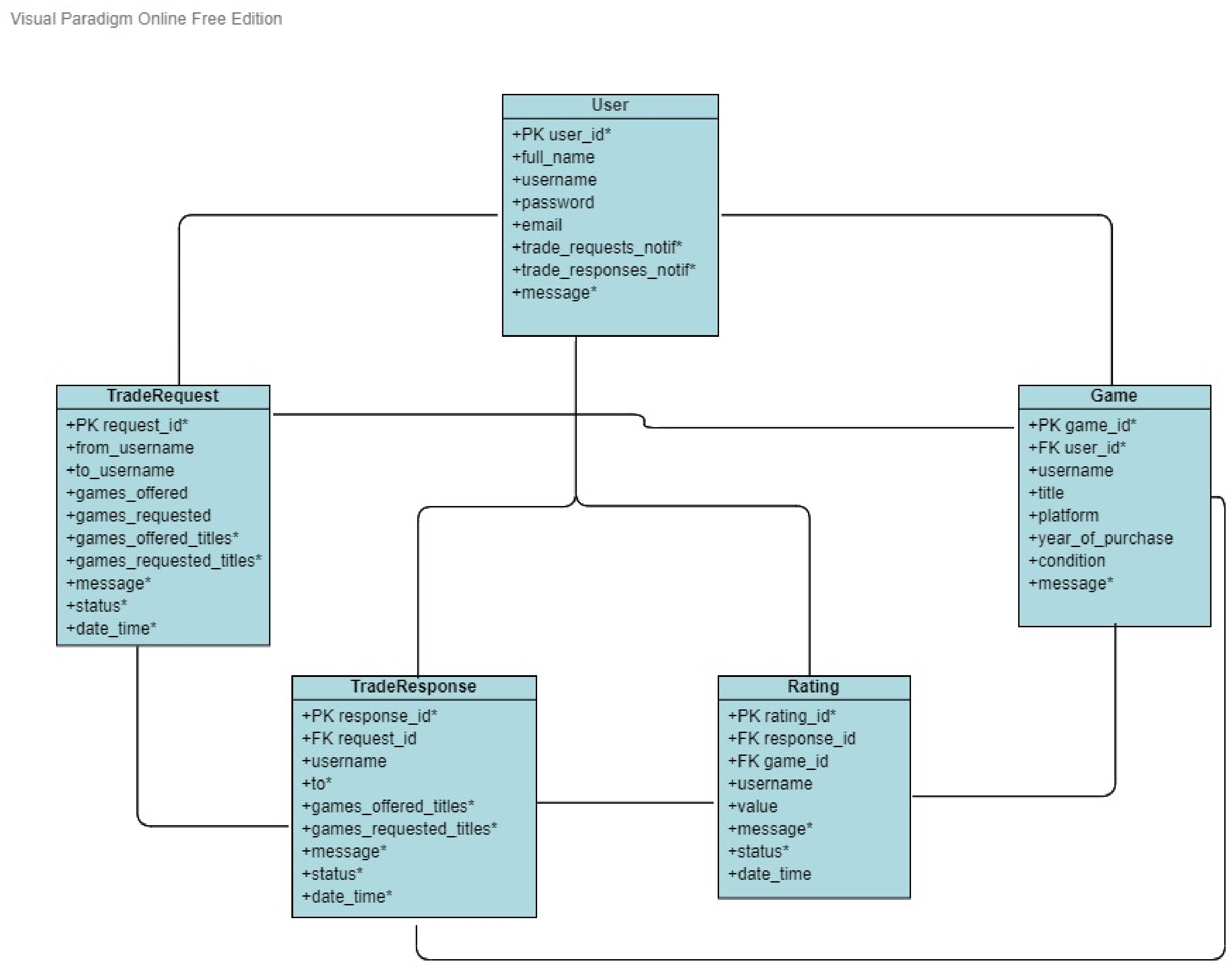
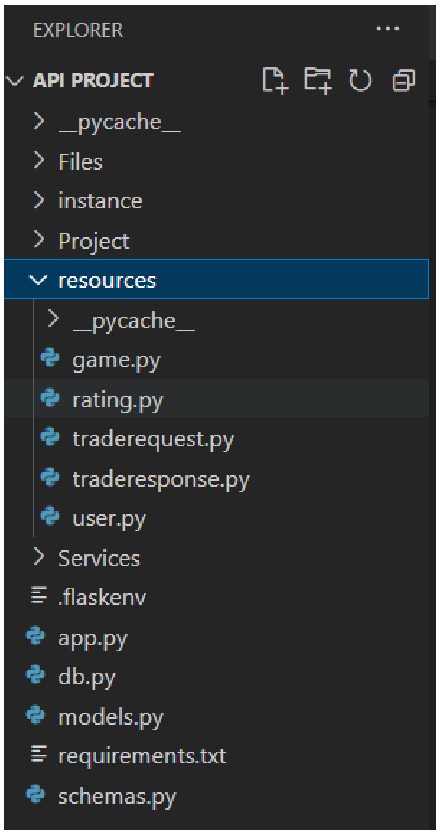


Figure 4: Tradey UML Diagram

4.2 Repositories and Files Structure

This figure represents the structure of the file developed in VS Code virtual environment:



```
EXPLORER
API PROJECT
  > __pycache__
  > Files
  > instance
  > Project
  v resources
    > __pycache__
      game.py
      rating.py
      traderequest.py
      traderesponse.py
      user.py
  > Services
    .flaskenv
    app.py
    db.py
    models.py
    requirements.txt
    schemas.py
```

Figure 5: VS Code Structure

5 Technology, Libraries & Variables

5.1 Technology

Python Python is a high-level, general-purpose programming language. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.[1]

Insomnia Insomnia REST Client is a powerful REST API client used to organize, store, and execute RESTful API requests elegantly. Insomnia is one of the fast REST clients that's available for Windows, Mac, and Linux. Insomnia REST client is a Free Cross-Platform Desktop Framework for testing RESTful applications.[2]

GitHub GitHub is a web-based interface that uses Git, the open source version control software that lets multiple people make separate changes to web pages at the same time. As Carpenter notes, because it allows for real-time collaboration, GitHub encourages teams to work together to build and edit their site content.[3]

5.2 Libraries

flask

flask-smorest

flask_marshmallow

python-dotenv

marshmallow

SQLAlchemy

Flask-SQLAlchemy

passlib

smtplib

pytz

flask-jwt-extended

sqlalchemy-jsonfield

References

[1] "Python (programming language)." [Online]. Available: en.wikipedia.org/wiki/Python_(programming_language)

[2] "Understanding insomnia rest client made easy 101." [Online]. Available: hevodata.com/learn/insomnia-rest-client/

[3] "An introduction to github." [Online]. Available: digital.gov/resources/an-introduction-github/