|  |
| --- |
| HDSSD |
| Requirements Specification (RS) |
| MatchGaming |

|  |
| --- |
| Marta Pacheco Merino  11/10/2022 |

Content

[1 Introduction 2](#_Toc119015755)

[1.1 Purpose 2](#_Toc119015756)

[1.2 Project Scope 2](#_Toc119015757)

[2 User Requirements Definition 3](#_Toc119015758)

[3 Requirements Specification 3](#_Toc119015759)

[3.1 Functional requirements 3](#_Toc119015760)

[3.1.1 Use Case Diagram 3](#_Toc119015761)

[3.1.2 Requirement 1: User takes the questionnaire 4](#_Toc119015762)

[3.1.3 Requirement 2: Test answers are integrated by the system 5](#_Toc119015763)

[3.1.4 Requirement 3: User interacts with the results 6](#_Toc119015764)

[3.1.5 Requirement 4: User interacts with their personal list 7](#_Toc119015765)

[3.2 Non-Functional Requirements 8](#_Toc119015766)

[3.2.1 Performance/Response time requirement 8](#_Toc119015767)

[3.2.2 Availability requirement 8](#_Toc119015768)

[3.2.3 Robustness requirement 8](#_Toc119015769)

[3.2.4 Security requirement 9](#_Toc119015770)

[3.2.5 Maintainability requirement 9](#_Toc119015771)

[3.2.6 Portability requirement 9](#_Toc119015772)

[3.2.7 Extendibility requirement 9](#_Toc119015773)

[4 Interface requirements 9](#_Toc119015774)

[4.1 GUI 9](#_Toc119015775)

[4.2 Application Programming Interfaces (API) 12](#_Toc119015776)

[5 System Architecture 13](#_Toc119015777)

[6 System Evolution 13](#_Toc119015778)

# Introduction

## Purpose

The purpose of this document is to set out the requirements for the development of an application called MatchGaming, whose aim is going to be to match users with videogames matching their personalities and personal tastes.

The intended customers involve mostly everyone with access to a technology with internet, and a minimum willingness to spend some of their free time playing a videogame.

Due to the kind of questions that are going to be made, maybe the minimum age to start using the app in an optimal way would be around 10 years old, but it would be a good idea to plan a future update to include more “childish” questions, so maybe kids who feel a bit weird or out of place can find something they like in the videogaming industry.

A limit for older people would be in case they have visual or hearing limitations (gaps that should also be solved in future updates), but if they have a technology available, the app will be usable for them as well.

## Project Scope

The scope of the project is to develop a responsive and dynamic web application using raw JavaScript as its main technology. It will retrieve information from the user, and generate a list of games that will be shown to the user, along with their imagery, description, tags, platform and other relevant information to see if the match between the answers provided by the user and the results is accurate.

The system shall have a strong call to an external API, to establish the database we are going to play with. Also, a good system of query parameters to be able to link the keywords that will be created as a result of the user answering to the questions provided by the system with this established database. To create said keywords, there won’t be a checkbox or something that gives the system the keywords directly; there will be a list of questions made to the user, that will afterwards link these answers to topic-related keywords.

As this will be done in the range of a month and a half, the preliminary version of the application will be quite basic, but counting with a list of further changes that will hopefully be applied after the delivery of the basic product. There won’t be associated costs if the application is just hosted in GitHub as a project; in the scenario of publishing it in a hosting, we could speak about an expense of ~€100 a year.

# User Requirements Definition

The user shall open a list of questions in a quiz style, and answer the questions in order of some data to be retrieved from them.

The user shall check the results of the database’s retrieval after answering the questions.

The user shall be able to filter what of the games shown on screen they want to keep in their personal list or not.

# Requirements Specification

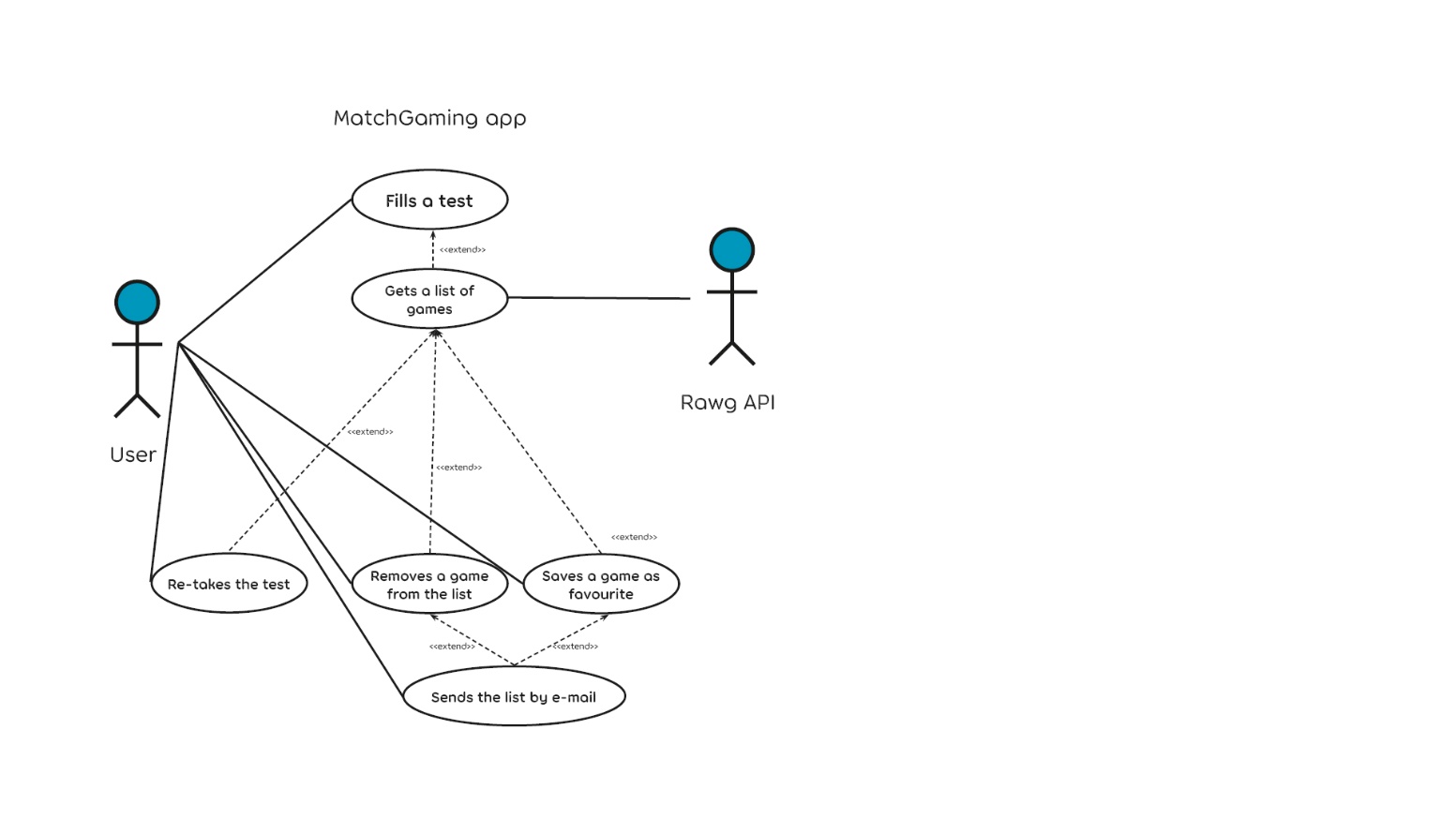
There is not a specific technical requirement for the user, as it’s going to be very simple and intuitive. No training will be needed to use it, and whoever who can read will be able to use it immediately, with no tech knowledge. If there is anything that might be a little bit confuse, a bubble explaining the feature will show up on the first time the user opens the app.

## Functional requirements

### Use Case Diagram

Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.

The Use Case Diagram provides an overview of all functional requirements.



### Requirement 1: User takes the questionnaire

#### Description & Priority

#### The user has to answer some questions in order to interact with the application, because that information is the base of what the app is going to show.

#### Use Case

01\_InteractionWithTest

**Scope**

The scope of this use case is to prove the need of the user to take the test in order to get the keywords needed by the system.

**Description**

As this application takes a kind of "personality test" to the user to retrieve keywords and fetch a list of games matching their answers, the test has to be done.

**Flow Description**

**Precondition**

The app has just been launched.

**Activation**

The use case starts when an <Actor> opens the app and starts checking on answers to the test's questions. The option to submit the test is greyed out.

**Main flow**

1. The <Actor> checks an answer to each question (Check A1).
2. The system retrieves the keywords that are attached in the back-end to each of the answers the <Actor> has selected, and stores them in an array of strings.
3. The <Actor> finishes checking answers to questions.
4. The button to submit the test is not greyed out anymore.

**Alternate flow**

A1: <Questions not answered>

1. The system won’t react, the button to submit the answers won’t change.
2. The <Actor> won’t be able to interact with the system anymore.

**Termination**

The system presents the next screen, which will use multiple query parameters to filter inside of the immense database we are using to sort a list of games matching these filters.

**Post condition**

These filtered results will be shown on screen to the user, so they can check which of them they like or not.

### Requirement 2: Test answers are integrated by the system

#### Description & Priority

Here is when the magic starts for the user: each of the questions’ answers are attached to different keywords that will be stored in an array, and placed as query parameters in the search results that will be seen right after the user presses on the submit button.

#### Use Case

02\_ListOfResults.

**Scope**

The scope of this use case is to create a solid string of query parameters based on the information retrieved by the user.

**Description**

This use case describes how the answers of the users will be linked to the system via software to generate a strong list of results.

**Flow Description**

**Precondition**

User has answered to some questions.

**Activation**

This use case starts when an <Actor> presses the “Submit” button at the end of the test.

**Main flow**

1. The system grabs the array of keywords
2. The system generates a link with query parameters filled with these keywords, making filters that will include console, age range, genres, tags, etc.
3. The <Actor> will then see a list of this filtered search (See E1).

**Exceptional flow**

E1: No results available

* 1. The system will show a “no results available” screen if the search, after all the filters, shows a total of 0 results. This is very unlikely, but might happen.

**Termination**

The system flow finishes on step 7 of the main flow, with the list of the filtered search.

**Post condition**

The system goes into a wait state until the user interacts again with the list provided.

### Requirement 3: User interacts with the results

#### Description & Priority

Now, the user has the list of results and can interact with them. There are two main choices, to like or to dislike them.

#### Use Case

03\_InteractWithResults.

**Scope**

The scope of this use case is to generate an array of results liked by the user after applying the filters and store them in case they want to send them by e-mail.

**Description**

This use case describes how the user will be able to check and save the games they want to keep in mind.

**Flow Description**

**Precondition**

User has submitted the test and is checking the results.

**Activation**

This use case starts when an <Actor> presses the “Like” or “Dislike” button in their list of results.

**Main flow**

1. The <Actor> clicks on the “Like” button after checking a game (see A1) (see A2)
2. The system adds the information of this game into the new array generated for this session.
3. The <Actor> will keep doing this filtering until they want to stop or they run out of results.

**Alternate flow**

A1: <Actor> clicks on “Dislike”

1. This list item will disappear from the screen.
2. The <Actor> won’t be able to interact with that game anymore.
3. We go back to step 10

A2: <Actor> resets the application

1. <Actor> might decide to click into the “Retry Test” button at any time.
2. The <Actor> will come back to the beginning of the test and all the data retrieved until then will be lost.
3. Flow would terminate here.

**Termination**

It’s up to the system or the user when to terminate with this flow.

**Post condition**

The system has a fulfilled array of content.

### Requirement 4: User interacts with their personal list

#### Description & Priority

Now, the user has the option of sending this list of filtered games to himself or someone else by e-mail.

#### Use Case

04\_SendMyList.

**Scope**

The scope of this use case is to grab this array in a clean and ordered way, and be able to encapsulate it into an e-mail that will be sent to the e-mail address provided by the user.

**Description**

This use case describes how the user will be able to save their own list and deliver it by e-mail.

**Flow Description**

**Precondition**

User has interacted with their list of matched games and liked at least one of them.

**Activation**

This use case starts when an <Actor> presses the “Send by e-mail” button in the bottom right corner of their screen of results.

**Main flow**

1. The <Actor> clicks on the “Send by e-mail” button (see E1).
2. The system shows a pop-up with a textbox to type an e-mail address.
3. The <Actor> types an e-mail address in the textbox.
4. The <Actor> clicks on “Submit”.
5. The system puts all the content inside of the array in an e-mail.
6. The e-mail is sent to the e-mail address typed by the <Actor>

**Exceptional flow**

E1: User has not liked any game

1. The pop-up won’t show. Instead, the <Actor> will see a bubble telling them that they haven’t liked any game yet and the email can’t be sent.

**Termination**

The system packs the content of said array and sends it to the e-mail provided by the user.

**Post condition**

The e-mail address provided by the user will receive a message with a list of the games the user has liked after using the application.

## Non-Functional Requirements

Specifies any other particular non-functional attributes required by the system. Examples are provided below. **Remove the requirement headings that are not appropriate to your project.**

### Performance/Response time requirement

The connection to the API should be clean and fast, and not take more than one second to load when opening the application.

### Availability requirement

In order to keep the users trust when publishing the app, it should be available to users more than 98% of the time during day and night (due to possible international usage). Because of this, updates will be applied altogether once a month (unless they are critical updates).

### Robustness requirement

The only actual requirements in terms of robustness are having a strong hosting that can hold a huge number of simultaneous users. The API has to be up in order of the app to work, but after investigating about it, looks like a safe resource so I’m not very worried about it.

### Security requirement

As data won’t be stored by the moment, this is not a requirement that I need to have in mind in this moment.

### Maintainability requirement

Maintenance and updates will be made together a minimum amount of once a month.

### Portability requirement

This will be a cross-device and cross-browser web application that will be runnable in computers, tablets and smartphones.

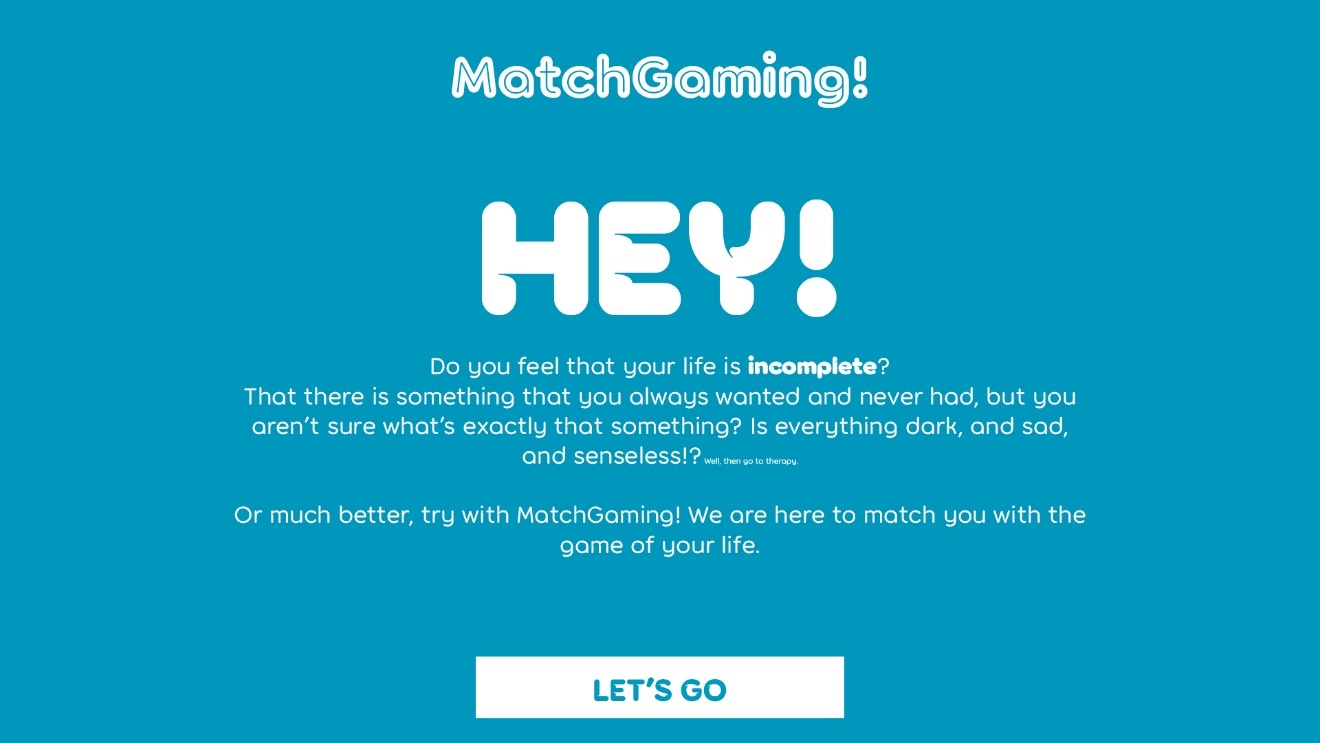
### Extendibility requirement

I have a lot of ideas about extendibility and would love to bring them to life, they are specified in the last point of this report.

# Interface requirements

## GUI

This is the first view of the application, although the text is provisional.



The user has only one choice, that is clicking into “Let’s go”. This will lead them to the following screen:



Here is the page where the user will answer some questions (along the lines of 20) that will make the system create an array of keywords. The user will click on the arrows to the left to the right to navigate through the questions. The option of submitting the test is greyed out until the user fills all the answers.



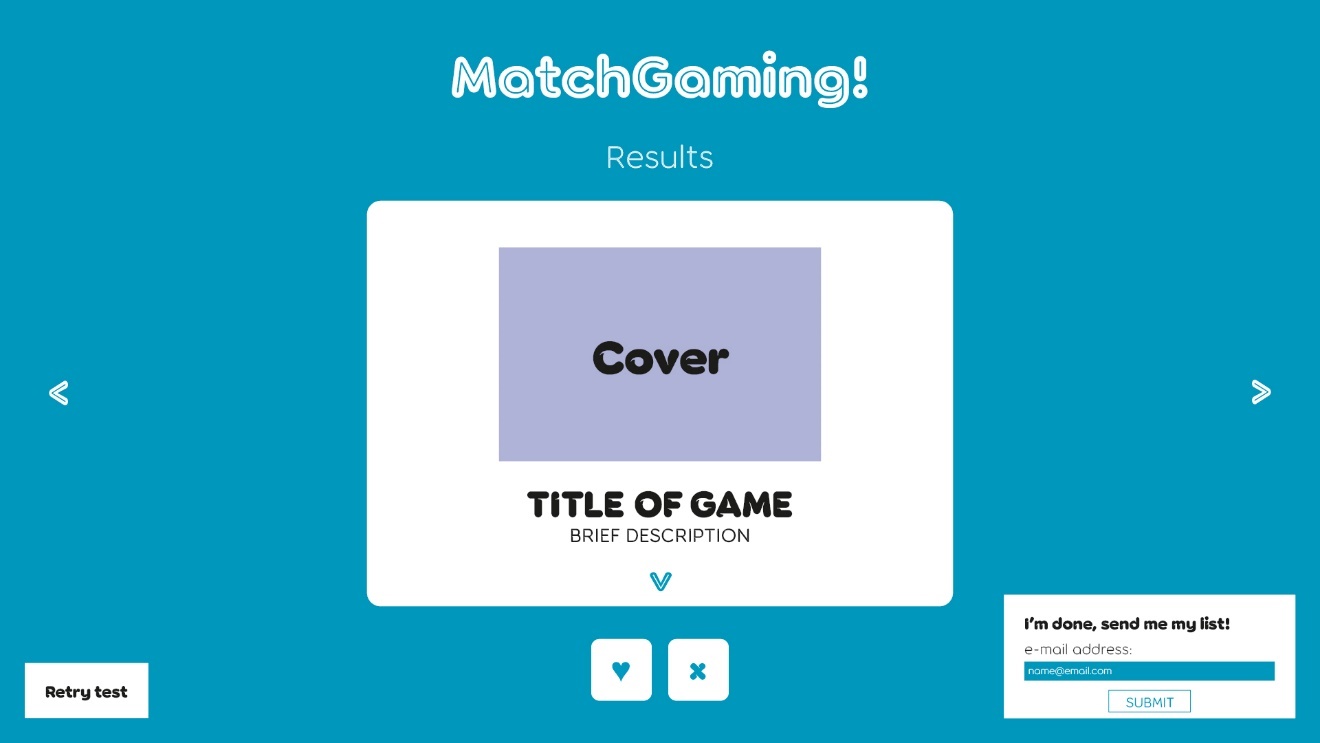
Now, the user should click into the Submit button as they have finished answering the test's questions.



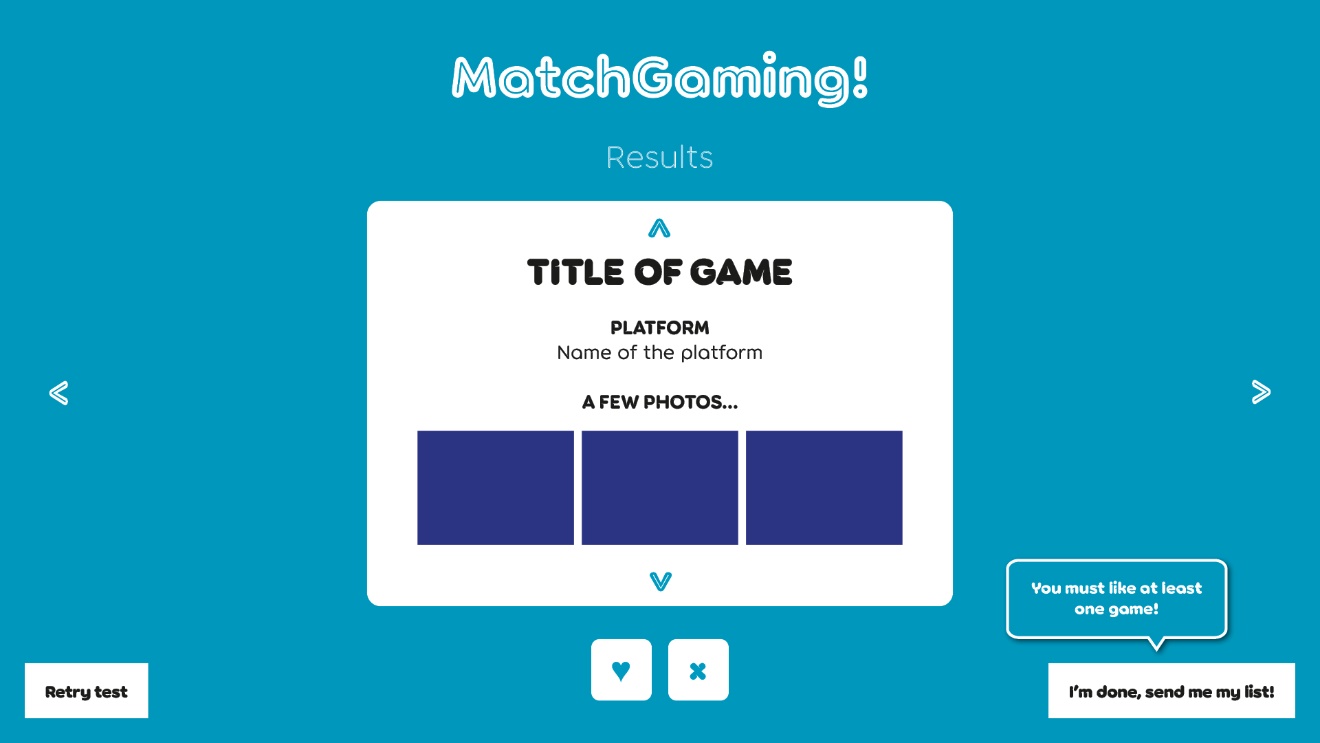
After the questionnaire is finished, this is the screen they are going to see. The interface will show just one game at a time, with a main view of the title, brief description and cover. The blue bubble we can see next to the arrow below the description will just show the first time, so the user knows how to scroll on the information. The next screenshoot shows the view after clicking on said arrow.



The side arrows will let the user navigate through the games. The heart button will include this element in an array made specifically for this session of the user. The cross will remove the item from the list. Clicking into the button on the bottom left will reset the page. Clicking into the button on the bottom right will open a little pop-up (next screenshot)



Here, the user will be able to deliver the list of selected games to their own e-mail address, or any other they want. In case the user hasn’t liked any game yet, the pop-up won’t show up. Instead, we will see the following bubble:

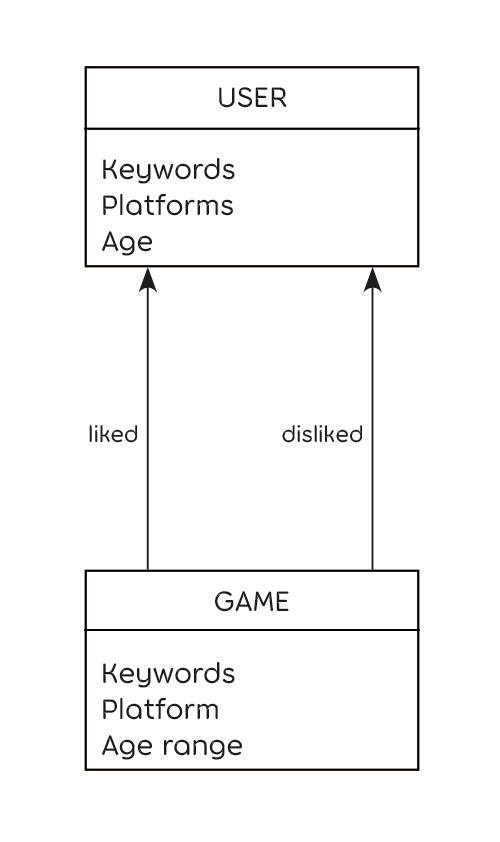


## Application Programming Interfaces (API)

My application is connected to Rawg.io, an API that hosts one of the world’s biggest databases in terms of videogames. It has all the features and requirements I looked for when planning the development of the application: it’s quite easy to use and call, it contains videogames of all kind of platforms (even phones, which approaches the app to elder people who won’t have a console or a computer but will do have a smartphone), it has a good system of filtering by platforms, keywords, developers, etc.

# System Architecture

The only elements I can add so far to a class diagram are the following:



This is due to the simplicity of the app in terms of how the elements interact with each other, but I hope to enlarge it with a user management system in a future.

# System Evolution

I am still planning how to develop the basic version of the app, but I already know some of the features I would like to add to it in a nearby future. Some of them are listed here:

* To give the user the possibility of removing some of the tags generated by their answers, in case they don't feel identified with them.
* Accessibility features like voice recognition to answer the questions.
* Account creation to save the lists (until then, we will give them the option to type an email address and we can send them the content of each attempt, and the session will be reset on page refresh). This, in fact, will be hopefully made with a user management system.
* Changing questions each time they open the app