MatchGaming

Product Design Specification

Version <1.0>

11/11/2022

Marta Pacheco Merino

VERSION HISTORY

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Reason** |
| 1.0 | Marta Pacheco Merino | 11/11/2022 | <name> | <mm/dd/yy> | Initial Design Definition draft |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**UP Template Version:** 12/31/07

TABLE OF CONTENTS

[1 Introduction 4](#_Toc119100083)

[1.1 Purpose of The Product Design Specification Document 4](#_Toc119100084)

[2 General Overview and Design Guidelines/Approach 4](#_Toc119100085)

[2.1 Assumptions / Constraints / Standards 4](#_Toc119100086)

[3 Architecture Design 4](#_Toc119100087)

[3.1 Logical View 5](#_Toc119100088)

[3.2 Hardware Architecture 6](#_Toc119100089)

[3.3 Software Architecture 6](#_Toc119100090)

[3.4 Security Architecture 6](#_Toc119100091)

[3.5 Communication Architecture 6](#_Toc119100092)

[3.6 Performance 6](#_Toc119100093)

[4 System Design 7](#_Toc119100094)

[4.1 Use-Cases 7](#_Toc119100095)

[4.2 Database Design 9](#_Toc119100097)

[4.3 Data Conversions 9](#_Toc119100098)

[4.4 Application Program Interfaces 9](#_Toc119100099)

[4.5 User Interface Design 10](#_Toc119100100)

[4.6 Performance 13](#_Toc119100101)

[4.7 Section 508 Compliance 13](#_Toc119100102)

# Introduction

## Purpose of The Product Design Specification Document

The Product Design Specification document is going to provide an overview of how the application *MatchGaming* is being structured at the current stage of the project’s development. We will, then, have a wide view of the architecture and the system of the application itself, and each of its components, put together in a document we can check and review to remember the guidelines the team has to follow.

It will also show a good overview of the graphic design of the app, in order to show to stakeholders and clients, should they have something to say about it. It could also be used as a presentation of the product if we finally develop it ourselves and want someone to invest into it in the future.

# General Overview and Design Guidelines/Approach

## Assumptions / Constraints / Standards

We have to keep in mind a few assumptions in order of the app to work fully, including on these:

* The external API our app is connected to (Rawg.io) is up and working properly. This will be critical, otherwise the app won’t be able to run results to the user after taking the initial test.
* A stable internet connection is a must by the user. It doesn’t need especially strong or fast, as the app is simple and the API has not extreme requirements.
* The application is being constructed using Windows 10, along with Visual Studio Code and the live server view in Google Chrome. It’s being constructed following the Mobile-First Design strategy. As the name suggests, it’s a strategy in which web designers first create products for mobile platforms. This can be achieved by first designing the web app on the smallest screen size possible, then progressively expanding it to greater screen sizes.
* During the whole process, tests will be run in Mozilla Firefox, Internet Explorer, Opera and Safari. Luckily, we have access to Mac, iPhone, Android phones, Android Tablets and an iPad developer simulator in Mac, so most of the systems are covered in our design testing.

# Architecture Design

Being connected to an external API, the interaction is permanent and stable. Information is sent and received all the time via query parameters, but never stored after the end of the user’s session. However, the perfect scenario would involve creating a user management system in which the data can be stored safely: that will definitely be a future update.

## Logical View

In this logical view we can see that the user will be able to access to the application through any devices and will have three main possibilities that will follow each other. The first one is filling a test whose questions will be generated by JavaScript. They will have connected some keywords that will be fetched by an array when the user checks each answer in the total of questions. All that process will be run locally, without the need of databases.

Now, it’s the turn of the API, that will receive a call through a link full of query parameters. These parameters will be fetched by the array generated in the previous step, and will return some results, that will be seen by the user on the screen. Now, the user will be able to interact with these results liking or disliking them. Liked items will once again be stored in a local array.

Once it’s done, they can send the list to a desired email address. Most of the process will be over then.

## Hardware Architecture

The application will firstly be hosted in GitHub, but a hosting will be purchased to test the application on it. To get to that, research about best functional servers globally has been made, throwing as a result that the best option in 2022 is MochaHost, setting a good price-quality relationship. They offer unlimited disk space and traffic, and extras like a guarantee of 100% uptime and a free SSL certificate[[1]](#footnote-1).  
No databases or Cloud Application Platforms will be needed by now (never saying never, as it’s likely that the app will keep growing up after the first delivery).

## Software Architecture

The magic of this assignment for me and what makes it the biggest challenge is the fact that it’s being fully developed with the only usage of Vanilla JavaScript, avoiding frameworks and libraries. Only PHP will be used if needed, and the other technologies to use are HTML and CSS. This is going to make things very tedious, and even knowing that it’s not the most optimal way to develop an application (because libraries and frameworks are made to simplify workflows), this is taken as a challenge to establish a solid base of knowledge, and start building from there.

## Security Architecture

It doesn’t apply to the project right now, as no data is going to be shared or stored on any kind of server.

## Communication Architecture

The communication from the app to the external API’s server will be done through Ajax (Asynchronous JavaScript and XML). With Ajax, web application developers can send data from a web server and retrieve it asynchronously (in the background). By decoupling the request/response cycle from the display and behavior of an existing page, Ajax allows web pages and by extension web applications to dynamically change content without having to refresh the entire page. Data may be retrieved using the XMLHttpRequest object. Despite the name “Ajax”, XML is not required (JSON would be common) and requests do not have necessarily to be asynchronous.

Through the Fetch API, we will retrieve the full database the website is based on, and play with the query parameters to generate different calls and get the information we require each time the app is running.

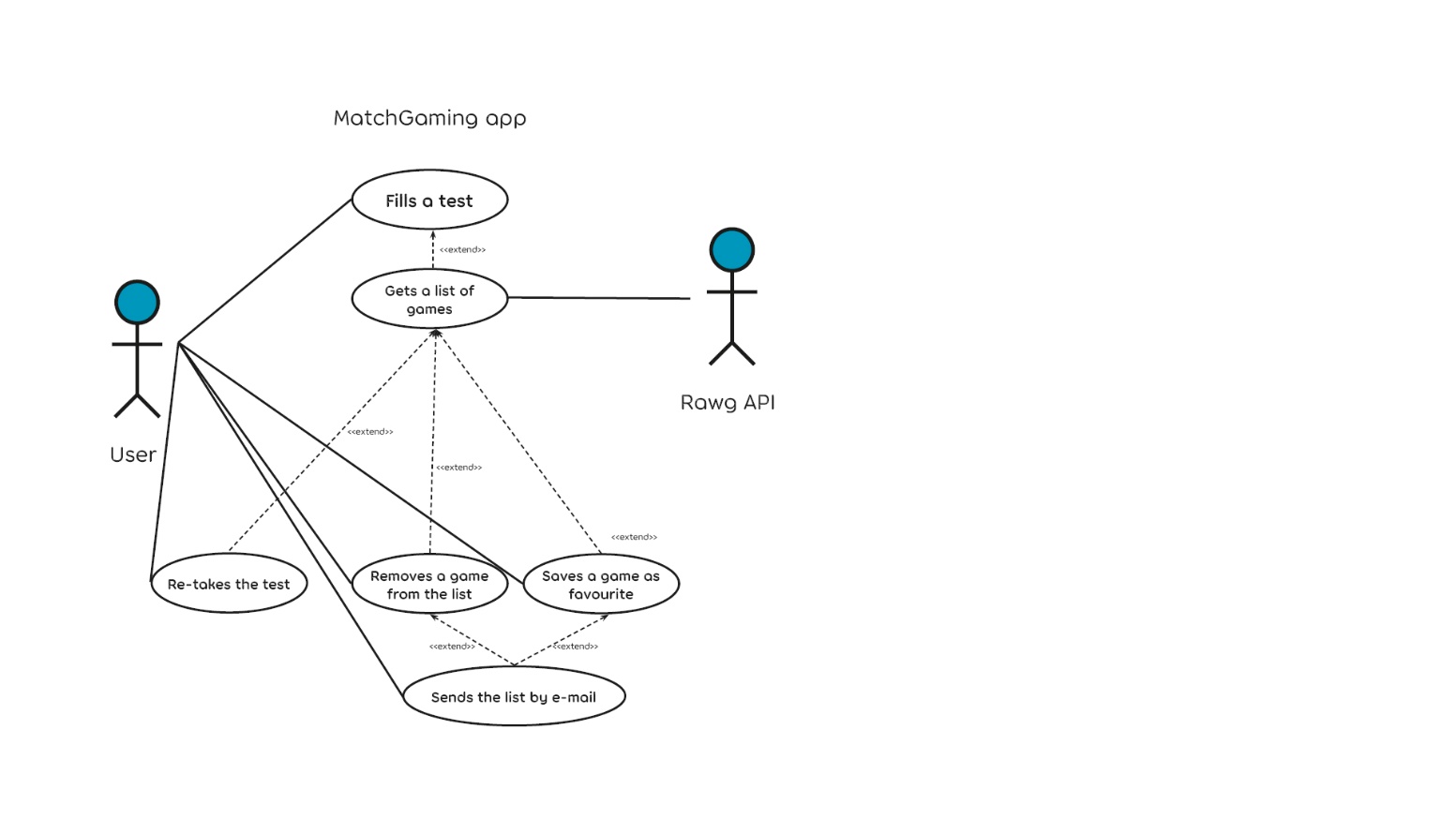
## Performance

The fact of this app just using Vanilla JavaScript has a lot of disadvantages in terms of complexity of the development process, but gives a really good point in how the performance is going to be: fast and clean. The usage of libraries always lowers the speed of the page’s load, even if it’s just a few milliseconds, that might be a big difference in a part of the world where bandwidths are still under the average.

# System Design

## Use-Cases

This use case diagram shows an overview of all the functional requirements that the app is going to have.



#### Filling the test: 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.

* 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. 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.
* The app has just been launched. 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. Then, the following flow happens:
  + The <Actor> checks an answer to each question.
  + 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.
  + The <Actor> finishes checking answers to questions.
  + The button to submit the test is not greyed out anymore.
* 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.
* These filtered results will be shown on screen to the user, so they can check which of them they like or not.

#### Showing the search results (API takes action here): 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.

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

#### This use case starts when an <Actor> presses the “Submit” button at the end of the test. Then, the following flow happens:

* + The system grabs the array of keywords
  + The system generates a link with query parameters filled with these keywords, making filters that will include console, age range, genres, tags, etc.
  + The <Actor> will then see a list of this filtered search.
* The system will then go into a wait state until the user interacts again with the list provided.

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

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

#### This use case starts when the user has submitted the test and the results are already on the screen. Then, the following flow happens:

* + The <Actor> clicks on the “Like” or “Dislike” button after checking a game.
  + On “Like” click, the system adds the information of this game into the new array generated for this session. On “Dislike” click, the game will disappear from the screen.
  + The <Actor> will keep doing this filtering until they want to stop or they run out of results.

#### There is an alternate action here, that would be the user deciding to reset the application, in which case the data would be lost and the app would start over new.

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

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

#### This use case starts when the user presses the “Send by e-mail” button in the bottom right corner of their screen of results. Then, the following flow happens:

* + The <Actor> clicks on the “Send by e-mail” button.
  + The system shows a pop-up with a textbox to type an e-mail address.
  + The <Actor> types an e-mail address in the textbox.
  + The <Actor> clicks on “Submit”.
  + The system puts all the content inside of the array in an e-mail.
  + The e-mail is sent to the e-mail address typed by the <Actor>

#### The system will then pack the content of said array and sends it to the e-mail provided by the user, and 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.

## Database Design

We are connected to Rawg.io’s API to make use of their videogaming database, and are not using any personal database at this moment or in the near future.

## Data Conversions

We will be receiving the information from the database as a JSON list, and we will extract the desired content and place in the front-end in a clean way using JavaScript for it, with commands like innerHTML, and giving format with CSS.

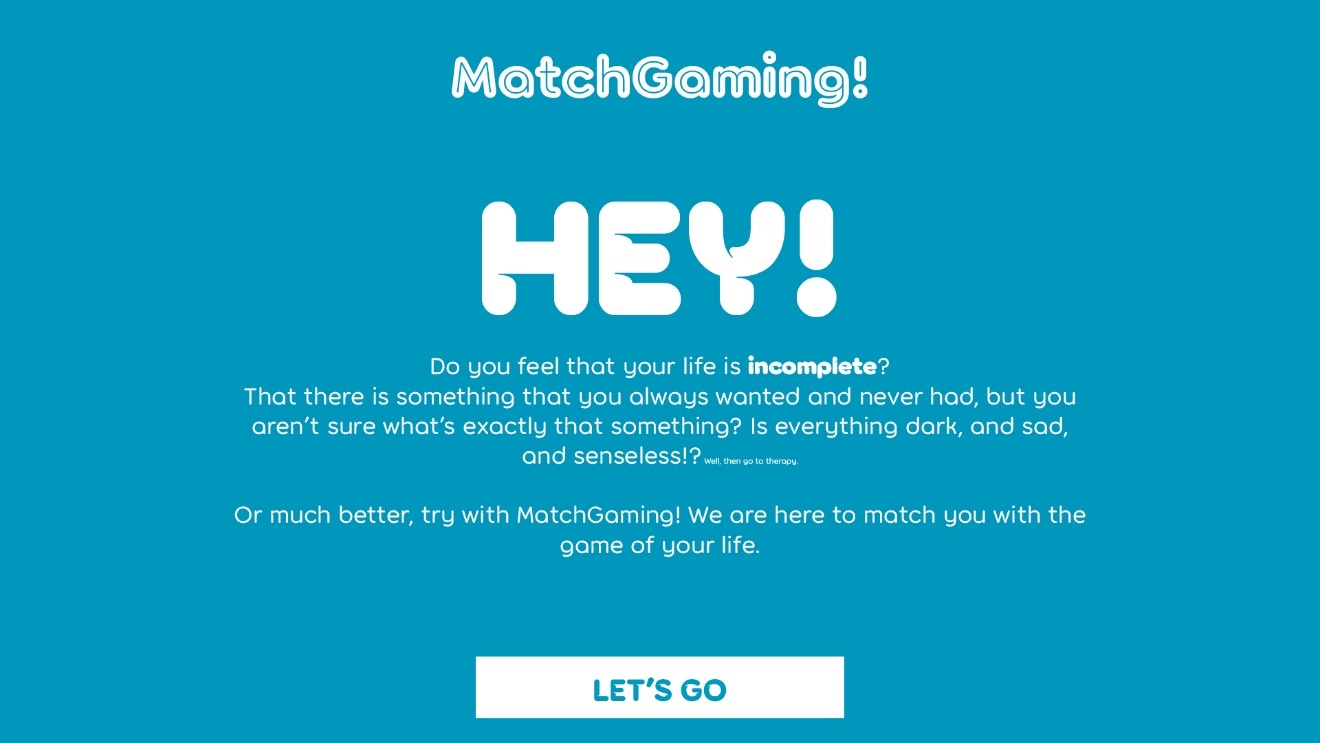
## Application Program Interfaces

This 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 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.

## User Interface Design

## 

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.



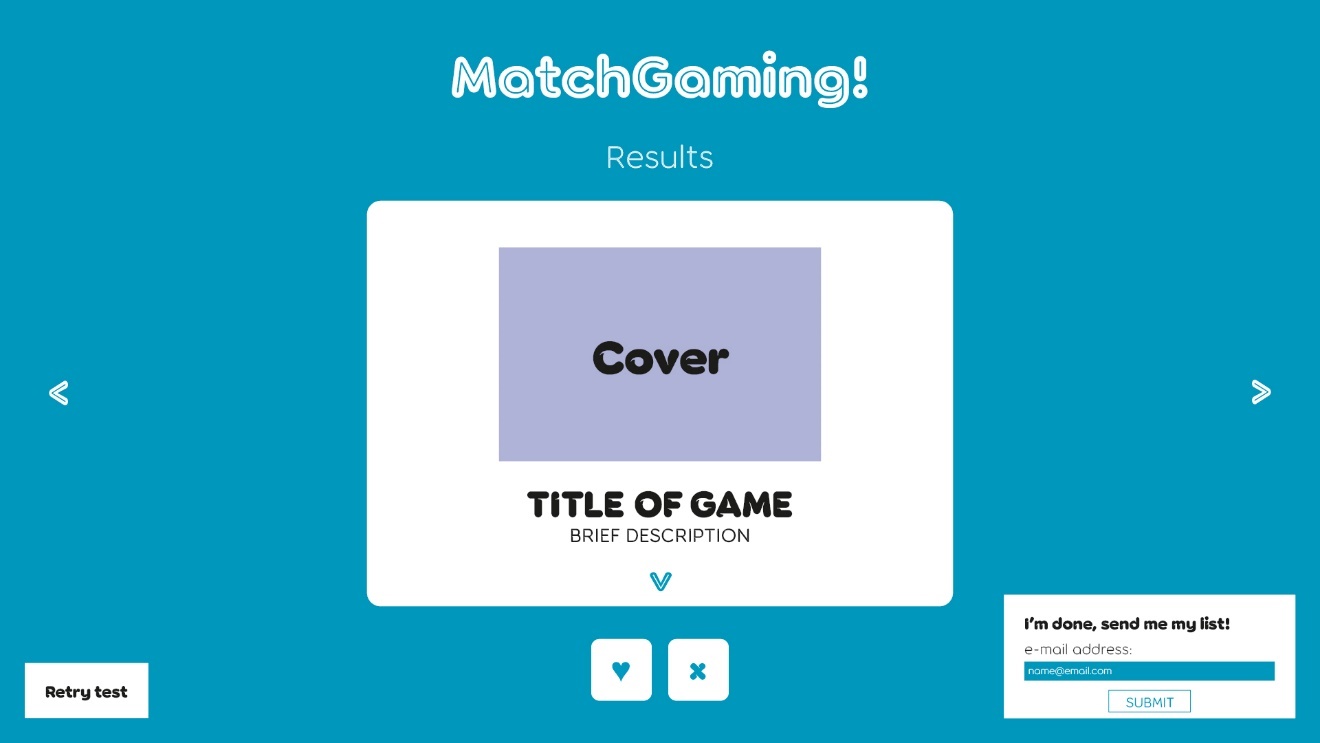
Now, the user should click into the Submit button as they have finished answering the test's questions.



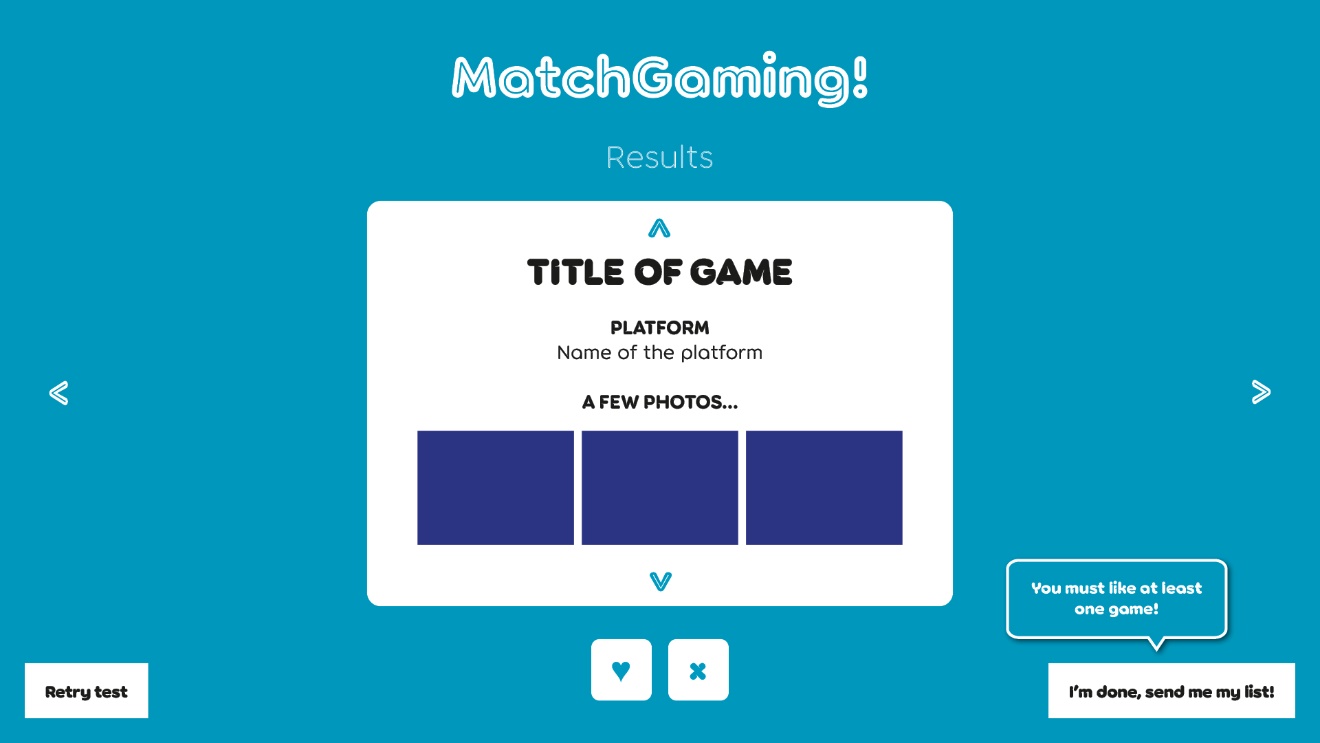
After the questionnaire is finished, this is the screen they are going to see. The interface will show just one game at a time, with a main view of the title, brief description and cover. The blue bubble we can see next to the arrow below the description will just show the first time, so the user knows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



## Performance

As stated earlier, the performance of the site will be optimized by the fact that we only depend on an external API, but we aren’t using libraries or frameworks that might slower the loading time of the page. User’s system just has to download the HTML, CSS and JavaScript folder. Using a strong hosting with good worldwide coverage will also be important to achieve our performance goals.

## Section 508 Compliance

The application will accomplish with all the standards and rules to make the site as accessible as possible (by the use of a correct tagging in HTML, adding “alt” content when applicable, etc.). However, any suggestions on how to improve it even more with the help of people with disabilities will always be highly appreciated.

# 

Appendix A: References

The following table summarizes the documents referenced in this document.

|  |  |  |
| --- | --- | --- |
| **Document Name and Version** | **Description** | **Location** |
| Persson, M. (2020). JavaScript DOM Manipulation Performance: Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks. | A document to compare and magnify the advantages and disadvantages of the use of Vanilla JavaScript | <https://www.diva-portal.org/smash/get/diva2:1436661/FULLTEXT01.pdf> |
| Garrett, J. J. (2005). Ajax: A new approach to web applications. | A document to understand a base about the AJAX Technology | <https://www.scriptol.fr/ajax/ajax_adaptive_path.pdf> |

1. 10 Best Web Hosting Services (November 2022). Source: Forbes - <https://www.forbes.com/advisor/business/software/best-web-hosting-services/> [↑](#footnote-ref-1)