U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Licenciatura em Engenharia Informática e Computação

Ano Letivo: 2021/2022

GIG Arcade

Projeto Integrador

Grupo PE37

Ana Rita Antunes Ramada (201904565)
Diogo Filipe Moreira Maia (201904974)
Francisco Renato Barbosa Pires (201908044)
Luis Carlos Barros Viegas (201904979)
Tutor: Rui Rodrigues

1. Introduction **2**

2. Product Vision **3**
2.1 Product Relevance 3

3. Architectural Proposal **4**
3.1 Logical Architecture 4
3.2 Technologies Used 5

4. Implementation **6**
4.1 Navigation Interface 6
4.1.1 Main Components 6
4.2 Repository 7
4.3 Data Processing 7
4.4 Challenges 8

5. Results **9**
5.1 Overview 9
5.2 Game's Information 10
5.3 Search 11
5.4 Filters 12

6. Conclusions **13**

7. References **14**

1. Introduction

For over ten years, a portfolio of game design and development projects has been gathered, one of considerable size, at MIEIC and MM, which unfortunately cannot be easily accessed.
This project intends to create a system that not only allows managing a games repository, but also to provide an interface to browse and play these prototypes, which can be used for demonstration and divulgation purposes.
Thus, the main objective is the creation of a system that makes available and tests the developed prototypes, by running them on a machine - the “GIG “Arcade”.
The system will be supported by a server that will serve as a repository for the prototypes, allowing, on one hand, the synchronization with the arcade, and also, an easy mechanism to add future projects.

2. Product Vision

When envisioning our product we had some key features we wanted to see implemented, namely ease of use and maintenance. It would also be beneficial if there was already some type of system we could adapt to our needs instead of building everything from the ground up.
With that in mind, the solution that came up was a game launcher, similar to other popular game launchers like “Steam”, in which the user could have access to detailed information about the games and the ability to launch games directly from there.
For the database, we chose to make use of gitlab, in that way new projects could be added as submodules to a repository serving as a database for the GIG Arcade, making it easier to expand the library of available games as they are being developed.

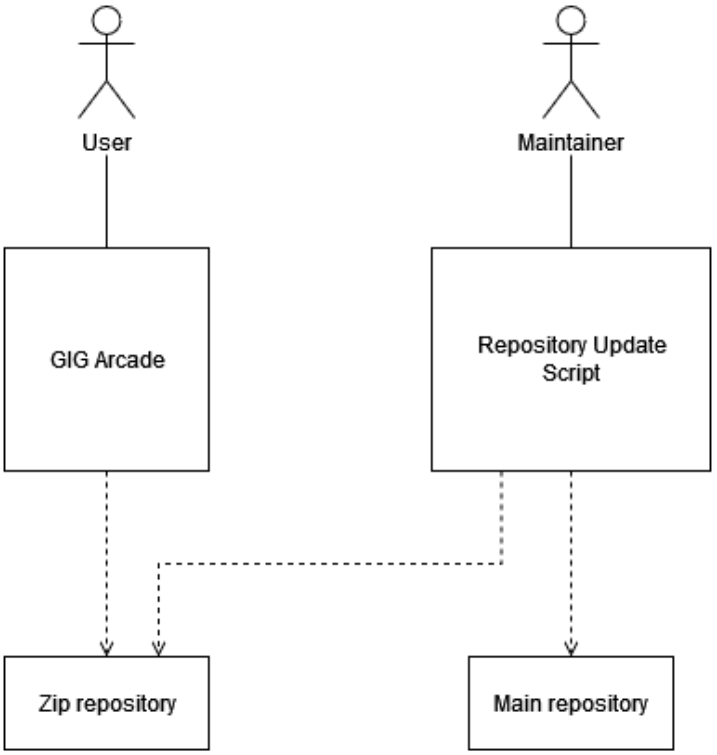
2.1 Product Relevance

At the moment, there is no product or platform that easily provides access to game projects developed by FEUP students.
This arcade, which will be available in the computer engineering department, will provide immediate access to these games for those who are curious to try them out, as well as an easy way for students and teachers to submit their new projects.

3. Architectural Proposal

In order to build the desktop application, the Electron framework was used. This framework supports CSS, JavaScript and HTML.
A powershell script allows the system maintainer to update the Zip Repository with new releases from Main.

3.1 Logical Architecture



GIG Arcade: The desktop application the users interact with. It is responsible for launching the games and presenting its information. To build the application, an Electron framework was used. This framework supports CSS, JavaScript and HTML.
Repository: Update Script: Allows the System Maintainer to update the Zip Repository with new project releases from the Main Repository.
Zip Repository: Holds the game releases in a compressed format so that the GIG Arcade may use it to update its local game data.
Main Repository: Is linked to the projects in development that may later be migrated to the Zip Repository at the time of release.

More details on these components can be found in the next topic, Implementation.

3.2 Technologies Used

Different technologies were used in this project. Our tech stack mainly consists of Electron and React [1], but we also use PowerShell [2], Git and WebPack.
Electron is used as a main structure for our desktop app, allows us to run the program, run the games and interact with the Operative System. This is the most important part of our program, it controls and initiates all of the other parts.
React is a Front-End library that we use to better display our contents. It allows for a prettier interface and a more accessible one, with interactive buttons, side menus, images and so on.
Git is used as a way to have a distributed database. With Git, we have a way for students to upload their work so that it's automatically integrated in the GIG Arcade system. Also, with Git, we have the possibility of one repository for every project so that every student only has access to their own project. This contributes to better security and integrity of the project.
PowerShell is used to communicate with git and organize the database inside the GIG Arcade, so that the files downloaded from Git are ready for Electron to import.
WebPack is used to bundle our Front-end and Back-end, Electron and React, respectively, in a distributed package.

4. Implementation

In order to streamline the implementation process, the project was divided into two main components, the application, that will serve as the navigation interface and data processor for the games, and the repository and its management/synchronization.

4.1 Navigation Interface

As referenced before, for the development of the front end of the application, it was decided to integrate the React library with the Electron framework.
React is an open source JavaScript library focused on creating user interfaces. Besides what was already mentioned, the main objective with its use in this project, was to build the interface based on components, making it easier to reuse them, and also, to minimize code conflicts, since two team members developed the front end.

4.1.1 Main Components

As mentioned, while implementing the navigation interface, we used different components.
The main one is called Body, it's where the Filter, Search and GridView components are used. The GridView, is where the user will be able to see all games available.
To interact with each game, either to see their information, displayed on the InfoCard component, or to launch a game, the user must interact with a game card, by clicking the info button, or play button, respectively.
More on the interface can be found at the results section of the report.
This next code block will demonstrate how the GridView component is made, serving as an example of how all the other components work and interact with each other.
Using a loop, the setGameCards function will return a set of game cards. Each GameCard will receive the respective game information.

```
function GridView(props) {
  function setGameCards() {
    let gameCards = [];
    for (let i = 0; i < props.games.length; i++) {
        gameCards.push(
        <GameCard
          className="game-card"
          image={getGameMediaPath(props.games[i].icon)}
          showInfo={props.showInfo}
          game={props.games[i]}
          name={props.games[i].name}
        ></GameCard>
      );
    }
    return gameCards;
   }
 return <div id="grid-container">{setGameCards()}</div>;
}
```

4.2 Repository

As mentioned previously, the arcade gets its game data from gitlab repositories.
There are actually two repositories: “main” and “zip”.
The first holds new projects as submodules and is to be solely used by the maintainer of the arcade that may add or remove projects.
Whenever a project deadline comes the students should assign a tag “RELEASE” to the delivery version of their project. The maintainer of the arcade can then run a simple powershell script on a directory containing both repositories. This script has a list of previous releases and checks the main repository for any new projects it does not recognize. For these new projects it will check for the aforementioned tag and,if found, it creates an archive (i.e. compresses the project to a .zip) and stores it in the “zip” repository, pushing any changes at the end.
It’s this “zip” repository that the arcade uses as the remote database. Whenever the application starts it pulls for changes in the repository and checks if there are any new archives it has not encountered yet and, if so, decompresses them into the “decompressed_games” directory from which the data will then be accessed.

4.3 Data Processing

After the powershell script which updates the local database ends its runtime, the application assumes the database is on pair with the remote database. In case the remote powershell ends in error, the application runs with the local database updated or not in the previous iteration of the application.
Assuming it all went as expected, the next step is to process each new game and filter any misspells and errors it may contain in its information sheet.
Initially, thanks to the powershell script, we have a text file containing the names of the directories corresponding to games introduced in the update and, with that information, we can go directly to the information files of the new games. This way, we can reduce the number of operations on the file system, by evading the need to go to each directory and check if a game represented by that directory already exists.
Secondly, we not only verify if every game has an information file (those who do not have that, will be ignored) but we also verify the file’s content on each information we need like game’s name, description, screenshots and icon filenames and it’s subsequent existence. Any time an information is out of boundaries or not specified as we need it, we fill it with a default one.
Lastly, we add all the new games’ information to a JSON file in order to maintain our decision of going to the file system as little as possible. This way we can access all the local games information by reading only one file.

4.4 Challenges

During the development of this project we faced various challenges, but two of them were the most time consuming. These two problems could have been avoided if we had more knowledge about the frameworks/libraries we used to develop the application but were problems we could only discover by testing, since we found the documentation of such problems were lacking in the regards of interactions with each other.

The first big challenge we faced was regarding the way we wanted to package our application. We started by replicating a simple React plus Electron boilerplate. The problem was that the boilerplate did not have any packaging incorporated into and development started without taking that under consideration. When the time to start developing releases came, we noticed that our filesystem had problems with the most renowned electron packaging modules: electron-builder and electron-forge [3] (being electron-forge the most famous for its consistency).
To solve this problem we had to use Webpack, a bundling module which not only allowed a better integration of React and Electron but also allowed us to utilize electron-forge to package the application. Although functional, this solution was very time consuming, because all the work had to be imported to the new file system and resolving new problems coming with this bundling style, such as asset loading and file copy to the end product, was also demanding.

The second big challenge relates to the front-end library and its ability to read dynamic inserted images. Because the application updates the local database on runtime, there was no way to import/require the images of the games' icons and their screenshots that were needed on runtime. This was a problem since the beginning of development, but decided to postpone it after solving the packaging issue. However, the problem got worse as we incorporated Webpack bundling. There is not much information on reading dynamic images from a webpack bundling incorporating React. Many things were tried, like dynamic import, public folder, etc, but the images wouldn't load.
However, by reading Webpack documentation [4] and how it's bundling works, we discovered that it already starts with a public folder, where all media could be loaded with a relative URL. We changed the database location to inside that folder, making all the media available there. The solution was pretty simple, but discovering it took a long time since we tried to go after the most common issues' solutions on React and Webpack documentations about topics like 'public folder' [5].

During development there were more minor challenges, but were quickly solved not posing any threat to the application development.

5. Results

At the time of the delivery, there was a temporary repository, working as a substitute for the real one. In order to test the implementation, 18 games were submitted to this git repository, simulating the conditions of a normal student, and everything worked perfectly.
The next screenshots will show the final product with more detail.

5.1 Overview

For the design of the arcade, we made sure to keep it simple and easy to use.
It only has one page, with three main components: the grid, where we have a card for each game available, a search button and a filter button. We'll talk about each component in more detail later on.

Image 1. GIG Arcade Interface Screenshot

The games are ordered by release date, so the most recent ones are shown first.
To quickly play any game, the user can click on the *Play* button on a game card. A new window will then open, and the game will launch.

5.2 Game's Information

After clicking the *Info* button, on the game card, a popup window will appear with information regarding the game.

Image 2. Information of the game Tales of DragonFall, shown on GIGArcade

The information card not only displays the information provided by the developers, but also some screenshots of the game, in some cases, a video and the *Play* button.
By clicking anywhere outside the card, the user goes back to the grid view, and the game's information will close.

5.3 Search

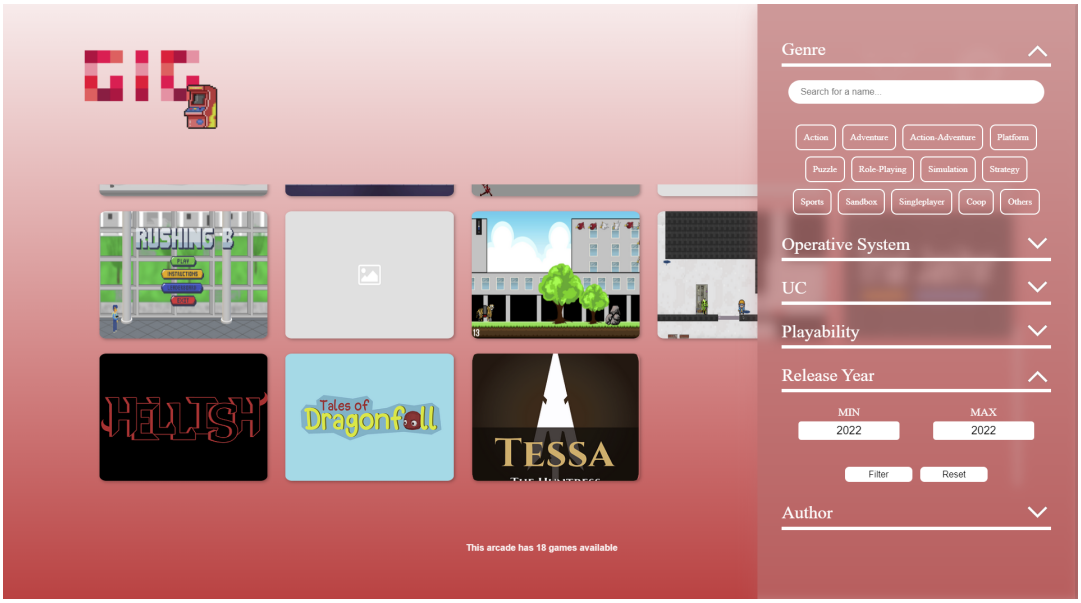
In order to search for a specific game, the user will click on the search icon, placed on the upper right corner, and a side menu will then open. The search bar is displayed at the top of the menu. Here the user will input a game title, or part of it, and the search results will appear at the bottom.

Image 3. Search side menu example

Just like the game information card, to close the search, the user needs to press anywhere outside the menu.

5.4 Filters

To be able to search for games through filters, the user has to click on the filter icon. A menu similar to the search one will then appear.
Here, six different filter options are available. Some of them are searchable, genre, UC and author, and the others are easily selectable, as it's possible to see in the picture below.



There was a discussion in the early planning stages, about how the filters would be applied. It was then decided that the search results would be the combination of all the filters applied, that is, if the CGRA curricular unit and the singleplayer mode are both selected, the games that will appear on the grid will be all the CGRA games and all the singleplayer games.
The filter menu closes in the same way as the search one. After it closes, and if any filter was applied, the games shown in the grid view will change accordingly and the selected filters will be shown by the left side of the filter icon.

6. Conclusions

To conclude our report, we believe we learned a lot with this opportunity, and we consider this to be one of the most enriching experiences of our last few years at FEUP.
We improved our knowledge on the process of software engineering, and the basic interactions between a team and a client, and the importance of beginning product development only after the basic requirements are ready and all the plan of work is traced. We were also made aware of the importance of properly learning how new frameworks/libraries work and interact before starting to work on them.
These are some points in which we generally improved, however, we also gained a lot of knowledge coming from the context of the work itself. Since we decided this is an application for Windows (since the majority of games were only playable on the Windows operative system), we learned about how its processes and threads work and all its capabilities and also how to structure filters and the difficulties behind a comprehensive and adaptable filter system.
The product is in a launchable state however there is some room for improvement. One of the things we would like to improve is the UI design of the media carousel of each game, allowing videos to appear on the lower carousel options. Other things we would like to improve are the algorithm behind the game's filtering and maybe the game's template of information, to enable a bigger flexibility on the game's name.
Besides improvements, there are some features we would make if we had more time with the product. The most obvious would be a button to clear the current filters. Furthermore, there were other plans for the application, like adding a carousel view of the games, a QR code option on non-playable games and the possibility of manually inject games on the local database, all of which were not possible due to time constraints.
In the end, we hope our project can be a stepping stone for a great arcade system, one DEI aims to achieve. We also hope to help FEUP's students, not only to showcase their gaming projects, but to entertain the ones who just wish to play a few games and check what their peers were able to accomplish in one semester.

7. References

1. Mandi Wise 2020, *DEV Community,* accessed 28 April 2022, <https://dev.to/mandiwise/electron-apps-made-easy-with-create-react-app-and-electron-forge-560e>;
2. Alexandra Altvater 2017, *Stackify,* accessed 2 May 2022, <https://stackify.com/powershell-commands-every-developer-should-know/>;
3. 2020, *Electron Forge,* accessed June 2022, <https://www.electronforge.io/templates/webpack-template>;
4. Rafael Rinaldi, Chris Villanueva, Olivier Gonzalez, Sam Chen, *Webpack,* accessed June 2022, <https://webpack.js.org/guides/public-path/>;
5. Meta 2021, *Create React App*, accessed June 2022, <https://create-react-app.dev/docs/using-the-public-folder/>;