B.Sc. (Hons) in Software Development

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

ARK Config File Manager

By
Ryan Hogan

April 25, 2025

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

Contents

1 Introduction **2**
- 1.1 Project Overview . . . 2
- 1.2 Problem Statement . . . 2
- 1.3 Target Audience . . . 2
- 1.4 Project Objectives . . . 3
- 1.5 Technical Context . . . 3
- 1.6 Example of Configuration File . . . 3
- 1.7 Motivation . . . 4

2 Methodology **5**
- 2.1 Development Approach . . . 5
- 2.2 Requirements Gathering . . . 5
- 2.3 Technical Research . . . 6
- 2.4 Tool and Technology Selection . . . 7
- 2.5 Implementation Process . . . 7
- 2.6 Testing and Evaluation . . . 8

3 Technology Review **9**
- 3.1 Selected Technologies . . . 9
- 3.2 Related Work . . . 10
- 3.3 Summary . . . 11

4 System Design **12**
- 4.1 Architecture Overview . . . 12
- 4.2 Backend . . . 12
  - 4.2.1 Main Class . . . 12
  - 4.2.2 Preload Class . . . 14
- 4.3 Frontend . . . 14
  - 4.3.1 view_page . . . 14
  - 4.3.2 account_page . . . 16
  - 4.3.3 share_page . . . 16
- 4.4 Assets . . . 16
  - 4.4.1 lists . . . 17
- 4.5 electronAPI Interface . . . 17
- 4.6 Communication Flow . . . 17
- 4.7 Conclusion . . . 18

5 System Evaluation **19**
- 5.1 Objectives . . . 19
- 5.2 Satisfaction . . . 19
- 5.3 Future Improvements . . . 20

5.4 Testing 20
5.5 Challenges 20

6 Conclusion **22**
6.1 Project Overview 22
6.2 Objectives and Achievements 22
6.3 Technical Implementation 22
6.4 Comparison and Innovation 23
6.5 Future Work 23
6.6 Development Methodology 23
6.7 Summary 23

7 Appendix **25**
7.1 Links 25

List of Figures

3.1 Beacon app logo 10

4.1 UML diagram of **ARK Config File Manager** 13

List of Tables

Chapter 1

Introduction

1.1 Project Overview

The **ARK Config File Manager** is a program created to aid a user in managing configuration files for the games *ARK: Survival Evolved*[1], *ARK: Survival Ascended*[2] and *ARK II*[3] developed by Studio Wildcard[4] and published by Snail Games[5].

1.2 Problem Statement

The *ARK* games have a large number of configurations accessible in-game and are quite straight forward. A casual player will be content with this level of configuration but for server owners or hardcore players, the in-game configuration menus are insufficient. In fact, there is no in-game configuration menu for server owners. The only way to configure the game is to edit the configuration files directly. This is a tedious process and can lead to errors if the user is not careful. The **ARK Config File Manager** aims to make this process easier by providing a graphical user interface (GUI) for editing configuration files.

1.3 Target Audience

The intended audience, as previously mentioned, is hardcore players and server owners as casual players will likely have no interest in further editing of their configuration files. These users will benefit greatly from this program as instead of highlighting then replacing a value for say, a true or false boolean, they can simply toggle it in the GUI with a single left click. This will save time and reduce the risk of errors when editing configuration files.

1.4 Project Objectives

The main objectives of the project are to ease the process of editing configuration files for the *ARK* games. The program will allow the user to open a configuration file, view its contents and edit them in a GUI. It goes without saying that the program will also allow the user to save the edited configuration file as a new file and the user can then load it into the game by replacing the file in the correct directory.

There is also another section to the program called the share page. This will allow users to share their configuration files with others and download configuration files from other users. This will be a great way for users to find new configurations and share their own. These configurations have a download/import counter, a rating system and a comments section so the user who is browsing can get an idea of what the configuration file does and if it is worth downloading. The user can also rate the configuration file and leave a comment on it. This will help other users to find the best configuration files and will also help the user to get feedback on their own configuration files.

1.5 Technical Context

Configuration files in the *ARK* games are stored in `.ini` files. These files are plain text files that can be opened and edited with any text editor. The configuration files are divided into sections, each section containing a number of key-value pairs. The key is the name of the configuration option and the value is the value of the configuration option. The key-value pairs are separated by an equals sign (`=`). The sections are separated by square brackets (`[ ]`). The **ARK Config File Manager** will allow the user to view and edit these configuration files in a GUI. The GUI will display the sections and key-value pairs in a table view, allowing the user to easily navigate the configuration file. The user will be able to edit the key-value pairs by clicking on the value cells and entering a new value, toggling it, etc. The program will also allow the user to add new sections and key-value pairs to the configuration file. When hovering over one of the key cells, the user will be able to see a tooltip with the description of what the key does in game. For keys without predefined descriptions, an AI-generated description will be provided, though these may not be 100% accurate.

1.6 Example of Configuration File

Below is an example of a configuration file from ARK: Survival Evolved:

```
[ServerSettings]
DisableStructureDecayPvE=True
DontAlwaysNotifyPlayerJoined=False
EnableExtraStructurePreventionVolumes=False
EnablePvPGamma=True
FastDecayUnsnappedCoreStructures=False
ForceAllowCaveFlyers=True
globalVoiceChat=True
ItemStackSizeMultiplier=1.0
KickIdlePlayersPeriod=36000000.0
ListenServerTetherDistanceMultiplier=9999.0
MaxPlayers=6
MaxTamedDinos=5000
MaxTributeItems=100
MaxTributeDinos=100
NightTimeSpeedScale=0.5
NonPermanentDiseases=False

[Ragnarok]
AllowMultipleTamedUnicorns=True
UnicornSpawnInterval=2
```

I have only grabbed a certain sample from around the centre of the first section since a lot of the beginning is true or false. As you can see, the configuration file is divided into sections, each section containing a number of key-value pairs. The key is the name of the configuration option and the value is the value of the configuration option.

1.7 Motivation

The motivation for this project was simply to make something that is usually time consuming and tedious, easier and more enjoyable. The *ARK* games are very popular and have a large community of players. Many of these players are hardcore players who spend a lot of time configuring their games to get the best experience possible. The **ARK Config File Manager** will make this process easier and more enjoyable for these players. The program will also allow users to share their configuration files with others, making it easier for users to find new configurations and share their own.

Chapter 2

Methodology

2.1 Development Approach

The way I handled this project was using an incremental development methodology, not any specific framework like Agile or Waterfall. My main goal was to create each feature of the application one after the other and I believe this approach was appropriate for this project as a solo developer because:

- I was able to focus on one feature at a time, which helped prevent anxiety and overwhelm from the daunting task that is a full final year project.
- In between each feature I could feel at ease and a sense of accomplishment as I had a working feature to show for my efforts.
- I was able to calmly test each feature as I went along, which helped me to identify and fix bugs early in the development process.
- Since my program was modular in nature, I could easily add new features without having to worry about breaking existing functionality.

Each feature I wanted to add to my program was broken down into smaller tasks, which made it easier to manage and implement. I would start with the design of the feature, then move on to the implementation, followed by testing and refinement before moving on to the next feature. This approach allowed me to build a solid foundation for my program and ensured that each feature was well thought out and implemented correctly.

2.2 Requirements Gathering

This is basically the question of what does my program need to do? Requirements for the **ARK Config File Manager** were derived from my personal experience

with the *ARK* game's configuration files and knowledge of the usual pain points of manually editing these files. The requirements of my program involve:

- Displaying my configuration files in a user-friendly way.
- Allowing the user to edit the configuration files in a GUI.
- Allowing the user to save these files.
- Allowing the user to share their configuration files with others.
- Allowing the user to download configuration files from other users.
- Allowing the user to rate and comment on configuration files.
- Allowing the user to view the rating and comments on configuration files.

While most of these requirements were derived from my own personal experience, I also took to the internet to ask on the online platform *Reddit* in the *ARK: Survival Evolved* subreddit[6] to see if there were any other features that I could add to my program that would be useful to other players. I received a good few responses from the community, which helped me to refine my requirements further.

2.3 Technical Research

Before going in to this I had known of an existing app called *Beacon*[7] that is far more advanced than my own program and has been around for quite a while. I had used this app in the past and I was aware of its features and functionality. I also knew that it was a very popular app in the *ARK* community, so I knew that I didn't want to compete with it but instead just be inspired by it. It's concept is pretty similar but it is a lot more advanced and perhaps a bit daunting for new players. I wanted to create a program that was simple and easy to use, while still being powerful enough to handle the needs of the player's who wanted to use it. As for the potential technologies on what I could use to create my program, I had a few ideas in mind. I had heard of *Electron*[8] before and decided to do some more digging on it. I found that it was a very popular framework for creating cross-platform desktop applications using web technologies like HTML, CSS, and JavaScript. I also found that it was very easy to use and had a lot of documentation and community support. I had considered using *React*[9] as I had used it before, but decided to keep things simple and use plain old JavaScript, HTML and CSS. I had a few choices for my databases. I had considered either *Firebase*[10], *MongoDB*[11] or *SQLite*[12]. I had used Firebase during the first

semester of my final year so I had some familiarity with this. Considering the recency bias I decided to go with this. I only briefly looked in to *MongoDB* and *SQLite* because of said recency bias. I also needed some AI generation for my project so I only really considered two different choices. *Google Gemini*[13] and *ChatGPT*[14]. These were pretty big names at the time of choosing and I had heard of both of them before. I decided to go with *Gemini* as it works with *Firebase* since *Google*[15] owns both of them. I also had a bit of experience with *Gemini* from the first semester of my final year so I decided to go with this.

2.4 Tool and Technology Selection

The reason I selected *Electron*, *Firebase*, and *Gemini* is because they are all quite popular, well documented, and have a lot of community support. *Electron* and *Gemini* in particular I had not used before so I wanted to give myself a bit of a challenge and learn something new. *Firebase* was an obvious choice since it was developed by *Google* and *Gemini* was also developed by the same crowd meaning they'd likely pair well together. These technologies fulfilled the needs of my program. *Electron* allowed me to create a cross-platform desktop application using web technologies, which made it easier to develop and maintain. *Firebase* allowed me to store and retrieve data easily such as storing user shared files and AI generated descriptions so that they don't have to be regenerated upon each load of the app, and also provided a lot of features like authentication for the user accounts on the app. *Gemini* allowed me to generate AI descriptions of settings in the *ARK* games for my program, which was a key feature that I wanted to include.

2.5 Implementation Process

I tried to follow the incremental development methodology as closely as possible. I started with the basic structure of the application, which included the main window and the menu bar. I then moved on to implementing the first feature, which was displaying the configuration files in a user-friendly way. Once that was done, I moved on to the next feature, which was allowing the user to edit the configuration files in a GUI. I continued this process until all of the features were implemented. I also made sure to test each feature as I went along, which helped me to identify and fix bugs early in the development process. I manually tested everything to ensure that each feature was working correctly. I also made sure to document my code with comments as I went along, which helped me to keep track of what I was doing and made it easier to maintain the code later on. I used *Git*[16] in combination with *GitHub*[17] for version control. I frequently

commited my code to the repository, which allowed me to keep track of changes and revert back to previous versions if needed. This was also very useful to remind myself of what I had been working on last if I had taken a break from the project for a while. I used modules in my code to keep things organized and modular. This made it easier to manage and implement each feature, as I could focus on one module at a time. I seperated my pages in to different folders, which made it easier to navigate the codebase and find what I was looking for. I used a consistent naming convention for my files and folders, which made it easier to understand the structure of the codebase. Each page's modules were contained in the same folder as the page itself, which made it easier to manage and implement each feature.

2.6 Testing and Evaluation

I manually tested each feature as I went along, so that I didn't go ahead and implement another feature when the previous one was broken. I also had someone else test the program and it seemed to work fine.

Chapter 3

Technology Review

3.1 Selected Technologies

Electron plays quite a big role in my program as it is the framework that I used to create the desktop application. It allows me to use web technologies like HTML, CSS, and JavaScript to create a cross-platform application that can run on Windows, Mac, and Linux. This was a key requirement for my project as I wanted to make sure that my program could be used by as many people as possible. Allowing me to use JavaScript, HTML, and CSS also made it easier for me to develop the application as I was already familiar with these technologies. *Electron* also has a lot of documentation and community support, which made it easier for me to find help when I needed it. It is kind of like your skeleton. It provides the basic structure of the application and allows you to build on top of it. It also handles a lot of the low-level details of creating a desktop application, which allows you to focus on the higher-level functionality of your program. You don't really think too much about it after it's set up, but it is a very important part of the application. It is also very easy to use and has a lot of features that make it easier to create a desktop application. Without it, I would have had to make this a web application, which sounds like a bit of a headache to me. I also don't think it would have been as user friendly as a desktop application.

Firebase is the database that I used to store and retrieve data for my program. It is a cloud-based database that allows you to store and retrieve data easily. It also provides a lot of features like authentication, which makes it easier to manage user accounts. I chose *Firebase* because it was familiar to me and had a lot of tutorials and documentation online. It also integrates well with *Electron*, which made it easier for me to use it in my program. *Firebase* provides a lot of handy dandy features like user authentication and a database. It made creating an account system quite easy and handles permissions for my database quite well. Without

Firebase, I would have had to do a lot of the heavy lifting myself, which I would not have enjoyed.

Google Gemini is the generative AI tool that I chose for my tooltip generation. This is a crucial yet small part of my program. One of the major objectives of my program is to allow people to figure out what each setting does. AI may not be 100% accurate because it is a game and not a real life thing, but it is a lot better than nothing. I chose *Gemini* because it was developed by *Google* and would play nice with *Firebase*. *Gemini* allows you to generate text based on a prompt, which is perfect for my use case. I can send it a prompt like "What does the setting 'X' do?" and it will return a description of what that setting does. This is a key feature of my program and without it, I would have had to manually write descriptions for each setting, which would have been a lot of work. I am grateful that all these technologies exist and without it, I would not have been able to create this program. I am also grateful that they all work well together and are easy to use. I believe that these technologies are the best choice for my project and I am happy with the way it turned out.

3.2 Related Work

Figure 3.1: Beacon app logo

There is a similar program out there called *Beacon*[7] that is far more advanced than my own program and has been around for quite a while. I had used this app in the past and I was aware of its features and functionality. I also knew that it was a very popular app in the *ARK* community, so I knew that I didn't want to compete with it but instead just be inspired by it. It's concept is pretty similar but it is a lot more advanced and perhaps a bit daunting for new players. I wanted to create a program that was simple and easy to use, while still being powerful enough to handle the needs of the player's who wanted to use it. My program differs from this in that it is a lot more simpler of an app visually and feature wise.

My vision for the app was simplicity for the user, while still being a useful tool. My program also differs in that it has a built in sharing system complete with ratings and comments. This is a feature that I believe is missing from *Beacon* and would be a useful addition to it. *Beacon* is great in that it offers in depth customisation of the configuration files and a lot of features that I don't have, but I believe that my program is a good alternative for players who want something simpler and easier to use. I also believe that my program is a good starting point for players who are new to the game and want to learn more about the configuration files and how they work. I hope that my program will help players to understand the configuration files better and make it easier for them to use them. My program is also entirely free, where as *Beacon* has portions of the program that are locked behind a paywall.

3.3 Summary

Due to my project being so niche, there weren't any papers or anything I could refer to about this project. There were only papers on unrelated projects that use the same technologies like *Electron* and *Firebase* but never both at the same time. Unfortunately this chapter was quite short as a result.

I imagine there is a lot better technologies out there I could have chosen, but I am happy with the ones I chose. These technologies allowed me to create the program in the way I imagined and I do not recall ever being held back by my chosen technologies.

Electron did the job of allowing me to make an application that runs on the desktop instead of a web browser. It also came with the added bonus of being able to use the app on all platforms (Windows, Mac, and Linux). The languages I could use in *Electron* were also a big benefit for me since I enjoy using JavaScript, HTML, and CSS.

Firebase was a great choice for me since it came without cost and was relatively easy to use. Having access to a database to support my program and also having a user authentication system out of the box for me was a big bonus.

Gemini being developed by *Google* meant it would work well with *Firebase* since they are both owned by *Google*. *Gemini* allowed me to do all the generative AI I needed for my program without issue. I was able to generate descriptions of the settings in the *ARK* games, which was a key feature that I wanted to include.

Chapter 4

System Design

My apologies for the hard to read diagram at Figure 4.1 but I am unsure on how to make it more readable. You will just have to zoom in on it.

4.1 Architecture Overview

The **ARK Config File Manager** uses a modular architecture with clear separation between frontend and backend components. As shown in Figure 4.1, the application is split into three main parts: Backend, Frontend, and Assets.

4.2 Backend

The backend of the application contains the core functionality that powers the entire system:

4.2.1 Main Class

The Main class serves as the entry point and application core, handling core operations:

- Management of application state through properties like `isDev`, `isMac`, and dimensions.
- File system operations including creating, reading, removing, and saving files.
- Firebase integration for authentication and data storage.
- Gemini AI integration for generating tooltips.

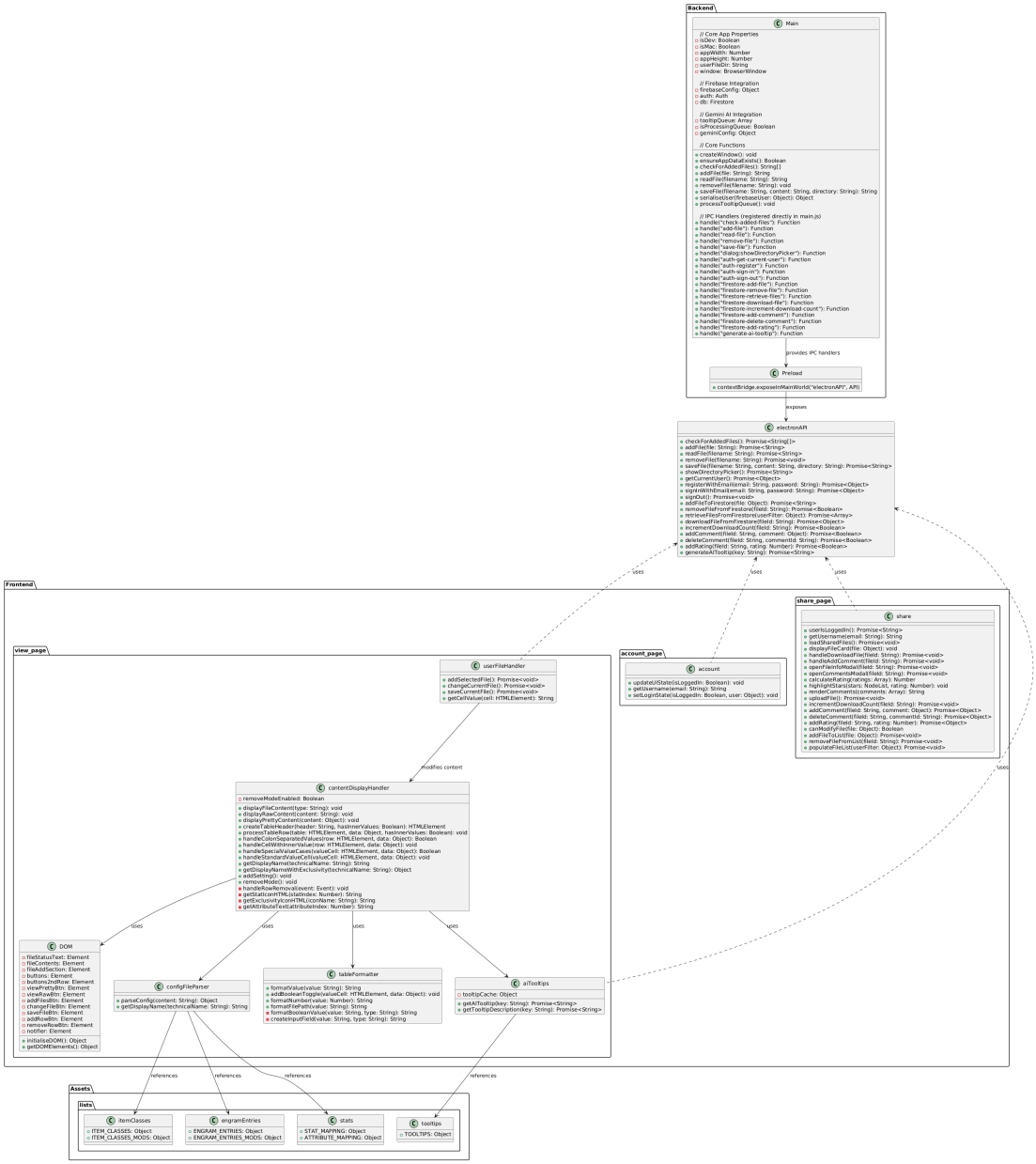

Figure 4.1: UML diagram of **ARK Config File Manager**

- User data serialisation for authentication purposes.
- Window creation and management via the `createWindow()` method.
- Processing of tooltip generation queue.

The Main class also registers all IPC handlers directly, serving as the communication bridge between the frontend and backend. These handlers include methods for:

- File operations (adding, reading, removing, saving).
- Directory selection dialogs.
- Authentication (registration, sign-in, sign-out).
- Firestore operations (adding, removing, retrieving files).
- Social features (comments, ratings, download tracking).
- AI tooltip generation.

4.2.2 Preload Class

The Preload class is a bridge between the Main class and the frontend. It exposes the backend functionality to the frontend through a promise-based interface.

4.3 Frontend

The frontend is divided into three main pages, each handling different aspects of the user interface:

4.3.1 view_page

This contains components responsible for displaying and manipulating configuration files:

DOM Class

Manages the Document Object Model elements and provides access to UI components:

- Initialises and retrieves DOM elements via `initialiseDOM()` and `getDOMElements()`.
- Contains references to UI elements like buttons, file status text, and content areas.
- Organises UI elements into logical groups (buttons, content sections).

configFileParser Class

Handles the parsing of *ARK* configuration files:

- Changes raw configuration text into structured objects via `parseConfig()`.
- Maps technical names to user-friendly display names.

contentDisplayHandler Class

Is probably the most important to the user interface functionality:

- Manages display modes (raw and pretty views) of configuration files.
- Constructs table representations of configuration data.
- Handles special value cases and formatting.
- Provides functions for adding and removing settings.
- Manages row removal functionality through a dedicated mode.
- Provides visual enhancements like stat icons and exclusivity indicators.

tableFormatter Class

Provides formatting services for table displays:

- Formats various value types appropriately (numbers, booleans, file paths).
- Creates interactive elements like boolean toggles.
- Makes input fields for different data types.

aiTooltips Class

Creates AI capabilities for greater user experience:

- Manages tooltip caching to improve performance.
- Retrieves AI-generated descriptions for configuration settings.
- Provides asynchronous access to tooltip information.

userFileHandler Class

Manages file operations from the user interface:

- Handles file selection and addition.
- Manages file changes and saves.
- Extracts values from table cells for saving.

4.3.2 account_page

Handles user authentication and account management:

- Updates UI state based on authentication status.
- Extracts usernames from email addresses.
- Manages login state across the application.

4.3.3 share_page

Adds social features within the application:

- Displays shared configuration files from other users.
- Creates visual representations of shared files as cards.
- Handles file downloads, comments, and ratings.
- Provides modals for file information and comments.
- Manages file uploads and sharing.
- Controls permissions for file modification.

4.4 Assets

Assets contains reference data important to the application's operation:

4.4.1 lists

Contains several classes that provide mapping and reference data:

- `itemClasses`: Maps the item classes to their in game names.
- `engramEntries`: Maps engram entries to their in game names.
- `stats`: Maps technical stat identifiers to user-friendly names and attribute mappings.
- `tooltips`: Provides predefined tooltip content for common configuration settings.

4.5 electronAPI Interface

A crucial interface bridging the frontend and backend:

- Provides Promise-based access to all backend functionality.
- Handles file operations (checking, adding, reading, removing, saving).
- Manages directory selection.
- Facilitates authentication (register, sign-in, sign-out).
- Provides Firestore integration (adding, removing, retrieving files).
- Enables social features (downloads, comments, ratings).
- Facilitates AI tooltip generation.

4.6 Communication Flow

The system architecture follows a communication pattern:

- The Main class implements IPC handlers to expose backend functionality.
- The Preload class bridges these handlers to the electronAPI interface.
- Frontend components (account, share, userFileHandler, aiTooltips) use electronAPI to access backend functionality.
- The contentDisplayHandler manages UI components using DOM, configFileParser, tableFormatter, and aiTooltips.
- Reference data from Assets informs parsing and display functionality.

4.7 Conclusion

The **ARK Config File Manager** implements a clean separation between frontend and backend components. The backend handles core functionality like file operations, authentication, and AI integration, while the frontend delivers an intuitive user interface across three main pages. Communication flows through the electronAPI interface, ensuring smooth data exchange between components. This modular architecture allows for maintainability and future extensibility of the application.

Chapter 5

System Evaluation

5.1 Objectives

Compared to what I set out to do in the introduction, I believe I have achieved most of the objectives to a satisfactory level. I have all the core features I wanted to implement and they all work as intended. All I could add to the project now is to continue polishing existing features, refactoring the code to be cleaner, and so on. The features I set out to implement were:

- Displaying my configuration files in a user-friendly way.
- Allowing the user to edit the configuration files in a GUI.
- Allowing the user to save these files.
- Allowing the user to share their configuration files with others.
- Allowing the user to download configuration files from other users.
- Allowing the user to rate and comment on configuration files.
- Allowing the user to view the rating and comments on configuration files.

5.2 Satisfaction

I have all of these features implemented and they all work as intended. I am happy with the way the program turned out and I believe it is a useful tool for players of the *ARK* games. I am also happy with the way the program looks and feels, as I believe it is user-friendly and easy to use. I am also happy with the way the program performs, as it is fast and responsive. Overall, I am very pleased with the way the project turned out and I believe it is a success.

5.3 Future Improvements

The parts of the program that I would polish further would be adding predefined descriptions for the settings rather than relying on the generative AI to generate all the setting descriptions for the tooltips. I decided not to do this as one: I was running out of time and two: I wanted to leave it up to the generative AI for demonstration purposes. If I had added predefined descriptions, I wouldn't be able to show off the generative AI as much. I would also like to add a few more features to the program, such as a search function for the configuration files and a way to filter the configuration files by rating or comments. I would also like to add a way for users to report inappropriate content in the comments or ratings. I believe these features would make the program even more useful and user-friendly.

5.4 Testing

How I went about testing my program wasn't anything too sophisticated. I simply went through each feature and made sure it worked as intended. I also had someone else test the program and it seemed to work fine.

I didn't have any major issues with the program and I believe it is stable and reliable. I did encounter a few minor bugs, but they were easy to fix and didn't cause any major issues. I did not test the program on different platforms (Mac and Linux) to ensure that it works on all of them simply because I don't have a MacOS or Linux system to test them on. I have faith in *Electron* that it would work on all platforms as it is designed to be cross-platform. I also did not test the program on different versions of Windows, but I believe it should work on all of them that work with *Electron* as well.

I squashed bugs as they came up while testing a feature upon completion of said feature. If anything big happened to mess up, I always had *GitHub* to revert back to a previous version of the code.

Through my code comments and *GitHub* commit messages, I was able to keep track of what I was doing and what I had changed. This made it easier to identify and fix bugs early in the development process.

5.5 Challenges

As for challenges, I'd say the most challenge I faced was learning how to use *Electron* and *Gemini*. I had never used either of them before and I had to learn how to use them from scratch. I found the documentation for both of them to be quite helpful, but it still took me a while to get used to them. I also had to learn how to use *Firebase* as well, but I had some experience with it from the

first semester of my final year so it wasn't too difficult to get a grasp of it. I had previous knowledge of how to use *JavaScript*, *HTML*, and *CSS* so I didn't have to learn those from scratch. I also had some experience with *Git* and *GitHub* so I didn't have to learn those from scratch either.

Chapter 6

Conclusion

6.1 Project Overview

The **ARK Config File Manager** was made to address a specific need within the *ARK* gaming community. Simplifying the time consuming process of editing configuration files for *ARK: Survival Evolved*, *ARK: Survival Ascended*, and the upcoming *ARK II*. This program aimed to transform what was typically a tedious, error prone manual process into an intuitive, user friendly experience.

6.2 Objectives and Achievements

The primary focus of this project was to create an application that would allow users to view and edit configuration files through a graphical user interface (GUI), save these modified files, and add social features that enable players to share, rate, and comment on configuration files. As demonstrated in the evaluation chapter, these core objectives have been successfully achieved.

6.3 Technical Implementation

Through an incremental development approach, I was able to build a modular application with clear separation between frontend and backend components. The use of *Electron* enabled cross platform compatibility, whilst *Firebase* provided great authentication and storage capabilities. The integration of *Gemini's* AI technology enhanced the application with intelligent tooltips that explain configuration settings to users.

6.4 Comparison and Innovation

Unlike more complex alternatives such as *Beacon*, the **ARK Config File Manager** prioritises simplicity and accessibility, making it particularly useful for new players or those who find existing solutions overwhelming. The social features implemented in the share page represent an innovation that facilitates community knowledge sharing and collaboration.

6.5 Future Work

While there remain opportunities for further work, such as adding predefined descriptions for settings rather than relying solely on AI generation and implementing additional quality of life features like search functionality and content moderation tools, the current program successfully meets all the core requirements established in the project's objectives.

6.6 Development Methodology

The development process followed a structured methodology with requirements gathering informed by personal experience and community feedback from platforms like *Reddit*. Technical research and tool selection were chosen carefully, with consideration of alternatives like *MongoDB* and *SQLite* before deciding on *Firebase*. Version control was maintained through *Git* and *GitHub*, allowing organised development and documentation.

6.7 Summary

In summary, the **ARK Config File Manager** successfully addresses the need for a user friendly tool to manage configuration files for the *ARK* games. Its combination of an intuitive interface, AI enhanced guidance, and social sharing features provides value to the community while demonstrating the successful application of modern web and desktop application technologies to solve these problems. The modular architecture ensures that future enhancements can be easily implemented, allowing the application to evolve alongside the needs of its user base.

Yes, this is a relatively short dissertation, but I would rather be short and sweet rather than long and repetitive just for the sake of adding more words. I believe I have covered everything I needed to cover and I am happy with the way the project turned out. I hope that this dissertation has provided a clear overview

of the project and its objectives, as well as the technologies and methodologies used to achieve them. Thank you for reading!

Chapter 7

Appendix

7.1 Links

- **GitHub Repository:** `https://github.com/Ryano2022/ARK-Config-File-Manager`
- **Demonstration Video:** `https://youtu.be/t-UE45hakVk`

Bibliography

[1] Studio Wildcard. ARK: Survival Evolved, 2017. Video game.

[2] Studio Wildcard. ARK: Survival Ascended, 2023. Video game.

[3] Studio Wildcard. ARK II, Unreleased. Video game.

[4] Studio Wildcard. Video game developer.

[5] Snail Games USA. Video game publisher.

[6] Reddit Contributors. Feature suggestions for ARK Config File Manager, 2024. Online discussion.

[7] Thom McGrath. Beacon Server Manager, 2016. Server management tool for ARK: Survival Evolved.

[8] GitHub, Inc. Electron.js, 2013. Open-source framework for building cross-platform desktop applications with web technologies.

[9] Meta Platforms, Inc. React, 2013. JavaScript library for building user interfaces.

[10] Google LLC. Firebase, 2011. Platform for developing mobile and web applications.

[11] MongoDB Inc. MongoDB, 2009. Cross-platform document-oriented NoSQL database program.

[12] SQLite Consortium. SQLite, 2000. Embedded relational database management system.

[13] Google LLC. Google Gemini, 2023. Large language model AI system.

[14] OpenAI. ChatGPT, 2022. Large language model AI chatbot.

[15] Google LLC. Google, 1998. Web search engine and technology company.

[16] Linus Torvalds and contributors. Git, 2005. Distributed version control system.

[17] GitHub, Inc. GitHub, 2008. Web-based hosting service for version control using Git.