Project Group 5
Johan Söderström
Axel Nilsson
Pavlos Papadopoulos

johan.soderstrom.9461@student.uu.se
axel.nilsson.3477@student.uu.se
pavlos.papadopoulos.0827@student.uu.se

https://github.com/Boxman1337/CommonSteamGames-

Common Steam Games

February 2021

1DL201

Program Design and Data Structures

1. Introduction

Steam is a gaming platform where people from all over the world come together to play games. The platform hosts roughly thirty thousand games and over a hundred million users[1], which has made online gaming rather convenient due to the vast amount of games and the large user base. But despite this larger amount of games on Steam, finding a game to play together with your friends has always proved to be quite difficult. Therefore, we decided to create a program that, although would not eliminate the problem, would make it easier to find games to play with your friends. We figured that one great way to help people, especially those with large libraries of games, would be to actually present all the games owned by everyone in this party of people that would want to play together. As previously mentioned, this would not entirely remove the hard decision of finding games to play together with your friends, but it would make it easier, as it gives you a list of games available for you to choose from without having to spend money buying the game you decide upon. Additionally, this list would not necessarily show you a list of games that everyone would want to play but rather a list of available games that everyone can play, without having to spend money. Knowing this, we were still interested in creating a program that would search for common games in our Steam game libraries, as there is currently no official way of doing this within Steam. We thought this program would be fitting, as we initially wanted to create a program that would make use of a public API (Application Programming Interface) and gather information from the internet.

Objective

Our objective was to create a program that takes the Steam64 ID of two or more people and then returns a list with all the games those people have in common in a comprehensible manner.

---

[1] "Steam, The Ultimate Online Game Platform", in *Store.steampowered.com*, , 2021, <https://store.steampowered.com/about/> [accessed 3 March 2021].

Background

We decided to create this program because we were interested in becoming more familiar with creating programs that retrieve data from the internet. Since the project assignment in the course Program Design and Data Structures was to create a program in Haskell, this idea was chosen as we thought it could be a potentially useful software. We also wished to become more knowledgeable in parsing JSON-files and presenting data gathered from JSON-files in a readable and well-kept manner. Additionally, we hoped to create an efficient workflow and become accustomed to working together with a Github repository, as we figured that it would prove useful in our future group projects.

Steam

Steam is a digital gaming and online distribution platform launched by Valve in 2003.[2] In Steam's *Store Page*, users can purchase digital copies of various video games which will then be linked to their particular account, rather than the computer itself. Additionally, Steam has a *Community Page*, which acts as the social section of the platform where users can add each other as friends, send messages to each other and write reviews for the games they have played.[3] Users can also create modifications and skins for various games and then release them in their official workshop, for other users to download and enjoy.

2. Using the program

User requirements

Public Steam settings

In order for the Steam API to be able to extract data from a Steam profile, that profile is required to be public. If your profile is private, you can make it public by going to your profile and pressing "Edit Profile" on the right. Once you press it, look for the privacy settings on the left side of the screen. It should be located somewhere on the bottom. Once in

[2] T Walker, "GDC 2002: Valve unveils Steam", in *GameSpot*, 2021, <https://www.gamespot.com/articles/gdc-2002-valve-unveils-steam/1100-2857298/> [accessed 3 March 2021].
[3] "Steam (service)", in *En.wikipedia.org*, , 2021, <https://en.wikipedia.org/wiki/Steam_(service)#User_interface> [accessed 3 March 2021].

the privacy settings, you can turn your profile public by clicking the highlighted and underlined text followed by “My Profile: ”.

Steam64ID

In order to garner the relevant information on each Steam user, the program requires its user to enter their Steam64IDs. There are multiple ways of finding your Steam64ID, however the easiest way would be to use an external Steam ID converter, such as steamid.io, that takes a link to your Steam profile and returns a list of all your Steam IDs. It is also possible to get a hold of your Steam64ID through Steam by going to your profile, right clicking and opening the source code for your profile page. A notepad should appear where you can search for “g_steamID” which should be followed by your ID.

GHCi requirements

Due to the current lack of an executable that one can click and simply make the program work, the user has to install some packages to make it work. Firstly the user is required to have GHCi and the Stack tool installed on their computer. Secondly, the user needs to have the required packages installed on their computer, however simply downloading the project through GitHub should also download the packages. This means that you only have to download the packages if you are rewriting the program from scratch. Lastly, an internet connection is required in order for Steam’s web API to be able to gather data.

Running the program

In order to run the program, all the user is required to do is to compile the ‘Main.hs’ file in Stack GHCi by simply typing ‘stack ghci’. Do recognize that issues may occur for various reasons if you type ‘stack ghci’ which will in some cases resolve themselves if you instead only compile the main function through stack by typing ‘Stack ghci ./app/Main.hs’. Do also make sure that you are in the right directory when you are running ‘Stack ghci’. Once all the files are compiled correctly, the user only needs to run the function ‘main’ by simply typing ‘main’ in the interactive environment. The IO action will then prompt the user to ‘enter a valid Steam64 ID to a PUBLIC Steam profile’, once a valid ID has been entered, the program will ask the user if they want to compare the following lists or if they want to continue adding users to the comparison list. If the user wants to add more Steam users to the comparison list, they are prompted to enter anything other than ‘True’ in the terminal. The program will then ask for a valid Steam64 ID once again and repeat this process until the user enters ‘True’.

When ‘True’ has been entered, the program prints a list of games that all Steam users in the comparison list have in common and additionally, for how long each user has played the game.

For example, entering the following IDs:
76561198046588035
76561198068497293
76561198072508175
Returns this list of games and playtimes:
The common games for
Glaus, God, lådan are:

Among Us: Glaus: 17 hours, God: 14 hours, lådan: 21 hours
Counter-Strike: Global Offensive: Glaus: 821 hours, God: 1524 hours, lådan: 532 hours
Garry's Mod: Glaus: 49 hours, God: 19 hours, lådan: 1146 hours
Left 4 Dead 2: Glaus: 1 hours, God: 0 hours, lådan: 32 hours
PAYDAY 2: Glaus: 0 hours, God: 10 hours, lådan: 177 hours
Portal: Glaus: 55 hours, God: 0 hours, lådan: 8 hours
Portal 2: Glaus: 44 hours, God: 4 hours, lådan: 100 hours
Risk of Rain 2: Glaus: 15 hours, God: 32 hours, lådan: 13 hours
Team Fortress Classic: Glaus: 0 hours, God: 0 hours, lådan: 0 hours
Total War: SHOGUN 2: Glaus: 0 hours, God: 0 hours, lådan: 0 hours

---

3. Creating the program

Tools

The tools we used for this project were few, not including all the essential ones like a computer or internet. We used a text editor with a terminal, for example Visual Studio Code or Emacs with Haskell mode, to write and run code.

Additionally, the Haskell tool Stack was used for easily installing and running packages in Haskell. It was more convenient than Cabal, since Cabal projects are not as easy to work with on different computers, which was one major requirement for this project. Additionally, there were several useful libraries and packages that were included in the Stack repository, which facilitated the package and library management.

Lastly, GitHub was used to share files and keep everyone in the group up to date. GitHub is easy to use and allows its users to roll back to previous versions of a program if they accidentally managed to mess something up. GitHub is convenient in group projects because it allows everyone in the group to work at the same time or different times without having to worry about having the latest version of the project. Simply put, it makes file sharing much easier.

Packages and libraries

Aeson - The program utilizes the Aeson package in order to gather the data from the web API and to then parse said data into the correct data types. The parsing ensures that, for instance, merely the game titles of the user's owned games are extracted, since that might be the relevant data.

Data.List - Since the owned games would be in a list, the list library was imported in order to make use of its various features, such as intersect, which takes two lists and returns all the intersections within the lists, basically all the common elements.

Network.HTTP.Conduit - This library specifically, the simpleHttp function, reads the API file from the internet and saves it as a string in the program, which other libraries can parse and gather information from. [4]

Data.Char - In order to make the program more user friendly, Data.Char was used to get a hold of its toUpper function that would dismiss the matter of caring about upper- or lowercase letters in the user's input.

---

[4] "Network.HTTP.Conduit", in *Hackage.haskell.org*, , 2021, <https://hackage.haskell.org/package/http-conduit-2.3.8/docs/Network-HTTP-Conduit.html> [accessed 3 March 2021].

System.IO - We wanted to create a program that would run recursively and also have user interactions, and because we were somewhat familiar with IO functions from past labs that we have had in this course, we figured that this option would best fit our needs, as an IO function make programming in Haskell more intelligible. It allows us to create an interactable environment which makes things more comprehensible and easy to use.

GHC.IO.Encoding - This library is used in order to use the setLocaleEncoding function. It is used to set the character encoding when writing the information to a txt-file to the UTF-8 one. This allows the program to write certain special characters such as trademark signs or nordic characters.

Data.Text - Due to the fact that certain game titles and usernames might include certain characters, such as ® or ™, the Text data type was imported in order to properly handle these characters. When the JSON-file is parsed, the information that is represented by letters is then decoded and converted into the Text data type in the program.

Data.ByteString.Lazy.Char8 - When retrieving the JSON-file from the internet, it is read in the ByteString format. Therefore this library was imported in order to properly write functioning custom data types and to properly convert the values in the JSON-file to data types that Haskell can understand.

Structure

Steam's Web API and Security

Steam's Web API can be used to retrieve information on Steam users via an HTTP request,[5] for example, Steam has an API that returns the latest news for a game, if you provide it with the app ID of the game you want news from (GetNewsForApp). Steam also has an API that returns the global achievements overview of a specific game in percentages, if you provide it with the ID of the game (GetGlobalAchievementPercentagesForApp). In order to specify arguments to an API all you have to do is to add arguments to its URL, which is fairly simple. Basically, this means that you would have to add something along the lines of

[5] "Web API Overview (Steamworks Documentation)", in *Partner.steamgames.com*, 2021, <https://partner.steamgames.com/doc/webapi_overview> [accessed 3 March 2021].

‘&steamid=’ followed by the ID of the user or the game you want to check for in the API. The program that we created utilizes, for the most part, the public web API that allows us to gather information on the games owned by the users we provide the Steam64 ID of (GetOwnedGames). We also make use of another Steam API to gather the usernames of the people we are comparing games for, in order to make it somewhat more comprehensible (GetPlayerSummaries). An API key is required to make the APIs work and everyone with a Steam account has the right to an individual key, which means that the key will remain linked to said account. The API key is applied in the URL just like the arguments, however, it is important to know that the key should not be shared with others, since scammers have been known to use these keys to steal items from people’s accounts.[6]

Parsing

To make use of the API’s data, the JSON-file has to be parsed. Parsing means interpreting the JSON-files information in a way that it can later be used in a specific programming language, which in this case is Haskell. The JSON-file may contain an unfamiliar data type, which means that the program would need to know the structure of the JSON-file in order to be able to pick out, for instance, a specific value such as the title of a game, in order to obtain usable information.

Data Types

In order to properly parse the JSON-file from the API, we had to write our own data types that resemble the structure of the file. For the list of games, we had to parse the GetOwnedGames interface of the API.[7] Since the file contained a list of games, we began with declaring a data type called “Game”. It contains information such as the game's name, for how long the user has played it, etcetera.

```
data Game = Game
      { appidGame :: Int
      , nameGame :: Text
      , playtimeForeverGame :: Int
```

[6] "What you should know about "Steam web API" scam. How not to lost [sic] your skins! :: Loot.farm", in *Steamcommunity.com*, 2021, <https://steamcommunity.com/groups/lootfarm/discussions/0/1608274347722600472/> [accessed 3 March 2021].

[7] "Steam Web API - Valve Developer Community", in *Developer.valvesoftware.com*, 2021, <https://developer.valvesoftware.com/wiki/Steam_Web_API#GetOwnedGames_.28v0001.29> [accessed 3 March 2021].

```
      , imgIconURLGame :: Text
      , imgLogoURLGame :: Text
      , playtimeWindowsForeverGame :: Maybe Int
      , playtimeMACForeverGame :: Maybe Int
      , playtimeLinuxForeverGame :: Maybe Int
      , hasCommunityVisibleStatsGame :: Maybe Bool
      , playtimeMACForeveRGame :: Maybe Int
      , playtime2WeeksGame :: Maybe Int
      } deriving (Show)
```

The JSON-file stores each game inside of another data type called “games”, which also contains the amount of games inside a variable called game_count. This type was named “UserGamesResponse”.

```
data UserGamesResponse = UserGamesResponse
      { gameCount :: Int
      , listOfGames :: [Game]
      } deriving (Show)
```

Lastly, this data type is used in a third one, GamesResponse, that contains one UserGamesResponse.

```
data GamesResponse = GamesResponse
      { userGamesResponse :: UserGamesResponse
      } deriving (Show)
```

In order to parse information about the user’s profile, we had to write separate data types for the GetPlayerSummaries interface. Although the method for obtaining the relevant information is the same as with the GetOwnedGames one.

Instances

Another requirement for properly parsing a JSON-file are the FromJSON instance declarations. These instances are necessary for the Aeson function eitherDecode to operate properly. The instances tell the decode function which value in the JSON-file is supposed to be associated with which value in its corresponding data type. The instance declaration for the Game data type looked like this.

```
instance FromJSON Game where
      parseJSON (Object v) = Game
      <$> v .: "appid"
```

```
    <*> v .: "name"
    <*> v .: "playtime_forever"
    <*> v .: "img_icon_url"
    <*> v .: "img_logo_url"
    <*> v .:? "playtime_windows_forever"
    <*> v .:? "playtime_mac_forever"
    <*> v .:? "playtime_linux_forever"
    <*> v .:? "has_community_visible_stats"
    <*> v .:? "playtime_mac_forever"
    <*> v .:? "playtime_2weeks"
```

Similar to the data types, these declarations are required to have the same structure as the JSON-file and also the same data types as in its corresponding custom data type. For example, '<*> v .: "name"' declares that the name of a game is of a *value*. Whilst, '<*> v .:? "has_community_visible_stats"' declares that has_community_visible_stats is of a *Maybe value*. This is due to the fact that if the user has set, for example their community stats to private, then the API is unable to retrieve that data. If that is the case, then its value becomes *Nothing* and the parsing can continue. Otherwise, the parsing would fail and an error would be returned instead, regardless of which value we are looking to extract. With a correctly written data type and FromJSON instance, the decode function can easily convert the value in the JSON-file to a Haskell comprehensible value, such as the name of a game, which will be converted into the Text data type.

Functions

fromJSON

```
gamesFromJSON :: String -> IO [String]
aliasFromJSON :: String -> IO [String]
playtimeFromJSON :: String -> IO [(String, String)]
```

There is one function for each piece of information that is retrieved from the API, for instance the gamesFromJSON function retrieves a list of a user's owned games, whilst the aliasFromJSON function retrieves the user's name. Lastly, playtimeFromJSON returns a list of games, but also with the user's playtime attached. These functions, nonetheless, operate in the same way. First, the function retrieves the JSON-file from the internet with the simpleHttp function from the Network.HTTP.Conduit library. It is then stored as a custom data type, such as GamesResponse in order to be decodable. Afterwards, the function decodes

the JSON data and returns an error if it fails. If it succeeds then the *map* function is used to tell the program specifically which piece of data should be returned. In the case of the games list, nameGame is mapped onto the list. Then this list is processed through the *unpack* function in order to convert the elements from the Text data type, to strings.

mapAliastoPlaytime

```
mapAliastoPlaytime :: [(String, String)] -> [String] -> [(String, String)]
```

In order to associate a game with two other elements, namely a username and that user's playtime on the game, the mapAliastoPlaytime function is used. The list returned from the playtimeFromJSON function contains a tuple with the titles of the game and for how long each title has been played. Both of these are retrieved from the GetOwnedGames interface. The mapAliastoPlaytime function takes this list and maps the user's Steam username to the playtime in the function. The username is returned from the GetPlayerSummaries interface and then put into the list. For example, a list that initially contains elements such as ("Videogame", "100 hours") becomes ("Videogame", "Username: 100 hours").

intersectPlayers

```
intersectPlayers :: [[(String, String)]] -> [(String, String)]
```

Once the list of games has been extracted, that list is added to another list, that acts as an accumulator, as the program runs recursively to gather as many Steam64 IDs as the user wants to compare. Once the user has gathered the Steam64 IDs and their games have been added to the accumulator, the user can type 'True' in order to signify that they have finished gathering Steam64 IDs. The program will then take this accumulator and send it to a recursive function that basically looks for intersections between these lists and adds all these points of intersection to a new list that then gets returned. This new list is the list that contains all the common games of the Steam64 ID.

Main and inputLoop

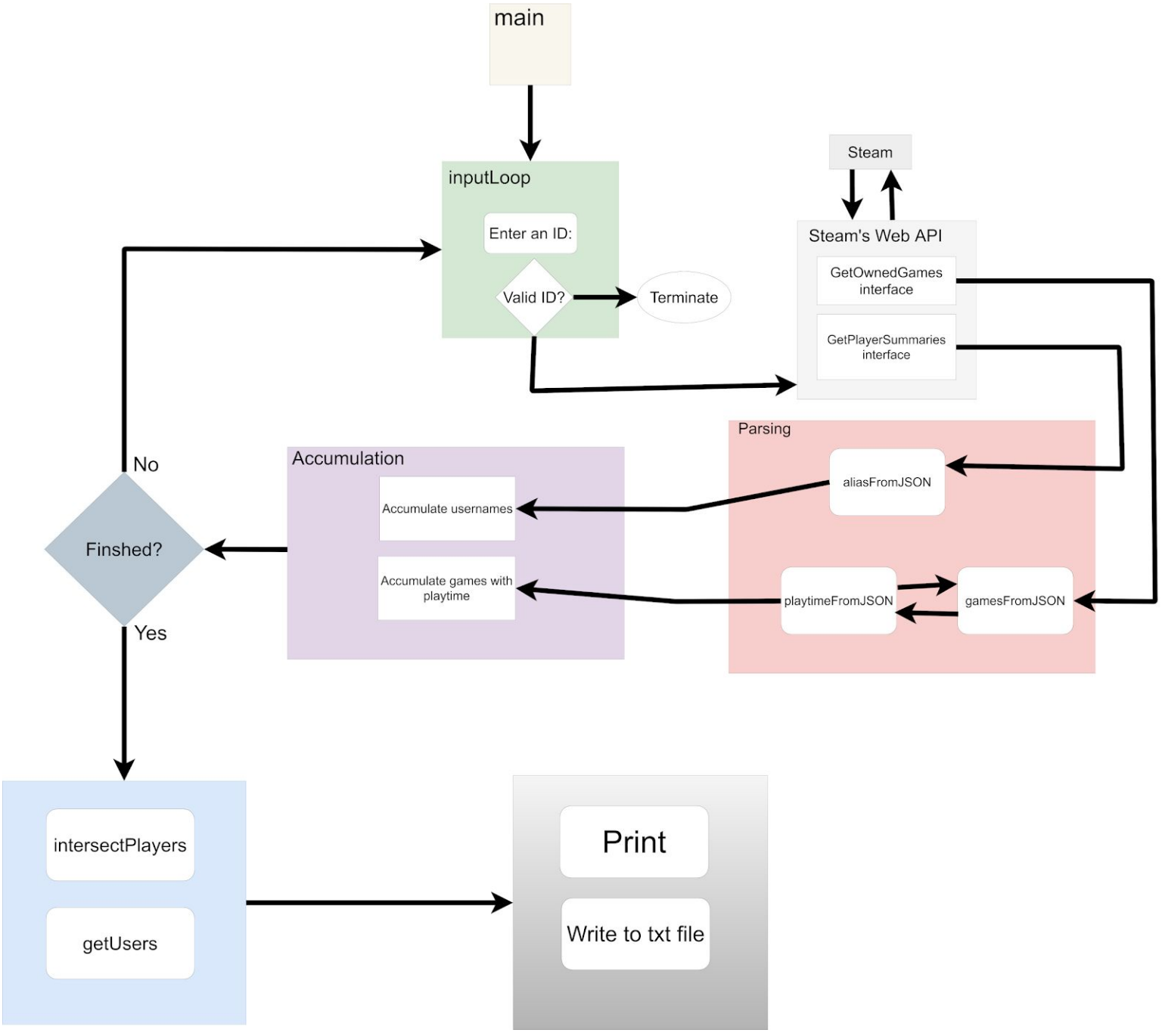
```
main :: IO ()
inputLoop :: [[String]] -> [[String]] -> IO ()
```

The main function, main, is an IO function that starts out by printing "Welcome to Common Steam Games!". Then it calls upon the IO function called inputLoop, which prints "Please enter a valid Steam64 to a PUBLIC Steam profile ... " then reads the user's input. Once a Steam64 ID has been input. It will print "If you want to compare the following lists, type 'True' ". However, if the user types True with only one list, it will return that list as it checks for intersections with itself. Simply put, this means that it checks for the games that it has in common with itself, namely returning a list of all the games owned. If the user however decides to add more than one Steam64 by typing anything other than 'True' it will recursively call itself and have the list of games as an argument. This list will keep getting longer the more users are put in, and exists for the sole reason of being able to remember the past Steam64 IDs that have been input. The inputLoop will continue to call itself until the user types 'True' after a valid ID has been submitted. Thereafter, the function will print out the commonly owned games in the interactive environment, as well as writing them to a txt-file.

Control flow

The program's control flow can be illustrated in this flowchart. The main function prints out a welcome message and then calls upon the inputLoop, which prompts the user to input a valid Steam64 ID. Once the user has done so, valid API URLs are created for the GetPlayerSummaries and the GetOwnedGames interfaces. These URLs are then used in the gamesFromJSON function, which retrieves the list of owned games, and the aliasFromJSON, which returns the username. The result from gamesFromJSON is inserted into playtimeFromJSON, the function adds the playtime for each game the user owns. The list of games with playtimes and the username are then inserted into their own respective accumulators. If the user wishes to enter more Steam64 IDs, then the process is repeated and the function will continue to accumulate game lists, playtimes and usernames. If the user enters "True", as in they are finished with entering IDs, then the two accumulatros will be put through their respective functions. The game list will be put into the mapAliastoPlaytime and intersectPlayers functions, where only the games that occur in all game lists are returned together with the usernames and for how long each user has played each game. The list of usernames is put into the minor function called getUsers, which simply puts all the usernames into a single string, which can then easily be printed. Lastly, the list of games and for how

long each user has played each game is printed along with the usernames. This information is additionally, written to a txt-file called CommonGames.txt.



4. Discussion

Further features

There are a myriad of features and adjustments that can be done in order to improve this program. However, we find the biggest issue to be the fact that we needed each user to not only install GHCi but also Stack and all the other libraries and packages to make it work. Although this is not a problem once you have it downloaded, it messes with our idea of creating a simple product that one can download and run instantly. The ideal scenario would be if it was available as, for example, an exe-file without any package requirements. Another feature that would make our program even more user friendly would be if the program could retrieve the Steam64 ID automatically, meaning that all the user would need to do is to enter a Steam profile's URL. In that case, the user would not need to research how to find the Steam64 ID themselves. Additionally, Since Steam's API does in fact allow a user's friends list to be returned, one possible further feature could be the ability to look through one's list of friends and choose them via username instead of URL for comparison.

Optimization and other improvements

There have been some issues when it comes to printing special characters in the terminal or the interactive environment, be it the aforementioned trademark signs or nordic characters. Although there are some fixes for this[8], it is nonetheless an issue that persists. However, writing the information to a txt-file is working without problems. It could also be resource friendly to temporarily save the data acquired from the API locally. This way, it could use that information instead of requesting it again from the web. This could certainly be useful as the user will most likely almost always include themselves in the comparison, and also some of the same friends as well. If the API were to be temporarily down due to server issues, then this locally saved data could ensure that the program still works for the saved users.

[8] "How can I set my GHCi prompt to a lambda character on Windows?", in *Stack Overflow*, 2021, <https://stackoverflow.com/questions/25373116/how-can-i-set-my-ghci-prompt-to-a-lambda-character-on-windows> [accessed 3 March 2021].

5. Conclusion

Overall, the group managed to create a program that did exactly what the goal stated. The cooperation and communication in the group worked quite well. We regularly updated each other on our progress in the group chat and often had group calls where we also discussed these matters further and updated our to-do list. We managed to successfully parse data from two different interfaces of Steam's API and additionally combine them for the username and playtime lines. The project has undoubtedly been educational as it has given each group member the task to learn new things when it comes to programming in Haskell and programming in general, such as JSON parsing, which will be useful for future projects.

6. References

1. "How can I set my GHCi prompt to a lambda character on Windows?". in , 2021, <https://stackoverflow.com/questions/25373116/how-can-i-set-my-ghci-prompt-to-a-lambda-character-on-windows> [accessed 3 March 2021].

2. "Network.HTTP.Conduit". in , 2021, <https://hackage.haskell.org/package/http-conduit-2.3.8/docs/Network-HTTP-Conduit.html> [accessed 3 March 2021].

3. "Steam (service)". in , 2021, <https://en.wikipedia.org/wiki/Steam_(service)#User_interface> [accessed 3 March 2021].

4. "Steam Web API - Valve Developer Community". in , 2021, <https://developer.valvesoftware.com/wiki/Steam_Web_API#GetOwnedGames_.28v0001.29> [accessed 3 March 2021].

5. "Steam, The Ultimate Online Game Platform". in , 2021, <https://store.steampowered.com/about/> [accessed 3 March 2021].

6. Walker, T, "GDC 2002: Valve unveils Steam.". in *GameSpot*, 2021, <https://www.gamespot.com/articles/gdc-2002-valve-unveils-steam/1100-2857298/> [accessed 3 March 2021].

7. "Web API Overview (Steamworks Documentation)". in , 2021, <https://partner.steamgames.com/doc/webapi_overview> [accessed 3 March 2021].

8. "What you should know about "Steam web API" scam. How not to lost your skins! :: Loot.farm". in , , 2021, <https://steamcommunity.com/groups/lootfarm/discussions/0/1608274347722600472/> [accessed 3 March 2021].

Code references

9. https://stackoverflow.com/questions/29941866/
(Accessed 14 Feb)

10. https://hackage.haskell.org/package/http-conduit-2.3.7.4/docs/Network-HTTP-Conduit.html (Accessed 15 Feb)

11. https://www.schoolofhaskell.com/school/starting-with-haskell/libraries-and-frameworks/ (Accessed 15 Feb)

12. https://artyom.me/aeson (Accessed 18 Feb)

13. https://hackage.haskell.org/package/bytestring-0.11.1.0/docs/Data-ByteString-Char8.html (Accessed 22 Feb)

14. https://jsonformatter.org/json-to-haskell (Accessed 22 Feb)

15. https://stackoverflow.com/questions/25373116/ (Accessed 1 March)

16. https://hackage.haskell.org/ (In general when it comes to finding functions and documentation. not accessed on a particular date, but almost every day)