Object Oriented Analysis & Design

Documentation Steam Stats

Team Members:

-Bas van der Linden, #500735218

-Justin Smid, #500780520

-Mike Traa, #500782355

Document version: 4

Class: IS205

Date: 15 Oktober, 2018

Table of contents

1. Introduction

1.1 Intro

This file holds all the documentation regarding our Steam stats application, an app that allows you to look up stats of any Steam account. We've chosen to split our documentation up into 2 parts; a general section which has information about the doings of the application as a whole, and an individual section where each of us wrote about our own user stories.

1.2 Vision

1.2.1 Goal

Our goal is to give people the ability to look up statistics from their or their friends' steam profiles so they can compare their stats or brag about how many hours they have played a game. We also want gamers to be able to share their account details via Facebook or Twitter.

1.2.2 Target group

Our target group is: gamers that use steam and want to learn more about their account. It's for gamers who want to check their most played games or many other different stats.

1.2.3 App overview

Our application will show statistics of a steam profile based on the name/id given by the user. When the name or id is correct, the app will show all kinds of statistics for that account. The user can see their most played games, they can see how many hours the have played their games for. We will add together all the played hours so the user can see how many hours/days the have played for in total. The user will also be able to share their account details to Facebook or Twitter. They can also use a link to view their steam account on the steam website.

2. UML diagrams

This section shows our UML diagrams which we use to guide the design of the application. These UML diagrams allow us to represent abstract ideas concretely in the form of diagrams.

2.1 Domain model

Our domain model describes our data, entities and behavior and how all these interact with each other

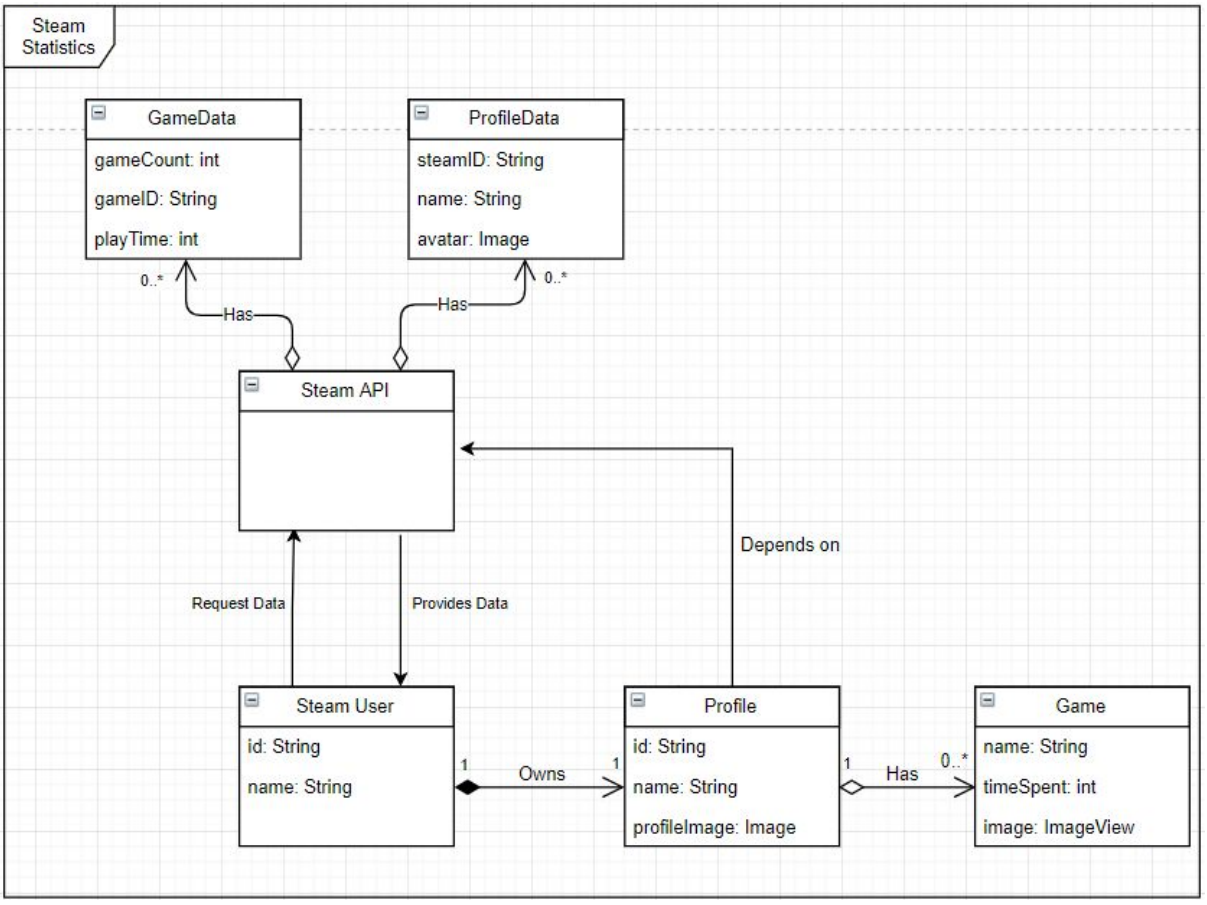


Diagram 1: Domain model

2.2 Use Case Diagram

Our use case diagram represents a user's interaction with our system. The diagram shows the relationship between this user and the users potential use cases of the system.

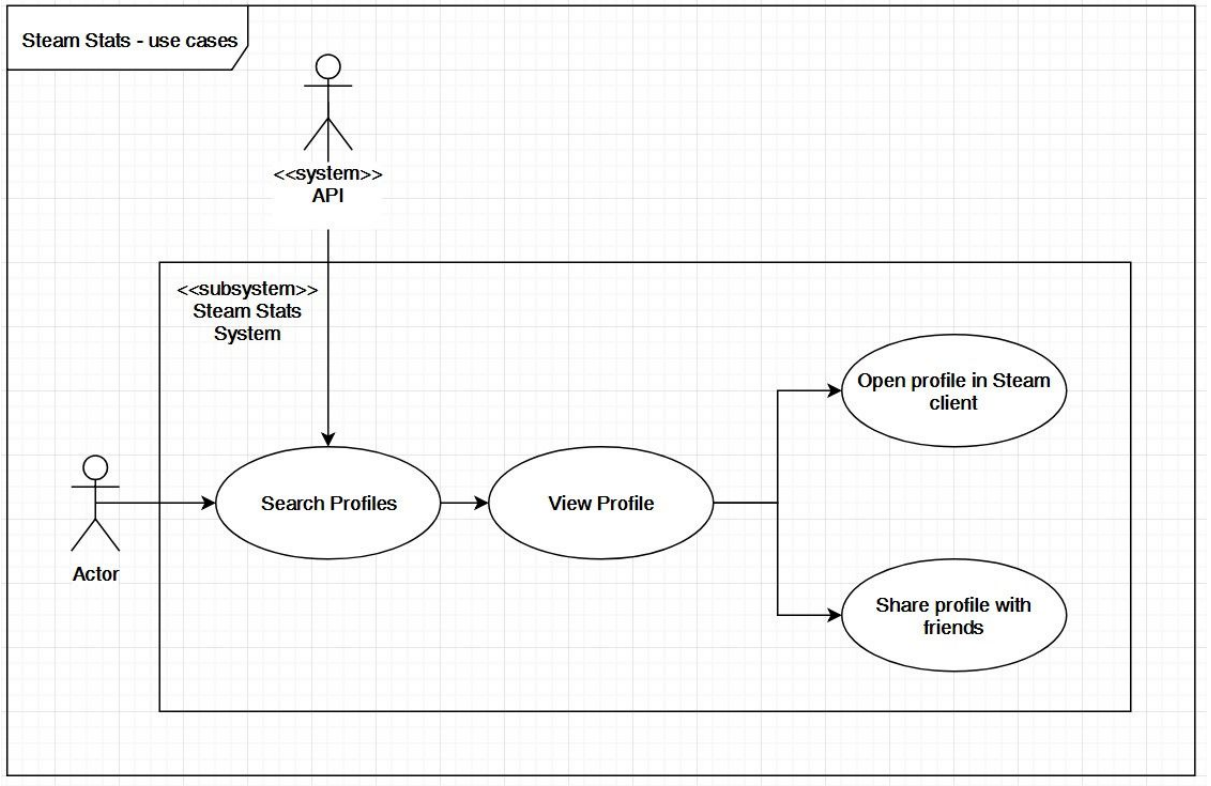


Diagram 2: Use case diagram

3. Patterns

In this section we explain how we made use of certain design patterns.

3.1 Model View Controller

Using MVC allows us to separate concerns in terms of programming. The benefit in doing this is that business logic and view logic are separated. If these different types of logic are mixed it creates dependency. A change in the view logic can very easily affect the business logic. One way we implement MVC is by using JavaFX's feature of using fxml files to create the view and using ordinary java classes to implement the business logic.

Model
The model part of MVC corresponds to the data-related logic of the application. The model represents the data being transferred between the View and Controller. In our application the data we request from the API fits this role.

View
The view part of MVC is used for all the user interface logic of the application. In our case the fxml files together represent the view.

Controller
The controller part of MVC acts as an interface between the Model and the View. When a change in the data occurs in the Model that change should be represented in the view. The controller processes our incoming requests, manipulates the data from the modal and interacts with the view to render the final result.

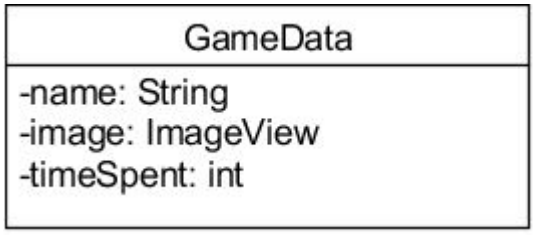
3.2 Strategy Pattern

Making use of the strategy pattern allows you to select an algorithm with which to handle things at runtime based on certain variables. In our application we've made use of this pattern when the user searches for a profile. The steam API from which we receive our profile data only takes profile IDs, but forcing users to figure out their steam ID in order to be able to search their profile in our application didn't seem like it was the most sensible thing so we made it compatible with the customizable steam name. To do so we look at the input given by the user and look at whether it's a series of numbers(through checking it against the regex: "^[0-9]*$"), if this check returns true we can directly make use of the API by sending it the given ID, but if the input is not just a series of numbers we first have to figure out the ID associated to the given profile's name.
This usage of the strategy pattern allows us to look at user input and alter what we send to the API request based on what input was given which allows us to be far more flexible with how we can look up profiles

4. Classes

In this section we go over the most important classes used in our application and talk about what and how they contribute.

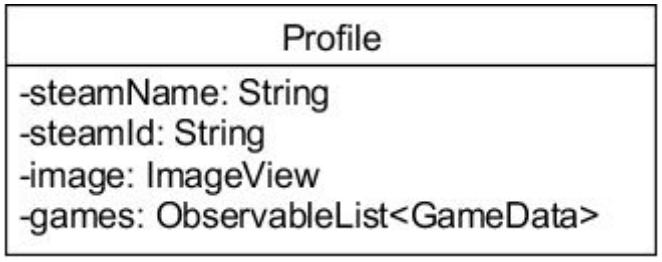
4.1 GameData



Class diagram 1: GameData class

This is a very simple class, it only holds a game's name, the amount of time the profile has spent on the game and it has the game's logo. This class was made so we could store the data we get from the steam API in an object so that we can show the game's information somewhere on the UI and perform certain actions on it such as adding up the time spent of all games to get the total time the user has spent on steam games.

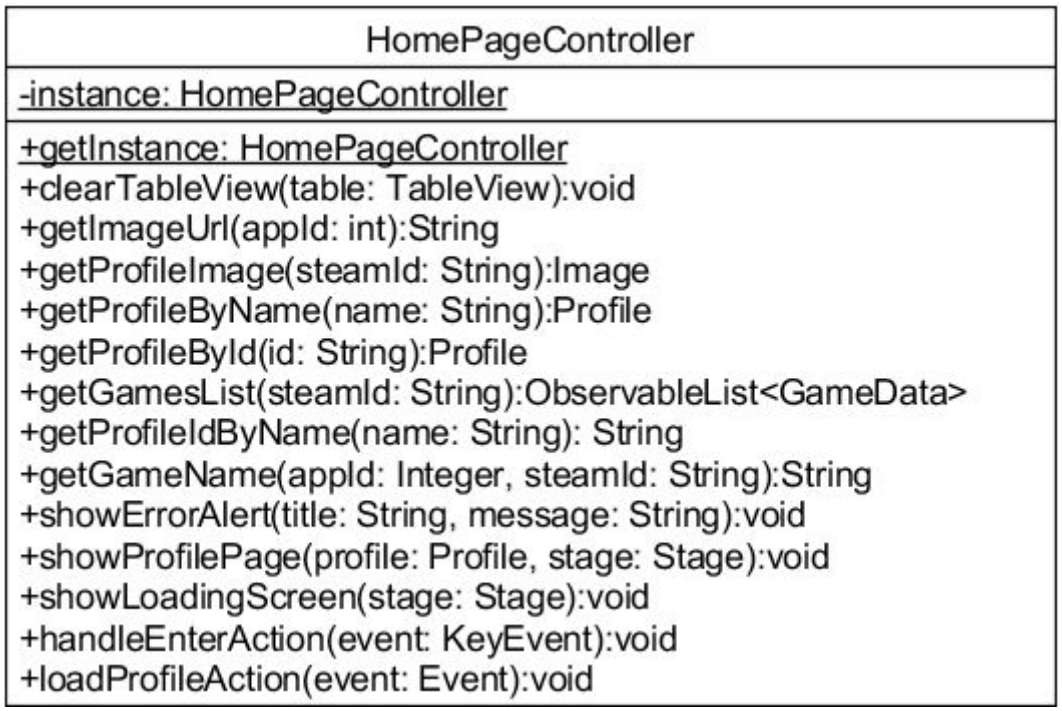
4.2 Profile



Class diagram 2: Profile class

This class holds the profile's steamName and steamId which are attributes used while retrieving a profile's data from the API. the games attribute is a list of GameData objects, this is what we use to display the games a user has played.

4.3 HomePageController



Class diagram 3: HomePageController class

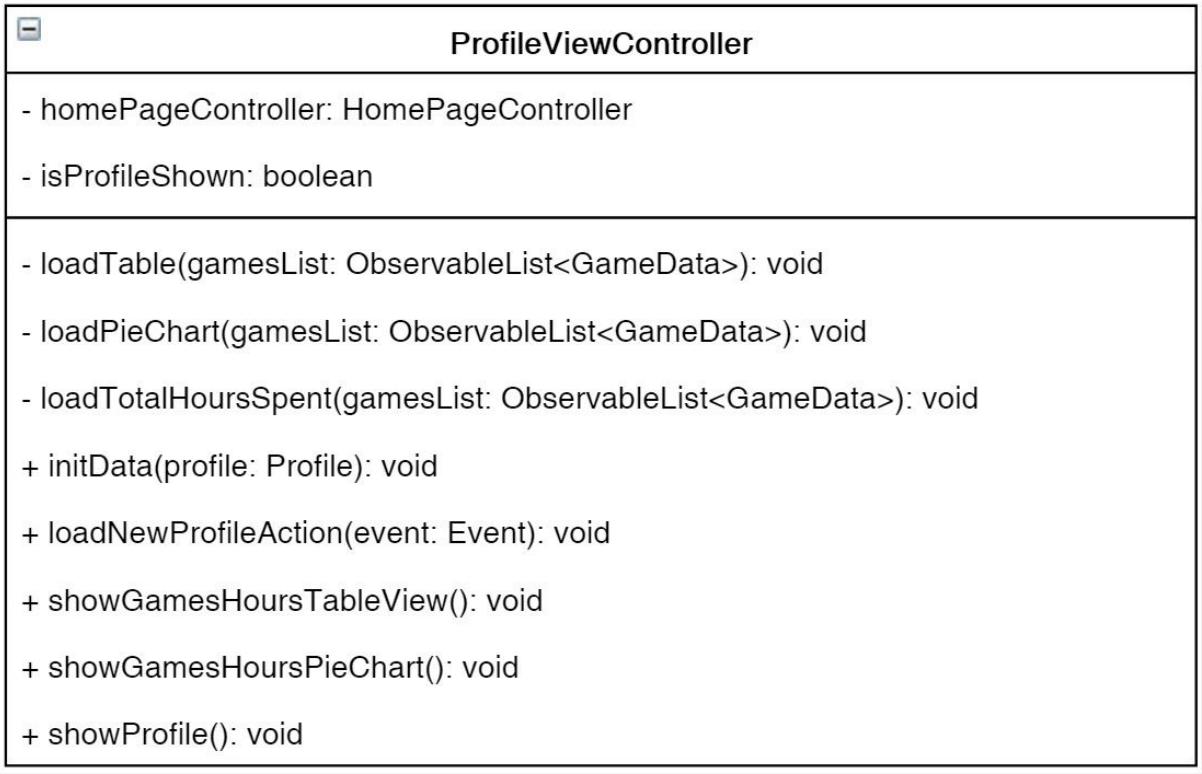
This is the class that holds most of the application's logic. The instance and getInstance() allow other controller classes to create instances of this controller to allow them access to this controller's functions, that prevents us from having to copy-paste the code that other controllers need from this controller to other controllers.
Most of the function names are pretty self-explanatory, I will explain a few that have somewhat interesting behavior.

getProfileIdByName(name: String) is the function that allows users to search for profiles based on their steam name as opposed to only being allowed to use their steam ID. The function takes the given name and requests the steamcommunity page of the given profile. This page happens to hold the profile's ID, so the function goes through the content until it stumbles upon "steamid" and then reads the following 17 numbers (steam IDs are always 17 numbers long) and returns these.

handleEnterAction(event: KeyEvent) is a simple but very convenient function, it allows users to search the given profile by just simply hitting the enter-key as opposed to having to press the "Load profiles" button.

loadProfileAction(event: Event) is the function that checks the user input and decides which function to call next based on what the user gave as input as is described in section 4.2 - Strategy Pattern.

4.4 ProfileViewController



Class diagram 4: ProfileViewController class

The homePageController attribute is an instance of a HomePageController, like mentioned before, this allows the controller to call methods from HomePageController without having to copy-paste any of the code.

The isProfileShown attribute is a boolean attribute used in the toggle control logic of the showProfile function.

loadTable(gamesList: ObservableList<GameData>) is the function that load the games into the pie chart, the result of which is shown in figure 7.

the showGamesHoursTableView() is the function that is bound to a button showing the tableview with the profile's statistics.

the showGamesHoursPieChart() is the function that is bound to a button showing the pie chart with the profile's statistics.

the showProfile() is the function that is bound to a button showing the webview of the steam profile page of the user.

initData(profile: Profile) is called when the application is done retrieving a profile's data and is ready to display it. The function sets the profile page's image, name and search bar to the correct values and then calls loadTable.

loadTable(gamesList: ObservableList<GameData>) is the function that loads the games into the table, the result of which is visible in figure 4.

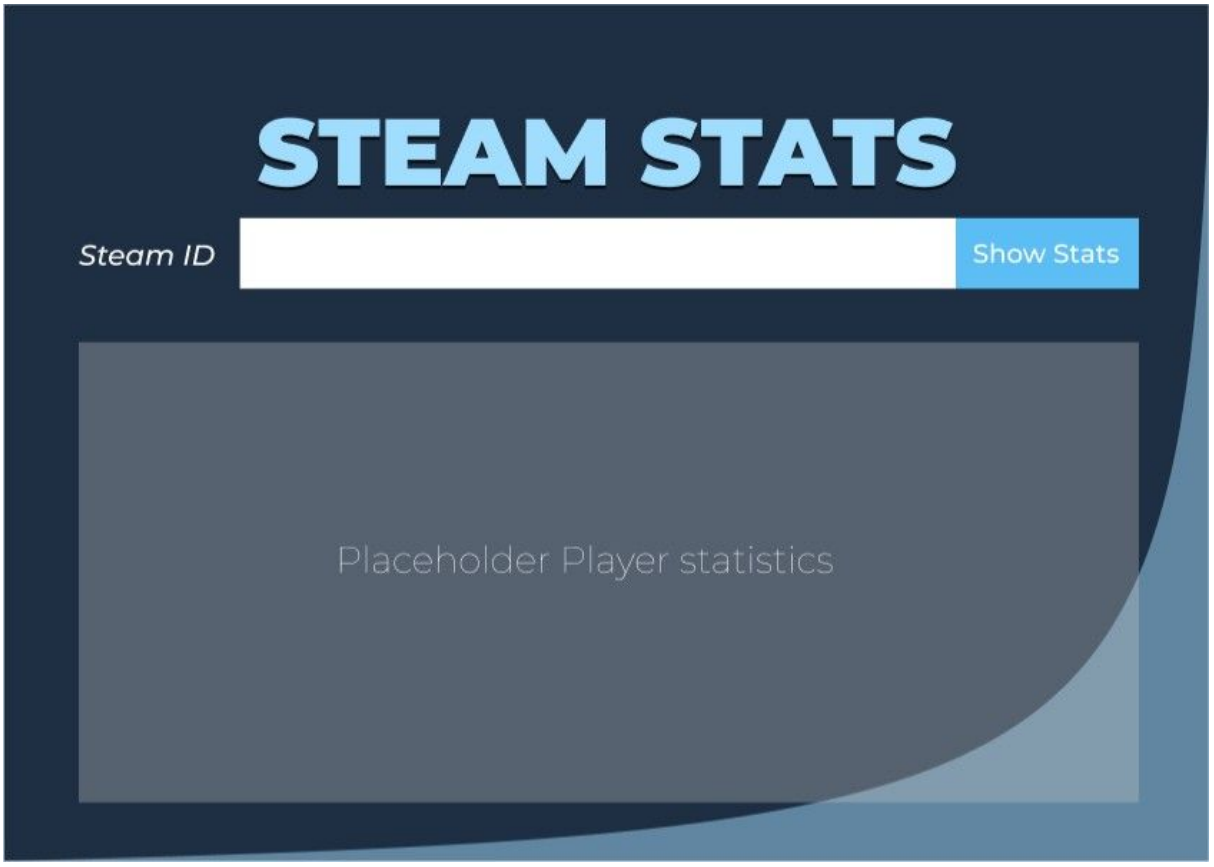
loadNewProfileAction(event: Event) serves the same purpose as the HomePageController's loadProfileAction function.

5. Design Iterations

5.1 Original user interface

In this section you can see the different stages our application design has gone through up to now.

The picture below shows the initial mockup which was later used as inspiration for the design of the actual application.



UI iteration figure 1: Original mockup of homepage

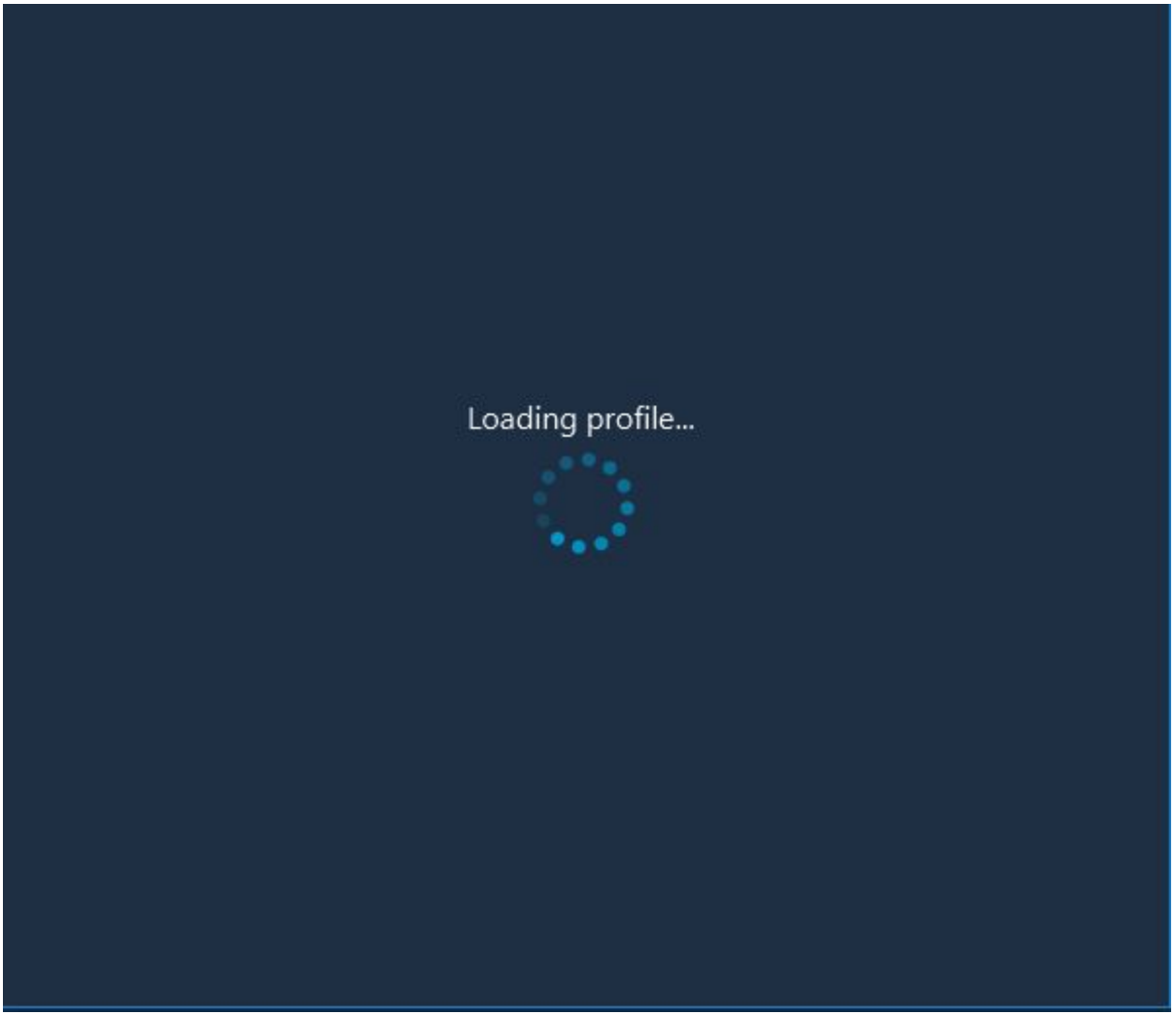
In the following picture you can see the first implementation of the profile page made with fxml. This design was part of the AppLayout branch.

UI iteration figure 2: First profile page

5.2 Current user interface

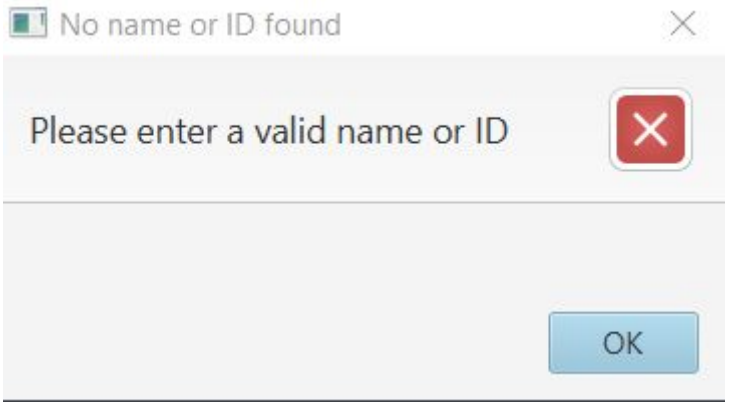
UI iteration figure 3: Final homepage

Figure 3 shows the starting page of the application, on it you can enter a steam name or ID and press enter or the “Load profile” button to search for the given profile.



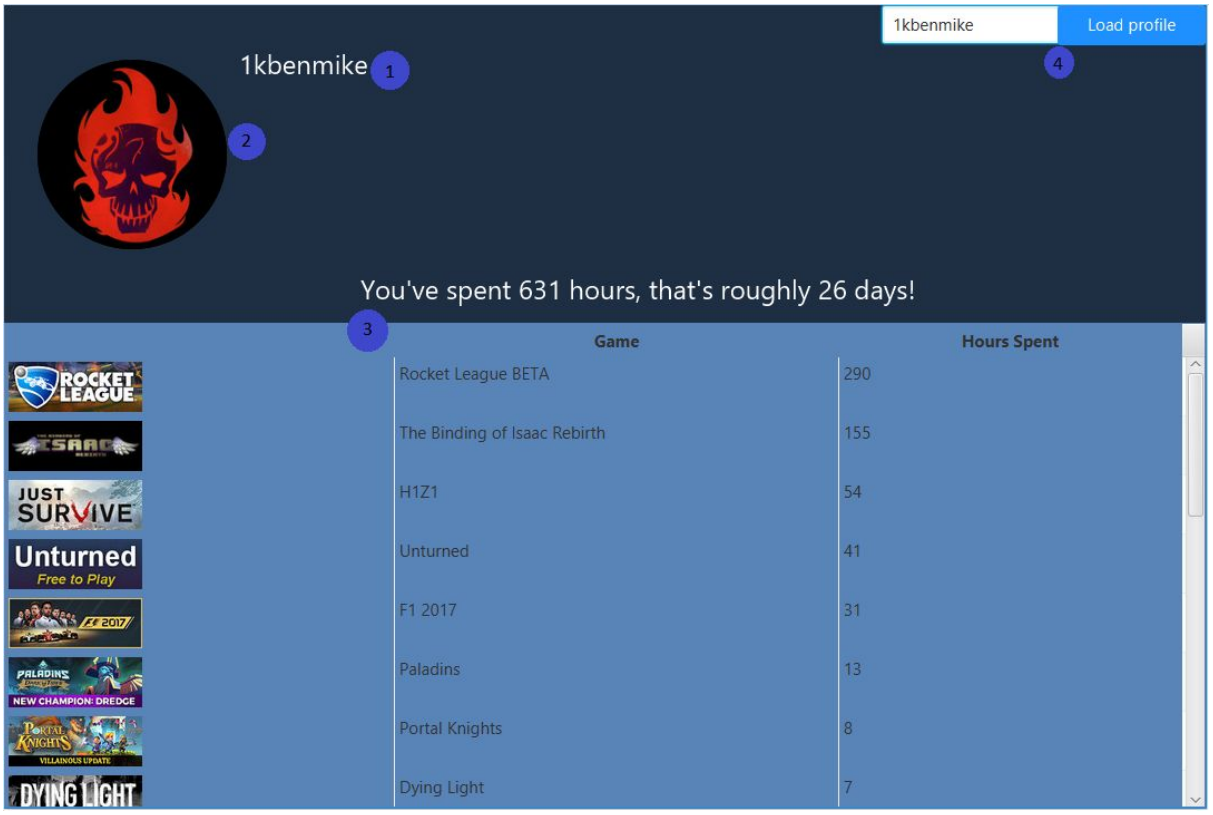
UI iteration figure 4: Loading screen

Figure 4 shows the loading screen which you'll be presented upon either pressing the "Load profile" button or hitting enter.



UI iteration figure 5: Error prompt

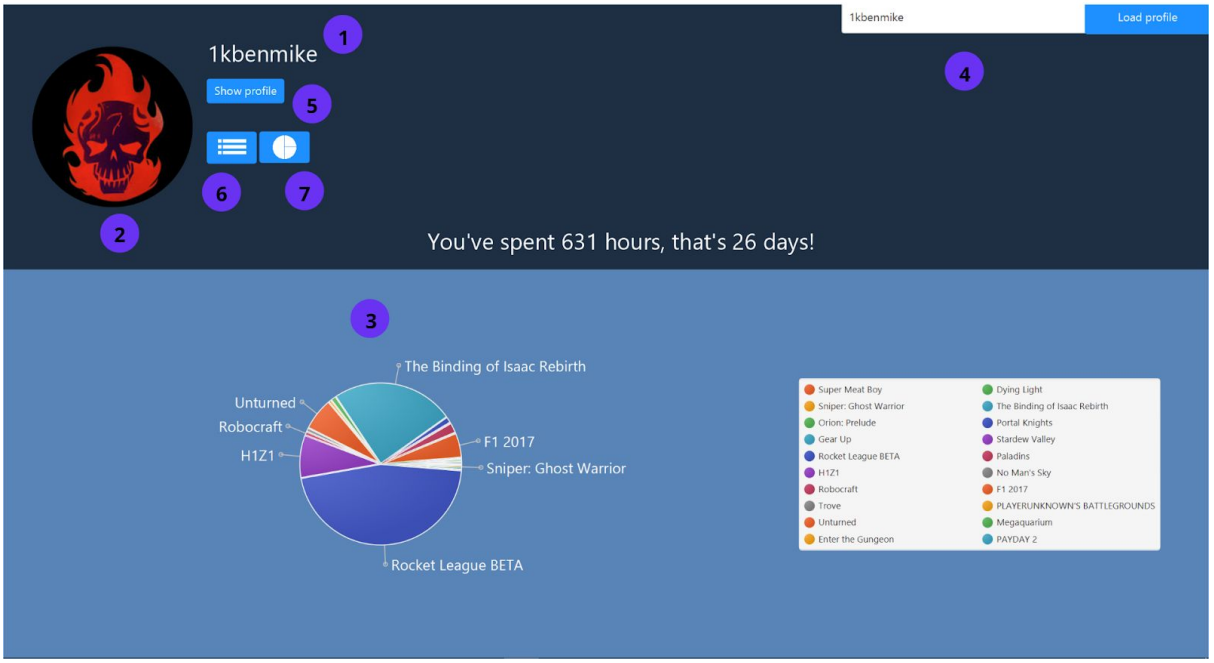
Figure 5 shows an error message which will be shown when you did not enter a name or ID when you pressed the button or hit enter.



UI iteration figure 6: Further improved profile page

Figure 6 shows the page you'll go to after the application is done loading a profile. On it are displayed the following items:

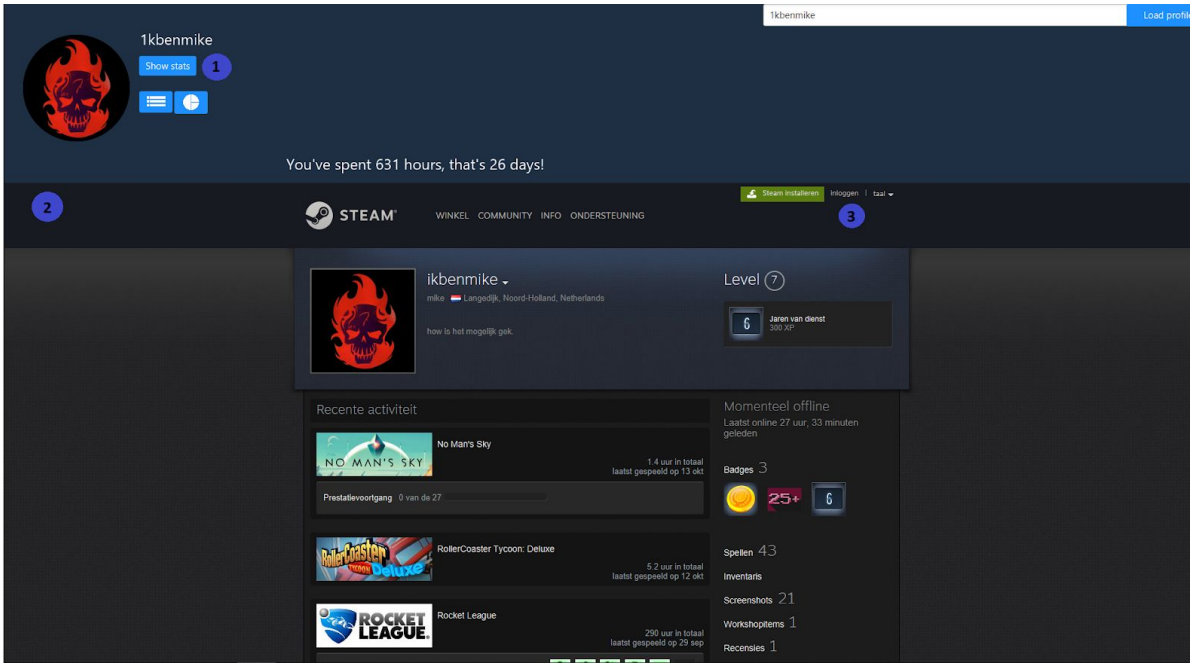
1. The name of the profile
2. The profile's image
3. A table showing the profile's games
4. Search bar & button with which you can search for another profile



UI iteration figure 7: Final profile page

Figure 7 shows the page you'll go to after the application is done loading a profile. On it are displayed the following items:

1. The name of the profile
2. The profile's image
3. A pie chart showing the profile's games
4. Search bar & button with which you can search for another profile
5. The 'show profile' button which shows the webview of a user's steam profile page
6. Button to show the listview with the user's profile statistics showing games and hours played
7. Button to show the piechart with the user's profile statistics showing games and hours played



UI iteration figure 8: Profile view from webview

Figure 8 shows the page you'll go to after you clicked the 'show profile' button from UI figure 7.5. On it are displayed the following items:

1. The 'Show profile' button which changed to 'Show stats' to return to the stats table
2. The webview with the account from the Steam website
3. The login button to log into Steam

6. Individual section

6.1 Justin Smid

6.1.1 User story 1

Trello: https://trello.com/b/kxodU6sK/scrumboard-steam-stats
User story: "As a user I would like to see which games on Steam I have played the most, so I can tell which game I liked the most"
Git commit: "Updated UI"
Git commit ID: `d4c4bb`

UML diagrams

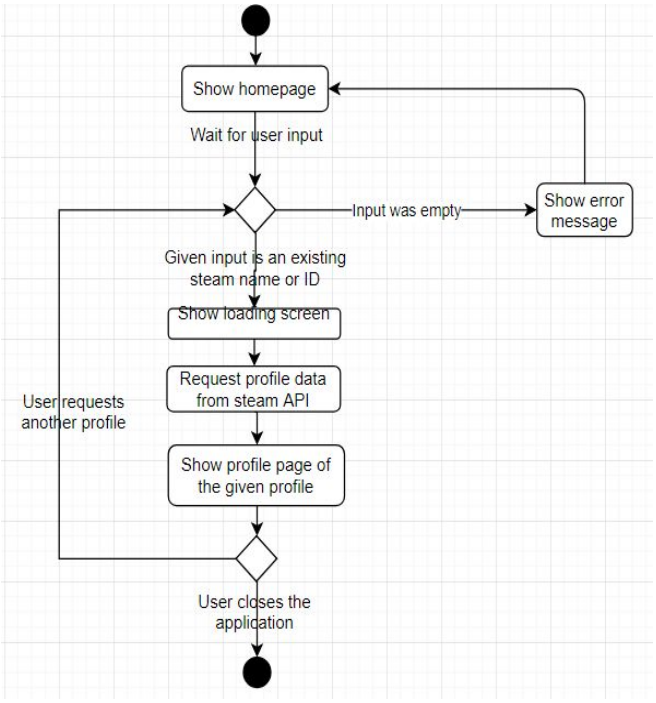


Diagram 3: Activity diagram showing what happens when a user requests a profile

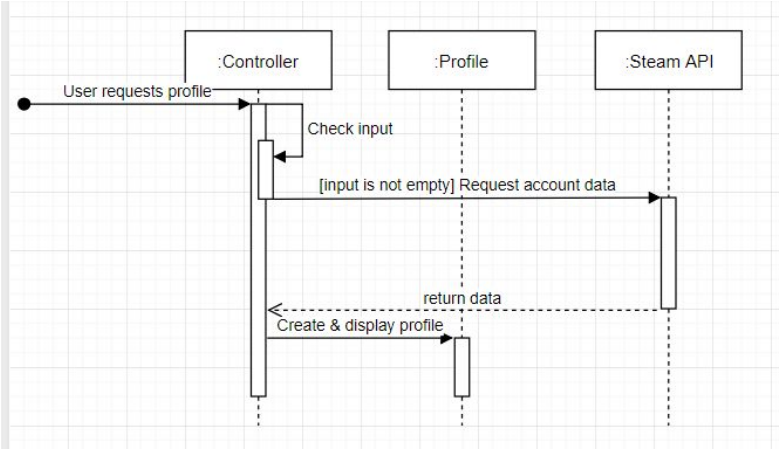


Diagram 4: Sequence diagram showing what happens when a user requests a profile

Diagrams 3 and 4, shown above, display what happens when a user requests a profile. When a user gives a profile name and requests to see said profile the controller will first check the input to see whether it's empty. If it is empty an error message will be displayed, otherwise the controller will display the loading screen and request the data associated to the given name from the steam API. Once the controller gets the data back from the API it will create a profile object which contains the data and then show the profile page of the requested account. When the profile page is shown the controller is idly waiting for the user to either request another profile or to shut down the application.

Acceptance criteria

The acceptance criteria for this user story are:

- User input(a steam ID/name) is given via a text field.

This can be seen in UI iteration figure 3

- A profile page with information about the profile's played games is shown after handling the user input.

This can be seen in UI iteration figure 6

- Displayed games are, by default, shown in descending order

Also shown in UI iteration figure 6

The user story states that a user wants to be able to request a profile and see the games that profile has played the most. To accomplish this we need to receive input from the user, send this input to the API so we can receive the data associated to the given account, retrieve that data and show it on the profile page. Because the user story specifically states that the user would like to see the games they've played the most we must make sure the games are displayed in descending order so that their most played game is at the top.

Patterns

The patterns that are used in this user story are described in section 3 of our report on page 6.

6.1.2 User story 2

Trello: https://trello.com/b/kxodU6sK/scrumboard-steam-stats
User story: "As a user I would like to see how much time in total I have spent playing games on Steam(in hours and in days), so I can see how much time I've spent on Steam games"
Git commit: "Now shows total time spent in days and in hours"
Git commit ID: cbf78c

Acceptance criteria

The acceptance criteria for this user story are:

- Profile page displays the total amount of time the account has spent on their games, both in hours and in days

This can be seen in UI iteration figure 6

This was a very small and simple user story, all that's needed for it is to display the amount of time the account has spent on their games. There aren't enough features introduced to warrant any UML diagrams as they'd not hold any useful or new information.

6.2 Bas van der Linden

6.2.1 User story

Trello: https://trello.com/b/kxodU6sK/scrumboard-steam-stats
User story: “As a user I would like to see how much time I have spent playing games on Steam in the form of a graph”
Git commit: “Merge branch ‘fixToggleButtons’”
Git commit ID: `c08d91f6`

UML diagrams

Both diagram 5 and 6 down below give insight into how the the chart displaying steam profile data works and how the user interacts with the system. I will talk more in depth about each diagram in the rest of this section.

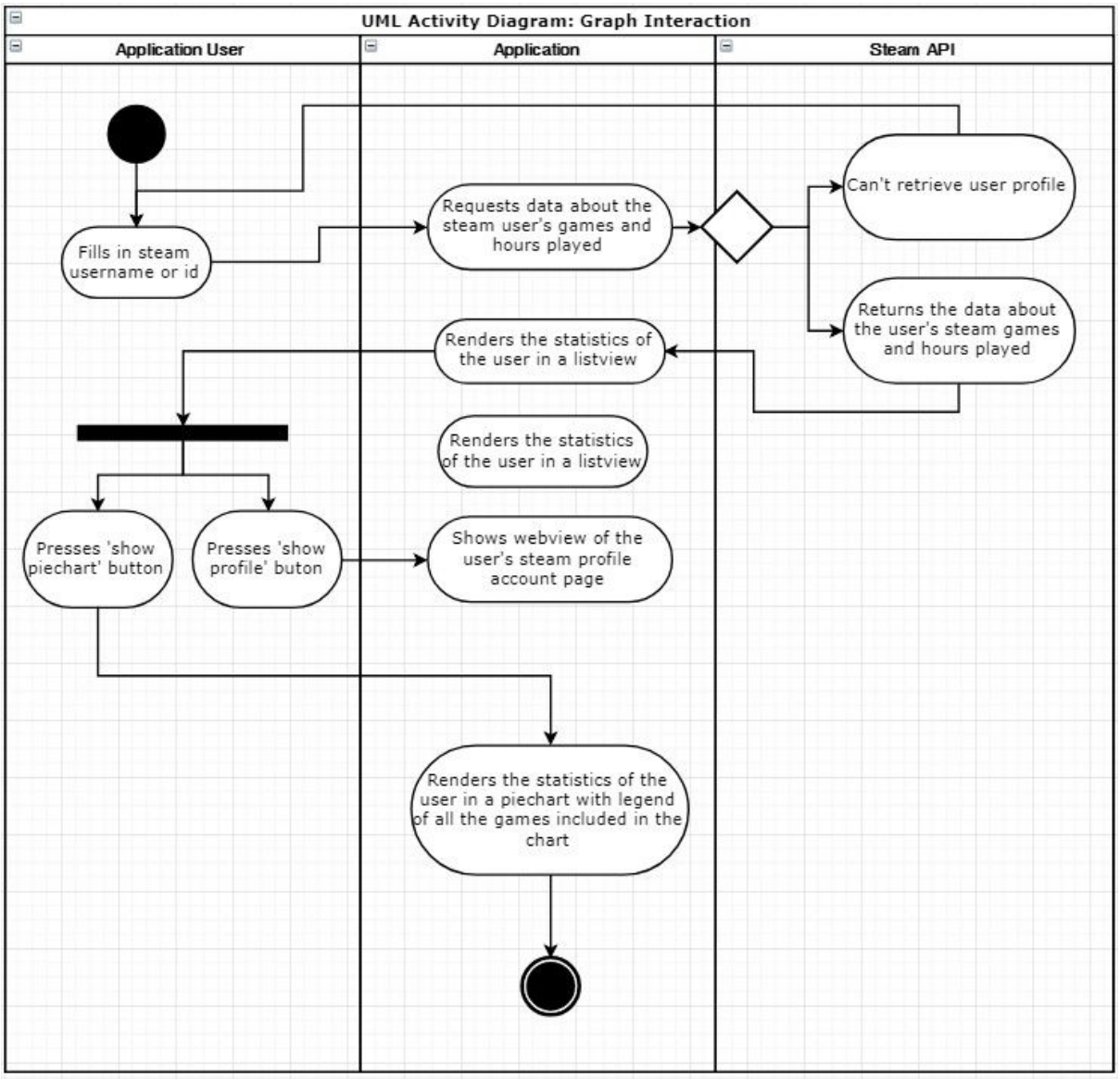


Diagram 5: Activity diagram showing how a user is able to see a graph displaying his/her steam statistics

In diagram 5 the different actors are each displayed as lanes. Every action on a lane corresponds to one of three actors: The application user, the application itself and the steam api. The diagram starts of with the user having to input his/her username or steam id. After which the application requests the steam user data from the api to be able to render the listview with the user's number of hours played per game. If the api can't retrieve the profile data from the api, it means that the profile doesn't exist, the user's profile is private or another unknown problem has occured. If this is the case we go back to the beginning of the diagram. If the profile data is retrieved correctly however the application is able to render the user statistics in a listview that shows per played game how many hours this user has played this game. After this the user can press two buttons: one button toggles a pie chart that shows the users played games and the amount of hours played per game, the other button shows a webview of the user's profile. if presses the button to display a pie chart, the data is

rendered in the application using the earlier requested data from the steam api. After this action the endpoint is reached.

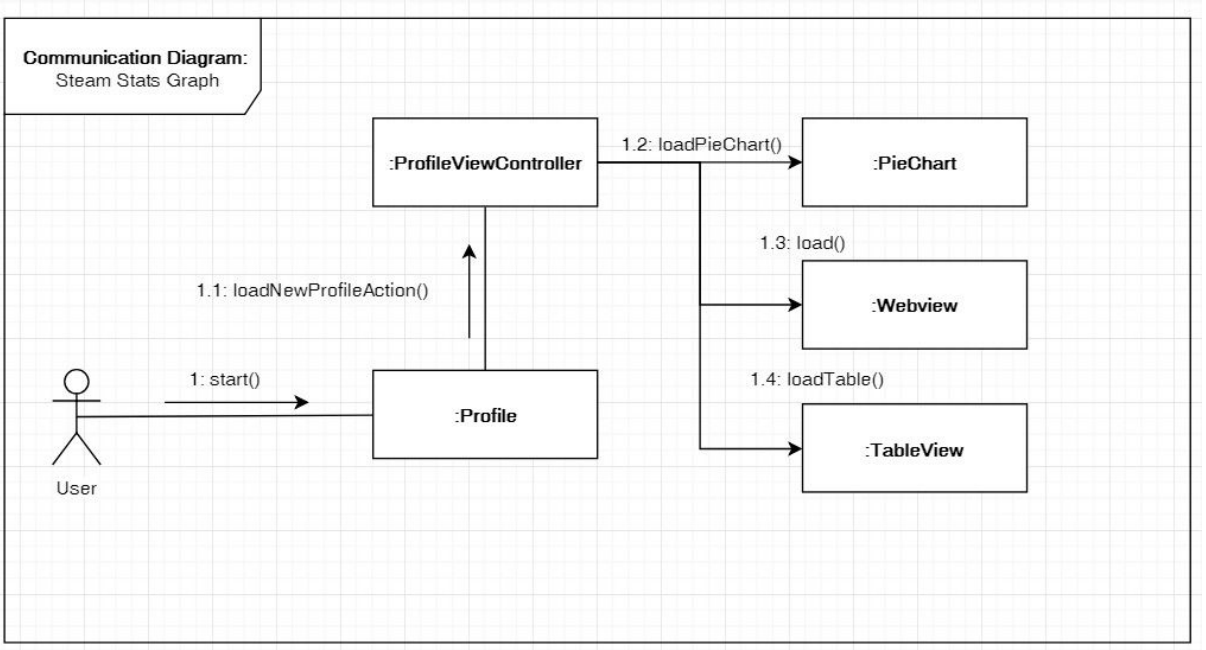


Diagram 6: Communication diagram showing how user can display profile information in different ways and the objects that play a part in this process

In diagram 6 shows the relation between the user starting the app and loading his/her profile and the objects and their relations which make loading and displaying this profile data possible. The numbering shows a possible order of actions a user can take. After each number a function is shown which is used to create components that can be displayed on screen. The user begins by starting the application. After this he creates a new profile using the profile object. The ProfileViewController then uses this profile object filled with data from the steam api to construct three other components: the pie chart with user statistics, the table with user statistics and the webview with the online steam profile of the user.

Acceptance criteria

The acceptance criteria for this user story are:

- A graph with information about the profile's played games is shown after handling the user input and pressing a button.

This can be seen in UI iteration figure 7

- The hours spent per game and the representation of this data in the graph must be accurate. The relative size of data items in the graph must be accurate.
- A user must be able to switch between the list representation of the data and the chart representation using buttons.

The user story says that a user must be able to see how much time he/she has spent playing games on steam in the form of a graph. The first component that is displayed after loading steam api data is the list with games and hours played. Because this is the default component that is shown. There had to be a button which could show the pie chart and another button to show the list again if the user had already toggled the pie chart. The buttons are have custom created icons to neatly describe their respective functionality. The

pie chart is coded in a way where the values of each data item is always correctly sized and weighted based on the total size of the values in the data set. The data from the steam api that is being used in the tableview is the reused in the pie chart. This means our application only has to load data once (when the users enters their steam name or id).

Patterns

The patterns that are used in this user story are described in section 3 of our report on page 6

6.3 Mike Traa

6.3.1 User story 6

Trello: https://trello.com/b/kxodU6sK/scrumboard-steam-stats
User story: “As a user I would like to view the account I searched on Steam, so I can add the account as a friend”
Git commit: “webview”
Git commit ID: `937ac8`

UML diagrams

Diagram 7 below shows how a the user can add the Steam account he looked up as a friend. I will talk more in depth about this diagram in the following section.

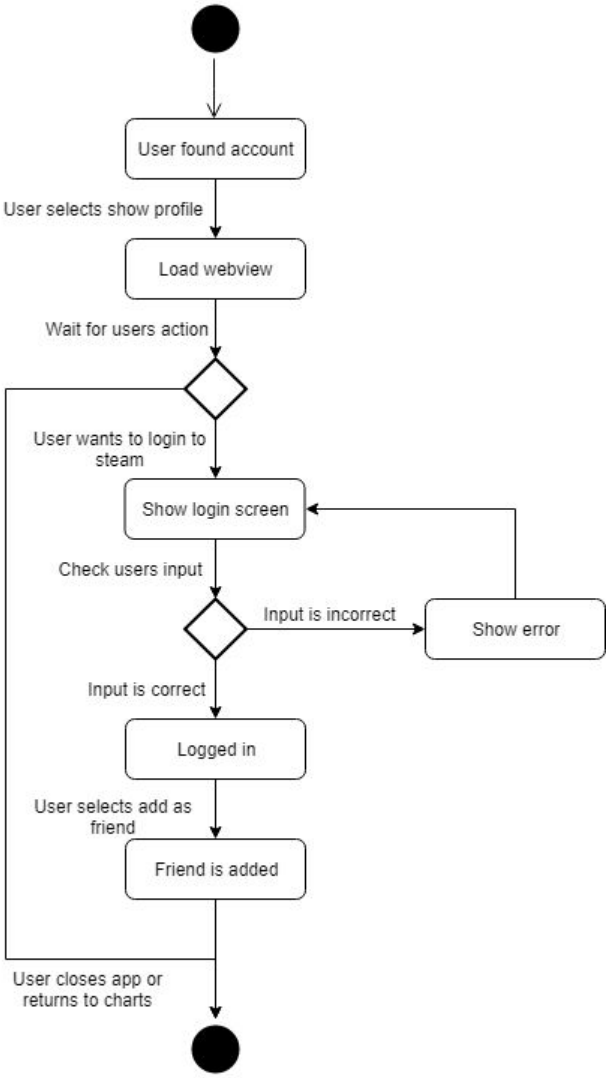


Diagram 7:Activity diagram of what happens when a user wants to add the account he looked up as a Steam friend

Diagram 7 shows what happens when a user wants to add the account he looked up as a new friend on Steam. At the start of this diagram has the user already found the account he was looking for. Now he wants to add that account as a new Steam friend. He will select the button ‘Show profile’ next to the profile picture. Then the webview with the correct Steam account from the Steam website is visible. Then the user can decide what he will do. When the user wants to add to account to his friendlist he has to login first. Logging in can also be done in the webview. If the user selects login within the webview he will be redirected to the Steam login page. If the user enters wrong info there he will be redirected back to the login screen. When he enters the correct username and password he will be redirected back to the account page. There the user has to click on add to my friendlist to add the account to his friend list. When the friend is added the user can return to view the stats or leave the app.

Acceptance criteria

The acceptance criteria for this user story are:

- A webview with the correct Steam account is showing when you press the ‘show profile’ button.

This can be seen in UI iteration figure 8

- The account shown in the webview must be the correct Steam account
- The user must be able to switch back to the stats table or pi-chart. This can be done using the buttons on top.

Patterns

The patterns that are used in this user story are described in section 3 of our report on page 6