Game in Progress: Design and Documentation of a Description-Based Game Recommendation System

Diego Eduardo Linares Trelles
(K12348021)

20th of June, 2024

Contents

1 Goal of the System **2**
- 1.1 What is its goal? . . . 2
- 1.2 How will it achieve its goal? . . . 2

2 Requirements **3**
- 2.1 Functional Requirements . . . 3
- 2.2 Non-Functional Requirements . . . 4
  - 2.2.1 External Interfaces . . . 4
  - 2.2.2 Performance . . . 4
  - 2.2.3 Attributes . . . 5
  - 2.2.4 Constraints . . . 5

3 Use Cases **6**
- 3.1 Main Use Cases . . . 6
- 3.2 Supportive Use Cases . . . 10
- 3.3 Use Case Diagram . . . 15
- 3.4 Traceability Matrix . . . 16
  - 3.4.1 Functional Requirements . . . 16
  - 3.4.2 Non-Functional Requirements . . . 16

4 Domain Model **18**
- 4.1 Initial Domain Model . . . 18
- 4.2 Current Database Implementation . . . 19

5 Architecture **20**
- 5.1 Architecture Diagram . . . 20
- 5.2 Component Description . . . 21

6 Design Questions **24**

Chapter 1

Goal of the System

Game in Progress (referred to for the remainder of this document as **GiP**) is a game recommendation system based on the descriptions given by developers, and reviews available on online game platforms (*e.g.* Steam).

1.1 What is its goal?

GiP allows players to discover new videogames without having to rely on the 'Similar Games' tab on game platforms, or the search bar. The former tends to only showcase a limited amount of fairly known games, while the latter usually only works through title names.

1.2 How will it achieve its goal?

GiP provides the users, through a web application, with a search bar that, instead of taking the title of a game, takes a description of what the user is looking for in natural language.

Based on the description given, the system would select, from a partially curated database of titles, a few with similar descriptions. A final selection would be obtained by going through the reviews of the games on `store.steampowered.com`, comparing them with the query for more similarities, and ranking them for the user.

Chapter 2

Requirements

2.1 Functional Requirements

The following functional requirements were initially set for the system:

Req01. When the user clicks on the description input field, the input box shall expand in size.

Req02. If a description with valid tokens has been inputted, when the user presses the search button, the system shall provide a list of games, with 50% average correlation or more.

Req03. When the user clicks the deeper search button, a slider input should become available.

Req04. If the user has activated the deeper search option, if the user uses the slider bar, the amount of similar games searched on Steam will be modified.

Req05. If a description without valid tokens has been inputted, when the user presses the search button, the system will indicate the user that at least one valid token is needed.

Req06. If the user has done a search, when the user clicks on a game, the system will redirect them to the corresponding Steam page.

Req07. If the user has done a search, when the user selects a sorting option, the curated table will sort by the corresponding criteria.

Req08. If the user has done a search and provided a rating to the result, when the user presses the rate button, their feedback is saved to the system.

2.2 Non-Functional Requirements

The following non-functional requirements were initially set as well:

2.2.1 External Interfaces

NfReq01. The start page of the system will have a single input field with a place-holder description and a deeper search button, all visible right away.

NfReq02. While the user waits for the search results, a loading modal will be displayed.

NfReq03. If the user has done a search, the ranking table will display only the top 10 results by similarity score.

NfReq04. While doing a search, the system does a first curation from a local database of game.

NfReq05. While doing a search, the system does a second curation of games from the Steam reviews.

NfReq06. If a deeper search has been activated, while doing a search, the system looks for similar games on the corresponding Steam page.

2.2.2 Performance

NfReq07. While under normal circumstances, the start page of the system should be available in ≈1 second for 90% of the users.

NfReq08. When the user has done a search, the system should be able to provide the results in less than 30 seconds[1].

NfReq09. When the user has done a search, the system will be able to do the

[1] We are aware this is a relatively slow performance, but we are limited by the latency of the Steam Reviews wrapper in Python.

first local curation in ≈10 seconds[2]

NfReq10. When the user has done a search, if the user re-sorts the table, the table should sort instantly (≈0.1 seconds).

2.2.3 Attributes

NfReq11. The system should work and be responsive in any modern desktop and mobile browser.

NfReq12. System recommendations should be comparable to other state-of-the-art search-by-content methods.

NfReq13. The system should be easily mantained to include new popular releases in the local database.

NfReq14. The system should limit the number of requests per IP to prevent any DDoS attacks.

2.2.4 Constraints

NfReq15. Both the system's front-end and back-end will be implemented in Python due to the availability of the required libraries.

NfReq16. Any search from the user will only interact with the local database through SELECT requests. The only INSERT request will be on feedback ratings.

NfReq17. Any updates to the database to add more videogames should be done manually.

NfReq18. The resources required by the system shall not surpass the ones provided by the Amazon Web Services free tier.

NfReq19. The operating system in which the system will be hosted will be the latest stable release of Ubuntu Server.

[2]Further experimentaiton with the Word2Vec library might modify this estimate. It might also vary as the database expands.

Chapter 3

Use Cases

3.1 Main Use Cases

These are the initial use cases for the GiP system:

Use Case: Description Search		
ID	UC01	
Description	Performs a basic search based on description and then by user review similarity	
Actors	Individual User, Review Search System (RSS)	
Stakeholders	Game Enthusiasts, Journalists/Content Creators, Developers, PWWR, Steam, Managers	
Pre-Condition	None (or the system is operative)	
Success End Condition	A list of games sorted by the similarity of the reviews is created and displayed for the user.	
Failure End Condition	There are no valid tokens to perform a description based search. No search is conducted.	
Main Success Scenario		**Linked UC**
1.	The user types a description on the input text box provided.	
2.	The user presses the search button.	
3.	The RSS performs an initial curation in the local database.	SUC01
4.	The RSS gets the reviews of the curated games.	SUC02
5.	The RSS sorts the list of the games by review similarity.	SUC03
6.	The system displays the list of games to the user.	
Alternative Scenario		
2. A1	The user input does not contain any valid tokens.	
2. A2	The system indicates the user that his input is not valid for search.	
2. A3	Go back to step 1.	
3. A1	The RSS does not find any games with enough similarity	
3. A2	The system indicates the user that his input is not valid for search.	
3. A3	Go back to step 1.	
Exception Scenario		
4. A1	The RSS takes more than 100 seconds to get the reviews from Steam	
4. A2	The user is given the option to try again later.	

Use Case: Description Deep Search		
ID	UC02	
Description	Performs a search adding similar games from the Steam Library to the curated list.	
Actors	Individual User, Review Search System (RSS)	
Stakeholders	Game Enthusiasts, Journalists/Content Creators, Developers, PWWR, Steam, Managers	
Pre-Condition	The user has typed a valid input.	
Success End Condition	A list of games sorted by the similarity of the reviews is created and displayed for the user.	
Failure End Condition	The list becomes too big for the NLP Model to process in a reasonable amount of time.	
Main Success Scenario		**Linked UC**
1.	The user presses the button for deeper search.	
2.	The user adjusts how many "Similar Titles" to consider for the search.	
3.	The RSS also looks for "Similar Titles" to the curated ones on Steam.	SUC04
4.	The Search is performed as in UC01	UC01
Alternative Scenario		
2. A1	The user inputs a number of similar titles greater than 20.	
2. A2	The system warns the user about the search not completing with those params.	
2. A3	Go back to step 2.	
Exception Scenario		
4. A1	The RSS doesn't complete the search in under 100 seconds.	
4. A2	The user is given the option to try again with other parameters.	

Use Case: Re-Sort the Games		
ID	UC03	
Description	Sorts the list provided to the user based on the criteria provided by them.	
Actors	Individual User, Review Search System (RSS)	
Stakeholders	Game Enthusiasts, Journalists/Content Creators, Developers, Managers	
Pre-Condition	The user has performed a description based search	
Success End Condition	The list of games provided to the user is sorted under a different criteria.	
Failure End Condition	The list of games provided is not sorted under the criteria desired by the user.	
Main Success Scenario		**Linked UC**
1.	The user presses the button for the sorting options.	
2.	The user selects under which criteria they wish the list to be sorted.	
3.	The user presses the "Sort" button	
4.	The system sorts the list under the selected criteria.	SUC03
Alternative Scenario		
2. A1	The selected criteria is the same as the one the list is currently sorted by.	
2. A2	The system warns the user that criteria is already applied.	
2. A3	The sort button becomes unavailable to press.	
2. A4	Go back to step 2.	
Exception Scenario		
4. A1	The system is not able to perform the sorting of the list appropriately.	
4. A2	The system informs the user that a problem occurred during the sorting	
4. A3	The system allows the user to attempt to sort again.	

Use Case: Getting More Game Information		
ID	UC04	
Description	Shows additional information on hover and takes the user to the Steam pages of a game.	
Actors	Individual User, Steam Platform	
Stakeholders	Game Enthusiasts, Journalists/Content Creators, Developers, Steam, Managers	
Pre-Condition	The user has performed a description based search	
Success End Condition	The list of games provided to the user is sorted under a different criteria.	
Failure End Condition	The user is not able de access the Steam page for the game.	
Main Success Scenario		**Linked UC**
1.	The user hovers one of the games in the displayed list	
2.	The system displays a modal with basic information about the hovered game	SUC05
3.	The user presses on the information modal	
4.	The system redirects the user to the Steam page for the game	
Alternative Scenario		
2. A1	The hovered game contains sensitive content.	
2. A2	The modal informs the user the information for this game is sensitive.	
Exception Scenario		
4. A1	The Steam page for the game is not available.	

Use Case: Rating a Search		
ID	UC05	
Description	The user is asked to rate the quality of the list provided as feedback.	
Actors	Individual User	
Stakeholders	Game Enthusiasts, Journalists/Content Creators, Managers	
Pre-Condition	The user has performed a description based search	
Success End Condition	The user's feedback is received and stored for future analysis.	
Failure End Condition	The user's feedback is not received.	
Main Success Scenario		**Linked UC**
1.	The user selects a star rating (1 - 5) for the search they just performed.	
2.	The user presses the rate button.	
3.	The system stores the params. of the search and the rating in a local DB.	UC01 / UC02
Alternative Scenario		
2. A1	The user presses the rate button without having provided a rating.	
2. A2	The system informs the user that a rating is needed.	
2. A3	Go back to Step 1.	
Exception Scenario		
3. A1	The system is not able to store the feedback after 20 seconds.	
3. A2	The system informs the user there has been a problem and to try again.	
3. A3	Go back to Step 2.	

Out of the initial 5 Use Cases that were provided here, **UC01, UC02 and UC05 were implemented** in the source code (considering only the Main Success Scenario), **UC04 was partially implemented** (missing the hover step), and **UC03 is yet to be implemented**.

3.2 Supportive Use Cases

The supportive use cases for the UCs detailed abover are the following:

Supporting Use Case: Local Database Curation		
ID	SUC01	
Description	The RSS conducts a curation of the local DB based on a description provided.	
Actors	Review Search System	
Stakeholders	Game Developers, Steam, Managers	
Pre-Condition	The user has provided a description to the system.	
Success End Condition	A curated list of games is provided for the next step of the system.	
Failure End Condition	The curated list of games provided is empty.	
Main Success Scenario		**Linked UC**
1.	The RSS goes performs a tokenization of every game description.	
2.	The RSS does a similarity comparison between the tokenized descriptions.	
3.	The RSS adds the game to the list if it reaches a similarity threshold.	
4.	The RSS uses the curated list for the next step of the search.	
Alternative Scenario		
4. A1	The curated list is empty.	
4. A2	The RSS reduces the similarity threshold needed to be added to the list.	
4. A3	Go back to Step 3.	
Exception Scenario		
2. A1	The system is not able to get a score after 100 / DB length seconds.	
2. A2	The game is skipped from the list.	
2. A3	Go back to step 2.	
4. B1	After 3 threshold reductions the curated list is still empty.	
4. B2	The system warns the user to provide a less specific description.	

Supporting Use Case: Getting Game Reviews		
ID	SUC02	
Description	The RSS gets the Steam Reviews of the curated games.	
Actors	Review Search System	
Stakeholders	PWWR, Steam, Managers	
Pre-Condition	The RSS has obtained a curated list of game to look for.	
Success End Condition	The RSS has a list of Steam Reviews to compare.	
Failure End Condition	The RSS is not able to obtain a list of Steam Reviews on the available time.	
Main Success Scenario		**Linked UC**
1.	The RSS performs a tokenization of the Steam Reviews for each game	
2.	The RSS does a similarity comparison between the description and reviews.	
3.	The RSS keeps the game in the list if it reaches a threshold of similarity	
4.	The RSS uses the list for the final sorting step.	
Alternative Scenario		
4. A1	The remaining curated list is empty.	
4. A2	The RSS reduces the similarity threshold needed and resets the list.	
4. A3	Go back to Step 3.	
Exception Scenario		
2. A1	The system is not able to get a score after 100 / List length seconds.	
2. A2	The game is skipped from the list.	
2. A3	Go back to step 2.	
4. B1	After 3 threshold reductions the curated list is still empty.	
4. B2	The system warns the user to provide a less specific description.	

Supporting Use Case: Sorting the Game List		
ID	SUC03	
Description	The RSS sorts the game list given by a certain criteria.	
Actors	Review Search System	
Stakeholders	Developers, PWWR, Steam, Managers	
Pre-Condition	The RSS has obtained a curated list by reviews of games to sort and a criteria.	
Success End Condition	The RSS returns a sorted list of games based on a criteria.	
Failure End Condition	The RSS is not able to sort the list of games correctly on the available time.	
Main Success Scenario		**Linked UC**
1.	The RSS compares the criteria of all the listed games.	
2.	The system checks which games have changed positions from the sorting.	
3.	The system returns the sorted list alongside with rank changes to the user.	
Alternative Scenario		
1. A1	Two games have the same score under that criteria.	
1. A2	The shown rank between the two is tied.	
1. A3	Continue to Step 3.	
Exception Scenario		
1. B1	The system is not able to sort the list in under 5 seconds.	
1. B2	The user is informed there has been a problem with the sorting.	
1. B3	The user is allowed to try again.	

Supporting Use Case: Searching for Similar Titles		
ID	SUC04	
Description	The RSS adds games considered similar by Steam to the curated list.	
Actors	Review Search System	
Stakeholders	Developers, Steam, Managers	
Pre-Condition	The RSS has been provided a least of curated games, and a depth parameter.	
Success End Condition	The curated list of games gets increased based on the depth of the search.	
Failure End Condition	The list of games is unable to be extended.	
Main Success Scenario		**Linked UC**
1.	The RSS assigns an amount of games to be added per game on current list. (The amount is based on similarity scores, and sum = depth)	
2.	The RSS access the "Similar Games" page of the game on Steam.	
3.	The RSS adds the first n entries assigned to the game to the list.	
4.	The RSS returns the extended list for the following step.	
Alternative Scenario		
2. A1	There are not enough similar games available to add for that game.	
2. A2	Go back to Step 1 and add the next game on the list.	
3. A1	The new list becomes larger than 20 entries.	
3. A2	The system warns the user in the modal that the search might not complete.	
Exception Scenario		
1. B1	The system is not able to access the Steam page.	
1. B2	The user is informed there has been a problem with the depth search.	
1. B3	The user is allowed to try again.	

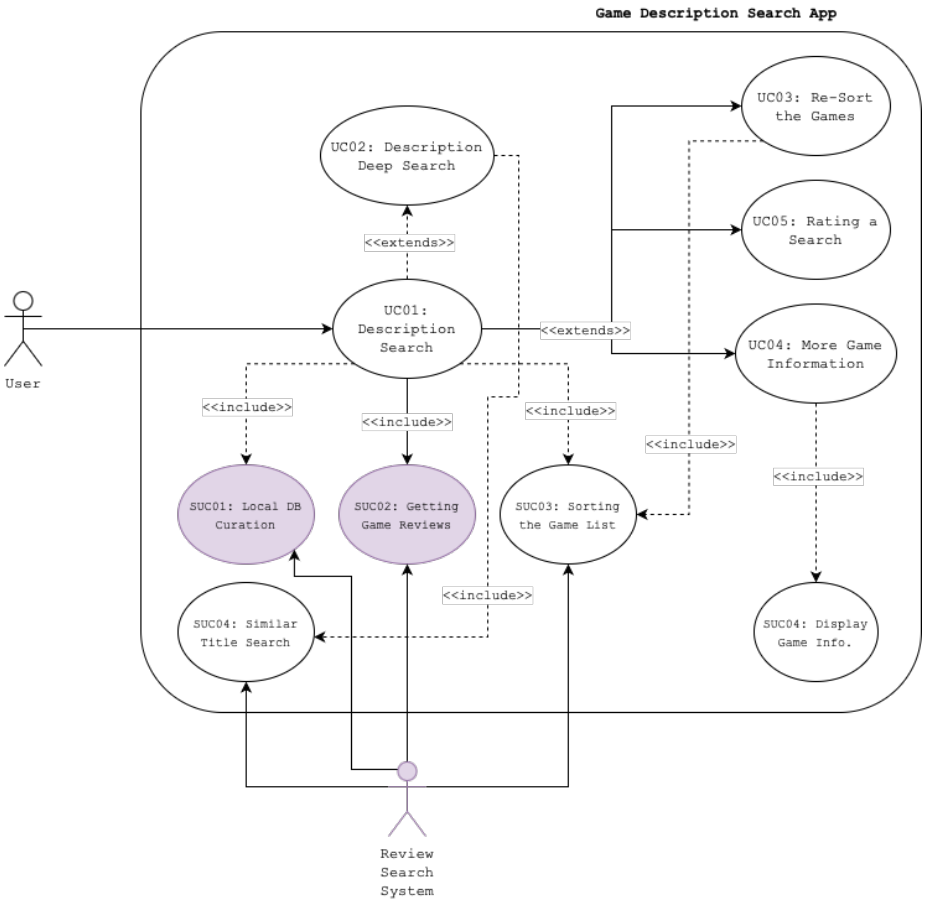
Supporting Use Case: Displaying Game Information		
ID	SUC05	
Description	The system displays a modal with information about the game.	
Actors	System	
Stakeholders	Game Enthusiasts, Journalists/Content Creators, Developers, Managers	
Pre-Condition	The game titles on the final list are available on the local DB.	
Success End Condition	The game information is displayed to the user on hover.	
Failure End Condition	The game information is not available to be displayed to the user.	
Main Success Scenario		**Linked UC**
1.	The system gets the main information of the game from the local DB.	
2.	The system pre-processes the information to display.	
3.	The system displays a modal with the information organized.	
Alternative Scenario		
2. A1	There is information missing in the local DB	
2. A2	The system adds a placeholder value in replacement.	
Exception Scenario		
1. B1	All the relevant information is missing from the local DB.	
1. B2	The system informs the user that info. for this game is not available.	

For these, **SUC01, SUC02 and SUC04 were implemented**, while **SUC03 and SUC05 are yet to be implemented** in the source code.

This means in total **6 out of 10 Use Cases were implemented**, 1 partially, and 3 are yet to be implemented in the repository available at `https://github.com/DiegoELT/game-in-progress`.

3.3 Use Case Diagram

We included all of the covered Use Cases in the following diagram:



Note 1: We considered `<<extends>>` for UCs 03 to 05 after, since they are only able to be performed after the UC01, but they are completely optional.

Note 2: Although the AI-Related SUCs are only SUC01 and SUC02, we considered the Review Search System to encompass SUC03 and SUC04 since all of those tasks are often performed together as steps of UC01 and UC02.

3.4 Traceability Matrix

As a part of the GiP system, every single User Case must be mapped to at least one Functional / Non-Functional Requirement

3.4.1 Functional Requirements

Req.	**UC01**	**UC02**	**UC03**	**UC04**	**UC05**	**SUC01**	**SUC02**	**SUC03**	**SUC04**	**SUC05**
01	■	■								
02	■	■				■	■	■	■	
03		■							■	
04		■							■	
05	■	■				■	■			
06				■						■
07			■							
08					■					

3.4.2 Non-Functional Requirements

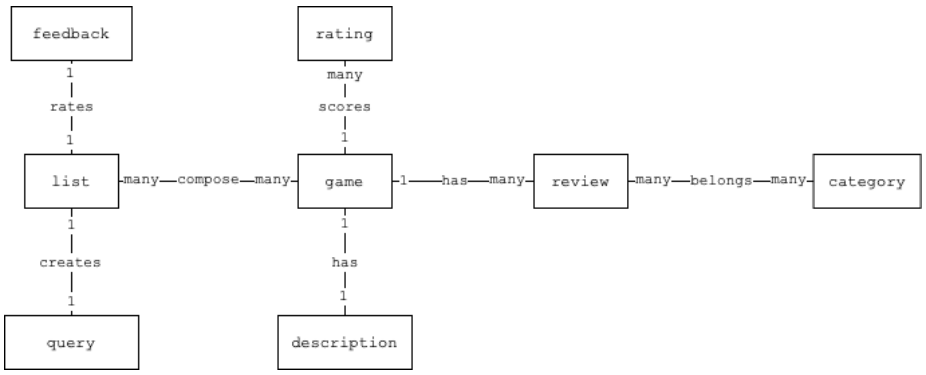
NfReq.	UC01	UC02	UC03	UC04	UC05	SUC01	SUC02	SUC03	SUC04	SUC05
01	■	■								
02	■	■				■	■	■	■	
03	■	■	■					■		
04	■	■				■				
05	■	■					■			
06		■							■	
07	■	■								
08	■	■				■	■	■	■	
09	■	■				■				
10			■					■		
11	■	■	■	■	■					■
12	■	■				■	■			
13						■				
14	■	■								
15	■	■	■	■	■	■	■	■	■	■
16	■	■		■	■	■				■
17						■				
18	■	■				■	■	■		
19	■	■	■	■	■	■	■	■	■	■

Chapter 4

Domain Model

4.1 Initial Domain Model

The domain model corresponding to GiP is the following:



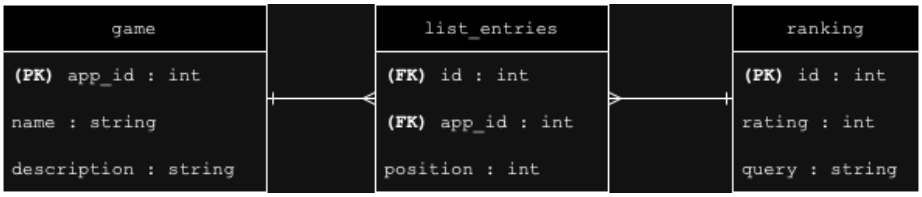
Some additional notes on it:

- The category of a review refers to the way Steam organizes their game reviews (i.e. helpful or funny).
- A description refers to the string a text that describes the game, either by the devs, or by the user. For the ones written by the user, a list is created when searching.

We are aware that a game has many more attributes than just review, rating and description (e.g. genre, publisher, etc.). But we are only tatking into consideration those attributes that are actually going to be used by the system. Same for the other components.

4.2 Current Database Implementation

Although the initial domain model, for the prototype implementation of GiP, the schema of the database follows this diagram:



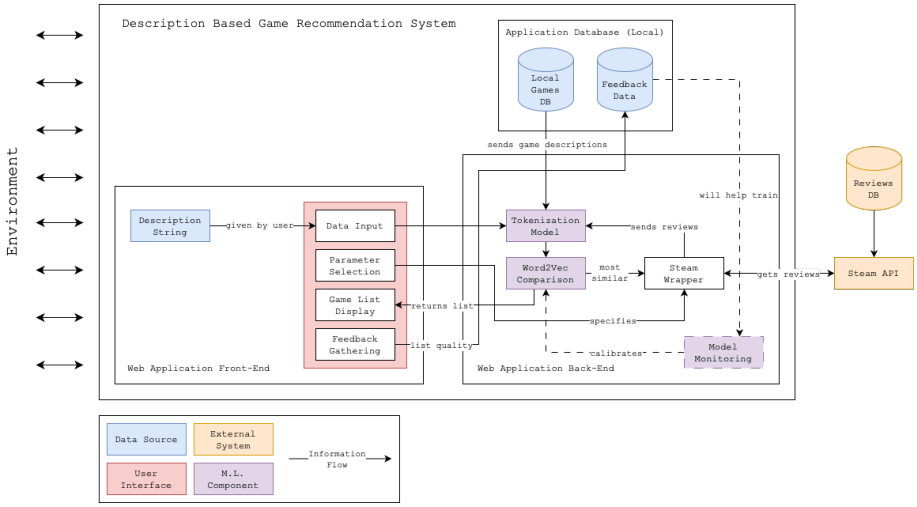
This shall be expanded to fit the domain model shown above when we include the additional component for model improvement.

Chapter 5

Architecture

5.1 Architecture Diagram

Following a UML adapted notation, the architecture diagram used as a guide to implement GiP is the following:



Some notes concerning its implementation in the prototype:

Note 1: Currently, the local game database and the feedback table are together on a single schema that is hosted locally.

Note 2: Currently the front-end / back-end distintion might be a bit blurry since the implementation has been done completely on Flask.

Note 3: Components and information flow shown in dashed lines are planned for a future iteration of the system and not actually included in the prototype.

5.2 Component Description

On first place we have the different components from the web-application front-end:

- **Description String and Data Input**: we are going to consider as a single component. Refers to the string that is going to serve as the query for the creation of the ranking. The information is parsed for security and sent to the backend for tokenization.
  - **Related Reqs**: Req01, Req02, Req05.
  - **Related NfReqs**: NfReq01, NfReq07, NfReq11, NfReq14, NfReq15, NfReq19
- **Parameter Selection**: refers to the slider that will be enabled if the user wishes to do a deeper search. Its use is optional and goes directly to the Steam wrapper as it only affects the amount of games retrieved.
  - **Related Reqs**: Req03, Req04.
  - **Related NfReqs**: NfReq01, NfReq07, NfReq08, NfReq09, NfReq11, NfReq15, NfReq19
- **Game List Display**: the main output of the system. After the processing of the query has been done, it features all the reccomendations the system have come up with. It also allows to link directly to the Steam page of the game.
  - **Related Reqs**: Req06, Req07.
  - **Related NfReqs**: NfReq02, NfReq03, NfReq10, NfReq11, NfReq15, NfReq19
- **Feedback Gathering**: refers to the star-rating system that can be given to the list of games provided by the system. In a future iteration of this project this could help to tune the similarity score between a description and a review.
  - **Related Reqs**: Req08.
  - **Related NfReqs**: NfReq03, NfReq12, NfReq15, NfReq16, NfReq19

The second part is the Application Database, also hosted in the Amazon Web Services, which consists of two schemas:

- **Local Game Database**: used to make the initial curation of games for the Word2Vec comparison. It contains a considerable list of popular titles with descriptions and other relevant information, and is updated manually in a periodic fashion.
  - **Related Reqs**: Req02, Req07.
  - **Related NfReqs**: NfReq02, NfReq04, NfReq09, NfReq13, NfReq16, NfReq17, NfReq18.
- **Feedback Data**: at the moment just conceptualized as a single table. It includes the query input as well as the rating of the table provided, as perceived by the user. May be used for tuning the similarity score in the future.
  - **Related Reqs**: Req08.
  - **Related NfReqs**: NfReq12, NfReq16, NfReq18.

Finally, on the Back-End of the Web Application we can identify 3 main components (and an additional one for the future):

- **Tokenization Model**: part of the Natural Language Toolkit implemented in Python. It's purpose is to take user descriptions, developer descriptions, and Steam Reviews and turn them into a list of tokens for comparison.
  - **Related Reqs**: Req02, Req05, Req07.
  - **Related NfReqs**: NfReq02, NfReq04, NfReq05, NfReq08, NfReq09, NfReq15, NfReq16, NfReq18, NfReq19.
- **Word2Vec Comparison**: also implemented in Python. Will take the two lists of tokens and compared them based on meaning. This will provide a similarity score needed to curate the games, first from the local database, and then based on the Steam Reviews.
  - **Related Reqs**: Req02.
  - **Related NfReqs**: NfReq02, NfReq03, NfReq04, NfReq05, NfReq08, NfReq09, NfReq12, NfReq15, NfReq18, NfReq19.
- **Steam Wrapper**: due to the limitations of the available base Steam API, we are using a wrapper implemented in Python to get the reviews and similar games. A downside of this approach is the limit rate of the requests. This may eventually get replaced by making the requests to Steam directly with a better API plan.

  - **Related Reqs**: Req02, Req04, Req06, Req07.
  - **Related NfReqs**: NfReq02, NfReq03, NfReq05, NfReq06, NfReq08, NfReq15, NfReq19.

- ***Model Monitoring***: this is an additional component that came up inspired by the example provided. Based on the feedback collected from the user, the way the similarity score is calculated can be tuned, so it's not only based on the output of the Word2Vec comparison. Currently considered for a future iteration of the project, though.
  - **Related Reqs**: Req02, Req08
  - **Related NfReqs**: NfReq03, NfReq08, NfReq12, NfReq15, NfReq18, NfReq19.

The Steam API itself which interacts with the wrapper, and their database in which they store their reviews are considered components external to the system and therefore not taken into account for this section.

Chapter 6

Design Questions

As a part of the design process, the following questions (alongside their respective answers) were formulated:

What does similarity score actually mean in our system?

In this initial iteration of the project, the similarity score is a weighted average of the comparison score given by the Word2Vec model we have in our back-end, which compares the user query and a set of (considered helpful) reviews extracted from Steam.

How can the feedback we receive from our users help to improve the recommendations?

The feedback received from the users can let us know which query terms are the ones that tend to cause faulty recommendations according to the users. Those can be used to tune the Word2Vec scores given for those tokens.

How can we explain the similarity score to the user?

Our initial approach will be giving a general explanation as a disclaimer when providing the composed list. Another possibility would be showing the users which of their tokens were deemed the most relevant when compared to the reviews, in order to formulate the recommendations.

How should we handle games which feature age-restricted content?

Due to the fact that our system currently does not have user accounts as a feature, we will not display any games which feature age-restricted content as

their main appeal. This can be done by easily parsing games by genre or rating.

How are we planning to scale when we start receiving more requests?

There are two main concerns for latency when we receive a higher amount of requests. First, optimizing the Machine Learning components by using higher processing power (which will lead us to change some NfReqs). Second, get a developer Steam API Key, in order to override the need for a wrapper, which limits the amount of reviews we can get.

Can we eventually allow for users to have accounts in our system?

We consider this a possibility in the future as it would allow for users to save their queries and also get recommended games with sensitive topics depending on the games. However, we are thinking of what other features could this provide to consider it a necessary addition to the system.

Do we plan to incorporate other game platforms into the system in the future?

This is one of our main goals in the future. However, it would require some research on how to obtain the reviews and descriptions for the games hosted there. Also, due to the overlap of games between platforms if the results improve enough to consider this expansion worth.

Will the same input always produce the same recommendations as output?

If it is done in a relatively short span of time (days), yes. However, as it is common for games to receive updates and for new reviews to get more relevant, the similarity score between a user query and the list of reviews might vary.

Should some game reviews be considered outdated and therefore ignored?

Yes, there are multiple games which reviews have changed massively after an announcement from their studio, or after a patch. Though this is a phenomenon usually exclusive to major titles, so they should be monitored for these cases.

Will the model be able to adapt to newer gaming terms which are constantly being adopted?

This is one of the most difficult issues to tackle, as we need to take into account the similarities of these new terms with the existing ones. At the moment our approach would be decomposing them into "known" tokens, but we are open to the idea of including them into the model in the future.

Is there a possibility to work with Steam in a more direct fasion than just using wrappers?

Yes, however, for that it is necessary to apply for a developer API Key for Steam. As of the moment of writing this, that has not been obtained. However, in the future, this would allow to reduce the latency to obtain the Steam Reviews and related games (when doing deeper search) by a considerable amount.

Should the system ever be ported to a mobile application or possibly a browser widget?

We don't see a purpose on creating an application for what we consider a limited set of features, as we consider it to be a case of over-engineering. It could possibly be adapted to a browser widget as sometimes ideas for games come from looking at videos on YouTube or reading gaming magazines, so being able to directly quote those might an ideal feature.