FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Application of Recommendation Systems on Immersive Environments to increase User Engagement

Miguel Gonçalves de Castro

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Luís Paulo Reis

July 30, 2022

Application of Recommendation Systems on Immersive Environments to increase User Engagement

Miguel Gonçalves de Castro

Mestrado Integrado em Engenharia Informática e Computação

July 30, 2022

Abstract

Immersive virtual tours are emerging when promoting spaces online and they have become much more effective and used for this purpose. However, this type of tool offers too much freedom to the user, which may lead to an incomplete tour, as users are able to jump to spaces, ignoring the important ones and missing important information within a space during their user journey. This problem compromises the objective of the experience due to the fact that, during the virtual tour, the owner that is promoting it wants the users to get to know the most important spaces and pieces of information as they are their selling points, which means that the users may become more attracted to the spaces and this will gather more clients for the virtual tour's owner. To solve this problem, automatic individual experiences were developed using artificial intelligence to create recommendation systems with machine learning, which optimises the journey by guiding the user through the spaces that will generate the most value for the owner. This aims to increase the user engagement by using any kind of tip to inform the users of what spaces they must jump to or what kind of information they should be looking at. 3Decide has a web platform, known as Alive Vision, which provides these immersive virtual tours using 360-degree images and 3D models. Therefore, this solution was developed on this platform in order to get virtual tours with a recommendation system that should improve the platform itself and all the types of virtual tours it already provides, such as smart tourism and e-commerce.

This dissertation shows how a recommendation system was developed and integrated in the virtual tours generated by 3Decides' Alive Vision. A server that provided two types of filtering, content-based and collaborative, was developed, in which the virtual tours could access to request recommendations and later display those recommendation to the user to improve the user journey. The results from the integration and implementation of the recommendation system and the immersive environment were analysed in order to understand the success of the development and the feasibility of applying this dissertation's solution in other platforms.

Keywords: Immersive Virtual Tours, Recommendation Systems, Artificial Intelligence, Machine Learning, Web Platforms

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Luís Paulo Reis for his availability to oversee this research, providing information and comments which allowed this dissertation to be concluded in a successful way. His expertise in the field was a great help and added enormous value to the project.

Besides my supervisor, I would also like to give my warmest thanks to 3Decide, specially to Eng. Carlos Rebelo, for the proposal of this thesis and the opportunity to work with them. Their support carried me through all the stages of developing this project, and for that I am very thankful.

Finally, I would like to thank my family: my parents, Jorge Castro and Paula Gonçalves, for providing everything I need in my life, my sister; Isabel Castro, for showing compassion and understanding throughout the writing of this dissertation; and lastly my girlfriend, Andreia Leal, for the continuous support, patience, and motivation that pushed me to complete this research. Without them, none of this would have been possible.

Miguel Gonçalves de Castro

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
ER	Extended Reality
HTTP	Hypertext Transfer Protocol
IMDb	Internet Movie Database
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbors
REST	Representational State Transfer
ROC	Receiver Operating Characteristic
SQL	Structured Query Language
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
URL	Uniform Resource Locator
VR	Virtual Reality

Chapter 1

Introduction

In the last few years, there was an increase in the use of digital platforms and applications and most companies and associations have adapted to this new digital era and try to use these virtual tools to reach their targets.

Immersive tours are one of the ways of promoting any business digitally, having the advantage of allowing the users to interact with the environment without the necessity of being present, thus being a powerful lever to appeal to the customers and profit from it [11].

This kind of virtual tool is able to provide different types of tours, either by creating an experience where the user can freely explore and interact or by creating a guided tour where the user is a mere listener and is limited to the pre-established script [5].

In spite of the chosen type of tour, immersive environments include a set of distinct experiences which vary in the immersive level; for example, there are 360-degree images which has a low immersive level, followed by 360-degree videos, augmented reality, and, the highest one, virtual reality.

This report will focus on immersive environments created with 360-degree images that allow the user to freely explore the content and the solution to increase the engagement on this type of virtual tours.

1.1 Context and Motivation

3Decide is a company which focus on developing visual technologies and contents and use them to create new business tools and apply them to any sector of activity, from marketing and sales to management. They developed a platform, called Alive Vision, that combines different types of media, such as 3D models, images, videos, 360-degree images and videos, among others, in order to create experiences that communicate with the users in a more visual way instead of displaying information in a traditional way as in a text display.

From this platform, a light version was developed with the purpose of creating simple and fast experiences. It all started due to the need of creating simple virtual tours where the user could explore 360-degree images, navigate through them and interact with some hotspots that contained information relevant to the position that they were in. New elements were added during the development to improve the quality of the information provided, such as modals with 3D models, videos, or gallery of images, and immersive elements which allowed to add the elements in a way that made it look like they belonged to the 360-degree image. The final product consisted on a platform where the creators were able to generate custom virtual experiences by simply configuring JSON files and adding the tiles.

Due to the customisation options, this platform became able to promote any kind of business, from real estate to museums and cultural environments, and, by using the elements that provided additional information, this light version could also function as an e-commerce web application, innovating the concept to a more immersive type in which the user could shop while interacting with the products.

The e-commerce potential and the versatility of the platform served as a start up for this dissertation, due to the fact that every virtual tour created give a high level of freedom to the user, consequently being necessary to come up with a solution to balance that freedom. Therefore, a a system that increases the engagement of the users to these immersive environments without removing the sense of free exploration is intended to be developed, which means that the platform should avoid limiting the users by using guided tours.

1.2 Objectives

The main goal of this dissertation is to develop a recommendation system that is capable of offering suggestions to the users about what information to look at and what spaces they should visit. Accordingly, in order to achieve this goal, the following objectives were defined:

- Search and study recommendation systems that are already in use on immersive platforms or general platforms that are relevant for the case, choosing the best approach for the development of Alive Vision.
- Identify features that should be added to the platform and explain how they are adequate for application to information systems, functionally and integration-wise.
- Develop a web-based proof of concept, resulting from the product of both previous points, which includes 3 major objectives:
  - Recommendation system for e-commerce integrated in the immersive platform;
  - Recommendation system which improves the user journey, i.e., suggesting the best path the user should take;
  - Recommendation system that provides a better routing inside each scene by highlighting points of interest;

These recommendation systems should gather their data from past experiences of other users and adapt to the current user

1.3 Document Structure

This dissertation is divided into 7 chapters, where the first chapter is the introduction to the dissertation, where the context and motivation and the objectives are presented. It is followed by a second chapter related to the state of the art which is introduced by a background of the technologies that will be used during the dissertation, also describing related work of recommendation systems on e-commerce, recommendation systems on immersive environments, and algorithms used when developing recommendation systems. The third chapter reports the problem that needs to be solved and the proposed solution, including methods to evaluate the solution and, lastly, describes the planned tasks for the dissertation. The fourth chapter describes the development of the server and the filtering algorithms included in it, and how the server can be accessed. It is followed by a fifth chapter that contains the integration of the immersive system with the recommendation system, explaining how it uses the recommendations returned by the server. The sixth chapter shows the results obtained from integrating an immersive environment with a server-side recommendation system, showing the user interface and the server's calculations, finishing with the evaluation of the algorithms. The last chapter concludes the dissertation and presents a plan to continue the work in the future.

Chapter 2

Recommendation Systems

2.1 Background

The Internet became a system which can help buyers meet sellers, and it enabled the creation of online marketplaces for different activities, including the sale of products, as is the case of eBay. There are a large number of companies that are investing on peer-to-peer business using online platforms as their base of operation [12]. These online marketplaces are considered electronic commerce (e-commerce), which is essentially a virtual space where commerce happens, and it allows sellers to expand their products, despite not increasing production, which offers a wider choice to consumers. For example, instead of having a physical bookstore with hundreds of books, sellers can make available millions of different books using virtual spaces, which can lead to tailored stores to their costumers where each costumer has access to products that are more relevant to them [36].

Immersive environments are systems that, implied by the name, try to create experiences where the user has the feeling of being present in a virtual world. This environments use technologies called extended reality and they refer to all real and combined environments and human-machine interactions, which include virtual reality, augmented reality and 360-degree videos and images [16]. There has been a rapid rising in popularity when talking about immersive technologies due to the value they add to many areas of people's lives, such as entertainment and training, as these systems offer the possibility of interacting with the environment while having a feeling of immersion, in spite of there being little to none development of personalised experiences [17].

Recommendation systems are considered as one of the most visible and successful technologies based on Artificial Intelligence, more specifically on Machine Learning, and they are being used on a regular basis whenever someone browses through most of the online platforms, such as e-commerce, media streaming or social networks. The main goal of this technology is to generate value to the consumer or the provider, or, as in most cases, to both of them, by suggesting things that should be of interest to the consumer which will lead to a bigger chance of consuming something from that platform [26, 8].

There are reports about how recommendation systems affect the business value, even though

the reports are not formal due to the fact that companies tend to avoid publicly sharing how they profit from this. These informal statements provided indicate that most actions on the platforms derive from any sort of recommendation, for example, YouTube claims that 60% of the clicks on the home screen are on recommended videos. It was revealed that the recommendations lead to a significantly increase in user engagement and, as a result, the business value is expected to exponentially grow. The fact that successful online platforms allocate a major part of their available interface to suggestions proves that this technology generates a lot of business value [26, 8, 4].

2.2 E-commerce

The popularity of recommendation systems affected every e-commerce platform, therefore the best work related are the most successful online platforms as is the case of Amazon [39, 30, 28, 10, 27, 4] and eBay [39, 12]. These two platforms set the standard regarding the features to implement when developing an e-commerce service [27].

Amazon uses an item-based collaborative filtering algorithm, developed by Amazon itself, which basically picks a small number of items from their database based on the current context and past experiences of the user. Their approach was developed on the assumption that if the user is looking for an item, and there were other people who were also looking for similar items, then the user must also want to look for those similar items. So, the algorithm search for users who bought the current item and fetch all the items those users purchased afterwards, and if the items were purchased with high frequency, those items are considered a good recommendation. The recommendations obtained have usually a high quality, especially if there is enough previous data, and the algorithm is able to scale to a high number of users and items [38].

Ebay also uses an item-based collaborative filtering algorithm. However they apply the filter to aggregations of item-level implicit user data. To choose the appropriate level of aggregation of items, they aggregate the items by categories and use a K-nearest neighbor search to find categories that are related to the current item. In their case, as they are searching for items to recommend, they also include the input entity, which is the category of the current item, so, in the end, the result of the search will have relevant categories and the category of the current item. After this first filter, they use seven different ways of obtaining the actual items to recommend [6]:

- Related Products: They use a similar collaborative filtering aggregated at the product level. The algorithm search the categories for products that have similarities to the seed item and select the most-viewed items for that given item.

- Co-views: This recall uses view behavioural signal, as opposed to related products which uses purchase behavioural signal. For instance, when obtaining the items through co-views, the coverage of recommendations increases and is more advantageous when the user is just browsing.

- Related Queries: This recall is used when the user searches for any item or category by using queries. They developed two caches, one that maps queries to items, where they associate the most popular items to a particular query, and another that maps related queries to user search queries, for example, "digital camera" is associated to the searched query "canon SLR". Therefore, whenever a user searches for a specific item, popular items related are also displayed.

- Compatibility: They added additional information to the items regarding the compatibility so that whenever items are being searched, the compatibility is enforced for the seed item so that the list of recommended items must be compatible with the seed item.

- Complementary of Similar: They use a K-nearest neighbor search whenever a search result doesn't have any item, in order to find elements that are textually similar.

- DeepRecs: This recall uses a text-based deep learning model, using text information from the seed and the recommendation items to find the items that most resemble the seed item through a neural network architecture.

- Popular: When there are no results from any of the previous approach, they return the most popular items from each category.

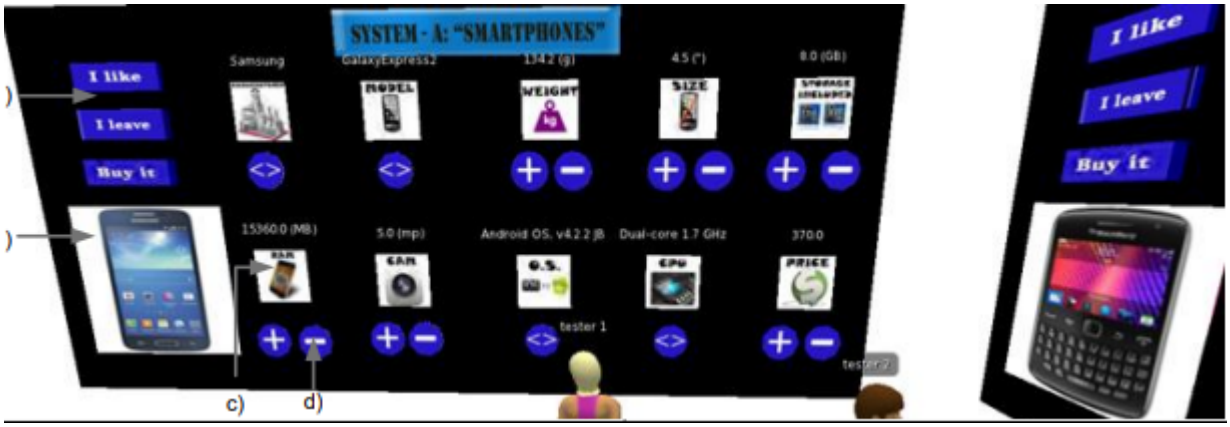
2.3 Immersive Environments

Recommendation systems on extended reality are scarce as little attention has been given to automatic suggestions to increase user engagement due to the fact that the development of these type of experiences is too complex as it mixes multiple fields [16]. There have been two proposals to fight this lack of recommendation systems on immersive environments where both based their project on systems already in use in e-commerce and applied them to virtual reality. One of the proposal was a prototype virtual reality store where the users could see a list of smartphones and they would rate each smartphone until the system showed them one they would like [7]. The other proposal was a platform that offered users the possibility of choosing any extended reality experience, much like YouTube does with videos, and users could select what they wanted to watch and rate it in the end [17, 16].

The smartphone store experience allowed multiple users to access a social virtual world where each user is able to critique smartphones individually or collaboratively. They integrated a collaborative filtering based on incremental critiquing, naming this approach collaborative conversational recommendation. Therefore, users would mutually help each other by critiquing each smartphone and get recommendations based on those critiques. The project consists on a virtual reality experience where each user would appear as avatars inside the experience and they have access to a recommendation interface where they can interact and critique smartphones, as it is possible to see on fig.2.1. Initially, the system provides the user with a smartphone based on previous critics of the user if there are any or based on the history of others users present on that session, then the

user can critique the recommendation by saying they want a different brand, a heavier smartphone, or simply stating if they like it or not. Whenever a user critiques a smartphone, the recommendation process begins and another smartphone is suggested. This recommendation and critique cycle continues until they decide to buy the recommended smartphone or to exit the experience [7].

Figure 2.1: Smartphone experience interface



The ER platform compiled a set of virtual reality experiences, from dancing performances to VR story simulation, where the user could select one, watch it and later rate it. In fig.2.2 it is possible to see the main menu where the user has access to all the experiences, also showing experiences that the system finds relevant to the user. The platform uses an AI component to suggest experiences using item-based collaborative filtering complemented with K-nearest neighbors. To keep track of the information for the collaborative filtering they have a database where they store all data about users, experiences and corresponding ratings, therefore, in order to recommend experiences, the following process is carried out [17]:

1. Load all the data about users, experiences and ratings into the database.
2. Filter all the experiences using collaborative filtering based on the ratings given by other users.
3. Use KNN to obtain experiences that are closer to the user's preferences.
4. Recommend the final list of experiences relevant to the user.

Figure 2.2: ER platform's main menu

2.4 Algorithms

As previously mentioned, recommendation systems are being widely used in e-commerce, which lead to an increase of studies regarding this technology, leading them to adopt machine learning algorithms which allowed computers to learn and process information, improving recommendations [32, 33]. Traditionally, there are techniques that don't necessarily involve machine learning, which attempt to filter the results based on information. The most commons methods are collaborative filtering, which rely on past actions related to the item and the user to try and predict the user's preferences; content-based, which uses the item description or the user's information, generally through web search mining, to fetch the items that are related to that information; and hybrid recommendation, which combines both collaborative filtering and content-based methods, attempting to minimise the drawbacks of each [10, 13, 18].

When using machine learning, there are four main classifications for each algorithm: supervised, unsupervised, semi-supervised, and reinforcement learning. Supervised and unsupervised learning are the most popular classifications for recommendations systems, having clustering algorithms, which are algorithms that try to group a given collection into clusters without previous information or training [25], as the most used for unsupervised learning, and support vector machines (SVM), which are algorithms that use decision boundary to separate classes, by using a subset of critical training samples a priori, and then matches the items to those classes [2], as the most used for supervised learning [32, 24].

Ensemble learning is also very popular. However their classification is not straightforward as it is not always supervised. This methods refer to the generation of a combination of multiple inducers, that can be of any type of machine learning algorithm, to solve a task, in this case to get recommendations, relying on the idea that the errors from an inducer will be compensated by the others inducers [34].

2.5 Summary

In summary, the use of recommendation systems have been increasing on every online platform, being most notorious when applied to e-commerce. Therefore, a lot of research has been carried out on different types of recommendation systems, with three different kinds of filtering emerging: collaborative filtering, content-based filtering, and hybrid filtering. Artificial intelligence and machine learning contributed to the development of these systems, having supervised, unsupervised and ensemble learning as the most popular when applying to recommendation engines. Immersive environments lack examples of recommendation systems integrated due to the high complexity of the development of the experiences, and, for that reason, there are some studies and experiments on extended reality that try to apply solutions from e-commerce platforms to immersive systems.

Chapter 3

Integration of a Recommendation System in an Immersive Environment

3.1 Problem

Immersive environments give users a sense of presence in the virtually created world they provide, which allows users to interact and explore the world freely without having to follow a narrative, which otherwise would lead to a storytelling path. 3Decide's platform, Alive Vision, also generates this type of virtual visits where users can explore the spaces as they see fit.

This freedom allowed by these environments is a double-edge sword as, in one hand, they offer an immersive world where users can do whatever they please and can explore each space at their own pace, and, on the other hand, users may miss crucial information, especially when the experience is selling something, being the latter the problem that this report is focusing on.

When a created a virtual tour using an immersive environment is created and the content is something that the creator is trying to sell, for example, an apartment, the creator hopes that users will explore everything and interact with every point during the user journey, which would be the best scenario. However, users may dislike a visit if the spaces they are exploring are not relevant to them, hence the user engagement would be very low, and both users and creator would not benefit from it. Therefore it is necessary to come up with a solution to integrate a system that leads users to look at information that is more relevant to them without forcing them, which takes away the freedom provided by these experiences.

Virtual tours developed within the Alive Vision platform contain multiple spaces that users can navigate through, and each space contains hot-spots with information of the area they are inserted. The platform also has the possibility to add e-commerce to virtual tours. All this features are relevant to the user engagement and, for that reason, it is necessary to develop a system that suggests products which are of interest for the user, whenever there is e-commerce, suggests hot-spots inside a space to look at, and suggests other spaces that may be of interest to the user.

3.2 Proposed Solution

To address the problem, the best solution to improve user engagement by increasing the possibility of showing the user something of their interest, without forcing the user to look at something or using a storytelling approach, is to implement a recommendation system that suggests other products, spaces, or areas to look at that the user will probably like. These systems are very common for most of online platforms, specially the most successful ones. Therefore, the approaches designed for these online platforms will be developed to solve the problem on immersive platforms.

Based on the three objectives proposed, a recommendation algorithm will be developed for each, one for the e-commerce, one for path suggestion, and one to highlight points of interest inside each space. Each recommendation system will run on server side and the client side will be responsible for requesting recommendations to the server which will be triggered by actions such as button clicks or elapsed time.

The e-commerce recommendation system will feature a collaborative filtering with clustering where the system will keep track of the interaction of previous users and use it to filter the products and obtain a list of viable recommendations. Clustering will be applied to the list obtained after filtering to get the item that resembles the most the item selected by the current user. This process will be triggered every time a user selects an item from the virtual tour; then after some time or if the user clicks the buy button, the client side will request a recommendation and the server will deliver the recommended item and a popup should appear in the immersive environment.

The path suggestion recommendation system will have a content-based filtering with clustering as well, where the system has access to all spaces available and will filter the spaces based on the data they have, thus generating a list of spaces relevant to the space the user is currently in. Clustering will have the same functionality as in the e-commerce recommendation system, where the algorithm will obtain the space most similar to the current space. It is assumed that if a user is exploring a space for too long, then it means they like that type of space, therefore the recommendation request will be triggered if the user remains in a space for a certain period of time.

The point of interest recommendation system will share the same algorithm as the e-commerce, making use of a similar collaborative filtering with clustering, and it will work in a similar way where the system stores the history of points of interest visited by previous users for each space and will generate a list of points of interest that are the most visited after a user looks at a certain area. Clustering will be used to get the best match out of the filtered list. Whenever a user clicks on a point of interest or looks at it for some time, a recommendation request is triggered and the point of interest recommended will be highlighted.

3.3 Solution Measurement

There are many evaluation methods of recommendation systems, such as error metrics, precision and recall, ROC curves, ranking score, and utility-based metrics [37, 15]. For the content-based

recommendation systems, a precision and recall method will be used, called precision@N, where N is the name number of items in a recommendation list, to measure the precision of the recommendation which consists on the rate of items in a recommended list that the user liked.

Precision@N calculates the utility of a recommendation list, which means the precision, using the formula at 3.1 where C_u refers to the number of items the user liked and R_u refers to the total of recommended items.

$$Precision@N: \qquad util(R_u) = \frac{|C_u \cap R_u|}{|R_u|} \tag{3.1}$$

In this case, the precision method will obtain beforehand the distance between an item and the selected item that the recommendation will be based on, and will use that distance to determine the number of items that a user would like. For example, if an item has a low distance, then it is considered that the item is relevant so it is added to the number of items that a user would like.

For the collaborative filtering, a different evaluation method is intended to be used (much more commonly used for measuring predictions) called the root mean square error, see eq. 3.2, which determines the error between observed values and the predicted values by the recommendation system [15, 14].

Root mean square error calculates the difference between the predicted rating for the item i, $\bar{r}_i$, and the real rating for that same item i, r_i, for each item available, being n the total of items. This method returns a positive value that the closer it is to 0, the better is the filtering performance [14].

$$Root\ Mean\ Square\ Error: \qquad rmse = \sqrt{\frac{\sum_{i=1}^{n}(\bar{r}_i - r_i)}{n}} \tag{3.2}$$

Later, user testing should be performed to measure the system as it is a common method in extended reality tours. These tests should be based on questionnaires where the testers can use an immersive experience that features recommendation systems and will answer some questions about the relevance of the recommendations.

Chapter 4

Implementation of the Recommendation System

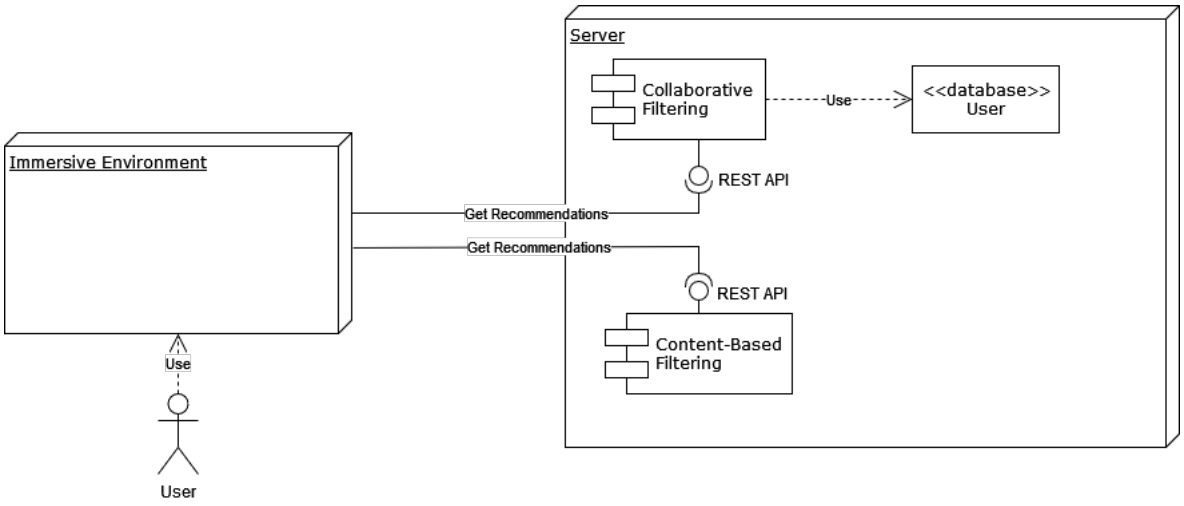
In order to develop the algorithms for filtering and recommendation, Python programming language was used, which included all built in libraries that are commonly used for machine learning, such as scikit-learn, pandas, and numpy, and features that allowed the creation of a Python based server were also used.

The server and the algorithms constitute the backend part of the project where all the predictions are made. The server is responsible for providing an interface to handle requests made, which includes handling the data received, applying the correct algorithm, and returning the recommended items. There are two algorithms implemented to compute recommendations - content-based filtering and collaborative filtering - and each uses machine learning techniques to obtain predictions.

Figure 4.1 shows a diagram with the modules available in the server which the immersive environment can use to obtain recommendations. The server from fig. 4.1 represents the entire recommendation system which displays two modules, one for each filtering method, that the user interface can access through a REST API, basically doing HTTP requests. The collaborative filtering module, since it is a user based filtering, uses a database to retrieve the users' history.

This chapter will describe how each filtering was implemented, along with an explanation of the choices made and how they were integrated in the server using Python libraries.

Figure 4.1: Interaction between the immersive environment and the server's modules

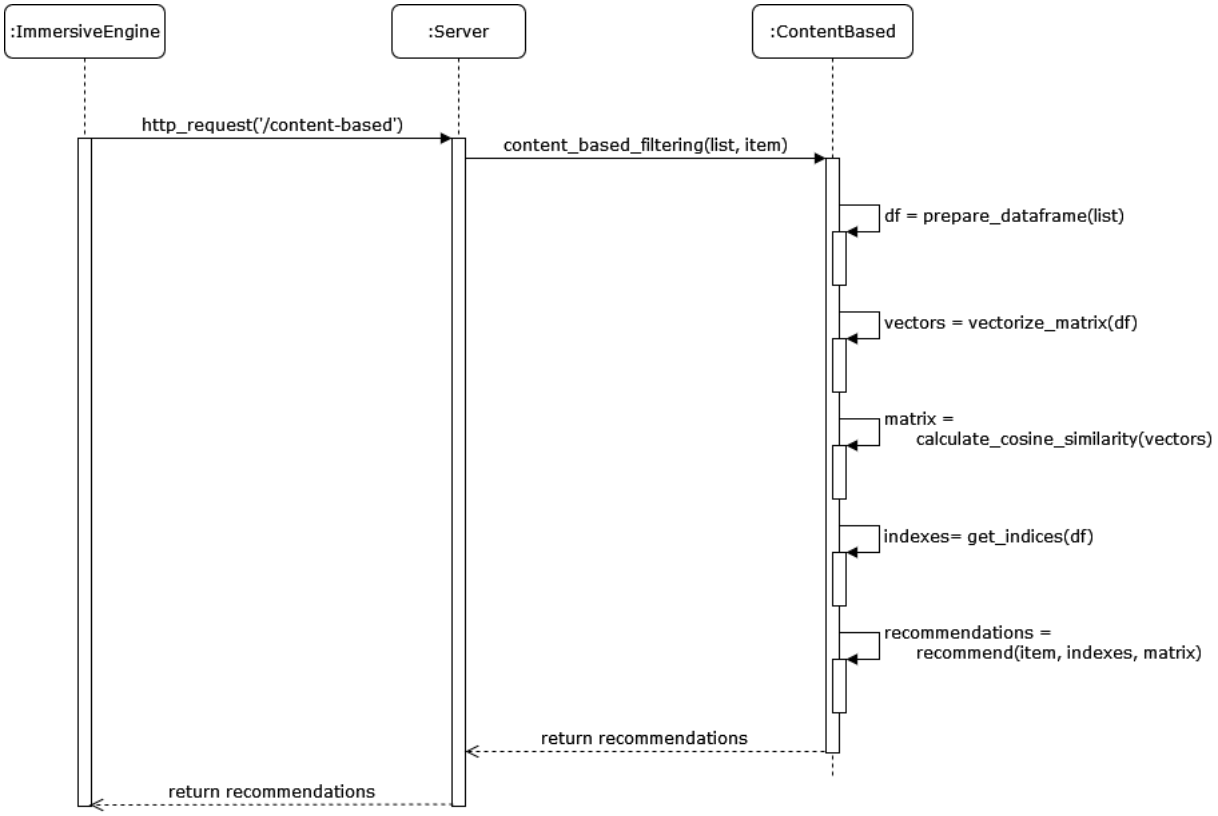


4.1 Content-based Filtering

Content-based filtering is a type of algorithm that calculates the similarity between an item and all items that are available and returns a list of items sorted by the resemblance to the given item. For this project, it was intended to develop a generic filtering which allowed the recommendation of any item from any category of items. Therefore it was implemented using clustering techniques which allowed the retrieval of similar items without a classification model.

The algorithm implemented follows the instructions seen in fig. 4.2, which are all the steps needed to obtain the recommendations based on the content of an item. The function from fig. 4.3 needs the selected item from the user and a list of all items that are going to be compared with, which includes the selected item, to calculate the similarities and recommend items. The algorithm starts by preparing a data frame based on the list provided, then uses the data frame to generate a matrix with the same number of rows as the list of items that contains the vectors corresponding to each item. The matrix is used to calculate the cosine similarity between the vectors and applies that similarity to the items' list to get the recommendations.

Figure 4.2: Content-based filtering sequence diagram



To prepare the data frame, python uses a library called pandas that is able to create a data frame from any type of list. In this case it creates from a JSON array, therefore producing a matrix where each row is an item and each column corresponds to the item features. After generating the

Figure 4.3: Content-based filtering algorithm

```
def content_based_filtering(list, item):
    df = prepare_data_frame(list)
    tfidf_matrix = vectorize_matrix(df)
    cosine_sim = calculate_cosine_similarity(tfidf_matrix)
    indices = get_indices(df)
    rec_indices = calculate_similarities(item, indices, cosine_sim)
    return df['title'].iloc[rec_indices]
```

data frame, it filters the columns to get only the title and description of each item, creating a data frame that contains the title and description of each item in each row.

After obtaining the data frame, it is used to generate vectors for each item by using a scikit-learn feature extraction function called TfidfVectorizer, as it is shown in fig.4.4. This function turns the description of each item into a two-dimensional vector based on the content of the description, making it easier to compare descriptions since it is only necessary to compare the coordinates of each vector. The matrix returned has the same number of rows as the number of items and has two columns, one is the vector and the other is the tf-idf value of the description, which is the statistical relevance of the description in relation to the set of descriptions of the data frame.

Figure 4.4: Algorithm to transform text to vectors

```
def vectorize_matrix(df):
    tfidf = TfidfVectorizer(stop_words='english')
    df['description'] = df['description'].fillna('')
    tfidf_matrix = tfidf.fit_transform(df['description'])
    return tfidf_matrix
```

Once the vectors are generated, they are used to calculate the cosine similarity making use of the cosine_similarity function from scikit-learn library. The similarity measure could be carried out by different techniques:

- Euclidean distance
  - Simple distance between two points by using the Pythagorean theorem [20, 41]:

$$Euclidean\ distance: \quad d(P_1,P_2) = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2} \tag{4.1}$$

- Manhattan distance
  - Sum of the distance of each point's axis values[20, 41]:

$$Manhattan\ distance: \quad d(P_1,P_2) = |x_1-x_2|+|y_1-y_2| \tag{4.2}$$

- Minkowski distance
  - It is a generalisation of the Euclidean and Manhattan distance that uses a variable p called order [20, 29]:

$$Minkowski\ distance: \quad d(P_1,P_2) = \sqrt[p]{(x1-y1)^p+(x2-y2)^p} \tag{4.3}$$

- Cosine similarity
  - Measures the cosine of the angle between two vectors [19], where the higher the cosine, the closer they are:

$$Cosine\ similarity: \quad cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||} \tag{4.4}$$

The Euclidean distance is the most commonly used method when calculating the distance between two points due to it's simplicity, measuring a straight line between those points, being used in many machine learning algorithm as is the case of image recognition [20, 41, 40].

The Manhattan distance has the best discrimination when examining high-dimensional data spaces, therefore it is the preferred method for calculating distances on data sets where no emphasis on outliers is desired [1].

The Minkowski distance loses to both Euclidean and Manhattan distance when choosing a method to calculate distances, since they are more popular and are both derived from this distance. However Minkowski distance are often used when it is calculated the distance between traditional vectors [3].

The cosine similarity was chosen over the other measures because it is more adequate when comparing texts as even if the distance between two texts is too far apart, they can still be similar if the angle between the vectors is small [21]. For example, two texts with different size have a big distance however, if their content is similar, the cosine of the angle formed by their vectors is closer.

The cosine similarity function calculates the cosine value between each vector, creating a matrix with the same number of rows and columns, which are the same as the number of items,

where each row and column correspond to an item and each value is the cosine similarity between the row item and the column item.

After calculating the cosine similarity, to obtain the recommendation list, the index of the target item in the cosine similarity matrix is obtained, then the row of that index is extracted by creating a data frame, and the values are sorted in ascending order thus obtaining the recommendations as a list of indexes, as seen in fig. 4.5. In the end, the list with the corresponding items' titles instead of the indexes is returned.

Figure 4.5: Function to obtain the recommendation list

```
def calculate_similarities(item, indices, cosine_sim):
    target_index = indices[item]
    similarity_scores = pd.DataFrame(
        cosine_sim[target_index], columns=["score"])
    item_indices = similarity_scores.sort_values(
        "score", ascending=False)[0:11].index
    return item_indices
```

The full code can be found in the appendix A.

4.2 Collaborative Filtering

Collaborative filtering is a user-based filtering where the system predicts which items the user will probably like based on the history of other users. The recommendation system stores all users' ratings of every item, and uses the stored data to predict the ratings of items the current user hasn't rated yet, as illustrated in fig. 4.6. In fig. 4.7, a general view of the implemented algorithm is shown, where it receives as parameters the connection to the database and the user ID. The algorithm starts by preparing a table with the users and their ratings, obtained from the connection to the database, then it gets the list of items from that table. Afterwards, it iterates over the list of items and for each item that the user indicated in the parameters hasn't rated, it predicts a rating and adds it to the recommendation list. In the end it returns the recommendation list sorted by the items' prediction.

The table with all the ratings is obtained by fetching from the database. This database is a server-less SQL engine, named SQLite, where the library sqlite3 already built-in python is used to perform all database action. The data returned from the database consists of a list of user history where each element of the list has an user ID, an item, and the rating of that item by that user, which means that there can be multiple elements with the same user id, in the case that the same user has rated more than one item. Therefore it is necessary to create a pivot table from that list where each row is a user and the columns are the items that exist. After getting the table, the list of items available is retrieved by getting the values of the columns and the index of the current user is obtained from the rows of the table.

Figure 4.6: Collaborative filtering sequence diagram

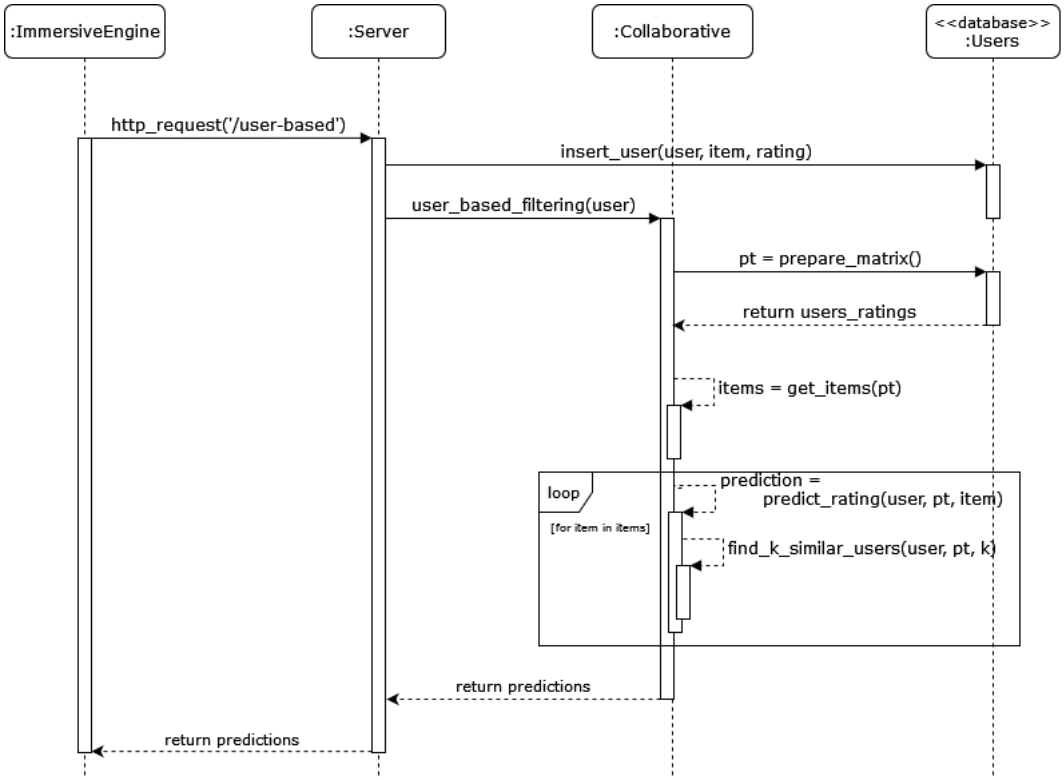


Figure 4.7: Collaborative filtering algorithm

```
def user_based_fitering(conn, user):
    recommendations = []
    pt = prepare_matrix(conn)
    if pt.empty:
        return []
    else:
        items = get_list_of_items(pt)
        user_idx = pt.index.values.tolist().index(user)
        k = len(pt.index.values) - 1
        if k > 10:
            k = 10
        for i in range(0, len(items)):
            if(math.isnan(pt.values[user_idx][i])):
                pred = predict_rating_item(user_id=user_idx, ratings=pd.DataFrame(
                    pt.values), item_id=i, k=k)
                recommendations.append(
                    {"item": pt.columns.values[i], "prediction": pred})
    return sorted(recommendations, key=operator.itemgetter(
        "prediction"), reverse=True)
```

In order to predict the rating of the items, the similar users to the current user based on their ratings must be found. The KNN algorithm is used for this purpose, as it is possible to see in the function from fig. 4.8. This function starts by filtering the ratings table by temporarily removing all the columns that the user hasn't rated, then it constructs an unsupervised learner model for searching neighbors, which is the learner NearestNeighbors from the scikit-learn library, and uses the filtered ratings to estimate the nearest neighbors, using the model. After estimating the neighbors, the model is able to indicate the k nearest neighbors, by using the function kneighbors, where k is the number of users or 10 if the number of users exceeds 10, returning the indexes of the nearest items and the corresponding distance. The similarity is the inverse of the distance, which means that the lower the distance, the higher the similarity.

Figure 4.8: Function to obtain similar users

```
def find_k_similar_users(user_id, ratings, metric='cosine', k=4):
    ratings = ratings.loc[:, ~ratings.iloc[user_id].isna()]
    ratings = ratings.fillna(0)
    model = NearestNeighbors(metric=metric, algorithm='brute')
    model.fit(ratings)

    distances, indices = model.kneighbors(
        ratings.iloc[user_id, :].values.reshape(1, -1), n_neighbors=k+1)
    similarities = 1-distances.flatten()

    return similarities, indices
```

Fig. 4.9 shows the algorithm to predict the rating of an item by an user. It uses the previously mentioned function (fig. 4.8) to obtain a list of similar users, then sums the similarities and calculates the mean rating of the user. Next, it iterates the list of similar users and for each neighbor it calculates the mean deviation of the neighbor rating for the specified item and multiplies it by the similarity value between the neighbor and the current user. The results of each similar user is summed in order to calculate the weighted average, which is later added to the mean rating calculated previously in order to obtain the prediction value, which is calculated using the formula in eq. 4.5, where u is the current user, i is the item, r is the rating, s is the similarity between users, and n is a user which belongs to N, which is the set of similar users.

$$Prediction: \quad p(u,i) = \bar{r}_u + \frac{\sum_{n \in N}(r_n, i - \bar{r}_n) \times s_u, n}{\sum_{n \in N} s_u, n} \tag{4.5}$$

This algorithm was based on the implementation of the same filtering algorithm by Chhavi Saluja [35], and the full code can be viewed in appendix B.

Figure 4.9: Function to predict the rating of an item

```
def predict_rating_item(user_id, item_id, ratings, metric='cosine', k=4):
    prediction = 0
    similarities, indices = find_k_similar_users(user_id, ratings, metric, k)
    ratings_tmp = ratings.loc[:, ~ratings.iloc[user_id].isna()]
    ratings = ratings.fillna(0)
    mean_rating = ratings_tmp.loc[user_id, :].mean()
    sum_wt = np.sum(similarities)-1
    product = 1
    wtd_sum = 0

    for i in range(0, len(indices.flatten())):
        if indices.flatten()[i] == user_id:
            continue
        else:
            ratings_diff = ratings.iloc[indices.flatten(
            )[i], item_id-1]-np.mean(ratings.iloc[indices.flatten()[i], :])
            product = ratings_diff * (similarities[i])
            wtd_sum = wtd_sum + product
    prediction = int(round(mean_rating + (wtd_sum/sum_wt)))
    return prediction
```

4.3 Server

The server carries out the communication between the immersive environment and the recommendation algorithms, functioning as an API, allowing the immersive interface to request recommendations based on the user interaction.

Due to the fact that both filtering algorithms were developed using Python, the server was also developed using that language, where its structure can be seen in fig. 4.10, making use of the SimpleHTTPRequestHandler class built in the http library from Python itself to create a new Server class to handle the requests made via HTTP. This class implements three methods: do_GET, do_OPTIONS, and do_POST, where the first handles all GET request, the third handles the POST requests, and the do_OPTIONS handles the OPTIONS requests, which is less usual when talking about HTTP requests. However it is necessary to send the CORS headers to the requester allowing the latter to do other requests without having the web browser blocking due to CORS policy.

The first thing the server does when it starts to run is to connect to the database where the users' history is stored. This connections stays open until the server stops to avoid having to connect to the database every time a request is made.

Both do_GET and do_POST allow the setting of routes for the request by indicating the path on the request. For example, it is possible to make a GET request for the url `http://server-url/list-all` where /list-all is the route which returns a specific response to that request, meaning that the route's feature allows the same type of request (GET or POST) to return different responses, as indicated in the example in fig. 4.11.

Figure 4.10: Server class to handle HTTP requests

```
from http.server import SimpleHTTPRequestHandler, HTTPServer

serverPort = 8080


class Server(SimpleHTTPRequestHandler):
    def do_GET(self):
        #GET requests
        pass

    def do_OPTIONS(self):
        #OPTIONS requests
        pass

    def do_POST(self):
        #POST requests
        pass


if __name__ == "__main__":
    webServer = HTTPServer(("", serverPort), Server)
    print("Running at port: ", serverPort)

    try:
        webServer.serve_forever()
    except KeyboardInterrupt:
        pass

    webServer.server_close()
    print("Server stopped.")
```

Figure 4.11: Example of server route

```
def do_GET(self):
    if(self.path == "/list-all"):
        list = list_all(self.conn)
        self.send_response(200)
        self.send_header("Content-type", "application/json")
        self.end_headers()
        self.wfile.write(json.dumps(list).encode("utf-8"))
```

The do_POST method allows two routes, one for the content-based filtering and the other for the collaborative filtering, as seen in fig. 4.12. The method starts by reading and converting to JSON the body of the request by using the rfile BinaryIO object from the server to read the bytes into a string, then loading the string using the JSON library. If the request is for the content-based filtering, the body must have a list of items and the selected item, then the content-based filtering function is called and the result recommendation list is sent back to the requester. If a user-based filtering is requested, the user needs to rate the item, since the point is to only recommend after the user has rated an item. Therefore, the server expects to receive from the body of the request the user ID, an item and the rate for that item. The server inserts into the database table the user's rate then it calls the user-based filtering function, returning to the requester a list of recommended items.

Figure 4.12: Server's do_POST method

```
def do_POST(self):
    data_string = self.rfile.read(int(self.headers["Content-Length"]))
    data = json.loads(data_string)
    if(self.path == "/content-based"):
        self.send_response(200)
        self.send_header("Content-type", "application/json")
        self.end_headers()
        items = content_based_filtering(data["items"], data["item"])
        self.wfile.write(json.dumps(items.values.tolist()).encode("utf-8"))
    elif(self.path == "/user-based"):
        self.send_response(200)
        self.send_header("Content-type", "application/json")
        self.end_headers()
        insert_user(self.conn, data["user"], data["item"], data["rate"])
        items = user_based_fitering(self.conn, data["user"])
        self.wfile.write(json.dumps(items).encode("utf-8"))
    else:
        self.send_response(404)
```

The server allows the creation of users' IDs when asked, by making a GET request to the /create-user route. In fig. 4.13 it is possible to see that whenever the request is made, the server starts by checking if there is a user_0 by calling the exist_user function, which is a database query that returns the number of entries with that user ID, therefore if that number is 0 it means the user doesn't exist, otherwise it generates random IDs until it doesn't exist. After obtaining an available ID, the system returns the user.

Figure 4.13: Function to generate users' ids

```
if(self.path == "/create-user"):
    id = 0
    user = "user_" + str(id)
    while(len(exist_user(self.conn, user)) > 0):
        id = random.randint(0, 9999)
        user = "user_" + str(id)
    self.send_response(200)
    self.send_header("Content-type", "application/json")
    self.end_headers()
    self.wfile.write(json.dumps(user).encode("utf-8"))
```

4.4 Summary

In summary, the recommendation system was build by developing a Python server which allows HTTP requests to retrieve recommendations. This server has the ability to use routing. Therefore, two types of filtering, content-based and collaborative, were created, which can be accessed by making POST request to the corresponding route. Both types of filtering were developed in Python, using machine learning algorithms. For content-based, it a vectorizer was used to convert text to vectors, then the cosine similarity was used to obtain recommendations. Regarding the collaborative filtering, the k-Nearest Neighbors algorithm was sued to obtain the similar users and afterwards, for each item, it predicts the rating for that item based on the user mean rating and the weighted average of the similar users, generating a list of ratings of items, which is sorted from highest to lowest rating, corresponding to the final recommendation list.

Chapter 5

Integration with the Immersive Environment

The main purpose of this dissertation is to apply a recommendation system to an immersive environment. In this case, the 3Decide's platform was used which allows the creation of web-based virtual tours.

This chapter explains the integration of the previous system developed (chapter 4) with the 3Decide's platform, in order to create a virtual experience with recommendations to increase user engagement.

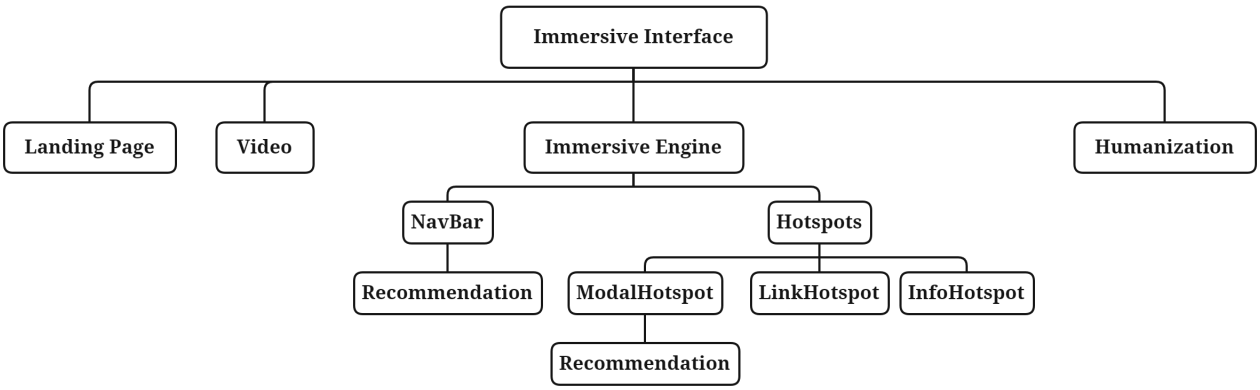
5.1 Immersive Environment Used

As mentioned in this chapter's introduction, the immersive environment used is the 3Decide's platform to create virtual tours, called Alive Vision light. The necessary code to integrate with the recommendation system API developed was implemented in this platform.

Alive vision light is a web based platform developed using the React typescript library and Marzipano [31] as the viewer for the 360-degree media. The structure of each component is represented in fig. 5.1 where it is possible to see that the platform has a parent container called explorer that controls the entire state of the platform, and also has a component which controls the viewer which is responsible for initializing the 360-degree images and adding the complementary information to those images. There are other components that are irrelevant to this dissertation, which are the video component and the humanization component, as well as the landing page which for this project only has the purpose of indicate to the user how they can interact with the experience.

The explorer component aggregates all the state and provides a context to allow any element inside the context to access and change the state. The initial state is loaded by reading from configuration files, which means that each virtual tour created by this platform can be customised by

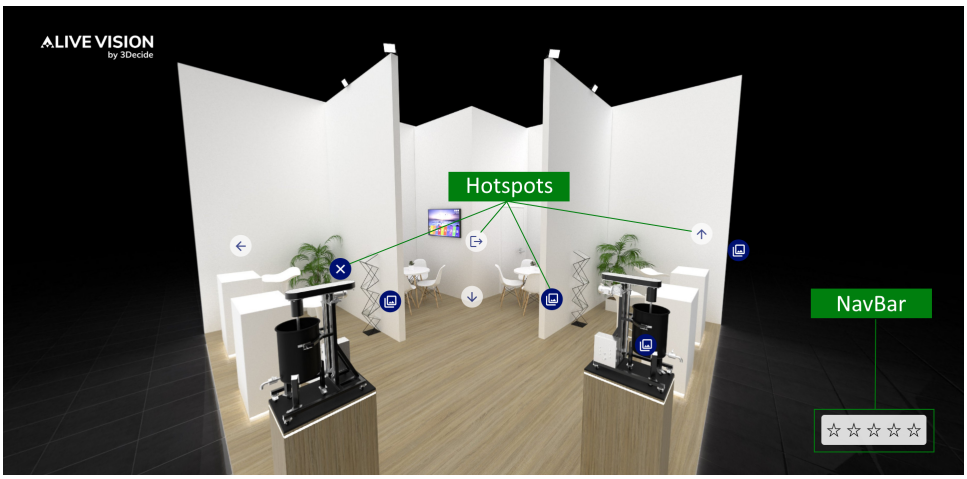
Figure 5.1: General components' structure of Alive Vision light



JSON files. Therefore, this container's only objective is to store the state and load all configurations needed to create the experience, to later render the Marzipano and the others components.

The component called Marzipano uses the library with the same name to display the viewer for 360-degree media. The platform stores the tiles generated by the Marzipano Tool and uses similar JSON files to the ones generated by this Tool to structure the virtual tour, hence, this component iterates over the data from the JSON file to prepare the engine and to load the 360-degree images from the tiles. Apart from the use of the viewer, this component additionally renders two components that help the exploration of the virtual tour created, one called NavBar which is short for navigation bar, and the other being the hotspots' container which carries elements that decorate immersively the viewer and allow actions to be performed, see fig. 5.2.

Figure 5.2: Components within a virtual tour



The navigation bar is a component that allows the use of templates and the required template is defined in the configuration files. Each template shares the same ideology of being fixed in the screen at all times despite the media that is being presented in the viewer. Therefore the navigation bar is always present so the user has access to shortcuts to change the displayed image or do other types of actions. For this work, a recommendation template which added a new feature that allows the user to rate the virtual scene shown was created.

The hotspots' container is a component that groups all type of hotspots which, up to now, are the link hotspots, information hotspots and modal hotspots. Hotspots are elements that Marzipano embeds in its viewer, creating the illusion that the 360-degree media has those elements incorporated in themselves. Each hotspot is positioned inside the media and has an action associated. Link hotspots are hotspots that when interacted with, the image is changed. Therefore, the hotspot creates a link between two images, which allows the navigation between scenes. Information hotspots show a small description whenever the user hovers the hotspot. Modal hotspots are elements that whenever they are clicked, a modal with information is displayed. Each modal is configured by selecting a template, and for the recommendation, a template to show information about a product with space to show recommendations was developed.

5.2 Recommendations using Content-based Filtering

Content-based filtering was applied to the situation where similar products need to be recommended to the user whenever they are browsing the modals. The application of this filtering in the previously mentioned immersive environment goes through, in a general perspective, a HTTP request to the server asking for similar items, using the content-based filtering anytime the user is looking at a certain product by interacting with the modal hotspots, then displaying the resulting list to the user so that the user can continue to browse products that presumably are of the user likening. The flow of the interactions between the user, the immersive environment and the recommendation system can be seen in fig. 5.3.

The request is automatically made whenever the user opens any product modal. For that, the user must interact with the hotspot to open the modal, which subsequently creates a timer of 3 seconds so that, when it is over, the request is made and the recommended items are shown. If the user decides to close the modal, that timer is cleared and nothing is done.

In order to perform HTTP requests to the server, the fetch function built in typescript is used, where it is passed as arguments the server URL, the body of the request composed by the item to compare to and the list of all products, and the headers commonly used to send HTTP request. Figure 5.5 shows the function created to get the recommendation that starts by obtaining the list of all products, which is an iteration over the list of modal hotspots passed to the template, where the list is obtained from the configuration file by declaring in the JSON an array of objects with at least a title and a description for each product. After obtaining the list of all products, the fetch function to request the recommendation to the server is called, which in the fig. 5.5 uses the localhost URL since the server was not deployed, and sends as body of the request an object which the item

Figure 5.3: Recommendations using content-based filtering sequence diagram

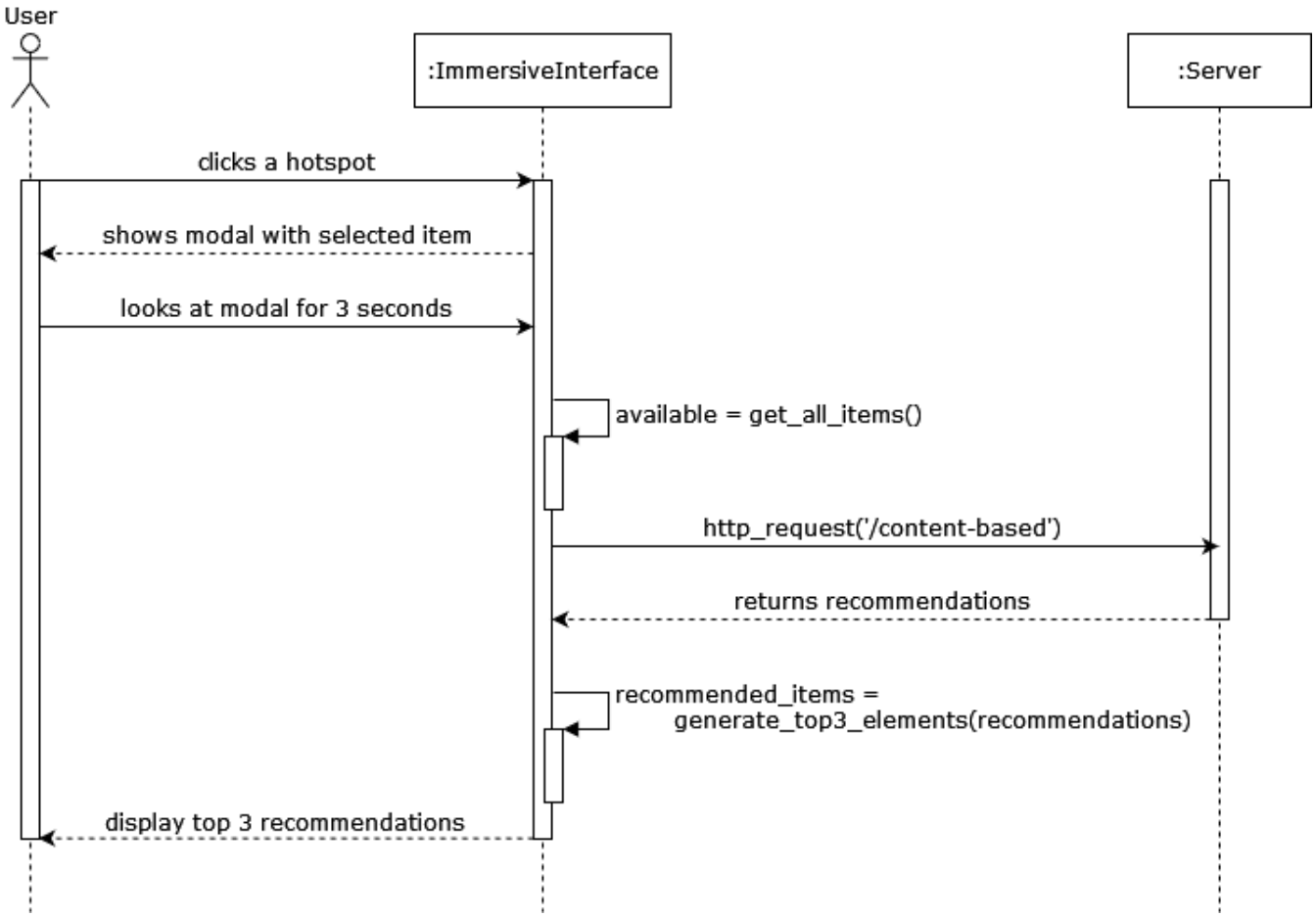


Figure 5.4: Snippet of a recommendation request

```
if (active && !timeoutId.current) {
  timeoutId.current = setTimeout(() => {
    getRecommendations();
  }, 3000);
}
if (!active) {
  timeoutId.current && clearTimeout(timeoutId.current);
  timeoutId.current = undefined;
}
```

property is the title of the product of the modal template and the items property is the obtained list. This fetch function call returns a response from the server where its data corresponds to a list of recommended products, where the first element of the list is removed, given that the first recommendation is the item itself, and the list is filtered to get only the top 3 recommendations. In the end, the template, which is a React component, updates the component's state with the filtered list.

Figure 5.5: Function to obtain a list of recommended products

```
const getRecommendations = () => {
  const available = products?.map((p) => {
    return { title: p.title, description: p.description };
  });
  const body = {
    item: props.hotspot.title,
    items: available,
  };
  fetch("http://localhost:8080/content-based", {
    method: "POST",
    headers: {
      Accept: "*/*",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(body),
  })
    .then((response) => {
      if (response.status === 200) {
        return response.json();
      } else {
        throw new Error("Something went wrong on api server!");
      }
    })
    .then((response) => {
      response.shift();
      response = response.slice(0, 3);
      setRecommended(response);
    })
    .catch((error) => {
      console.error(error);
    });
};
```

When the state is updated with the recommended list, the component must create elements based on the items of that list so that the user can select the next product to see, as seen in fig. 5.6. Thus, whenever the list of recommendation is not empty, the data of each product is retrieved by

searching for its key on the list of hotspots from the configuration file, and a clickable element is created, showing the product picture and its name, and if any element is clicked, the current open modal is closed, the viewer pans to the position of the clicked hotspot and in the end, the new selected modal opens, showing the product, restarting the recommendation process all over again, but for that selected item.

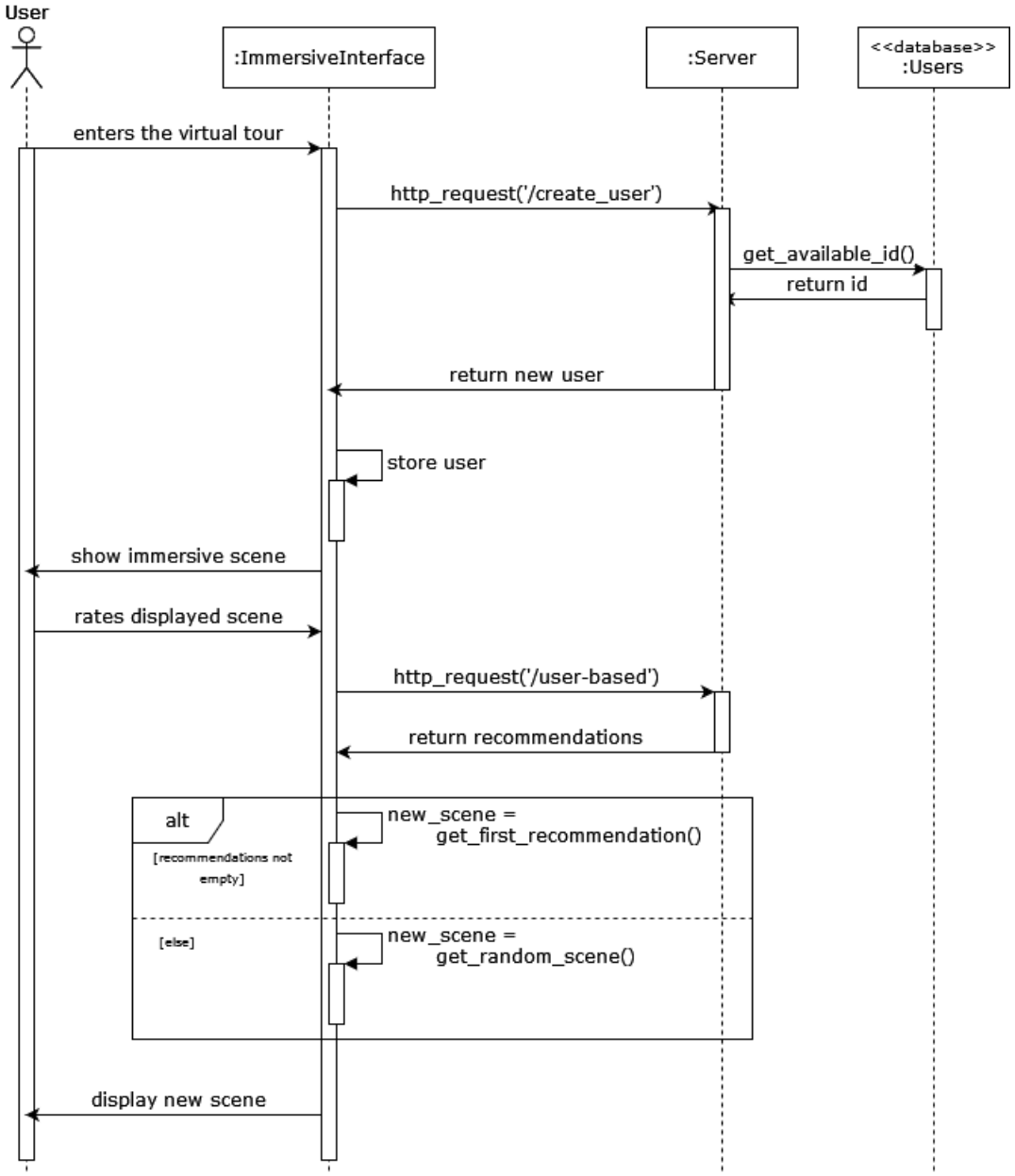
Figure 5.6: Creation of recommendation elements by iterating over the recommended items

```
let recommendedElements: JSX.Element[] = [];
if (recommended.length > 0 && products) {
  recommendedElements = recommended.map((r, i) => {
    const rec = products.find((p) => p.title === r);
    const index = rec ? products.indexOf(rec) : -1;
    return (
      <div
        key={`recommend-${i}-${props.id}`}
        className="recommended"
        onClick={() => {
          const tmp = state.hotspots;
          const idx = tmp.findIndex((h) => h.element.key === props.id);
          tmp[idx].active = false;
          setRecommended([]);
          setState({ ...state, hotspots: tmp, autoplay: false });
          rec &&
            props.viewer.lookTo(rec.yaw, rec.pitch, () => {
              const tmp = state.hotspots;
              const idx = tmp.findIndex(
                (h) => h.element.key === `modal-${state.scene}-${index}`
              );
              tmp[idx].active = true;
              setState({ ...state, hotspots: tmp, autoplay: false });
            });
        }}
      >
        <img src={rec?.image} alt="" />
        <div className="title">{rec?.title}</div>
      </div>
    );
  });
}
```

5.3 Recommendations using Collaborative Filtering

Collaborative filtering was used to recommend spaces of interest to the user in order to improve the user journey by suggesting spaces that similar users have liked previously. Overall, this process uses a rating template that allows the user to rate from 1 to 5 the image that they are seeing. When the user rates the space, the platform sends to the server, through HTTP, the rating and the server returns a list of recommended spaces, from which the platform selects the best recommendation and shows that space to the user. This filtering sequence is illustrated in fig. 5.7.

Figure 5.7: Recommendations using collaborative-based filtering sequence diagram



Since this is a user-based recommendation, there is the need to create users to keep their

history. Hence the Alive Vision platform lacks a login feature and a users database, a basic system of storing the user ID, generated by requesting an available ID to the server, in the browser local storage was implemented, as observed in fig. 5.8. In order to create the user, the system checks if there is a user in the local storage and, if there isn't, the platform performs a HTTP request by using the fetch function and when a response is given, the local storage is updated with the returned ID and an empty list of visited spaces is initialized, which is later used to confirm if any recommended space was already visited.

Figure 5.8: Creation of a user

```
useEffect(() => {
  const storage: Storage = window.localStorage;
  if (configs.recommendations && !storage.getItem("aLIVE_user")) {
    fetch("http://localhost:8080/create-user", {
      method: "GET",
      headers: {
        Accept: "*/*",
        "Content-Type": "application/json",
      },
    })
      .then((response) => {
        if (response.status === 200) {
          return response.json();
        } else {
          throw new Error("Something went wrong on api server!");
        }
      })
      .then((response) => {
        storage.setItem("aLIVE_user", response);
        storage.setItem("aLIVE_visited", JSON.stringify([]));
      })
      .catch((error) => {
        console.error(error);
      });
  }
}, []);
```

All the action happens within the navigation bar template, which is a simple bar that contains five star buttons that the user can select to indicate the chosen rate for that image. Each star button, when clicked, sends the HTTP request to the server, which, as mentioned previously, is the one running on localhost due to the lack of a deployment, sending, as the body of the request, the rate associated to the star, the user ID obtained from the local storage, and the ID of the media being shown, as seen in fig. 5.9.

After making the request, the system waits for a response with the list of recommendations, being this list sorted from the best recommendation to the worst. Therefore the only item from that list that matters to this use case is the first one. Once the response arrives, it is determined

Figure 5.9: Request of the recommended list by sending a rate

```
const user = window.localStorage.getItem("aLIVE_user");
user &&
  fetch("http://localhost:8080/user-based", {
    method: "POST",
    headers: {
      Accept: "*/*",
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      user: user,
      item: state.scene,
      rate: rate,
    }),
  })
    .then((response) => {
      if (response.status === 200) {
        return response.json();
      } else {
        throw new Error("Something went wrong on api server!");
      }
    })
```

if any recommendation exists or if the list is empty. In the first case, since the list of recommendations has at least one recommendation, a second confirmation is made, which is to check if the scenes recommended do, in fact, exist in the virtual tour, by iterating the scenes presented in the configuration file and determining if the ID of the recommended items matches the ones on the file. If the virtual tour contains the recommendations (fig, 5.10), then the global state, from the explorer component, is updated the current image with the image id of the first element of the list of recommendations. If the recommendations don't exist in the virtual tour, it means that all the recommendations should be discarded, therefore being the same as the second case where the recommendation list is empty. In the second case (fig. 5.11), to prevent the flow of the use case to stop, a random scene is chosen to present to the user. This random scene is selected by picking a random scene from the configuration files, and if the visited list from the local storage contains that scene, a new random one is picked until a scene that was never seen by the user appears. In the case of the visited list being empty, the next scene is randomised until the random scene is different from the current scene. In the end, the state is updated with the new random media, showing it to the user.

This operation only ends when a new image is shown. Therefore, whenever a new scene is selected, either randomly or by being recommended, the platform must always change the current media, allowing the user to explore the new image and rate it to restart this recommendation process.

Figure 5.10: Scene update in case of a successful recommendation

```
let recommendationList: string[] = response;
console.log(response);
if (recommendationList.length > 0) {
  const contains = recommendationList.filter(
    (r) => state.data.scenes.find((s) => r === s.id) !== undefined
  );
  if (contains.length > 0) {
    setState({ ...state, scene: recommendationList[0] });
```

Figure 5.11: Scene update in case of a failed recommendation

```
} else {
  //get random panorama not seen;
  const tmp = window.localStorage.getItem("aLIVE_visited");
  const visited: string[] = tmp ? JSON.parse(tmp) : [];
  let randomScene: MarzipanoScene =
    state.data.scenes[
      Math.floor(Math.random() * state.data.scenes.length)
    ];
  if (visited.length < state.data.scenes.length) {
    while (visited.includes(randomScene.id)) {
      randomScene =
        state.data.scenes[
          Math.floor(Math.random() * state.data.scenes.length)
        ];
    }
  } else {
    while (randomScene.id === state.scene) {
      randomScene =
        state.data.scenes[
          Math.floor(Math.random() * state.data.scenes.length)
        ];
    }
  }
  setState({ ...state, scene: randomScene.id });
  console.log("the recommended items do not exist");
}
```

5.4 Summary

To sum up, in order to integrate the recommendation system, 3Decide's platform, called Alive Vision light, was used as the immersive environment, which generates virtual tours using 360-degree media. The filtering processes from the server were applied to the recommendation of products (content-based) whenever a user was looking at an item, and to the recommendation of new spaces (collaborative) for the user to explore in order to improve the user journey. For the sake of obtaining either recommendations, the platform resort to the fetch function built-in typescript to make HTTP requests to the server where the recommendation system was located. The collaborative filtering required the existence of users. Therefore, the local storage from the browsers was used to store and retrieve users' ids.

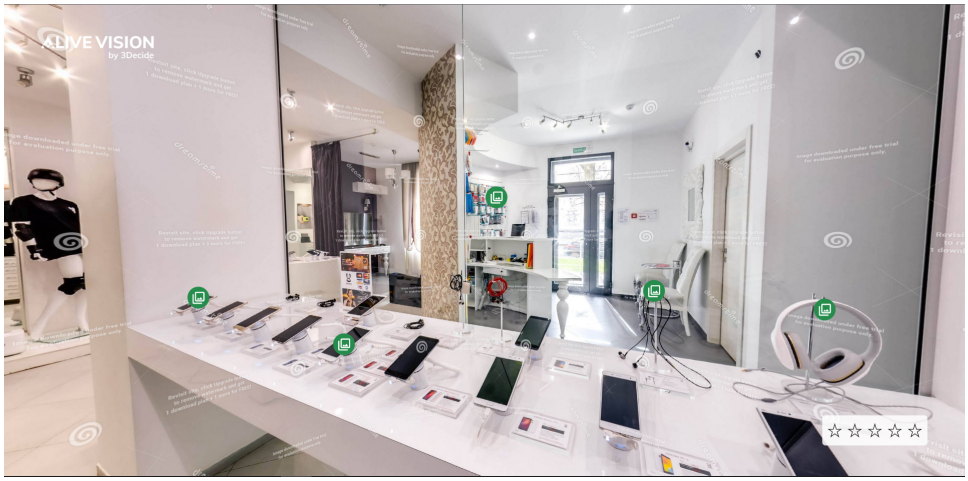
Chapter 6

Results

In order to test the solution developed, described in the previous chapters 4 and 5, a virtual tour using the 3Decide's Alive Vision platform was created. This experience uses 6 different 360-degree images to create 6 scenes that the user is able to explore. Each image, obtained from an online platform of images in order to create a more real environment, however, do not represent any real store, corresponds to a different store. For every scene, 10 products were introduced, based on items from real store to obtain concrete descriptions for the recommendations, that the user can inspect and interact with. Therefore, the virtual tour is composed by a smartphone store, a jewellery store, a shoe store, a bike store, a clothes store, and a gadget store.

In fig. 6.1, it is possible to see a scene from the virtual tour created, in this case being the smartphone store, where it shows the rate bar in the bottom right corner and the different hotspots as green buttons, each corresponding to different products.

Figure 6.1: Scene of the virtual tour created



Each filtering algorithm was also evaluated by using a precision method for the content-based filtering and an error metric for the collaborative filtering. Both evaluation methods used different

datasets to measure each metric. The datasets use real data from real users that rated real movies in order to obtain a more realistic evaluation without doing user testings.

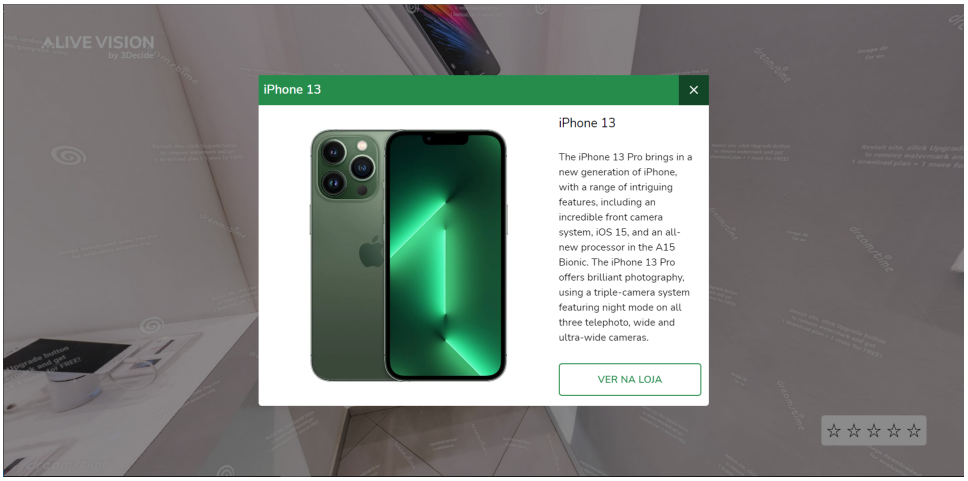
This chapter will show how each objective proposed was fulfilled using the virtual tour created. For each objective the result of the filtering will be explained and the resulting interface will be displayed as well. The results of the recommendations' measurements using the mentioned methods will also be presented.

6.1 Recommendation of Products

Products were recommended by using the content-based filtering. For that, a list of all the available products were gathered to be sent to the server, along with the item to compare to, then the virtual tour waits for the server to respond with the recommended list.

To start the recommendation request, the user must first open any modal with a product by clicking on any hotspot from the scene. When the user clicks on the desired hotspot, a modal with the product information is displayed so the user can read it, as seen in fig. 6.2.

Figure 6.2: Product modal opened before any recommendation is made



It is assumed that a user likes a certain product if they spend some time looking at it. Therefore if the modal stays open for at least 3 seconds, the immersive environments sends the HTTP request to the server asking for recommendations based on the product that is being seen and the gathered list of all products.

Once the server receives the request, it starts the content-based filtering algorithm having as parameters the list of products and the selected item. Each product is turned into vectors, resulting in a matrix where each line is a product description and each column is the vector and the TF-IDF value for each word in the description, as it is partially observed in fig. 6.3.

After obtaining the vectors, the server uses them to calculate the cosine similarity between items, calculating a value between 0 and 1 that corresponds to the similarity between the descriptions, being 1 the value for equality between texts. Fig. 6.4 shows the results after calculating the

Figure 6.3: List of items received by the server and corresponding transformation to vectors

	title	description
0	Samsung Galaxy A53	With its fantastic Super AMOLED 120Hz Infinity...
1	Samsung Galaxy S22	With its new look incorporating an armour alum...
2	iPhone SE	The iPhone SE (2022) is a brilliant cheaper sm...
3	iPhone 13	The iPhone 13 Pro brings in a new generation o...
4	Samsung Galaxy Covers	This smart LED cover is the perfect way to pro...
5	Samsung Galaxy Watch4	Get to know your body composition with the Sam...
6	Fitbit Versa 3	Fitbit smartwatches are designed to keep you f...
7	Apple Watch Series 7	The Apple Watch 7 is the perfect way to keep u...
8	Apple Beats	In your ears or around your neck, Apple Beats ...
9	Apple AirPods Max	With the AirPods Max, Apple has reimagined wha...

	0
0	(0, 278)\t0.11497259833177073\n (0, 265)\t0...
1	(0, 41)\t0.13853546275110357\n (0, 68)\t0.1...
2	(0, 113)\t0.16232172869080272\n (0, 174)\t0...
3	(0, 259)\t0.1528940469376744\n (0, 169)\t0....
4	(0, 129)\t0.24422905875722933\n (0, 183)\t0...
5	(0, 252)\t0.159409310227058\n (0, 234)\t0.1...
6	(0, 194)\t0.14295713329410778\n (0, 179)\t0...
7	(0, 235)\t0.1653421591425025\n (0, 93)\t0.1...
8	(0, 36)\t0.17379064497481286\n (0, 193)\t0....
9	(0, 74)\t0.16925911442294336\n (0, 232)\t0....

similarity, where the lines and columns indexes correspond to the same product. Therefore the main diagonal of the matrix is always 1.

Figure 6.4: Matrix of similarity between products

	0	1	2	3	4	5	6	7	8	9
0	1.000000	0.100136	0.065615	0.142423	0.043326	0.037706	0.019148	0.009777	0.000000	0.010009
1	0.100136	1.000000	0.054373	0.101101	0.029586	0.041182	0.028273	0.022685	0.017399	0.063390
2	0.065615	0.054373	1.000000	0.127981	0.028649	0.000000	0.026915	0.124518	0.065440	0.090456
3	0.142423	0.101101	0.127981	1.000000	0.000000	0.000000	0.009557	0.000000	0.000000	0.018701
4	0.043326	0.029586	0.028649	0.000000	1.000000	0.079201	0.000000	0.058364	0.000000	0.000000
5	0.037706	0.041182	0.000000	0.000000	0.079201	1.000000	0.016468	0.090767	0.000000	0.000000
6	0.019148	0.028273	0.026915	0.009557	0.000000	0.016468	1.000000	0.061578	0.035908	0.034972
7	0.009777	0.022685	0.124518	0.000000	0.058364	0.090767	0.061578	1.000000	0.045893	0.048944
8	0.000000	0.017399	0.065440	0.000000	0.000000	0.000000	0.035908	0.045893	1.000000	0.068237
9	0.010009	0.063390	0.090456	0.018701	0.000000	0.000000	0.034972	0.048944	0.068237	1.000000

In the end of the filtering process, the server obtains the indexes of the list and fetches from the similarity matrix the line that corresponds to the selected item, sorting from highest similarity to lowest, resulting in an array of products (fig. 6.5) which represent the recommendation list, where the first item is the selected product and the last item is the least similar one. This result list is the response sent back to the requester, which is the immersive environment.

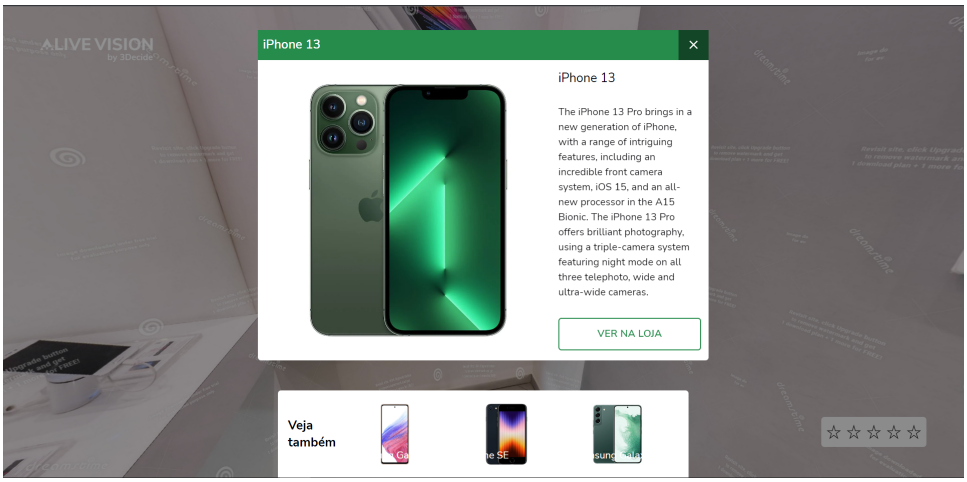
When the virtual tour receives the list with the recommendations, it shows to the user a section with the top 3 recommendations, ignoring the first since it is the same as the current product being seen, as shown in fig. 6.6. This section allows the user to click on any recommended item and the virtual tour, when a recommendation is selected, jumps to that product, closing the one opened,

Figure 6.5: List of products sorted by highest similarity to lowest

3	iPhone 13
0	Samsung Galaxy A53
2	iPhone SE
1	Samsung Galaxy S22
9	Apple AirPods Max
6	Fitbit Versa 3
4	Samsung Galaxy Covers
5	Samsung Galaxy Watch4
7	Apple Watch Series 7
8	Apple Beats

moving to the location of the item, and opening the new modal.

Figure 6.6: Product modal opened after new items were recommended



6.2 Recommendation of Spaces

In order to improve the user journey, it was presupposed that suggesting new scenes that the user might like would solve the problem. Therefore, since each image is not accompanied by any description that permits comparison with other images, besides the image itself, it was decided that the best approach would be to use collaborative filtering. In this sense, a rating bar that allows the user to tell how much they like the scene they are watching was created.

The recommendation process starts when the user rates the scene by clicking any star from the rate bar, which translates into a rate value, as observed at the bottom right bar of fig. 6.7 where the number of highlighted stars corresponds to the rate the user wants to give. Whenever the user clicks on any star, the platform sends to the server the rate the user fetched from the local storage has given to the scene they are seeing, expecting that the server will return a list of scenes recommended.

Figure 6.7: Rating bar showing the rate value

When the server receives the request for scenes, it starts by inserting the user rate in the database, then it obtains the list of all users' rating from that database (fig. 6.8) so that it can determine the similar users, to later predict the rating of the others items based on the history of the other users.

Figure 6.8: List of users' ratings from the database

item	0-store-smartphone	1-store-jewelry	2-store-shoes	3-store-bikes	4-store-clothes	5-store-drones
user						
user_0	4.0	1.0	2.0	3.0	4.0	5.0
user_00	5.0	1.0	2.0	3.0	3.0	4.0
user_01	5.0	1.0	2.0	3.0	3.0	4.0
user_02	5.0	1.0	2.0	3.0	3.0	4.0
user_03	5.0	1.0	2.0	3.0	3.0	4.0
...	...	...	...	...	...	...
user_57	4.0	1.0	2.0	4.0	3.0	5.0
user_58	4.0	1.0	2.0	4.0	3.0	5.0
user_59	4.0	1.0	2.0	4.0	3.0	5.0
user_7559	4.0	NaN	2.0	3.0	2.0	3.0
user_8681	3.0	NaN	NaN	NaN	NaN	NaN

Once the server has the ratings of all users, it gets the list of scenes and for each one, if the user hasn't rated that scene, it tries to predict the rating the user would give.

Anytime the server tries to predict a rating, it finds the most similar users based on the rating, and based on the similarities values and the user mean rating, it calculates a prediction. In fig. 6.9 it is possible to see on the left side the similarity values and on the right the corresponding user index. In this case, the similarities are close to 1 for every user because there are few spaces and a lot of users, in comparison, therefore there are groups of users that have rated equally each space meaning that however the user rate each space there is a high probability that it will match the rates by those groups, generating high similarity scores.

After predicting for each item and storing the prediction in a list, that list is sorted from the highest prediction to the lowest (fig. 6.10), returning that list to the requester, meaning that that

Figure 6.9: Similarities

```
[1.         0.96590921 0.96590921 0.96590921 0.96590921 0.96590921
 0.96590921 0.96590921 0.96590921 0.96590921 0.96590921] [[49 42 51 50 48 47 46 45 44 43 52]]
```

list is the recommendations based on the users' history.

Figure 6.10: Sorted list with all predictions

```
[
  {'item': '4-store-clothes', 'prediction': 5},
  {'item': '2-store-shoes', 'prediction': 3},
  {'item': '3-store-bikes', 'prediction': 3},
  {'item': '1-store-jewelry', 'prediction': 2},
  {'item': '5-store-drones', 'prediction': 2}
]
```

The virtual tour, when it receives the response from the server with the recommended list, jumps to the best recommendation which is the first of the list. Fig. 6.11 shows a new image after the user rated the previous scene. In this case, it shows the clothes store due to the fact that the server returned that scene with the highest prediction.

Figure 6.11: New scene after rating

Following the completion of the recommendation process and the image changes, the user is free to explore the new scene and restart the recommendation by clicking the desired star from the rate bar.

6.3 Recommendation of Points of Interest

To recommend points of interest, two solutions were tested, however none of them were successful. Since Alive Vision platform doesn't have a feature to detect points of interest it was difficult to implement a filtering for it.

The first solution experimented was to fire the recommendation request whenever the user looked at the same area for some time. Nevertheless, since each scene is composed by numerous points, it was hard to keep track of those areas to recommend other points of interest, therefore this solution couldn't be applied because it was impossible to store on the database the ratings for each area seen to use collaborative filtering, and those points didn't have any content so it was not possible to use content-based filtering.

The second solution was to use informational hotspots, which required the creator of the tour to select the areas of interest and write descriptions about them. This solution could work with any filtering (content-based or collaborative) since it was possible to compare the description of each hotspot and use the content-based filtering to obtain similar points of interest, and it was also possible to persist in the database a rating based on the time the user spent with the hotspot open, which means that if the user didn't spend any time looking at it, it was a low rating and if they spent a considerable amount of time it was a high rating. Therefore,collaborative filtering could be used as well. However, this solution would fill the immersive environment with informational buttons, hiding most part of the image and covering part of the area of interest that the hotspots were trying to highlight, which is not ideal considering the immersive aspect of the experience. For that reason this solution was discarded.

The proposed solution would be to improve the platform with a feature to include regions that might be of interest, allowing the users to look at them while exploring and if they stop at the area of interest for a significant amount of time, it would be considered that the point of interest would be rated highly by the user. Therefore it would be possible to use collaborative filtering to recommend other areas. This solution could also be improved by using machine learning to detect possible regions of interest in the image instead of having to add those regions to the virtual tour while creating it.

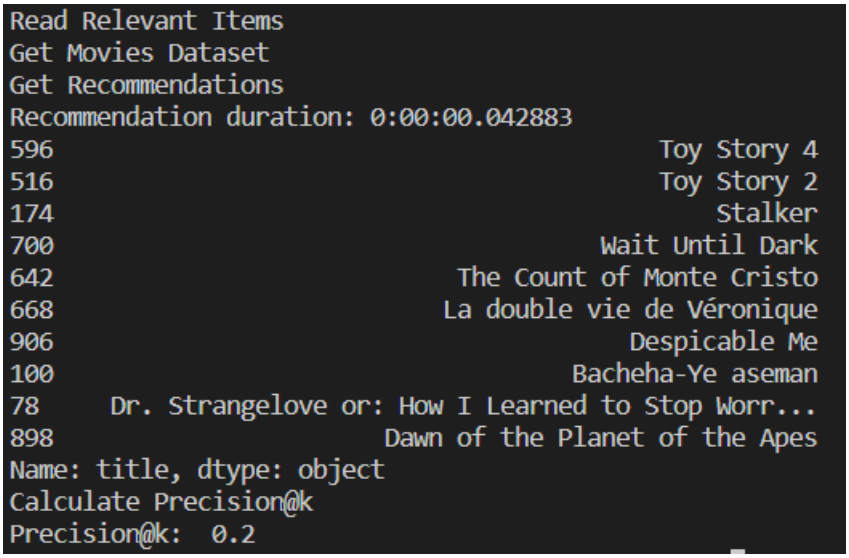
6.4 Measurement of the Recommendations

For measuring each recommendation filtering, a function for each evaluation method described in chapter 3 was developed. Precision@N, see eq. 3.1, was used to evaluate the performance of the content-based filtering and root mean square error, see eq. 3.2, was used to measure the execution of the user-based filtering.

Content-based filtering used a dataset of movies that contained the top 1000 movies from IMDb along with each movie description [22]. This list of movies is the base list of items used to get the top 10 recommendations from the content-based algorithm and, for testing purposes, the movie Toy Story was selected to obtain the 10 similar movies. The relevant items, for the precision@N method, were previously selected from the list of similar items from the IMDb's Toy Story page [23]. In order to calculate the precision@N value, the number of recommended movies returned from the content-based filtering function that were also in the relevant movies list was calculated.

Figure 6.12 shows the results obtained by using the developed precision@N function. The content-based filtering algorithm had a precision of 20%, and had a duration of less than 1 second. This results show that the algorithm has a poor performance, despite the duration being extremely good.

Figure 6.12: Results of Precision@N method



The poor precision of the content-based filtering may be due to the use of only the description to calculate the similarity, which means that the algorithm needs the descriptions to have a lot of details to return more viable results, which is not the case of the descriptions of the dataset. Therefore, the content-based filtering needs to be improved to support other properties of the items in order to get better results. For example, for the movies, it could also include the genre and target audience. Other similarity methods could also be applied that could improve the similarity values between items, which would improve the precision, resulting in a better algorithm.

Collaborative filtering used a different dataset of movies that involved 100000 ratings by 943 users on 1682 movies [9]. The root mean square error function developed used a test database to store the users' ratings from the dataset. While storing the data, part of the ratings of users 1 to 20 were not inserted in the database in order to have real ratings to compare to the predictions, meaning that those users were used to predict the ratings.

Figure 6.13 shows the collaborative filtering performance based on the root mean square error metric. The algorithm had a root mean square error of between 0.89 and 2.00 for each user and predicting the rating of more than 1000 items for 20 users took 9 minutes and 23 seconds, averaging 28.15 seconds per user. The performance of this filtering is considered poor, both the error and the duration of the prediction calculations.

The reason the performance of the collaborative filtering is so poor, despite some values being acceptable, is the lack of ratings of the dataset. Therefore, if the dataset had more ratings for the user, more ratings could be inserted, and the errors would be smaller, because the more ratings the user has made, the more accurate the predictions will be. However, the algorithm could still

Figure 6.13: Results of Root Mean Square Error method

```
Prediction duration for 20 users : 0:09:23.022479
RMSE Calculation
RMSE(u1) = 1.53
RMSE(u2) = 1.32
RMSE(u3) = 1.59
RMSE(u4) = 1.46
RMSE(u5) = 1.91
RMSE(u6) = 1.46
RMSE(u7) = 1.15
RMSE(u8) = 1.57
RMSE(u9) = 1.11
RMSE(u10) = 0.89
RMSE(u11) = 1.41
RMSE(u12) = 0.91
RMSE(u13) = 1.57
RMSE(u14) = 1.22
RMSE(u15) = 1.78
RMSE(u16) = 1.17
RMSE(u17) = 2.00
RMSE(u18) = 1.15
RMSE(u19) = 1.82
RMSE(u20) = 1.90
```

be improved by using better parameters when calculating the nearest neighbors, for example, test other metrics, such as the Minkowski distance.

6.5 Summary

In conclusion, it was possible to create an immersive environment that uses a recommendation system to improve the experience. Although the virtual tour created is only a prototype, it included a feature to recommend products and a feature to recommend other spaces to improve the user journey. The recommendation of points of interest was not implemented due to the fact that the platform used didn't support the inclusion of points of interest and a solution was not found. In spite of the successful creation of the virtual tour with the recommendation features, the filtering algorithms implemented didn't have the same successful evaluation, which means that both algorithms need improvements.

Chapter 7

Conclusions

3Decide has developed a platform that creates immersive tours with the use of 360-degree images, which raised a problem due to the freedom the users have in this type of experiences, where the user may miss important information. Nevertheless, that might contribute to the increase of the user engagement, which benefits both the user and the creator.

The solution consisted in developing recommendation systems that suggest places the user might find interesting. However there is little research on recommendation systems applied to immersive environments, despite the numerous systems applied in online platforms, namely e-commerce platforms, therefore the idea was to develop e-commerce recommendation systems on immersive experiences.

The recommendation system was developed using Python, and two algorithms that returned recommendations were created - a content-based filtering algorithm and a collaborative one. The recommendations could be retrieved by sending HTTP requests to a server, also developed in Python, where both filtering algorithms were running. The content-based filtering used a cosine similarity method to obtain the 10 most similar items, while the collaborative filtering used the k-Nearest-Neighbors algorithm to obtain the most similar users and predict the ratings. The collaborative filtering uses a database of users' ratings to store each users' history and use it to make the predictions.

3Decide's platform was used as the base immersive environment to integrate the recommendation system. This platform supports the use of immersive elements that can show information which were used for the recommendation of items via content-based filtering. These elements, when opened, would show similar elements by requesting them, sending a HTTP request to the recommendation system server. The collaborative filtering was applied to each scene of the virtual tour generated by the platform. The user could rate the image they were seeing and the virtual tour would request spaces, also by sending a HTTP request to the server.

To conclude, it was possible to create an immersive environment where recommendations could be made to the user while exploring that experience. The solution where a server side

recommendation system is used to assist an immersive experience was successfully implemented and can be applied to most environments. This solution allowed the creation of immersive e-commerce where e-commerce products could be shown to the users while they were shopping, and also allowed the creation of experiences where spaces of interest could be recommended to the user to avoid missing them, improving the user journey.

7.1 Future work

Despite the successful integration and creation of an immersive virtual tour with a recommendation system, the filtering algorithms revealed poor performances. Therefore it is necessary to improve the algorithms by testing different methods of obtaining similarities and carrying out more performance tests with different datasets. It is also important to make user testing in order to really test the performance, since the metrics used to the evaluation are purely statistical, and, for that reason, live testing could return better results.

Since all proposed objectives where not completed, a solution for the recommendation of points of interest also needs to be developed in the future. The proposed solution to this feature is to implement the possibility of defining regions beforehand that might be of interest and use collaborative filtering to suggest the ones that might attract the user.

One day, it would be interesting to see this solution applied to other immersive platforms, like virtual reality and augmented reality, since both offer a greater immersive level than the 360-degree virtual tours. These platforms could use the recommendation system server developed, and use HTTP requests to integrate the recommendations on their experiences considering that there are already engines that support this type of requests.

Appendix A

Content-based filtering

```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd


def content_based_filtering(list, item):
    df = prepare_data_frame(list)
    tfidf_matrix = vectorize_matrix(df)
    cosine_sim = calculate_cosine_similarity(tfidf_matrix)
    indices = get_indices(df)
    rec_indices = calculate_similarities(item, indices, cosine_sim)
    return df['title'].iloc[rec_indices]


def prepare_data_frame(list):
    df = pd.DataFrame(list)
    df = df[['title', 'description']]
    return df


def vectorize_matrix(df):
    tfidf = TfidfVectorizer(stop_words='english')
    df['description'] = df['description'].fillna('')
    tfidf_matrix = tfidf.fit_transform(df['description'])
    return tfidf_matrix


def calculate_cosine_similarity(tfidf_matrix):
```

```
    cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
    return cosine_sim


def get_indices(df):
    df = df[~df['title'].isna()]
    indices = pd.Series(df.index, index=df['title'])
    indices = indices[~indices.index.duplicated(keep='last')]
    return indices


def calculate_similarities(item, indices, cosine_sim):
    target_index = indices[item]
    similarity_scores = pd.DataFrame(
        cosine_sim[target_index], columns=["score"])
    item_indices = similarity_scores.sort_values(
        "score", ascending=False)[0:11].index
    return item_indices
```

Appendix B

Collaborative filtering

```
import math
import operator
import pandas as pd
import numpy as np
from sklearn.neighbors import NearestNeighbors

from sqlite import list_all


def user_based_fitering(conn, user):
    recommendations = []
    pt = prepare_matrix(conn)
    if pt.empty:
        return []
    else:
        items = get_list_of_items(pt)
        user_idx = pt.index.values.tolist().index(user)
        k = len(pt.index.values) - 1
        if k > 10:
            k = 10
        for i in range(0, len(items)):
            if(math.isnan(pt.values[user_idx][i])):
                pred = predict_rating_item(user_id=user_idx, ratings=pd.DataFram
                    pt.values), item_id=i, k=k)
                recommendations.append(
                    {"item": pt.columns.values[i], "prediction": pred})
    return sorted(recommendations, key=operator.itemgetter(
        "prediction"), reverse=True)
```

```
def prepare_matrix(conn):
    list = list_all(conn)
    df = pd.DataFrame(list)
    if(df.empty):
        return df
    pt = pd.pivot_table(data=df, values="rate", index="user",
                        columns="item")
    return pt


def get_list_of_items(pt):
    return pt.columns.values


def find_k_similar_users(user_id, ratings, metric='cosine', k=4):
    ratings = ratings.loc[:, ~ratings.iloc[user_id].isna()]
    ratings = ratings.fillna(0)
    model = NearestNeighbors(metric=metric, algorithm='brute')
    model.fit(ratings)

    distances, indices = model.kneighbors(
        ratings.iloc[user_id, :].values.reshape(1, -1), n_neighbors=k+1)
    similarities = 1-distances.flatten()

    return similarities, indices


def predict_rating_item(user_id, item_id, ratings, metric='cosine', k=4):
    prediction = 0
    similarities, indices = find_k_similar_users(user_id, ratings, metric, k)
    ratings_tmp = ratings.loc[:, ~ratings.iloc[user_id].isna()]
    ratings = ratings.fillna(0)
    mean_rating = ratings_tmp.loc[user_id, :].mean()
    sum_wt = np.sum(similarities)-1
    product = 1
    wtd_sum = 0

    for i in range(0, len(indices.flatten())):
```

```
        if indices.flatten()[i] == user_id:
            continue
        else:
            ratings_diff = ratings.iloc[indices.flatten(
            )[i], item_id]-np.mean(ratings.iloc[indices.flatten()[i], :])
            product = ratings_diff * (similarities[i])
            wtd_sum = wtd_sum + product
    prediction = int(round(mean_rating + (wtd_sum/sum_wt)))
    return prediction
```

Appendix C

Work Plan

In fig. C.1 it is possible to see the plan. The project will last until the end of May and will have four main tasks:

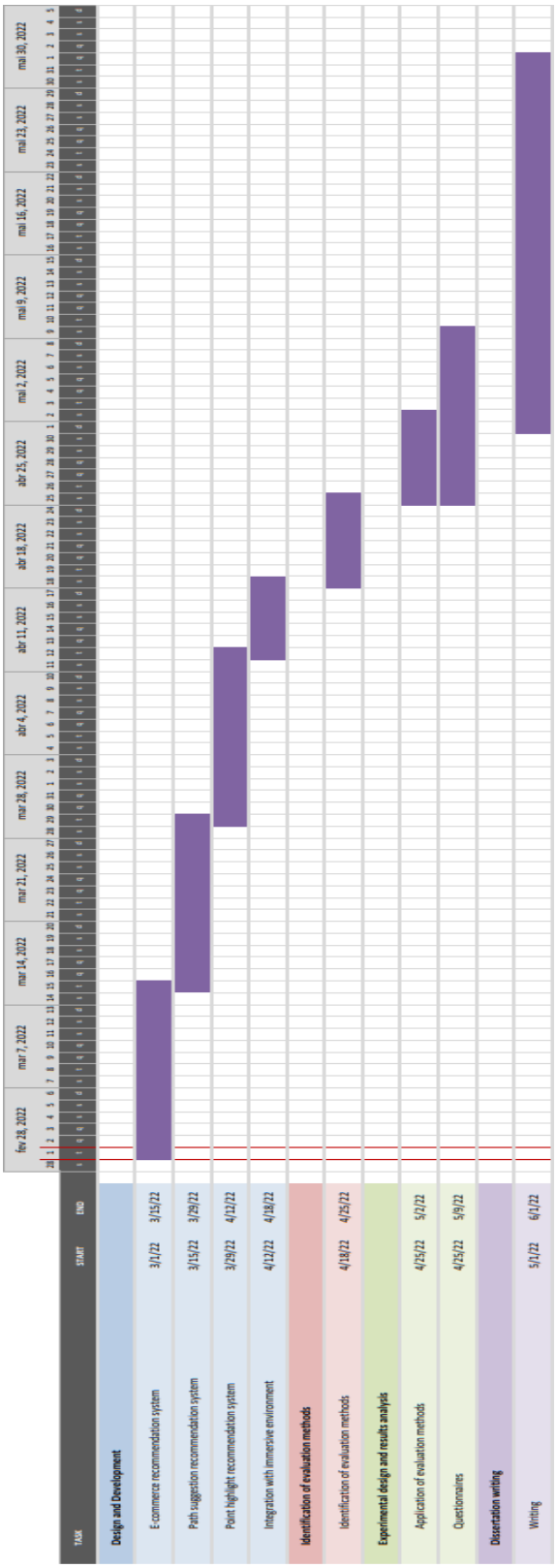
1. Design and development: this phase is divided into four sub tasks where the first three correspond to the development of each recommendation system, and will last 14 days each, while the last sub task is to integrate the recommendation system with a virtual tour and create an immersive environment with a recommendation system.

2. Identification of evaluation methods: this phase will have a simple task to be completed within one week which is to search and deeply explore evaluation methods that are relevant to the developed recommendation systems.

3. Experimental design and results analysis: this phase will be divided into two tasks and will last 14 days. The first will use the evaluation methods found to measure the quality of the recommended items by the systems developed, and the second will depend on user testing where some questionnaires will be prepared and groups of people will test the experience developed and answer the questions afterwards.

4. Dissertation writing: the last phase will fall on the writing of the dissertation and despite it is planned to be written in the last month, in fact the dissertation should be written along the development of the project.

Figure C.1: Gantt chart



References

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. *On the surprising behavior of distance metrics in high dimensional space*, volume 1973 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. IBM T.J. Watson Research Center, 2001.

[2] Ali Anaissi and Madhu Goyal. Svm-based association rules for knowledge discovery and classification. In *2015 2nd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, pages 1–5, 2015.

[3] Leila Cristina C. Bergamasco and Fátima L.S. Nunes. Intelligent retrieval and classification in three-dimensional biomedical images — a systematic mapping. *Computer Science Review*, 31:19–38, 2019.

[4] Amitabh Biswal, Malaya Dutta Borah, and Zakir Hussain. Chapter eleven - music recommender system using restricted boltzmann machine with implicit feedback. In Shiho Kim and Ganesh Chandra Deka, editors, *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, volume 122 of *Advances in Computers*, pages 367–402. Elsevier, 2021.

[5] Andrea Bönsch, David Hashem, Jonathan Ehret, and Torsten W. Kuhlen. *Being Guided or Having Exploratory Freedom: User Preferences of a Virtual Agent's Behavior in a Museum*, page 33–40. Association for Computing Machinery, New York, NY, USA, 2021.

[6] Yuri M. Brovman. Complementary item recommendations at ebay scale. Available at https://tech.ebayinc.com/engineering/complementary-item-recommendations-at-ebay-scale/, February 2019.

[7] D. Contreras, M. Salamó, I. Rodríguez, and A. Puig. An approach to improve user experience with conversational recommenders through a 3d virtual environment. In *ACM International Conference Proceeding Series*, volume 10-12-September-2014, 2014. Cited By :3.

[8] Chandramouli Das, Abhaya Kumar Sahoo, and Chittaranjan Pradhan. Chapter 12 - multi-criteria recommender system using different approaches. In Sushruta Mishra, Hrudaya Kumar Tripathy, Pradeep Kumar Mallick, Arun Kumar Sangaiah, and Gyoo-Soo Chae, editors, *Cognitive Big Data Intelligence with a Metaheuristic Approach*, Cognitive Data Science in Sustainable Computing, pages 259–277. Academic Press, 2022.

[9] Prajit Datta. Movielens 100k dataset. Available at https://www.kaggle.com/datasets/prajitdatta/movielens-100k-dataset.

[10] A. Da'u and N. Salim. Recommendation system based on deep learning methods: a systematic review and new directions. *Artificial Intelligence Review*, 53(4):2709–2748, 2020. Cited By :23.

[11] F. De Canio, E. Martinelli, M. Peruzzini, and G. Marchi. The use of virtual tours to stimulate consumers' buying and visit intentions: an application to the parmigiano reggiano cheese. *Italian Journal of Marketing*, 2021(3):209–226, Sep 2021.

[12] L. Einav, C. Farronato, and J. Levin. Peer-to-peer markets. *Annual Review of Economics*, 8:615–635, 2016. Cited By :210.

[13] Alexander Felfernig, Lothar Hotz, Juha Tiihonen, and Claire Bagley. Chapter 3 - configuration-related topics. In Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, editors, *Knowledge-Based Configuration*, pages 21–27. Morgan Kaufmann, Boston, 2014.

[14] Fethi Fkih. Similarity measures for collaborative filtering-based recommender systems: Review and experimental comparison. *Journal of King Saud University - Computer and Information Sciences*, 2021.

[15] Achraf Gazdar and Lotfi Hidri. A new similarity measure for collaborative filtering based recommender systems. *Knowledge-Based Systems*, 188:105058, 2020.

[16] I. Gironacci, K. Vincs, and J. Mccormick. A recommender system of extended reality experiences. In *ACM International Conference Proceeding Series*, pages 96–100, 2020. Cited By :2.

[17] I. M. Gironacci. Extended reality experiences prediction using collaborative filtering. In *SIGGRAPH Asia 2019 Doctoral Consortium, SA 2019*, 2019. Cited By :1.

[18] Jennifer Golbeck. Chapter 13 - social information filtering. In Jennifer Golbeck, editor, *Analyzing the Social Web*, pages 191–201. Morgan Kaufmann, Boston, 2013.

[19] Vagisha Gupta, Shelly Sachdeva, and Neha Dohare. Chapter 8 - deep similarity learning for disease prediction. In Vincenzo Piuri, Sandeep Raj, Angelo Genovese, and Rajshree Srivastava, editors, *Trends in Deep Learning Methodologies*, Hybrid Computational Intelligence for Pattern Analysis, pages 183–206. Academic Press, 2021.

[20] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2012.

[21] Jiawei Han, Micheline Kamber, and Jian Pei. 2 - getting to know your data. In Jiawei Han, Micheline Kamber, and Jian Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 39–82. Morgan Kaufmann, Boston, third edition edition, 2012.

[22] Omar Hany. Imdb top 1000. Available at https://www.kaggle.com/datasets/omarhanyy/imdb-top-1000?resource=download.

[23] IMDb. Toy story: Os rivais (1995) - imdb. Available at https://www.imdb.com/title/tt0114709/?ref_=fn_al_tt_1.

[24] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 2015.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, sep 1999.

[26] D. Jannach and M. Jugovac. Measuring the business value of recommender systems. *ACM Transactions on Management Information Systems*, 10(4), 2019. Cited By :30.

[27] D. Jannach, L. Lerche, I. Kamehkhosh, and M. Jugovac. What recommenders recommend: an analysis of recommendation biases and possible countermeasures. *User Modeling and User-Adapted Interaction*, 25(5):427–491, 2015. Cited By :97.

[28] M. Jugovac and D. Jannach. Interacting with recommenders-overview and research directions. *ACM Transactions on Interactive Intelligent Systems*, 7(3), 2017. Cited By :67.

[29] V. H. Kamble and M. P. Dale. *Machine learning approach for longitudinal face recognition of children*, pages 1–27. Machine Learning for Biometrics: Concepts, Algorithms and Applications. Academic Press, 2022.

[30] A. Mahadevan and M. Arock. Integrated topic modeling and sentiment analysis: A review rating prediction approach for recommender systems. *Turkish Journal of Electrical Engineering and Computer Sciences*, 28(1):107–123, 2020. Cited By :3.

[31] Marzipano. Marzipano - a 360° viewer for the modern web. Available at https://www.marzipano.net/.

[32] I. Portugal, P. Alencar, and D. Cowan. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, 97:205–227, 2018. Cited By :244.

[33] W. Qu, K. . Song, Y. . Zhang, S. Feng, D. . Wang, and G. Yu. A novel approach based on multi-view content analysis and semi-supervised enrichment for movie recommendation. *Journal of Computer Science and Technology*, 28(5):776–787, 2013. Cited By :7.

[34] O. Sagi and L. Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4), 2018. Cited By :440.

[35] Chaavi Saluja. Collaborative filtering based recommendation systems exemplified.. | by chhavi saluja | towards data science. Available at https://towardsdatascience.com/collaborative-filtering-based-recommendation-systems-exemplified-ecbffe1c20b March 2018.

[36] J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1-2):115–153, 2001. Cited By :1143.

[37] Thiago Silveira, Min Zhang, Xiao Lin, Yiqun Liu, and Shaoping Ma. How good your recommender system is? a survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics*, 10(5):813–831, 2019.

[38] Brent Smith and Greg Linden. Two decades of recommender systems at amazon.com. *IEEE Internet Computing*, 21(3):12–18, 2017.

[39] R. Ureña, G. Kou, Y. Dong, F. Chiclana, and E. Herrera-Viedma. A review on trust propagation and opinion dynamics in social networks and group decision making frameworks. *Information Sciences*, 478:461–475, 2019. Cited By :156.

[40] Liwei Wang, Yan Zhang, and Jufu Feng. On the euclidean distance of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1334–1339, 2005.

[41] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*, pages 1–621. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, 2016.