DESIGN AND IMPLEMENTATION OF AN E-COMMERCE RECOMMENDER SYSTEM

VINCENT KYALO

J17-6650-2015

THIS PROJECT REPORT IS SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE OF THE MACHAKOS UNIVERSITY

2023

DECLARATION

I hereby declare that this project documentation is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at any university.

STUDENT:

Sign……. Date……

Name……

SUPERVISOR:

Sign……. Date……

Name……

ABSTRACT

Recommender systems are a part of our everyday interactions with computers. Personalized product recommendations increase customer satisfaction and improve product discovery. This has led to their widespread adoption in various industries. Amazon attributes 35% of its revenue to their recommendation engine. Up to 75% of what consumers watch on Netflix comes from the company's recommender system. It therefore follows that a lot of research on developing better recommendation techniques has been conducted in recent years. In Kenya, however, e-commerce websites have underestimated the value of a proper recommendation service. The purpose of this project is to analyze the various techniques used in e-commerce recommender systems. Recent publications in the field are examined and existing issues in recommender systems are identified. The researcher will then design and develop an e-commerce web application and apply one of the recommendation techniques to develop product recommendations. Final outcomes will enable small scale e-vendors to understand the implementation of a recommender system from start to finish and provide insights to academicians and practitioners.

DEDICATION

To my family, for their abundant support throughout my academic journey.

ACKNOWLEDGEMENT

I am sincerely grateful to Mr. Erick Omuya for his excellent supervision, support and understanding throughout this project.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

E-commerce is the exchange of products/services and payments through telecommunication systems. In Kenya, the advent of e-commerce is informing how we shop and increasingly outsizing the impact of traditional retail. In 2014, information by the Communications Authority of Kenya (CAK) put the value of B2C e-commerce in Kenya at Sh4.3 billion. With the proliferation of mobile data services such as m-banking, mobile payment platforms such as M-Pesa and affordability of handsets from the Asian market, there is a growing consensus that e-commerce in the country will continue to grow. With investors rushing to secure their fair share of this market, there is fierce competition in the field. In order to increase sales, e-vendors need to effectively provide relevant content to their consumers. This can be done through recommender systems. A Recommender System (RS) can be defined as an information filtering system that predicts users' preferences and recommends potentially interesting items to these users (Adomavicius & Tuzhilin, 2005). Amazon.com is the epitome of successfully implemented recommender systems, attributing 35% of their sales revenue to their recommendation service. RSs are explored in more detail in the next chapter.

1.2 Problem Statement

E-commerce offers businesses logistical flexibility, allowing them to partner on a global scale with suppliers and operate with minimum capital. As a result, businesses can provide a great variety of products online. While such exhaustive catalogues are often useful for comparisons, they pose a challenge upon the user of sifting through millions of products. In order to find a given product online, users often make use of queries in search engines. While e-commerce vendors have made efforts to improve searching tools in their websites, these tools are only as good as the query the user formulates.

When searching for products online, most users do not know exactly what they are looking for, they have conflicting desires about what features their ideal product should have. In fact, psychological studies have shown that people construct their preferences as they learn about the available products (Payne, 1993). Customers are not able to discover new products that they may

like through traditional search tools. For e-vendors, this translates to a lack of customer satisfaction and ultimately low sales revenue. These problems can be addressed using recommender systems. This study describes the design and implementation of such a system and its integration into an e-commerce website.

1.3 Objectives of the Study

1.3.1 General Objective

To develop an E-commerce platform that offers users personalized product recommendations.

1.3.2 Specific Objectives

2. To provide an overview of the various recommendation techniques used and their challenges.
3. To provide users with personalized recommendations by implementing one of the recommendation techniques in e-commerce.
4. To allow users to shop online by designing an intuitive customer journey that responds appropriately to different screen sizes.
5. To enable the administrator to list their products & monitor orders by providing them with a secure portal.

1.4 Scope of the Study

This study will review some of the existing recommendation techniques used, develop a recommender system, and integrate it into an e-commerce website.

1. While various offline metrics can be used to evaluate RSs (e.g. accuracy, novelty, diversity) the only way to determine the success of a RS is to run online A/B tests with users. However, due to time limitations, that is beyond the scope of this study.
2. Although this paper aims to provide a practical approach to recommender systems, it is not built to operate at the massive scale of online giants. It also does not take into account privacy and ethical concerns. More detailed suggestions are made in chapter 6 of this study.

1.5 Justification of the Study

In recent years, researchers have focused their efforts on lowering the computational complexity of recommendation algorithms. As a result, studies in RS are usually crammed with technical jargon, offering little to none practical implementation. This study offers a more practical approach

to building recommender systems in e-commerce. It serves as a basis for small scale e-vendors in hoping to reap the benefits of RSs. Finally, it will serve as a basis for other researchers who are interested in recommendation technology.

1.6 Limitations of the Study

a) Machine Learning algorithms require extensive datasets in order to give accurate predictions. Obtaining this data will be no trivial task

b) The tools, languages & frameworks used in this study are bound to become obsolete. This is the inherent nature of software.

CHAPTER TWO

LITERATURE REVIEW

This chapter gives a brief history of recommender systems. It describes the purpose of RS and the various techniques employed in recommendation. It identifies some of the issues in RS and aims to give an overview of their application in e-commerce.

2.1 History of Recommender Systems

In their day-to-day activities, humans rely on recommendations for one purpose or the other. In ancient civilizations, this manifested itself in simple ways, such as what religion to profess and what crop to cultivate. (Sharma & Singh, 2016) Today, the choices available to us are manifold. While buying a cellphone, we visit numerous websites to check reviews on phones that might interest us. The same applies when we are going for movies, looking for books to read, stocks to buy, etc. We constantly seek recommendations from our peers, friends and family. It is therefore natural that we extend recommender systems to our interactions with computers.

Grundy (Rich, 1979) a computer-based librarian, represents an early entry into the RS domain. It was a fairly primitive system, grouping users into "stereotypes" by using short interviews. In the early 1990s, there was an overload in online information spaces and collaborative filtering was introduced to address this issue. *Tapestry* (Steven Lindell, 2009) was a manual collaborative filtering system that allowed users to query for items in an information domain based on other users' opinions or actions.These systems were soon succeeded by automated CF, automatically locating opinions and aggregating them to provide relevant opinions. *GroupLens* used this technique to show relevant *Usenet* articles to users. Automated CF led to development of various RSs, including *Ringo* for music, *Bellcore Videos Recommender* for movies and *Jester* for jokes (Ekstrand, Riedl, & Konstan, 2010).

By the late 1990s, there were commercial applications of Recommender Systems. Perhaps the most recognizable application of RS technologies is Amazon.com. Based on various factors, including purchase and browsing history, they recommend products to users. Research in RS was further catalyzed by the launch of the 2006 Netflix Prize, a one-million-dollar reward for research that could improve the accuracy of their recommendation algorithm by 10% (Bennett & Lanning, 2007).

2.2 Purpose of Recommender Systems

RSs have become widely used by some of the highest rated websites. (Amazon, YouTube, Netflix). LinkedIn, a social network for professionals, also uses a RS to offer you suggestions on people you might know. According to (F.Ricci et al, 2011), there are a number of reasons for this adoption. We highlight some of them here that pertain to the field of e-commerce.

1. Increase user fidelity – customer loyalty is important in e-commerce. In RS, the longer the user interacts with a website, the more refined their user model becomes. As the recommender output becomes more relevant more clients can be retained.
2. Selling more diverse items – e-vendors are interested in selling as much of their products as possible. RS enable the user to select items that might otherwise be hard to find.
3. Increasing the number of items sold – one of the main purposes of RS. They improve sales in e-commerce by recommending items that are likely to suit the user's needs and wants
4. User profiling – e-vendors can better understand users through analysis of RS data. This data can either be explicitly collected or predicted by the system.

2.3 Recommendation Approaches

2.3.1 Content Based Filtering

The system learns to predict items based on a user's previous activity. Items are recommended based on their attributes, as opposed to most techniques that rely on aggregate user behavior. For example, in a movie RS, we could use attributes such as the movie genre and the movie release date to recommend items to a given user. If Wambui enjoys watching romantic movies, then it may be reasonable to recommend to her other romantic movies.

Limitations of CB Filtering

1. Overspecialization – This is where a RS is not able to recommend unexpected, yet suitable items. For example, in the case of the movie RS above, a user who likes action movies could receive too many suggestions about the same genre.
2. Eliciting user feedback - This is where the quality of recommendations can only be improved by more historical data from the user. Unless the user above explicitly gives more feedback to the system, they will only receive suggestions for action movies.

2.3.2 Collaborative Filtering

The basic concept behind collaborative filtering (CF) is that two or more individuals with common interests in one area are inclined towards similar products/items in another (Sharma & Singh, 2016). The System predicts items for a particular user based on the items previously rated by other users (Adomavicius & Tuzhilin, 2005). CF techniques are widely used in building e-commerce RSs. As such, all the algorithms we implement in this study will be CF based.

Collaborative approaches overcome the problem of overspecialization in CB systems since they can recommend very different (novel) items as long as other users have already shown interest for these different items.

Collaborative approaches can be broken down into two general categories; *neighbourhood* and *model-based* approaches. In neighbourhood-based CF, the user-item ratings r stored in the system are directly used to predict ratings for new items. This can be done using either *user-based* or *item-based* recommendation. For user-based CF, the neighbors of a user u are the users v whose ratings r on I_{uv} are most correlated. In item-based approaches, we predict the rating of a user u for an item i based on the ratings of u for items similar to i. Two items are considered similar if several users of the system have rated them in a similar fashion. For this study, we implement both user-based and item-based KNN.

Model based approaches use ratings to learn a predictive model. The model is trained using the available data and later used to predict ratings for new users. They do not have the requirement of loading numerous data into memory and thus can be computationally efficient. There are numerous model based techniques including Bayesian Clustering, Singular Value Decomposition, Maximum Entropy, Support Vector Machines, etc. These are beyond the scope of this study.

Limitations of Collaborative Filtering

1. Cold start – where there are new users/items with little historical data to work with.
2. Data sparsity – having a low percentage of rated items in your data set.

2.3.2.1 Neighborhood based Collaborative Filtering

Formal definition

Let the set of users in the system be denoted by *U* , items by *I*, and ratings by *R*. We denote the set of possible values for *R* as *S*. For our purposes, *S* = [1, 2, 3, 4, 5].

The rating of a given user u ϵ U given for an item i ϵ I can be denoted by r_{ui}. To identify the subset of users that have rated an item i, we use the notation U_i . likewise, I_u denotes the subset of items rated by the user u.

The items rated by users u and v, $I_u \cap I_v$ are represented by I_{uv} .Likewise, U_{ij} represents the users that have rated the two items i and j.

Given a set of ratings r, the goal is to learn a function f: $U \times I \rightarrow S$ that predicts the rating of a new user u for a new item i. We write $P(r_{ui})$ to denote this predicted rating.

2.3.3 Components of collaborative filtering

Rating Normalization

Due to differences in culture, preference and background, users have personal scales even when an explicit one is defined. Some users are reluctant to give high/low ratings to items that they liked/disliked. We use normalization schemes to attempt to globalize these individual ratings (Herlocker, 2017)

a) Mean-centering

A raw rating r_{ui} is transformed to a mean-centered one $h(r_{ui})$ by subtracting it from the mean. In item-item CF, the mean centered normalization is calculated as shown.

$$h(r_{ui}) = r_{ui} - \bar{r}_i$$

Equation 1Mean Centering

where $\bar{r}_i$ is the mean rating given to item i by user u.

b) Z-score normalization

This method of normalization also considers the spread in the individual rating scales. In item-based methods, it divides the mean-centered rating by the standard deviation of ratings given to item i.

$$h(r_{ui}) = \frac{r_{ui} - \bar{ri}}{\sigma_i}$$

Equation 2 Z score normalization

Similarity weight computation

We need a formula to determine how similar two users/items are to one another. The two techniques discussed here will be sufficient.

a. Cosine based similarity

Users u and v are treated as two vectors in m-dimensional space, where $m = |I_{uv}|$

The similarity between two vectors can be measured by computing the cosine between them.

$$Cosine(u, v) = \frac{u \cdot v}{|u||v|}$$

$$CV(u, v) = \frac{\sum_{i \epsilon I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \epsilon I_u} r_{ui}^2 \cdot \sum_{j \epsilon I_v} r_{vj}^2}}$$

Equation 3Cosine Similarity

b. Pearson-based similarity

The above method does not account for mean and variance of the ratings made by the two users, and this measure does just that.

$$\frac{\sum_i \big((x_i - \bar{\imath})\big)\big((y_i - \bar{\imath})\big)}{\sqrt{\sum_i (x_i - \bar{\imath})^2}\sqrt{\sum_i (y_i - \bar{\imath})^2}}$$

Equation 4Pearson Similarity

For our purposes, to calculate the similarity between two items *i* and *j*, we compare ratings made by users that have rated both these items.

$$PC(i, j) = \frac{\sum_{u \epsilon U_{ij}} (r_{ui} - \overline{r}_i)\big(r_{uj} - \overline{r}_j\big)}{\sqrt{\sum_{u \epsilon U_{ij}} (r_{ui} - \overline{r}_i)^2 \big(r_{uj} - \overline{r}_j\big)^2}}$$

Equation 5Pearson-based User Similarity

Item based vs user based KNN

There are various factors to consider while choosing between Item and user-based KNN. These include:

a) Accuracy – if there are more users than items, which is often the case in e-commerce, item-based approaches can produce more accurate results. However, if there are more users than

items, as is the case in our dataset, we may get more accurate predictions from user-based neighborhood methods. (Ricci, Rokach, Shapira, & Kantor, 2011)

b) Efficiency – When the number of users is more than the number of items, item-based recommendations are more computationally efficient.

c) Novelty/serendipity – Item-based methods recommend to the user items that are similar to those they have already rated. This may lead to less diverse recommendations. User based techniques are more likely to have serendipitous results.

2.3.2.2 Top N Recommenders

Given a user-item matrix M and a set of items I_u rated by the user u, identify an ordered set of items O such that $|O| \leq N$ and $O \cap U = \emptyset$ (Mukund Deshpande, 2004). In KNN, we predict what a user would rate a given item and provide them a list of items that are most highly rated. In top N recommenders, we do not treat recommendation as a regression problem. Instead, we find the most similar users to you and return items that they liked.

In the case of item-item top N, we find all the items you liked and return the top N items that are most similar. As a result, we cannot measure the accuracy of these results. We rely on measures such as Hit Rate and ARHR.

2.4 Evaluating CF Recommender Systems

There are various ways of evaluating a RS. In this study, we measure the following.

2.4.1 Accuracy

We can assume that a RS that provides more accurate predictions will be preferable to the user. This method of evaluation is preferred because it can be carried out completely offline. There are various methods of measuring the accuracy of predictions.

a) Root Mean Squared Error (RMSE)

One of the most popular prediction accuracy measures. It is calculated as shown.

For a given test set T, we find the difference between the known ratings r_{ui} and the system generated predictions. We square this and find the average. The lower the RMSE score, the better.

$$RMSE = \sqrt{\frac{1}{|T|}\Sigma_{(u,i)\epsilon T}(P(r_{ui}) - r_{ui})^2}$$

Equation 6 Root Mean Squared Error

b) Mean Absolute Error (MAE)

RMSE will greatly penalize large errors. MAE is a great alternative to it. The difference between the two techniques is only in the magnitude of errors made.

$$MAE = \sqrt{\frac{1}{|T|}\Sigma_{(u,i)\epsilon T}|P(r_{ui}) - r_{ui}|}$$

Equation 7 Mean Absolute Error

2.4.2 Hit Rate

To evaluate the quality of the top N recommendations in a RS, we look at the number of hits and their relative position within the top-N items. The number of hits is the number of items in the test set that were also present in the top-N items returned to each user. The two metrics that we will use are *Hit Rate* and *Average Recpirocal Hit Rank* (Mukund Deshpande, 2004). A HR of 1.0 shows that the algorithm was always able to recommend the hidden items.

$$\text{Hit Rate(HR)} = \frac{\text{numberofhits}}{|\text{U}|}$$

Equation 8 Hit Rate

However, In measuring hit rate, the position of the recommendation is not considered, all hits are treated equal. ARHR can be used to account for this.

$$Average\ Reciprocal\ Hit\ Rank(ARHR) = \frac{1}{|U|}\Sigma_{i=1}^{h}\frac{1}{p_i}$$

Equation 9 ARHR

2.4.3 Novelty

Recommendations are novel if the items suggested are new to the user, i.e. they did not know about them. We could measure novelty through perceived quality (Oscar Celma, 2008). This requires feedback from the user. For offline experiments, we assume that more popular items are less novel.

2.4.4 Diversity

We can use item to item similarity to determine how similar items in a list are (Ricci, Rokach, Shapira, & Kantor, 2011). Diversity can be considered to be the opposite of similarity. Diversity normally comes at the expense of accuracy.

2.4.5 User Coverage

Prediction accuracy in CF algorithms often increases with the amount of data, however the recommendations will be mostly to a small portion of the users/items where vast amounts of data exist. Coverage is the proportion of users for which the algorithm can recommend items (Ricci, Rokach, Shapira, & Kantor, 2011)

2.3.3 Knowledge Based Systems

These systems use information about users and products to pursue a knowledge-based approach to generating a recommendation, reasoning about what products meet the user's requirements (Burke, 2000). There are two types of knowledge-based recommenders, case-based and content-based recommenders. Case based recommender systems use a similarity function to estimate how much user requirements match the recommendation (Ricci, Rokach, Shapira, & Kantor, 2011). The similarity score represents the utility of the recommendation to the user. Case-based and constraint-based recommenders are similar concerning the knowledge they use and their functionality. The systems collect user requirements, make recommendations based on the knowledge of how well they meet the requirements, repair inconsistencies in situations where no solutions were available, and provide explanations for recommendation results (Ricci, Rokach, Shapira, & Kantor, 2011). The significant difference between the two types is in the way they calculate solutions: case-based recommenders make recommendations based on similarity metrics whereas constraint-based recommenders exploit a predefined knowledgebase that contains explicit rules on how to relate user requirements to product features (Falkner, Falfernig, & Haag, 2011).

2.3.4 Demographic-based systems

Demographic information may be used to identify the types of users that like a particular object. *Lifestyle Finder* classifies users in 62 predefined clusters and makes recommendations to a particular user based on the group he belongs to (Pazzani, 1999). Demographic-based recommendations can also be made using the region/country of the user. Netflix offers movies based on the area in which the user lives (Sharma & Singh, 2016). Therefore, only movies available in the user's location can be part of a recommendation.

2.3.5 Community-based recommender systems

Community-based recommender systems work on the assumption that people are more likely to rely on recommendations from friends rather than from other similar but unknown individuals. Thus, the approach makes recommendations based on the preferences of the user's friends (Ricci, Rokach, Shapira, & Kantor, 2011). Community-based systems differ from collaborative filtering in that the former relies on similarity with friends while the latter is based on similarity with any users. The approach uses the social network of the user's friends and their ratings to make recommendations.

2.3.6 Hybrid Recommendation Techniques

Combine two or more recommendation methods, such as CB and CF in order to improve RS performance and address shortcomings of individual problems. For instance, collaborative filtering has some inherent drawbacks. One of them is the first rater problem. Items have to be rated at least once before they can be considered for recommendation. However, the CB approach does not face this problem, since its recommendations are based on the attributes of a product (Ricci, Rokach, Shapira, & Kantor, 2011). A combination of the two approaches can be used to overcome the first rater problem. Netflix recommends movies to the user based on their likes as well as the likes of other, it is a hybrid of collaborative filtering and content-based filtering. (Sharma & Singh, 2016).

2.5 Summary of Gaps

There are numerous studies on RS technologies and techniques in e-commerce. Some underscore the challenges and limitations of the various recommendation techniques, while others are surveys designed to provide an overview to other researchers. However, the researcher has not found any prior study that attempts to implement a recommender system and integrate it into an e-commerce website.

CHAPTER THREE

RESEARCH METHODOLOGY

Research Methodology is the science of studying how research is done scientifically (Kothari, 2004). This chapter will define and describe the research methods that we will use in this study. It explains the approach used in gathering the required data and outlines the steps that taken throughout this study.

3.1 Research Design

Research design is the conceptual structure within which research is conducted (Kothari, 2004). It constitutes the blueprint for the collection, measurement and analysis of data. Applied research aims at finding a solution to an immediate problem facing a society or industrial/business organization (Kothari, 2004). This is the type of research undertaken in this study. The justification for performing this kind of research is that this study attempts to address a real-world problem and suggest a solution to it.

3.2 Data Collection Methods

The e-commerce website specializes in selling drinks online. The data of interest to the research includes what criteria buyers use when looking for drinks online. It was collected through content analysis. Content-analysis consists of analyzing the contents of documentary materials such as books, magazines, newspapers and the contents of all other verbal materials which can be either spoken or printed (Kothari, 2004). In this case, contents of drinks e-commerce websites were analyzed and reviewed. This data that showed the salient features what buyers consider when looking for drinks online. Such data from prospective customers informed the design and implementation of the RS prototype.

3.3 System Development Methodology

The systems development life cycle (SDLC) is the process of understanding how an information system (IS) can support business needs by designing a system, building it, and delivering it to users (Dennis, Wixom, & Tegarden, 2015). A methodology is a formalized approach to implementing the SDLC.

Considering the time constraints for the study, this research adopted the Object Oriented Systems Analysis and Design (OOSAD) approach. While this approach can be coupled with any of the

traditional methodologies, the study employed the Rapid Application Development (RAD) methodology. It offers reduced development cycles, an advantage over waterfall development and a critical factor to the success of this study. Since data and processes are so closely intertwined, this approach provides a balance by decomposing the problem into objects that contain both data and processes.

Any object-oriented approach to developing systems must be:

i) Use–Case Driven – This means that use cases are the primary modeling tools defining the behavior of the system. A use case describes how the user interacts with the system to perform some activity, such as placing an order or searching for products.
ii) Architecture Centric – The underlying architecture of the system specification drives the construction and documentation of the system.
iii) Iterative and Incremental

Planning, typically the first step in OOSAD, involves steps such as: identifying an opportunity, developing a work plan and identifying staffing requirements. Most of these steps either do not apply or have already been covered in the first chapter. This chapter therefore did not focus on system planning, seeing as this is an academic endeavor.

3.3.1 Requirements Gathering

A requirement refers to a statement of what the system must do, or what characteristic it needs to have (Dennis, Wixom, & Tegarden, 2015). The purpose of requirements gathering is to turn the very high-level explanation of the business requirements to a more precise list of system requirements that ultimately lead to system design.

3.3.2 Systems Analysis

The second of four main phases of the SDLC, the analysis phase answers the questions of *who* will use the system, *what* the system will do, and *where* and *when* it will be used. The output of the system analysis phase is a requirement document. The detailed analysis of the proposed system is provided in *Chapter Four*.

3.3.3 Systems Design

Third phase of the SDLC. The design phase decides how the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms, and reports; and the

specific programs, databases, and files that will be needed (Dennis, Wixom, & Tegarden, 2015). A detailed systems design is provided in *Chapter Four.*

3.3.4 Systems Implementation

The final phase in the SDLC is the implementation phase, during which the system is actually built (or purchased, in the case of a packaged software design). This is the phase that usually gets the most attention, because for most systems it is the longest and most expensive single part of the development process. The implementation is described in more detail in *Chapter Five.*

CHAPTER FOUR

SYSTEM ANALYSIS & DESIGN

According to (Dennis, Wixom, & Tegarden, 2015), this phase addresses the questions of *who* will use the system, *what* the system will do and *where* and *when* it will be utilized. Object oriented analysis involves identifying the salient classes and their relationships and determining system requirements. To do this, the researcher identified the actors of the system and their various interactions with it.

4.1 System Requirements

The main functional requirements of the prototype for a recommender system, as proposed in this study, include the following:

a. **Ability to run on a web browser:** The proposed system can run on all popular web browsers including Google Chrome, Mozilla Firefox and Safari.
b. **Ability to rate products:** The system affords customers functionality to rate products on a scale of 1 to 5. This data is then used to generate appropriate recommendations.
c. **Ability to display recommended products:** The system displays customized products recommendations based on the item based collaborative filtering.
d. **Ability to add items to manage products and orders:** The proposed system provides the administrator with the functionality to create, update, view and delete both products and orders from the database.
e. **Ability to authenticate customers and administrator**: The system also provides the functionality for authenticating both the customers and the administrator. The administrator requires authentication in order to manage orders and products. The customers require authentication in order to customize their recommendations.

4.2 Use Cases

Use case 1: Add Product

This use case describes the interactions of an administrator as they add new products to the database. The administrator first navigates to the login page of the administrator panel. They then enter their email and password. Upon successful authentication, they are redirected to the

dashboard, where they can see various statistics about the e-commerce website. They navigate to the products section where a list of all products will be displayed. On this page, the administrator is provided with a button that allows them to add new products to the database. Upon clicking this button, they are provided with a pop-up dialog where they key in all the required attributes for a product. They save the product and the table automatically refreshes once the dialog is closed, showing the new product.

This use case has two exceptions. The first is an authentication exception, which occurs if the administrator's credentials are not valid. The second occurs if the staff attempts to save a product before specifying all of its required attributes. When either exception occurs, the administrator is provided with helpful error messages by the system.

Use Case Name: Add Product	**ID:** UC-1
Actor: Administrator	
Description: The administrator populates the database with products by making use of an intuitive user interface.	
Trigger: The administrator needs to add products to the database	
Preconditions 1. The actor is an authorized admin 2. The website is available	
Normal Course 1. The user navigates to the products page 2. The system displays the products page 3. The user clicks on the add product button 4. The system displays a modal for product details 5. The user adds product details and submits them 6. The system adds the product to the database and refreshes the products table to display added products.	
Exceptions E1: Login credentials are invalid 1. The system displays an error message 2. The administrator is logged out	

3. The system displays the login page E2: The administrator does not provide one or more required product details 1. The system alerts the administrator of missing values 2. The system does not save the product 3. The system displays the add product modal
Postconditions 1. One product is added to the database

Use case 2: Rate Product

This use case depicts the interactions of the customer with the system as they rate products. Upon the first visit to the online shop, they are required to register for an account. Once they are successfully authenticated, the users are provided with a catalogue of all the products in the database. Each product listing will have a rating widget that allows the user to rate a product on a scale of 1 to 5.

This use case has one exception. This is an authentication exception, which occurs if the user is not signed in. Only logged in customers are authorized to rate products.

Use Case Name: Rate Product	**ID:** UC-2
Actor: Customer	
Description: The customer rates products on a scale of 1 to 5. This information is later used to generate recommendations for them.	
Trigger: The customer wants to get better product recommendations.	
Preconditions 1. The actor is an authenticated user 2. There are products already added to the database	
Normal Course 1. The customer visits the home page 2. The system displays the products catalogue 3. Each product has a rating widget 4. The customer rates the product on a scale of 1 to 5 5. The system submits this information to the database	
Exceptions E1: Login credentials are invalid 1. The system displays an error message 2. The customer is logged out 3. The system displays the login page	
Postconditions 1. One product rating is added to the database	

Use case 3: Buy Products

This use case describes the interactions of the customer with the system as they make an order online. The customer visits the shop and is provided with a listing of the available products. Each listing has a button that allows the customer to add the products to a virtual cart. The use case continues until the customer has added all the products they would like to purchase. Once done, they click on a checkout button that redirects them to a page where they can select the delivery location. They then provide additional information about the order, such as their address, the name of the receiver and their phone number. The order is then placed to be received by the administrator.

Use Case Name: Buy Product	**ID:** UC-3
Actor: Customer	
Description: The customer shops on the e-commerce website. They are able to add products to a virtual cart and place an order	
Trigger: The customer wants to purchase products.	
Preconditions 1. The website is available 2. There are products already added to the database	
Normal Course 1. The customer visits the home page 2. The system displays the products catalogue 3. Each product has an 'add to cart' button 4. The customer adds as many products as they please to their cart 5. The system will cache this information 6. The user adds their shipping information 7. The system displays an interface requiring the user to fill in their details such as name and phone number. If they are already authenticated, this form is prefilled by the system.	
Exceptions E1: A system error occurs 1. The system displays an error message 2. The customer takes action as advised by the system	
Postconditions 1. The administrator receives the customer's order 2. The order details are added to the database	

Use case diagrams

These diagrams provide a visual representation of the users interactions with the systems.

We provide diagrams for the use cases described above.

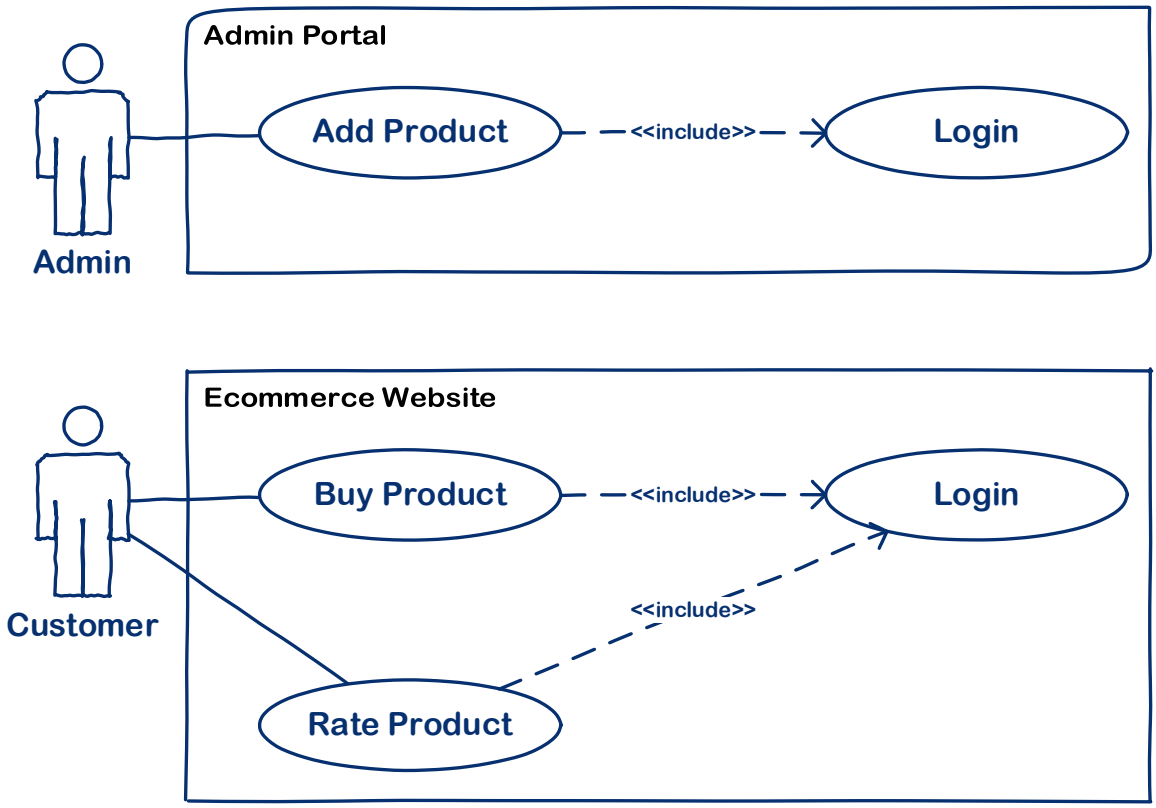


Figure 1Customer and admin use case diagrams

4.3 Class Diagram

A class diagram is a static model that shows the classes and the relationships among classes that remain constant in the system over time (Alan Dennis, 2015). We use class diagrams to show the structural model of the system.

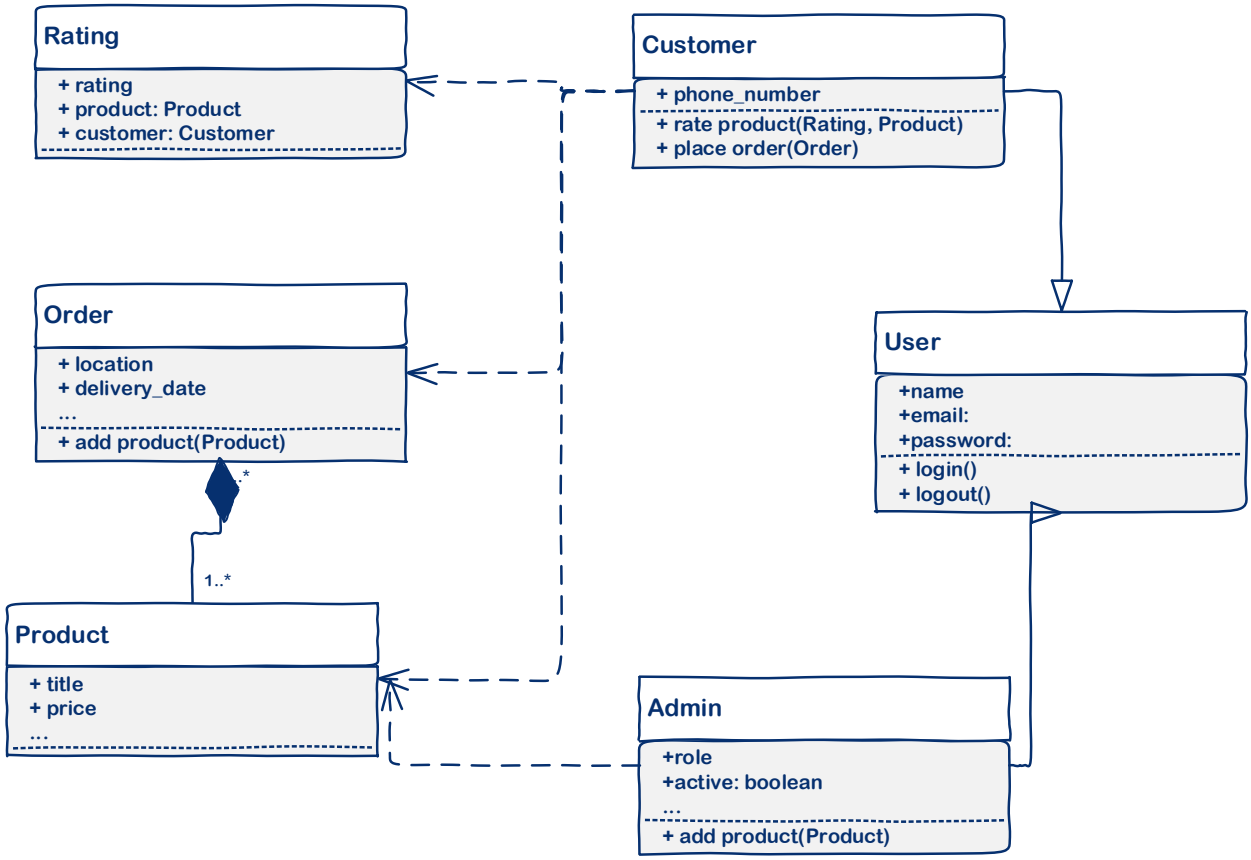


Figure 2 Class Diagram for the system

CHAPTER FIVE

SYSTEMS IMPLEMENTATION AND TESTING

5.1 Introduction

This is the last phase of the SDLC. In this phase, the researcher built the code base and tested its various functionalities to ensure that it performed as required. The architecture of the proposed system has a few components. The database stores data on the products, orders and user ratings. The API and the Recommender System interact with the database and run individual web servers. The Admin Portal provides the administrator with a user interface for managing the products and customer orders. The Frontend interacts with both the API and the Recommender in order to provide a clean user experience for the customer. The system architecture is illustrated below.

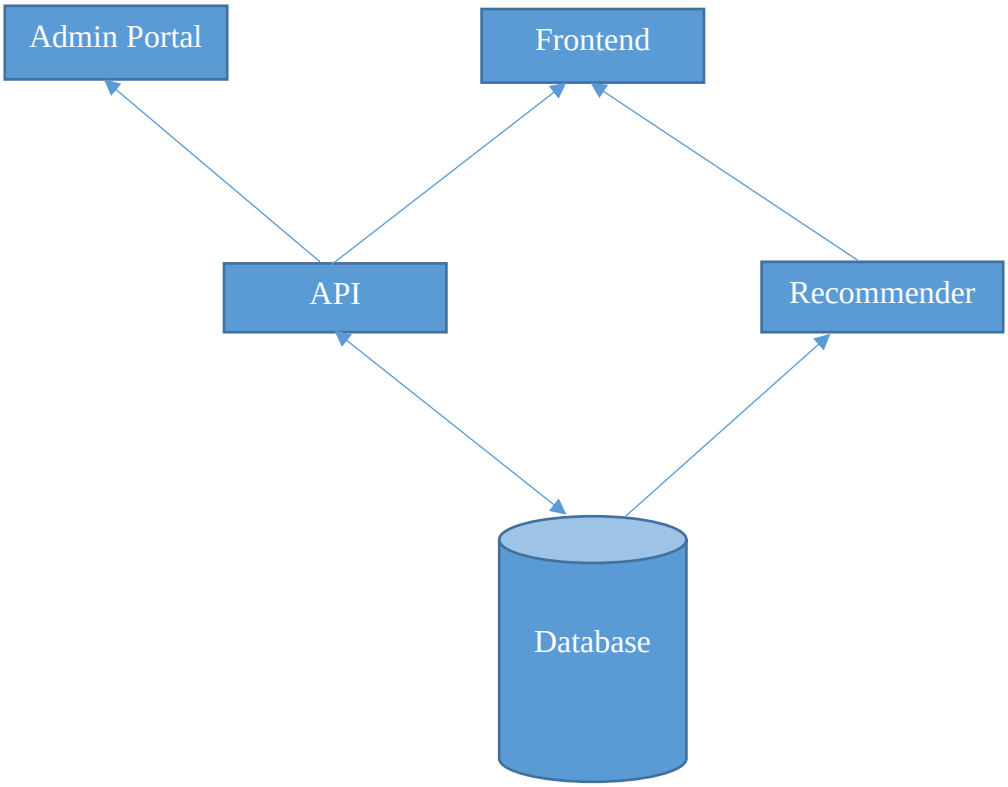


Figure 3 System architecture

5.2 Development Tools

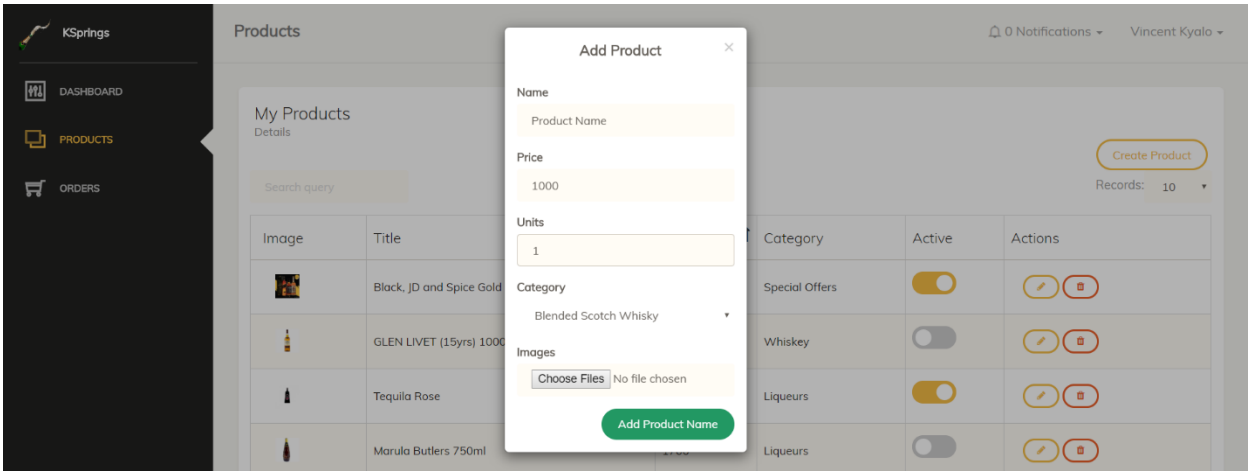
The researcher used various frameworks to develop the system. The API made use of the PHP framework Laravel to enable quick prototyping. The RS was written in Python and interacted with the frontend through a Flask API. This decision was made largely due to the proliferation of scientific libraries that greatly simplified the development process. The Frontend and Admin Portal used the Vue JavaScript Library due to its approachable and versatile nature. The researcher used MariaDB as the database and Nginx as the webserver.

5.3 User Interfaces

User interfaces were built using HTML and styled using CSS. Instead of writing CSS from scratch, the researcher made use of Tailwindcss, a utility-first CSS framework for rapidly building custom designs. The researcher also leveraged the reactive nature of the Vue.js frontend framework to design interactive user interfaces.

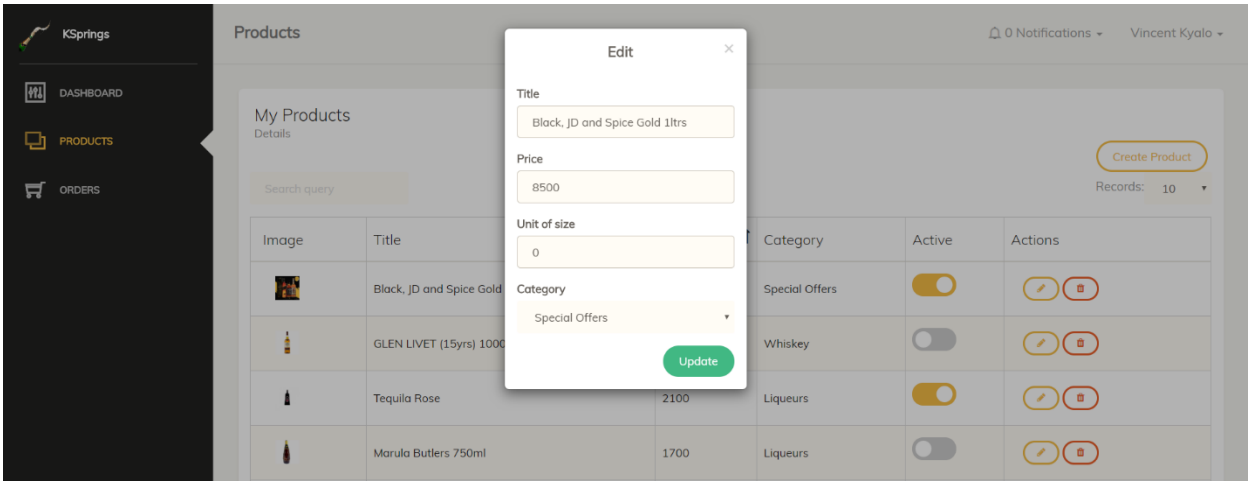
5.3.1 User Interface for Adding New Products to the Database

This interface is used by the administrator to add new products to the database. Logically, this is where the flow of the system starts, since this process must be carried out before customers can start using the system. The user interface is illustrated below.



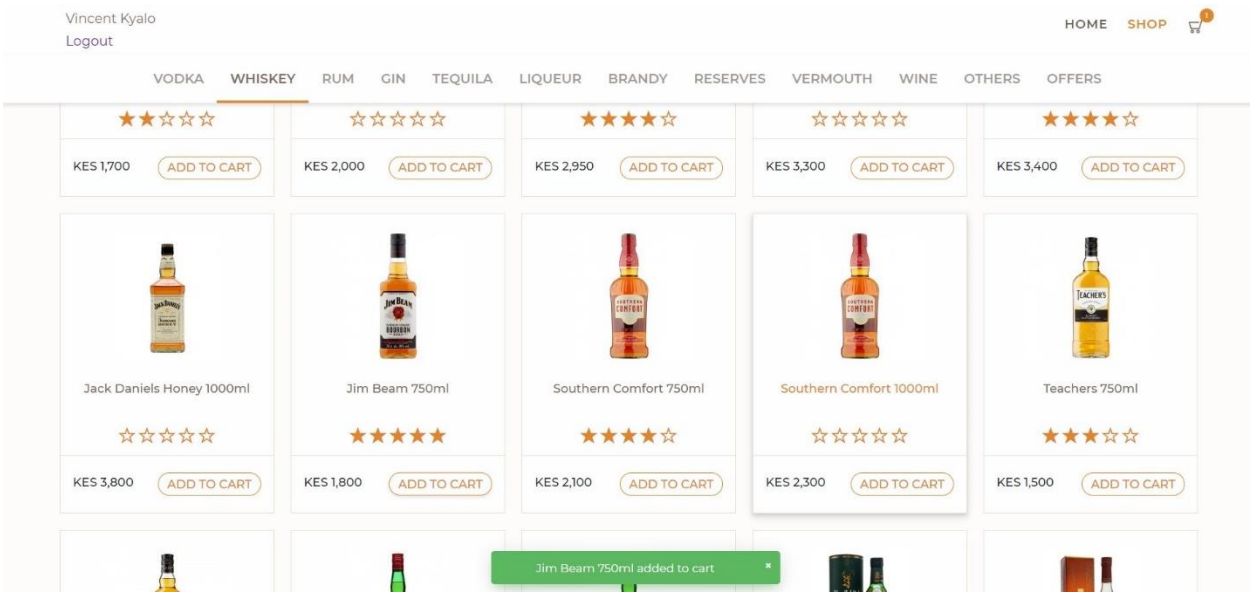
5.3.2 User interface for editing product details

An administrator can edit product details. This action requires authentication. The user interface is illustrated below.



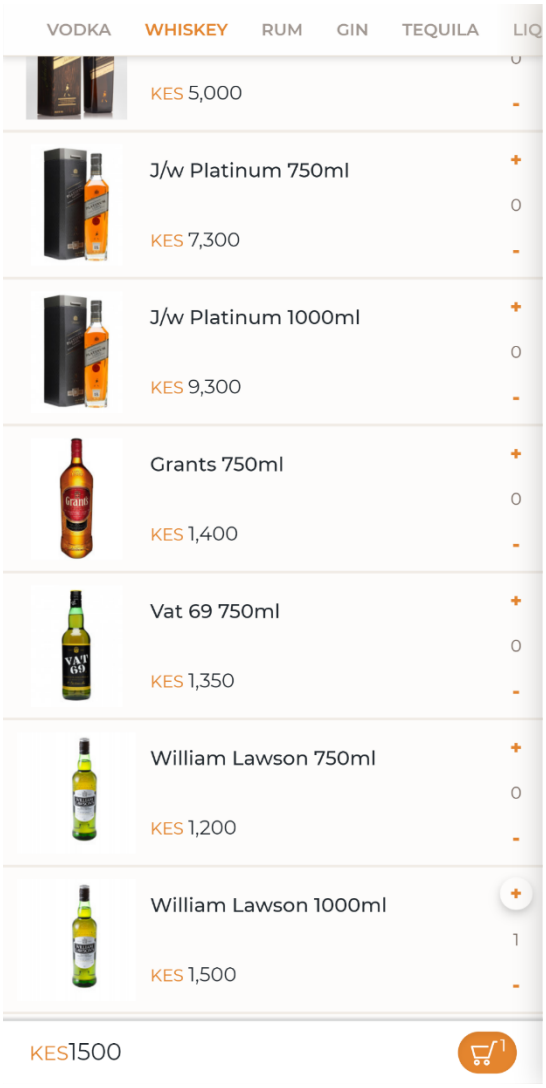
5.3.3 User Interface for Shopping

This is where users spend most of their time. This user interface is used by customers to shop for products. It also serves as a data collection interface in that users provide ratings for products they may have purchased in the past. This data is what feeds the RS, it translates to better recommendations for the user.



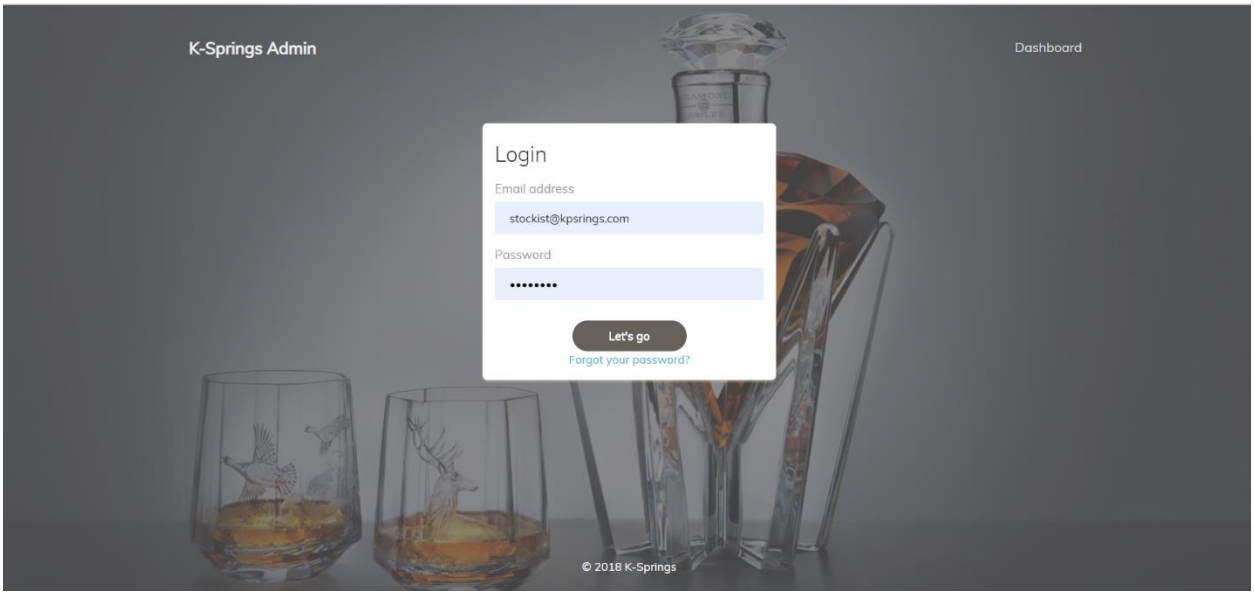
5.3.4 User Interface for Shopping on Mobile

Part of the requirements for the prototype was to provide a user experience that scales appropriately to different screen sizes. Shown below the mobile version of the interface used by the customer to shop for products.



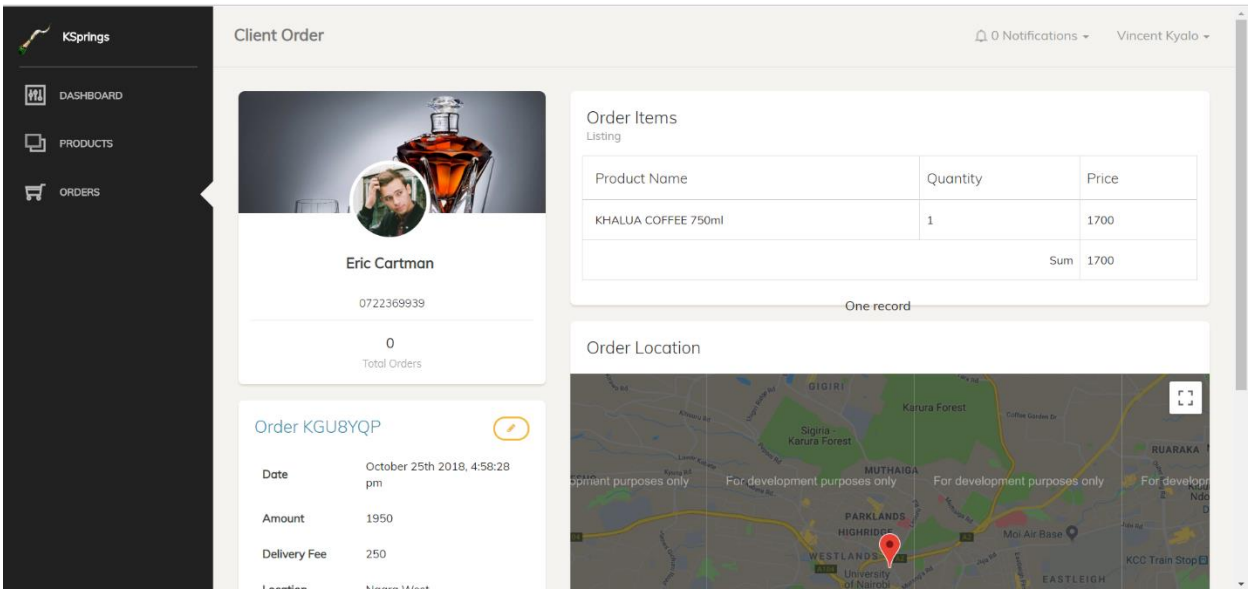
5.3.5 User Interface for Authenticating the Administrator

This interface is used by the administrator to log in to their portal where they can manage products and customer orders.



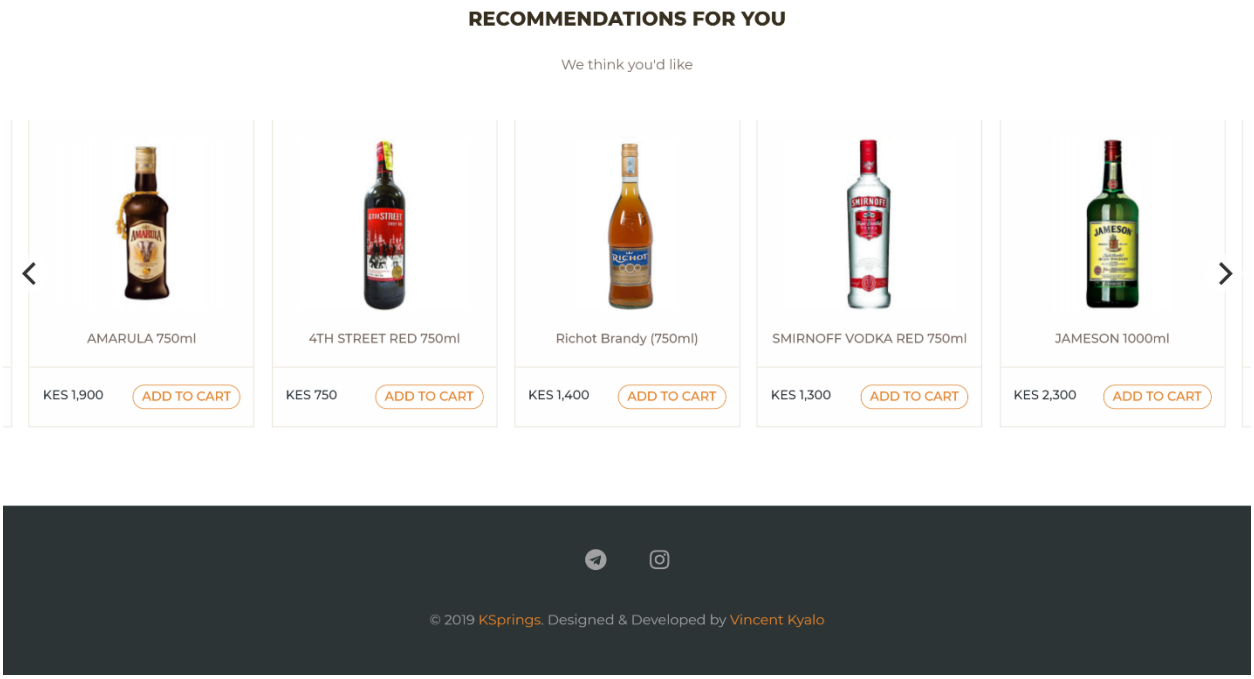
5.3.6 User Interface for Managing Customer Orders

This interface is used to manage customer orders. Here the administrator can view the customer summary, order details and update the order status. The administrator must be authenticated to perform these actions.



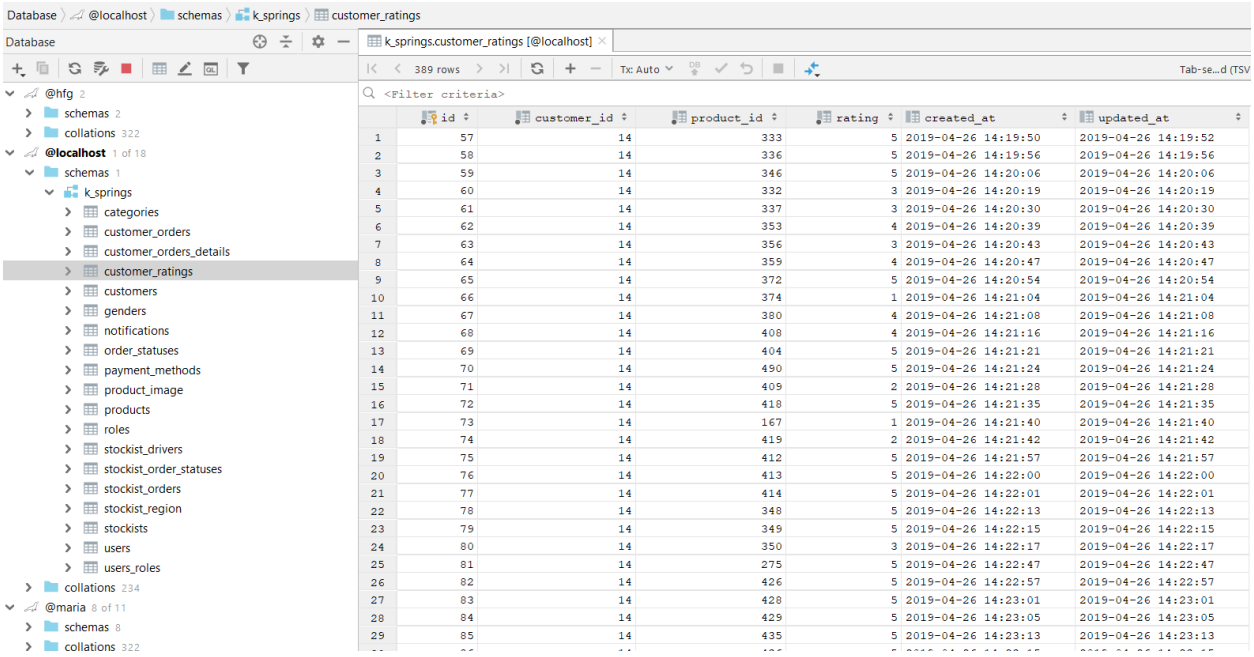
5.3.7 User Interface for Displaying Recommendations

The main deliverable in this system. This user interface displays personalized recommendations to customers. The customers must be authenticated.



5.4 Database

The study made use of the MariaDB database server. The figure below shows a snapshot of the customer ratings table in the database in the development environment.



5.5 Testing

The researcher adopted a Test-Driven Development approach to ensure that the system performed according to its specification. Various types of tests were performed, including: unit tests, functional tests and integration tests. Unit testing is a software testing method by which individual units of source code, together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use (Kolowa & Huizinga, 2007). Functional tests are carried out to test multiple components in collaboration. Integration tests follow a business process: components work together to achieve a business objective.

Since the system API was written in PHP, the researcher made use of the PHPUnit testing framework. Unit tests were carried out on the 'Product' and 'Order' models. Functional tests were carried out on the 'Product Controller' and 'Order Controller'. Integration tests were carried out on major business processes such as creating, viewing, updating and deleting a product. Finally, the actions of registering and authenticating users were also tested.

Functionality	**Tested?**	**Passed?**
Register a customer	Yes	Yes
Authenticated a customer	Yes	Yes
Add a new product to the database	Yes	Yes

View all products in the database	Yes	Yes
Update a given product	Yes	Yes
Delete a given product from the database	Yes	Yes
Generate and display recommendations	Yes	Yes

The researcher further carried out usability tests. Usability testing is a type of software testing where a small set of target end users of a software system use it to expose usability defects. The researcher recruited 5 volunteers and assigned the testers various tasks. The researcher then observed the testers to uncover any usability issues and improve end-user satisfaction. The prototype proposed on this study was tested on various web browsers including Google Chrome (Version 75.0.3770.142), Mozilla Firefox (Version 68.0) and Microsoft Edge.

5.6 Testing recommendations

The researcher made use of the open-source surprise library (Hug, 2020) to evaluate the various recommendation algorithms. For KNN-based rating prediction algorithms, the results were as follows:

```
# Results

Algo       RMSE       MAE        HR         ARHR       Coverage   Diversity  Novelty
User KNN   1.1862     0.9645     0.1176     0.0077     1.0000     0.9998     71.3118
Item KNN   1.1994     1.0204     0.2941     0.0869     0.8824     0.9984     40.7099
Random     1.6830     1.3859     0.2941     0.0792     1.0000     0.9996     53.9706

# Legend

RMSE:      Root Mean Squared Error. A lower score means better accuracy.
MAE:       Mean Absolute Error.A lower values mean better accuracy.
HR:        Hit Rate; how often we are able to recommend a left-out rating. Higher is better.
ARHR:      Average Reciprocal Hit Rank - Hit rate that takes the ranking into account. Higher is better.
Coverage:  Proportion of users for whom recommendations above a certain threshold exist. Higher is better.
Diversity: 1-S, where S is the average similarity score between every possible pair of recommended items
Novelty:   Average popularity rank of recommended items. Higher means more novel.
```

Figure 4KNN metrics results

In order to find the k-nearest neighbours, the cosine similarity metric was used for User KNN and pearson's was used for Item KNN. Both these algorithms had more accurate recommendations than Random predictions. Item based KNN outperformed User KNN in terms of hit rate. As discussed earlier, User KNN produced more novel and diverse recommendations. User based KNN produced more accurate recommendations. As discussed, we can expect this to change as the

number of users grows. We are more likely to get better accuracy scores using item-based techniques in large commercial systems where the number of users is higher.

Testing top-N recommenders

We measured the quality of our top N recommenders by splitting our dataset into a training set and a test set (Mukund Deshpande, 2004). We then built our similarity matrix using only items in the training set and calculated Hit Rate and Average Reciprocal Hit Rank.

	HR	ARHR
Item CF	0.6667	0.0726
User CF	0.6	0.1821

Table 5 Hit Rate metrics for top N recommenders

Both top N recommenders outperformed KNN based algorithms in terms of hit rate. User CF had a much better hit rate compared to item CF.

```
kyalo@Carbon:~/code/k-springs/k-springs-surprise/rs$ python cf_user.py
Loading product ratings...

Computing product popularity ranks so we can measure novelty later...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
HR 0.6
ARHR 0.18211880711880715
kyalo@Carbon:~/code/k-springs/k-springs-surprise/rs$ python cf_item.py
Loading product ratings...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
train_set : num_users: 15 num_items: 96...
HR 0.6666666666666666
ARHR 0.07265413236001471
```

Figure 6 User & Item based top N metrics

CHAPTER SIX

CONCLUSIONS AND RECOMMENDATIONS

6.1. Introduction

Recommender Systems are an essential part of how we shop online. However, the task of building an RS can often seem daunting to software developers. The researcher proposed the development of an e-commerce recommender system. This chapter discusses the conclusions and recommendations of this study.

6.2. Discussion

Among the objectives of the study were:

1. To provide an overview of the various recommendation techniques used and attempt to identify existing issues in recommender systems.
2. To provide users with personalized recommendations by implementing one of the recommendation techniques in e-commerce.
3. To allow users to shop online by designing an intuitive customer journey that responds appropriately to different screen sizes.

In chapter 2, we provided a brief history or Recommender Systems. We then reviewed various recommendation techniques. We made formal definitions of the problems, listed tradeoffs and techniques used to evaluate the various recommendation algorithms. This achieved the first objective.

In chapters 3, 4 and 5, the researcher achieved the remaining objectives by implementing various recommendation algorithms and evaluating them using metrics discussed in chapter 2. The desktop and mobile user interfaces were presented in chapter 5.

6.2.1 Focus on accuracy

The Netflix Prize, a $1 million machine learning competition launched in 2006, was focused on improving rating prediction accuracy through RMSE (Carlos A Gomez-Uribe, 2015). As such, a lot of research has been carried out on the same. However, as we have seen, focusing on rating prediction accuracy alone is not sufficient to determine the effectiveness of a RS.

If the goal is to provide top N suggestions, it makes sense to focus on metrics such as Hit Rate and ARHR.

6.2.2 Limitations of the prototype

- Data sparsity – With less than 400 customer ratings, the dataset gathered by the researcher was insufficient. Recommendation algorithms such as KNN are sensitive to sparse data.
- Scale – Many recommendation algorithms proven to work well on small problems fail to provide good results on a large scale (Paul Covington, 2016).

6.3. Recommendations

Avoid rating prediction – Even though they launched the Netflix Prize, the company no longer collects 5-star rating data. This means they are no longer focused on rating predictions.

Cold start – Our RSs do not account for new users/items. They simply return no recommendations since these entities do not have entries in the similarity matrix. One option is to recommend popular items or use implicit data.

Stop lists – our recommenders do nothing to protect against vulgarity, terrorism, political extremism, adult oriented content, etc. In the world of responsible AI, these are all things that should be filtered against in your dataset.

6.4. Future Work

Because different algorithms have different strengths and use-cases, recommenders often utilize hybrid recommendation approaches (Paul Covington, 2016) (Carlos A Gomez-Uribe, 2015) in which more than one recommendation algorithm is used. We could for instance combine Content Based and Collaborative Filtering algorithms. The implementation of such a system is worth exploring.

6.5. Conclusion

This study set out to develop an e-commerce recommendation system. In doing so, the main challenge faced by the researcher was data scarcity. To provide meaningful recommendations, we need vast amounts of data. The researcher also realized that while neighborhood-based techniques provide a way to measure the performance of an algorithm through accuracy, they

should not be used in isolation. The source code for this study is readily available and can be built upon.

References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 734-749.

Alan Dennis, B. H. (2015). *System Analysis & Design: An Object Oriented Approach with UML*. Wiley.

Bennett, J., & Lanning, S. (2007). The Netflix Prize. *KDD Cup and Workshop 2007.* San Jose.

Burke, R. (2000). *Knowledge Based Recommender Systems.* Irvine: Marcel Dekker.

Carlos A Gomez-Uribe, N. H. (2015). The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*.

Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *System Analysis & Design.* John Wiley & Sons, Inc.

Ekstrand, M., Riedl, J., & Konstan, J. (2010). Collaborative Filtering Recommender Systems. *Foundations and Trends In Human Computer Interaction*.

Falkner, A., Falfernig, A., & Haag, A. (2011). Recommendation Techniques for Configurable Products. *Ai Magazine*, 99-108.

Herlocker, J. &. (2017). An Algorithmic Framework for Performing Collaborative Filtering. *ACM SIGIR Forum*.

Hug, N. (2020). Surprise: A Python library for recommender systems. *The Journal Of Open Source Software*.

Khusro, S., Ali, Z., & Ullah, I. (n.d.). Recommender Systems: Issues, Challenges, and Research Opportunities.

Kolowa, A., & Huizinga, D. (2007). *Automated Defect Prevention: Best Practices in Software Managemen.* Wiley-IEEE Computer Society Press.

Kothari, C. (2004). *Research Methodology: Methods & Techniques.* New Delhi: New Age International Publishers.

MacKenzie, I., Meyer, C., & Noble, S. (n.d.). Retrieved from McKinsey & Company: https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers

Mukund Deshpande, G. K. (2004, January). Item-based top-N recommendation algorithms. *ACM Transactions on Information Systems*, pp. 144-177.

Oscar Celma, P. H. (2008). A new approach to evaluating novel recommendations. *Conference on Recommender Systems, RecSys.* Lausanne, Switzerland.

Paolo Viappiani, B. F. (2006). Design and Implementation of Preference-Based Search. *Web Information Systems – WISE 2006: 7th International Conference on Web Information Systems Engineering.* Wuhan, China.

Paul Covington, J. A. (2016). Deep Neural Networks for YouTube Recommendations. *RecSys '16: Proceedings of the 10th ACM Conference on Recommender Systems.* Boston Massachusetts.

Payne, J. (1993). *The Adaptive Decision Maker.* Cambridge University Press.

Pazzani, M. (1999). A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*, 393-408.

Rejoiner. (n.d.). Retrieved from Rejoiner: http://rejoiner.com/resources/amazon-recommendations-secret-selling-online/

Ricci, F., Rokach, L., Shapira, B., & Kantor, P. (2011). *Recommender Systems Handbook.* New York: Springer.

Rich, E. (1979). User modeling via stereotypes. *Journal of Cognitive Science*.

Sharma, R., & Singh, R. (2016). Evolution of Recommender Systems from Ancient Times to Modern Era: A Survey. *Indian Journal of Science and Technology*.

Steven Lindell, J. H. (2009). *From Tapestry to SVD: A Survey of the Algorithms That Power Recommender Systems.* Haverford College Department of Computer Science.

APPENDICES

Appendix 1: Schedule

The project's activities were scheduled as shown in the Gantt chart below.

Activities	2	4	6	8	10	12	14	14	18	20
Feasibility Study	■									
Study		■								
System Analysis			■	■						
System Design					■	■				
Coding							■	■		
Testing									■	
Implementation										■
Documentation	■	■	■	■	■	■	■	■	■	■

Appendix 2: Code Samples

Metrics.py – module used for measuring different metrics

```
import itertools

from surprise import accuracy
from surprise import KNNBaseline

class Metrics:

    @staticmethod
    def mae(predictions):
        return accuracy.mae(predictions, verbose=False)

    @staticmethod
    def rmse(predictions):
        return accuracy.rmse(predictions, verbose=False)

    @staticmethod
    def hit_rate(top_n_predicted, test_set_predictions):
        """
            A hit is the number of items in the test set that were also
            present in the top-N items recommended to a user.

        Parameters
        ----------
        top_n_predicted : dict
            The top N predictions for all the users in the test set
        test_set_predictions : list
            Predictions (uid, iid, r_ui, pr_ui) made on the test set.
            The test set was excluded from the training data

        Returns
        -------
            hit-rate (HR) = num_hits / num_users
            A HR of 1.0 indicates that the algo was always able to recommend
            any of the hidden items
        """
        hits = 0

        for test_uid, test_iid, r_ui in test_set_predictions:
            for iid, _ in top_n_predicted[test_uid]:
                if test_iid == iid:
                    hits += 1
                    break
```

```
        # Compute overall precision
        return hits / len(test_set_predictions)

    @staticmethod
    def average_reciprocal_hit_rank(top_n_predicted, test_set_predictions):
        summation = 0

        for test_uid, test_iid, r_ui in test_set_predictions:
            hit_rank = 0
            rank = 0
            for iid, pr_ui in top_n_predicted[test_uid]:
                rank = rank + 1
                if test_iid == iid:
                    hit_rank = rank
                    break
            if hit_rank > 0:
                summation += 1.0 / hit_rank

        return summation / len(test_set_predictions)

    # The proportion of users for which the system can predict 'good' ratings.
    @staticmethod
    def user_coverage(top_n_predicted, num_users, rating_threshold = 4.0):
        hits = 0

        for uid, predicted_ratings in top_n_predicted.items():
            hit = False
            for iid, pr_ui in predicted_ratings:
                if pr_ui >= rating_threshold:
                    hit = True
                    break
            if hit:
                hits += 1

        return hits / num_users

    @staticmethod
    def diversity(top_n_predicted, full_train_set):

        # item-item similarity is used to measure diversity.
        sim_options = {'name': 'pearson_baseline', 'user_based': False}
        diversity_algo = KNNBaseline(sim_options = sim_options)
        diversity_algo.fit(full_train_set)
```

```
        n = 0
        total = 0
        sims_matrix = diversity_algo.compute_similarities()

        for uid, predicted_ratings in top_n_predicted.items():
            pairs = itertools.combinations(predicted_ratings, 2)
            for pair1, pair2 in pairs:
                inner_id_1 = diversity_algo.trainset.to_inner_iid(pair1[0])
                inner_id_2 = diversity_algo.trainset.to_inner_iid(pair2[0])
                similarity = sims_matrix[inner_id_1][inner_id_2]
                total += similarity
                n += 1

        S = total / n
        return (1 - S)

    # We assume that more popular items are less novel
    @staticmethod
    def novelty(user_predictions, popularity):
        n = 0
        total = 0

        for uid, predicted_ratings in user_predictions.items():
            n += len(predicted_ratings)
            for iid, _ in predicted_ratings:
                total += popularity[iid]

        return total / n
```

cf_item.py – Top N user CF

```
from surprise import KNNBasic
import heapq
from collections import defaultdict
from operator import itemgetter
from surprise.model_selection import LeaveOneOut
from metrics import Metrics
from evaluation_data import EvaluationData
from data.repository import RatingsRepository

def load_data():
    repository = RatingsRepository()
    print("Loading product ratings...")
    data = repository.load_customer_ratings()
    print("\nComputing product popularity ranks so we can measure novelty
later...")
    rankings = repository.get_item_rankings()
    return (repository, data, rankings)

repository, data, rankings = load_data()

evalData = EvaluationData(data, rankings)

# Train on leave-One-Out train set
train_set = evalData.get_loocv_train_set()
sim_options = {
    'name': 'cosine',
    'user_based': True
}

model = KNNBasic(sim_options=sim_options)
model.fit(train_set)
# user to user similarity matrix
corr_matrix = model.compute_similarities()

# left_out_test_set = model.test(evalData.get_loocv_test_set())
left_out_test_set = evalData.get_loocv_test_set()

top_n = defaultdict(list)
k = 10

# Generate recommendations for every user in the trainset
for uiid in range(train_set.n_users):
```

```
# for uiid in train_set.all_users():
    similarity_row = corr_matrix[uiid]
    similar_users = []

    for sim_uiid, sim_score in enumerate(similarity_row):
        if sim_uiid != uiid:
            similar_users.append((sim_uiid, sim_score))

    # Get top k most similar users to this one
    k_neighbours = heapq.nlargest(k, similar_users, key= lambda t: t[1])

    # Get the stuff they rated, and add up ratings for each item, weighted by
    # user similarity
    candidates = defaultdict(float)
    for k_uid, k_sim_score in k_neighbours:
        for iid, k_rui in train_set.ur[k_uid]:
            candidates[iid] += (k_rui / 5.0) * k_sim_score

    already_rated = {}
    for iid, _ in train_set.ur[uiid]:
        already_rated[iid] = 1

    # Get top-rated items from similar users:
    n = 20
    user_id = train_set.to_raw_uid(uiid)

    for iid, weight in sorted(candidates.items(), key = itemgetter(1), reverse =
True):
        if iid not in already_rated and n > 0:
            product_id = train_set.to_raw_iid(iid)
            top_n[user_id].append((product_id, 0.0))
            n -= 1

# Measure
print("HR", Metrics.hit_rate(top_n, left_out_test_set))
print("ARHR", Metrics.average_reciprocal_hit_rank(top_n, left_out_test_set))
```