Final Year Project Report

Full Unit – Final Report

An ecommerce website with standardized features and recommendation systems

James Martin

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Reuben Rowe and Erjill De Vera

Department of Computer Science

Royal Holloway, University of London

April 05, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 17374

Student Name: James Martin

Date of Submission: 05/04/2024

Signature:

Table of Contents

Chapter 1: **Rationale**

Modern ecommerce technologies emphasise maximal sales and trust in various methods. Commercial standards backbone core features ubiquitous in ecommerce websites creating easy shopping experiences through user-centered and company-centered design for features aimed at traversing less for products interested in, while maintaining a desired graphics style through its objects and architecture, creating a foundation for boosting profits. Recommender systems learn from users' past behaviours, while manipulating big data to produce personalised results similarly reducing traversal by displaying predicted products for direct access. Powerful artificial intelligence techniques deal with large and complex information spaces namely 2 approaches I will be discussing and evaluating for Recommender systems: Content-based Filtering and Collaborative Filtering.

Different ecommerce sites display similar core features integral for the users' shopping experience while optimising profit opportunities:

- Categories
- Search bar
- Recommended items
- Basket display
- Payment process
- Order preview

Filtering systems as described with the first 3 bullet points, contribute to displaying the users' desired item in the minimal number of clicks and time due to rapidly condensing into subsets of items displaying choices the users are more likely to want and may pay for, increasing the ease of traversal while positively impacting the average user shopping experience in the process. Basket display and payment processes also highly impact user shopping experience with many implementations on the market that differ between styles and architecture, but the main efforts are to display the items with ease and minimal traversal which may smoothly transition directly into the payment process reassuring the user of wish for with its corresponding price and payment methods. While an abundance of intricate features with their respective implementations exists giving certain ecommerce websites an edge over its competition in this project my aim is to implement the main standard features that I introduced above, with evaluation of recommender methods used.

While organisations and ecommerce based communities have designed a plethora of architectures displaying core standard features improving user shopping experiences, the integration of recommender systems are less known and tend to be deliberately hidden due to its economic impact on online shopping giving companies an edge that they're less willing to share, allowing third-party firms and programs specialising in modified recommender systems to profit without its inner mechanics being cared for as long as it improves user interactions and financial returns for the contractee. Although I am not building highly fine-tuned precision based modified recommender systems, I will build examples using both content-based and collaborative filtering approaches, for a fundamental evaluation of basic recommender systems which are based upon in the current market alongside pre-established standard features of an ecommerce.

In this project report, I will describe the background research and papers read to understand the full context of my project along with the development cycle and implementations used to deliver on the goals as well as a review of the process, how things could have gone better and its relation to current issues in the computing world.

Chapter 2: **Literature Review**

2.1 Introduction

A lot of research had taken place within this project, therefore critically extracting useful information considering context from the following:

- how relevant the source is to my project
- how reputable the source is
- when the source was written therefore its historical knowledge of past events
- whether the source is reliable and accurate

was analysed before myopically implementing everything that was read. Having to implement a full stack for my project, many sources will have somewhat relevancy, but are most likely not full stack/ focus more on specific sections of a project therefore making sure this information stays in context of the project and not extrapolated was important in its success. Reputation of a source has intertwined features with its reliability and accuracy; providing evidence and ground theory alongside correct referencing creates a more reputable source allowing more information to be extracted knowing that a product of utmost quality will be assured. Finally, despite the reputation and its accuracy, sometimes information changes, and new context or innovations influence the foundations of a paper relegating it to become less accurate, reliable and relevant therefore is important to mentally note which sections may have/ or identify new techniques that optimises the system.

2.2 Using Content-Based Filtering for Recommendation

Source: R. Van Meteren and M. Van Someren, "Using Content-Based Filtering for Recommendation 1." Available: https://users.ics.forth.gr/~potamias/mlnia/paper_6.pdf

Users change. What they like or dislike is malleable giving recommender systems a nightmare when dealing with dynamic rates of trends which occur in a fierce environment with some economic influence attached. In this paper, content-based filtering is viewed as a classification task, combining user profiles and item features in a PRES (Personalized Recommender System) architecture to create an example of a content-based filtering approach. The PRES architecture compares a user profile with documents in its collection and adapts using positive feedback to create a better user model. The source meticulously goes through the entire process implementation from preprocessing to user profile – document comparing to find a degree of similarity which positive feedback adapted user models for a more accurate recommender which was extremely useful in grasping a firm basis of content-based filtering despite it being targeted towards a specific domain. The domain itself was changing at a rapid pace whereas an ecommerce domain rarely changes rapidly (i.e. its features don't convert into something else, they're mostly static) and therefore time depreciation is not a major issue, despite trends changing. This paper was created on 12th of May 2000, which seems extremely outdated in comparison to the history of content-based recommender systems and shows with the algorithm description for similarity which uses mere vectors to calculate with no optimisation techniques like matrices and a plethora of modification that could be carried out to speed the process. The paper also agrees that there are hindrances with its technique that cannot be improved on unless collaborative filtering is combined, making it one of the early prototypes of PRES architecture but seems extremely accurate and has stood time as core fundamentals reverberate through other papers and is built on early developments of machine learning as seen through their referencing.

2.3 Recommendation systems: Content-Based Filtering

Source: Zeynep Beyza Ayman, "Recommendation Systems: Content-Based Filtering," *MLearning.ai*, Dec. 6, 2022. https://medium.com/@zbeyza/recommendation-systems-content-based-filtering-e19e3b0a309e (accessed Mar 23, 2023).

A blog depicting the implementation of a content-based filtering system using python and pandas library uses a movies_metadata.csv dataset to create a term frequency – inverse document frequency matrix of the fields/features in the dataset, allowing cosine similarity to the probability of which movies or terms in movies are closer to others, hence ordering the recommendations. The implementation gives relevance since it both is an implementation of content-based filtering and uses python, which is used in my project, however this blog is very simple and focuses majorly on the implementation and the equations used but doesn't focus on the meaning behind it and is uncertain where this information is sourced or based on due to the lack of referencing making it less credible. The terminology used is very simple and the structure is like a guide which is skewed to the audience who most likely want a quick implementation that works with step-by-step instructions but does seem accurate especially after cross-referencing with other sources. It does use modern technologies and libraries to optimise its implementation but is uncertain on its evaluation since there are no mentions on the metrics it tries to optimize nor any evaluation method creating a less credible source but can be adapted.

2.4 Memory Based Collaborative Filtering – User Based

Source: C. Maklin, "Memory Based Collaborative Filtering — User Based," *Medium*, Aug. 10, 2022. https://medium.com/@corymaklin/memory-based-collaborative-filtering-user-based-42b2679c6fb5

A web article describing a single implementation method using python of a collaborative filtering technique using their previous knowledge upon the topic to adapt a core formula, analysing each iterative modification. The structure of the article was extremely clear and methodological explained throughout allowing readers at any stage to at least read the introduction while more fluent programmers can continue to understand its development. Since my project also uses python, this web article was relevant however, despite being uploaded in August 2022, the usage of modern libraries to assist with the development was limited – mainly numpy and sklearn for the algorithmic processing causing questions on whether this system could be the most efficient, and since no evaluative techniques were used, the practical accuracy and coverage of this system is unknown. The core theory behind many of the driving motives was clear and well explained built on established theory making this article somewhat reliable.

2.5 Recommendation systems: Principles, methods and evaluation

Source: F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261–273, Nov. 2015, doi: https://doi.org/10.1016/j.eij.2015.06.005

A 2015 paper giving an extremely in-depth and high coverage overview of content-based, collaborative and hyrid filtering approaches on top of evaluation metrics with each analysed individually, meticulously stating advantages and disadvantages alongside a multitude of examples with plentiful recognition of where this information originated through good citing. An extremely important paper that has influenced my own view on recommendation systems, its potential and displaying high foundational knowledge/ core theory with accuracy has allowed me to cross reference other blog/ web articles with this to understand their credibility, but doesn't provide any

implementation within any programming language only using formulas or crude diagrams abstracting the entire system.

2.6 Unveiling the power of Django's MVT architecture

Source: T. Cherickal, "Unveiling the Power of Django's MVT Architecture," *Medium*, May 27, 2023. https://thomascherickal.medium.com/unveiling-the-power-of-djangos-mvt-architecture-6554ff41af64.

A 2023 blog stating the standard architecture used in Django's web development projects – MVT (or MTV) which is structured in 5 main sections, an introduction, sections for each M, V and T, and a conclusion about the architecture as a whole clearly demonstrating each functionality with their corresponding position within the working environment. Given that Django is the web framework to be used, it was extremely important in understanding its architecture, and provided further reason when choosing between Beakr and Django, however touches the surface of its structure and doesn't go into detail of other design patterns that could be fused within this architecture providing limited knowledge of how this structure could be built on top of or how it could be adapted, only stating its advantages within each section. Despite being a blog, no references was used but is accurate when cross reading with other sources emphasising MVT as a best practice when using Django.

2.7 Summary

Sources described in 2.2 and 2.3 contribute to the implementation of content-based filtering which is relevant for my project but in different architectures since R.Van Meteren and M. Van Someren's solution focusing on PRES a user profile mechanism incorporating user models and positive feedback to develop a dynamic recommender system whereas Zeynep Beyza Ayman uses tf-idf and cosine similarity to manipulate item feature similarity to create predictions in python. R.Van Meteren and M. Van Someren's paper is better at giving a concrete foundation but worse at its implementation for my project's scenario which Zeynep Beyza Ayman's solution compensates for but has a lack of foundation, both giving their positives towards developing a content-based recommender system.

The collaborative filtering source described in 2.4 describes a clear implementation technique in python which is uses the same back end language as my project but doesn't provide any evaluation techniques unlike source 2.5 giving extensive descriptions on theory over not only collaborative filtering, but other filtering systems too giving a modern overview of the state of recommendation systems and in which metrics we should try to optimise and focus upon giving plentiful options to adapt even their example algorithms creating a constructive and reliable structure of information extremely useful with my project, even stating its influence in an ecommerce domain with many of its examples and introductory paragraphs.

Source 2.6 provides a crucial explanation of the main architecture used in Django decomposing each section of the MVT architecture into understandable and important statements of use in a practical environment extremely useful when I first started using this web framework.

Chapter 3: **Background Reading**

3.1 State of the Art Web Development

3.1.1 Introduction

Ecommerce has been a major influence over decades to the extent that it has built a market for web builders allowing anyone to create a website using the simplified tools provided however, for major ecommerce companies, sometimes this is not enough to be at the forefront of the industry. Ebay and Amazon have developed techniques since the mid-1990s [25], maintaining lead upon the genre combining state of the art frameworks, machine learning, human computer interaction, buying experience and far more to produce a large profit margin and dominance over the scene to far branch out in multiple sectors not limiting to ecommerce. Getting the framework correct is a major task that starts the journey of creating an ecommerce and must be tailored towards the job at hand to further enhance quality and realistically, sales which is the main metric of a good ecommerce. The edge in my project is the use of recommender systems and therefore frameworks with machine learning and statistical manipulation is a must in choosing the programming language as a base for the web framework.

3.1.2 Django framework

Django mainly being known as a back-end framework has its front-end capabilities through templates containing the html, css and js files allowing built in functionality without any additional main front-end framework and is a very good web framework for python which has many libraries for manipulating statistics and matrix/arrays necessary for recommender systems. Python lends itself perfectly with mathematical based problems, approaching with libraries aimed at both statistical and matrix/array solutions. While further researching collaborative and content-based filtering systems and their approaches, statistical, matrix based and decision trees all complement preexisting python libraries such as sklearn, NumPy, spaCy and pandas allowing computational efficiency and simpler implementation methods superior against other popular options. As the system is an ecommerce website, a web framework is necessary therefore Django with its huge repertoire of websites such as Instagram, Spotify and Pinterest[26], 2 of which are social medias sites with clear examples of multiple usages of recommendation algorithms whether that is to “follow” new friends, a for you page (FYP), finding similar songs of interest and much more I deemed it as a prime candidate to use for the web framework as well as options to collect user data since the data types collected will result in different implementations of the recommendation algorithms.

3.1.3 Beakr framework

Since statistical manipulation is necessary, another base programming language I considered was R. R environment is entirely devoted towards statistical computing even helping analysis through graphical visualization of data, and since it provides many statical techniques and resources for machine learning implementing many of the recommender system techniques would be succinct. R has Beakr as a go-to web framework which provides simple and stable web services which is massively expandable[27]. Beakr contains most front-end services needed with request handling allowing backend communication with R. This combination would allow implementation of an ecommerce needed for my project with the packages manipulating matrices and other structures for recommender systems.

3.1.4 Conclusion

Both frameworks contain all that is needed for my project and would be good candidates however, I chose Django due to its high usage with ease of documentation allowing recovery of mistakes made without losing too much time. I have also personally used Springboot which has a similar architecture

to the Model View Template architecture used in Django so many skills can be transferable with little adaptation making it a logical option.

3.1.5 Databases

In my project I chose Django's default database SQLite which is a relational database with its ease of use and simplicity making it an option for this small project.

3.2 Architectural paradigms

3.2.1 MTV (Model Template View) architecture

Django web framework has a Model-Template-View (MTV) architecture [24]. Template holds the html files displaying the static elements of what will be on the webpage. View is what the user will see after the browser is rendered so is responsible for CSS and JS communication with the html files, the requests the user will carry out throughout the systems runtime and the communication between the database interface (model) to manipulate the database. Model allows the backend to create a structure for the database while acting as an interface and is accountable for maintaining their interactions.

Despite View in MTV render requests to produce a response in a pipeline manor, the request themselves need to be sent and received. This is done through the URL pattern:

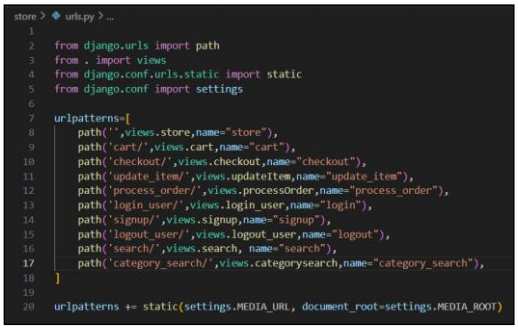


Figure 1 - URL pattern example

Once a request is taken, Django compares the path of request, redirecting it to the corresponding views class that will handle the request and subsequently return the rendered version.

MTV architecture is the main backbone of Django and increases ease of use due to its concrete structure with some flexibility (mainly in View) creating a concrete strategy towards problems which is reproduceable and maintainable.

3.3 Standard Features

3.3.1 Categories

Categories condense all products into specific subsets allowing easy traversal of desired products improving the chance the user finds and buys a product. Amazon leverages this method in multiple techniques:

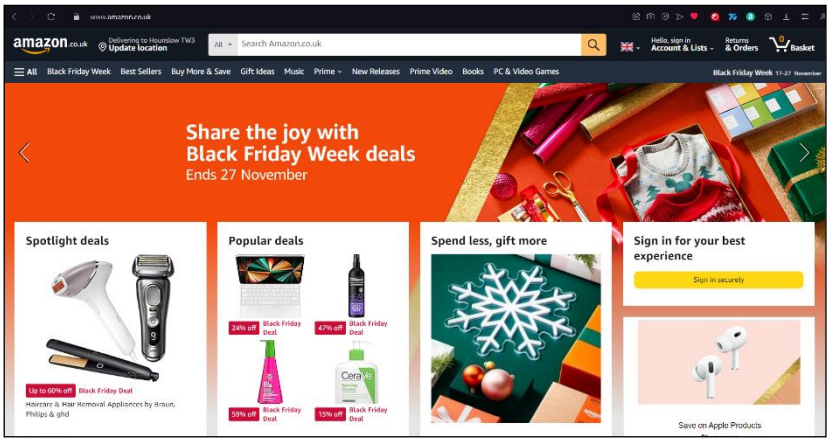


Figure 2 – Amazon.co.uk [1] home page

Firstly, when clicking on the navigation bar 'All' category, a side bar appears displaying all trending features as well as shopping by department using directly the categories of each individual item:

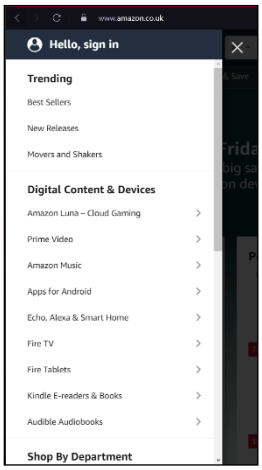


Figure 3 – Amazon [1] 'All' category sidebar

Hence allowing filtration of desired category with ease of access. Other categories on the navigation bar don't display many categories dependent on the products shown i.e., 'Best Sellers' or 'Black Friday Week' which aim for more serendipity and access to offers that users may not usually attempt but relate to many items that the user may have already displayed interest in with previous interactions of the website.

In contrast, eBay adopt categorical filtration dependent on the products they have, also displaying multiple instances of categorical filtration within their main page:

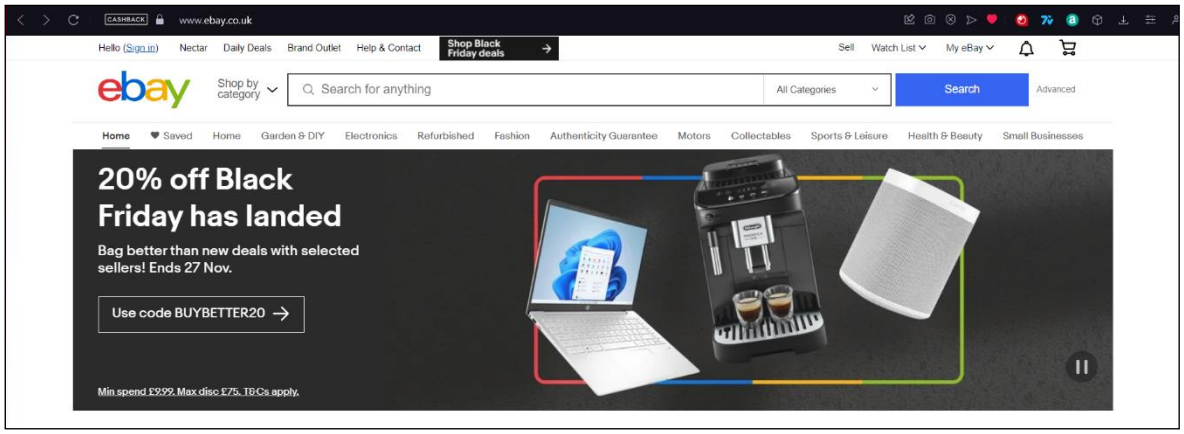


Figure 4 - ebay.co.uk [2] home page

The home pages have very similar design in respect to their logo placement comparative to the search bars, navigation bar and drop-down boxes for categories in the search bar. However, unlike Amazon, eBay uses categorical features dependent on their products and subsequently allow drop down boxes onto their navigation bar that amazon implemented instead as a sidebar:

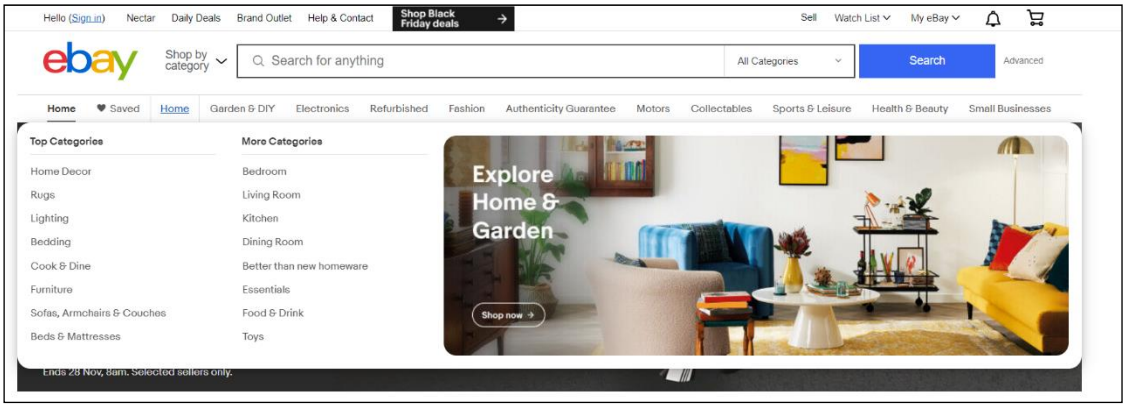


Figure 5 - eBay's [2] drop-down box from navigation bar

eBay therefore greatly acknowledge the use of subcategories throughout each category shown in its navigation bar allowing ease of traversal through this method of subcategorization. The main disadvantage with this method is the prevention of displaying too many categories. The categories displayed on the navigation bar are limited to a select few designed to target the user, therefore a method to display the other categories is necessary which is carried out through the drop-down box left of the search bar:

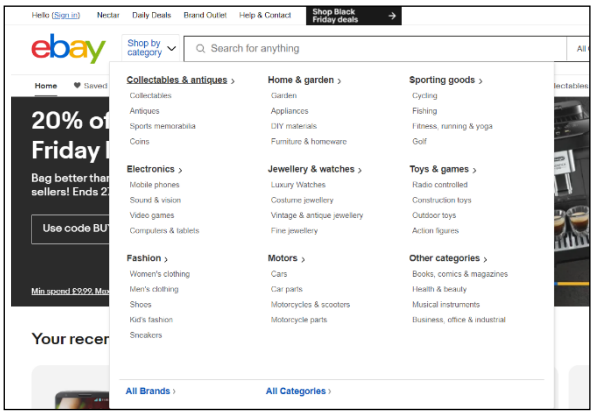


Figure 6 - eBay's [2] category drop-down box

3.3.2 Search bar

Search bars condense through direct access returning a set of items related to the given search request and is a popular feature in most ecommerce websites. Both Amazon and eBay use search bars as an integral feature to narrow the broadness of datasets and products they have to offer but even with non-broad datasets with few categories according to spacebetween.co.uk [4] up to 30% of the visitors will use the search bar therefore is still necessary to implement. Most ecommerce's provide a simple search system:

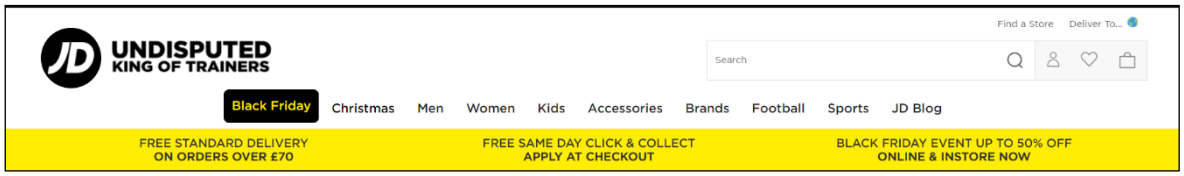


Figure 7 – JD [3] search bar

Allowing the user to enter the desired product or keywords and providing a result back to the user. But Amazon and eBay with their broad datasets, have an additional feature to the standard search system. As displayed in figure 1 and figure 3, the search systems have a categorical filter beside them, allowing the category to also be queried returning a more precise set of products while reassuring the customer their input aligning with the topic they desire.

3.3.3 Basket

A basket/order review is extremely important in reassuring and maintaining trust with the user allowing transparency with products attempted to be bought and their respective price tag.

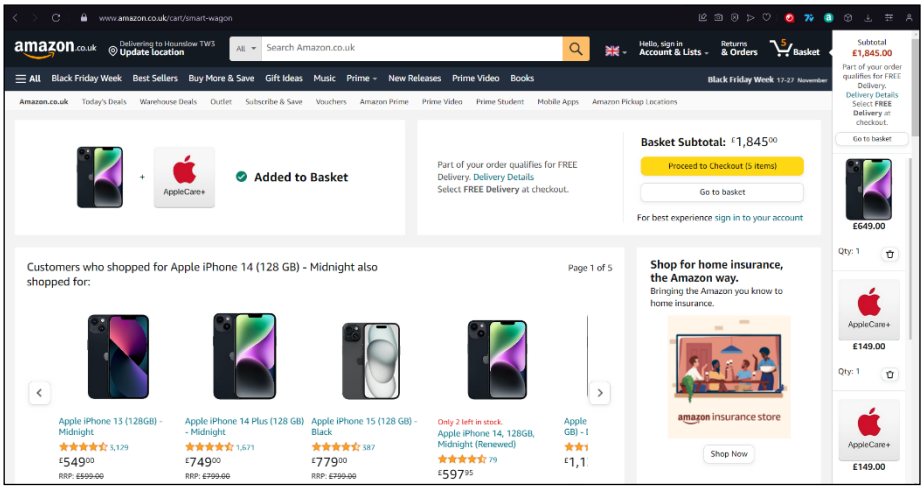


Figure 8 - Amazon [1] basket implementation

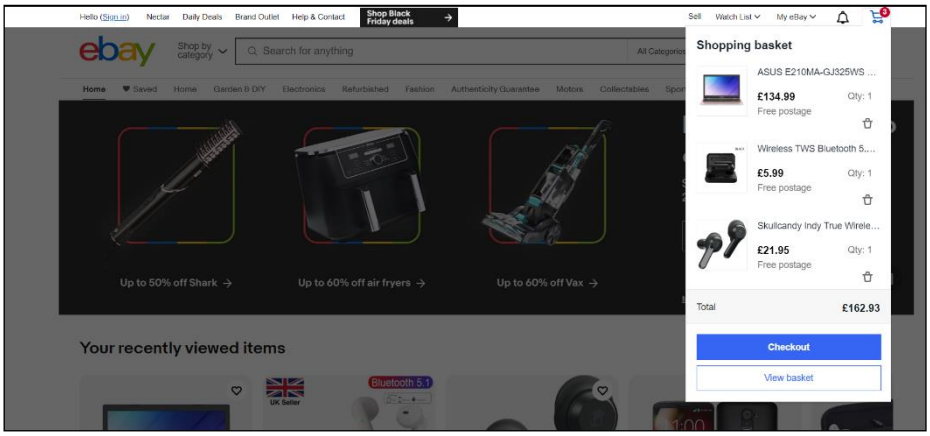


Figure 9 - eBay's [2] basket implementation

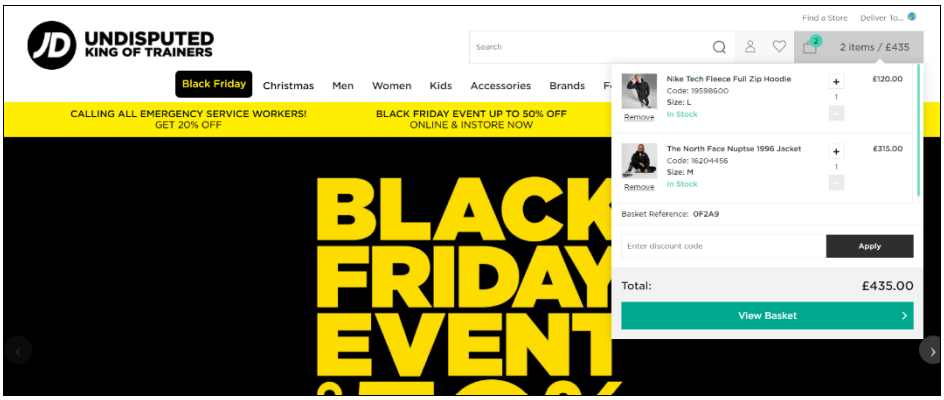


Figure 10 - JD's [3] basket implementation

Most basket implementations work directly on the home page and carry through most pages like the search and navigation bars but can also have their independent pages. JD and eBay implemented their baskets in a similar method as a drop-down box after hovering the cart icon whereas Amazon's implementation consists of a side bar method, but all methods consist of the same content: a list of items, their image, label, price and quantity adjustment options. The separate page shopping basket has a clear architecture expanding the home page basket in a more legible format without adding any extra information but has a checkout option.

3.4 Content-based Filtering Research

3.4.1 Introduction

Recommender systems are a unique example of information filtering systems, they apply filtering algorithms to then give a specific result recommendation to the user. Content-based filtering systems recommend items like ones the user had bought in the past (in our case). A simple example:

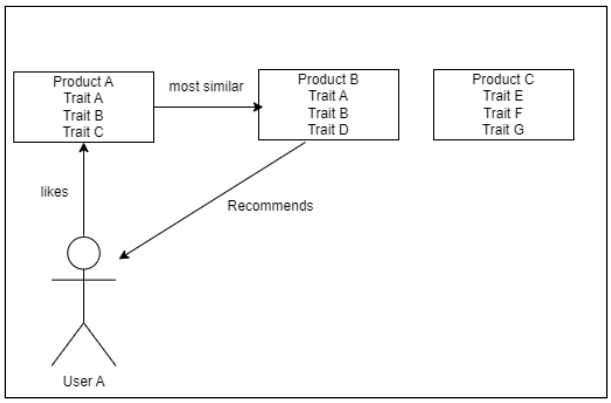


Figure 11 – Simple Content-Based Filtering Example

User A likes Product A, the Content-based filtering algorithm finds the most similar product, in this example is by specific traits where product A and B share the greatest number of traits therefore Product B is recommended. User profiles collect user preferences and past behaviours generally formatted as a vector and is a necessary step for most content-based filtering approaches [5,6,10]. Manipulating a user profile provides a deep history creating more consistent and accurate user interests, allowing a product similar to these interests to be founded as a recommendation, however not all content-based filtering algorithms require user profiling, providing a more spontaneous recommendation on least past action. Regardless of the use of user profiling, content-based filtering compares only products and doesn't need other user information for recommendations allowing versatile user scalability of the ecommerce website serving as a major advantage as well as avoiding the cold start problem experienced in collaborative filtering [14]. Although also providing easier implementation methods, content-based filtering suffers heavily from the domain that it must work from since it must be hard engineered causing struggle to generate from scratch as each product added must be tagged, categorised, priced etc. Requiring a lot of domain knowledge causes technical difficulties causing potential data preprocessing especially while dealing with continuous data and traits with further subcategories since they generally contain a lot of unwanted tokens and punctuations providing certain approaches to be adapted.

3.4.2 Vector Space Approach – cosine similarity

The cosine similarity approach bases its fundamentals that the similarity between two vectors can be computed using cosine similarity, more specifically, for two vectors a and b, the dot product of a and b divided by the product of their lengths formally:

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}$$

Figure 12 - Cosine Similarity Formula [7]

This computes the cosine of the angle, described θ (theta), between the vectors implying the smaller the angle, the more similar the vectors are hence the more similar the products are. Products are not naturally in vector form; they almost always contain words therefore preprocessing is required.

The domain highly influences what type of preprocessing is required but a shared main aim of getting each trait into a state for vector conversion to then be used in the cosine similarity formula. The first step is to convert null values from non-numerical fields into an empty string, preventing errors when

fields aim to be manipulated. Tokenizing each field allows a token soup to carry out frequency analysis therefore allowing vectorization for a set of products however, there are multiple methods in tokenizing and it's important to potentially tokenize differently dependent on the field in question. For example, a product named 'air fryer' and a different product named 'air conditioner' must be tokenized differently therefore in this instance, separating each field as its own token is desired whereas, a product named 'Samsung phone' and a product named 'Huawei phone' do share real world similarities (they're both phones) therefore splitting at a word-to-word basis may be preferred. Since both scenarios impact a single field, both tokenization methods cannot be applied simultaneously fortunately however, preexisting sentence similarity models such as spaCy libraries allow more accurate predictions on the similarity of words and phrases even if they're not grammatically alike such as 'cat' and 'dog' having a similarity score of 0.80168545 [9] as well as giving vector norm scores allowing for modifications to their similarity score algorithms.

3.4.3 Classification Approach – Decision Trees

The decision tree methodology classifies a data set into branch segments constructing rules leading to nodes for further branches or leaf nodes representing the outcome (in our case the products). Like the cosine similarity approach traits must be tokenized however may not require vectorization since splits can be carried out on categorized data by comparing its frequency before and after each split. The effectiveness of this approach entirely depends on the quality of each split/rule determined by the model and its representation in real world scenarios. Seemingly optimal splits may provide arbitrary and meaningless similarities comparative to nodes in the same subtree therefore crucial in the success of this approach. Many preexisting decision tree implementations such as sklearn's DecisionTreeClassifier have multiple built-in methods for splitting the data, mainly: gini and entropy for classification approaches [11].

The Gini approach splits the nodes on all available variables, selecting the split which results in most homogeneous sub-nodes calculated via the Gini impurity formula:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Figure 13 - Gini Index Formula

If Gini = 0, each object is in its respective classes meaning the tree is pure, whereas if Gini is higher then some objects are not classified in its correct class potentially requiring another split. Splitting towards 0 is generally not a good idea and often requires pruning to prevent overfitting allowing more generalization for the model.

The entropy approach, in context of decision trees, is a measure of disorder or impurity in a node like Gini, with similar values as Gini of 0 to represent leaf nodes all belonging to one class and equally divided classes as 1 but using the formula:

$$E = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Figure 14 - Entropy Formula

A major disadvantage of both these splitting methods is that as a classifier approach, discrete data can be used in these algorithms, but the continuous value is in the form of a label meaning it works better on categorical features, which in our case prevents the complete use of fields like price in our predictions hindering usage of data provided. Where calculations test whether a variable would be good to split against a specific value causes issues when continuous data as labels will tend to not have many products of the same value causing overfitting and a non-accurate model.

After the training set is complete and a decision tree model is created, since our input product/data could correspond exactly with the same product, since the product we are choosing has also been used with the training set for the model to avoid recommending the same product (which the user is probably not going to buy again) traversing to its closest leaf node allows the next most similar product to be recommended.

3.5 Collaborative Filtering Research

3.5.1 Introduction

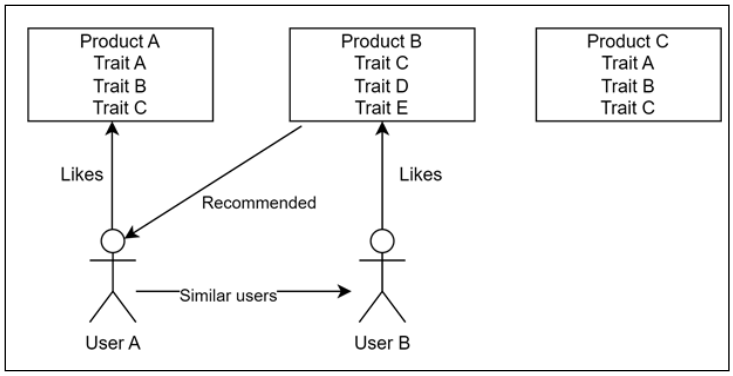


Figure 15 - Collaborative filtering example

Collaborative filtering uses previous user-product interactions, recommending other items other consumers with similar interactions had bought. These historical choices will be recorded. Since it doesn't depend on many product traits, item profiling is not necessary allowing easy product scalability for the ecommerce [14]. To avoid overfitting and directly comparing the primary user to the most similarly interacted user, users are generally grouped via user profiling allowing a consensus when recommending specific items. Collecting user-product interactions can be explicit, where the user bought item A as well as items B, C and D therefore a different user buying item A should be recommended items B, C and D or can be implicit including data points: searches, actions, purchase history on other websites etc. Both of which can be extremely valuable in providing values of user-item interactions in the form of a utility matrix (generally used in most collaborative filtering recommender methods). Advantage of not requiring much domain knowledge allows the lack of individually hard inputting products (at least all the products features) in order to carry out collaborative filtering speeding up algorithm startup time, however once the algorithm has been implemented, a cold-start problem occurs requiring users to train the model adding new user-item interactions as well as providing difficulties to newly added products since they too will not have previous user-item interactions. The model will only be as good as how much information and interactions the user provides. Another advantage over content-based filtering is that the model can help users discover new interests since the likelihood a user having similar tastes proceed to have similar interests provide communication of interests between recommendations.

3.5.2 Memory Based User-User Approach

User-user based memory approach makes predictions as described above, based on a similar user's preferences. The goal is to compute similarity scores between a user and all items, find the top threshold of items and consequently recommend them. To compute the scores, we must contain the rating/interaction values of other users compared to these items. The algorithm I will use is:

$$S(u,i) = \bar{r_u} + \frac{\sum_{v \epsilon V} (r_{vi} - \bar{r_v}) \cdot w_{uv}}{\sum_{v \epsilon V} w_{uv}}$$

Figure 16 - User-user memory-based approach algorithm [12]

Score (S) of user (u) upon item (i) determines the predicted rating the user will give. The main backbone of this algorithm is it will calculate the sum of the ratings of other users (v) on the specified item multiplied by the weight of similarity between user u and user v's profiles. The denominator (sum of weights of similarity between user u and user v) divides the numerator to ensure score is at a suitable range while not biasing towards an individual user. The average of ratings provided by user u denoted $\bar{r}_u$ provides the average of the user scores therefore removing any biases the user tends towards when rating products, i.e. removes biases when consistently ranking high on items even though they may not like it. $\bar{r}_v$ does the same but removes biases that each specific user may influence on their own ratings hence standardising the overall score prediction for user u on item i.

Despite weighting ratings dependent on the similarities of each user, selecting the entire census has more demerits when applied to ecommerce system, requiring a lot of computing power for little improvements in results and would provide a hindrance to user scalability. Therefore, selecting which users to include with these calculations is vital. The following methods can be used as described:

- Select users with similarity scores above a threshold.
- Select the top-N users ranked by similarity scores.
- Select users within the same cluster.

While the first two options provide standard computations already implemented within the algorithm, the last option would require analysis using user profiling to model clusters formed.

3.5.3 Model Based Approach – matrix factorization

This method leverages matrix algebra to manipulate rating matrices to discover features that will serve as the purpose of a recommendation. Preexisting matrix factorization in the context of Collaborative Filtering exist such as Latent Semantic Analysis (LSA), Latent Dirichlet Allocation (LDA), Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) however, I will discuss a method applying PCA and SVD. As the user matrix contains an item at each column, a user at each row and potentially a rating in each cell, the matrix faces a major problem [13]. The matrix has a high order of dimensions causing it to be too big and computationally challenging if put directly into SVD techniques. Fortunately, users generally don't rate every specific product (not practically) causing most user-item matrices to be sparse in nature allowing techniques to reduce its size and simplify into its core features is possible. PCA is a technique for dimensionality reduction removing redundancy as much as possible by finding the most significant components in the data set, extracting these patterns. Once the user to feature matrix (representing user preferences) and item

to feature matrix (representing item preferences) has been split through reduction, we reconstruct back into a lower dimension matrix.

3.6 Evaluation methods of Collaborative and Content-based Filtering

3.6.1 Introduction

Collaborative and content-based filtering exposes major merits and demerits, preventing the consensus of a 'best' approach. But with these merits and demerits, metrics for measurements emerge allowing statistical comparisons accelerating the field into better developments through identifying preferential aspects of a filtering system. The main categories we are interested in optimising are accuracy and coverage of a recommender system. Other categories: serendipity, diversity and novelty will not be covered due to their reliance on large user item interactions grasping a reliable result which is unlikely due to the lack of information I can gather simultaneously relying on data sets for important fields required for evaluation.

3.6.2 Accuracy

An accurate recommender system increases the consistency of producing correct predictions based upon user preferences, in a wider context, delivering personalized content more relatable to the user giving a higher likelihood of buying the product hence on a business perspective is a driving metric in quantifying profit with sustainability. Statistical strategies describing the rate of correct predictions against all predictions demonstrates accuracy therefore approaches like Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are ideal methods:

$$\text{MAE} = \frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} |r_{ui} - \hat{r}_{ui}|$$

Figure 17 – MAE Source https://surprise.readthedocs.io/en/stable/accuracy.html [16]

Mean absolute error takes the sum of absolutes of differences between actual user rating on item and predicted user rating on item and divide that by the total predicted data points. All content-based and collaborative-based approaches returning a predictive rating result can be applied with a history recording all results (or simply splitting the dataset into training and test sets) and can be applied one time for a quantitative result. Through taking averages over a large dataset, anomalous predictive ratings minimally impact the MAE values lending itself perfect as an accuracy measure[14].

$$\text{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2}.$$

Figure 18 – RMSE Source https://surprise.readthedocs.io/en/stable/accuracy.html [16]

Root Mean Square Error has a similar core to MAE, both finding the sum of a difference and dividing by the total number of predictions, however RMSE squares each difference before the sum, and square roots after all calculations. Both formulae aim to reduce the total score closer to 0 and will never produce a negative output allowing somewhat comparisons between these accuracy methods. Unlike MAE, anomalous predictions are more punishing due to the difference being squared

however, deliberately produces higher error scores gives ecommerce businesses the choice to value accuracy with consistency[15,17].

3.6.3 Coverage

Cold start problem, the problem of being unable to predict items that have no previous history or relation to other items is a huge problem specifically with collaborative filtering, coverage calculates the proportion of items and users that a recommender system can predict. Depending on the filtering system implementation, item coverage or user coverage analysis is preferred where item coverage relates to the proportion of items it can predict therefore more suitable for content-based systems with user coverage relating to user-user interactions prediction coverage, more suitable for collaborative filtering.
For item space coverage, since it is the proportion of items predicted over the entire coverage, a simple calculation can be used:

$$cov = \frac{|I_p|}{I}$$

Figure 19 - Coverage formula source: https://link.springer.com/article/10.1007/s13042-017-0762-9

Where Ip is the list of items predicted, its length is calculated therefore giving a direct proportion from 0 to 1, 0 representing the lack of coverage and 1 representing that all items could be predicted.

For user space coverage, using the same method above could be applicable however, with accuracy thresholds, full user space coverage is always possible while accepting low accuracy thresholds and vice versa, an excellent predictor could have high accuracy with low coverage therefore making it extremely difficult to carry out coverage for user-user collaborative filtering. Despite unable to produce coverage statistics for user space, I will carry out coverage among the accuracy measures above next term[14,15].

Chapter 4: **Software Engineering Process**

4.1 Planning

At the start of the project, a planning document highlighting the scope/final deliverables and its timeline was created alongside risks and mitigations however as the process carried out many of the final deliverables and timeline were adapted and even by the first term a second planning document was created.

4.1.1 Original Timeline

To implement all 3 recommendation systems on time, the main features of an ecommerce system must be developed early in the process as a basis for the recommendation systems to be built off, and once the main system is built, the 1st recommendation system using Content based filtering will be implemented. In the second term, the 2nd recommendation system using Collaborative filtering is necessary and serves as blocks of code to combine with the developed Content based filtering system to implement the 3rd hybrid recommendation system.

Term 1

Week 1 – Create basic localized web structure - intended to lay the foundations for Django web framework project including uploading work onto git (since it will also be my first-time using Django web framework, I will also need to research on how to get it working).

Week 2-4 – Implement ecommerce features – sign in/register, search bar, categories, payment system, track order, display items and more.

Week 5-6 – Study in depth a Content based filtering recommendation system – cosine similarity approach for Content based filtering appears most promising for numerical comparisons when calculating the dot product between 2 vectors (features of the product) will produce a value stating its likelihood, and plot description based systems initially for keywords based characteristics (such as category, ingredients etc.) combining all vectors into a data soup to then carry out term frequency to then carry out cosine similarity [20]. More on the algorithms potential design and implementation will need to be thought through.

Week 7-8 – Implement researched Content based filtering recommendation system – this is merged with the above week and hopefully will start drafting on week 6 since this system could require a lot of time to develop if stuck on unforeseen problems.

Week 9 – Upload to webserver - Lower on the priority list but adds finishing touches for the report and presentation in the following weeks.

Week 10-11 – Prepare for interim report and presentation.

Term 2

Week 1-2 – Study in depth a Collaborative filtering recommendation system - user behavior needs to be tracked and finding methods may be the highest difficulty hence added another week for study over Content based filtering system. There are 2 classes of techniques for collaborative filtering: a memory-based approach and a model-based approach where memory-based approaches use techniques such as cosine similarity or Pearson correlation (and other statistical variants too) whereas model-based approaches use singular value decomposition (SVD), neural networks or matrix factorization (and other approaches too). In these weeks I will research on the model-based

approaches since it is easier to be scalable with larger amounts of data than memory-based approaches.

Week 3-4 – Implement Collaborative recommendation system using a model-based approach algorithm.

Week 5-6 – Study in depth a Hybrid approach to recommendation system – research whether merging the 2 previous recommender systems will be possible using a hybrid approach otherwise research on methods that use one of the developed recommender systems and can be merged with another that I will have to implement from scratch. Since hybrid approaches mainly aim to fix problems from either the Content based filtering approach or the collaborative filtering approach there are many different options that can be taken to create a desired result. Most prominent is the cold start problem which occurs specifically in collaborative filtering approaches hence research could be needed to ensure optimal results on the recommendation algorithm.

Week 7-8 – Implement Hybrid recommendation system – merging both preexisting algorithms together to improve overall effect of the algorithms as a hybrid or creating a new algorithm and merging it with a preexisting algorithm to create a hybrid algorithm.

Week 9-11 – Prepare for final report – 70% of this course depends on the final report hence deserves 3 weeks to meticulously work on it.

4.1.2 Adapted timeline

Due to the overestimation of how quickly I could learn a framework I hasn't worked with before on top of not leaving enough time for the basic structure of the ecommerce the general structure of the project was converted to getting the standard features of the ecommerce and research upon content-based and collaborative filtering done so that heading into term 2, a content-based and collaborative filtering technique could be implemented.

Term 1

Week 1 – Create basic localized web structure - intended to lay the foundations for Django web framework project including uploading work onto git (since it will also be my first-time using Django web framework, I will also need to research on how to get it working).

Week 2-6 – Implement ecommerce features – sign in/register, search bar, categories, payment system, track order, display items and more.

Week 7-9 – Study in depth both Content based filtering recommendation system – cosine similarity approach for Content based filtering appears most promising for numerical comparisons when calculating the dot product between 2 vectors (features of the product) will produce a value stating its likelihood, and plot description based systems initially for keywords based characteristics (such as category, ingredients etc.) combining all vectors into a data soup to then carry out term frequency to then carry out cosine similarity [7] and Collaborative filtering system - user behavior needs to be tracked and finding methods may be the highest difficulty hence added another week for study over Content based filtering system. There are 2 classes of techniques for collaborative filtering: a memory-based approach and a model-based approach where memory-based approaches use techniques such as cosine similarity or Pearson correlation (and other statistical variants too) whereas model-based approaches use singular value decomposition (SVD), neural networks or matrix factorization (and other approaches too). In these weeks I will research on the model-based approaches since it is easier to be scalable with larger amounts of data than memory-based approaches.

Week 10-11 – Prepare for interim report and presentation.

Term 2

Week 1-4 – Implement Content-based Filtering System improving the model and decide where to display recommended data.

Week 5-8 – Implement Collaborative recommendation system deciding where to display recommended data and how it will integrate with the rest of the system.

Week 9-11 – Prepare for final report – 70% of this course depends on the final report hence deserves 3 weeks to meticulously work on it.

4.1.3 Deliverables

The project objective is to make an ecommerce website with specifically the requirements of:

- Search bar (must enable at least direct access to inputted specific items)
- Sign in/ register (taking minimum email, password)
- Account details (profile, saved payment systems, options to accept personalized experiences)
- Payment system
- Categories tabs (displaying specific categories of items for easier access to grouped related items)
- Front page (and each page) including images and succinct information of goods/ deals / events etc.
- Page(s) displaying products after clicked on chosen category

Also implementing recommender systems:

- Content based filtering system
- Collaborative filtering system

4.2 Design

4.2.1 UML state diagram

Understanding the flow of the website is essential in the design process for me as the developer and to minimize any redundancies or complications that could have effects on the user. As seen from other ecommerce sites, most of its functionality is displayed on the first page to give simplicity for users to reduce the number of clicks needed to reach from one section of the site to another, therefore I have attempted to minimize the number of choice pseudo-states (diamonds) to reduce the number of decisions the user has to make. The first UML state diagram was made for the first term, describing aspects and layout of the basic ecommerce functionality among login and signup:

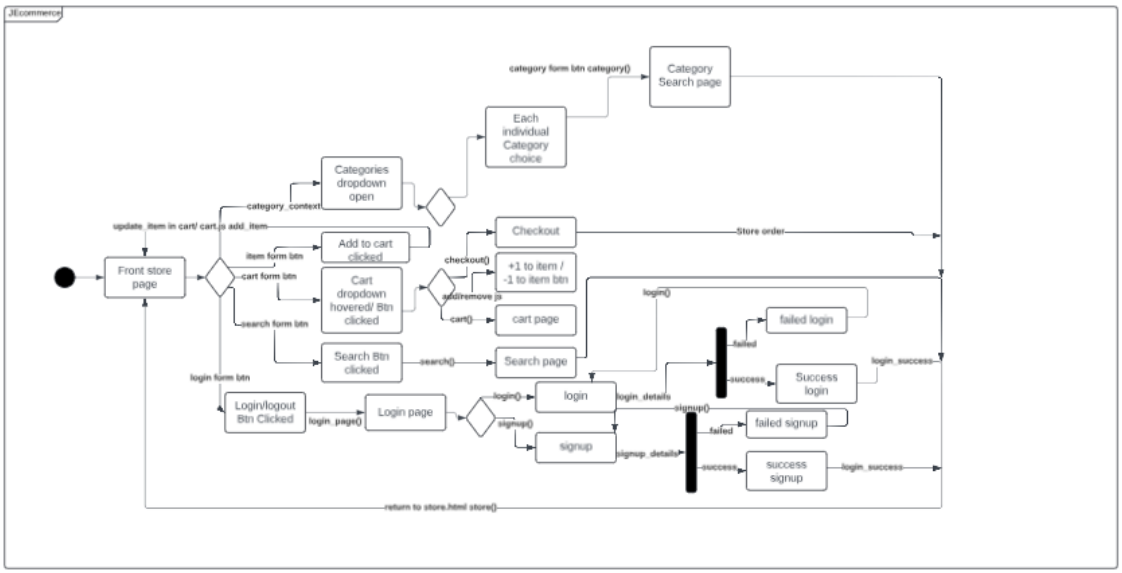


Figure 20 - Term 1 UML state diagram

Many states include dropdown menus which change the state of the front page minimally while giving users new information that they deem necessary for example with the categories dropdown which was planned to drop individual category choices rather than sending them to individual pages.

The initial project plan document had collaborative and content-based filtering on a separate for you page to give users options however while creating the UML state diagram for term 2, this implementation method seemed long winded and with further research upon other ecommerce websites, recommender systems mostly naturally integrate themselves within the shopping experience and isn't a separate step. This also relieves the need for the user to search for items they might potentially want, therefore giving them recommendations on the same page. Since I used 2 recommender systems, I didn't want both implementations to be on the same section of the program/page, therefore implementing one on the front page dependent on the items added into the basket, and one on the checkout page to take the user basket as a whole and create a user profile meaning the term 2 state diagram changed minimally:

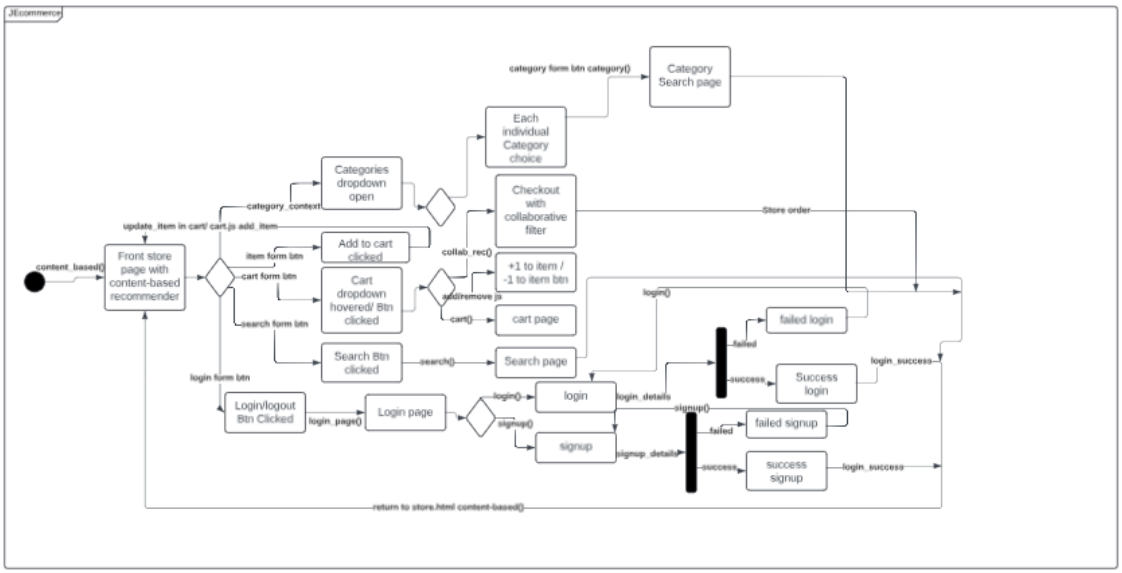


Figure 21 - Term 2 UML state diagram

The state of the front page includes content-based recommender with the state of the checkout page using collaborative filtering recommender hence integrating themselves minimally to the overall structure of the system. Another benefit is if the deployment strategy was phased, then seamlessly integrating this feature allows preexisting users to not get confused by a change in UI design and structure.

4.2.2 Database schema

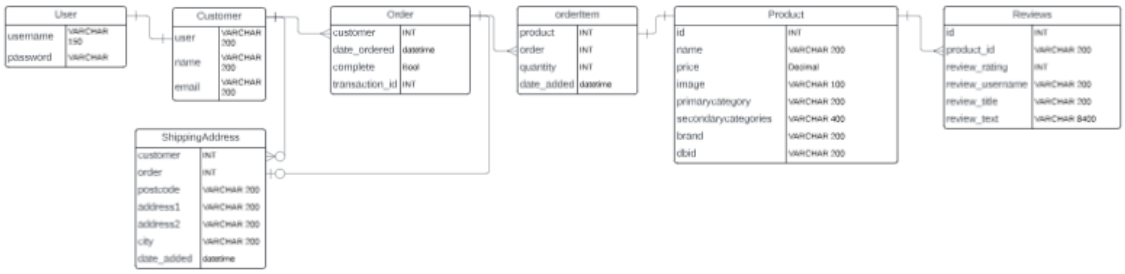


Figure 22 - Database schema

User and Customer contain information about the user which is currently limited to username, name, email and password. For guest users, they will have to input an email address at Order phase to (in the real world) keep track of their order and serves as a good identifier, however in my project is not acted on any further. Product holds information about each item but differs from orderItem which takes the product chosen by the user for a specific order and attaches a quantity for multiplicity. To maintain incomplete orders, a complete field is included to order and allows the user's basket to remain even in different sessions, therefore shipping address connect to Order but only after it has been processed.

Reviews is needed for collaborative filtering and contains the user's name, rating and review text, however since this is taken from a dataset, the review_username may not be of usernames in user, preventing any foreign key from being created between the two tables. If review information was collected, then will be modified slightly to fit the situation.

4.2.3 UI Design

Ecommerce websites have many similar/ standard features that makes their UI design an experience transferable between other ecommerce's, and to maintain that structure I will incorporate as many standard features as possible in my own work. As described before, the front page must have many features, however this mustn't hinder its tidiness, so a balance is aimed. A dropdown box is widely used in almost all ecommerce sites combined with categories in the navigation bar for those with more specific products (i.e. a clothes shop may have men, women, children as categories built into the navigation bar since they're frequently used filter terms). Since my dataset is more generic, I will only use a drop-down box:

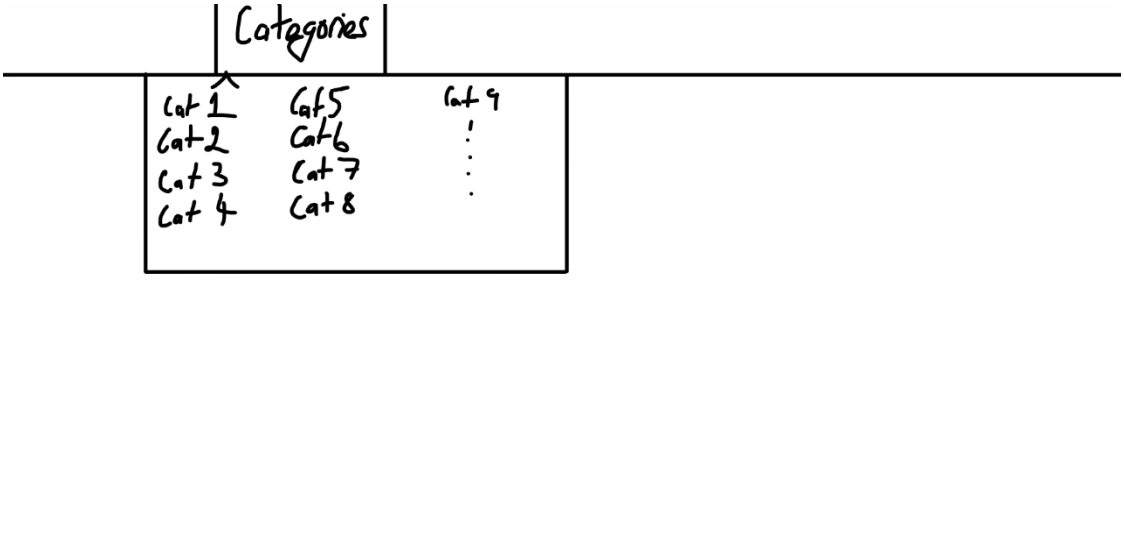


Figure 23 - Categories UI Design

As another filtering system is by search and must be incorporated in the front page. A typical implementation is to put the search bar either at the top right corner or near the centre (generally never at the top left due to the ecommerce's icon located their), however the cart will be implemented on the far right, therefore the search bar will be to its left:

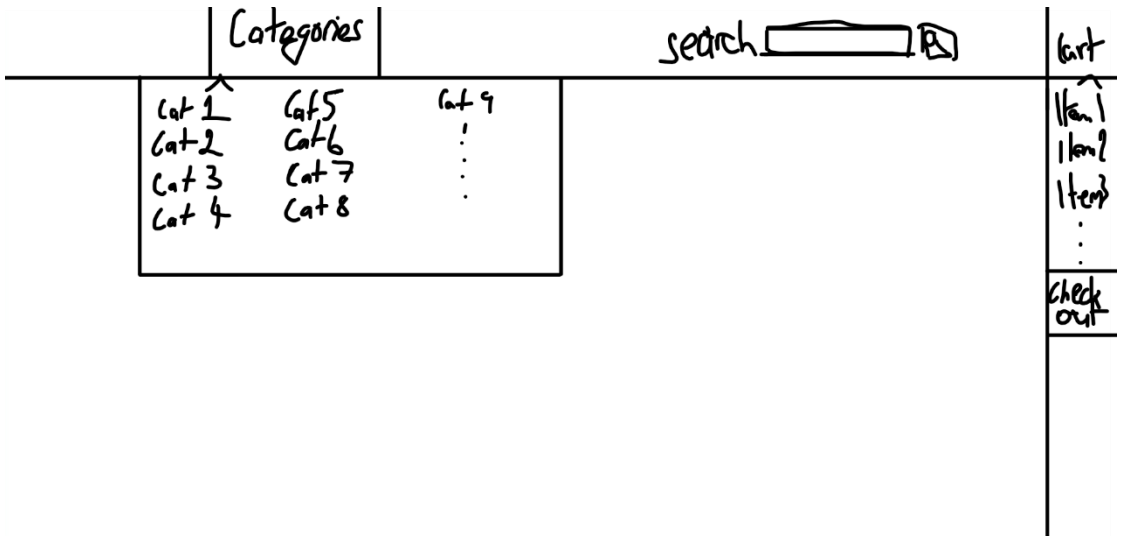


Figure 24 - Categories, search and basket UI design

The items will hence be displayed under the navigation bar, displaying their image, item name, price and method to put into basket.

For the implementation of a recommender system, where the user will see its results is also extremely important but can also be influenced by its technique. For the content-based filtering system, since I'll use an item profile to make the predictions, it makes sense to display the results on the front page since each addition of item to the user's basket will result in different recommendations to be created. Since the collaborative filtering system will use a user profile, creating results after a variety of items get put in the basket, allocating checkout to display it seems sufficient. Recommendations on the front page are generally not on the front row, unless flash sales or enticements are attached to catch the user's attention therefore acting more as announcements. Due to this, putting the recommendations on a lower row (i.e. third or fourth) allows the user to look at items available on the front page first, then have a look at recommendations. Lower than the fourth row, and there's a chance that the user will never see them since it may be 2 scroll pages lower rendering it practically difficult to access which doesn't help the user nor help profits for the company. For recommendations on the checkout page, amazon carries out a separate page before the checkout page designed to give users the choice of items, they deem would like, giving a final chance serendipity check before the user buys their basket, however others just implement it within the final buyout phase as well, so seems to be an implementation choice. I chose to put it within the payment page since I can limit the number of items recommended without losing the number of options the user may want.

4.3 Development

4.3.1 Standard features

Term 1's development goals were to complete the standard features and making the basic ecommerce features. The category dropdown, search bar, cart, login and logout as well as basic item display were created with the front page:

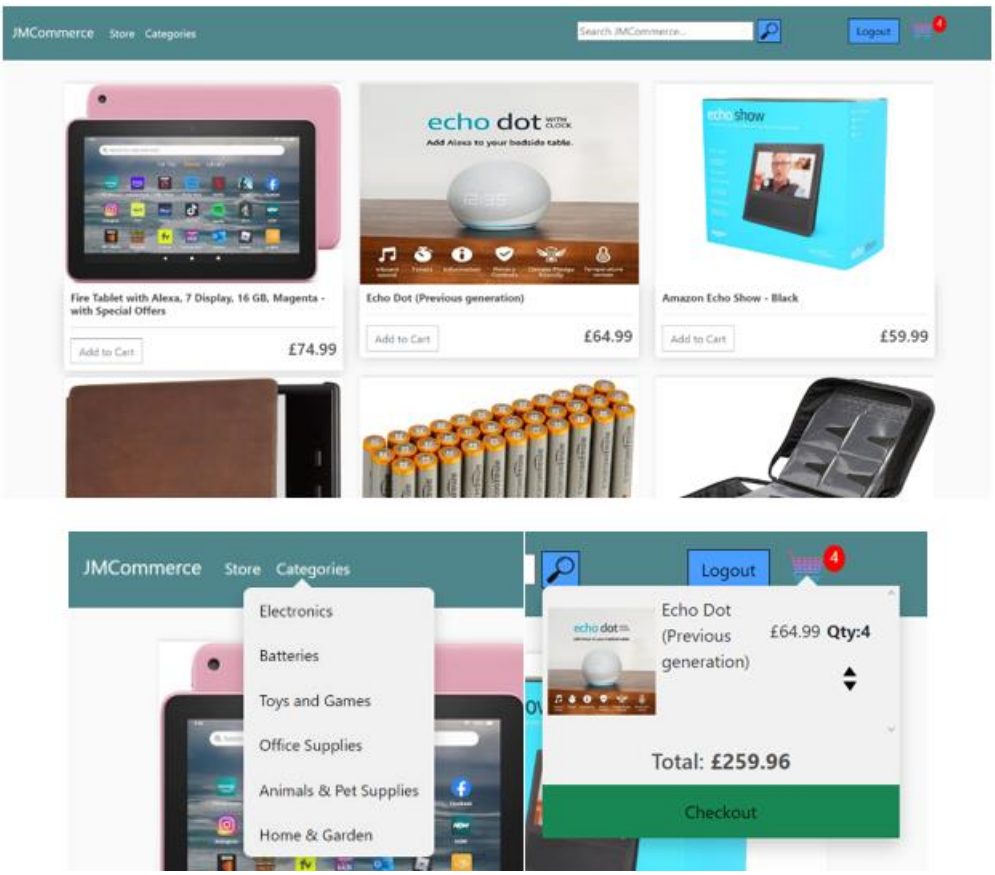


Figure 25 - Item display, category display and basket display

The goals on the front page were mainly executed, with changes made to make sure correct formatting for example all prices were made sure to 2 decimal places:

```
<h4>£{{product.price|floatformat:2}}</h4>
```

since there were tendencies to have a recurring number despite no manipulation from the database value to the front end.

```
{% for category in categories %}
  <form method=POST action="{% url 'category_search' %}>
    {% csrf_token %}
    <div>
      <input class="searchcontainer" type="hidden" name="categorysearch"
      value= "{{category.primarycategory}}"/>
      <button class= "categorydisplay" type="submit">
      {category.primarycategory}}</button>
    </div>
  </form>
{% endfor %}
```

The categories were implemented dynamically, therefore taking the distinct of all product's categories and manipulating it into its separate context allowing front end usage which once clicked, will filter only items of said category. Similarly, the search bar carries out almost the same comparing the name of the item to inputted item but allows leniency with not being the same word by using __contains, filtering the products based on if the name contains the user searched query.

```
def search(request):
  .
  .
  .
    if searched == "":
      products = Product.objects.all()
      context['products']=products
      return render(request,'store/store.html',context)
    products = Product.objects.filter(name__contains = searched)
    context['products'] = products
    context['searched'] = searched
  return render(request,'store/search.html',context)
```

The up and down buttons in the dropdown of the cart menu allows change of quantity for each item and was implemented using JS. Since both guest users and authenticated (logged in) users needed to be provided for, the implementation of the basket and hence the change quantity buttons were different. For an authenticated user, an Order variable stored current items and dynamically updated it in the database (with complete field as false until payment) allowing users to come back from sessions with their basket remembered. For guest users, since they are not a user of the ecommerce, to maintain scalability options, their data cannot be saved the same way as authenticated users since it could lead to the database being flooded by bot account that could easily access the website hence creating an order each time it was accessed which could lead to Denial Of Service Attacks. Therefore cookies, local to the guest user's personal browser are used to store the cart therefore allowing update of quantity through a simple cookie while allowing the user to close the page and reopen without losing their progress (provided they don't delete their cookies from browser).

```
if(user==='AnonymousUser'){
  if(action=='add'||action=='adds'){
    if(cart[productId]==undefined){
      cart[productId] = {'quantity':1}
    }
    else{
      cart[productId]['quantity']+=1
    }
  }
```

```
  if(action=='remove'){
    cart[productId]['quantity']-=1
    if(cart[productId]['quantity']<=0){
      delete cart[productId]
    }
  }
  document.cookie = 'cart=' + JSON.stringify(cart)+";domain=;path=/"
  location.reload()
}
.
.
.
```

This allows dynamic addition and removal of user items for guest user to be stored within a cookie.

4.3.2 Content-based filtering system

The content-based filtering technique I implemented was the cosine similarity method finding similarity between items to recommend most feature similar ones. Features in this project were hand selected based upon fields given from the product dataset used totalling to 4: product name, product categories, product secondary categories and product brand. In human nature, we like items that have similar names to items we already like creating a prime candidate for a feature in a content-based filtering system, just like categories (and the more in-depth secondary categories). Brands (especially in clothing) have high influence on whether the user may buy it or not hence is also included as an additional feature. Other features such as price is also applicable however was not chosen as I wanted the features for this task to be noticeable when results of similarity occur. The cosine similarity method requires numerical values which all 4 features are not based upon, hence creating a matrix to depict our item-feature relationship is carried out through term frequency - inverse document frequency (tf-idf), however before we can use a count vectorizer, to make sure all words are of equal state, we must use preprocessing:

```
def character_preprocessing(word):
  preprocessing_table =
  str.maketrans({'@':'','%':'',':':'',',':'','(':'',')':''})
  word = word.lower()
  word = word.translate(preprocessing_table)
  return word
```

For each word we make the word lower case since upper case doesn't provide substantial context meaning and we use translate() to take the string preprocessing_table and return no character in its position effectively removing it. It is important to remove some of these punctuations but especially the comma since the secondary categories is comma delimited and after converted to text the comma will remain present, therefore products with equally many secondary categories will be evaluate slightly more similar than no influence (since tf-idf algorithm isn't linear and sorts this out to a certain extent).

The count vectorizer finds the frequency of all the words in a feature and put it into a word soup giving values to the nature of the words in each item. Noticeable for product name, stop words was incorporated because specific words like to, a etc. Give minimal impact to the overall meaning of the words and are common terms within other product names however, since the likelihood they're apart of all product names is slim, therefore tf-idf may not detect this leading to a substantial value given for a word with little semantic value and hence is removed using stop_words = english. On the contrary, if stop words were incorporated within brand or categories, they would give context value since we can assume they would be already filtered/not included hence to not accidentally remove stop words that may have meaning, we do not check for stop words within these features.

```
def cosine_similarity_recommends(orderItem):
  all_processed_product = text_preprocessing()
  name_processed_product = all_processing_product[0]
  primary_processed_product = all_processed_product[1]
```

```
  secondary_category_product = all_processed_product[2]
  brand_category_product = all_processed_product[3]
  PV = CountVectorizer(stop_words = 'english')
  SV = CountVectorizer()
  BV = CountVectorizer()
  NV = CountVectorizer()
  .
  .
  .
```

Once the vectors are created, used the cosine similarity method and then combined them.

```
def cosine_similarity_recommends(orderItem):
  .
  .
  .
  name_cosine_similarity = cosine_similarity(name_matrix)
  primary_cosine_similarity = cosine_similarity(primary_matrix)
  secondary_cosine_similarity = cosine_similarity(secondary_matrix)
  brand_cosine_similarity = cosine_similarity(brand_matrix)
  all_cosine_similarity = name_cosine_similarity +
  primary_cosine_similarity + secondary_cosine_similarity +
  brand_cosine_similarity
  id = orderItem.product_id
  orderItem_cosine = all_cosine_similarity[id-1]
  li = []
  for i in range(len(orderItem_cosine)):
    li.append([orderItem_cosine[i],i+1])
  li.sort()
  sortIndex = []
  for x in li:
    sortIndex.append(x[1])
  sortIndex.reverse()
  return sortIndex
```

The first for loop goes through the orderItems and attaches the corresponding id value since the values were taken from the database in id order i+1 is attached to each orderItem_cosine[i] value for list li. Li is then sorted regarding orderItem_cosine[i] hence ordering the similarity values from smallest to largest. Since we only care about the ids that this corresponds to, we can disregard the evaluation hence x[1] is included, which is then reversed to give the highest to lowest evaluations.

```
def store(request):
  .
  .
  .
  if request.user.is_authenticated:
    customer = request.user.customer
    order, created = Order.objects.get_or_create(customer=customer,
    complete=False)
    orderItems = OrderItem.objects.filter(order=order)
  else:
    order=[]
    for item in context['items']:
      order.append(item['product']['id'])
    orderItems = OrderItem.objects.filter(product__in = order)
  context['recommends'] = cosine_similarity_recommender(orderItems)
  return render(request, 'store/store.html',context)
```

Retrieving the items from either the guest user or authenticated user is different therefore orderItems must be created to create equal grounding, allowing both types of users to receive recommendations as indicated in with the context['recommendations'] which will be manipulated on the front end:

```
{% for product in products %}
  {% if forloop.counter0 == 6 and recommends %}
  <div class = "recoCon">
    <br/>
    <h3>Recommended Items based on Items in Your Basket</h3>
    {% for recommend in recommends %}
    <div class = "col-lg-4">
      <img class="thumbnail" src="{{recommend.image}}"/>
      <div class="box-element product">
        <h6><strong>{{recommend.name}}</strong></h6>
        <hr/>
        <button class="btn btn-outline-secondary add-btn update-cart"
        data-product={{recommend.id}} data-action="add">Add to
        Cart</button>
        <h4 style= "display:inline-block; float:right">
        £{{recommend.price|floatformat:2}}</h4>
      </div>
    </div>
  {% endfor %}
  </div>
  .
  .
  .
```

Since each row contains 3 products, I put the products on the 3rd row using the forloop counter to position it giving an outcome of:

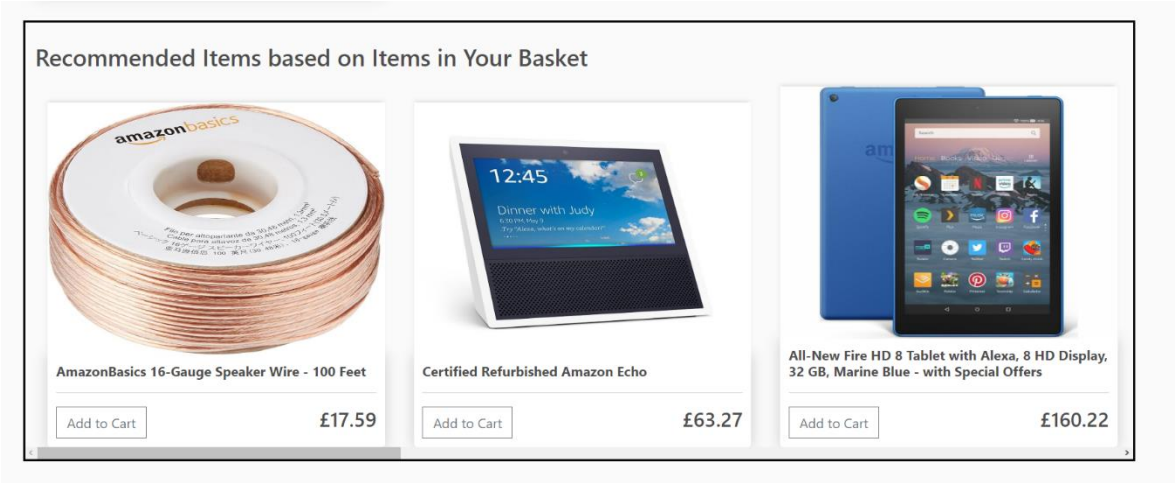


Figure 26 - Content-based Filtering system horizontal scroll display

4.3.3 Collaborative filtering system

The collaborative filtering recommender I implemented is the model-based approach: matrix factorisation. Similarly to the content-based filtering technique, this method also requires a matrix but of usernames, items and corresponding review rating. A major issue I encountered was the use of my dataset. The dataset I have used [19] after reducing it to 50 select products contains over 20000 reviews and despite knowing that this will create a sparse matrix and utilizing this to found an approach (SVD), the filtering method would still take roughly 3 minutes to create a recommendation, which is too long and breaches performance quality. Therefore, as a band-aid solution, I sampled a random 1200 reviews using the random library python has built in (1200 was chosen considering performance and was adapted after the system was built, I initially chose 200).

```
def collabFiltReco(request):
```

```
  context = userAuth(request)
  review = Reviews.objects.all()
  usernames = review.values_list('review_username',flat=True).distinct()
  random_sample = random.sample(range(0,usernames.count()-1),1200)
  reader = Reader(rating_scale=(1,5))
  distinct_username_list = list(usernames)
  sample_names = [distinct_username_list[i] for i in random_sample]
  sampled_users = review.filter(review_username__in = sample_names)
```

I used the Pandas library to influence a lot of the matrix construction using pandas DataFrame structure which is needed when using Surprise library's SVD method. Hence using this information, the matrix is formulated:

```
User_item_matrice = pd.DataFrame({'usernames':[u.review_username for u in
      sampled_users],'items':[i.product_id for I in
      sampled_users],'rating':[r.review_rating for r in sampled_users]})
```

This contains each sampled user's username, the item that they rated, and the rating from the pre-existing user's reviews. To make recommendations on our user and their existing basket, we create a user profile containing their username, items and rating (which is set to 4). Extracted this data is different for both guest and authenticated users but similar and will conform with the other reviews in the DataFrame and hence can be concatenated onto the rest of the reviews DataFrame for the SVD process.

```
items = context['items']
customer_item_matrice = pd.DataFrame
cim = []
if request.user.is_authenticated:
  usersname = str(request.user.customer)
  for item in items:
    item_quantity = item.quantity
    rating = 4
    item_id = item.product.dbid
    cim.append([usersname,item_id,rating])
else:
  usersname = "anonymous"
  for item in items:
    item_quantity = item['quantity']
    rating = 4
    item_id = item['product']['id']
    cim.append([usersname,item_id,rating])
customer_item_matrice =
pd.Dataframe(cim,columns=['usersnames','items','rating])
user_item_matrice =
pd.concate([user_item_matrice,customer_item_matrice],ignore_index=True)
```

To use the surprise SVD module, we must convert the DataFrame into a dataset, which can be done using another surprise module, surprise Dataset. This allows the use of building a trainset and testset necessary for the model-based approach.

```
Data =
Dataset.load_from_df(user_item_matrice[["usernames","items","rating"]],
reader)
algo = SVD()
trainset = data.build_full_trainset()
algo.fit(trainset)
testset = trainset.build_anti_testset()
predictions = algo.test(testset)
```

4.4 Testing

4.4.1 TDD

Test-driven development was used throughout the process both on the front-end and the back end.

Test-driven development starts with a test that describes the wanted functionality which will initially fail. We then make the test pass using faking, and finally refactor it to maintain the passed test for new tests that also describe the functionality. Since these tests describe what the program should do, passing these tests means that we have working code.

Before both front end and backend tests, setup is required which adds a test database of a single user, customer, 2 products, 2 order items and a shipping address.

```
def setUp(self):
    self.client = Client()
    self.signup_request = {'email':'test_user@email.com','name':'test_user','username':'test_user','password1':'password321',
                           'password2':'password321'}
   # self.form = RegisterUserForm(self.test_request)
#    self.form.save()
    self.customer = Customer.objects.create(
        name = 'test_name',
        email = 'test_name@email.com'
    )
    self.product1 = Product.objects.create(
        name = 'product1',
        price = 19.99,
        image = 'https://www.webfx.com/wp-content/uploads/2021/10/generic-image-placeholder.png',
        primarycategory = 'category',
        secondarycategories = 'secondarycategory1, secondarycategory2',
        brand = 'brand'
    )
    self.order1 = Order.objects.create(
        customer = self.customer,
        transaction_id = 1,
    )
    self.order_item1 = OrderItem.objects.create(
        product = self.product1,
        order = self.order1,
        quantity = 2
    )
    self.product2 = Product.objects.create(
        name = 'product2',
        price = 10.08,
        image = 'https://www.webfx.com/wp-content/uploads/2021/10/generic-image-placeholder.png',
        primarycategory = 'category',
        secondarycategories = 'secondarycategory1, secondarycategory2',
        brand = 'brand'
    )
    self.order_item2 = OrderItem.objects.create(
```

Figure 27 - TDD setup method

This allows tests to directly check these cases for whether certain items/orders/user exist as they should whilst controlling the environment making it easier to debug/ identify aspects of the code that isn't working as intended, however, to ensure that these objects can be created properly, model tests are required to make sure that there is a basis for setup and the database:

```
#Model Testing
#Test 1
#Ensures setup is correct, and Customer model correctly updates
def test_setUp_customer(self):
  self.assertEquals(self.customer.name, 'test_name')
  self.assertEquals(len(Customer.objects.all()),1)
```

By checking the customer.name = 'test_name', this ensures that the setup is properly running, with the Customer model being updated as intended since its length is now 1. And is repeated for all other models and tests the methods within these models such as the Order model's getCartItems and getCartTotal which both are crucial features of the basket implementation and display:

```
#Test 6
#Checks properties of Order model getCartTotal and getCartItems
def test_order_properties_model(self):
  self.assertEquals(self.order1.getCartItems, 5)
  self.assertEquals(self.order1.getCartTotal.__float__(),70.22)
```

To test both the front end and views.py backend, I initially tested if the required page existed:

```
#Views Testing
#Test 2
#Tests the login view gets the correct template login.html
def test_login_GET(self):
  response = self.client.get(reverse('login'))
  self.assertEquals(response.status_code,200)
  self.assertTemplateUsed(response, 'store/login.html')
```

This page needs to be returned and not from a redirection hence the assertEquals with response.status_code = 200, and for further knowledge of which page it used, we placed the template that we want to be the login page.

```
#Test 3
#Tests existing user with correct credentials can successfully be logged
in
def test_login_correct_POST(self):
  form = RegisterUserForm(self.signup_request)
  form.save()
  user = authenticate(username='test_user',password='password321')
  self.customer2 = Customer.objects.create(
    user = user,
    name = 'test_name',
    email = 'test_name@email.com'
  )
  test_login_request = {'username':'test_user', 'password':password321'}
  response = self.client.post(reverse('login'),test_login_request)
  self.assertEquals(response.status_code,302)
  self.assertRedirects(response, '/')
```

Test 3 tests whether a user with correct credentials properly redirects to the main page. Giving login functionality for a user with correct credentials but is faked by returning the successfully logged in page without considering the username nor password. However, test 4 prevents faking by testing a username and password combination that fails causing a refactor to occur to allow both tests to pass incorporating views.py file (backend manipulation) to properly both check whether the user exists in the database and properly directs to a page given the results.

```
#Test 4
#Tests existing user with incorrect credentials cannot successfully login
def test_login_incorrect_POST(self):
  test_login_request =
  {'username':'test_false_user','password':'password321'}
  response = self.client.post(reverse('login'),test_login_request)
  self.assertEquals(response.status_code,302)
  self.assertRedirects(response,'/login_user/')
```

4.4.2 Unit Testing

Unit testing tests individual logic units ensuring their behavioural functionality works. The aforementioned logic units in my project refers to functions and was used to mainly tests the backend functionality and could've been used in front-end testing however most tests would've been overlaps or repeats of tests done in the TDD phase, therefore was not included.

Especially with the recommender algorithms, unit testing was necessary to ensure that their functionality was working therefore tests such as:

```
def test_Character_Preprocessing(self):
  processed_word = character_preprocessing("t@e%s:t,i(n)g")
  self.assertEquals(processed_word, "testing")
```

character_preprocessing simply aimed to remove any of these characters @%:,() (since they would give no contextual meaning rendering it irrelevant with our recommendation system), and therefore feeding a word with all uses and asserting it to its expected outcome – testing, reassured me that this function carried out its purpose with high coverage but not full coverage since a control test nor boundary tests had been carried out, therefore:

```
def test_Character_Preprocessing_Control(self):
  processed_word = character_preprocessing("testing")
  self.assertEquals(processed_word, "testing")

def test_Character_Preprocessing_BoundaryFront(self):
  processed_word = character_preprocessing("%testing")
  self.assertEquals(processed_word, "testing")

def test_Character_Preprocessing_BoundaryEnd(self):
  processed_word = character_preprocessing("testing%")
  self.assertEquals(processed_word, "testing")
```

This provides full coverage allowing character_preprocessing function to be assured quality if the event the function using this function breaks. This process was repeated for all the filtering system algorithms. I also attempted to use this for the updateItem function which changed the quantity of an item for an authenticated user, however since the function used json.loads(request.body), the request context had to be manipulated as a HttpRequest() to instantiate request.body but this created a further problem which was not resolved. HttpRequest() doesn't allow instantiation of request.user which holds the current user information even if client.forcelogin(user) is used preventing simulatenous manipulation of request.body and request.user which this function had used preventing any coverage from taking place.

Chapter 5: **Technical Decision Making**

5.1 Content-based filtering system

Before developing a singular system, choosing the system itself took deliberate effort analysing tools and materials available to create a practical recommender algorithm. The main 2 deeply researched methods included the vector space approach - cosine similarity and the classification approach – decision trees but many factors enticed me to use the vector space approach. Despite this project being relatively small in terms of products collected, a major disadvantage of decision trees lies with limited power to learn complicated rules making scalability a difficulty. Especially with an ecommerce domain, products can be added, removed, updated (i.e. price) on a frequent basis and with ecommerce giants like amazon, scalability in their recommender systems is a must. With tf-idf for vector space approach, text processing analysis is magnified creating a deeper understanding of the features given, lending itself perfect for this project which contains fields of mainly text values and doesn't have issues with overfitting (but can be mitigated using pruning for the decision trees).

Choosing the vector space approach, despite researching about the cosine similarity approach, there are other similarity algorithms available such as Euclidean distance also being another commonly used metric. Cosine similarity focuses on the angle between two vectors, whereas Euclidean distance concentrates on the straight-line distance between two points in space. As a result, cosine similarity is better able to capture pattern similarities between two data sets, even when their magnitudes vary making it more robust, therefore was chosen for my project.

Using stop words within the count vectorizer process depended entirely on the feature at hand. Despite conclusively only having product name to have stop words, all features were considered to initially have stop words but was corrected since if brand, primary category and secondary categories have stop words within them, it is most likely of context significance therefore should be retained as a coupling to the preceding or succeeding word(s)/ token(s).

Regarding the display of the content-based filtering system, many considerations took place such as where (i.e. which page and where on the page) to display the results and how the results should be displayed. A lot of inspiration was taken from other pre-existing solutions which collectively, seamlessly blended the results into its page making it obvious that they were recommendations but not separate from other objects or sections. Recommendations in general are mostly shown on the front page (while shopping), and near the checkout page (like the real shopping experience when items are put above the checkout conveyer belt) which I wanted to mimic, therefore since the content-based filtering technique considers individual items, updating the basket and putting it on the front page seemed logical. As for the position, most ecommerce either place it on the sides (like an advertisement) or integrate it between other items on a separate row or both. I decided to just implement one of these methods since my product dataset was not big enough (only 50 different products) creating a situation where multiple of the same product may be recommended creating a lack of novelty. Since my technique could support a rating given to all other products for each product, showing just one product doesn't fully show the extent of accuracy the algorithm had become on the client's side, therefore allocating a row for the recommendations was decided.

5.2 Collaborative filtering system

Collaborative filtering systems mostly rely on a user profile which can be collected using a plethora of data provided by the user i.e. ratings given by the user when reviewing a product to clicks on products, time spent viewing specific items, and other meta data. For my project the user profile was merely based upon the items inside the user's basket due to the lack of interaction between the user and the program (since I had no people to collect data on and no datasets existed connected to the

pre-existing reviews dataset used [19]) which could potentially cause one of collaborative filtering system's major problems: the cold start problem. Therefore, to create a user profile that had sufficient data to be acted upon, basket items were used to create the user profile with each item in the basket containing the rating 4. The rating was chosen as 4 due to the premise that if an item is in the basket and the user enters checkout (this system is displayed at checkout), the likelihood that they have confidence in the items they want to buy is high and therefore can be corresponded to a higher rating than 3, but is not chosen as 5 due to not knowing whether the user will actually like the product or not. I considered manipulating the quantity of the product in basket to create a more accurate rating such as using a constant which I will call c_1, multiplied by the log of the quantity q added by a different constant c_2 (roughly between 3 to 4) to create: rating = $c_2 + c_1 \log(q)$ however realised this could exceed the maximum rating 5 for bulk buys with large quantities (which could be fixed by setting an upper bound), but the final reason why I chose not to proceed with this method was because sometimes the quantity doesn't correspond to how much someone likes the product for example at a market, oranges could be sold singularly, but you are not likely to buy a singular orange, and might buy multiple (especially if you are buying for multiple people) therefore doesn't correspond to how much you like the product or not meaning if the quantity exceeds 10 in our example rating formula above, an entire c_1 would be added to rating which might dramatically change the overall rating value hence changing the user profile drastically. With a limited user profile, unfortunately this also made the options available for a collaborative filtering system limited, and ultimately ended up mostly using the model-based approach, matrix factorisation but needed some adaptations inspired by the user-user memory-based approach. After creating the matrix factorisation algorithm, the major issue that occurred was the performance over a large review dataset. Since the dataset contained over 20000 user-item-rating entries, iterating through all of these was time consuming to the extent that it took 3 minutes for a recommendation which is not acceptable. Memory-memory based approach takes a select number of users which I implemented in my own work to limit the number of users to 1200. This was chosen mainly based upon performance since it lowered it to almost instantaneous (< 1sec), allowing the user to not take too much notice, any higher i.e. 2000 users made it roughly 3 seconds but varied largely since most of the high reviewing users had been selected at this point causing similarly large user-item-rating dataframes to that of every entry in the Reviews dataset.

5.3 Scope of the project

Due to the lack of initial knowledge and competency with a new framework, delays had occurred causing a risk to the success of the project. As the project manager, I had to decide how this would be resolved, therefore I considered the project management triangle:

Figure 28 - The project management triangle source:
https://en.wikipedia.org/wiki/Project_management_triangle

Changing the quality of the deliverables rarely contributes to the success of the project and most of the time makes the project fails which is illustrated on the project management triangle, therefore other constraints must be considered to improve this situation. Since the time was constricted, either scope would have to be reduced or cost would have to be increased. Increasing cost would generally

be to either introduce some technology or technique that would speed up the process allowing all the scope to be delivered on time or (most of the time) is to just hire more people to complete the job. Since this project is a single person final year project and hiring someone else to do the work with me would be a violation of plagiarism, the only other option was to reduce the scope, therefore choosing which deliverables to keep was at highest priority.

At the time of the decision, I had almost finished the standard features of the ecommerce, and therefore keeping these deliverables was necessary. This includes:

- Categories
- Search bar
- Login/logout
- Basket display
- Order preview
- Checkout and payment

Therefore, the remaining unimplemented features were:

- 2 Content-based filtering systems
- A collaborative filtering system
- A hybrid filtering approach
- Filtering system evaluation methods

Since the extra content-based filtering system was an idealistic addition, as a low priority deliverable, it was the first to be dropped from the scope. As for the remaining 4 deliverables, choosing which should be carried out was a difficult task since they were all on the same priority band, however experiencing a lack of knowledge on the topic which resulted in this situation to begin with, I deemed it important to understand content-based filtering system and collaborative filtering system, therefore the hybrid filtering approach and evaluation methods were also taken out of the scope. The decision carried out allowed these 2 filtering systems to be given enough time to be implemented fully.

5.4 Design patterns and architecture

Django itself has many tools lending itself exclusively to a model template view architecture, therefore made sense to keep this key characteristic of Django for web development. Understanding this and considering I had chosen a bottom-up vertical slicing approach to implement individual use cases/ deliverable goals, models.py containing all the database models must be implemented before coding the front end. A singleton design pattern for the models could be implemented overriding the creation every time a new one is created, however I chose to implement all models using the factory design pattern since specific models like OrderItem and Product require multiple different instantiations at the same time. For other models that could potentially have required a singleton, like User and Customer, I am not scared for multiple instantiations therefore am not worried for it being of factory format as it doesn't expose the creation logic meanwhile prevents global access which would be available in a singleton keeping security on these models.

Chapter 6: **Critical analysis and discussion**

6.1 Project process

The project scope changed drastically throughout each term mostly due to the lack of knowledge coming into the first project plan document which went completely overboard with the number of deliverables in the time span available, causing a lot of them to be reduced/removed for example I initially wanted 2 content-based filtering system, a collaborative filtering system and a hybrid approach giving myself a mere 2 weeks to complete the main ecommerce structure while doing all other University work at hand reduced to a content-based filtering system, a collaborative filtering system and the main ecommerce system. If I had structured the project more realistically from the start it may have been possible to include the hybrid approach however due to the time spent reorganising the project scope, more deliverables had to be removed which in my opinion was one of the biggest failures of the project since I would've liked to create a hybrid approach which is at the forefront of recommender systems currently or evaluated the pre-existing systems on different metrics (that would've been more exciting to me). If I were to carry out this project again, I would unexpectedly just focus on a single filtering technique whether it is collaborative filtering, content-based filtering or hybrids approaches and create 2 different algorithms allowing time for evaluative techniques near the end of development, in detail I would carry out the basic ecommerce structure over the first 4 weeks, research 2 techniques/implementations and evaluative methods of a single filtering approach over another 3 weeks and start implementation of one technique near the end of the first term, finishing it in the 2nd week of the 2nd term then implementing the last technique over 3 weeks, leaving another 3 weeks for evaluative methods therefore leaving enough time for both reports. Ultimately, most of the deliverables were executed creating my current system that I can accept.

6.2 Findings and conclusions

Ecommerce has very similar styles only adapted by the domain/genre of the products sold and in its raw form is just a shopping experience formulated online to maximise profits. Many of the ecommerce features imitate other ecommerce's allowing a main backbone to the shopping experience (just like supermarkets contains isles and shelves structured in the same way no matter which supermarket you go to – unless it's Ikea) but is crucial when maintaining user interest and engagement since they don't have to learn something new to interact with the website therefore maximising profit. These include containing search bars, categories, carts, checkouts, navigation bar, login/logout features and more which I have illustrated within my own work. Cosine similarity approach for a content-based filtering systems can create many predictions from compact datasets with little performance issues especially with relatively small product databases and features (I used 50 products with 4 features each) however can be further optimised using matrix adaptations like PCA and SVD which wasn't tested in this project, but the potential was there. Model based matrix factorisation approach for Collaborative filtering created noticeable performance issues above 1200 user-item-rating entries is not impossible to go over this number (the calculation time increases a lot after each 1000, at 1200 produced <1 sec however at 1500 had produced < 3secs for the algorithm to be carried out) therefore for systems with many user-item-rating entries, potentially a different algorithm to carefully select similar users and compare their ratings is needed. Other than performance no other metrics were evaluated preventing any deep analysis of the techniques used, and pre-emptive precautions prevented analysis of classic issues occurring especially in collaborative filtering methods such as the cold start problem since a dataset with many user-item-ratings were used mitigating many of the problems which should have been experienced first hand to understand its issues. With many accuracy evaluations available such as RMSE and MAE described in chapter 3.6, with more time a deeper analysis of the recommender systems could be carried out, however

comparing them seems meaningless despite them both being recommender algorithms, their inner functionality and purposes are extremely different meaning the analysis may only further highlight its discrepancy, not giving any indication of how the methods could be improved and in which metrics.

6.3 Deliverable analysis

6.3.1 Standard features – categories

The category section was implemented according to how I intended – using a drop-down box and near the top left corner, however with only 6 primary categories, I could've added secondary categories to potentially boost the number of filters possible to further accelerate the buying process for users who are looking for items with that secondary category tag.

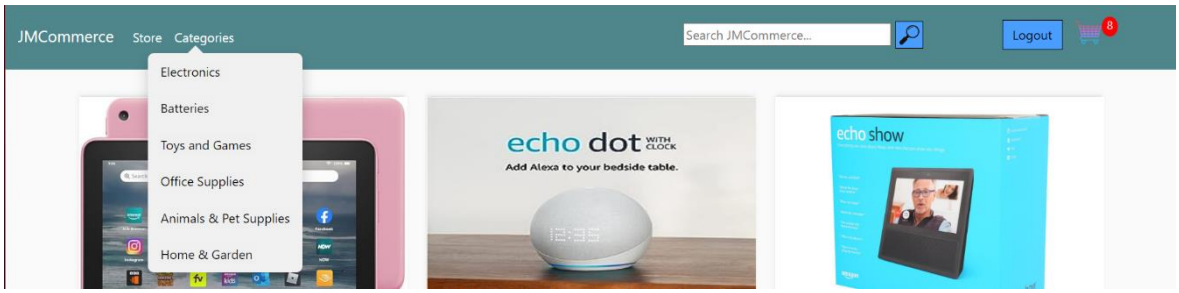


Figure 29 - Categories drop-down box

The overall aesthetic of simple plain colour was also planned since it an ecommerce containing a variety of different products, but if an overarching theme or brand was added such as sports or JD, a colour scheme could be added onto it as needed.

It also dynamically and correctly changes to its corresponding page meaning if more products were to be added to the database, no adjustments are required in the code.

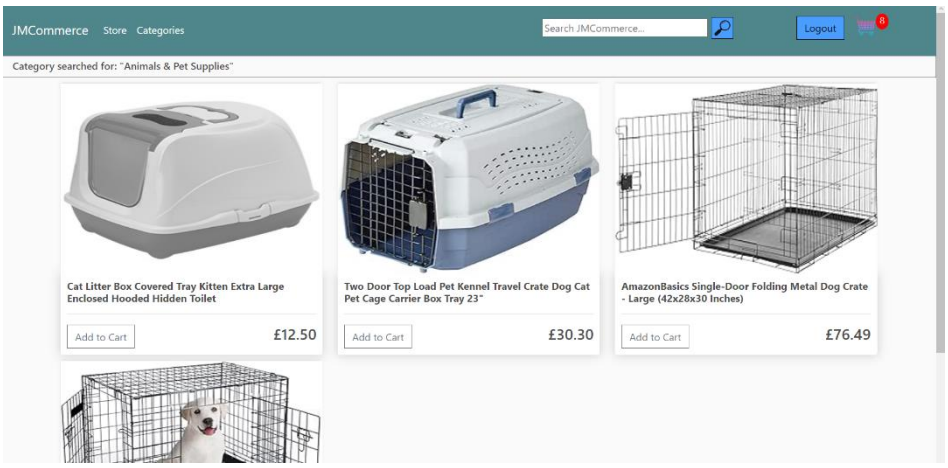


Figure 30 - Categories page when 'Animals & Pet Supplies' is clicked

6.3.2 Standard features – search bar

Search is implemented similarly to category search, using the same filtering method however searches the name__contains allowing direct access for specific items:

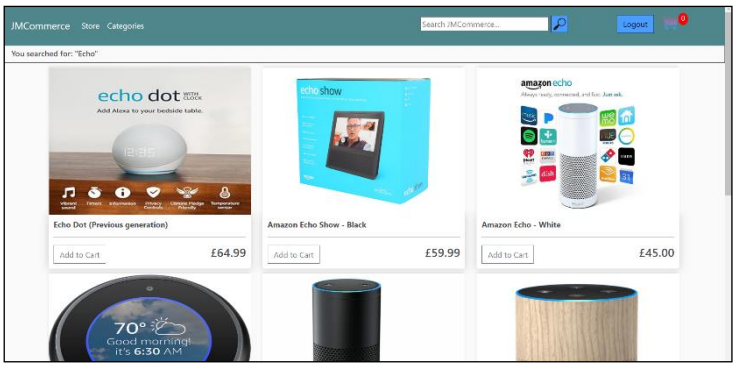


Figure 31 - Search page when 'Echo' is searched

In the example above, echo was searched with the system returning all products with echo in its name allowing customers to select items from this subset to potentially buy. As from the normal product display, the category display and search display all example carry out consistent styles with same colour schemes for generalisation throughout the system enabling familiarity for recurrent users. Despite using name__contains in the search query, it only finds if the query is a subset of each term in product names which could be adapted since mistypes is not accounted for which other ecommerce systems have on top of sometimes a recommender system inside the search bar itself, giving a drop-down of recommendation options. But overall, provides basic functionality.

6.3.3 Standard features - cart

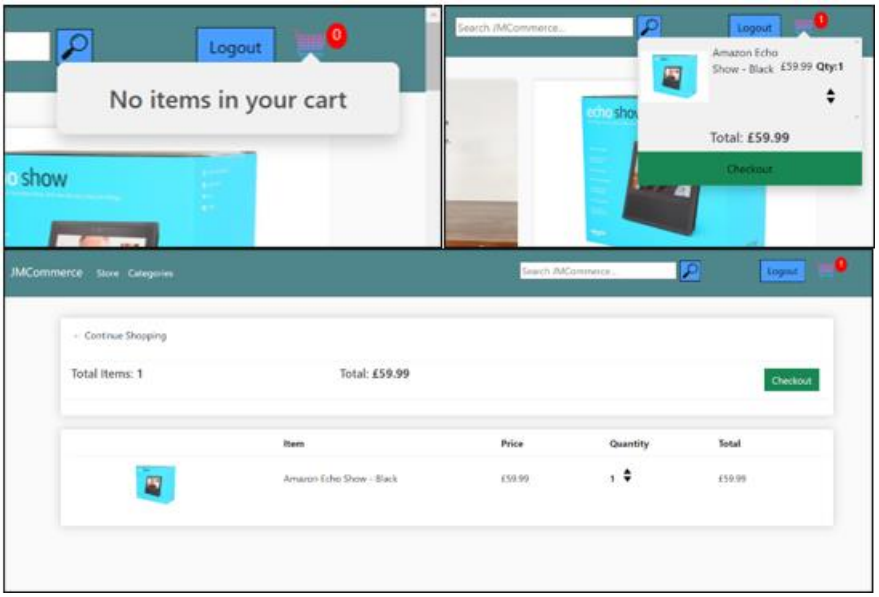


Figure 32 - Basket drop-down box display with Basket page display alongside empty basket display

Considering multiple implementations of basket, a drop-down cart display directly from the home page (or any page) is a popular option and one that I have implemented. Since the drop-down menu scales down every aspect of the cart providing a succinct summary, at times users may want to check their entire basket in detail therefore a second technique has been implemented (clicking the basket displays a simple page of all items). Both methods allow addition and subtraction of quantity for both types of users, albeit if they're a guest or an authenticated user which required different implementation. Rather than displaying a total of £0 with a checkout, no items in your cart clarifies the user of any misconceptions while the standardised green button for checkout allows the user to complete their order from both the home page and the extended basket page allowing easy flow towards their shopping experience. Buttons for quantity are interactive and use previous logic shown

in figure 17 to both add and remove items from the cart. Preventative measures disallow dropdown baskets to continue downwards making the total and checkout button hard to access, therefore a hard cap to the max-height of the items themselves is introduced in the CSS:

```
.cartdropdown{
  max-height:500px;
  overflow-y:scroll;
}
```

6.3.4 Payment Process

Despite being the only feature that hasn't been implemented fully, orders can be processed and stored with the address of the user. Modern ecommerce paired with offline stores often integrate order processing for both guest users and logged in customers. Due to the carts being implemented as a cookie, adding guest users is a seamless process but must add their name and email as additional fields when checking out.

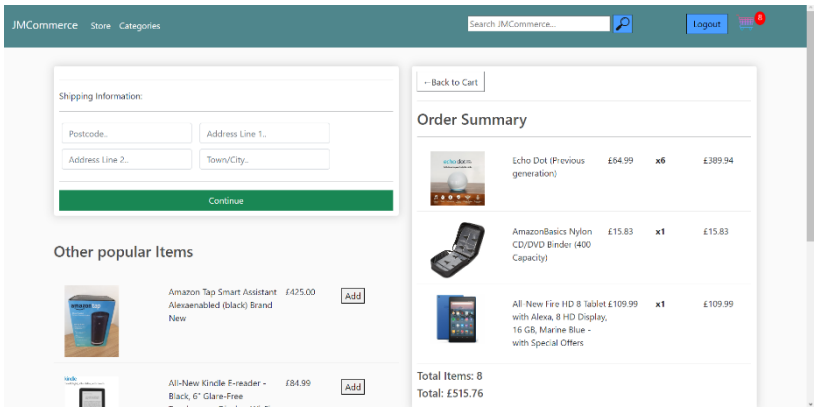


Figure 33 - Checkout with logged in user

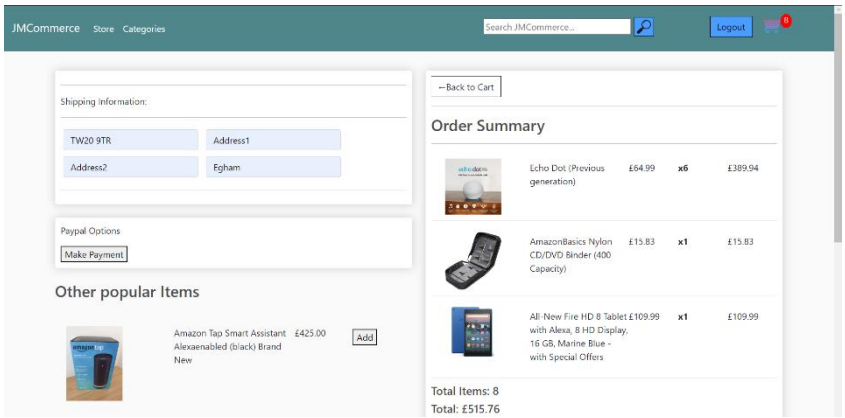


Figure 34 - payment display

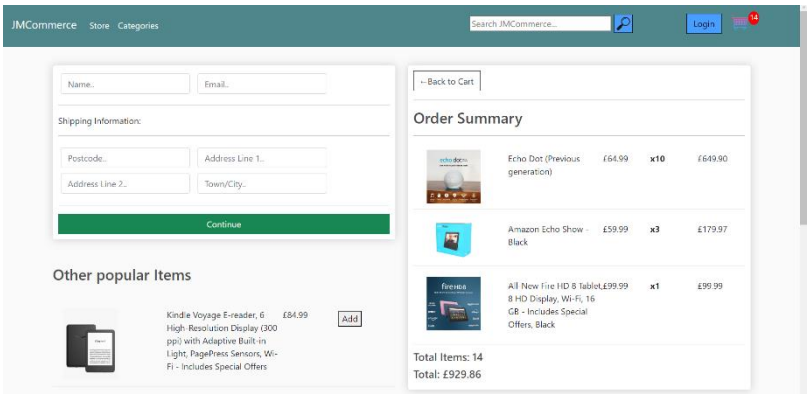


Figure 35 - guest user checkout

In figure 25, payment display describes paypal options, which I did experiment with however, a license is required for websites and paypal themselves must acknowledge a trustworthy site to minimise scams. Since I neither have a developer license and I'm working on localhost, the payment api blocks even the template integration preventing usage. I could have potentially taken the user payment details/ given boxes for them as proof of concept however no meaningful application would be carried out and therefore was decided against until proper implementation was available. The order will be successfully stored in the database alongside their name, email and shipping details.

6.3.5 Sign up, Login and Logout

Django has built in libraries for login namely Django.contrib.auth where we can import authenticate, login and logout libraries providing succinct and safe implementations of a login system.

```
def signup(request):
    if request.method == "POST":
        form = RegisterUserForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data['username']
            password = form.cleaned_data['password1']
            email = form.cleaned_data['email']
            name = form.cleaned_data['name']
            user = authenticate(username = username, password = password)
            customer, created = Customer.objects.get_or_create(
                email=email,
            )
            customer.user = user
            customer.name = name
            customer.save()

            login(request,user)
            messages.success(request,("Registration successful"))
            return redirect('store')
        else:
            messages.success(request,("Registration not successful"))
    else:
        form = RegisterUserForm()
    return render(request, 'store/signup.html',{'form':form,})
```

Figure 36 - Signup implementation

RegisterUserForm takes the inputted user information checking validity while authenticate ensures uniqueness of user information preventing duplication of usernames allowing the user to be saved.

```python
def login_user(request):
    if request.method == "POST":
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(username=username, password = password)
        if user is not None:
            login(request,user)
            return redirect('store')
        else:
            messages.success(request,"There was an error logging in, try again...")
            return redirect('login')
    else:
        return render(request,'store/login.html',{})

def logout_user(request):
    logout(request)
    messages.success(request,"Successfully logged out")
    return redirect('store')
```

Figure 37 - login and logout implementations

Authenticate method in login checks if the username, password combination exists and logs in if successful. Logout is an easier implementation since the user must already been in a logged in state for this method to be called allowing 3 lines to carry out this process. Messages a separate implementation similar to context dictionary that is passed through however gives streamlined replies towards errors messages which is displayed as shown below in the yellow box:

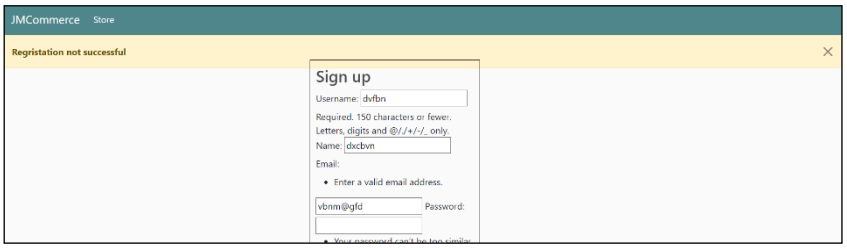


Figure 38 - message box

Using Django.contrib.auth methods additionally allow forms to give feedback on which element of an incorrect registration is wrong, providing clear bullet points for the user to fix.

6.3.6 Content-based filtering system

Using cosine similarity method, the content-based filtering system looks positive however I would've liked to carry out some of the evaluation techniques to understand its accuracy numerically on this domain since I understand that specific items are recommended since they look similar but no further which is a disappointment, however I like its implementation on the page:

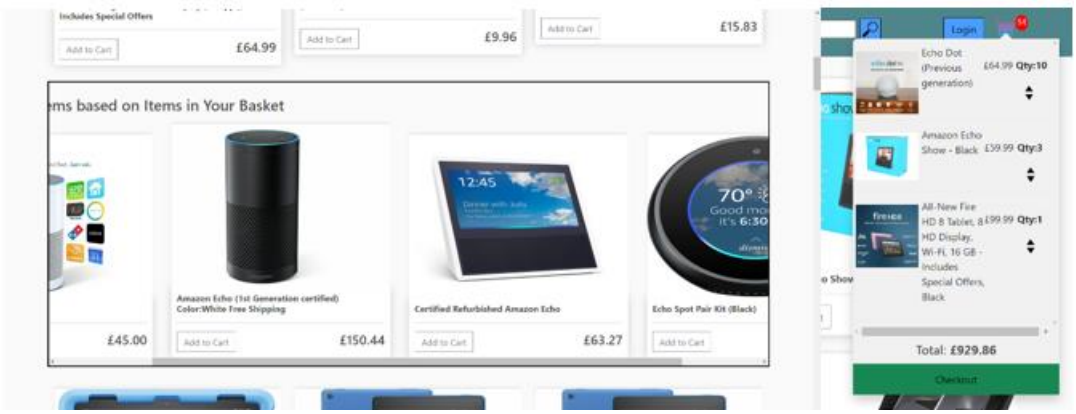


Figure 39 - Content-based Filtering system results display using items from this basket

Using a side scroll bar to allow users to further look at the recommendations give a neat look to the website while not bombarding users with recommendations if they are not interested.

```
def cosine_similarity_recommender(orderItems):
  products = []
  for orderItem in orderItems:
    reco_items = cosine_similarity_recommends(orderItem)
    products+= Product.objects.filter(id__in=
    (reco_items[1],reco_items[2],reco_items[3]))
  return list(set(products))
```

Additional changes were needed to make the user experience better (and slight bug fixes). This function used to take the top 3 highest similarity products to return to the user as recommendations however the top product was itself which wasn't intended, therefore products added started from 1, then 2, then 3 to disregard the 0^{th} element. Another nuance added was to add newly recommended items onto the front of the context therefore each additional item added to basket directly changes the first 3 so users can dynamically see their changes without needed to use the scroll bar for their newer product recommendations. Finally, 2 or more items could have the same product in their top recommendations and therefore duplicates would be added. To fix this I just added list(set(products)) to remove any duplicates since one of the traits of the set structure is to not have any duplicates, and list was used to convert it back to the original structure. So, a lot of effort was made to make sure the implementation for the user side was logical yet creative. The only other change that was missed was the potential need for stochastics, since despite the algorithm doing what is intended, for each user who enters the same items in the basket, the recommender system will output the same items which doesn't give any serendipity, an important metric for human interaction since we all crave something new.

6.3.7 Collaborative filtering system

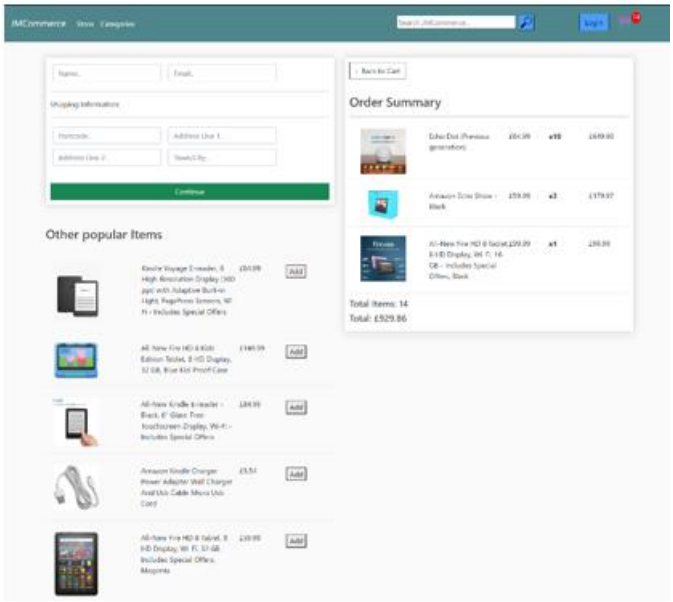


Figure 40 - Checkout page with Collaborative Filtering display

5 recommendations are given to the user on the final checkout page, giving them options to choose from based upon their basket as a user profile. The algorithm implements as intended although like the content-based filtering system, I would've liked a numerical value of accuracy/ other metrics rather than seeing that similar items are being recommended (which might not be accurate for collaborative filtering systems). One nuance that I like however is by introducing sampling, which was originally a fix for performance, the same basket may not produce the same recommendations which adds serendipity to the algorithm, letting the user experience a variety of products before finally checking out. Overall, the implementation into the checkout page seems neat and not overly intrusive giving users options to consider while displaying necessary information and an add button.

The content-based filtering system needed less information in the database since it was just the products and was easy to implement causing minimal issues, however the collaborative filtering system required many intricacies and adaptations to both the Reviews dataset and the number of entries used at once as well as the lack of a full user profile – dataset pair rendered a lot of collaborative techniques difficult to implement, therefore either finding different metrics for the user profile (i.e. clicks/ views etc) with a larger quantity would have been ideal but difficult considering the circumstance of just a University project (I'm not going to pay people to interact with my project which isn't going to give me any finance back).

6.3.8 Evaluation methods

Evaluation methods were not implemented in this iteration of the project resulting in a lack of confirmation on whether the recommender algorithms were effective. The effectiveness of a recommender system can be produced from multiple different metrics, and analysis of a single metric is not enough to determine whether the recommender system is optimal since we may want multiple different characteristics such as accuracy, coverage, serendipity, novelty and many more. Moreover, for some of these characteristics like serendipity, getting the right amount is also important because depending on the implementation method, for example with cosine similarity method for content-based filtering additional serendipity may directly impact the accuracy (which can be good since sometimes this algorithm is prone to overfitting and giving too perfect of a match), which most of the time results in the hybrid approach being adopted.

Chapter 7: **Professional Issues**

7.1 Artificial Intelligence

Artificial Intelligence, an ambiguous topic that has grown increasing popularity over the last decades, adapting rapidly and exceeding most visions through what it has already achieved and its potential, making this a difficult term to define. Professor John McCarthy (who coined this term in 1955) has described AI in his 2007 paper [https://www-formal.stanford.edu/jmc/whatisai.pdf] as: "the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable." A main focus on intelligent machines have shown prevalence in current years, the newest, ChatGPT a large language model chatbot using NLP to carry out most of its functionality which provides seemingly accurate answers to trillions of questions users have already asked it but doesn't end there with AI found in many factions that are eager to further make use of this new innovation.

Ecommerce has transformed over its existence and any edge in the market could produce vast profit margins which is largely sought for many companies in this industry. With artificial intelligence and machine learning techniques rising recently, to maintain at the forefront, a plethora of tools have piqued interest for both user side and behind the scenes: recommender systems, customer help chat bots, fraud detection, fraud prevention and many more. These tools can add personalisation and provide help to the user making the system more accessible and 24/7 - another important aspect of ecommerce. Being mostly beneficial for many topics, it brings a lot of freedom and innovation towards many of the things we interact with today improving general quality of life and easing the use of human computer interaction. Recommender systems have been directly implemented within my project, using machine learning algorithms to create personalised results to entice users to buy products that they usually buy or are similar to what they buy or items they usually don't bother finding but are statistically founded in a relation to actually be products they may like, all emphasising users to buy more, improving profit margins.

Even with recommender systems, many ethical issues arise making it questionable whether they should be allowed on the market. Given the example above, users' propensity to buy more is increased which could lead to compulsive shopping behaviours, with 5.8% of US general population being reported to have CBD (compulsive buying disorder) according to [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1805733/#:~:text=Compulsive%20buying%20disorder%20(CBD)%20is,in%20the%20US%20general%20population.], enticing the population to carry out more of these behaviours which can lead to many negative effects both economically in the general population and mentally as seen with CBD. The extent at how much personal information taken was kept to a minimum, i.e. I could have taken clicks, timestamps of viewing specific products, the IP at which the computer is located, all contributing to a much larger and more powerful user profile but might be argued that this is going over the boundaries of how much information is allowed to be stored on somebody and despite most websites displaying "accept cookies", "reject cookies" etc, most people especially those new or not experienced with technology do not fully understand the implications and just view it as another button, and therefore was not included within my project, the bare minimum amount of cookies that I stored on the user was their basket for guest users.

Help chat bots are being implemented in multiple facets including ecommerce providing answers to both frequently asked questions and more intricate questions rendering customer service providers to lose their jobs due to the bots' reliability, accuracy and cost making citizens fear for job displacement to occur at their workplace therefore recognising this job displacement and providing

methods to reallocate them is the job of everyone involved in the development of AI to give reassurance to the general public.

Chapter 8: **Bibliography**

Links used while covering section 3:

[1] – Amazon, "Amazon.co.uk: Low Prices in Electronics, Books, Sports Equipment & more," *Amazon.co.uk*, 2023. https://www.amazon.co.uk/

[2] – eBay, "Electronics, Cars, Fashion, Collectibles, Coupons and More | eBay," *eBay*, 2019. https://www.ebay.co.uk/

[3] – JD Sports, "JD Sports," *JD Sports*, 2015. https://www.jdsports.co.uk/

[4] – "The importance of a good eCommerce search bar," *Spacebetween.co.uk*, 2017. https://www.spacebetween.co.uk/blog/the-importance-of-a-good-ecommerce-search-bar

[5] – R. Van Meteren and M. Van Someren, "Using Content-Based Filtering for Recommendation 1." Available: https://users.ics.forth.gr/~potamias/mlnia/paper_6.pdf

[6] – Y. Adilaksa and A. Musdholifah, "Recommendation System for Elective Courses using Content-based Filtering and Weighted Cosine Similarity," *IEEE Xplore*, Dec. 01, 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9702788

[7] – "Math for Devs - Cosine Similarity in Python," *DEV Community*, Jun. 14, 2023. https://dev.to/josethz00/math-for-devs-cosine-similarity-in-python-2524 (accessed Nov. 21, 2023).

[8] – A. Elkhattam, "IMDb Movie Recommendation Chatbot," *Medium*, Dec. 07, 2020. https://a-elkhattam.medium.com/imdb-movie-recommendation-chatbot-942f84dfa0dc

[9] – "Python | Word Similarity using spaCy," *GeeksforGeeks*, Jul. 17, 2019. https://www.geeksforgeeks.org/python-word-similarity-using-spacy/

[10] – zeynep beyza ayman, "Recommendation Systems: Content-Based Filtering," *MLearning.ai*, Dec. 6, 2022. https://medium.com/@zbeyza/recommendation-systems-content-based-filtering-e19e3b0a309e (accessed Mar 23 2023).

[11] – S. Dash, "Decision Trees Explained — Entropy, Information Gain, Gini Index, CCP Pruning..," *Medium*, Nov. 02, 2022. https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c

[12] – C. Maklin, "Memory Based Collaborative Filtering — User Based," *Medium*, Aug. 10, 2022. https://medium.com/@corymaklin/memory-based-collaborative-filtering-user-based-42b2679c6fb5

[13] – Nguyen, Dung & Nguyen, Loc. (2010). "Model-based approach for Collaborative Filtering." Available: https://www.researchgate.net/profile/Loc-Nguyen-101/publication/321753015_Model-based_approach_for_Collaborative_Filtering/links/5a4f80894585151ee284dfb3/Model-based-approach-for-Collaborative-Filtering.pdf

[14] – F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261–273, Nov. 2015, doi: https://doi.org/10.1016/j.eij.2015.06.005

[15] – Madhusree Kuanr and P. Mohapatra, "Assessment Methods for Evaluation of Recommender Systems: A Survey," *Foundations of Computing and Decision Sciences*, vol. 46, no. 4, pp. 393–421, Dec. 2021, doi: https://doi.org/10.2478/fcds-2021-0023.

[16] – "accuracy module — Surprise 1 documentation," *surprise.readthedocs.io*. https://surprise.readthedocs.io/en/stable/accuracy.html

[17] – T. Silveira, M. Zhang, X. Lin, Y. Liu, and S. Ma, "How good your recommender system is? A survey on evaluations in recommendation," *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 5, pp. 813–831, Dec. 2017, doi: https://doi.org/10.1007/s13042-017-0762-9.

[18] – Julia Korsun (2023, September 5). "10 Popular Django Websites That You Probably Know" Available: https://djangostars.com/blog/10-popular-sites-made-on-django/

[19] – "customer reviews of amazon products," *data.world*. https://data.world/datafiniti/consumer-reviews-of-amazon-products/workspace/file?filename=Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv (accessed Nov. 18, 2023).

[20] - Michael Fuchs (2020, October 3). "Recommendation Systems - Plot Description-based Recommender" Available: https://michael-fuchs-python.netlify.app/2020/10/03/recommendation-systems-plot-description-based-recommender/

[21] - P. Resnick – N. Iacovou – M. Sushak – P. Bergstrom – J. Riedl (1994): GroupLens: An open architechure for collaborative filtering of netnews, In Proceedings of the ACM Conf. Computer Support Cooperative Work (CSC),pp. For section 2.1. Introduction

[24] - T. Cherickal, "Unveiling the Power of Django's MVT Architecture," *Medium*, May 27, 2023. https://thomascherickal.medium.com/unveiling-the-power-of-djangos-mvt-architecture-6554ff41af64.

[25] - Y. Tian and C. M. Stewart, "History of E-Commerce," Jan. 2011, doi: https://doi.org/10.4018/9781599049434.ch001.

[26] - "Django introduction," *MDN Web Docs*. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

[27] - hansmrtn, "beakr – A small web framework for R | R-bloggers," Oct. 30, 2020. https://www.r-bloggers.com/2020/10/beakr-a-small-web-framework-for-r/ (accessed Mar. 27, 2024).

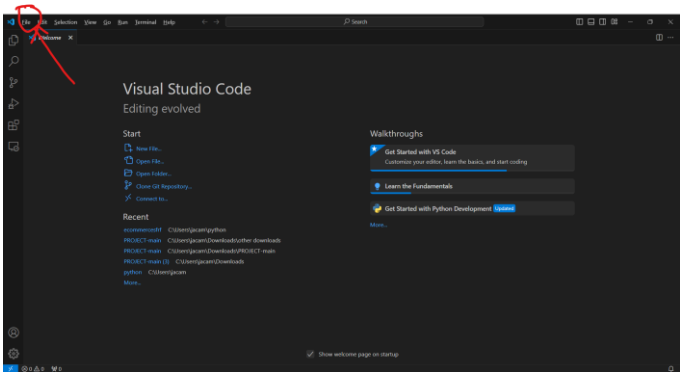
Chapter 9: **Appendix**

9.1 User Manual

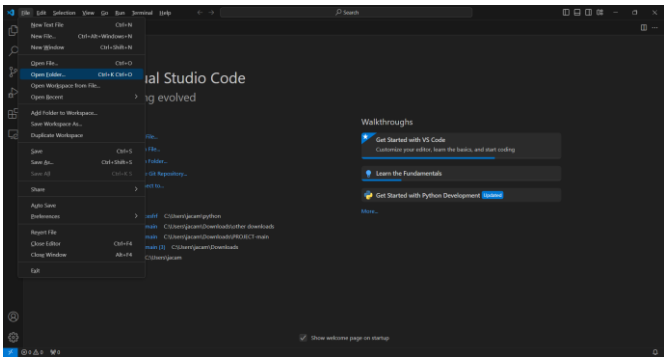
Firstly, download the zip file of the project from the main branch of the gitlab page.

Since this program was not deployed, an IDE is required to run this software which I have chosen as Microsoft Visual Studio Code, therefore must be downloaded from this website, depending on the operating system you are using will depend on which option you will download: https://code.visualstudio.com/download, you can also use guides only to help you since there is an abundance if you search Microsoft visual studios download + the OS you are working on.

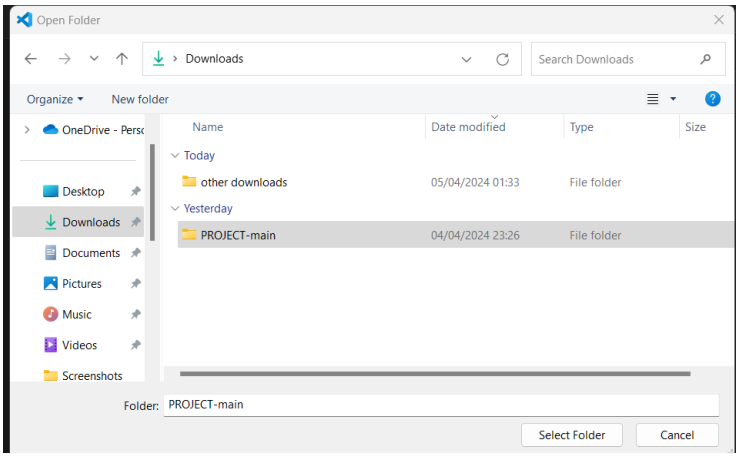
Once the IDE is setup, unzip the project. To load the project into a working environment, click on File, at the top left corner:



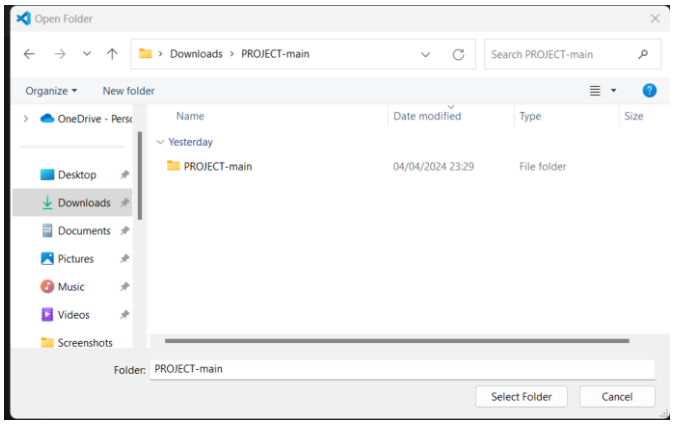
Then click on folder opener:



And locate the folder we need, which is the folder inside the unzipped main folder therefore, not this one:

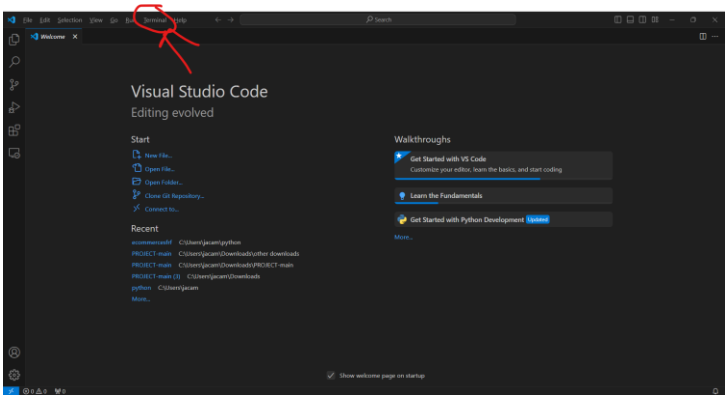


But clicking this folder giving the next folder deep:



And click select folder.

Then open a terminal (which if it isn't already open) by clicking at the top of the menu – 'terminal'



And to run the system, type in the terminal, `python manage.py runserver` will run the program on your browser at http://localhost:8000

9.2 Proof of concept Video

https://www.youtube.com/watch?v=b_hCGKmDmHA

9.3 Diary

Task	Date	Description	Complete
Create Project Template	October 14th	Create an empty initialiser including required packages and dependencies, aimed to get to an extent where subsequent contributions directly update the project. Also required downloading of Django and python systems and direct terminal manipulation to startapp (initialiser)	Yes
Create product display	October 21st	Create home page displaying products from the database, creating a product model	No, the views, url are implemented however the model with database of products is not
Create product display	October 22nd	Finish model with database of products (at least template fields) implementation	Yes
Create add button, adding product to cart	October 22nd	Manipulate button form to update cart and display on a separate page	Yes
Create order page	October 22nd	Transform the cart to present a final order/ payment page	No, displaying the products have been complete however models corresponding to user payment and shipping information has not been implemented

Create order page	October 24th	Implement payment and database input of order for both guest and users	Yes, however while testing with paypal api a few unsolved issues arose
Adjusting CSS	October 24th	Revamp the website to look more professional and cleaner	Yes
Login and logout functionality	November 17th	Add login and logout functionality connecting user to preexisting customer table	Yes
Add dropdown cart	November 18th	Produce a drop down of cart addition to the navbar cart button	Yes
Add search bar	November 18th	Produce a search bar and button that queries the database, displaying the results on a separate search page	Yes
Add category drop down	November 20th	Display distinct results of product's primary category field as separate links in a drop down, returning a page with products of respective category	Yes
Research content-based filtering techniques	November 21st	Researched content-based filtering techniques using cosine similarity and decision trees	Yes, however not formatted in report at this time
Research collaborative filtering techniques	November 23rd	Researched collaborative filtering approaches including user-user approach and matrix factorization	Yes, however not formatted in report at this time
Research evaluative techniques	November 23rd	Researched evaluative techniques for content-based and collaborative filtering resulting in methods for accuracy and coverage	Yes, however not formatted in report at this time
Bug fixing	November 30th	Fixed a bug where login was comparing username to email, a bug where categories would not show if the cart was empty, refactoring transaction id to become primary key for Order table	Yes
Dataset manipulation for Reviews	December 15th	Adding and truncating the [19] dataset to provide review information in preparation for the collaborative filtering system	Yes, however, transferring onto the

			database still needs to be done
Transferring dataset onto the database	December 16th	Ensuring wanted changes to the dataset are made whilst transferring it to the database	Yes
Content-based filtering system design	January 17th	Choosing where this filtering system should be implemented and how it should be implemented	Yes
Content-based filtering system implementation	February 1st	Back-end implementation of the content-based filtering system specifically focusing on text preprocessing and some of the algorithm	Yes
Content-based filtering system implementation	February 2nd	Back-end implementation of content-based filtering system finishing the cosine similarity algorithm and for authenticated users manipulates the back-end filtering system	No, guest users and authenticated users need to use this algorithm, however some of the core features are implemented
Content-based filtering system implementation	February 3rd	Front-end implementation of the content-based filtering system is implemented	Yes
Content-based filtering system implementation	February 4th	Back-end and front-end implementation of content-based filtering system for guest users building upon preexisting code for authenticated users	Yes
Unit testing for content-based filtering system	February 17th	Ensuring each component of the filtering system works correctly and as intended	Yes, however without full coverage
Collaborative filtering system implementation design	February 23rd	Choosing which filtering system to implement and where it should be implemented	Yes
Collaborative filtering system implementation	February 24th	Back-end implementation of collaborative filtering system, specifically using all review entries and with only authenticated users	Yes, but there are issues with the performance (takes 3 mins

			to reload the webpage due to the recommender algorithm)
Collaborative filtering system implementation	February 25th	Back-end implementation of collaborative filtering system, specifically refactoring the program so that samples were taken to ensure scalability and performance	Yes
Collaborative filtering system implementation	March 10th	Front-end implementation of collaborative filtering system, specifically for authenticated users, displaying recommended items correctly	Yes
Collaborative filtering system implementation	March 16th	Front-end and back-end additions for guest users so that their data can be manipulated to display recommendations too	Yes
Unit testing for collaborative filtering system	March 20th	Testing algorithm and predictions are carried out as expected with the get_top_n algorithm properly displaying the top 10 products of interest	Yes