Development of Tasker

Using a Recommender System to Provide the Best Services Relevant to a Given User

Daniel Azzopardi
ID: 0183600L
Department of ICT
University of Malta
daniel.azzopardi.18@um.edu.mt

Christina Barbara
ID: 0037900L
Department of ICT
University Of Malta
christina.barbara.18@um.edu.mt

Charlton Sammut
ID: 0490200L
Department of ICT
University Of Malta
charlton.sammut.18@um.edu.mt

ABSTRACT

Efficiently identifying and comparing reliable local services such as tile laying and pet sitting, which typically require the service provider to be physically present, without having to sift through numerous unrelated content, is not something that can easily be done using the platforms currently available. The purpose of this proof of concept was to better understand if it was possible to build such a platform without making use of an unnecessarily overcomplicated system while still being able to connect consumers to the service provider which best fits their needs.

The platform proposed for this proof of concept is an e-services website which allows users to either search for a service or advertise a service they are providing. Whilst variables such as minimal clicks were taken into consideration when designing this proof of concept the feature which most enhances user experience is the recommender system. The website in question gives both personalized and non-personalized recommendations. This is done by making use of various data points and weightings, such as popularity and distance, that are processed by the recommender system which in turn shows the user services and categories it believes might be of interest.

KEYWORDS

Recommender system, E-services, Website

INTRODUCTION

Nowadays, most businesses are turning to online advertising to survive in such a highly competitive business landscape. This allows them to market and advertise their services to a vast audience while maintaining low operating costs [1]. On the client side, most internet users are increasingly turning to online platforms in an attempt to find services which best suit their needs due to the substantial amount of information present online. Although there are platforms, such as Facebook, on which services are advertised it is very hard to efficiently identify and compare reliable services without having to sift through numerous unrelated content. This is especially true for local services which require the service provider to be physically present for the job such as tile laying, electrical work, pet sitting and so on.

Aim & Objectives

The artefact described in this paper was put together as a proof of concept to see how feasible it is to build a platform on which clients could find service providers which best suit their needs in an efficient and optimal manner. The idea behind Tasker is to have a platform through which a user can find services, or alternatively, offer their own services in order to attract clients. The main AI component, used to provide a more optimized experience, is a recommender system which recommends services to the user based on variables such as the user’s location, previous searches, user ratings and so on. Connecting the client to the most suitable service provider aims to deliver a higher rate of customer satisfaction after the service is complete.

The following sections describe the steps taken to come up with the proof of concept in question. A literature review was first done to better understand current technologies and solutions pertaining to the solution being put forward. The methodology and evaluation of the artefact are then detailed and the resulting limitations and possible future developments are discussed.

LITERATURE REVIEW

The choice overload hypothesis states that as the amount of possible items available to a given user starts increasing, their motivation to make a choice and satisfaction with their

final chosen item starts to decrease [3]. As the amount of available items available on a platform starts to increase this becomes an issue that developers have to handle and a recommender system quickly becomes a vital and much needed application for the platform. A Recommender System is a program that provides suggestions for items to a given user. “Item” is the general term used to denote what a given recommender system recommends to its users [4]. These suggestions usually relate to various decision-making situations [4], such as what videos to view or what items to purchase, but in this particular case the situation can be presented as “What services would the user be mostly likely to make use of?”
These suggestions provided might not always be correct, but they have the greatest potentiality / probability to be liked by the user [4], [5]. Modern recommender systems use different methods to extract patterns from the user [5]. Modern recommender systems give two types of recommendations, personalized and non-personalized recommendations. Personalized recommendations give recommendations based on the current user’s preferences, these preferences are normally brought from their past searched / used / bought items. On the other hand non - personalized recommendations give recommendations based on certain heuristics such as the number of times a given item has been visited by other users, or if a current item is trending [5].

As stated in [5] which is a literature review on recommender systems, many domains that make use of recommender systems gain significant improvements due to recommender systems, however the currently available recommender systems are far from the ideal model, and further research is needed to improve accuracy.

ANALYSIS AND METHODOLOGY

This section describes the various functionalities and steps taken when building the website.

Front-End Development

With regards to the front-end of the website, HTML and CSS were used. Where certain components had to be coded from first principles, Bootstrap 4 was used to make front-end development faster and easier. This allowed basic features such as buttons and cards to be added without consuming as much time as would have been required to code each component from scratch. Most of these components however still had to be altered to fit the overall design of the website. Components requiring some form of animation such as navigation bars, buttons, toggles and collapsible elements also make use of JavaScript. Any CSS and JavaScript files used by Bootstrap [2] were referenced using links. Although this increases the risk of the link breaking it reduces the amount of bandwidth required by the server to transmit the pages to the client. Should this proof of concept be taken further an if-statement could be used to fall back on local copies should the links generate an error, thus ensuring that the Bootstrap elements never break should the file source no longer become available.

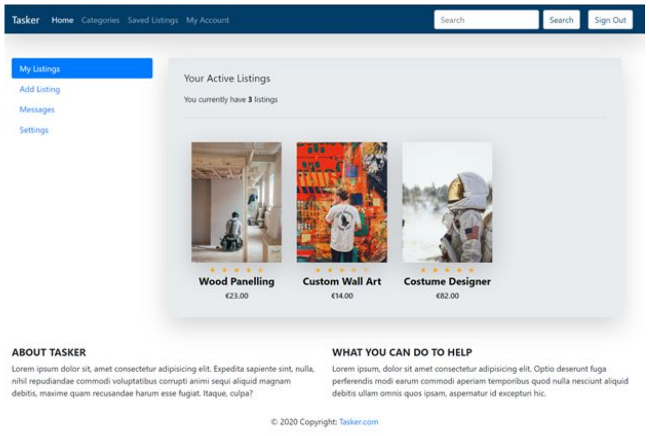
Since such a website relies heavily on images to market the different services being offered the following code was added to ensure missing images did not break the website:

onerror="this.src='images/nia.png';"

This code is placed inside the <img> tag and states that; should the image which the website is trying to display not be found an image not found PNG file (‘*nia.png*’) is displayed instead.

While a templating engine would make some aspects of the code cleaner and easier to alter, it was decided that when taking into consideration the small number of pages used by the website, adding a templating engine would simply add unnecessary overhead in execution time and memory to build the pages, without the benefits that would come with using it when building vast websites.

The design of the website aims to minimize as much as possible the amount of clicks the user needs to achieve their goal. It was also ensured that the proof of concept could be viewed using both desktop and mobile devices. Sample screens can be seen in figures 1 and 2 below respectively.



Your Active Listings

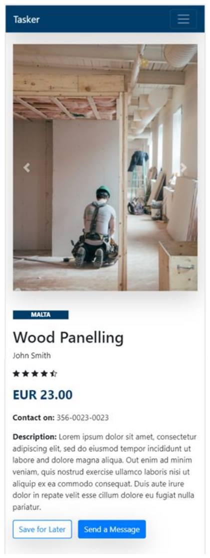
You currently have **3** listings

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Expedita sapiente sint, nulla, nihil repudiandae commodi voluptatibus corrupti animi sequi aliquid magnam debitis, maxime quam recusandae harum esse fugiat. Itaque, culpa?

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Optio deserunt fuga perferendis modi earum commodi aperiam temporibus quod nulla nesciunt aliquid debitis ullam omnis quos ipsam, aspernatur id excepturi hic.

© 2020 Copyright: Tasker.com

Figure 1: Screenshot of the My Account page, My Listings section on a desktop device



Wood Panelling

John Smith

EUR 23.00

Contact on: 356-0023-0023

Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore and dolore magna aliqua. Out enim ad minim veniam, quis nostrud exercise ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in repate velit esse cillum dolore eu fugiat nulla pariatur.

Figure 2: Screenshot of top part of the Product Page on a mobile device

Tasker's Database

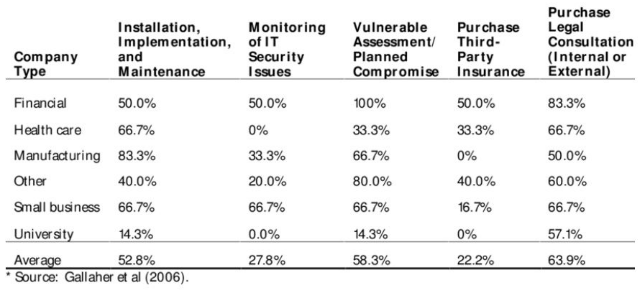
The created HTML was converted into PHP, to allow dynamic updating of the website through its connection to the database. This is aided by the implemented recommender system, which will choose which information from the database is to be retrieved and displayed to a signed-in user.

Tasker's database, declared as taskerdb, consists of 3 tables, used to store existing users, services, and available categories.

The user table contains basic user information which is requested upon registration, including the user's name, surname, date of birth, email, and password. To ensure the user's security, the stored password is hashed prior to being entered into the database. Along with this information, the table also contains fields to cater for future additions to the user's personal details, providing spaces for a user biography, phone numbers, and pictures. In an improved version of Tasker, the user would be able to populate this information, and view it on a personal user page.

User passwords are hashed through the PHP function password_hash(), one of its parameters being PASSWORD_DEFAULT [17]. This parameter presently makes use of the bcrypt algorithm [18], which with time, may be replaced as stronger hashing algorithms are incorporated into PHP .

An improvement on storing passwords, although the current method presently suffices, would be to outsource passwords rather than store them in a database. As shown below, a number of companies already perform outsourcing for numerous reasons, one of which being the monitoring of IT security issues [19].



Company Type	Installation, Implementation, and Maintenance	Monitoring of IT Security Issues	Vulnerable Assessment/ Planned Compromise	Purchase Third-Party Insurance	Purchase Legal Consultation (Internal or External)
Financial	50.0%	50.0%	100%	50.0%	83.3%
Health care	66.7%	0%	33.3%	33.3%	66.7%
Manufacturing	83.3%	33.3%	66.7%	0%	50.0%
Other	40.0%	20.0%	80.0%	40.0%	60.0%
Small business	66.7%	66.7%	66.7%	16.7%	66.7%
University	14.3%	0.0%	14.3%	0%	57.1%
Average	52.8%	27.8%	58.3%	22.2%	63.9%

* Source: Gallaher et al (2006).

Figure 3: A summary showing the percentage of companies from different industries that perform outsourcing, source [19]

Services are stored in the database provided that fields containing service provider information, such as phone numbers and email, and fields defining the service through description, service title, price, location, and the category to which the service belongs to, are filled. A number of fields will be defaulted to a value depending on the data type of said fields, as these will be updated as users interact with the service's display page. Such fields include the longitude and latitude of the service based on its location, the number of people belonging to a certain age group who view the service, and the service's rating. The rating is set to 0 by default upon service creation, and will accordingly display that the service has not yet received ratings. This value would be altered by giving a signed-in user the option to review services, thus altering said service's rating.

The created category table contains an image with which the category will be displayed, alongside its name and the number of services it contains. The number of services is accordingly updated when a user creates or deletes a listing.

The tables used to store user and service information respectively contain the fields 'likes' and 'popularity'.These are populated while a signed-in user is making use of the website, and their values will be used by the recommender system to show the user services and categories it believes to be of interest to the user. 'likes' indicates which categories the user is interested in through a dictionary of the form {"Category" : x, "CategoryN" : y}, where 'Category' and 'CategoryN' represent possible categories, and 'x' and 'y' represent the number of times a user has viewed those categories. 'popularity' stores a decimal value signifying how attractive a service is to users, regardless of whether they are signed in. If a user simply views a service's page, 'popularity' will be incremented by 0.01, and if the user opts to send a message to the service provider, it will be incremented by 0.05. The latter occurs provided that the user is signed in.

The service table also contains a number of columns, used to identify how many people have viewed the service based on the age group they belong to. The values of these columns, as well as the service's location - narrowed down to a city with longitude and latitude where applicable - aid the recommender system in determining the most appropriate recommendation.

Existing Input Forms and Entry Validation

The current version of Tasker has 3 forms in which a user can enter data, these being register and sign-in forms, and a form used to create a new listing. Each field of all forms is greatly validated to ensure that the user does not enter information which does not follow a specific format.

When filling in the registration form, a user is instructed to create their password with lowercase letters, uppercase letters and numbers, having a total length of at least 8 characters. If this, or any other requirement is not fulfilled by the user, they are given an appropriate error message and are told to re-enter their details. This would be improved by preventing the page from refreshing, so data entered by the user is not erased, and they would not have to re-enter all their details.

By making use of mySQLi PHP functions, an update on their mySQL counterpart, the database is better protected against possible SQL injections.

It was initially intended to allow a user to be able to add, edit, and delete their listings. Presently, only adding and deleting listings are implemented, these altering the service table as necessary, while also updating the number of services found in the category table. As editing a listing would be quite similar to adding a listing in its implementation, it was decided that the development of other features would be given more priority over its implementation.

Development Of The Recommender System

Tasker's recommender system uses both the user's preferences and data, along with data collected from other users to calculate which recommendations it gives. The recommendation system works by making use of different data points, and giving different weighting to the data points that are deemed more important. Then using a metric that will be later explained, calculates some probability for each service. After which an 'n' number of services are chosen (without replacement) using a modified roulette wheel selection algorithm, where the higher the probability of a given service, the higher the chance of it being chosen.

The following data is used by the recommender system to calculate the probability of a given service:

- Rating of the service, the higher this number the more likely it is to be recommended over other services.
- How popular the service is, the higher this number the more likely it is to be recommended over other services.
- How many similar services the user has viewed / purchased.
- How many similar users have purchased the service.
- Physical geographical distance from the user to the service, the smaller this number the more likely it is to be recommended over other services.

These data points were chosen with reference to [6]. The data points are stored in the database and when changes are made they are updated accordingly. Firstly the rating of a given service is calculated by the average of all the ratings, and it is updated with each new rating.

Secondly, the popularity value is calculated by:

- Adding 0.05 for each new unique user who views the service.
- Adding 0.01 for each non-unique user who views the service.
- Adding 0.5 for each user who purchases the service.

Different values were considered and tested, and it was ultimately decided with reference to [6], [7] and [8], that these values best fit Tasker's use. As stated by [6], [7] and [8] some recommendation systems tend to suffer from a popularity bias. A popularity bias happens when a recommender system starts recommending popular items much more frequently when compared to less popular, niche items. In some cases where a recommendation system suffers from a strong popular bias it would recommend the less popular items rarely or not at all, which would cause these items to die out and receive no traffic. In this particular implementation the problem is solved in two ways, the first one being that the popularity value is not the only variable that decides if a service is to be recommended or not, and secondly is the fact that even services with a low probability of being chosen still have the chance of being chosen. This allows the user to be recommended to new services that they might have never heard of but still find interesting, and if they don't find it to their liking, the probability of that item being recommended again decreases even further. As stated in [9], another measure that could be taken is to add an exponential decay value. This would make it so that old popular items don't stay popular unless they are receiving views / purchases. This was not added to Tasker as since Tasker is a proof of concept, it does not have any users and all services would eventually decay to the same popularity.

The third value that is considered by the recommendation system is to check how many similar services the current user has viewed. Since each service belongs to a category, it can be assumed that each service in a given category is related, even if by a small factor. So for each user a dictionary was created inside of the database, where the keys of this dictionary are equal to the names of the categories, and the value represents how much the user likes that given category. This value is calculated by:

- Adding 1 for each category visited.
- Adding +10 for each purchase made in that category.

These values were decided by both testing the recommendation system and by making reference to [10] and [11].

The fourth value that is considered by the recommendation system is to check how many similar users have viewed / made use of a given service. This similarity value can be based on many factors, as shown in [12], [13]. But since Tasker has no users it would be difficult to find / generate dummy data, so for the sake of this proof of concept, similarity between two users was only calculated using similar age groups.

The fifth value that is considered by the recommendation system is to check the physical distance between a user and a service. This is done by asking the service provider to add an address, the latitude and longitude values of this address are then calculated and stored on the database. When the user makes use of the website, their location is obtained, and the distance between them and the service is calculated using the haversine formula. The smaller this distance, the greater the probability of a given service to be chosen.

Each of these data points were given different weighting depending on their importance. The following weights were chosen after rigorous testing and by referring to [6]:

- Rating = 20%
- Popularity = 10%
- Likeness = 55%
- Age = 5%
- Distance = 10%

It should also be noted that should a service provider choose to purchase a premium listing, the probability of it being recommended is more than doubled. This allows for a service provider to easily advertise their product, especially when compared to other expensive and more complicated solutions.

Implementation Of The Recommender System

To implement the recommender system python3 was used, the documentation of python3 can be found here: [14]. All of the python code was stored in a python file called rSystems.py. This file connects to the website by making use of the php function shell_exec [15]. The following pseudo code was followed while developing rSystem.py

```
rSystem(user, services, n, premiumOnly){
  weights = [0.2, 0.1, 0.55, 0.05, 0.1]
  topCategories = Top 4 categories with the highest
                  likeness score
  returnValue = {}
  //where likeness --> +1 to each category visited and +10
    for each message sent  in that category
  for service in services{
    if(premiumOnly and service.isPremium){
      continue
    }

    if (user.city == service.city or service.city == “N.A”) and
      (service.rating >= 3 or service.isPremium) and
      (service.category in topCategories){
        p = 0
        p += (service.rating * weights[0]) +
             (service.popularity * weights[1]) +
             (user.likeness[service.category] *weights[2])

        //where popularity -->+0.01 to each visitor,
          +0.5 to each new message sent
         p += service.age[user.age] * weights[3]
```

```
            //The more people in the same age group as the
              user, the more it is likely to be recommended
            if service.location != "N.A" and user.location !=
            "N.A"{
                distance = getDistanceBetween(user.location,
                                service.location)
                p = (p * weights[4]) / distance
            }
            if service.isPremium{
                p *= 2.5
            }
             returnValue[service.id] = p
        }
    }
    return random.choose(returnValue.keys(),
                            distribution=returnValue.items(),
                            numberOfItemsToReturn = n)
}
```

The biggest hurdle of implementing the above function was to parse the large amount of data it needs. Eventually the below data structures where chosen:
user: List of user's information where:
userInfo[0] = age (int)
userInfo[1] = country (string)
userInfo[2] = location (string)
userInfo[3] = likes, represented as a dictionary with categories as keys (strings), and likeness as value (ints)
services: A list of all the services, where a service is a list where:
service[0] = category (string)
service[1] = country (string)
service[2] = rating (double)
service[3] = A list containing integers, each position in the list corresponds to an age group
service[4] = isPremium (bool)
service[5] = popularity (double)
service[6] = longitude (double)
service[7] = latitude (double)
service[8] = id (int)
n: Number of services to be returned (int)
hasToBePremium: If true only returns premium services (bool)

Another difficulty that was encountered was to find the user's current location. This was solved by making use of the geopy library.[16] To make use of geopy one must register and get a key. While geopy is free it has limitations on how many requests a given key can send per minute. Since Tasker is just a proof of concept this will do, but however an 'if' statement had to be added in case the Tasker key runs out of requests. If this 'if' statement is entered, the recommendation system will temporarily not work. If Tasker was to be implemented fully, a different tool would have been used to ensure that the recommendation system is constantly working.

TESTING AND EVALUATION

This section describes the testing and evaluation done on the system.

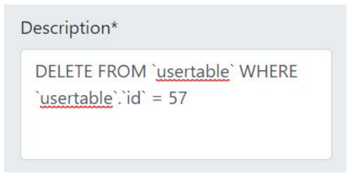
Front-End Testing

UI testing for front-end was done by going through the website and checking that all the front-end components were working as expected. The website front-end was tested on both desktop and mobile devices. Testing on desktop was done using Google Chrome and Microsoft Edge to ensure that the interface works on more than one platform. Due to hardware limitations testing could not be done on Safari. Although programs such as Cypress could have been used to automate testing, due to the small number of pages used by the website testing was done manually. For most of the front-end testing, particularly testing which required server side interaction to manipulate front-end elements, an end-to-end testing approach was taken.

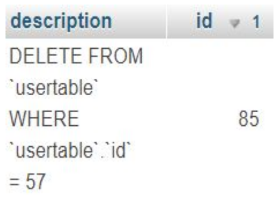
Back-End Testing

To ensure that user registration and sign-in, and the creation and deletion of a listing work as intended, it was first tested that a user can successfully perform said operations through the provided HTML forms. Values which are not to be accepted by the forms were also entered, to ensure that the operation, as required, will stop the user from performing the operation.

To check against SQL injections, PHP statements were entered in the forms to confirm that no harm would be caused to the database. An example of such a test is as follows, where a statement to delete a user is entered in the form used to create a service. As seen, this is simply changed into a regular string and stored as such in the database.


Description*

DELETE FROM `usertable` WHERE `usertable`.`id` = 57



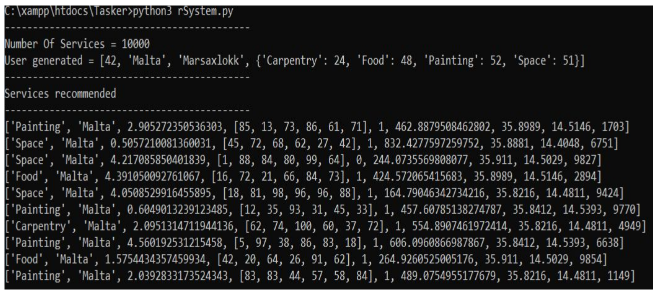
description	id 1
DELETE FROM `usertable` WHERE `usertable`.`id` = 57	85

Figures 4 and 5: Testing for an SQL injection

Testing the recommendation system

Testing the recommendation system came with it's set of difficulties, as to test a recommendation system one requires data, but since Tasker has no active users it does not contain any data. One solution that was attempted was to find data sets from online sources, however no data set with the correct format could be found, and the ones that were somewhat suitable would have taken too long to convert into a way that rSystem.py could read.

Ultimately to solve this problem a python script was written to generate dummy data. This script creates 10,000 different services, and a user with random data. It first outputs the generated user, followed by the services it recommends for this user.



```
C:\xampp\htdocs\Tasker>python3 rSystem.py
--------------------------------------------
Number Of Services = 10000
User generated = [42, 'Malta', 'Marsaxlokk', {'Carpentry': 24, 'Food': 48, 'Painting': 52, 'Space': 51}]
--------------------------------------------
Services recommended
--------------------------------------------
['Painting', 'Malta', 2.905272350536303, [85, 13, 73, 86, 61, 71], 1, 462.8879508462802, 35.8989, 14.5146, 1703]
['Space', 'Malta', 0.5057210081360031, [45, 72, 68, 62, 27, 42], 1, 832.4277597259752, 35.8881, 14.4048, 6751]
['Space', 'Malta', 4.217085850401839, [1, 88, 84, 80, 99, 64], 0, 244.0735569808077, 35.911, 14.5029, 9827]
['Food', 'Malta', 4.391050092761067, [16, 72, 21, 66, 84, 73], 1, 424.572065415683, 35.8989, 14.5146, 2894]
['Space', 'Malta', 4.0508529916455895, [18, 81, 98, 96, 96, 88], 1, 164.79046342734216, 35.8216, 14.4811, 9424]
['Painting', 'Malta', 0.6049013239123485, [12, 35, 93, 31, 45, 33], 1, 457.60785138274787, 35.8412, 14.5393, 9770]
['Carpentry', 'Malta', 2.0951314711944136, [62, 74, 100, 60, 37, 72], 1, 554.8907461972414, 35.8216, 14.4811, 4949]
['Painting', 'Malta', 4.560192531215458, [5, 97, 38, 86, 83, 18], 1, 606.0960866987867, 35.8412, 14.5393, 6638]
['Food', 'Malta', 1.5754434357459934, [42, 20, 64, 26, 91, 62], 1, 264.9260525005176, 35.911, 14.5029, 9854]
['Painting', 'Malta', 2.0392833173524343, [83, 83, 44, 57, 58, 84], 1, 489.0754955177679, 35.8216, 14.4811, 1149]
```

From the above example one can see how all of the services recommended are from categories that the user has a strong liking to. Nine out of ten of the services are also premium (shown by the 5th element), this is due from the strong boost premium recommendations get. Most of the services also have a relatively high rating (shown by the 2nd element). All of the recommended services all have a high amount of users around the age of 40.

For one to run this test, they simply need to:

- Have python installed on their machine, alone with the geopy library.
- Go to the directory where rSystem.py is located
- Run the following command:
  python3 rSystem.py

LIMITATIONS AND IMPROVEMENTS

There are various areas in which this artefact could be improved further beyond being a basic proof of concept. With regards to listings, allowing use of multiple images per listing, having a fully functioning section for saved listings and allowing sorting and filtering of results could enable the recommender system to be fine-tuned and improved further by taking into consideration variables pertaining to what users, after being assigned a set of recommended services, typically filter out.

With regards to further improvements to ease of use, allowing messaging functionality inside the website and having user profiles would allow users to view other services being offered by a particular user and easing interaction. This also opens up areas related to how long users typically interact with a service provider before deciding to buy or not buy their service. Other improvements include allowing sign in and registration to the website using popular options such as Facebook and Google, as well as implementing the 'forgot password' feature and mobile verification.

Most of the future improvements discussed in this section, such as a filter/sorting panel and saved listings page, already have front-end support provided by the artefact produced for this proof of concept.

One of the bigger limiting factors faced by testing such a system is the lack of data that would be generated should a larger user base be navigating the platform and creating legitimate listings. This lack of data also causes the recommender system to return sub-optimal results than would have been returned with using a larger data set.

CONCLUSION

From this proof of concept it was highlighted how important recommendation systems are, as a platform can quickly die out without one. They are constantly being used, and they subconsciously make decisions for a user. They choose what music a user listens to what shows they watch, so it is very important that they work as intended as they can greatly affect one's day.

REFERENCES

[1] S. Gilaninia, M. Taleghani and H. Karimi, Longdom.org, 2013. [Online]. Available: https://www.longdom.org/open-access/internet-advertising-and-consumer-behavior-in-the-purchase-of-products.pdf. [Accessed: 29- Jun- 2020].

[2] "Documentation", Getbootstrap.com. [Online]. Available: https://getbootstrap.com/docs/4.5/getting-started/introduction/. [Accessed: 29- Jun- 2020].

[3] B. Scheibehenne, R. Greifeneder and P. Todd, "Can There Ever Be Too Many Options? A Meta-Analytic Review of Choice Overload", Journal of Consumer Research, vol. 37, no. 3, pp. 409-425, 2010 [Online]. Available: https://www.researchgate.net/publication/48210291_Can_There_Ever_be_Too_Many_Options_A_Meta-analytic_Review_of_Choice_Overload

[4] F. Ricci, L. Rokach and B. Shapira, *Recommender Systems Handbook*. 2010, pp. 1-29.: https://www.researchgate.net/publication/48210291_Can_There_Ever_be_Too_Many_Options_A_Meta-analytic_Review_of_Choice_Overload

[5] K. Madadipouya and S. Chelliah, "A Literature Review on Recommender Systems Algorithms, Techniques and Evaluations", *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, vol. 9, no. 2, 2020 [Online]. Available: https://www.edusoft.ro/brain/index.php/brain/article/view/693. https://www.researchgate.net/publication/48210291_Can_There_Ever_be_Too_Many_Options_A_Meta-analytic_Review_of_Choice_Overload

[6] D. Jannach, L. Lerche, F. Gedikli and G. Bonnin, *What Recommenders Recommend – An Analysis of Accuracy, Popularity, and Sales Diversity Effects*. Springer, Berlin, Heidelberg, 2013. https://www.researchgate.net/publication/48210291_Can_There_Ever_be_Too_Many_Options_A_Meta-analytic_Review_of_Choice_Overload

[7] K. Nagatani and M. Sato, "Accurate and Diverse Recommendation based on Users' Tendencies toward Temporal Item Popularity", 2017 [Online]. Available: http://ceur-ws.org/Vol-1922/paper7.pdf.

[8] H. Abdollahpouri, R. Burke and B. Mobasher, "Managing Popularity Bias in Recommender Systems with Personalized Re-ranking", 2019 [Online]. Available: https://arxiv.org/pdf/1901.07555.pdf.

[9] Y. Ding and X. Li, "Time Weight Collaborative Filtering", 2005 [Online]. Available: https://cseweb.ucsd.edu/classes/fa17/cse291-b/reading/p485-ding.pdf. [Accessed: 30- Jun- 2020]

[10] U. Leimstoll and H. Stormer, "Collaborative Recommender Systems for Online Shops", 2007 [Online]. Available: https://www.researchgate.net/publication/220893443_Collaborative_Recommender_Systems_for_Online_Shops

[11] M. Khalaji, K. Mansouri and S. Mirabedini, "Improving Recommender Systems in E-Commerce Using Similar Goods", *Journal of Software Engineering and Applications*, vol. 05, no. 02, pp. 96-101, 2012 [Online]. Available: https://www.researchgate.net/publication/270538521_Improving_Recommender_Systems_in_E-Commerce_Using_Similar_Goods

[12] M. Nilashi, K. Bagherifard, O. Ibrahim, H. Alizadeh, A. Lasisi and N. Roozegar, "Collaborative Filtering Recommender Systems", *Research Journal of Applied Sciences, Engineering and Technology*, vol. 5, no. 16, 2013 [Online]. Available: https://www.researchgate.net/publication/287952023_Collaborative_Filtering_Recommender_Systems.

[13] A. Celdrán, M. Pérez, F. García Clemente and G. Pérez, "Design of a recommender system based on users' behavior and collaborative location and tracking", *Journal of Computational Science*, vol. 12, pp. 83-94, 2016 [Online]. Available: https://www.researchgate.net/publication/287148849_Design_of_a_Recommender_System_Based_on_Users'_Behavior_and_Collaborative_Location_and_Tracking

[14] 3.8.3 Documentation", *Docs.python.org*, 2020. [Online]. Available: https://docs.python.org/3/. ,

[15] "PHP: shell_exec - Manual", *Php.net*, 2020. [Online]. Available: https://www.php.net/manual/en/function.shell-exec.php,

[16] "Welcome to GeoPy's documentation! — GeoPy 2.0.0 documentation", Geopy.readthedocs.io, 2020. [Online]. Available: https://geopy.readthedocs.io/en/stable/#.,

[17] "PHP: password_hash - Manual", Php.net. [Online]. Available: https://www.php.net/manual/en/function.password-hash.php. [Accessed: 30- Jun- 2020].

[18] N. Provos and D. Mazieres, "Bcrypt Algorithm", Usenix.org, 1999. [Online]. Available: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node5.html. [Accessed: 30- Jun- 2020].

[19] B. Rowe, "Will Outsourcing IT Security Lead to a Higher Social Level of Security?", in Workshop on the Economics of Information Security (WEIS), North Carolina, 2007. [Accessed: 30- Jun- 2020].