DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

PERSONAL NEWS RECOMMENDATION SYSTEM

by

Sezgin SEVEN

November, 2019

İZMİR

PERSONAL NEWS RECOMMENDATION SYSTEM

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Computer Engineering

by

Sezgin SEVEN

November, 2019
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"PERSONAL NEWS RECOMMENDATION SYSTEM"** completed by **SEZGİN SEVEN** under supervision of **ASSOC. PROF. DR. ADİL ALPKOÇAK** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Adil Alpkoçak

Supervisor

Assoc. Prof. Dr. Tuğkan Tuğlular

(Jury Member)

Assoc. Dr. Şeniz Demir Uteli

(Jury Member)

Prof.Dr. Kadriye ERTEKİN

Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

OKURSS Project, which was developed simultaneously with this study, was supported by Tubitak 1512 entrepreneur programme with the project number of 2150289, and was developed at Dokuz Eylül University's Technopark DEPARK's Beta Building by LemonSOFT Ltd. Şti. . I want to express my gratitudes to Assoc.Prof. Dr. Adil Alpkoçak, who has been the leader of OKURSS project, and Dr. Okan Öztürkmenoğlu, who has helped me tremendously and for all kind of support thanks to my brother Oğuzhan KAYIŞ.

Sezgin SEVEN

PERSONAL NEWS RECOMMENDATION SYSTEM

ABSTRACT

News recommendation systems are decision support mechanisms that make it easier for people to make choices. In this thesis, we designed and implemented a personal news recommendation system using recommendation system approaches. In addition to content-based filtering and collaborative filtering approaches, we implemented a keyword-based recommendation system. We created users who read daily news in the categories we set for test data. We received feedback from users created for test data by recommended news relevant to their interests and we showed these results.

Keywords : Recommendation system, Personal news recommendation systems, text mining

KİŞİSEL HABER ÖNERİ SİSTEMİ

ÖZ

Haber öneri sistemleri insanların seçim yapmalarını kolaylaştıracak karar destek mekanizmalardır. Bu tez çalışmasında öneri sistemi yaklaşımlarını kullanarak kişisel haber öneri sistemi tasarlayıp uyguladık. Öneri sistemleri yaklaşımlarından içerik tabanlı filtreleme, işbirlikçi filtreleme ve anahtar kelimeye dayalı öneri yöntemlerinin uygulamalarını gerçekleştirdik. Test verisi için belirlediğimiz kategorilerde günlük haberler okuyan kullanıcılar oluşturduk. Bu kullanıcıların ilgilerine uygun haberler önererek test verisi için oluşturduğumuz kullanıcılardan geri beslemeleri aldık ve bu sonuçları gösterdik.

Anahtar kelimeler : Öneri sistemleri, kişisel haber öneri sistemleri, metin madenciliği

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

1.1 Background Information

Today, the most important source of information is undoubtedly the Internet. Because there is a lot of information and resources in the Internet, people find it difficult to access the accurate information. That we have too many resources about information does not necessarily mean that it is easier to access it, nevertheless it can be an extra effort to find reliable information among too many resources. Most users may not want to make this effort and maybe they can get information from only a source. Instead, that there are multiple resources about the information or news which users want to read facilitate an ease of reading preference. In this way people whose opinions are different can see and elaborate what other people write about a topic, and this situation will provide an easily commendable platform. This method provides both easy access to information and can transfer information to the user. Hereby, recommendation systems will be activated. Recommendation system is a topic, which facilitate ease of access to information. Nowadays, recommendation systems, which are applied numerous fields, uses different methods. Basically two of these methods are quite common: Content-based filtering and collaborative filtering approaches.

Recommendation system is defined in a study is systems that recommend appropriate elements according to the user's personal preferences and characteristics, without any effort from the user (Özgöbek & Erdur, 2015). At the same time, while the user does not make any effort to find news sources, the user may want to read the news in a readable format. Most of the news and information resources on the site while users read an article, appearing around the article or the user's advertising and similar objects that disrupt the reading environment. Providing a clean readable environment together with the recommendation system has been an element that will support the recommendation system we have developed. The need for these applications is increasing day by day and people are showing great interest. Reliable news continues to be very difficult to reach.

OKURSS is a social media application which helps information, being a modern day person's need, to be accessible, readable in a suitable environment, and helps users to communicate with other users. Its purpose is to enable people to access information and to support the dissemination of information through readings. Recognizes the readers by following and analysing the reading habits of OKURSS users. In this way, it has a smart structure that recommends the most suitable articles to its users.

1.2 Problem Definition

Information, thanks to the Internet, is not only about libraries or books and papers, it spreads all over the globe, easily and fast. At the same time, it increases constantly, day by day. That information can be so easily produced and spread is causing us to face a massive information stack. To access information can be possible by writing some keywords to any search engine via a device that has an access to Internet. Instead of people trying to access information, information accessing to people will make information more accessible.

A high number of readers take a lot of time to find articles that are appealing to them. At this exact situation, recommendation systems do this job for readers. The purpose of recommendation systems is to help readers access information that can be appealing to them or can be preferred by them, without any effort. Recommendation systems are highly used in our modern times from shopping to applications that contain photographs and videos. Apart from these, news sites, with the purpose of recommendation news, can be seen using such systems. However, there are few systems which make news recommendations for the user. In this thesis, we aimed for the news is to be recommended for people’s choosing, and people to read this news. Taşçı (2015) says that the more the recommended news are read, the more successful the recommendation system is.

1.3 Aim of The Thesis

The aim of this thesis is to design, develop and implement a news recommendation system. We used the well-known recommendation systems approaches, to obtain news list results with hybrid approach and to compare the results of different methods. We also aim to provide a solution to the challenges encountered in recommendation systems area.

1.4 Thesis Outline

This thesis consists of six main parts. In the second section, we present the importance of the news recommendation system and its benefits to people. In the third section of the thesis, similar studies and what kind of challenges these studies face as well as how they are solved. In the fourth section, we argued recommendation systems and methods of recommendation systems which content-based recommendation systems, collaborative filtering and hybrid-method approach and how they are used in our project. We described these methods in detail and how they are evaluated and applied in the project. Besides, we also explained user's profile creation and how we solved cold start problem with user profiles and keyword-based recommendation. In the fourth section, we gave technical details of OKURSS application and how recommendation system algorithm works in OKURSS. Section five represents the experiments we conducted to evaluate the recommendation system. In the last section of thesis, we presented the results we obtained and give a look at the future studies in this domain.

CHAPTER TWO
LITERATURE REVIEW

In this section, we surveyed the news recommendation systems and similar studies and academic articles about the recommendation systems. We added the necessary benefits to our system and benefited from the experience of these authors. We anticipated the problems that we might face with these articles and overcome some problems beforehand. As all the systems have, recommendation systems have different application methods and algorithms. Most renowned ones are content-based filtering, collaborative filtering, and hybrid approach model. There are also studies on these in the literature.

Taşçı (2015)'s study that content based media tracking and news recommendation system, news are classified, summed up, and recommendation system is designed by correlating news to the user profiles. Three methods of recommendation systems, content-based filtering, collaborative filtering, and hybrid-method approach, are used in this study. News are classified by keywords and summed up. These summaries are used to make recommendations based on time and on user profile.

A Personalized Online News Recommendation System makes a personalized news recommendation system with collaborative filter based on dynamic update (Saranya & Sadhivasam, 2012). News' popularity and whether it is new or not are also considered. When analysing news, there is a problem in scaling, however this problem is solved by using hadoop technology. In the conclusion, news that come from different sources and different categories are grouped according to different categories with a more personalized style.

In another study which works on the topic of user profile and how to calculate user's attention, content-based filtering is applied with k-NN approach (Pazzani & Billsus, 2007). Similarly, we also used k-NN approach in our study. This study, apart from k-NN, is used Naive Bayes classification system algorithm (Pazzani & Billsus, 2007). The importance of last read article by the reader is pointed out in this study, as

we do. k-NN is one of the most important non-parameter algorithms in pattern recognition field and it's a supervised learning predictable classification algorithm (Yong, Youwen & Shixiong, 2009). k-NN classification algorithm predicts the test sample's category according to the k training samples which are the nearest neighbours to the test sample, and judge it to that category which has the largest category probability (Yong et al.). Besides recommendation systems, it is important to be experienced about data mining and text mining, due to the fact that unstructured texts take more space, and are hard to process.

An Analysis of Recommender Algorithms for Online News elaborates recommendation systems as online systems and compares their performances (Doychev, Lawlor & Rafter, 2014). They stored articles with MongoDB and they used Elasticsearch. They used fifteen recommenders is that, six popularity-based, seven content-based, one feedback-based, and one recent-based recommenders in total. In addition, the already read ones are eliminated before starting the reading.

In another study, while pairing documents that are going to be recommended and users by defining each as virtual elements, and storing each user and documentation combination, finds similar users (Suchal & Navrat, 2010). Furthermore, they used this similarity to recommend documents and these document processes are done in Full Text Search library, as well as we do in our study.

Using Topic Models in Content-Based News Recommender Systems, researchers do a content-based recommendation system for a small group of news and a small group of users. Three categorization algorithms are compared: Naive Bayes, k-NN, regression and regularized linear regression. With regards to these conclusion, the worst with the worst case is Naive Bayes, k-NN has the best performance. In the Cold Start Simulation, LDA (Latent Dirichlet Allocation) has the best improvements statistically for all the methods (Luostrainen & Kohonen, 2013). Figure 2.1 shows the basic steps of methodology used and Table 2.1 shows the conclusion

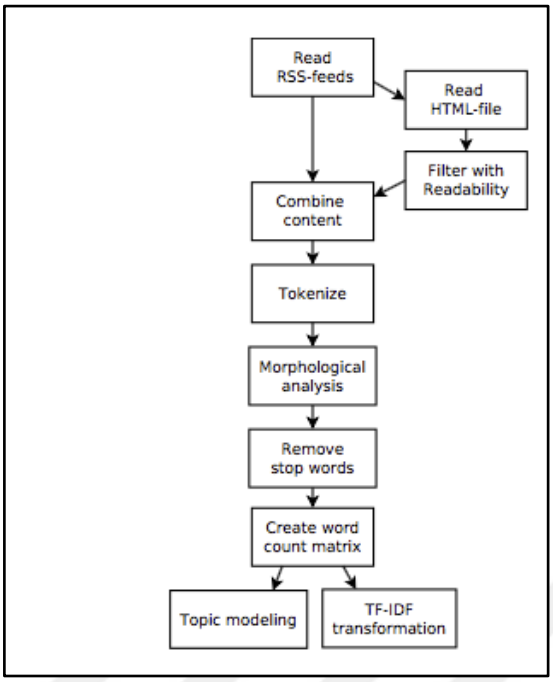

Figure 2.1 The data collection and pre-processing (Luostarinen & Kohonen, 2013)

Table 2.1 The rating distributions from each user (Luostarinen & Kohonen, 2013)

	Rating					
User	1	2	3	4	5	Total
1	629	393	783	912	1174	3891
2	297	310	487	823	960	2877
3	521	638	322	519	187	2187
4	100	197	259	341	204	1101
5	54	84	107	162	147	554
6	33	69	81	100	44	327
7	16	8	90	82	77	273
8	30	20	37	18	41	146
9	3	13	35	48	20	119
10	1	3	27	29	45	105

In another research, however, takes a different approach to the study by regarding the article's reading time, length of the article, and word placement in the article. Clicking behaviours of Google News users are investigated. Additionally, they used Bayesian Classification algorithm for analysing user behaviour (Liu, Dolan & Pedersen, 2009).

In another study, resembling to our study, users submit what they are interested as words and phrases at the first place to keyword based recommendation system. Later on the study, system tries to recommend best results based on user's submission (Schafer, Frankowski & Sen, 2007).

In StreamRec study, is regarded as whether an article is read first or last. If user re-evaluated any given article, another process is initiated. In Hot News Recommendation System, recommendations encapsulate the hottest and most read news that are published in different sources in the same day (Chandramouli, Levandoski, Eldawy & Mokbel, 2011).

Uzun (2005), argues that all the words in an article do not have the same chance to be the keyword. He says even if the frequency of a word is high, it should not be chosen as the keyword, right away, he argues that, it is clear that all the words do not have the same prior chance to be a key word in a document. Firstly, the all tokens in the document should be identified by using delimiters such as spaces, tabs, newlines, dots, etc. Even though propositions may appear many times in a text, they should not be labelled as a keyword. The second term in the TFxIDF score seems a barrier for such words, but may not be 3 adequate. Therefore, it can be a solution to eliminate by assigning prior probability of zero. Moreover, non-alphanumeric characters and numbers should be eliminated (Uzun, 2005).

Content-based news recommendation article compares the similarities between articles, creates an user model, and make recommendations according to these two steps. They found user models from past logs and from similar users who have read same articles (Kompan & Belikova, 2010).

We used Cosine Similarity approach to calculate document similarities. Başak (2009) says that, Cosine Similarity expresses similarities between two *n*-dimensional vectors as angle via cos, and it is frequently used for document comparing. Bielikova and Kompan states that cosine similarity is the most efficient approach. (Bielikova & Kompan, 2010). PostgreSQL Full Text Search library provides fundamental methods

and data of text-mining. Hoffman(1999), too, focuses on bag of words while finding the similarities between documents. With Latent Class Analysis, he makes documentations separated and modelled. In another study, Hoffman (1999), contents' similarity calculation is done by calculating semantics and frequency rate of words. Documents' similarity calculations done according to Fisher Kernel Function and Probabilistic Latent Semantics model.

In another study which is a blog reading application, the authors crawled the blog sites and delivered them to the users in the social media environment and the mobile reading environment. In this study, they used Apache Nutch technology and used PostgreSql as the database server (Seven & Dost, 2016).

Recommendation systems have some recognized challenges. We also encountered these problems. In a recent study, difficulties encountered in the recommendation system were revealed (Ozgobek, Gulla & Erdur, 2014), which are cold starting, recency, changing attention of the users, contents that are not structural, and privacy problems. Cold start problem is the process of recommending news without having any information about a new user. We solved this problem keyword based recommendation system and selecting the categories of news that might be of interest in the beginning. Recency problem is about the desire to read the most recent news, we solved this problem by collecting new news every 15 min in every fifteen minutes. Even though there is no solution for changing user attention, this problem may be solved by recommending the most important news to the user. Analysing the non-structural contents is one of the hardest problems, and needs extra working time. However, text-mining is the solution for this problem.

The result of our research on the technology side, PostgreSQL has provided positive results for us as the Full Text Search library. The vector space model we obtained with ts-vector and document has been an important tool in finding similarity. We first used the Apache nutch tool to crawl news online. However, the use of RSS feeds has given us more productive results, as some news sources block us when we arrive with nutch and use more RSS feeds. In the parsing of the news htmls, we first

used the Windows form browser tool. However, when parsing too much html, we gave up this method because of the memory leak problem. Instead, we performed a html parse with a nuget library. In web services, .net's web api technology has provided us with many technology infrastructures, which has given us positive results. With token based authentication approach, we aimed to create a more secure system infrastructure and we realized this. On the client side, the use of popular javascript technologies AngularJS and Typescript enabled us to create a pleasing visual interface and a much more performance application.

CHAPTER THREE
RECOMMENDATION SYSTEM APPROACH USED IN THE THESIS

In this chapter we present in detail our approaches and algorithms of the personal news recommendation system. These models generally elaborate on content similarities and user profiles.

3.1 Content Based Filtering

In this method, we made recommendation based on the content and preferences the user has read in the past. While developing the recommendation algorithm, we considered the *k*-NN classification approach. We compared to new document one by one with users' previously liked news. We used cosine similarity and PostgreSQL Full Text Search to calculate similarities between documents. We recommended this new news to the most similar ones based on the news users liked before. At this point, we thought of each user as a class. The elements of each class consist of news that users have previously liked. At this stage, what we need to do is to find out which class the new incoming fabric belongs to. We got the first *k* news that looked like the most. The number *k* here represents the most optimal number determined by us. We recommend the new news to users who like the first *k* news. Figure 3.1 shows how the system works.

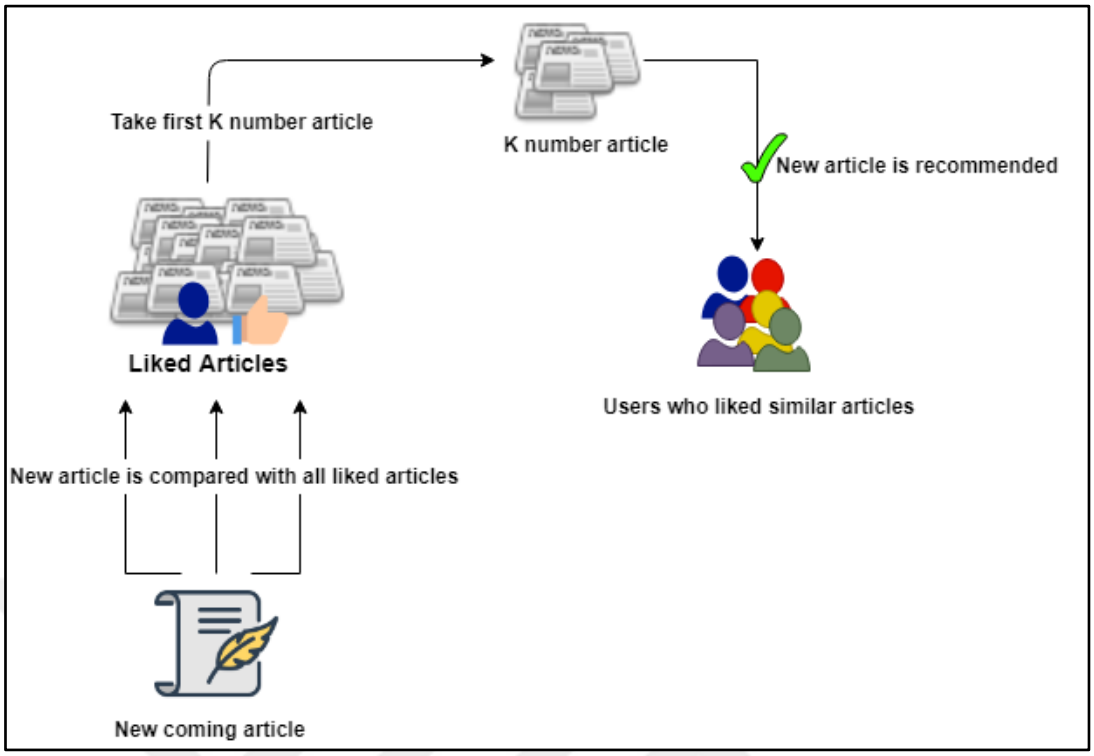

Figure 3.1 Content based filtering method of OKURSS

The number *k* here is determined by the following mathematical formula. The exponent part of the log statement contains the total data set. In other words, it represents all the documents in which this news is compared, the contents of the news. In the base of the log statement, as in Figure 3.3, the number of news with a rank score similar to 0,98 is written. Accordingly, the k number giving in the formula 3.1.

$$k = \log_{total\ news} similar\ news \tag{3.1}$$

When a news is added to system, we saved to database vector space models(VSM) of contents with the help of a written trigger in database, in this way the conditions for comparing documents with VSM are set. Function that is responsible for cosine similarity and the trigger saving documentations' content to the database are shown in Figure 3.2 and Figure 3.3. In order to create a ts-vector of documents, the trigger executed for the article table at the registration stage of each article and ts-vector conversion is performed automatically.

```
CREATE OR REPLACE FUNCTION articletsvtriggerfunction() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Check that empname and salary are given
        IF NEW."ArticleCleanSource" IS NULL THEN
            NEW.article_tsv := to_tsvector(NEW."Description");
        ELSE
            NEW.article_tsv := to_tsvector(NEW."ArticleCleanSource");
        END IF;

        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER articletsvtrigger BEFORE INSERT OR UPDATE ON "Article"
    FOR EACH ROW EXECUTE PROCEDURE articletsvtriggerfunction();
```

Figure 3.2 Generating text search type vector of article news code

```
CREATE OR REPLACE FUNCTION public.find_distance_article(
    querey_param text)
    RETURNS TABLE(rs_articleid integer, rs_title text, rs_rank real, rs_userid integer)
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE
    ROWS 1000
AS $BODY$
BEGIN
   RETURN QUERY
SELECT "arc"."Id","Title",max(ts_rank(article_tsv,query)) as rank,"op"."UserId"
  FROM  "Article" as "arc"
  inner join "UserOperation" as "op" on
  "arc"."Id" = "op"."ArticleId",plainto_tsquery(querey_param) as query
  group by "arc"."Id","op"."UserId"
  having max(ts_rank(article_tsv,query)) > 0.9999
 order by rank desc;
END
```

Figure 3.3 Find distance between two document code

3.2 Collaborative Filtering

In this method, we grouped users and news. We applied the recommendation method happens by finding the similarities between these groups. Collaborative filtering is a filtering process that uses other users' views on articles or news. In our project, adding friends to benefit the collaborative filtering takes place. This instance will lead to finding similar users, since similar users will read and elaborate similar news, the process of collaboration expands. With the help of this situation, same news

can be recommended to similar users. It is highly possible that, since content-based filtering recommends same genre of news, this method may be less boring for the users. However, there is less collaborative filtering in project.

Our collaborative filtering system works like this: the users, when they read an article, help filtering system with their preference of news, since their preference of news are recommended to similar users. It is important to find similar, this process happens via deducting similarities between news that are liked and read by the users, these deductions are renewed with each news. In this way, dynamism of the system increases. To find similar users, number of read articles in a category is brought. For example, one user has read *X* number of articles in *B* category, *Y* number in *C* category. This information is brought for every user. During the comparison, users' reading number in the same category are proportioned. For example, A user have read one hundred articles in *X* category, whereas *C* user read fifty articles. When these numbers are proportioned, the result is 0.5. This proportioning is done for every user, at the end, the users with the closest number will be identified as similar users. Thanks to this process, new articles will be recommended to similar users more accurately. Recommendation information is saved with the user and article information. Figure 3.4 shows our collaborative filtering algorithm pseudo code.

There are two different methods in the collaborative filtering, one being user-based, the other being news-based. Users are taken as a set in user based method. Similar news is recommended to the users in these sets. In news-based method, however, these user sets are formed regarding the news the users read. Thus, recommendation system makes smarter decisions, gradually.

```
function find_similar_users_by_collaborative_algorithm()

    reading_categories
    matched_users
    restricted_users
    score <- 0
    sum_value <- 0

    FOR each main_item on the reading_categories
        IF restricted_users not contain main_item.user_id THEN
            score <- 0
            FOR each sub_item on the reading_categories
                IF sub_item.user_id not equal main_item.user_id THEN
                    sum_value <- 0
                    FOR each main_category on the main_item.categories
                        FOR each sub_category on the sub_item.categories
                            IF main_category.id equal sub_category.id THEN
                                sum_value += get_rate(sub_category.count_read,
                                                      main_category.count_read)
                                break
                            END IF
                        END LOOP
                    END LOOP
                    IF score less than sum_value THEN
                        score.user_id <- sub_item.user_id
                        score.value <- sum_value
                        restricted_users.push(score.user_id)
                    END IF
                END IF
            END LOOP
            matched_users.push(main_item.user_id)
        END IF
    END LOOP

    return matched_users
```

Figure 3.4 Collaborative filtering algorithm pseudo code

As an example, Figure 3.5 shows the users *x*, *y*, *z* and *t*, and the read categories of these users, and the read counts for those categories. When we want to find the user most similar to *x*, when we rate and add the number of readings of the same categories, the user pair with the highest rate will be the most similar users. Function to calculate similarity of two users can be formally written as in Eq. 3.2.

$$SIM(X,Y) = \sum_{c\in\{sport,news,\ldots,art\}}^{n} \left\| \frac{x_{read}^{c}}{y_{read}^{c}} \right\| \tag{3.2}$$

where x_{read}^{c} represents, number of news of user *X* has read in category *c*.

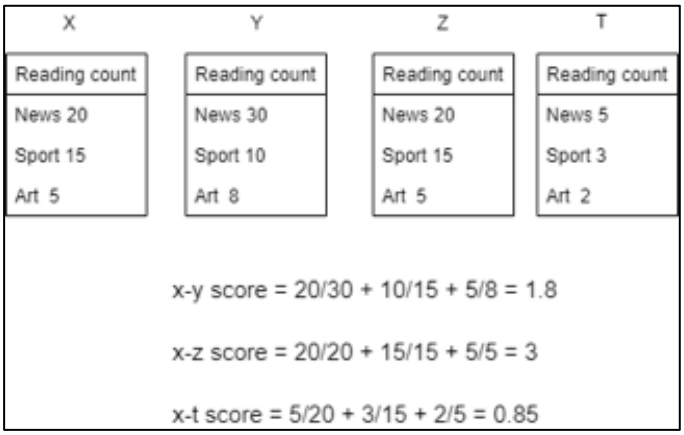

Figure 3.5 Sample data from collaborative users

3.3 Keyword Based Recommendation

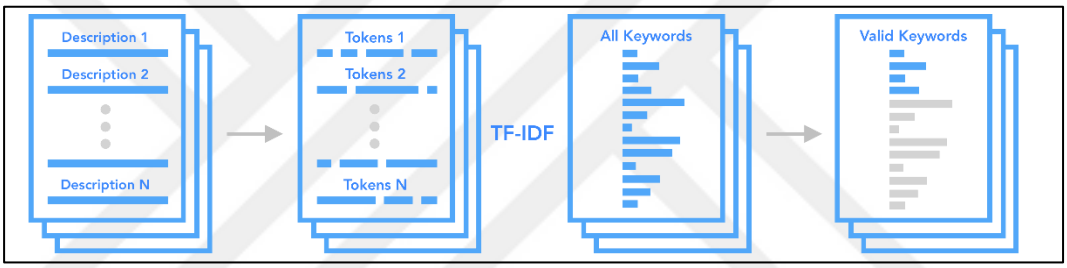

Figure 3.6 Keyword extraction method (Newcomb, 2018)

In this method, the users submit the keywords about a topic or a category. In our project, keywords are the most frequent words chosen by counting every word. The most frequent ten words are compared with the words that users submit, the users with the most word match get recommendation for that news. We implemented these steps when extracting keyword from documents:

- Remove stop words from documents.
- Extract title and metadata from document.
- Detection of words in bold italics or other special characters.
- Detection of more than 10 words.
- Matching these extracted words to keywords entered by users.

3.4 Algorithm

Our recommendation system algorithm is based on hybrid approach. We created the recommendation algorithm by combining our different algorithms in one place. Recommendation system algorithm have 2 stages: Batch Recommendation System and Real Time Recommendation System. Batch Recommendation System in 3 different steps. We have the full text of the news and the user read the news in the past. Thus, we have provided a suitable environment to realize the content-based recommendation system.

Firstly, we made the news recommendation to the users with the method we explained under the title of content-based recommendation system. Secondly, we implemented the recommendation system with the collaborative method that is the recommendation system based on the relationships with the users' friends. Finally, in order to realize the recommendation of users' favourite keywords, we scanned all keywords in the new incoming news and run the latest keyword-based recommendation system. These three methods work step by step in the recommendation system engine. In the real time recommendation system, we added popular articles and news lists from the user profile in addition to those made in the batch. These two methods are explained separately in the next section. The methods applied as hybrid in the recommendation system algorithm affect the result list with different coefficients. We formulated this approach in formula 3.3. *RS* refers to result set of documents, cb content based approach, cl collaborative and kw is keyword.

$$RS = RS^{cb} \cup RS^{cl} \cup RS^{kw} \tag{3.3}$$

$$RS_k^{cb} = \{d_1, d_2, \ldots, d_n\}, k < n \tag{3.4}$$

$$RS^{cl} = \{u_1, u_2, \ldots, u_n\} \tag{3.5}$$

$${}^{kw}_{k}RS = \{d_1, d_2, \ldots, d_n\}, k \in d_n \tag{3.6}$$

where $d1, d2, .. dn$ represents, *set of documents in Eq 3.4 and 3.6,* $u_1, u_2, \ldots, u_n$ represent, set of users in Eq. 3.5.

CHAPTER FOUR

RECOMMENDATION SYSTEM APPLICATION: OKURSS

In this chapter, we detailed OKURSS (www.okurss.com) which is a personalized news reading application with technical details. We explained the goal of the OKURSS project, detailed functions of the project, technical details about development of the project, what the system is made of, and sustainability of the system. Also we added the screenshots from different platforms such as web browser, Android and IOS. Figure 4.1 shows the web platform.

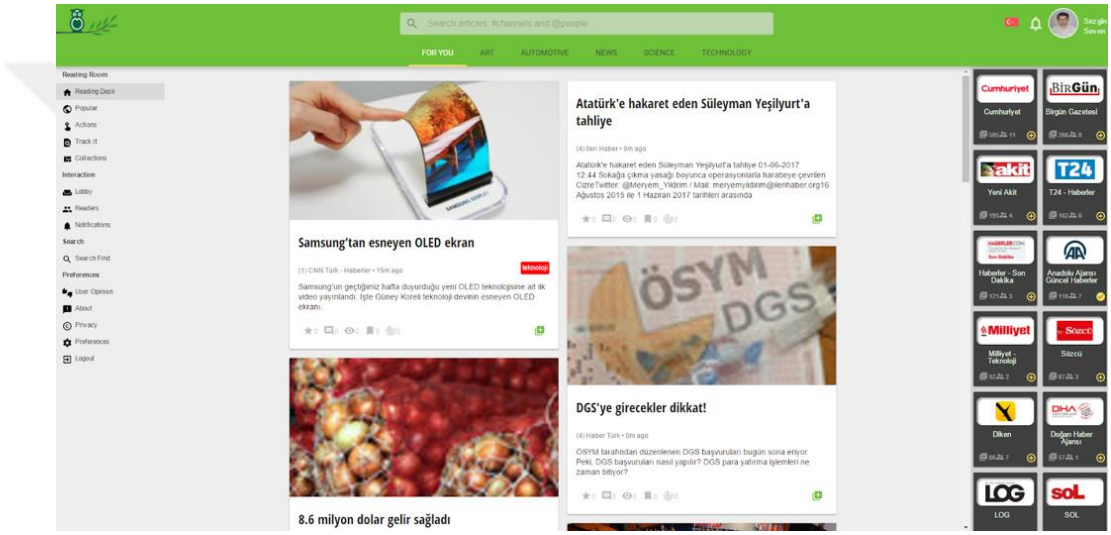

Figure 4.1 OKURSS homepage

OKURSS is a project that has a lot of functions besides reading news. In the project, which has a social media side, one can share what has been read, can make comments about news, can compare the same news from different channels. The properties of OKURSS as follows;

Reading desk is the screen where one can find recommended news. Recommended news are shown here, in the recommendation system algorithm. This section has the most suitable news for the reader. Figure 4.2 shows the screen where news is read. We have created a clean reading environment to allow users to focus only on the content to be read. Users can star in the news on the left-hand panel, add to the archive, share and track of news. Users can also change the reading themes from the top right panel.

And they can also enter comments from the area just below the news.

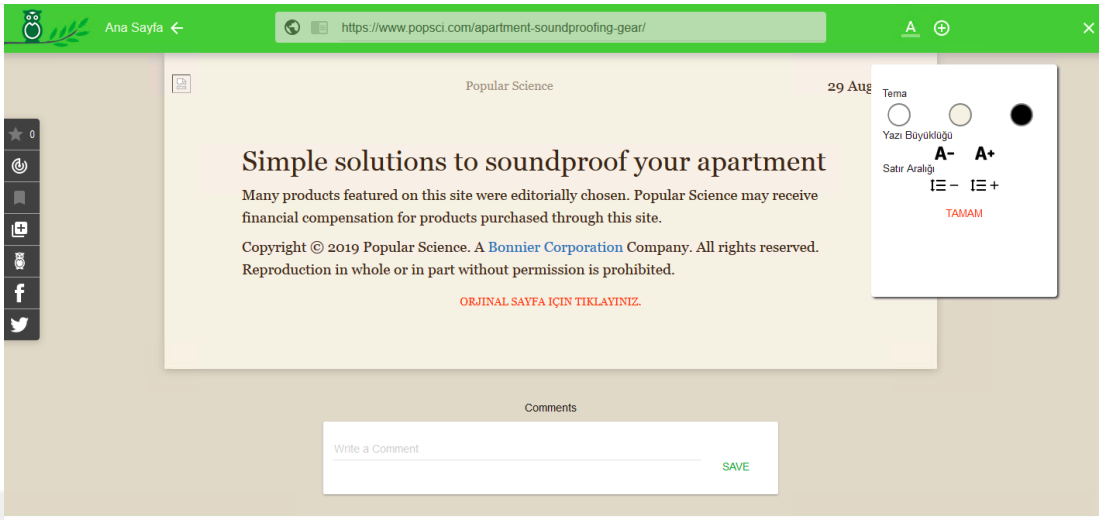

Figure 4.2 News reading screen

Popular articles are shown on the screen where one can find the news that has most readings, daily, weekly, monthly. This section is designed for reader to keep up with the agenda.

In activity screen, users can like, archive, comment, and follow the news. These actions are listed in this section. With this screen, users are able to read what they like in the past and see what they like. In OKURSS, we can think of it as a journey to the reading past.

In tracking section works with the recommendation system and this system lists users preference of created keywords. Users submit the keywords which they prefer to follow. Recommended news based on these keywords are listed in this section.

In the Collections tab, users can create a magazine-like collection by bringing the news together. And they can publish these collections as a news channel and share them with other people. Under Collections, an environment where users can edit their collection is provided. Figure 4.3 show creating new collection screen.

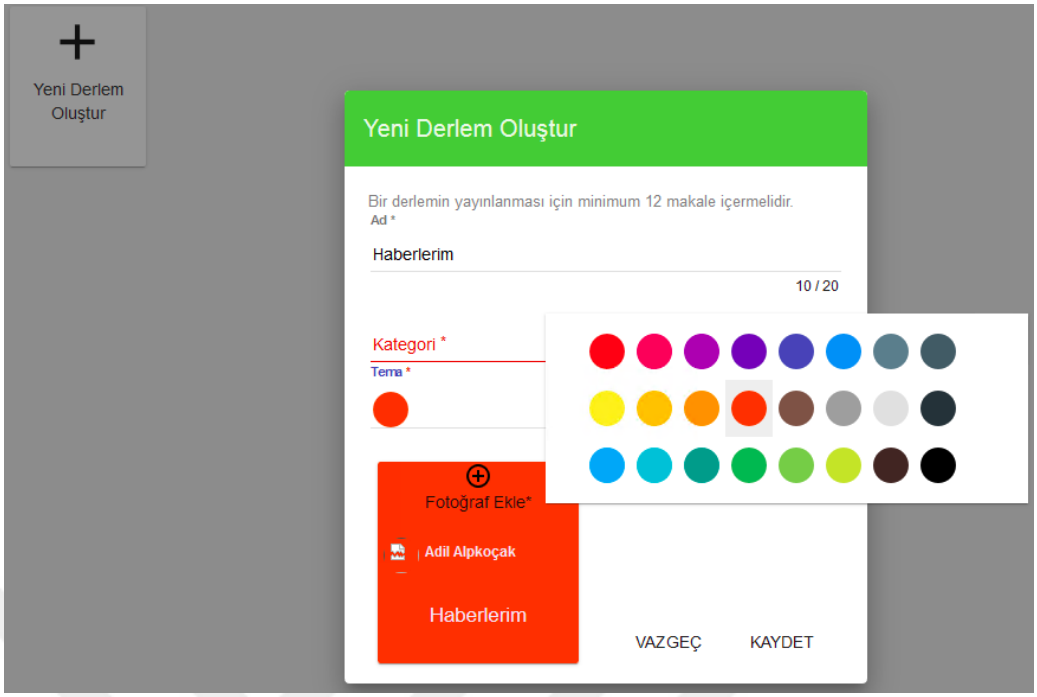

Figure 4.3 Create new collection

Lobby is a section where users can see what their followers comment or like. The reason why this section is called lobby is that it resembles to a place which people gather and read news. We added lobby screenshot to Appendix 1.

Readers section is the section where all the readers are listed. Notifications is where the application notifications are listed. Search and Find section is used for searching people, news, and channels by entering some words.

News content resources in our system is made up of RSS addresses of news, blogs, or content creating sites. In this context, getting content data plays the most important role for our system, and the first step of this system is to get data from RSS resources. The second step is to get contents' html from url information provided by RSS, and to parse html. After clean contents are made, these contents' classification and working of the system takes place in third step. Fourth step, with the data provided by first three steps, is a web service layer that directs data collected in the first three steps to front end projects, after the data are elaborated by some logical processes. In addition, since application has a social media site, web service performs acts concerning user processes. Fifth and last step is to develop client applications so as to transfer data to

user. These applications are in Android, Web Browser, and iOS. Figure 4.4 shows all parts of the system with technically.

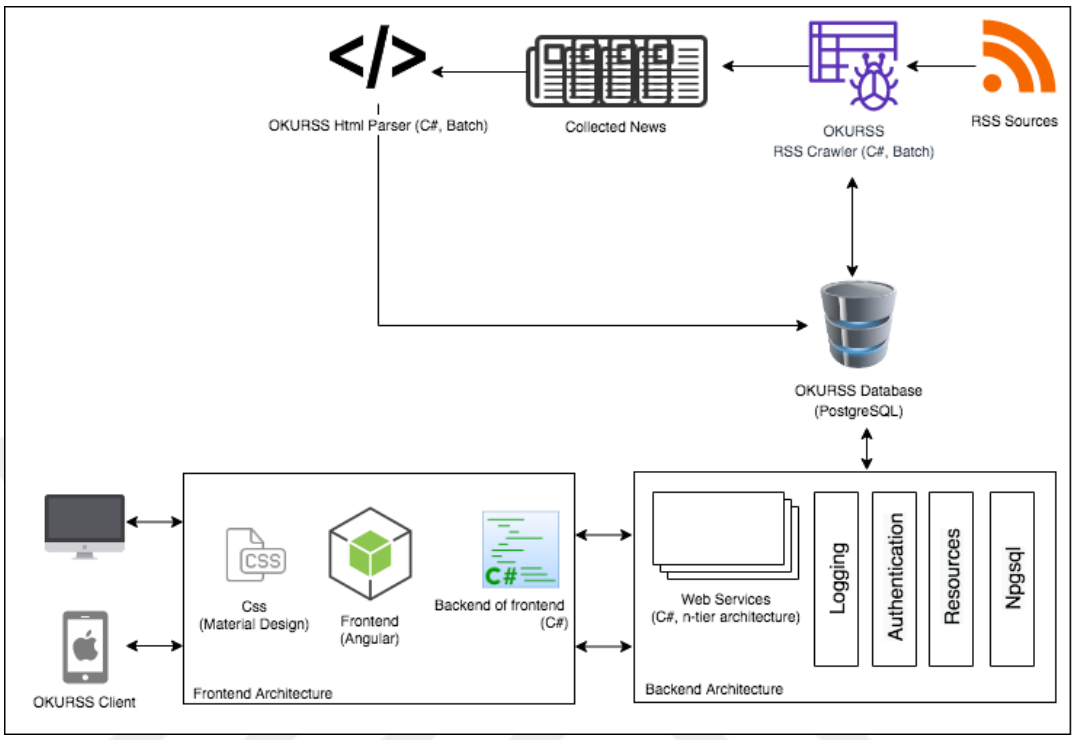

Figure 4.4 OKURSS backend system architecture

4.1 Database

In this section we explained database tables in detail and we shows the ER diagram. To create tables in the database, we used code-first approach. We used PostgreSQL, as database, and PgAdmin as database tool. We detailed how .NET Entity Framework and PostgreSQL connection is maintained via Npsql is discussed, in web service section. Figure 4.5 shows detailed descriptions of tabled and database the ER diagram.

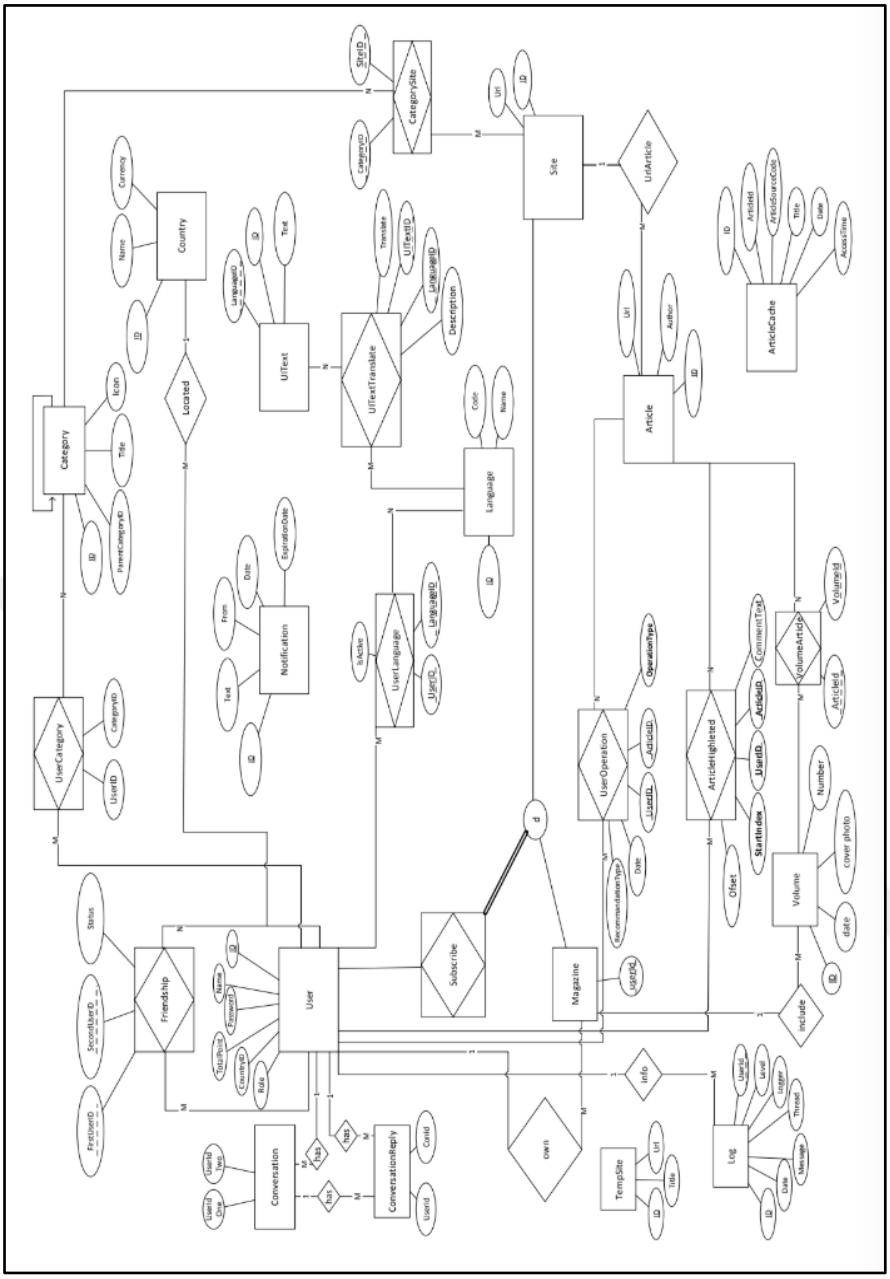

Figure 4.5 OKURSS Database ER Diagram

User is the table that stores the information of the users registered to the OKURSS application and the users to be placed by admin. Category is the table where the categories of articles are kept. UserCategory is the table that stores the users' preferred categories. Friendship is a table where friend relationships between users using the application are determined. Notification, Message operations occur only between admin and users. Article is the table that contains the articles within the article sites given as Url. The URLs collected by Nutch are accumulated in this table. UserOperation is a table that shows the activities that users do about the articles they read. (Like, Comment, Recommend, Read, Click). ArticleHighleted is a table that holds markups that users make on articles they read. The URLs for the sites to be crawled are found here. ArcticleCacheChannel, User-added posts are added first to the ArcticleCacheChannel table and then to the Channel table. CategoryChannel is the table that defines the categories of the channels. All user interface components to be used in UIText, Mobile and web applications are kept in this table. Language holds different language information. UITextTranslate is a table where UI Texts are kept in different languages. UserLanguage is a table that holds which language users prefer. Country is the table that holds the country to which the users belong. Collection is a table that allows users to collect their articles and create magazines. Collection, Journals and some properties of these issues are kept. CollectionArticle, Articles that will be included in the journal issues are kept in this table. SubscribeCollection is the table that holds the magazines that users follow. SubscribeChannel is a table that keeps track of the sites (articles, news, blog) that users follow. Comment is used for messaging between users and allows communication between two users. ArticeCache is a table that will feed the cache mechanism to be created for faster delivery of article articles to the user. Log, Logging structure will be created in order to send errors in the application and inform the user's operations.

4.2 Collecting News From RSS

The purpose of this application is to collect the url addresses of news by browsing the RSS sources periodically. We explained the methods used for getting news from RSS sources, technical details of the RSS parser system. Also we showed schematic presentations of RSS parser. We developed this application to parse all RSS standards. In order to get news from RSS sources, RSS channels made by an admin in an admin panel follow and register news channels regularly with a daily auto-working application. Firstly triggered RSS parser application collects url addresses of news from RSS channels and also saves these collected url addresses in the database. We take titles, url, summaries, and image urls of news from RSS source and we saved this information to database in "ArticleCacheBuffer" table. This table serves as the exchange of data between different programs that trigger each other. Afterwards, we triggered the html parsing application, which we explained in the next section, and html parsing takes RSS information from buffer table and it downloads the detail html of news. Figure 4.6 shows how the RSS parser application works in detail.

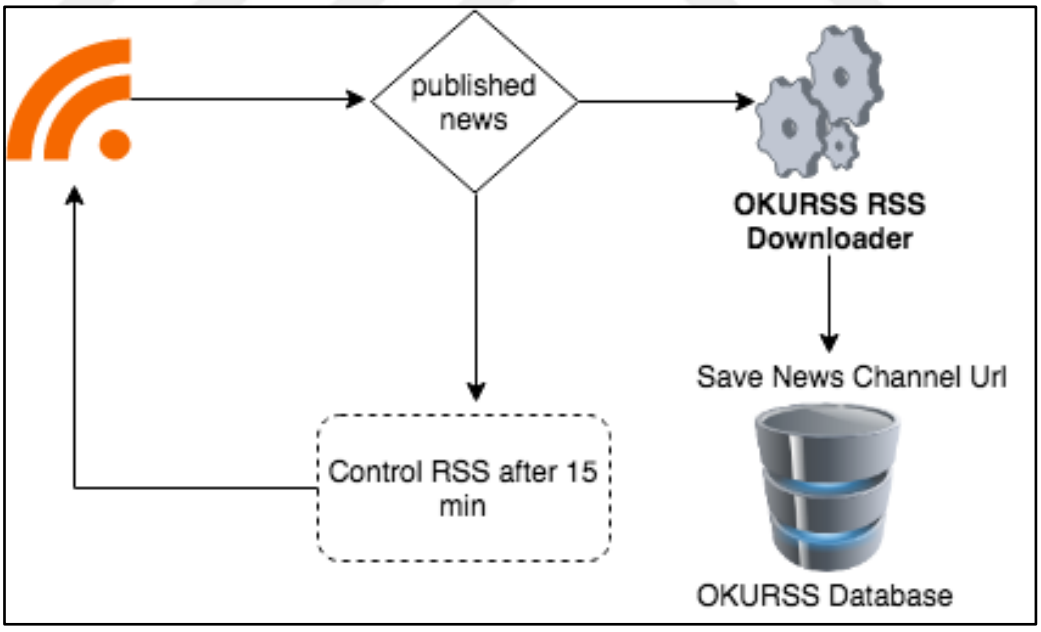

Figure 4.6 Collecting news from RSS

4.3 Html Downloading and Parsing

The purpose of this application is to download the html files of the news channels online from the Internet and save the contents related to the news in the database. In the previous section, news urls taken via RSS Downloader are registered to "ArticleCacheBuffer" table, and trigger html parsing device. The triggered html parsing application starts to download contents of the urls from "ArticleCacheBuffer" table with HtmlAgilityPack library. The downloaded content's html, title, images, and other urls are taken, this information are the basics of an news. These information feed "Article" and "ArticleCache" tables. Article table stores basic data of news that does not contain too much data. The ArticleCahce table contains all the content and detailed information about the news. In order to store a lot of data for the news, we cleared ArticleCache table, which stores the big content of the news, in different periods. With a trigger that we added to the Article table, we converted all of the content of news to VSM (vector space model) and saved to the Article table. VSM will be used for the purpose of calculating similarities between the articles. Detailed information about the calculating similarities between documents is detailed in Recommendation System Section. Figure 4.7 shows how the html parsing system works in detailed in.

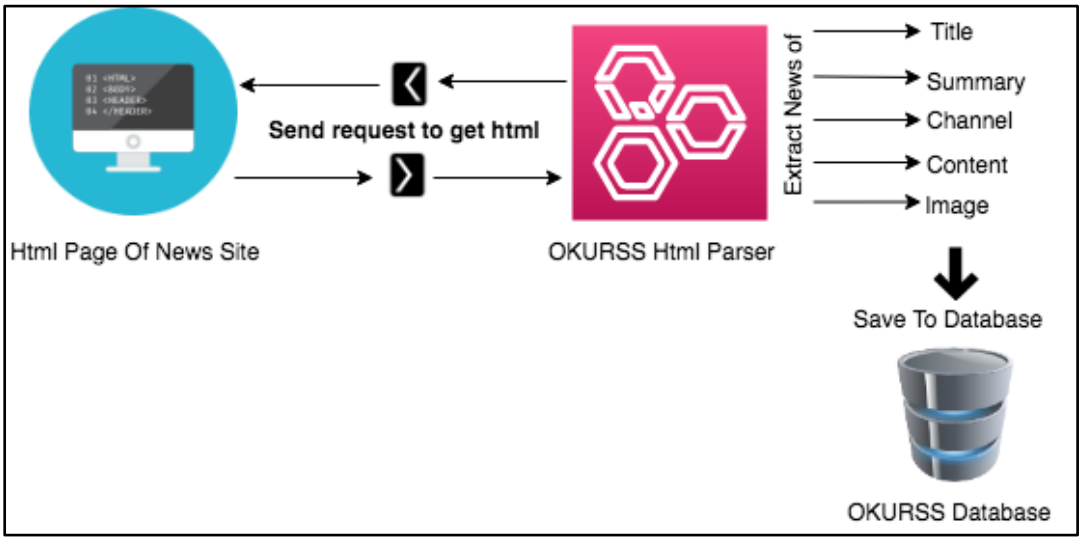

Figure 4.7 Html parsing

4.4 Online Application Layer

The purpose of this application is to communicate between database and clients. Everything up until this section is about preparing the dataset. This section elaborates on the service project that is designed to present the pre-set data to client applications. We developed this service project with the architecture of RESTFull API, framework of Microsoft .NET, and MVC Web API project. Web service project facilitates the connection between client and server. This connection happens with the governing of http protocols. By standing between client and server via IIS, connection is maintained methods such as; POST, GET, PUT, DELETE. Figure 4.8 shows the http API working logic.

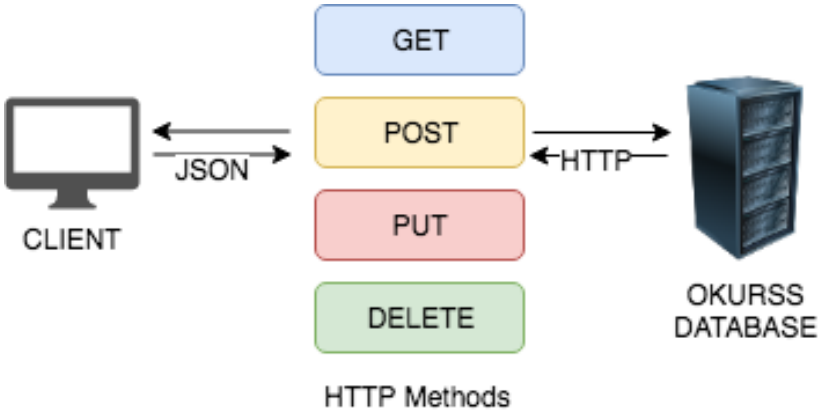

Figure 4.8 Http API 2.0 Protocol

We built up the project on MVC software architecture and *n*-tier architecture. MVC (Model View Controller) is the rule of dividing code and view. Codes' different purposes become more developable, testable, and easily repairable via division. Model is the layer on which data-related operations are performed. User interface operations are executed in the view layer. The controller provides the connection between the model and the view. Regarding this issue, we used MVC software architecture. Figure 4.9 shows the MVC's logic.

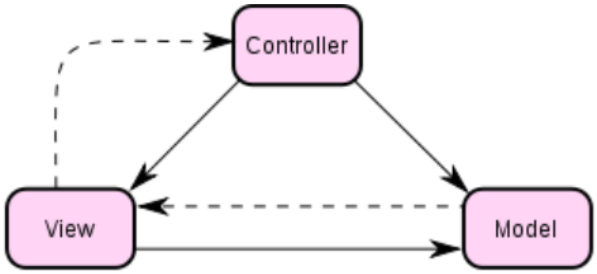

Figure 4.9 MVC Structure

Web Service project is also manages all of the backend task. We developed all the parts that should be in a backend project and ensured that the needs are complete. Figure 4.10 shows all part of the web service backend project and in the following sections we explained them all parts.

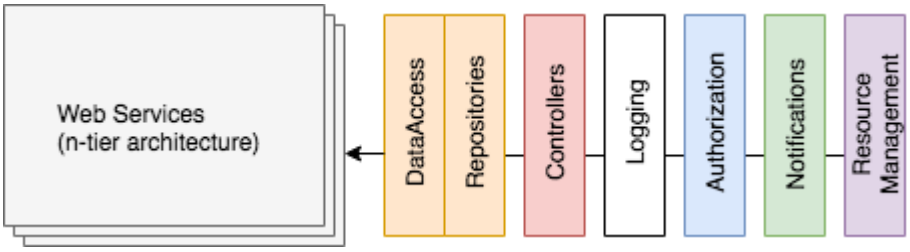

Figure 4.10 Backend architecture parts

4.4.1 Token Based Authentication

The purpose of the authentication layer is that, when communicating with clients is to make an authorized connection with the security layer. When connecting with the other applications, one wants one's programs and services to connect with other programs one has. That's why one has to add a security layer between services and client. Client is authorized with a token when interacting with server. Service provides service to the client which comes with the token. This protocol is called token based authentication, and its life cycle is shown in Figure 4.11.

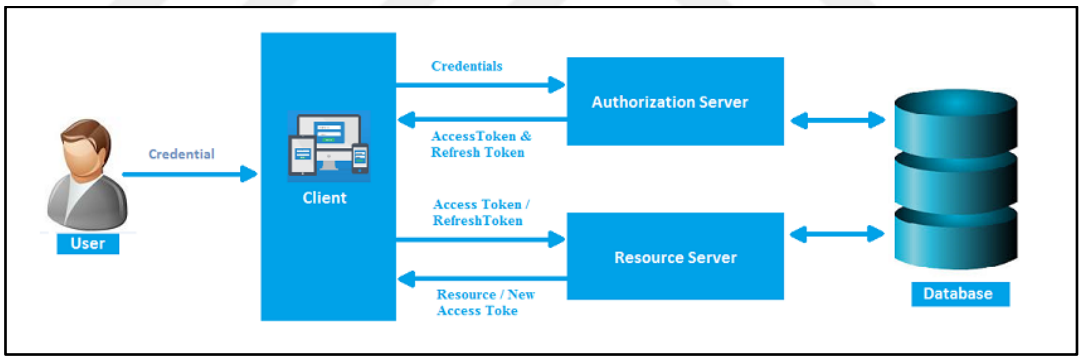

Figure 4.11 Token based authentication lifecycle (Ecanarys, 2017)

We make the identity control with token authentication functions by users entering their emails in login section. For users that have signed up successfully, a token is sent to the client by the server. We used this generated token to represent the user for all the web service calls. In token based authentication construction, this token made for user is sent to server whenever there is a request generated by the user. Via this token, user information becomes reachable for the server. Figure 4.12 shows our token based authentication construction's identity approval code.

```
public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
{
    context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { "*" });
    AccountManager manager = new AccountManager();
    AuthenticatedUser person = manager.IsFacebookUser(context.UserName);
    if (!String.IsNullOrEmpty(context.Password) && person == null)
    {
        person = manager.GetUserAuthentication(context.UserName, context.Password);
    }
    if (person != null && person.Status != UserStatusEnum.NotActived.GetHashCode())
    {
        var identity = new ClaimsIdentity(context.Options.AuthenticationType);

        identity.AddClaim(new Claim("Id", person.Id.ToString()));
        identity.AddClaim(new Claim("Name", person.Name.ToString()));
        identity.AddClaim(new Claim("Surname", person.Surname.ToString()));
        identity.AddClaim(new Claim("Email", person.Email.ToString()));

        context.Validated(identity);
    }
    else if (person != null && person.Status == UserStatusEnum.NotActived.GetHashCode())
    {
        context.SetError("invalid_grant", "Lütfen mailinize gönderilen linki onaylayın");
    }
    else
    {
        context.SetError("invalid_grant", "Kullanıcı adı veya şifre yanlış.");
    }

    await base.GrantResourceOwnerCredentials(context);
}
```

Figure 4.12 Token based authentication code

4.4.2 Logging

Logging is records that keep tracks of every activity in the system. Log constructions are important simply due to the fact that they detect cyber leaks, and system's failures. Since these kind of failures can happen, system needed this kind of a logging system. For logging, we used log4net library. Log4Net is a quite successful logging device for .Net.

In Log4Net system architecture, there is no direct layering. This architecture happens to be an outsider-third-party library. Logging is used in every layer. Configuration systems are other steps that logging can integrate to the system. Settings for log4net are uploaded to Web Config. These configurations decide on which layer the logging will happen.

There are seven layers of log4net;

• ALL; all the messages are logged.

• DEBUG; Logging level which is for developmental stage.

• INFO; we can log status information that may be beneficial during working.

• WARN; we can detect important things that are not errors.

• ERROR; error status is detected, system still works, though.

• FATAL; the messages that say app will close and do no more function.

• OFF; nothing is logged.

We saved all these logs to our database. It is possible that one can reach these logs either via admin panel or via sending a request to this table. Table 4.1 shows our log table in database.

Table 4.1 Log table

Log	
ID	number
UserId (FK - User)	number
Message	text
Thread	text
Level	text
Logger	text
Date	date

4.4.3 Database Connection and Configuration

We used PostgreSQL as a database server. We used Entity framework in order to connect the database and web service project, which is an ORM library possessed by .NET. Entity framework provides connection between the database and the web service project, hence, it is used for administering the database.

Since Entity Framework is a Microsoft product, it has a direct relation with other Microsoft products. However, PostgreSQL is a product of open source coded database, hence, it is obligatory that one has to use another library, in this case it is Npsql. Both Npsql and PostgreSQL work as a provider for the database. So, we used Npgsql library together with PostgreSql. Fundamentally there are two entity framework approaches, one being database-first and the other being code-first.

According to Database-first approach, classes for the tables are made after the pre-created database provides the tables, on the contrary, in the code-first approach, which we used code first approach, classes are made by .NET at the first place. These classes represent the tables in the database, connection between these tables are done via mapping. For example, if there were a foreign key connection between two different tables, either [ForeignKey("EntityID")] annotation, or adding the object that is foreign to the table would be sufficient. Tables' attribution can be specified by annotations. As another example, [Required] annotation means that attribute cannot be empty. So as to reach out to the tables' database, there is a set-up context for classes, after the classes are made up. Figure 4.13 and Figure 4.14 shows that our context class is derived from DbContext which is an EntityFramework interface.

```
5 references
public virtual DbSet<User> User { get; set; }//ok
0 references
public virtual DbSet<UserDevice> UserDevice { get; set; }//ok
0 references
public virtual DbSet<Category> Category { get; set; }//ok
0 references
public virtual DbSet<CategoryUser> CategoryUser { get; set; }//ok
0 references
public virtual DbSet<Article> Article { get; set; }//ok
0 references
public virtual DbSet<ArticleCache> ArticleCache { get; set; }//ok
0 references
public virtual DbSet<ArticleHighleted> ArticleHighleted { get; set; }//ok
0 references
public virtual DbSet<Conversation> Conversation { get; set; }//ok
0 references
public virtual DbSet<ConversationReply> ConversationReply { get; set; }//ok
0 references
public virtual DbSet<Country> Country { get; set; }//ok
0 references
public virtual DbSet<Friendship> Friendship { get; set; }//ok
0 references
public virtual DbSet<Log> Log { get; set; }//ok
0 references
public virtual DbSet<Notification> Notification { get; set; }//ok
0 references
public virtual DbSet<Tempsite> Tempsite { get; set; }//ok
0 references
public virtual DbSet<UIText> UIText { get; set; }//ok
0 references
public virtual DbSet<UITextTranslate> UITextTranslate { get; set; }//ok
```

Figure 4.13 DbContext entities

```
0 references
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{

    modelBuilder.HasDefaultSchema("public");
    modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

    modelBuilder.Entity<Category>().HasKey(e => new { e.Id });

    modelBuilder.Entity<Tempsite>()
        .HasKey(e => new { e.Id });

    modelBuilder.Entity<User>()
        .HasKey(e => new { e.Id });

    modelBuilder.Entity<CategoryUser>()
        .HasKey(e => new { e.UserId,e.CategoryId });

    modelBuilder.Entity<ArticleCache>().HasKey(e => new { e.ArticleId });

    modelBuilder.Entity<Friendship>().HasKey(e => new { e.Id });

    modelBuilder.Entity<ArticleHighleted>().HasKey(e => new { e.Id});

    modelBuilder.Entity<Conversation>().HasKey(e => new { e.Id });
```

Figure 4.14 DbContext model creating for entity

After creating the models belonging to the entries, we sent to database via migration method. We used commands below for applying the changes in database. Figure 4.15 shows our migration files to be sent database.

PM> add-migration –initial

PM> update-database

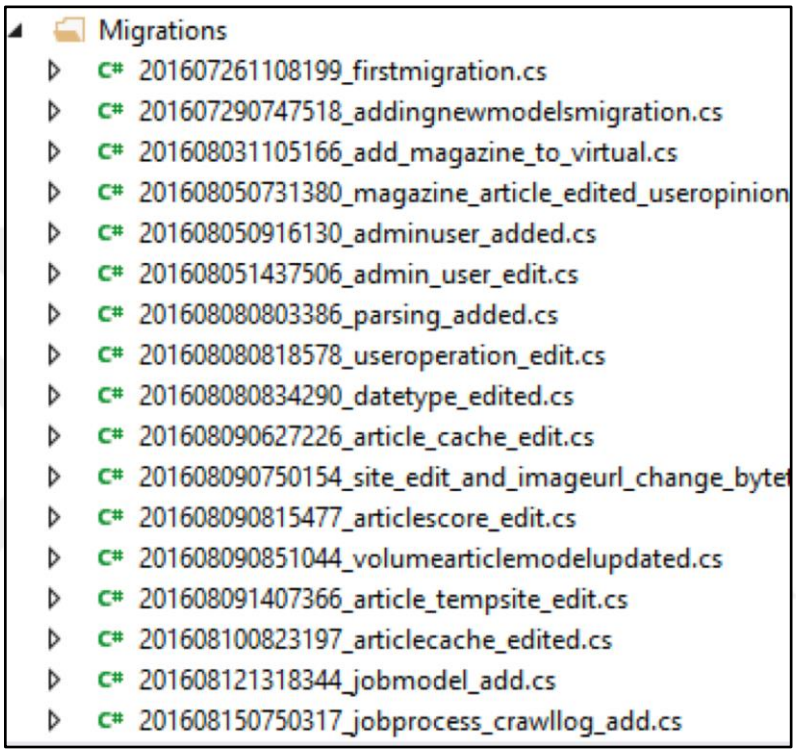

Figure 4.15 Migration files

"Up" method shows new tables and attributes, whereas, "Down" method shows to be deleted ones. A migration folder is created for every migration process, and since these migration folders are kept, it is possible for one to rollback any time. At the same time, these migration folders created by code first method make database more easily transmittable. With the help of Create Script, these can be applied to any database and free us from the job of backing up each time.

4.4.4 Repository Management

After the database is created, we created repository classes that make database processes function and provide access to database. We used functions such as, add, update, find, get etc. are main functions. We created this classes as generic. We derived every repository class from a BaseRepository. Figure 4.16 shows the class diagram about some part of the classes and BaseRepository class which encapsulates all database processes.

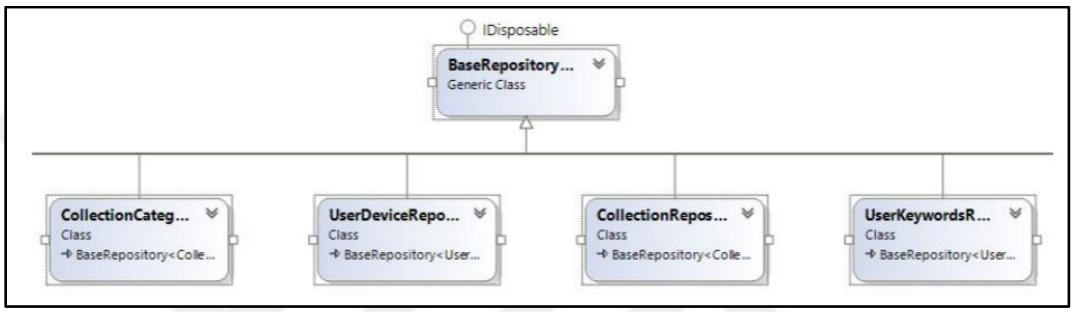

Figure 4.16 Repositories class diagram

4.4.5 Resource Management

We implemented different language to OKURSS. Due to the fact that OKURSS is a multilingual application, contents have to be supporting different languages, thus, contents are provided as dynamic. In the first level, OKURSS can be only used either in Turkish or in English, however, if the flag at the up right corner is to change to another, language of content and system will also change. RSS channels are added for English news to admin panel. When saving news, a flag determines content's language. Figure 4.17 shows English content OKURSS.

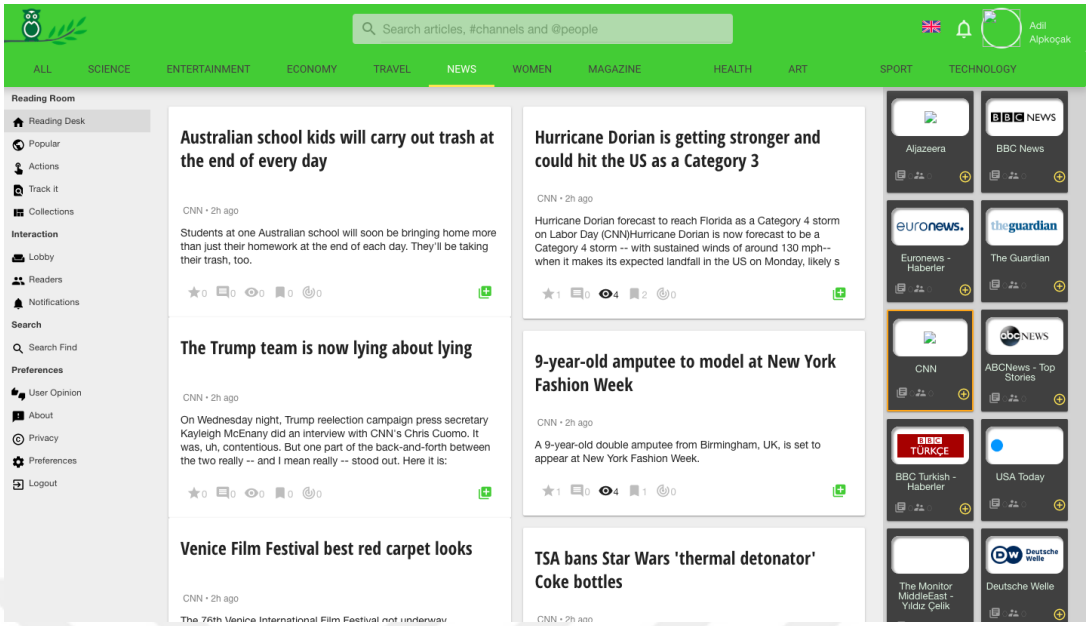

Figure 4.17 OKURSS English content

We designed two different tables for resource management. In the first table, a tag is assigned to each component and all of these tags are added to the table. Each element that appears as text on the screen is added to this table. Secondly, there is a value table containing the Turkish and English meanings of these tags. Turkish and English meanings are extracted by matching the table to which the tags are added. Table structures are shown below.

Table 4.2 UIText database table

UIText : All user interface components to be used in mobile and web applications are kept in this table.	
ID	integer
LanguageId	integer
Title	text

Table 4.3 UITextTranslate database table

UITextTranslate : This is the table where the tags in the "UI Text" table are kept in different languages.	
UITextId	integer
LanguageId	integer
Translate	text
Description	text

Table 4.4 Language database table

Language : Stores different language knowledge**.**	
ID	integer
Name	text
Code	text

4.4.6 Notification

In this section we explained notification service in OKURSS. Processes of notification sending are also developed at web service part. We used Push Sharp library, with this library we sent notifications to the IOS and Android apps. APNS is used for IOS, whereas, Firebase is used for Android. For users to get notifications, they have to submit their IDs to the system, an these information, including the devices', are kept safe while signing up to OKURSS. We stored device id information in a database table. Figure 4.18 shows some part of device data. Sending notification process happens by matching IDs and device information. OKURSS admin sent notifications at every entering message, users sent notification each other at news shared, and at friendships.

UserId [PK] integ...	Name text	TokenId [PK] character varying (5000)	IsActive boolean	Version text
69	Android	cZU8WAdNcHY:APA91bEpCNy55dQBLSuWrz4Dj...	true	5.1.1
69	iPhone	fff837ca0f1407243f4f15789a0610677408126f4...	false	10.2.1
74	iPad	4bc71a61d3568bbb5cdad9066e05ee48c28b12...	true	10.2.1
74	Android	cTYZQrpV20U:APA91bFNVpPdrd7A8HwUzRkRDm...	true	6.0.1
74	iPhone	d9c6607fa25c2861c0088bbdc93c16eb5be73b9...	true	10.3.1
74	iPhone	f0010926109ce8b95b1ad0eb116c9609ed57fbc...	true	10.3.1
74	iPhone	fdf9db8dd59bd1ed15dd5185ea2451a2bff046a...	true	10.2.1
77	iPhone	bb0a67cdc144e27304869c73526151f20b4e61...	true	10.2.1

Figure 4.18 Users device info for notification service

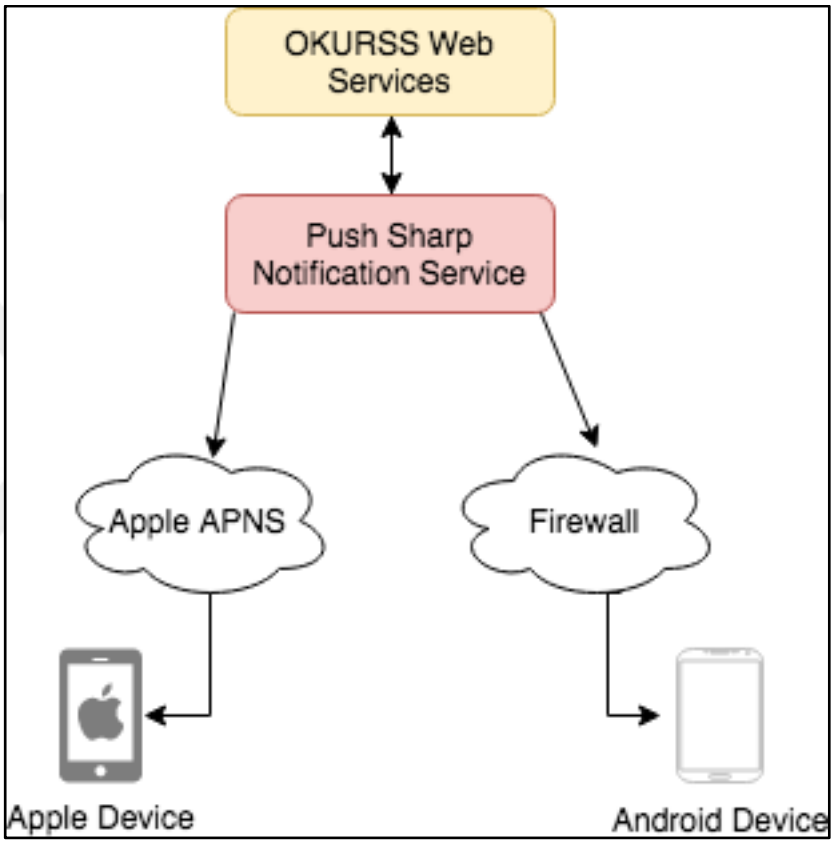

Figure 4.19 OKURSS notification service working logic

Figure 4.19 shows notification service working logic. This service consists of two parts as APNS(Apple Push Notification Service) and Firebase. Figure 4.20 shows our android firebase notification code implementation. Firstly, we prepared the data to be sent to the client. This data must serialize to JSON format. Then we added the necessary parameters to the header. Finally, we sent the prepared data to the client streamly. Figure 4.21 shows for APNS notification code. In this code, we took the registered device tokens from the database and added notifications to these devices, each of which was unique to the queue. The payload variable represents data in JSON

format. Figure 4.22 shows the prepared data variable. This data includes information about for notification type. If this type about sharing a news, notification message include about news information such as url address, title, content etc. Else if notification about friendship, the message has users information, else notification about system message the message has only text message.

```
var serializer = new JavaScriptSerializer();
var json = serializer.Serialize(data);
Byte[] byteArray = Encoding.UTF8.GetBytes(json);
tRequest.Headers.Add(string.Format("Authorization: key={0}", applicationID));
tRequest.Headers.Add(string.Format("Sender: id={0}", senderId));
tRequest.ContentLength = byteArray.Length;
using (Stream dataStream = tRequest.GetRequestStream())
{
    dataStream.Write(byteArray, 0, byteArray.Length);
    using (WebResponse tResponse = tRequest.GetResponse())
    {
        using (Stream dataStreamResponse = tResponse.GetResponseStream())
        {
            using (StreamReader tReader = new StreamReader(dataStreamResponse))
            {
                String sResponseFromServer = tReader.ReadToEnd();
                string str = sResponseFromServer;
            }
        }
    }
}
```

Figure 4.20 Firebase notification code

```
apnsBroker.OnNotificationSucceeded += (notification) =>
{
    Console.WriteLine("Apple Notification Sent!");
};
apnsBroker.Start();

foreach (var deviceToken in MY_DEVICE_TOKENS)
{
    apnsBroker.QueueNotification(new ApnsNotification
    {
        DeviceToken = deviceToken,
        Payload = _payload
    });
}

apnsBroker.Stop();
```

Figure 4.21 APNS notification code

```
_payload = JObject.Parse("{ \"aps\" : { \"alert\" : { \"title\" : \"" +
                        message + "\", \"body\" : \"" + article.Title + "\",\"articleUrl\":\""
                        + article.ArticleUrl + "\"},\"sound\" : \"default\",\"category\" : \"message\",\"badge\":"
                        + badge + ",\"mutable-content\": 1},\"attachment-url\":\"" + article.ImageUrl + "\"}");
```

Figure 4.22 Sample json data

4.4.7 Controllers

In this section we explained the controllers, which are the structures that the web services are separated on the basis of their work. There are seventy-two web services in total, and these sources are divided into eight groups according to their functions. These controllers can be listed below.

Account Controller; these services are responsible from user accounts. Registration, Facebook login, account verification, password update, and information update are some of the task these services have. Registration service for example is shown in Figure 4.23. First, with the HttpPost keyword, it is indicated that this method will be used with POST. The position which is at the top and working with the Route keyword is the name how the web service is called. To use this method, it is obligatory to use that name. After necessary checks are done, whether the registration is successful or not is shown to the user in JSON. In this code, the required fields are checked first and if there is any missing information, the service returns a negative response. After checking the missing area, email check is done. After all these checks, if the business logger is successfully completed by working in other layers, the registration process is performed and the service returns to successful response.

```
[HttpPost]
[Route("register")]
0 references | Sezgin Seven, 64 days ago | 1 author, 6 changes
public IHttpActionResult Register([FromBody] RegistrationRequest user)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    RegistrationResponse result = new RegistrationResponse();
    if (String.IsNullOrEmpty(user.Password) ||
        String.IsNullOrEmpty(user.Email) ||
        String.IsNullOrEmpty(user.Name) ||
        String.IsNullOrEmpty(user.Surname) ||
        String.IsNullOrEmpty(user.Password))
    {
        result.Message = "Eksik bilgiler mevcut, lütfen eksik alanları doldurunuz.";
        result.IsSuccess = false;
        return Ok(result);
    }
    if (!utilities.IsValidEmail(user.Email))
    {
        result.Message = "Lütfen geçerli bir mail adresi giriniz.";
        result.IsSuccess = false;
        return Ok(result);
    }
    result = registerManager.RegistrationUser(user);
    return Ok(result);
}
```

Figure 4.23 Registration web service method

Article Controller; in this class, we added the operations about articles. The web service method that is responsible for article flow in main page is placed in this class. There are multiple stream types in the main screen, but, their transfer to the main page is done by one method. These stream types are respectively, “for you”, recommended news, “lobby”, “featured”, and the news on channels and categories. We implemented these streams by queries written in the database. Queries can come from multiple tables as well as only from one table. Figure 4.24 shows stream query that feeds lobby, as an example;

```
SELECT
"uo"."UserId",
"uo"."ArticleId",
array_to_string( array_agg( distinct "uo"."OperationType" ), '-' ) as operationtype,
min("uo"."OperationDate")
FROM  "ArticleCache" AS "arc"
INNER JOIN "Friendship" as "fr" ON "fr"."FirstUserId" = useridparam
INNER JOIN "UserOperation" as "uo" ON "uo"."ArticleId" = "arc"."ArticleId"
AND "uo"."UserId" = "fr"."SecondUserId"  AND "uo"."OperationType" != 3
GROUP BY "uo"."UserId","uo"."ArticleId"
ORDER BY min("uo"."OperationDate") DESC
LIMIT Paging_PageSize
OFFSET PageNumber;
```

Figure 4.24 Lobby stream select query

In this query is shown in Figure 4.24, by inner joining Friendship and "UserOperation" table, the activity done by user's friends can be seen. Since we grouped "OperationType", we returned multiple actions in one line, and these processes are listed according to their dates, so as to show newest ones at the top.

The articles coming from database go under some other functions based on users, such as; total reads, likes, comments, how many people archived. Articles are taken from the database as raw data containing only certain fields. We generated a new modelling to this raw article data in the Business layer. The new model is sent to the client. Figure 4.25 shows this process.

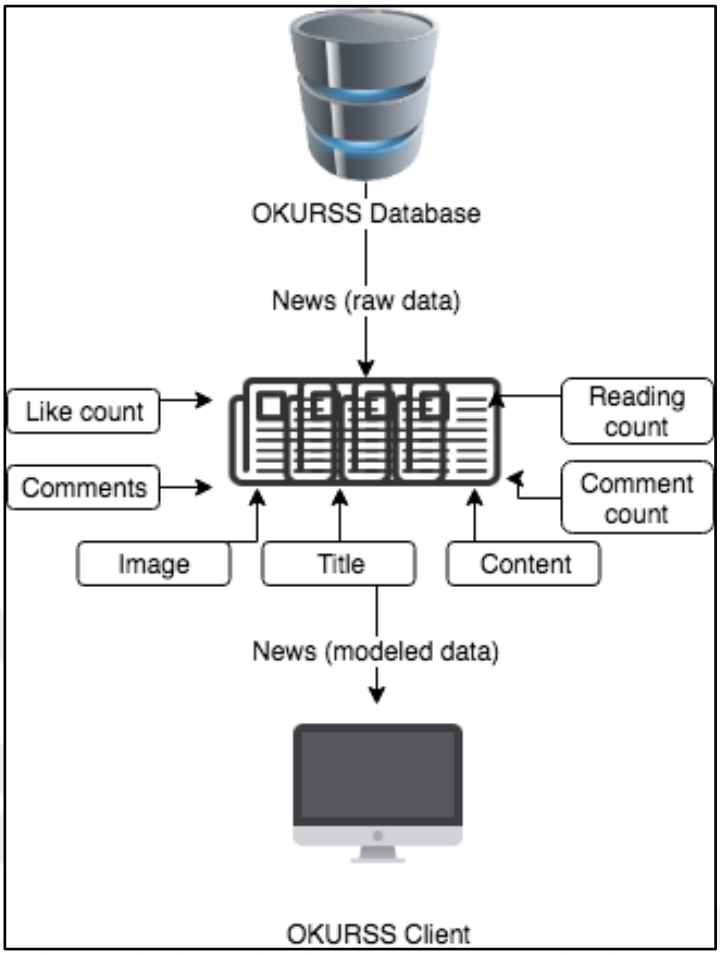

Figure 4.25 Converting raw article to modelled article

Friendship Controller is the class where the friendship operations happen, adding a friend, unfriending, and seeing the followings and the followers. Notification Controller manages notification operations. Channel Controller manages news RSS channels, as well as listing the channels, searching the channels, and subscription. Collection Controller manages the collection operations such as, creating a collection, adding or removing article, subscribing to a collection, and listing are true operations this section has.

4.5 Web Application

In this section of the thesis, we explained user guide and technical details of the web platform, and we added the screenshots of OKURSS. The purpose of this web project is to show the data prepared in the previous chapters to the users in the most meaningful and beautiful way. We have used up-to-date web technologies in this project. We have developed a flexible system, responsive design that is appealing to high usability.

We developed web interface by running an angular NPM application on .NET MVC platform. The reason why it is done this way is to make web service calls safer and more stabilized. Figure 4.26 show the general view of the project. An angular app works in Index.cshtml, under ng-app. Index.cshtml works as a view and communicates with Index controller. In the figure, one of the service calls done in Index Controller is shown. This service call is made asynchronously, hence, one service call does not have to wait the other service call.

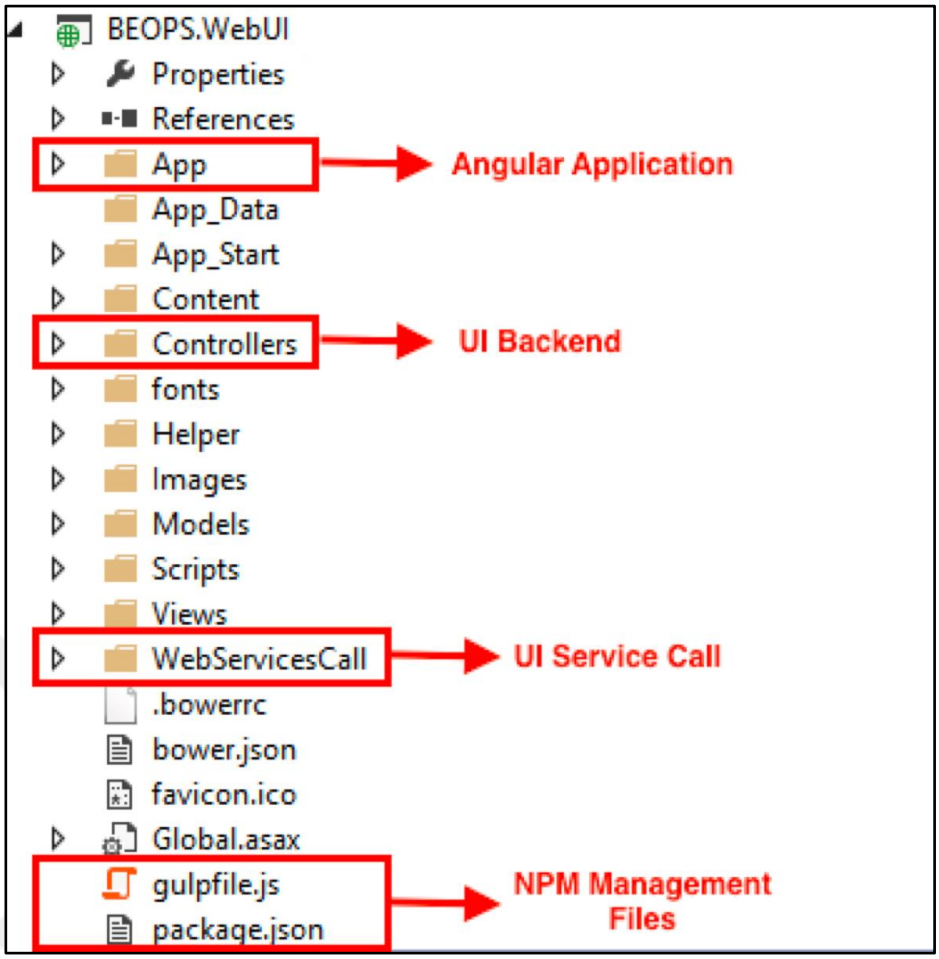

Figure 4.26 Web application project explorer

4.5.1 Service Call From MVC Controller to Web API

We used HttpClient class of .NET to service calls. After the prepared request, we made service call for url with invoke method. Index Controller is solely used for service calls. We added the all web services list in appendix. We determined these service addresses according to the names that help services to go outside. Via these service urls, one can reach to OKURSS' web services. Figure 4.27 shows the http call from client to server code of OKURSS. After creating an http client object, we created the form data of the request to be sent. After filling in the request header, we made the call with *sendasync* method and put response and put a payload. We returned the response model to the relevant model and sent it to the client.

```
public async Task<LoginResponse> PostAsyncLoginMethod(TokenBasedLoginRequest data, string url)
{
    try
    {
        var httpClient = new HttpClient();
        LoginResponse loginResponse = new LoginResponse();
        httpClient.BaseAddress = new Uri(StaticServiceMethods.LoginAuth);
        var request = new HttpRequestMessage(HttpMethod.Post, url);
        var formData = new List<KeyValuePair<string, string>>();
        formData.Add(new KeyValuePair<string, string>("grant_type", "password"));
        formData.Add(new KeyValuePair<string, string>("username", data.username));
        formData.Add(new KeyValuePair<string, string>("password", data.password));
        request.Content = new FormUrlEncodedContent(formData);
        var response = await httpClient.SendAsync(request);
        var payload = JObject.Parse(await response.Content.ReadAsStringAsync());
        var token = payload.Value<string>("access_token");
        if (token == null)
        {
            loginResponse.Message = payload.Value<string>("error_description");
            loginResponse.IsSuccess = false;
            return loginResponse;
        }
        loginResponse.IsSuccess = true;
        loginResponse.AuthenticatedUser = new AuthenticatedUser
        {
            access_token = token
        };
        return loginResponse;
    }
    catch (Exception e)
    {
        return new LoginResponse
        {
            Message = "Şu anda sistemsel bir hata mevcut, en kısa sürede düzeltilecektir.",
            IsSuccess = false
        };
    }
}
```

Figure 4.27 HttpClient Post Method

We implemented the service calls as asynchronously. Because if one service is running while waiting for the other, the user waits for excessive network traffic and this causes timeout errors. The call-back function of each service meets its own service and does not wait for the others. So users can browse the page more quickly and comfortably. Asynchronous execution of a function in C# is provided by the keyword *await*. Thus, a function is not expected to other work. The other codes can resume. Figure 4.28 shows our asynchronous code.

```
2 references
private async Task<T> InvokeAsync<T>(Func<HttpClient, Task<HttpResponseMessage>> operation,
                                     Func<HttpResponseMessage, Task<T>> actionOnResponse = null)
{
    if (operation == null)
        throw new ArgumentNullException(operation.ToString());

    ConfigurateHttpClient(httpClient);

    HttpResponseMessage response = await operation(httpClient).ConfigureAwait(false);

    if (!response.IsSuccessStatusCode)
    {
        var exception = new Exception("Resource server returned an error. StatusCode : {response.StatusCode}");
        exception.Data.Add("StatusCode", response.StatusCode);
        throw exception;
    }
    if (actionOnResponse != null)
    {
        return await actionOnResponse(response).ConfigureAwait(false);
    }
    else
    {
        return default(T);
    }
}
```

Figure 4.28 HttpClient asynchronous invoke method

4.5.1 Frontend Technologies Of OKURSS

We used Angular app running on Index.cshtml. Angular app has its own project index, hence, it works as if it is a different app. Project construction of Angular is shown below. This application is based on MVVM design pattern due to Angular's nature. MVVM is composed of three parts, respectively; Model, View, and ViewModel. Model is made of entity classes that are used to represent the data coming from web services. View is the visual interface in which data are transposed to users. It works as a bridge between user and application. ViewModel, however, works as a bridge between the visual interface and the model, in other words between Model and View. There is no direct interaction between View and Model. View does related operations via ViewModel. View Model has no direct connection to view, and does not know anything about View. Figure 4.29 shows angular MVVM model structure.

Up until this point, JavaScript responsible for functioning is argued. In addition to that visual interface's most important parts include CSS folders. For CSS library, material design which is supported by Google and Angular, Htmls that are mentioned in the page components part, are used for material design.

Material design components are as follows;

- Material Card
- Material Checkbox
- Material Input
- Material Menu
- Material Tabs

It is adequate to write material design the modules that are under supervision of Package.json and App.ts, in order to add material design to the project.

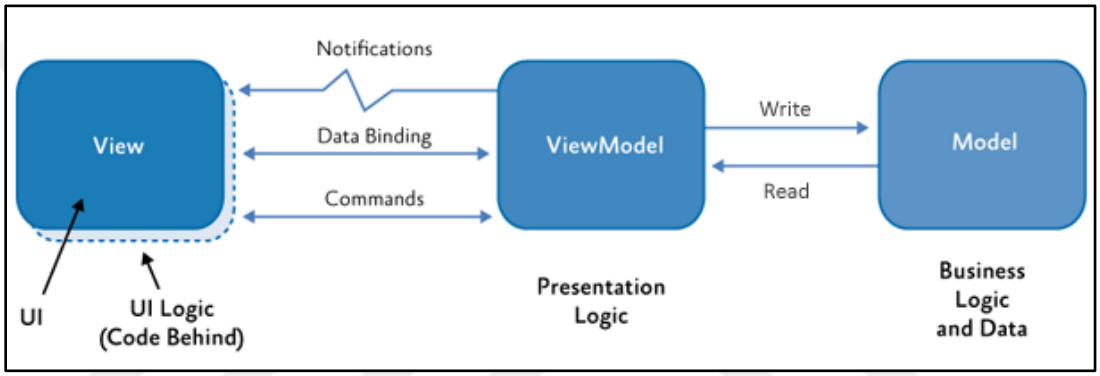

Figure 4.29 Angular MVVM Model (Slideshare, 2013)

Also we used Typescript the reasons concerning modularity and more structured syntax. Typescript is an open source language with object orientation and the aspect of editability which is developed and supported by Microsoft. With the presence of editor in the Typescript, codes can be translated to JavaScript. Addition to Angular and Typescript, we used Bower for packing the project, Gulp for editing and detection of the errors, NPM for administration of the library.

4.5.2 Angular Components Of OKURSS

The nature of the AngularJS and Typescript required that our project be modular. We developed the software pattern by dividing the project into different small modules to disable our project as much as possible from dependency injection, which is one of the solid elements. The frontend project has following components.

4.5.2.1 Directives

Directives are, in a broad sense, expandable html tags. We developed directives for creating new html element by adding new aspects to existing html elements. We, too, make use of directives in our project frequently. We designed special html elements for this project. Since directives can be created once and used everywhere, they eased our job in the project. Directives have folders, one of them is a html folder that encapsulates all the html elements, and the controller that governs the folder. The names given to directives are given in their places. Directive names are given according to this rule; "small lettered initial letter, second word's initial can be capital letter, however one can use "-" to make it small." Figure 4.30 shows all of our directives.

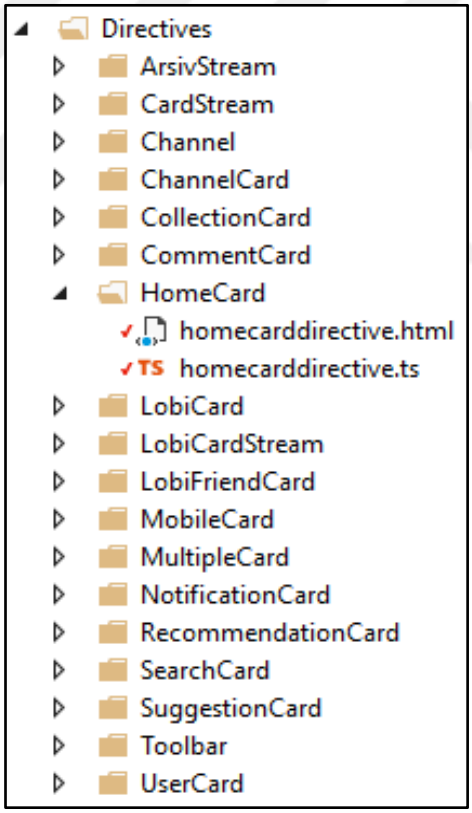

Figure 4.30 OKURSS directives

4.5.2.2 Home Card Directives

We created this directive for represented from article cards in the main screen. Figure 4.31 shows the home card directive. Home Card Directive is the composed of an image, title, description, and action buttons.

Figure 4.31 News card directive

We designed this card as a directive, thus by changing one card, all of the cards will be changed. This card's directive has two files, one being the html folder possessing visual aspects of the card, the other being a typescript folder. Html file is composed of html elements and CSS. Figure 4.32 shows a directive code in detail.

```
<md-card style="height:100%;max-width:500px;">
    <div class="center-cropped">...</div>
    <md-card-title>...</md-card-title>
    <md-card-content>
        <div>...</div>
        <p style="line-height: 1.3;" ng-bind-html="::vm.summary"></p>
    </md-card-content>
    <md-card-actions>...</md-card-actions>
</md-card>
```

Figure 4.32 News card directive code

This html file with angular should of course be controlled. The controller does this. It is one of the most important things in MVC of Angular JS. There is no action or activity in a screen without the controller. Controller can be written both in JavaScript or typescript. We used Typescript to develop directives. Typescript-generated controller has constructor objects, functions, methods or inheritance etc. This is closer to object-oriented JavaScript coding. Figure 4.33 shows a part of HomeCard directive controller.

```
static $inject = ['$location', 'constantService', 'dataService', '$cookieStore', '$timeout', '$mdSidenav', '$mdDialog',

constructor(private $location: ng.ILocationService,
    private constantService: app.common.services.ConstantService,
    private dataService: app.common.services.DataService,
    private $cookieStore: ng.cookies.ICookieStoreService,
    private $timeout: ng.ITimeoutService,
    private $mdSidenav: ng.material.ISidenavService,
    private $mdDialog: ng.material.IDialogService,
    private $mdPanel: ng.material.IPanelService,
    private $scope: ng.IScope) {
    console.log()
}


redirectArticleDetail(): void {

    if (this.clusterid != "0") {
        window.localStorage.setItem("cluserid", this.clusterid);
        this.showDialog('ShowClusteringController', 'App/Views/HomePage/Templates/showclustering.tmpl.html')
    }
    else {
        this.$location.search({ url: this.url, pagenumber: this.pagenumber, stream: this.readstream, channelid: this.cha
        this.readcount += 1
        this.isread = true
        this.showDialog('ArticleDetailDialogController', 'App/Templates/ArticleDetail/articledetail.tmpl.html')
    }
}
```

Figure 4.33 News card directive controller

There needs to be some descriptions, in order to transfer data to a directive dynamically. We passed the dynamic data to directive and made configuration as following in Figure 4.34. “templateUrl” parameter is html file of directive, “controller”

class is the name of the directive class where the directive controls and codes are located. “controllerAs” is alias name of controller class. “scope” is dynamic parameters of directive. “=” operator means the right parameter can be static variable and do not change, “@” operator means the right parameter is dynamic and change with two way bindings property of Angular. “Angular.module” function specifies in which module directive is defined. “Angular.directive” function specifies the name of the directive. The capital letter and “-” rule applies. It comes after - starting with a capital letter. In Figure 4.34 example, “userSuggestionCard” is used as a directive “<user-suggestion-card>”.

```
export class UserSuggestionCardDirective {
    restrict: string = 'E';
    templateUrl: string = 'App/Directives/SuggestionCard/suggestioncarddirective.html';
    controller = UserSuggestionCardDirectiveController;
    controllerAs: string = 'vm';
    bindToController: boolean = true;
    scope: any = {
        'users': '=',
        'followtext': '@',
        'notfollowtext': '@'
    };
}

angular
    .module('BeopsApp')
    .directive('userSuggestionCard', [() => { return new UserSuggestionCardDirective() }])
```

Figure 4.34 Using directives in a controller

4.5.2.3 Templates

We designed dialog boxes as templates, these templates have, as directives, a html and a typescript folder. Figure 4.35 shows a template example about sharing article.

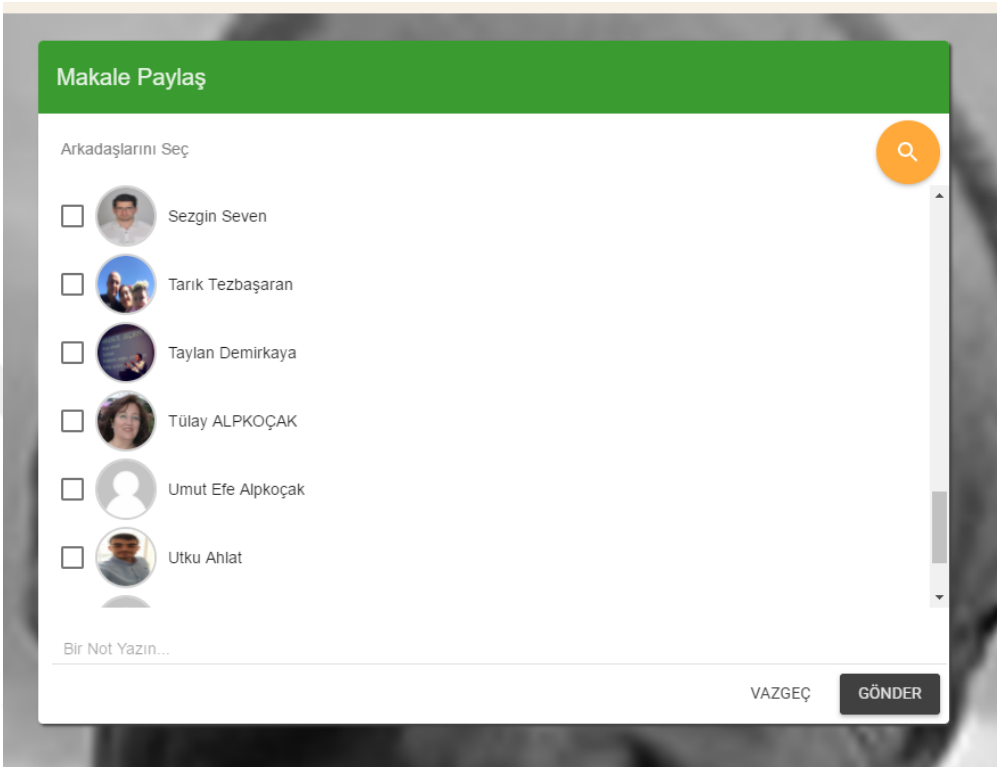

Figure 4.35 Share a news with friends

When this template is shown, html and controller are called, as follows;

```
this.$mdDialog.show({
    controller: 'UserOpinonController as vm',
    templateUrl: 'App/Templates/UserOpinion/useropinion.tmpl.html',
    clickOutsideToClose: true
});
```

Figure 4.36 Open template angular code

4.5.2.4 Services

Services are structures created for common functions in AngularJS projects. We created services for the structures used in our project. In particular, we used it to make post and get calls to Controllers in a Web UI project. We created the get and post

methods which are the basic functions of RestAPI under these services. Figure 4.37 shows the IDataService interface and basic methods.

```
interface IDataService {
    get(resource: string): ng.IPromise<app.domain.EntityBase[]>;
    getSingle(resource: string): ng.IPromise<app.domain.EntityBase>;
    post(resource: string, entity: app.domain.IEntity): ng.IPromise<app.domain.EntityBase>;
    update(resource: string, entity: app.domain.IEntity): ng.IPromise<app.domain.EntityBase>;
    remove(resource: string): ng.IPromise<any>;
}
```

Figure 4.37 Angular http service interface

Figure 4.38, below, shows get and post methods, first parameter represents name of the post method, second parameter is for parameters that are for post requests.

```
post(resource: string, entity: app.domain.IEntity): ng.IPromise<app.domain.EntityBase> {
    var self = this;
    var deferred = self.qService.defer();

    self.httpService.post(resource, entity)
        .then(function (result) {
            deferred.resolve(result.data);
        }, function (error) {
            deferred.reject(error);
        });

    return deferred.promise;
}
```

Figure 4.38 Angular http service post

We represented post methods by their names in the Controller. We listed these methods under the name ConstantService. Thus, when a service is used more than once, it is called from here and managed from a single location. Figure 4.39 show the screenshot of some of these classes.

```
export class ConstantService implements IConstant {
    loginUrl: string;
    facebookLogin: string;
    getStreamUrl: string;
    registerUrl: string;
    addUserAction: string;
    getUserProfile: string;
    uploadImage: string;
    updateUserProfile: string;
    subscribeChannel: string;
    getCollectionsOfUser: string
    addCollectionArticle: string;
    saveUserReaderOptions: string;
    getReaderOptions: string;
    createNewCollection: string;
    getAllCategories: string;
    editArticlesOfCollection: string;
    getArticlesOfCollection: string
    deleteCollection: string;
    getSubscribedChannel: string;
    getDiscoveredChannel: string;
    getSubscribedCollection: string;
    getDiscoveredCollection: string;
    subscribeCollection: string;
    shareArticle: string;
    addUserComment: string;
```

Figure 4.39 Angular http services url constants

4.5.2.5 Views

The views is components that all of the screen. We used all of the directives, templates, and services in views. We created views beforehand, and all the other aspects work for these views. There are two views in this project, one being the main page, the other being user profile. Operations such as, showing news cards, placement of menu, toolbar, listing of the channels and categories are placed in the main page.

Menu is placed at the left, search bar, user information and categories are at the top, channels are at the right. All of these operations can be managed in one screen. Operations happen via JavaScript processes without changing the page.

We designed a view for profile page in main screen. Figure 4.410 shows the screenshot of the profile screen. Besides controlling information about user profile, one can see their self's activity, can update categories, and can see the followers and the followers.

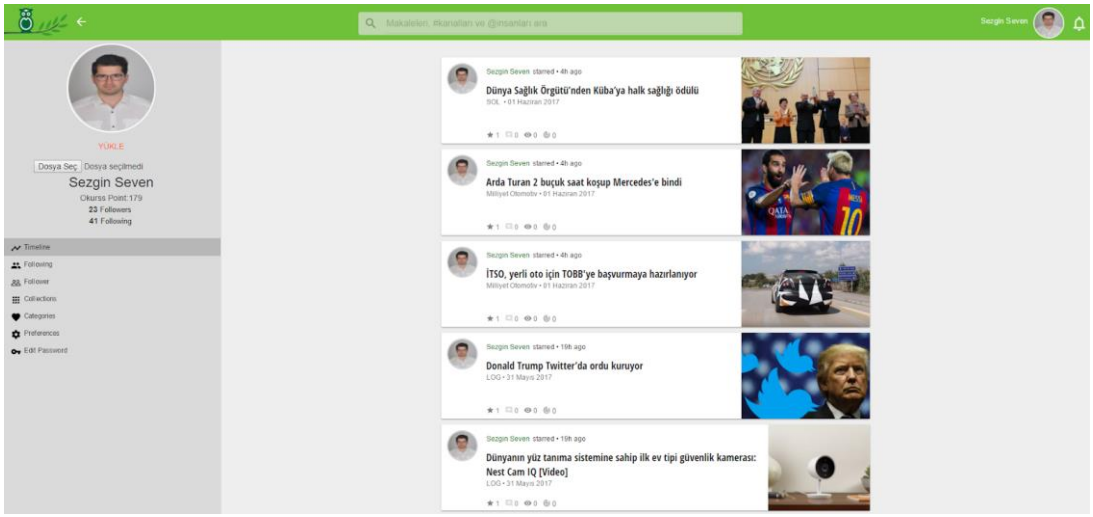

Figure 4.40 OKURSS Profile Page View

4.5.2 Responsive Design

Responsive design is a web design that is responsive to the screen resolutions of computers, tablets and mobile devices and provides a comfortable navigation on every platform regardless of device and resolution. We have developed the OKURSS project in accordance with the principles of responsive design. We have designed designs to be compatible with the web browser of each device. Figure 4.41 shows the mobile platform screen of OKURSS. The responsive reading page is shown in Appendix 3.

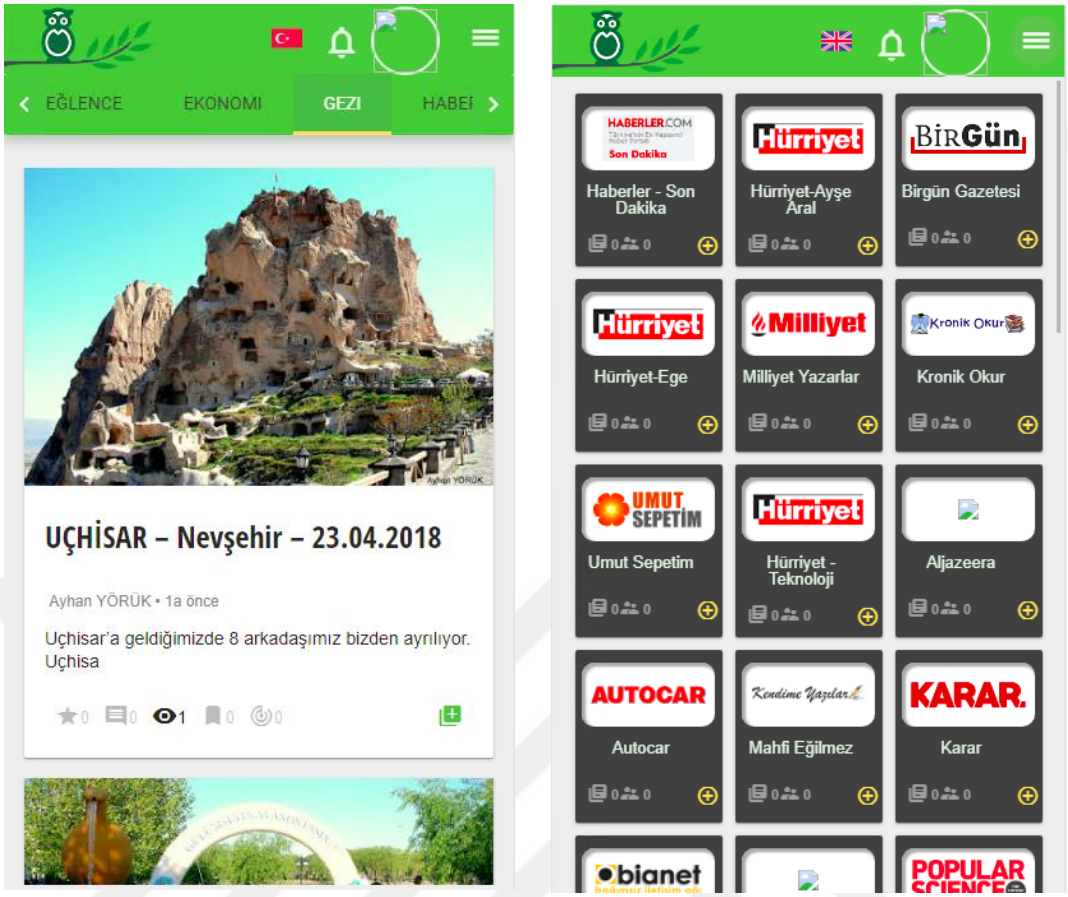

Figure 4.41 Responsive design in mobile

4.6 Deployment

In this section, we explained where and how the deployments of the projects. So far, we mentioned about the the project's technical basis. It is necessary that after the project is set and done, at the client side, user use this service. To accomplish this purpose a remote server in which the deployment folders are is is needed. As OKURSS is a .NET project, the server has to work on Windows, otherwise it won't work. With IIS (Internet Information Service) which provides .NET project's methods, application's working on the server is ensured. After that, a domain name, in this case www.okurss.com/wservice, we defined. Every call to be made to this domain will be directed to the remote server via SSH connection, which is a secure protocol. Application starts working thanks to this directed connection. Figure 4.42 shows the deployment files and environments.

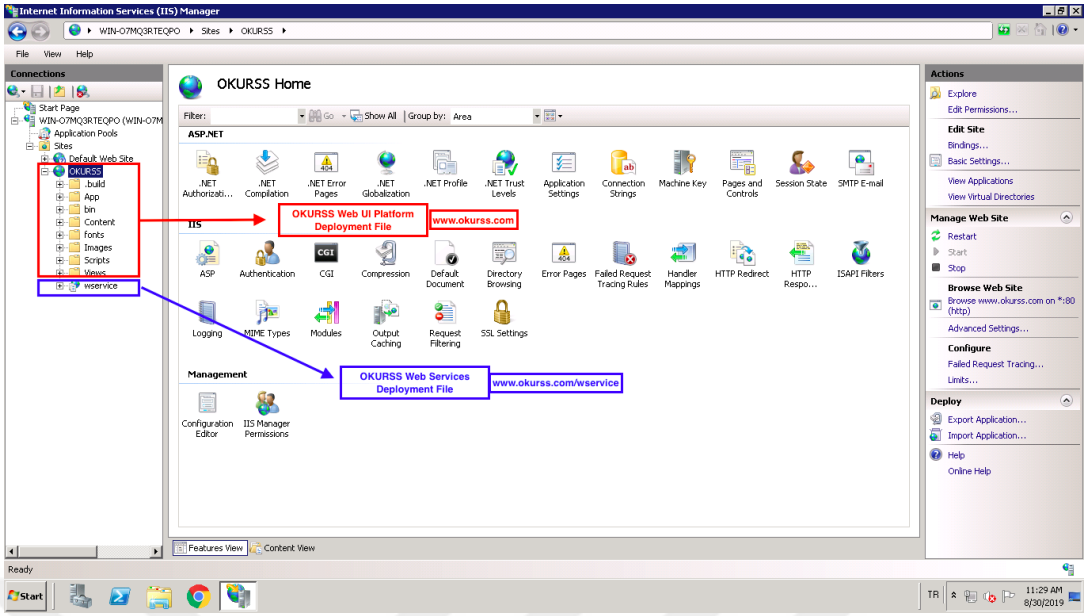

Figure 4.42 Deployment to IIS

CHAPTER FIVE
EXPERIMENTATIONS

In this chapter, we shared and explained recommendation system algorithm results, system performances, user data results. The experiment has been conducted with users which get recommendations of certain news in certain categories with recommendation system. Every day, these users read the news primarily recommended to them. Table 5.1 show the most read news categories of 6 selected users in one month. According to the scenario we have identified, Adil and Emre, Okan and Sezgin, Görkem and Eren read the most similar news categories. Figure 5.1 shows the most recommended news categories on a daily basis. These users are created according to specific scenarios. 6 users read similar news every day in groups of 2 people. The purpose of this scenario is to see if the recommendation system works correctly and whether similar users are recommending the similar news. With this scenario, the collaborative filtering algorithm has been run and similar users have been identified in the beginning. Table 5.2 shows the result of collaborative filtering.

Table 5.1 News reading database records for one month

UserName character varying (100)	**Category** character varying (20)	**max** bigint
Adil	Haber	94
Emre	Haber	84
Eren	Spor	255
Görkem	Spor	249
Okan	Teknoloji	17
Sezgin	Teknoloji	15

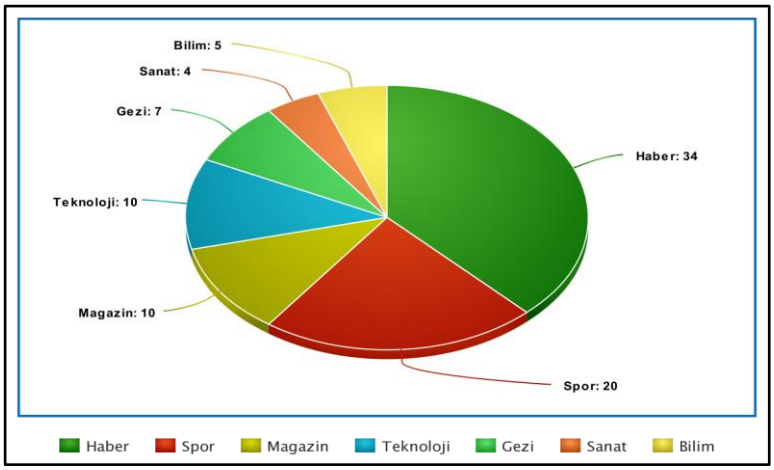

Figure 5.1 News recommendation distribution by category

Data in Table 5.1 is high importance for methods of recommendation system algorithms, as the news users have read before are significant for content based recommendation systems. Similar news will be recommended to users having similar reading preferences, by investigating similar news according to the content based filtering, at the same time. The values, shown with the name pairs below, are for the calculating the matches, which means that out of ten, higher scores indicate higher similarity rate between the users

Table 5.2 Matched users with collaborative filtering

Matched User	Matched Score (max scrore : 10)
Sezgin - Okan	6.06
Adil - Emre	5.31
Eren - Görkem	6.12

We tested the algorithm that compares the similarities between news contents by news tracking system in OKURSS. In the news-tracker-screen in OKURSS, similar news to the news that users want to read are brought to the user. To track the news, one has to click to the rightmost button in the left part of the screen. Thus, in addition to the algorithm that makes similarity comparison, we tested search process and system performance. Accordingly, the tracking system brought about similar news within 4.2 seconds. Figure 5.2 shows the screen that including similar news of tracking system.

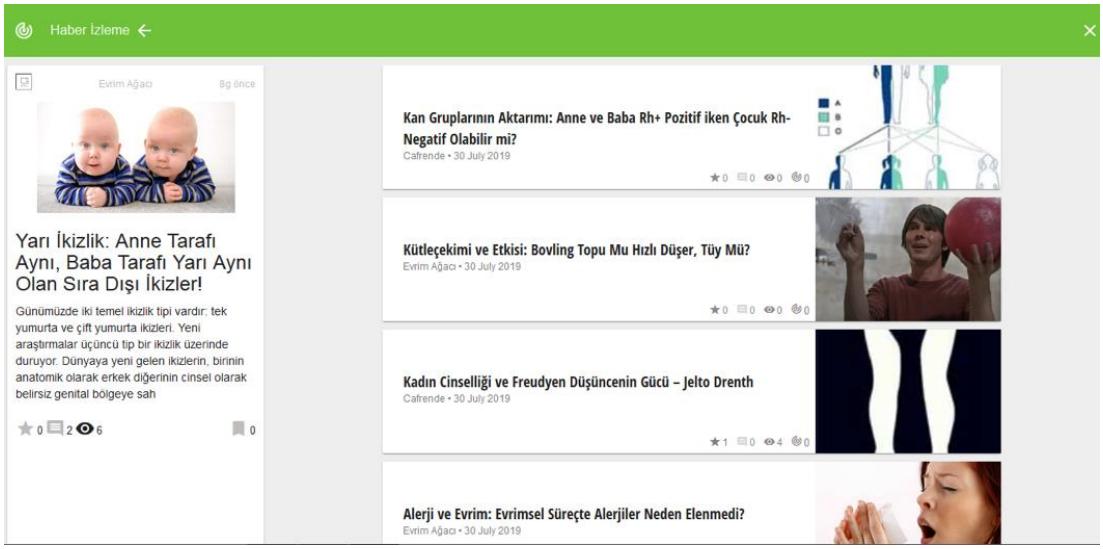

Figure 5.2 News tracking for evaluate document similarity

Collecting news from RSS channels, html parser and recommender engine are batch programs that run consecutively. Each of these applications trigger each other after operation and operates in sequence. If an error occurs in one of these batch operations, the operation of the others does not interfere. If necessary, the system will continue to work.

According to the keyword based recommendation system, Users of OKURSS submit the keywords they desire to the system. When the recommendation system works, these keywords are concerned. Figure 5.4 shows screenshot of news that is keyword based recommended news card. Keywords are pointed out with a red frame.

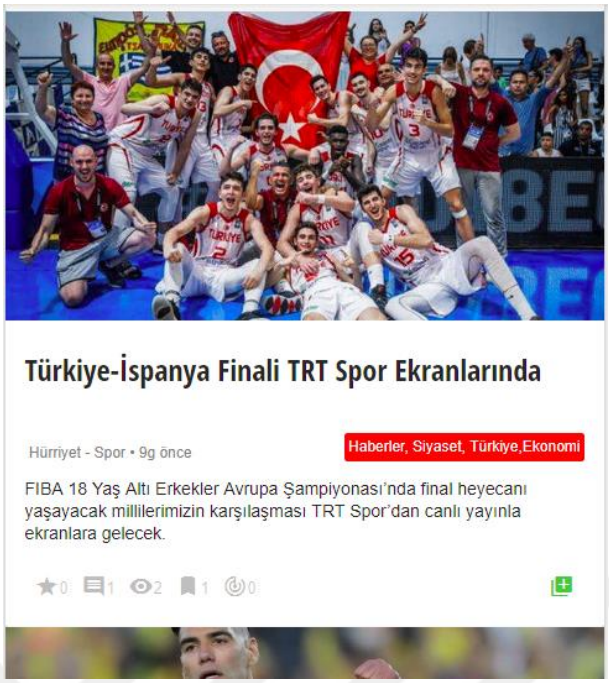

Figure 5.4 Keyword based recommendation news card

CHAPTER FOUR
CONCLUSION

In this thesis, we presented how the recommendation systems are customized for users and news. We used traditional solutions of recommendation systems and have expanded and applied these methods. We tested each method separately. We proceeded with feedback from users in order to conduct tests of recommendation systems in a healthy way. First, we determined the categories of news to be read on a daily basis. In these identified categories, users read a certain number of news on a daily basis. We also directed these users to follow each other in the system. At the end of this scenario, the categories of news that users read and interested were revealed. When the recommendation system we developed based on the data generated as a result of this study, the proposed news became news in the categories assigned to the users. This result tested the accuracy of the content-based recommendation system. As a result of recommending the similar news to similar users identified in the first scenario, it showed the accuracy of collaborative filtering. Entering keywords from users' profiles and recommendation news based on these keywords showed a result of the keyword-based recommendation system. We also used the news watch screen to show similarities between documents. The more similar the news on the left side of the screen and the news listed on the right side, the more similar the document similarity algorithm worked.

Recommendation systems can provide important information, although not as common as news and written content. Reading habits can show their interest in many other things and personality analyses can be done accordingly. At the same time, this system also contributes to readers. Providing people with articles they will like will increase their reading habits. In future studies, by improving the algorithm of personal recommendation system and making smarter suggestions, OKURSS will be the most important and real news source where all news is gathered and published in one place.

REFERENCES

Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., & Kochut, K. (2017). A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*. Retrieved January 20, 2019 from https://arxiv.org/abs/1707.02919.

Başak, S. (2009). *Türkçe dokümanların benzerliği*. Retrieved May 12, 2019 from https://www.selcukbasak.com/download/TurkceDokumanBenzerligi.pdf.

Chandramouli, B., Levandoski, J. J., Eldawy, A., & Mokbel, M. F. (2011). StreamRec: a real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 1243-1246.

Doychev, D., Lawlor, A., Rafter, R., & Smyth, B. (2014). An analysis of recommender algorithms for online news. In *CLEF 2014 Conference and Labs of the Evaluation Forum: Information Access Evaluation Meets Multilinguality, Multimodality and Interaction, 15-18 September 2014, Sheffield, United Kingdom*, 177-184.

Hofmann, T. (2000). Learning the similarity of documents: An information-geometric approach to document retrieval and categorization. In *Advances in Neural Information Processing Systems*, 914-920.

Kompan, M., & Bieliková, M. (2010). Content-based news recommendation. In *International Conference on Electronic Commerce and Web Technologies*, 61-72.

Kotian, D. (2019). *Securing ASP.NET Web API using Custom Token Based Authentication*. Retrieved May 10, 2019 from https://www.ecanarys.com/Blogs/ArticleID/ 308/Token-Based-Authentication-for-Web-APIs

Liu, J., Dolan, P., & Pedersen, E. R. (2010). Personalized news recommendation based on click behaviour. In *Proceedings of the 15th International Conference on Intelligent User Interfaces,* 31-40.

Luostarinen, T., & Kohonen, O. (2013). Using topic models in content-based news recommender systems. In *Proceedings of the 19th Nordic Conference of Computational Linguistics,* 239-251.

Mukti, S. (2018). *When to use PostgreSQL Full Text Search and trigram indexes.* Retrieved January 21, 2019 from https://www.medium.com/@sukorenomw/when-to-use-postgresql-full-text-search-and-trigram-indexes-4c7c36223421.

Newcomb, D. (2018). *Automated company keyword extraction*. Retrieved June 15, 2018 from https://blogs.oracle.com/ai/automated-company-keyword-extraction

Özgöbek, Ö., & Erdur, R. C. (2015). Öneri sistemleri ve bir uygulama alanı olarak haber öneri sistemleri. *Akademik Bilişim Konferansları*, Eskişehir. Retrieved January 20, 2019 from https://ab.org.tr/ab15/kitap/120.pdf.

Özgöbek, Ö., Gulla, J. A., & Erdur, R. C. (2014). A Survey on Challenges and Methods in News Recommendation. In *WEBIST*, 278-285.

Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web,* 325-341.

Saranya, K. G., & Sadhasivam, G. S. (2012). A personalized online news recommendation system. *International Journal of Computer Applications*, *57*(18), 975-984.

Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In *The Adaptive Web,* 291-324.

Seven S. & Dost N. (2016). *Mobile application and web crawler for blog reading,* Undergraduate Thesis, Dokuz Eylul University Engineering Faculty Department of Computer Engineering, İzmir.

Solieman, T. (2009*).* *Introduction of angular js.* Retrieved July,10 2017 from https://www.slideshare.net/tamersolieman/introduction-of-angular-js

Suchal, J. & Návrat, P. (2010). Full text search engine as scalable k-nearest neighbor recommendation system. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, 165-173.

Taşçı, S. (2015). *İçerik bazlı medya takip ve haber tavsiye sistemi*. Graduate Thesis, Hacettepe University, Ankara.

Uzun, Y. (2005). *Keyword extraction using naive bayes*. Retrieved May 12, 2019 from https://www. cs. bilkent. edu. tr/~ guvenir/courses/CS550/Workshop/Yasin_Uzun. pdf.

Xia, Z., Xu, S., Liu, N., & Zhao, Z. (2014). Hot news recommendation system from heterogeneous websites based on bayesian model. *The Scientific World Journal*, 2014.Retrieved January 20, 2019 from https://www.hindawi.com/journals/tswj/2014/734351/abs/.

Yong, Z., Youwen, L., & Shixiong, X. (2009). An improved KNN text classification algorithm based on clustering. *Journal of Computers*, *4*(3), 230-237.

APPENDICES

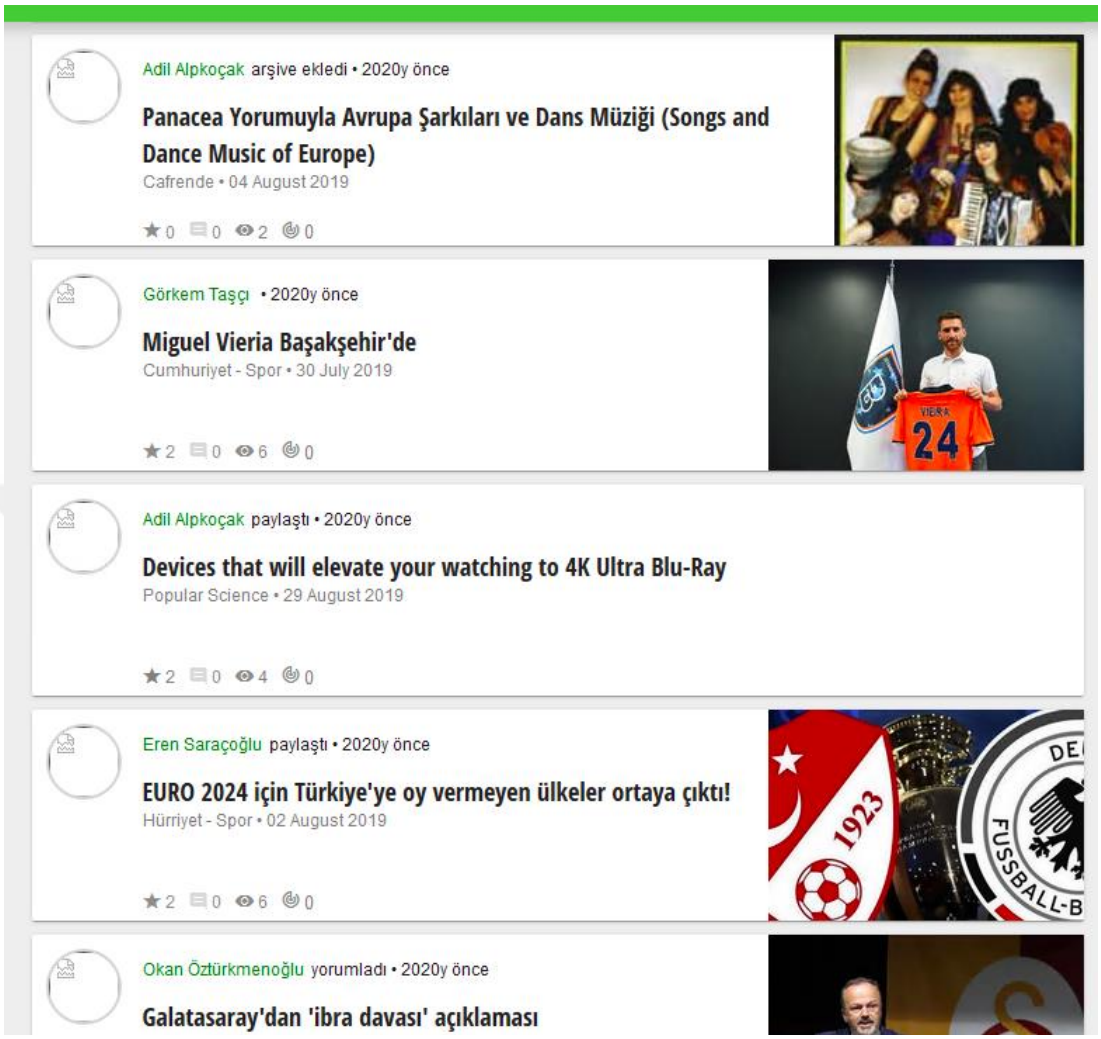

Figure A.1 Lobby screen

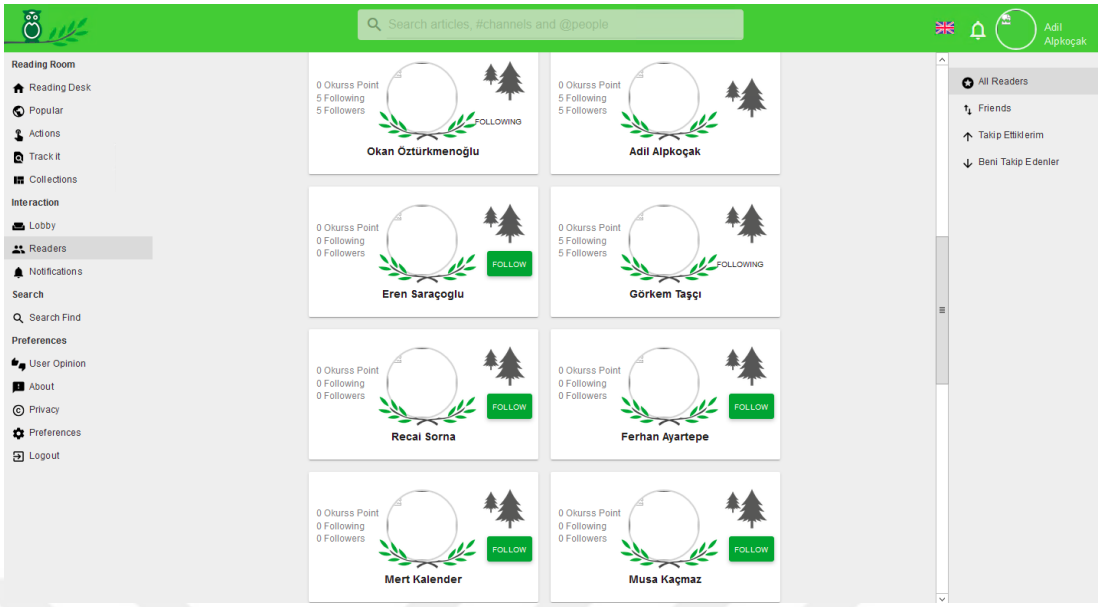

Figure A.2 Reader list screen

Figure A.3 Responsive reading screen