BACHELOR THESIS

Ladislav Maleček

Books Recommender System via Linked Open Data

Department of Software Engineering

Supervisor of the bachelor thesis: Mgr. Ladislav Peška, Ph.D.
Study programme: Informatics
Study branch: Software and Data Engineering
I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .......... date ............ signature of the author
Title: Books Recommender System via Linked Open Data

Author: Ladislav Maleček

Department: Department of Software Engineering

Supervisor: Mgr. Ladislav Peška, Ph.D., department

Abstract: This thesis focuses on using recommender system’s methods on Linked Open Data in a domain of books. After thorough analysis of multiple available Linked Open Data sets, we have concluded that data sets of sufficient size and quality already exist. Together with careful analysis of the structure and quality of the data, recommender system web application has been developed based on retrieved data from a Wikidata endpoint. The application design allows an incorporation of data from multiple sources. A novel approach for generating recommendations utilizing multi language tags extracted from Wikipedia was used. We have shown that it is possible and viable to use Recommender systems on top of the Linked Open Data, but the common recommender system’s algorithms have to be modified in order to deal with a huge amount of sparsity in the data.

Keywords: Linked Open Data, recommender systems, books, website
I hereby dedicate this work to my family; if it weren’t for them, I never could be where I am today.

I would also like to thank my supervisor Mgr. Ladislav Peška, Ph.D. for his time, shared feedback and knowledge.
# Contents

**Introduction** 3

1 Data 4
   1.1 Linked Open data (LOD) 4
      1.1.1 Web 1.0 5
      1.1.2 Web 2.0 5
      1.1.3 Web 3.0 5
      1.1.4 Semantic web 5
      1.1.5 RDF Format 6
      1.1.6 SPARQL query language 6
   1.2 Analysis of available LOD endpoints 7
      1.2.1 WikiData endpoint 7
      1.2.2 DBpedia endpoint 8
      1.2.3 Other endpoints 9
   1.3 Wikidata endpoint analysis 11
      1.3.1 Structure of data 11
      1.3.2 Wikipedia information 11

2 Recommender systems 14
   2.1 Introduction to basic concepts 14
      2.1.1 Collaborative recommendations 14
      2.1.2 Content-based recommendations 15
      2.1.3 Knowledge-based recommendations 16
      2.1.4 Hybrid recommendation approaches 17
   2.2 Techniques used in our system 17
      2.2.1 Tag extraction using TF-IDF 17
      2.2.2 Similarity search algorithm 19
      2.2.3 Obtaining preferences about users 19
   2.3 Proposed recommender system 20
      2.3.1 Content-based system based on LOD properties 20
      2.3.2 Content-based system based on tags from Wikipedia 20
      2.3.3 User-based collaborative filtering system 21

3 Software design 22
   3.1 Structure of the application 22
   3.2 Database 23
   3.3 Server 25
   3.4 Client 25
   3.5 Security 25

4 User documentation 28
   4.1 Homepage 28
   4.2 Register and login 28
   4.3 Searching for books and authors 30
   4.4 Detail of books and authors 31
4.5 Feedback and user profile ........................................... 31
4.6 Requirements ......................................................... 31

5 Administrators documentation ........................................ 33
  5.1 Server requirements .................................................. 33
  5.2 Deploying the application ............................................ 33
    5.2.1 Building the application ....................................... 34
    5.2.2 Configuring app settings and creating database ............. 34
    5.2.3 Nginx ............................................................ 35
    5.2.4 Our application as a service ................................... 36
    5.2.5 Securing our application ...................................... 37
    5.2.6 Possible problems ............................................... 37
  5.3 Mining data from the application’s interface ...................... 38

Conclusion ........................................................................ 39

Bibliography .................................................................... 40

Attachments ..................................................................... 42
Introduction

Open Data and Recommender systems

We can observe very substantial increase in the amount of data on the Web in the recent years. One of the ways people can make this data useful is through machine readable annotations. Then even the biggest data sets can be analyzed and utilized with ease using a computer. A standard for annotating these data was created by W3 group. If we use this format together with publishing the data to be freely available to everyone, then we call these data sets Linked Open Data (hereinafter LOD). Many data sets contains huge amount of encyclopedic knowledge. Such as Wikidata, or DBpedia.

One of the areas we could make these data sets useful is in computer programs that will recommend items to the users based on their preferences. We call these programs recommender systems (hereinafter RS). They can provide a lot of interesting and useful information from the data. They are utilized in a variety of areas including books, movies, music, social networks, but even in research articles and highly specialized commercial environments.

Up until now, there is not much research work done in the combining RS with LOD. Even though the data sets with interesting and usable information are growing in size rapidly. Application that would use LOD data as a base for RS in domain of books haven’t been created yet, even though the data is available. We would like to analyze the open linked datasets about this domain and use one of them as a background for a web based book recommender system. The system will automatically use new data that will be added to the data points, by the creators, community and more powerful information extraction.

A web application based on this data has been created and is available at https://recommender.projekty.ms.mff.cuni.cz

Aim of the work

The aim of this work is to create a web application that would provide its users with relevant book recommendations based on user’s feedback and provided reading history. The application will use only data that can be acquired freely from LOD datasets. The application should be responsive, simple in design and user friendly. Part of the work is an analysis of used the LOD dataset. The problem that we can see at a first glance is the distribution of applicable knowledge in the datasets. Some data points have data about books with a lot of information about items, but some with not much or even any at all. We should carefully analyze and then adapt the recommending methods that will be used.

The main challenge of this work is to extract the data about books using the SPARQL query language, interpret them in a nice form to the users and then use retrieved data to recommend relevant, similar and interesting books. Another important problem is that we can expect the cold start problem because of the size of the user base that cannot be expected to be sufficient for the user based methods at the school work project size.
1. Data

In this chapter, we will explore what is Linked Open Data, its properties and whether it is suitable to be used together with recommender systems.

1.1 Linked Open data (LOD)

In computer science, linked data is a method of publishing structured data so that they can be interlinked and become more useful through semantic queries. It builds upon standard Web technologies such as HTTP, RDF and URI, but rather than using them to serve web pages for human readers, it extends them to share information in a way that can be automatically read by computers. This enables data from different sources to be connected and queried as one huge source of knowledge (see Figure 1.1).

Linked Open Data (LOD) is Linked Data which is released under an open license, which does not impede its reuse for free.

— Tim Berners-Lee

![Figure 1.1: Shows the biggest LOD datasets as of 2014, every single bubble is one data endpoint, and every arrow is interconnection that binds the endpoints together.](image)

---

1 often capitalized as Linked Data

2 Hypertext Transfer Protocol - [https://www.w3.org/Protocols/](https://www.w3.org/Protocols/)

3 Resource Description Framework - [https://www.w3.org/RDF/](https://www.w3.org/RDF/)

4 Uniform Resource Identifier - [https://www.w3.org/Addressing/](https://www.w3.org/Addressing/)
1.1.1 Web 1.0

The web as we know it today is not the same as at the birth of the world wide web era. It all started as only static web pages that only provided their users with static information. The user was only a visitor viewing the pages. They could only interact with the web by traversing between pages using page links. No direct communication between users or with the owner of the website was available. This type of web is now known as the Web 1.0. Examples of the Web 1.0 are common simple static pages such as personal and business landing pages or web catalogs.

1.1.2 Web 2.0

As the size of the internet grew and the technology progressed, there was an increasing need to provide users with more functionality as an edge over the competition. Around the year 2004 the term the Web 2.0 was used for the first time with the birth of blogs, social networks and other web pages that were based on dynamic content. This type of web pages considers their users as creators, side by side with the content from the web owner. Users can share data, pictures, and experiences among themselves as well as interact and create their own content. It is all about the social networking and community. And that is the absolute opposite of the web of the first generation where the information relationship between users and the web was only one-sided and passive from user’s point-of-view. Examples of Web 2.0 are blogs, forums and social networks such as Facebook\(^5\) Twitter\(^6\) Reddit\(^7\) YouTube\(^8\) and much more.

1.1.3 Web 3.0

As the World Wide Web is increasing in size and traffic every year even more, there is an unavoidable need for processing data automatically. The term Web 3.0 is not entirely clear in its meaning yet. But one of the views is that it is just another expression for semantic web. Or that semantic web is one of many technologies that will define the Web 3.0. Together with enhancements of web 2.0 such as a much wider use of video, partially artificially intelligent web or shared web applications like Google Docs\(^9\) and others. Though, the definitions vary.

1.1.4 Semantic web

We view the semantic web as an extension of the Web in the way that it is possible to be processed by machines. The main idea is that data could and we would want it to be used repeatedly, across multiple applications and shared between users. The term “Semantic web” was heavily supported by the “father” of the Web Mr. Tim Berners-Lee\(^10\) He pioneered the idea at [2].

\(^{5}\)https://www.facebook.com/
\(^{6}\)https://twitter.com/
\(^{7}\)https://www.reddit.com/
\(^{8}\)https://www.youtube.com/
\(^{9}\)https://www.google.com/docs/about/
\(^{10}\)https://www.w3.org/People/Berners-Lee/
1.1.5 RDF Format

The most fundamental technology of the semantic web is Resource Description Framework. RDF is a standard model for data interchange on the web. RDF extends the Web using URI. In the semantic web, URI uniquely describes a real thing, a property or some other web resource. We can describe relationships between things using “triplets” that consists of three URIs. They have a form of subject-predicate-object expressions. For example, we can describe expression “Hobbit was written by J.R.R. Tolkien” as the following:

- “Hobbit” - is subject of the expression
- “Was written by” - is a predicate that represents the relation between subject and object
- “J.R.R Tolkien” - is an object of this expression

In a real world example, the triplet would look like this - example from Wikidata source:

- http://www.wikidata.org/entity/Q74287
- http://www.wikidata.org/prop/direct/P50
- http://www.wikidata.org/entity/Q892

RDF can support not only triplets but also quads. They are triplets extended for use between multiple sources. Triplet can have not only URI as an object but also other text data such as XML, JSON or simple text data. For example, a text label or a web address pointing to an image file.

So, RDF describes objects and their relationships with other objects in the same or other data point. The data that naturally forms an oriented graph can be represented using many formats. One of them is a line-based format called N-Triples, which is used by most RDF databases. We can also use a nice human readable format called Turtle and many more.

1.1.6 SPARQL query language

The SPARQL query language is defined as a part of RDF standard and it is used to query data in RDF format. It is basically an SQL analogy for the RDF format. The queries can be loosely compared to “key-value” databases queries based on the similarity with the triplets’ form of the data. It can be analogous to some NoSQL databases such as MongoDB.

---

11 Extensible Markup Language - https://www.w3.org/XML/
12 JavaScript Object Notation - http://www.json.org/
13 https://www.w3.org/TR/n-triples/
14 https://www.w3.org/TR/turtle/
15 SPARQL Protocol and RDF Query Language - https://www.w3.org/TR/rdf-sparql-query/
16 Structured Query Language - https://www.britannica.com/technology/SQL
17 Originally referring to “non SQL”, “non relational” or “not only SQL
18 https://www.mongodb.com
SPARQL provides a full set of standard mathematical operations like JOIN, SORT or AGGREGATE that are also found in regular database software. And it is the only standard for retrieving RDF data. Most of the publicly available datasets provide endpoint query service using simple HTTP protocol. We must specify the query as a part of the HTTP request. Some endpoints also provide an interactive web application which can be used to easily create and test queries. One of them is [www.query.wikidata.org](http://www.query.wikidata.org).

The following query returns all people from database together with their emails:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE
{
    ?person a foaf:Person .
    ?person foaf:name ?name .
}
```

In this example, every variable starts with a question mark. PREFIX is used to shorten URIs inside of the query’s body. So, foaf:Person is just an abbreviation for “http://xmlns.com/foaf/0.1/Person”. Inside of the WHERE block there are triplets ended with a dot or semicolon. Dot ends the line and semicolon continues to describe the subject so that it’s not necessary to be repeated on the second line. Any single URI in the triplet can be substituted for a variable. And if the variable is mentioned multiple times it needs to satisfy all conditions at once.

### 1.2 Analysis of available LOD endpoints

We are going to explore multiple public data endpoints and then chose one with the most information for our needs, respectively the endpoint with most books available.

#### 1.2.1 WikiData endpoint

The Wikidata endpoint is a part of the Wikipedia movement towards the semantic web. It stores all information that is available to users at many Wikipedia pages as an infobox information as mentioned at [3]. It is available at [https://query.wikidata.org/](https://query.wikidata.org/) and it supports HTML queries as well providing an interactive web application endpoint. Specification about HTML query standard can be found at documentation page [20]. In the dataset, there are around 85 000 books and around 35 000 authors [21].

---

[1] [https://www.wikidata.org/](https://www.wikidata.org/)


[21] As of the time of writing April 2017
1.2.2 DBpedia endpoint

The DBpedia endpoint is the biggest LOD dataset publicly available. Tim Berners-Lee described DBpedia as one of the most famous parts of the decentralized Linked Data effort. As mentioned at [1], it is based on Wikipedia and it tries to extract as much data from the Wikipedia pages as possible. It provides its users with a HTML based API and with an interactive web endpoint application, can be found here: https://dbpedia.org/sparql. Even though the application is not as nice and interactive as the one provided by Wikidata. The dataset contains around 35 000 books and 14 500 authors.

Upon closer inspection, there could be more books extracted from the data set, but it would be very difficult. Some books are not labeled as books but as some subcategory of a book. But there are also many non-book items that would be retrieved this way. So, the data would not be as clean as when retrieving using only the rdf:type “Book” type object relationship. The reason for this is that the data on DBpedia come from the mining of Wikipedia pages, there is not much, if any human work behind the categorization. For example (as we can see on Figure 1.2), we have a record about book Harry Potter and the Philosopher’s Stone but it is not rdf:type of Book but of a rdf:type Thing. There are multiple clues as how to be able to recognize if it is a book. For example, (as shown below) it is listed as ”British novels adapted into films”, and we could traverse the LOD graph from that to the rdf:type of ”Book” through multiple graphs jumps to the concept of Literature(as we can see from Figure 1.3). But this traversing is very computationally intensive and it is not error-free by far. When starting the search from the concept of Literature there are also data such as Creative writing programs, this concept is not about books, but about classes and courses where one can improve their writing skills. Distinguishing between valid and false concepts found in this way is very difficult.

![Figure 1.2: Harry Potter and the Philosopher’s Stone entry in DBpedia.](https://dbpedia.org/page/Harry_Potter_and_the_Philosopher's_Stone)
1.2.3 Other endpoints

The semantic web is increasing in size to the extent that it is almost impossible to search for every possible dataset and compare between them. It is also safe to say that the two mentioned data sets are possibly the biggest ones publicly available and if there is a bigger one regarding books, it will be too country or area specific for our use. For example, some book libraries could provide their book databases as an LOD.

One of the libraries is “British National Bibliography as a Linked Open Data”[^26]. It contains over 3.9 million records but it is country specific to the United Kingdom and the Republic of Ireland. That means it lacks many of the greatest books and lacks the variety we would like to provide. Also, it doesn’t have as much content attributes as we can see the comparison with Wikidata data point on Figure 1.4 and Figure 1.5.

[^26]: [http://bnb.data.bl.uk/](http://bnb.data.bl.uk/)
It is also little too big for the purpose of this work, we strive in the first place to implement an interesting data set, not the biggest one. The application will be designed in a way that an addition of another endpoint is possible and simple. So one can, for example, add the aforementioned British National Bibliography to the system and increase the English book domain substantially.

The choice of an optimal data set is not the main target of this work. The target is to pioneer the idea of using a book recommender system on top of an LOD in the book domain. We can see from the Figure 1.6 that the data set provided by Wikidata is bigger than that provided by DBpedia. Therefore we have chosen the Wikidata endpoint as it is a data set that is easy to explore and sufficient in size.

Figure 1.5: British National Bibliography endpoint record for the first Harry Potter book.

Figure 1.6: Size comparison between DBpedia and Wikidata endpoints
1.3 Wikidata endpoint analysis

We have chosen the Wikidata endpoint because it offers the most book related information from the explored datasets. Also, it has a nice web interface for query building and testing. In this chapter, we will look at the data more in depth and analyze them from the point-of-view of information that could be presentable to users as well as from the perspective of a recommender system.27

1.3.1 Structure of data

The LOD data is from the essence of its design in graph form, but we would like to simplify and normalize it as much as possible. For the ease of storing it and querying over it efficiently. We don’t want to represent all entities with an n:m relationship. This approach would be too costly for the recommendation and database system. On the following diagram (Figure 1.7), we can see the relationships of the data, we have displayed only the most important ones for books and authors. The lighter elements represent concepts that will be later saved to a database as separate tables. The darker blue represents textual data.

Figure 1.7: Diagram representing the most important relationships between our LOD data in Wikidata endpoint

1.3.2 Wikipedia information

Almost every book and author have a Wikipedia page link with the data. The Wikidata dataset is closely related to Wikipedia itself. We would like to mine the articles and use them as a tag based recommendation system. For this, we will use a simple page download and trim mechanism. Before processing the page

27All data statistics are up-to-date as of the time of writing 4.2017
further, we will save it to a local storage. The download process takes a long time. For the sake of simplicity, we will assume that the Wiki Pages themselves do not change too often and we will not deal with refreshing the locally saved wiki pages\textsuperscript{28} when updating the local recommender’s data. We have chosen the method of saving the downloaded data in \textit{one simple operating text file per page} format.

We have not found any interface of Wikipedia that would provide only the clean text information. So, we will download the Wiki page using a simple HTTP request and appending “?action=raw” to the URL of the page. The downloaded data contains multiple simple markup-like information annotations for example “Ref” statements or curly braces for additional machine readable information. We want to trim the page of this information so that we are left only with clean text. Mining of the tags will be described in \textit{chapter 2}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1_8.png}
\caption{We can see the the total size of data when dividing languages by their size(number of Wiki pages).}
\end{figure}

In the graph (see Figure 1.8), we can see the number of downloaded wiki pages for languages divided into language groups by their size. For example, in the first category, we can see that languages with more than 10000 wiki pages make up about one-third of the total available information. From the graph, we can also see that languages with less than 100 downloaded wiki pages are in the majority, but the amount of information is small. For this reason, we are going to remove them from the recommender data.

The graph (see Figure 1.9) represents what happens to our data when we remove languages that have less than specified number of wiki pages. We can

\textsuperscript{28}Also as Wikipedia page
Clearly see that when we remove languages from our data with less than 100 entries (191 languages) the data is not affected much. When removing languages with less than 1000 entries, there is a slight increase in the count of books without any wiki page. That means that there is a decrease in the coverage of tag based recommendations on those particular books. We want to find a good balance between the number of languages that enters our RS and between the domain size of the language. For example, when we have a language with only 20 books covered, there is only a small chance that the book tags acquired from wiki pages will match and if they do there is almost no variety in the result - we are comparing only 20 books.

Based on the analysis from the previous line graph, we have chosen to remove languages below the size of 100 and penalize languages with a size of 100-1000 so that they are taken into account only when there is a substantial match found. The 1000 limit is on the border where we can see that the number of books with no wiki page is starting to rise. We want to avoid that as much as possible.

Figure 1.9: Plot which shows how the count of Wikipedia pages for each book changes when not including languages with a certain book domain size.
2. Recommender systems

In this chapter, we will introduce recommender systems and we will provide a classification of the most popular algorithms into groups and evaluate them by their advantages and disadvantages. Then we will discuss design choices made to the proposed recommender system tailored to our specific needs of linked data and book domain.

2.1 Introduction to basic concepts

RS (recommender system) is an information filtering algorithm that seeks to make the best prediction for its users about whether they will like a certain item. Or to recommend the best set of items in the domain. Over the last few years, recommender systems have become increasingly popular in a variety of areas including books, movies, search queries and other domains where it is not possible or easy to search through all items and pick the best one manually. Nowadays we can find RS in almost all specialized domains one can think of.

For example, if we have an online bookstore, we as the owner of the store would like to offer our customers the best book for their reading taste. If we can recommend the right book, there is much bigger chance that the customer will consider to purchase a book. And that, together with customer satisfaction, is the main economical pressure that leads to the development and deployment of RS in a commercial environment. We will discuss the main types of recommender systems as well as their advantages and disadvantages. After that, we will look closely at the interactions between users and RS together with methods as how to gather relevant information about our users.

2.1.1 Collaborative recommendations

The basis of this idea as the name implies is collaboration. We can assume that if you like the same items as another user, then there is a high probability that you will also like the items he does and you haven’t seen yet. We can say, that you share the same interests.

For example, if users A and B have purchased the same items and recently the user B has liked or bought a new item that the user A has not yet seen, it is apparently a good idea to recommend this item to the user A.

This approach is called collaborative filtering (abbreviated to CF) from the idea that the system filters the most promising items in a collaboration between users. It is now being used in the majority of recommender systems. These types of systems have been studied for more than 20 years now.

The main advantage of CF is that the system does not need to know anything about the items that it recommends. It only filters the items based on the similarities between users. The data entered in this system doesn’t need to be maintained or checked for correctness. The attributes are not important for recommendations. On the other hand, by not exploiting the properties of the items for obvious matches, we can lose a substantial amount of information. And it can lead to poor recommendations that only rely on the knowledge of its users.
The second problem that will certainly be a big issue for our application is that CF systems suffer from the so-called “cold start” problem. The algorithm needs to have a lot of user data and history available for interesting recommendations to be made. The recommender’s accuracy will suffer substantially, if there are no users to be compared and searched. The system cannot be used if there is no data at all. The next discussed content-based systems can be used to suppress this problem alone or in a hybrid combination.

According to the summary in [5], the main techniques in CF are:

- User-based nearest neighbor similarity, see [6] and [7]
- Item-based nearest neighbor similarity, see [8]
- Probabilistic recommendation methods
- Matrix Factorization techniques, see [9]

### 2.1.2 Content-based recommendations

Content-based (abbreviated to CB) systems exploit the available information about items. We can then recommend items based on similarity with other items that the user already liked or bought.

For example, in the bookstore domain, if a user A likes a book, we can recommend him a similar book as the one he liked. We can recommend the ones with the same author, genre, type, and more. Content-based systems automatically create and save profiles of users interests based on the items they have liked, visited or made any other interaction that could be viewed as a preference. This profile is then used to find items that are similar or in another way interesting to the user. Under items, we can imagine a broad variety of things, ranging from eshop items, news articles to web pages inside search engines.

We can learn the user’s preferences implicitly by observing the user as described above or explicitly by asking the user for a feedback about the object.

When compared with CF systems there is a substantial disadvantage. We need to understand the data over which we are making the recommendations. We can do this using methods from machine learning, information retrieval and information filtering. On the other hand, we do not need to have a big user base before the recommendations become fairly accurate and interesting. It all depends on our algorithm and quality of items’ annotation. In other words, there is no “cold start” problem which was described with CF methods. We can even recommend new items immediately as they are inserted into the system. In CF we would need to wait until the users start to interact with the item. In many cases, this type of system is not only used in e-commerce but also in other areas where there is a problem with the size of data. As we call it information overload.

Another problem is shallow content analysis as described in [10]. In comparison with CF the content-based systems cannot perform or are bad at an extraction of deeper connection and hidden similarities between users. There can be information that we have not described in the item’s properties or a connection that is hard for CB algorithms to grasp.

Next problem is overspecialization. If we are only recommending items that

---

1 Matrix Factorization techniques are among the current state-of-the-art
are similar to items that user already likes, there tend to be only recommendations of more of the same. This leads to the effect of obvious recommendations that are not always desired. Users also strive for novelty. So we would like not to only filter out items that are too different but also items that are too similar.

The cold start problem also exists in this type of recommendation. We cannot recommend any items if we don’t know user’s preferences. We can simply analyze his behavior for a while, wait until he likes some items explicitly or ask him for feedback directly. One other method is to ask the user to pick some items from an artificially created set of as much diverse and different items as possible.

According to the summary in [10], the main techniques in CB are:

- Keyword-based Vector Space Model, see [11] and [12]
- Nearest neighbors, see [13]
- Probabilistic methods
- Linear Classifiers, Rule mining, Random Forests and other machine learning approaches
- Explicit decision models

2.1.3 Knowledge-based recommendations

For domains that are used by users only once in a while as for example, buying a new television (we can believe that most users have a TV for multiple years) it is hard to monitor and save a user’s preferences. After such a long time, we can also assume that the preferences will be different. In those cases, we would like to exploit a different aspect of the data. That is when we would like to understand the data in the way, that if the user will tell us their preferences we will be able to find the best items based on those explicit preferences only. Knowledge and content based approaches can sometimes overlap as discussed in [5]. However, the difference is mostly in the interaction with the user. We will ask the user about their preferences and then with the knowledge of the data lead them to his final choice. We need to know the differences between the properties of items and the property rating function. For example, when buying a new cell phone, we need to know that it is better when weight is lower and the resolution higher, not the other way around. Then we can filter out the best option for the user. The main disadvantage is that we need to manually prepare every item category separately. We certainly cannot compare TVs with washing machines by the same metrics as well as between each other.

According to the summary in [5], the main techniques in knowledge based systems are:

- Constraints-based recommendations, see [14]
- Similarity search
- Dealing with unsatisfiable requirements and empty result sets
- Proposing repairs machine learning approaches
- Critiquing
- Preference Elicitation, see [15]
2.1.4 Hybrid recommendation approaches

As we can see from the techniques discussed above, all of them have their disadvantages. By combining them together to eliminate their problems, we could create a recommender system that is superior to any single technique used separately. The CF systems tend to recommend badly with the cold start problem and the content-based systems are bad at exploiting the hidden context. When we combine multiple recommender systems we call the combined system a hybrid recommender system.

There are many options as for how to combine multiple RS together, according to the summary from [5]:

- Monolithic hybrids
  - Feature combining
  - Feature augmentation
- Pipeline hybrids
  - Cascade
  - Meta-level
- Parallelized hybrids
  - Mixed hybrids
  - Weighted hybrids, see [16]
  - Switching hybrids

The main advantage is that when used right they remove imperfections and shortcomings of simpler recommendations techniques. In difficult situations, they return better results. But the problem with this approach grows with the complexity of the used techniques as well as the way of combining them. It is harder to manage and optimize such systems because of the complexity and increased “black boxiness” of the system.

2.2 Techniques used in our system

2.2.1 Tag extraction using TF-IDF

When we look at the Wikidata endpoint, we see that almost every book has its own Wikipedia page available. We would like to exploit that information for the use in our recommender system. Below, we first describe a naive approach to this problem and then we describe a slightly more advanced TF-IDF method for extraction of tag-based information from a text.

Naive approach: We could simply take a list of all words and their frequencies in a document and then compare the documents between themselves based on this list. This approach has multiple problems. First of all, it will have similarity between almost every document because of language elements like “a” or “the”, and similar, usually called “stop words”. We could get rid of them by simply removing them from the list, but that has its own limitations. Next problem is that this approach does not account for different document lengths. It causes that
the longer documents have a substantial edge over the short ones. The TF-IDF approach solves all those problems.

**TF-IDF:** It is the most common method for mining the importance of words in a document as discussed in [17]. The name comes from *term frequency - inverse document frequency*. And it is described by this formula:

\[
TFIDF(t, d) = TF(t, d) \times IDF(t)
\]

The **TF** represents term frequency \(TF(t, d)\) in document. It can have multiple variations such as:

- **Raw count** - \(f_{t,d}\)

\[
TF(t, d) = \text{number of times term } t \text{ occurred in document } d
\]

- **Simple Boolean frequency**

\[
TF(t, d) = 1 \text{ if the term } t \text{ occurs in document } d \text{ then else } 0
\]

- **Term frequency**

\[
TF(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}
\]

- **Log normalization**

\[
TF(t, d) = 1 + \log(f_{t,d})
\]

- **Augmented frequency** double K normalization:

\[
TF(t, d) = K + (1 - K) \left( \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}} \right)
\]

The **IDF** represents inverse document frequency \(IDF(t)\), it is a measure of how much information the word carries. It can, for example, identify the most common words such are the stop words and decrease their importance.

- **Unary**

\[
IDF(t) = 1
\]

- **Inverse document frequency**

\[
IDF(t) = \log \left( \frac{N}{n_t} \right)
\]

- **Inverse document frequency - smooth**

\[
IDF(t) = \log \left( \frac{N}{1 + n_t} \right)
\]
• Inverse document frequency max

\[ IDF(t) = \log\left(\frac{\max_{t' \in d} n_{t'}}{1 + n_t}\right) \]

• Probabilistic inverse document frequency

\[ IDF(t) = \log\left(\frac{N - n_t}{n_t}\right) \]

In the proposed recommender, we will use Term frequency, together with Inverse document frequency.

### 2.2.2 Similarity search algorithm

When making a recommendation, we often want to find the most similar items to some other item, or to some idealization of a perfect item that we can compose from user’s preferences. For this, we can use a set search algorithm called k-nearest neighbors (hereafter kNN). It is most popular in classification problems. But it can be used in recommender systems for its lazy nature as well. Let’s assume that the user right now views a book A, we would like to recommend books that are similar to the book A. When using a kNN algorithm we will find the most similar books using a metric function.

Some of the metric function choices are:

• Euclidean metric (continuous variables)
• Simple boolean metric
• Overlap metric (for text variables)
• Learned distance metrics (such as Large Margin Nearest Neighbor or Neighbourhood components analysis)

### 2.2.3 Obtaining preferences about users

**Explicit book rating:** Users can leave a rating about any book they like or have read. It is represented via a 5-star rating system where zero stars represent the lowest rating and 5 stars represent the highest.

**Explicit book rating with reviews:** Explicit rating with only a text review about a rated book. This type of rating, together with explicit ratings without reviews, has the highest priority in the recommender system.

**Visited book pages:** When the user visits a book, the system evaluates this behavior as a positive hint about user’s preference for the particular book.

**Search queries:** Every query that the user searches for is also evaluated as a small positive preference.

There is a debate about the importance of negative rating. For that reason and for a small simplification, we won’t include negative ratings to the content based recommender process.
2.3 Proposed recommender system

After thorough analysis of the data, we would like to build a hybrid recommender system that will combine multiple simpler ones. In this chapter, we will describe and propose a recommender system that will be implemented in the backend of our application.

We will design a recommender based on method from subsection 2.3.1 and subsection 2.3.2 at a book detail page, where we will use the now viewed book as a single preference point for which we will find the closest matches.

Next, we will use content-based methods from subsection 2.3.1 and subsection 2.3.2 in conjunction with user-based method from subsection 2.3.3 to recommend the user specifics books they might like at the home page.

2.3.1 Content-based system based on LOD properties

The first sub-recommender of the proposed hybrid system will be based on data obtained from LOD dataset. This recommender will work even when additional endpoints are added to the system. We have chosen a set of most common attributes that are mined from endpoints. List of attributes is below.

- Title
- English name
- Czech name
- Original language
- Publisher
- Authors
- Genres
- Characters
- Description

The first step is to evaluate user’s preferences. We will take all rated books of the user with positive rating 4 or 5 stars. We will find three nearest neighbors using a 3NN algorithm for every rated book. The 3NN will use simple metric - the score count where we increase score of candidate book by one for every single matched attribute. After this step, we will merge books by their score. If some book is present multiple times, we will sum up the scores.

2.3.2 Content-based system based on tags from Wikipedia

First of all, we would like use the mined tags only as a possible improvement mechanics in our algorithm because most of LOD endpoints do not have any data which could be transformed into tags. Therefore, after adding a new datapoint without tag information, the datapoint would not have the same weight as one with tag extracted information.

We store the top ten tags (by the TF-IDF score) for every book from every language it has Wiki page entry in. Many books are available in multiple Wikipedia language mutations. Because of the simplification and limiting the
computational complexity, we will use only tags for the top five languages that the books have, ordered by the amount of books we have for each language. For example, if the book has tags in English language, we will always use the English tags as the English is a language with most data from the mined Wiki pages.

For each language, we will find the top ten books by the computed scores. Where between books we use simple sum of tag score multiplications of matched tags. For example, if a book A and book B have two similar tags, tag1 and tag2, we will multiply the score for tag 1 of the book A with the tag 1 of the book the B and add that up it with the result of the multiplication between book A’s tag 2 and book B’s tag 2.

When we have top 10 lists for all languages, we will group the list by the books. As a group score, we will use the sum of the score from every language for the book. This way, we highlight the books with more matches and still provide space for books that have less but higher scoring tag matches.

2.3.3 User-based collaborative filtering system

Using user’s book ratings, we will find the most similar users using the nearest neighbor algorithm and then recommend books that have the best rating from the closest neighbors and that haven’t been seen by the user yet.

Our nearest neighbors algorithm will use simple euclidean squared distance metric. We are not using the Pearson distance metrics because it is more computationally complex and the project will mostly rely on the content based recommendation methods because of the cold start problem. If necessary the algorithm can be modified easily.

Let a user for which we want to make a recommendation be UserA. The algorithm first gets all the positive ratings from UserA where the rating is 3, 4 or 5. Then we extract all users that rated positively at least one book as a UserA has rated. We count metrics between all these users and UserA and pick the four most similar users. Then we pick the best books. Starting with a rating of five from the most similar user that the UserA hasn’t seen yet. Followed by others five stars rated books, then go to 4 stars rated books starting again with the most similar user.
3. Software design

In this chapter, we will discuss the details of the application’s software design. First, we will come up with a structure of the application and after that, we will examine individual parts.

3.1 Structure of the application

Technologies used, server side:

- .NET Core
- .NET Core MVC
- ASP.NET Core
- Entity Framework Core
- SQLite
- xUnit.net
- jQuery
- Javascript
- Json
- Ajax
- Razor
- HTML
- CSS
- Bootstrap

We have chosen .NET Core together with C# programming language as a base of our application technologies. It is a new development platform designed and maintained by Microsoft and the GitHub community. It is open-source and cross-platform. It was created as a part of the Microsoft big move to be able to sustain the competition on Linux environments.

Data will be stored on the server and the mining part of the application will be

1. .NET Core - https://dotnet.github.io/
2. .NET Core Model View Controller - https://github.com/aspnet/Mvc
3. ASP.NET Core - https://github.com/aspnet/
5. SQLite - https://www.sqlite.org/
6. xUnit.net - https://xunit.github.io/
7. jQuery - https://jquery.com/
12. CSS - https://www.w3.org/Style/CSS/
called only when there is a substantial amount of new data at SPARQL endpoints. The mining part will support retrieval of additional information on demand.

Recommender engine and search engine will use only locally stored data. These components will be called directly from the MVC Controller. We can see overview of application’s architecture on Figure 3.1.

Figure 3.1: The main architecture of the application

### 3.2 Database

First, we need to discuss how to query and store data. We could write the whole application without almost any data stored locally, but the problem is that the SPARQL endpoints are not as reliable as we would like. In the end, it is only to some extent experimental free service above which we have absolutely no control. Secondly, the queries can take up to 30 seconds to be processed. We would like to offer our user much more reliable and faster service. Based on previous reasons, we would like to store as much data locally as possible.

For the purpose of storage, we chose a standard SQL database SQLite. It is mostly used on mobile devices. This database is quite special, and very different from others. It is actually only a file stored in the file system. You have to provide the support for reading this file at the side of your application. It is not a standalone database application.

For our application, we will use Entity Framework. It is an object-relational mapper, which acts as a middleman between your application and your database. You can then write all data related queries through an interface that automatically converts data from the database to classes designed inside your application. For example, in our system we use .NET LINQ queries to access our data directly from the database.

---

Example of a simple Entity Framework LINQ query that retrieves user’s ratings that have bigger score than 2:

```csharp
var userRatings = db.Ratings.Where(r => r.UserId == userId && r.Rating >= 3)
    .ToList();
```

LINQ is a programming model and methodology that essentially adds formal query capabilities into Microsoft .NET-based programming languages. LINQ offers a compact, expressive, and intelligible syntax for manipulating data. Entity framework is a set of mappings from LINQ to database SQL queries. It supports many database providers such as Microsoft SQL Server, SQLite, MySQL and more. We have chosen the combination of SQLite and Entity Framework Core for the provided simplicity, faster query design, and the simple installation and deployment. It can be deployed on a server only by copying a database file and setting the path for the file inside our configuration system.

We will use the Code first way of creating and maintaining database schema. Entity Framework provides us with the ability to use special attributes together with OOP\(^{15}\) class syntax in C# to create the database schema. The Entity Framework then returns these classes as the retrieved objects from LINQ queries.

List of the possible databases for which the Entity Framework Core binding\(^{16,17}\):

- Microsoft SQL Server
  - Managed by Microsoft
  - Widest support of functionality
- SQLite
  - Managed by Microsoft
  - Limited functionality when updating database schema
- PostgreSQL
  - Part of Npgsql project
- IBM Data Server
  - Maintained by IBM
- MySQL
  - Managed by MySQL community
- InMemory
  - Maintained by Microsoft
  - Used for testing, data not persistent
  - Full functionality
- Oracle
  - Oracle team is evaluating support
- MyCat
  - Managed by the Pomelo Foundation Project

The SQLite binding has a problem that it cannot update the database schema. We bypassed this problem by creating the updated database schema inside dif-

\(^{15}\)OOP - object-oriented-programming
\(^{16}\)binding - also as database provider
\(^{17}\)As of the time of writing 5.2017
ferent database file and then manually edited our main database file using DB Browser for SQLite.

3.3 Server

On the server side, we will use standard MVC pattern that is supported natively on the .NET Core platform. The server is the central component and manages all requests from clients. When a client requests a page, first, the server will use the router to direct the request to the right controller. The controller manages all other parts such as the recommender engine and others, it composes the data and inserts them to the model. The model is then forwarded to the view which uses Razor syntax together with HTML and CSS to compose the web page from the model. The web page is then sent back to the client as a response to the request.

Razor is a markup language that lets you insert server-based code into web pages. The code is processed on the server. It is different from the user-side scripts. The user-side scripts run after the response is received on client side inside user’s browser.

3.4 Client

We want to provide our users with the smoothest experience possible. Because of this, we will load the long running operations as are recommendations and pictures using AJAX calls. AJAX call is used to receive, send or update data on the web page without the need of reloading the page. We have a separate controller on the server for handling all AJAX calls. AJAX call is then handled on the client side using jQuery. For example, when the page with recommendations is loaded, the client sends AJAX calls to the server to retrieve all data that will take longer to be process by the server. This way the page itself is loaded quickly and then the long running elements are updated as they are processed.

3.5 Security

Security of the application is provided by combining multiple technologies. For storing, managing and accessing login information and passwords on the server side, we use standard .NET Core .ASP tools.

They are:

\begin{verbatim}
Microsoft.AspNetCore.Authorization;
Microsoft.AspNetCore.Identity
\end{verbatim}

These two packages provide many options for customization and handling of the sensible data. Passwords are stored hashed in our database.

The use of all these technologies would be incomplete without using HTTPS. When using HTTP protocol, all sensitive information is going through the web in

\footnote{DB Browser for SQLite - \url{http://sqlitebrowser.org/}}

\footnote{Hyper Text Transfer Protocol Secure}
an unsecured format. We will use HTTPS together with a certificate issued from a Global Certification authority. The application itself communicates only with HTTP, but we will deploy the application behind a reverse-proxy server that will handle the encryption using SSL and communicate with clients only by HTTPS. The unsecured HTTP communication is only between our application and the proxy server on the deployed operating system. We can use IIS on Windows Server and NGINX on the Linux deployment. Representation of the communication is shown on Figure 3.2. The ASP.NET Core server Kestrel can be used alone, but it is not yet mature enough and the developers of the .NET Core platform do not recommend to use it that way in a production environment.

Figure 3.2: Communication between user and the application

The .NET Core together with Entity framework provides protection against multiple types of threats. We will now name the most common ones:

- **SQL injection**
  - SQL injection is the most common security breach, in our application, it is handled by the Entity Framework. The framework automatically escapes all possibly harmful combinations of characters. The only way to do this kind of attack is if there is used EF function of Raw SQL Queries, then we need to check for SQL injection by ourselves. We use Raw SQL Queries only in the manage part of the application that is not publicly available and is protected behind a separate administrator’s password.

- **Cross-Site request forgery (also known as XSRF or CSRF)**
  - This type of attack is based on the trust between a browser and a website. Usually exploiting authentication token used, for persistent

---

20 Abreviation for public and private key pair certificates
21 CA - is a trusted entity that issues electronic documents that verify a digital entity’s identity on the Internet
23 IIS - Internet Information Services (formerly Internet Information Server) is an extensible server created by Microsoft
24 Windows Server is an operating system adapted to be used in a server environment developed by Microsoft
25 NGINX is an open-source, free server that can act as a HTTP server, reverse server and many more... [https://www.nginx.com/](https://www.nginx.com/)
26 Kestrel is an open-source, cross-platform web server for ASP.NET Core platform
login. It can be misused by the attacker to call requests on the targeted site from the attacker site. Users can defend from this type of attack by logging off from websites and clearing their browser’s cookies history.

– We defend from this exploit by including a unique token inside every request. The token then needs to be send back by the client for verification. .NET Core ASP automatically inserts these tokens inside POST methods.

• Cross-Site Scripting (XSS)
  – It is a vulnerability which enables client side scripts to be placed inside web pages. It enables the attacker to then steal cookies and session tokens or change the content of the web page using DOM manipulation.
  – We are protected from this exploit by using Razor syntax, it automatically escapes all input and output data. We only need to be careful when inserting dynamic data inside HTML element script. We also need to take care when inserting untrusted data inside URL parameters.
4. User documentation

This chapter describes and explains how to use the web application. We present the application interface and logic from the standpoint of a standard user. The application can be found at

https://recommender.projekty.ms.mff.cuni.cz

4.1 Homepage

The home page (see Figure 4.1) will be the first page for most users. A user is greeted with the link for searching the data in the top center. Most navigation actions are available from the navigation bar on the absolute top. From the left, the user can search or view some information about the project and on the right register and login.

![Homepage of the application](https://example.com/homepage.png)

Figure 4.1: The main page of the application.

4.2 Register and login

The registration and login are simple and straightforward. From any page of the application, a user can click on the Login or Register control in the right corner of the application. After loading of the Registration page (shown below), the user needs to fill in all the text areas. They are: Nickname, email, and password. The provided email address is then used for the logging in together with the password. After the registration is completed, a user needs to sign in on the Login page (shown below).

After the registration is completed, a user needs to sign in on the Login page (see page 29).

1At least till the author’s bachelor thesis defense - 6.2017
Figure 4.2: The register page.

Figure 4.3: The login page.
4.3 Searching for books and authors

A user can search the database using the search functionality (see Figure 4.4), it is accessible from the main page or from any page using the link control at the left top corner of the application. The page is simple, if user wants to search, they need to input their query to the search bar and press enter or the search button. Results will be shown under (see Figure 4.5).

Figure 4.4: The search page.

Figure 4.5: The search page with results for the query “great gatsby”
4.4 Detail of books and authors

When a user clicks on the link from search, recommendation or other place referencing to book or author, they will be redirected to the detail of the book. We will show the detail for books. It is mostly the same for the authors, only with slightly different layout and less functionality.

On the detail of the book, we can see its name, author and other information, together with a picture on the right side. Over the image, there is the current score of the book, based on reviews of other users. Below the image, a signed-in user can use the “Rate this book” button, if they would like to add a rating to the book.

![Figure 4.6: Top part of a book detail.](image)

On the bottom part of the “detail page”, user can find a tab menu containing two types of recommendations, reviews from other users and dynamic data about the book loaded from the endpoint if it is available.

4.5 Feedback and user profile

A signed-in user can send feedback using a form which can be shown by clicking on the element Send feedback which is accessible from all pages on the navigation bar.

4.6 Requirements

For the best user experience, we recommend:

- Modern and up-to-date web browser
- Reliable and fast internet connection
- Enabled Javascript
- Support of HTTPS
Figure 4.7: First tab of recommendation for the book.

Figure 4.8: Multiple clickable tabs of different groups of dynamically loaded information.

Figure 4.9: Send feedback form.
5. Administrators documentation

In this chapter, we will describe as how to install and run a server for the proper functioning of the web application. First, we will state the list of requirements and how to deploy services needed for running the application. Then we will describe how to generate application’s database and run mining requests for acquiring of LOD data.

5.1 Server requirements

The .NET Core application can be run in two different modes. One of them is a self-contained application that includes all code for running the application. It does not need to have the runtime environment present on the server. It comes packed with it. The second one is the standard mode, which does require to have a runtime environment present at the server.

For our documentation, we will use the self-contained version, which is build and published for the Ubuntu 16.04 x64 system. It is a part of this bachelor thesis together with the source code and the software documentation as an electronic attachment.

Server requirements:

- Ubuntu Server 16.04 2 LTS
- Reliable internet connection and public IP address
- 2 GB of memory
- 7 GB of free storage (SSD preferred)
- At least 2 core CPU @ 1 GHz
- Administrative permissions

Applications requirements:

- Nginx
- ufw

Building requirements:

- .NET Core Runtime >= 1.1
- .NET Core SDK >= 1.0.3
- Internet connection, used by the package system

5.2 Deploying the application

We will describe the process as to how to run the application behind the reversed proxy server on the HTTP protocol. Description of the usage of the HTTPS

\footnote{For application, database, Ubuntu operating system and all supporting programs}

\footnote{Required version of .NET SDK and Runtime can be downloaded here - \url{https://github.com/dotnet/core/blob/master/release-notes/download-archive.md}, the official release as of time of writing 5.2017 does not meet our requirements.}

\footnote{Note, if the user just want to test the application, just run the ”BookRecommender”(or .exe) from compiled binaries provided as apart of the electronic attachment}
encrypted protocol is above the extent of this work and many people have already
done great work when describing the process. One of the publicly available great
sources can be found here.\[^4\]

For simplicity, we will assume that the application is present in the user’s
home directory at:

```
/home/<user>/netcore/br/
```

And the database file is present at:

```
/home/<user>/netcore/database/BookRecommender.db
```

### 5.2.1 Building the application

We need to have a .NET Core SDK installed.

Then simply from the project folder run this set of commands:

```
    dotnet restore
    dotnet build -r ubuntu.16.04-x64 -c Release
    dotnet publish -r ubuntu.16.04-x64 -c Release
```

The published application will be in:

```
    ./bin/release/netcoreapp1.1/ubuntu.16.04-x64/publish
```

### 5.2.2 Configuring app settings and creating database

Our application has 3 different setting options. They can all be configured inside
`appsettings.json` file.

The default configuration is shown below.

```json
{
    "Database": {
        "Connection": "Filename=/home/<user>/netcore/database/BookRecommender.db;cache=shared"
    },
    "Mining": {
        "WikiPagesStorage": "/home/<user>/netcore/booksWikiPagesStorage/",
        "Password": "milujemejidlo"
    }
}
```

- Database-Connection is the path to the database file.
- Mining-WikiPagesStorage is the path to the downloaded Wikipedia pages.
- Mining-Password, is a password for the access to the manage part of the application.

The empty database file is one of the attachments of this bachelor thesis. We can use it simply by copying to the desired location. If we want to generate our own clean database, we need to have .NET Core SDK installed. Then we can call commands below from the application’s source code folder to create a new database.

```
dotnet restore
dotnet build
dotnet ef migrations add Initial-Migration
dotnet ef database update
```

The database should be created at the location from our specified connection string.

5.2.3 Nginx

Nginx along with Apache are the most popular web servers of today’s internet. Nginx can also be used as a reverse-proxy server, load balancer and HTTP cache. It is free and open source software. We will use Nginx software as a reverse-proxy server to protect our application against multiple malicious threads. As the native server is not yet mature enough to be used bare in the production environment.

First, install Nginx using commands below and then run the service as a super user:

```
sudo apt install nginx
sudo service nginx start
```

By now the Nginx server should be running and listening on the default localhost:80. We can validate that by looking on the localhost address using curl.

```
curl localhost:80
```

Now we need to configure Nginx to act as a reverse proxy server for our application by rewriting the default config file found at /etc/nginx/sites-available/default

Modification example using nano editor:

---

---

5 Apache HTTP Server - [https://httpd.apache.org/](https://httpd.apache.org/)
6 Load balance is server which distributes requests across multiple other servers
7 HTTP Cache server saves returned requests and reuses them as a response when the same request is seen.
8 Reverse-proxy server acts as a pass through from the internet to application’s Kestrel server
sudo nano /etc/nginx/sites-available/default

We will input this code to the file:

server {
    listen 80;
    location / {
        proxy_pass http://localhost:5000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection keep-alive;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}

After the editing and saving the file, we can verify that our configuration is valid and then apply the changes to the Nginx server.

    sudo nginx -t  # validate the configuration file
    sudo nginx -s reload  # restarting the service

After this, we should be able to access our server using port 80 on our localhost, or from a public ip if available.

### 5.2.4 Our application as a service

We could run the application by itself, but it is not prepared to handle all catastrophic events that can occur. We would like to have the application restarted automatically if there is some sort of failure, both on the application side or the server side, such as the application crashing or the server restarting.

First, we will create a new service file definition as follows:

    sudo nano /etc/systemd/system/kestrel-bookrecommender.service

We will input this code\(^9\) to the file:

    [Unit]
    Description=.NET Core Book Recommender application

    [Service]
    WorkingDirectory=/home/<user>/netcore/br
    ExecStart=/home/<user>/netcore/br/BookRecommender
    #Restart service after 10 seconds if crashed
    Restart=always
    RestartSec=10

\(^9\)Note, user has to have a permission to access and manipulate with all files and folders of the application.
SyslogIdentifier=dotnet-bookrecommender
User=user #account under which should the service run
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target

After saving the file, we can enable and then run the new service:

systemctl enable kestrel-bookrecommender.service
systemctl start kestrel-bookrecommender.service

We can check if it is running by:

systemctl status kestrel-bookrecommender.service

Which starts to follow the service console output. Now, our service should be running and we can check the application by:

curl localhost:5000

5.2.5 Securing our application

It is necessary to configure proper firewall rules to minimize the threat from the internet. We will use ufw firewall system.

First, we need to install the firewall:

sudo apt-get install ufw

Next we set a simple rule to limit access only to port 80 on which the Nginx server listens. If accessed remotely don’t forget to allow port used for remote access too.

sudo ufw allow 80/tcp

After setting the rules, we can start the firewall:

sudo ufw enable

5.2.6 Possible problems

If server does not run because of missing

system.net.http.native

Then, please install .NET SDK on the server. The steps are described on the official site of .NET Core platform[^10].

If server is missing libunwind.so.8 library, then please install it by using this command:

sudo apt install libunwind8

If service cannot start, check if .br/BookRecommender has permission to run. If not add the permission:

sudo chmod +x BookRecommender

[^10]: How to install .NET SDK - [https://www.microsoft.com/net/core#linuxubuntu](https://www.microsoft.com/net/core#linuxubuntu)
5.3 Mining data from the application’s interface

The administrator can start the process of data mining simply from the mining page (see Figure 5.2) by going to the web application’s manage URL <server address>/Manage. The administrator needs to be a logged in user. The page (see Figure 5.1) prompts us to fill in the administrator’s password from the appsettings.json file. After the first non-public access, the application will remember the user as an admin.

To start the mining, the administrator can simply click on the start mining button. The resource mining will then start. Alternatively, he can click on the stop button to stop the scheduled mining. Once the mining is running, it cannot be stopped. It can only be stopped when it has waiting status. We can use the start all and Stop all buttons to avoid repetitive task when we want to mine all resources. The “start refreshing” and “stop refreshing” buttons will start and stop synchronization of the mining states with the server. We need to start the synchronization to view the progress of the mining.

Figure 5.1: Get access page, accessible from <server address>/Manage.

Figure 5.2: The manage page from which can be started the mining tasks.
Conclusion

We have explored multiple LOD data points from which, we have chosen Wikidata data set as the most fitting one. We have designed a data miner component which transforms data from Wikidata API service to our relational database. The data miner can be easily extended with other data points.

We have extracted tags from Wikipedia pages, which have been referenced from the LOD data set as a source of additional information. On the top of the LOD data and tags from Wikipedia, we have designed a recommender system, which can appropriately recommend books by using the most popular recommender systems’ methods.

The web page has been developed to have a resizable and swift interface using modern design tools. The server dynamically loads other information about books directly from the data point and sends them to the clients as requested. Among the dynamic information, we can find additional language information. This information could be used together with auto-translate functionality in the user’s browser as a localized version of the web page. The application has its own administrators’s environment from which all mining can be done by the administrator.

We have found that it is certainly possible to use the datasets for making recommendations. Even if the datasets are not yet sufficiently big to be used in a commercial environment. In the case that we would have limited our recommender to use only user-based collaborative filtering, then some of the datasets would reach commercially viable sizes. The problem is. this would come at the expense of sparse information density about books and authors which could be shown to the users of the application.

Future work

There is a huge number of available data points. Almost all could be used even when they don’t contain richly annotated book records. In these case of data sparse data points, we could use only a user-base content filtering for generating recommendations. The application was designed with the focus on it being easily expanded by additional datasets. It is desirable to extend the application with language specific data sets to be used in the multinational environment. Our data set provides sufficient data background only for a handful languages such as English and Spanish. From which English is by far the biggest one.

Furthermore, many performance improvements can be introduced. For example, the so called offline evaluation can be implemented to improve robustness and scalability of the system.
Bibliography


Attachment

Electronic attachment provided as a part of this thesis with the following contents:

- Text of the thesis in a PDF format
- Programming documentation
- Application’s source code
- Application in binary form compiled for:
  - 64bit Ubuntu 16.04 LTS system
  - 64bit Windows 10 system
- Empty database file for the application