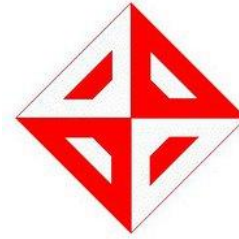


Middle East Technical University
Department of Computer Engineering



RECOMMENDER SYSTEM

Software Requirements Specification Document

V1.1

Dcengo Unchained



Duygu Kabakcı – 1746064

Işinsu Katircioğlu – 1819432

Sıla Kaya - 1746122

Mehmet Koray Kocakaya – 1746189

December 29, 2013

CHANGE HISTORY

Date	Revision	Comment
24.11.2013	1.0	Created
29.12.2013	1.1	2.1.2. User Interfaces is updated 3.2. Functional Requirements is updated 4.1.1. Data Objects is updated 5.2. State Transition Diagrams is updated 6.1. Team Structure is updated

PREFACE

This document contains the Software Requirements Specification (SRS) of a Recommender System. This document is prepared according to the “IEEE Recommended Practice for Software Requirements Specifications – IEEE Std 830 – 1998”. Main purpose of this documentation is to give detailed information about the software requirements and functionalities of the Recommender System.

The Recommender System aims to enhance the existing recommendation tools served to music streaming and downloading tools. In this project, the challenge is to overcome the major constraints and carry out the basic functionalities of such systems. The target audience who wants to make use of this system, can find all related requirements information in this document. It assists the software developer team, the stakeholders and the end users.

The first section gives the definition, purpose and scope of the Recommender System. The following sections include detailed description and requirements of the project. This specification is the primary document upon which all the design, implementation, and test/validation plan will be based.

TABLE OF CONTENT

1. INTRODUCTION	5
1.1. PROBLEM DEFINITION.....	5
1.2. PURPOSE.....	5
1.3. SCOPE	6
1.4. DEFINITIONS, ACRONYMS AND ABBREVIATIONS	6
1.5. REFERENCES	7
1.6. OVERVIEW	8
2. OVERALL DESCRIPTION	8
2.1. PRODUCT PERSPECTIVE	8
2.1.1. SYSTEM INTERFACES.....	9
2.1.2. USER INTERFACES.....	9
2.1.3. HARDWARE INTERFACES.....	12
2.1.4. SOFTWARE INTERFACES.....	12
2.1.5. COMMUNICATION INTERFACES.....	14
2.1.6. MEMORY	14
2.1.7 OPERATIONS	14
2.1.8. SITE ADAPTATION REQUIREMENTS	14
2.2. PRODUCT FUNCTIONS	14
2.3. CONSTRAINTS.....	15
2.4. ASSUMPTIONS AND DEPENDENCIES	15
3. SPECIFIC REQUIREMENTS.....	16
3.1. INTERFACE REQUIREMENTS	16
3.2. FUNCTIONAL REQUIREMENTS	16
3.2.1. USER USE CASES	17
3.2.1.1. USE CASE: GENERATE DATA.....	17
3.2.1.2. USE CASE: GET RECOMMENDATION.....	18
3.2.2. INTERAGENT.....	18
3.2.2.1. USE CASE: PROVIDE DATASET	18
3.2.2.2. USE CASE: UPDATE DATASET	19

3.2.2.3. <i>USE CASE: INTEGRATE WEB SERVICE</i>	19
3.3. <i>NON-FUNCTIONAL REQUIREMENTS</i>	20
3.3.1. <i>PERFORMANCE REQUIREMENTS</i>	20
3.3.2. <i>SECURITY</i>	20
3.3.3. <i>USABILITY</i>	20
3.3.4. <i>PORTABILITY</i>	20
3.3.5. <i>SAFETY</i>	20
3.3.6. <i>OTHER REQUIREMENTS – GNU GPL LICENSE</i>	21
4. DATA MODEL AND DESCRIPTION	21
4.1. <i>DATA DESCRIPTION</i>	21
4.1.1. <i>DATA OBJECTS</i>	21
4.1.2. <i>DATA DICTIONARY</i>	23
5. BEHAVIORAL MODEL AND DESCRIPTION	23
5.1. <i>DESCRIPTION FOR SOFTWARE BEHAVIOR</i>	23
5.2. <i>STATE TRANSITION DIAGRAMS</i>	24
6. PLANNING	27
6.1. <i>TEAM STRUCTURE</i>	27
6.2. <i>ESTIMATION</i>	27
6.3. <i>PROCESS MODEL</i>	28
7. CONCLUSION	29

1. INTRODUCTION

1.1. PROBLEM DEFINITION

Recommendation systems are software tools or knowledge discovery techniques that provide suggestions for items to a user. Items can be music, books, movies, people or social groups. The aim of these systems is to recommend items that a user is likely to be interested in and learn more about user preferences and constraints. This project is based on improving the existing recommendation systems and generating more accurate recommendations for music streaming and downloading applications with large number of users and tracks.

1.2. PURPOSE

This document provides a detailed description of Software Requirements Specification (SRS) for Recommendation System. It is prepared according to “IEEE Recommended Statements for Software Requirements Specification - IEEE Standard 830 – 1998”.

The Software Requirements Specification (SRS) document is intended to provide the requirements of the Recommender System project and the expectations of the stakeholders. The document includes the project perspective, data model and constraints of the overall system.

The intended audiences of the document are project managers, developers, testers and end users.

- Project managers review the document and determine whether the planned system fulfills the requirements. They notify the developers to fill up missing parts.
- Developers provide consistency by using the documentation.
- Testers use the documentation for verification and validation.
- End users make use of this document to learn about the scope of the system and its capabilities.

1.3. SCOPE

The targeted software product is a Recommendation System. The system makes an extensive use of user data to come up with reasonable track predictions about user preferences and there are two main paradigms in this sense: Content-based and collaborative filtering. As a beginning, the scope of the system is scaled down to collaborative filtering. This type of filtering is based on collecting and analyzing huge amount of user information; users' behaviors, activities, preferences and similarities to other users. The scope consists of determining similar users according to the similarities between their evaluations of certain track attributes. The system is aimed to have additional solutions for big data and sparse information. The test data will be provided from Argedor. The test data for the system are planned to contain millions of users and irregular data such that some user-track relation remains unknown. The project limits the improvement of the recommendation system around these challenges and favors accuracy over performance while giving the final suggestions. As a milestone, the system should follow the methods to enhance accuracy, the performance enhancements are not included in the scope. The Recommender System will be a Web Service and the end users will be directly exposed to the suggestions made for them.

1.4. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Term	Description
API	Application Programming Interface
Big Data	A collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications
Cold Start	The issue that the system cannot draw any inferences for users or items about which it has

	not yet gathered sufficient information
IEEE	The Institute of Electrical and Electronic Engineers
OS	Operating System
RMSE	Root Mean Square Error
UML	Unified Modeling Language
Web Service	A software function provided at a network address over the web or the cloud
UI	User Interface

1.5. REFERENCES

- [1] Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*, 22(1), 5-53. doi:10.1145/963770.963772
- [2] IBM. (n.d.). Portability and Interoperability. Retrieved October 27, 2013, from <http://www.ibm.com/developerworks/webservices/library/ws-port/>
- [3] IEEE. (1998). IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society.
- [4] Rubens, N., Kaplan, D., & Sugiyama, M. (2011). Active Learning in Recommender Systems. In P. B. Kantor, F. Ricci, L. Rokach & B. Shapira (Eds.), *Recommender Systems Handbook* (pp. 735-767). New York, USA: Springer.
- [5] Sparx Systems. (2000). UML 2 Tutorial. Retrieved October 26, 2013, from http://www.sparxsystems.com/resources/uml2_tutorial/
- [6] Peter Neubauer.(2009). Neo4j vs Hadoop. Retrieved October 27, 2013, from <http://lists.neo4j.org/pipermail/user/2009-April/001118.html>

1.6. OVERVIEW

This document includes six chapters:

- Overall description
- Specific requirements
- Data Model and Description
- Behavioral Model and Description
- Planning
- Conclusion

The software requirements begin with the overall description of the project. The product perspective, major functions and constraints are covered briefly. The software and hardware interfaces are stated in the next section as well. The following chapter gives the detailed requirements and functionalities of the software product. The functional and nonfunctional requirements are explained broadly. The design model and design constraints are mentioned extensively. One complete chapter is reserved for the data description and the behavioral model separately. SRS document involves UML diagrams for the project design. The use case, class and state diagrams are depicted in related sections. The documentation is finalized with the estimated schedule and team work for the project.

2. OVERALL DESCRIPTION

2.1. PRODUCT PERSPECTIVE

Recommendation system is a software tool that provides suggestions for items to a user. Our recommendation system is a component of a larger system which is a music streaming and downloading web application. The application integrates our system to main web application as a web service. Web services extend the World Wide Web infrastructure to provide the means for software to connect to other software applications. Our web service helps existing larger system to increase in usage of system by accurate suggestions. Moreover, this web service is free and open source with a GNU General Public License (GPL).

Music downloading and listening service is supported by Argedor. This service provides users with the legal right of accessing music files. It also enables the users to download music in mp3 format in case of purchasing music packages. The service provides the opportunity of

displaying latest news and music lists along with going over the lyrics and album information. Moreover, there are two types of track lists, professional lists provided by application and user lists created by individual user. Users can display both two types of lists. All these actions generate dataset to our system for accurate recommendations based on users' and items' similarities. ARGEDOR -supporter company- acts like an interagent between users and our recommendation system. The dataset is the most important part of our system therefore main system's existence and wide usage are important for generating recommendations on our system. The external interface and the interconnection between the large system and the Recommender System can be seen in Figure 1.

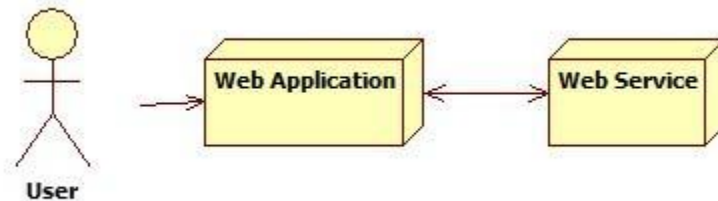


Figure 1: Block diagram

2.1.1. SYSTEM INTERFACES

There is just one system requirement which is Internet Access.

2.1.2. USER INTERFACES

Music streaming and downloading web application have a user-friendly GUI to provide ease of use and effectiveness to the users with different roles. Besides, our web service is added to web application. Therefore, we are not going to make new user interfaces for the application. Our web service will be integrated to web application and the recommendation result will be displayed through web application GUI. The only interface which we will implement is web service interface to display recommendation for companies using this web service.

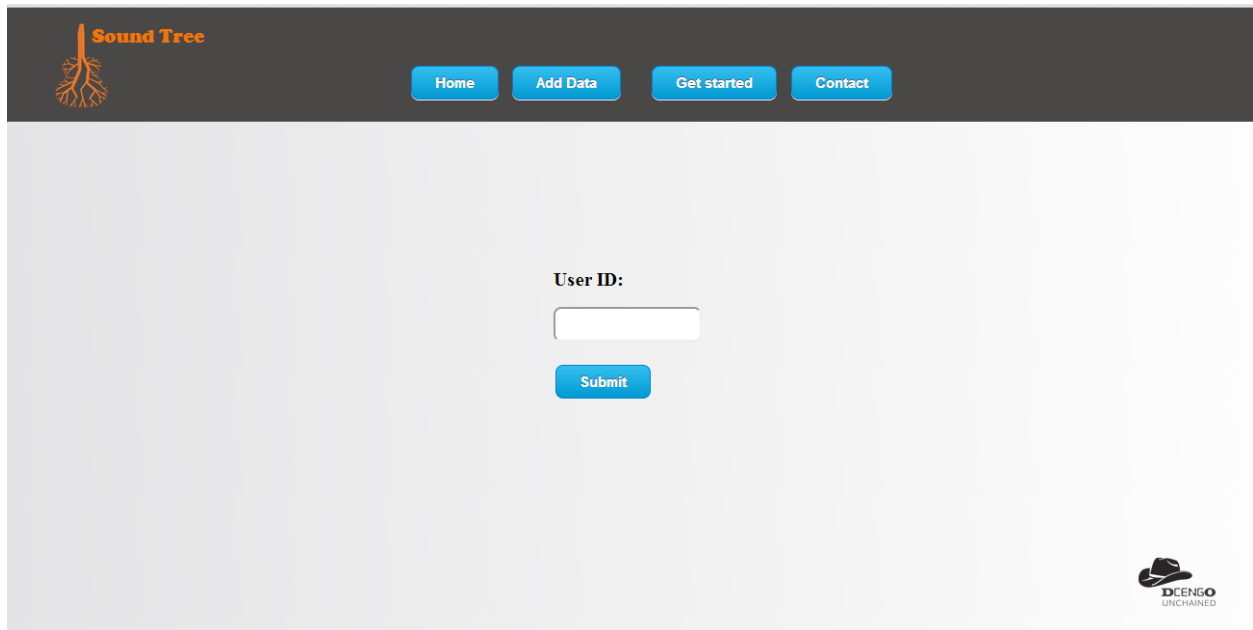



Figure 2: Web Service Interface

In Web Service Interface, there are four pages Home, Add Data, Get Started and Contact (Figure 2). This page (Get Started) contains a text field and a button. The user of the system who want to get recommendation fill the text field with proper user id. After s/he fill the text field and click the submit button, new page (Figure 3) is opened and recommendations are shown in this page. It contains a table with three columns which are Song Name, Performer Name and Album Name. Recommendations are listed in the table according to their song, performer and album names. The row number of the table can change according to how many recommendations are given to the user.



Home Add Data Get started Contact

Song Name	Performer Name	Album Name
Hadi Ordan	Gökhan Keser feat. S?la	Hadi Ordan
Yoruldu	S?la	Türkçe Pop 2012
Sevi?meden Uyumayal?m	S?la	Power Türk En ?yiler 2009 - Avrupa 2009
Bir O?lumuz Var	Demet Akal?n	Giderli 16
A?iyorum	Demet Akal?n	Giderli 16




Figure 3: Web Service Interface

In Add Data page (Figure 4), there are five segments. The user of the page should fill these five segments with text file. Song, Performer and Album document include information about them. Map document (relationship between song, performer and album) matches these three items. User Logs Document contains all of the information about played or downloaded songs. These five documents must be formed according to dataset provided by Argedor. When Submit button is pressed, database is updated with information in these text files. This page is created to update data and add new information when new log, song, album, performer and map data is created.

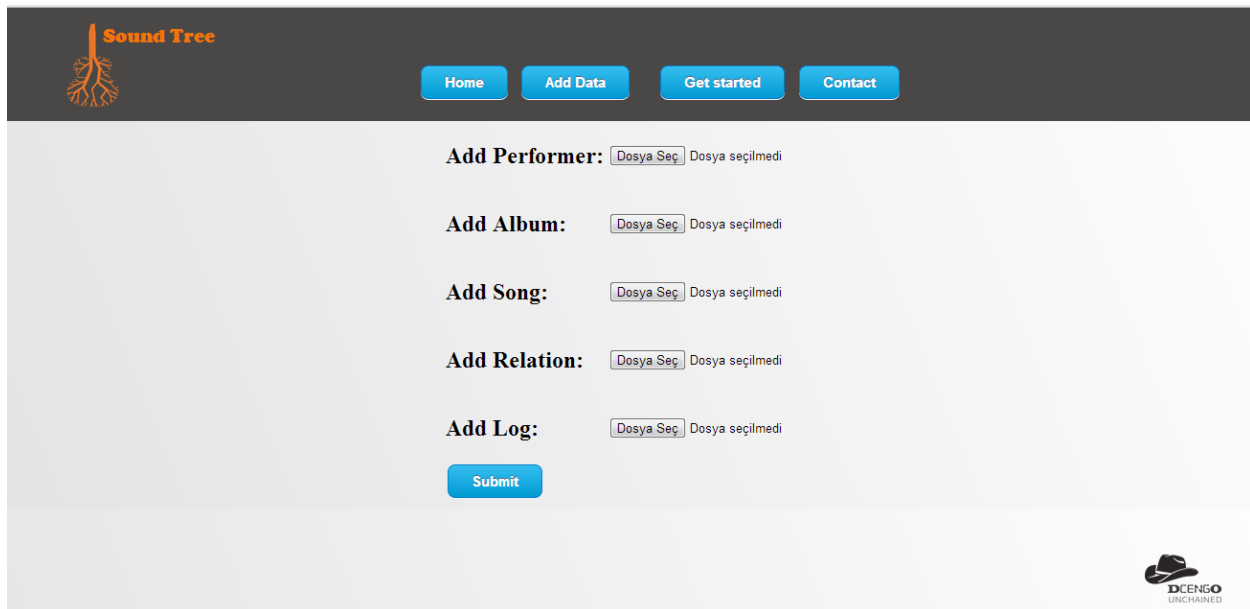


Figure 4: Web Service Interface

2.1.3. HARDWARE INTERFACES

Recommendation system offers accessibility to a great extent by storing your collection data on cloud. This brings out the requirement of a network interface on the device. User should have a device with valid internet connection, Wi-Fi or 3G, to be able to keep collection synchronized with cloud.

2.1.4. SOFTWARE INTERFACES

The data of the project is organized in a graph because the aim is to handle big data. A graph is a collection of nodes and edges (relationships) that connect node pairs. Key-value pairs are defined on nodes and edges. This is a powerful way of representing relations in the project data. In a graph database, relationships are first-class citizens. They connect two nodes. These two nodes and relationships can hold excessive amount of key-value pairs. Graph database stores data in a graph, the most generic form of data structures, capable of elegantly representing any kind of data in a highly accessible way. Graph database is approximately 1000 times faster for many queries on connected data than a relational database. Moreover, graph database is more intuitive for modeling many domains as a graph. Because of these reasons, graph database will be used in the project. Neo4j is a Java based open source graph database and it is the most

popular graph database. Its last version Neo4j 1.9, which will be used for our system, was released on 2007. Neo4j is well suited for many web use cases like metadata annotations, social networks, tagging, wikis and other network-shaped data sets. As mentioned before, it is a graph oriented model for data representations. Instead of static or rigid tables, columns and rows, we can work with a flexible graph network consisting of nodes, edges (relationships) and properties. It has supported many programming languages like Python, Ruby, Java, .Net, Perl, JavaScript etc. Neo4j is chosen instead of Apache Hadoop due to these reasons. Hadoop is mostly concerned with relatively flat data structures and it is extremely fast and scalable regarding retrieval of simple objects, like values, documents or even objects. However, if one should do deeper traversal of e.g. a graph, then it is required to retrieve the nodes for every traversal step (very fast) and then match them in some other manner (e.g. in Java or so) – slow. Whereas, Neo4j is built around the concept of “deep” data structures. This gives almost an unlimited flexibility regarding the layout of the data, domain object graph and very fast deep traversals (hops over several nodes) since they are handled natively by the Neo4j engine down to the storage layer and not the client node. The drawback is that for huge amounts (over one billion nodes) the clustering and partitioning of the graph becomes non-trivial, which is one of the areas this project is based on [6].

Java will be used as the programming language for this project. The last versions of Java Standard Edition 7 was released on July 28, 2011 and Java Enterprise Edition 7 was released on May 28, 2013. As mentioned before, Neo4j is implemented via Java, so finding and using Neo4j libraries for Java will be easier. This is the most important reason why we choose Java to implement the project. Another reason is Java’s performance. Java platform achieves superior performance by adopting a scheme by which the interpreter can run at full speed without needing to check the run-time environment.

Another software product which will be used is Amazon Elastic Compute Cloud (EC2). It is designed to make web-scale computing easier for developers. EC2 allows users to rent virtual computers on which to run their own computer applications. EC2 was developed on August 25, 2006 by Amazon.com.

2.1.5. COMMUNICATION INTERFACES

This system is a web application; therefore network connection with TCP/IP protocol is necessary.

2.1.6. MEMORY

At the beginning of the Recommendation System project, we will be using our local computers. Therefore minimum 2 GB (64-bit) RAM and 20 GB (64-bit) hard disk space will be needed.

2.1.7 OPERATIONS

Our web service makes recommendations even if the user hasn't played any tracks. This is cold start and the recommendation operations related to this phase are not user specific. For gaining better recommendations, user shall search, listen or download tracks. These recommendations are specific for users.

2.1.8. SITE ADAPTATION REQUIREMENTS

We are going to study with big data on this project. However, our local computers restrict us and we cannot deal with big data. Therefore, we need to deploy our project on a web server.

2.2. PRODUCT FUNCTIONS

Recommendation System is composed of the following fundamental features:

Users:

Generate Data

Get Recommendation

Interagent:

Get Recommendation

Provide Dataset

Update Dataset

Integrate Web Service

2.3. CONSTRAINTS

Recommendation systems suffer from three major problems; cold start, huge data and sparsity. Cold start is a potential problem in computer-based information systems which involve a degree of automated data modeling. Specifically, it concerns the issue that the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information. There are basically two types of this problem. The first one occurs when a new user starts using the system. The system knows little about their preferences and it is necessary to pick some training points for rating so that the system begins learning what the user wants [4]. The second type occurs when a new product is introduced to the system. Again it is essential to rate this item in order to quickly improve the prediction accuracy about it [4].

The next problem is huge data. In many of the environments that these systems make recommendations in, there are millions of users and products. It is quite a challenge to produce high quality recommendations and perform many recommendations per second for millions of customers and items. Thus, a large amount of computation power is often necessary to calculate recommendations. Moreover, at the beginning of the project, we are going to study on our local computers. We are not going to manage with huge data because we have limited memory. After we start using server, we are going to practice with big data.

The third problem is sparsity. Users that are very active contribute to the rating for a few number of items available in the database and even very popular items are rated by only a few number of users available in the database. Because of sparsity, it is possible that the similarity between two users cannot be defined, rendering collaborative filtering useless. The rating matrix R , which is mentioned in the previous section, might be full of missing entries since many users vote for just a few of the items.

2.4. ASSUMPTIONS AND DEPENDENCIES

Our end product is planned to be a REST-compliant Web service. There must be Internet connection.

3. SPECIFIC REQUIREMENTS

3.1. INTERFACE REQUIREMENTS

The user interface of our recommendation system has already been created and used in larger music streaming and downloading system. Therefore, we won't implement any user interfaces but our recommendation system can be modified in larger system and related interfaces. We only implement a simple user interface for showing system recommendations. This user interface gives chance to display recommendations before integration of web-service and existing music streaming and downloading web application. In main system, user logins first and starts playing and downloading tracks which are created our recommender web service's data. Therefore, Output of our recommendation system is shown in integrated main web application's interface.

3.2. FUNCTIONAL REQUIREMENTS

This section encapsulates the major software functions and data flow among the participants of the system. The participants include the user and the interagent. The user, in other words the end-user, is the person that makes use of music streaming and downloading web application. The interagent is the R&D team of ARGEDOR and this actor provides the necessary tools to maintain the connection between the user and our system. The overall use case diagram is illustrated in Figure 5.

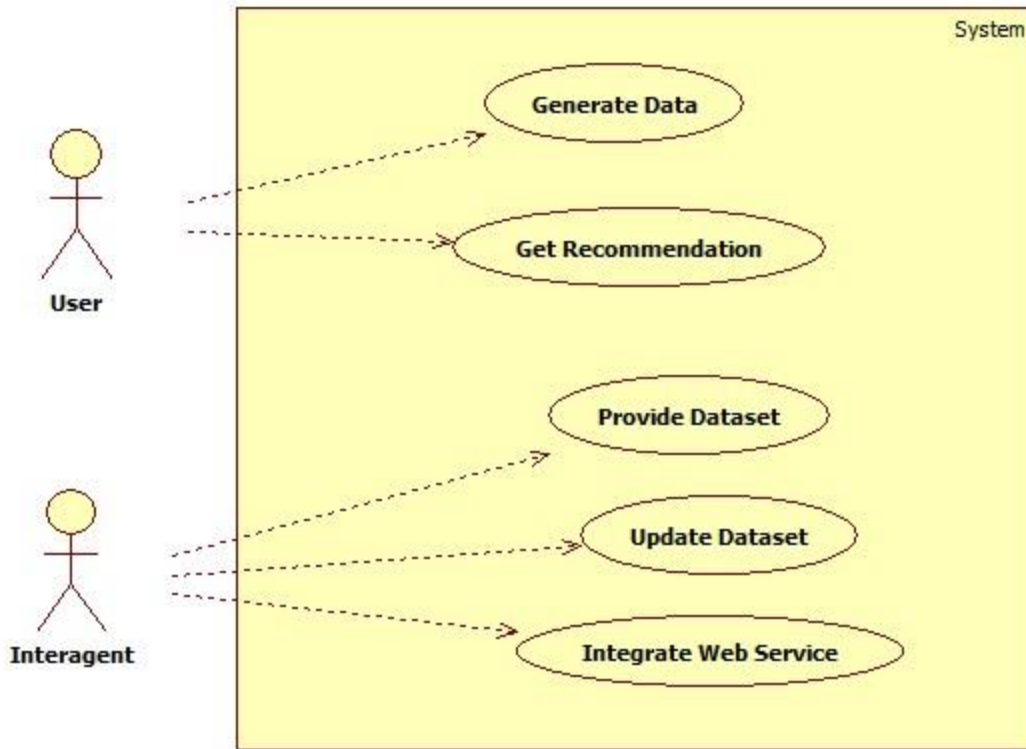


Figure 5: Use Case diagram

3.2.1. USER USE CASES

3.2.1.1. USE CASE: GENERATE DATA

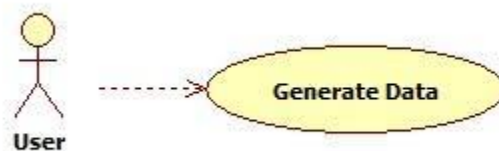


Figure 6: Generate Data

User can listen or download track from music streaming and downloading web application. Track information is collected according to album information, artist information, time of action, user information, rating value and channel. This information will fill the database.

Every track has unique album, artist, time of action, user, rating value and channel. So, this listening process will generate practical data (Figure 6).

3.2.1.2. USE CASE: GET RECOMMENDATION



Figure 7: Get Recommendation

System can suggest track(s) as a recommendation to user based on the dataset which is refined by users' collaborative approach. The main function of our system shows these tracks based on recommendation algorithms. When a user chooses recommendation part in application, he/she will get the most important point of the project recommendation and project gives user a chance to choose track through recommended tracks according to his/her own previous choices (Figure 7).

3.2.2. INTERAGENT

3.2.2.1. USE CASE: PROVIDE DATASET



Figure 8: Provide Dataset

Interagent provides dataset, in other words the big data, in cooperation with music streaming and downloading application (Figure 8).

3.2.2.2. USE CASE: UPDATE DATASET



Figure 9: Update Dataset

Music streaming and downloading web application collects 1 million data every day. Interagent also delivers this 1 million data to our web service. These updates are necessary for making more accurate recommendations (Figure 9).

3.2.2.3. USE CASE: INTEGRATE WEB SERVICE

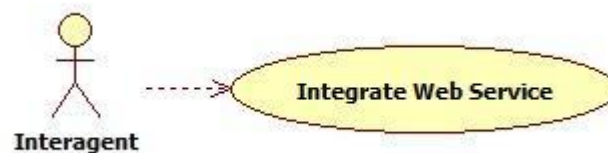


Figure 10: Integrate Web Service

After the recommendation system project is completed, inter agent integrate this web service to music streaming and downloading web application. Then, users will access to our web service and receive recommendations through the web application (Figure 10).

3.3. NON-FUNCTIONAL REQUIREMENTS

3.3.1. PERFORMANCE REQUIREMENTS

Performance of making recommendation and updating this recommendation is very important issue because we are aiming to make the system real-timed. In other words, the system should have enough speed that users of the system cannot realize the processing of data. In order to make system real-timed, at the end of listening track or after purchasing track system shall update recommendations. Besides, our web service should handle multiple users at the same time.

3.3.2. SECURITY

Database has to be reached securely and its data should not be broken. It also should not change except interagent updates. Moreover, since our dataset contain some personal information of user such as userid, tracks he/she listened, security design is important in the web service.

3.3.3. USABILITY

The scope of the product is widespread. The only requirement is using music streaming and downloading web application. Besides, people from every age shall easily use the system.

3.3.4. PORTABILITY

The role of Web services in enterprise application integration scenarios is to make it easier to tie applications running on heterogeneous platforms together; to help them overcome the communication gaps that arise from decisions to use one development environment over another; and to help abstract such choices so that business developers no longer have to keep track of what operating system or what development environment or what technology decisions have been made [2]. Since the project we work on is a web service, it is portable.

3.3.5. SAFETY

No safety requirements have been identified.

3.3.6. OTHER REQUIREMENTS – GNU GPL LICENSE

The project will be released under GNU General Public License, which allows the use, change and redistribution of the software freely. The philosophy of this license is applied to this recommendation system to achieve the greatest possible use and development by public.

4. DATA MODEL AND DESCRIPTION

4.1. DATA DESCRIPTION

4.1.1. DATA OBJECTS

This subsection of the document explains system's classes and their relations with each other. Most of the system functionalities are represented in Figure 11.

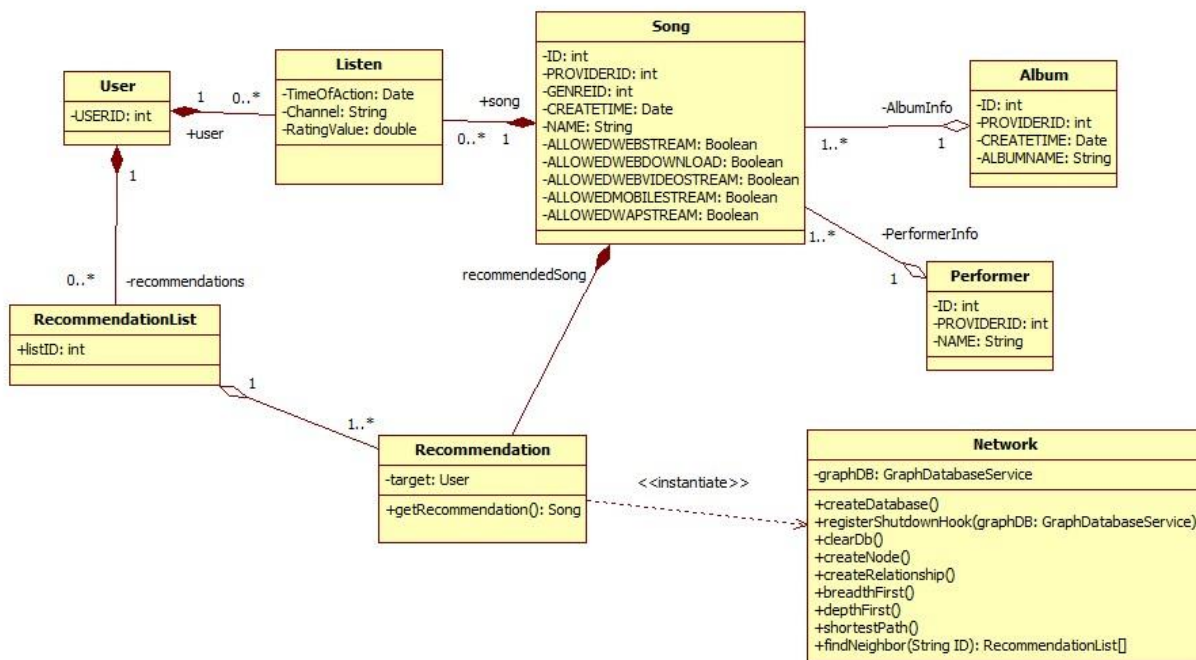


Figure 11: Class Diagram

User: This class represents the user entity and it keeps a unique id.

Song: This class represents the song entity and it has the fields defined in the previous section. It has an aggregation association with Album and Performer classes. It contains

AlbumInfo and *PerformerInfo* fields. An *Album* instance or a *Performer* instance can be mapped to one or more *Song* instances.

Album: This class represents the album entity and it has the fields defined in the previous section.

Performer: This class represents the performer entity and it has the fields defined in the previous section.

Listen: This class is a group of instances that represent listening action. Each time a new log is processed, an instance of this class is created. Its fields are defined in the previous section. This class has composition association with *User* and *Song* classes. It contains *user* and *song* fields.

Network: This class encapsulates database related methods and it keep a *GraphDatabaseService* object which is the *Neo4j* database object. It has some specific methods such as *createDatabase*, *registerShutdownHook*, *clearDb*, *createNode*, *createRelationship* methods to construct/remove a database and add/delete nodes. The similarity construction requires the traversal of the graph database which is done via *depthFirst*, *breadthFirst* and *shortestPath* methods. The similar user and recommended songs are determined via *findNeighbor* method.

Recommendation: This class represents the recommendation object to be returned to the external system. It contains either a cluster of users that are found similar or a cluster of songs. The user to which this recommendation will be given is kept inside *target* field. This class has a composition association with the *Song* class and contains *recommendedSong* field. It also has an aggregation association with *RecommendationList* class. Whenever an instance of *RecommendationList* is deleted, the corresponding instances of *Recommendation* class still exist.

RecommendationList: This class represents the group of recommendations given to a specific user. It has a unique id. This class has a composition association with *User* class.

4.1.2. DATA DICTIONARY

Term	Definition
userID	User ID are not same with real ones, company change them in order to provide security and it is unique
songID	songID is used for specific track
albumID	albumID represents a unique album containing tracks
artistID	This attribute specify the artist uniquely
timeOfAction	timeOfAction shows the date and time which user download or listen track
ratingValue	This attribute is estimated according to that user download or listen track
channel	Channel means that user listening track from browser, him/her own list or another user list

5. BEHAVIORAL MODEL AND DESCRIPTION

This section provides the overall behavior of the system. Major events and states are displayed in a state chart diagram.

5.1. DESCRIPTION FOR SOFTWARE BEHAVIOR

The state chart in Figure 14 illustrates the behavior of the Recommender System within a larger system. The larger system is a stand-alone music application that our software product is going to be integrated to. The recommender system is shown as an independent sub state. The states that remain outside the Recommender state are out of the scope of this project. However, these states and related events are necessary for our software product to proceed. The pre states

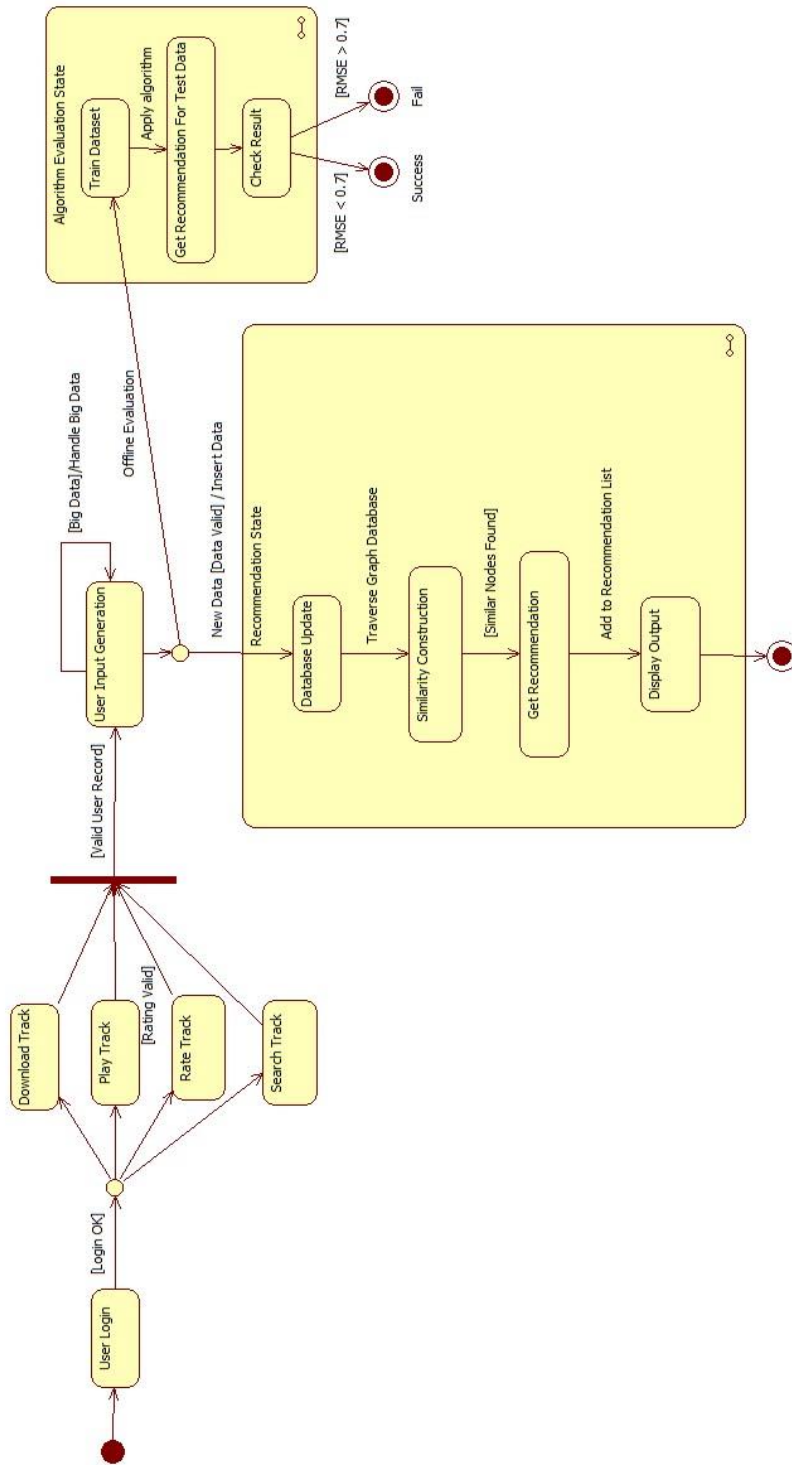
and post states of the Recommender state are assumed to be completely encountered. The host application takes user input through the downloaded, played, rated and search track information. Records for each user are formed and they are supplied to our Web Service. The behavior of our Web Service mainly resides on database transactions. Since the project team uses graph database, the system traverses the large graph, which is actually the storage and notation method of the big user data, to find out the closest match to a user or to a track. After evaluating how successful this match is, the recommendation is presented to the user.

5.2. STATE TRANSITION DIAGRAMS

The state chart diagram in Figure 13 shows all the states, triggers and conditions for each transition and related events. The trigger induces a state change given that the condition is satisfied. The conditions are written inside the square brackets.

As soon as the system is initialized, User Login state is triggered. It is followed by the login action. If the login information is valid, the user is directed to the actions involving tracks. The user can either download a track, play a track, rate a track or search a track. These states are the main sources for user data. The transition from these states is a synchronized action in order to obtain a complete user data. If the user data is valid, User Input Generation state is initiated. This state has a self-transition due to the big data condition. The project relies on the assumption that the project model is consistent with big data. It is indicated by the Handle Big Data event. As long as there are millions of users and tracks in the system, this state is instantiated again. When the dataset is completed, the actual Recommendation sub state is triggered. The arrival of new data triggers the database update on the condition that the new data are valid. The insertion event occurs at this point. New data are inserted into the graph database. New nodes and edges are formed inside the huge graph. The next state is Similarity Construction. The event related to this transition is the traversal of graph database. Instead of writing queries in a relational database, the recommender system requires traversing the graph nodes to find the most similar users. Get Recommendation state is the most crucial part of this set of behaviors. The closest track(s) according to the previous preferences of a user is/are determined in this state. As the last state, the output is displayed and the Recommender State is finalized.

In the Algorithm Evaluation State, the algorithm is tested on the user data in offline mode. Some part of the data is used for training while the rest is used for testing. RMSE (Root mean square error) is used in order to evaluate the accuracy of the system. It is a measure of how accurate the recommended songs are according to the actual logs. The threshold value for RMSE is currently fixed as 0.7. If the error gets higher, the system fails.



6. PLANNING

6.1. TEAM STRUCTURE

The four members of the Dcengo Unchained team are Duygu Kabakcı, Işinsu Katırcıoğlu, Sila Kaya, Koray Kocakaya. The team structure is as follows:

Task	Members
Graph Traversal Algorithm Component	Duygu Kabakcı, Işinsu Katırcıoğlu
Neo4j Database Component	Duygu Kabakcı, Işinsu Katırcıoğlu
Neo4j API Component	Duygu Kabakcı, Işinsu Katırcıoğlu
Machine Learning Algorithm Component	Sıla Kaya, Mehmet Koray Kocakaya
UI Component	Sıla Kaya, Duygu Kabakcı
Weka API Component	Sıla Kaya, Mehmet Koray Kocakaya
Evaluator Component	Mehmet Koray Kocakaya, Işinsu Katırcıoğlu
Recommender Component	Sıla Kaya, Mehmet Koray Kocakaya

6.2. ESTIMATION

Estimation Date	Task
04.10.2013	Deciding Project
13.10.2013	Project Idea Proposal
22.10.2013	Elevator Pitch
30.10.2013	Software Requirements Specification (SRS)
30.10.2013 - 07.11.2013	Researching Requirement Systems and Tools

07.11.2013 - 14.11.2013	Trying Software Tools
14.11.2013 - 18.11.2013	Documenting Design Report
18.11.2013	Design Report
18.11.2013 - 04.12.2013	Trying Software Tools
04.12.2013 - 16.12.2013	Documenting Updated Reports
16.12.2013	Updated Reports
16.12.2013 - 20.01.2013	Implementing Base Recommendation System

6.3. PROCESS MODEL

We will apply agile model for our recommendation system so that system can respond quickly to changing requirements without excessive rework. Agile method is based on an iterative approach, each iteration involves planning, requirements analysis, design, implementation, testing. Each iteration takes approximately four weeks. Once we will generate the initial version of recommendation system, then our system will be developed according to accuracy of recommendations, performance results on scaled big data.

7. CONCLUSION

This Software Requirement Specification document is prepared to give requirement details of the project “Recommender System”. The detailed functional and nonfunctional requirements, system, user, software and hardware interfaces, data and behavioral model are stated in an extended outline. This document will be helpful at constituting a basis for design and development of the system to be developed.

This page intentionally left blank