RoommateSync
Final Report

Rim Benhallam

Supervisor:
Asmaa Mourhir

Coordinator:
Dr. Yassine Salih-Alj

Spring 2016
RoommateSync

Capstone Final Report

Approved by the Supervisor:

Prof. Asmaa Mourhir
ACKNOWLEDGEMENTS

First, I would like to express a special gratitude and my deepest thanks towards my supervisor Prof. Asmaa Mourhir for her guidance and arrangement of all facilities to carry out my capstone project. I would like to acknowledge her crucial role for investing all of her time and efforts, despite being busy with her duties, in giving me the necessary pieces of advice, supervision and support in achieving the project goals.

Furthermore, my sincere thanks and appreciation go to Dr. Yassine Salih Alj who provided us with the necessary motivation and valuable pieces of advice.

Last but not least, my deepest gratitude and love go to my parents for their unconditional support and attention throughout my life.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ................................................................................................................. 2
1. LIST OF FIGURES .......................................................................................................................... 4
2. ABSTRACT .................................................................................................................................. 5
3. INTRODUCTION ............................................................................................................................ 6
3. STEEPLE ANALYSIS ....................................................................................................................... 8
   SOCIAL ........................................................................................................................................... 8
   TECHNOLOGICAL ......................................................................................................................... 8
   ECONOMIC ................................................................................................................................... 8
   ENVIRONMENTAL ......................................................................................................................... 8
   POLITICAL ................................................................................................................................... 8
   LEGAL ........................................................................................................................................... 8
   ETHICAL ....................................................................................................................................... 9
4. SOFTWARE ENGINEERING METHODOLOGY .............................................................................. 9
5. REQUIREMENTS ENGINEERING ................................................................................................ 10
6. TECHNOLOGICAL FEASIBILITY STUDY .................................................................................... 13
7. METHODOLOGICAL APPROACH ............................................................................................. 18
8. SOFTWARE ARCHITECTURE ......................................................................................................... 27
9. SOLUTION DESIGN ..................................................................................................................... 31
10. IMPLEMENTATION ..................................................................................................................... 33
11. TECHNOLOGICAL ENABLERS .................................................................................................. 43
12. CONCLUSION AND FUTURE WORK ....................................................................................... 45
13. SOURCES ................................................................................................................................... 46
## 1. LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>RoommateSync Use-Case Diagram</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Partner and Vukotic’s experiment on finding extended friends [4]</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Comparison between REST and SOAP [5]</td>
<td>16</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Algorithm Flowchart</td>
<td>26</td>
</tr>
<tr>
<td>Figure 5</td>
<td>RoommateSync Software Architecture</td>
<td>29</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Modeling Data with Property Graph for RoommateSync</td>
<td>31</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Neo4j Data Visualization</td>
<td>32</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Calculating compatibility between users from one side</td>
<td>35</td>
</tr>
<tr>
<td>Figure 9</td>
<td>matching score for common music likes</td>
<td>35</td>
</tr>
<tr>
<td>Figure 10</td>
<td>matching score for common movie likes</td>
<td>36</td>
</tr>
<tr>
<td>Figure 11</td>
<td>matching score for common sport likes</td>
<td>36</td>
</tr>
<tr>
<td>Figure 12</td>
<td>RoommateSync Login</td>
<td>37</td>
</tr>
<tr>
<td>Figure 13</td>
<td>User’s Profile</td>
<td>39</td>
</tr>
<tr>
<td>Figure 14</td>
<td>RoommateSync Quiz</td>
<td>41</td>
</tr>
<tr>
<td>Figure 15</td>
<td>User’s potentials roommates</td>
<td>42</td>
</tr>
</tbody>
</table>
2. **ABSTRACT**

The purpose of this project is to offer an effective and new way for matching between roommates not only based on users’ common answers to the roommate questionnaire but also based on mutual interests extracted from social network such as Facebook.

The approach for matching between roommates will have two major phases. The first phase is concerned with using OKcupid’s, one of the best American based international online dating websites, method of matching with a little improvement, as a basis for pairing between roommates who provided common answers to the roommate questionnaire. The second phase focuses on consuming services from Facebook social network as a mean to match between roommates based on common interests such as music, movies, and sports. The final matching percentage between each two users is calculated by combining the sub matching scores.

RoommateSync’s idea is to change the typical traditional way of matching between roommates while addressing the issue of conflicts that might arise between them.
3. INTRODUCTION

As the semester begins, housing selection for roommates draws near and students are considering who they will be paired with. There is no doubt that having a good or bad roommate experiences can impacts positively or negatively on student’s college life. In order to provide students with a smooth integration to college life, it is inherent for them to live in a relaxing environment free of anxiety and struggle in order to achieve academic success.

College rooming is an important aspect for student’s social well-being for many reasons. First, roommate relationships are unique in a way they experience living together and thus being able to come to compromises and agreements. Second, sharing accommodation with the wrong person might result in personality mismatches. Much of the growing body of research and studies done in universities are exploring the impact of roommates on students’ academic performance and social-well functioning. In fact, in a survey organized by the American College Health Association, 5.6% of undergraduate students have reported that roommate conflicts had a negative impact on their academic performance resulting in receiving an incomplete grade or dropping the course [13].

Most of universities rely on the use of a lifestyle questionnaire tool as a simple mean to match between roommates. However, such approach does not seem to be efficient for two main reasons. First, the roommate matching quiz does not allow students to define themselves accurately as questions are more oriented towards a ‘yes’ or ‘no’ answers only. Second, students get very lazy when it comes to answering questionnaires.
Thus, in this work we attempt to reduce the widespread tendency of roommate conflicts through a mobile app that pair up between students based on different criteria with a minimal intervention of users, that we will be referring to as RoommateSync.

RoommateSync aims at tackling the above mentioned limitations by: (i) enhancing the roommate questionnaire; with each question the student has the ability to specify the level of the question’s importance; specify an answer and how he/she would like the other person to answer; (ii) offering an effective way to match between potential roommates with whom students will be able to get along with; it tries to answer as much questions as possible with little intervention of students by extracting personal data such as preferences and lifestyles from their Facebook profiles.
3. STEEPEL ANALYSIS

SOCIAL

The social impact of RoommateSync is to bring together students with common lifestyles and preferences as roommates relationship plays an important role in student’s social functioning within the university. Finding the potential roommate does not only result in a smooth transition from dependent teenagers to responsible adults but also marks a positive experience.

TECHNOLOGICAL

The application uses the latest techniques of Hybrid mobile app development that allows, unlike native mobile development, the support of many mobile platforms with the use of web technologies such as JavaScript, HTML, and CSS.

ECONOMIC

The application has no direct economic implications.

ENVIRONMENTAL

It can influence people behaviors towards publishing soft copies to encourage friendly environmental activities.

POLITICAL

The application will have a slight political impact as it will decrease the likelihood of roommates’ fights by matching them based on common interests and lifestyles.

LEGAL

The application will not violate any copyright law. The application will conform to the standards of open source software.
ETHICAL

Regarding the ethical impact, the application will preserve the privacy of students by protecting the user’s extracted data from Facebook social network. Personal information and interests will not be disclosed to third parties. All of their information is only stored in our database for matching purposes and are never used or shared outside this scope.

4. SOFTWARE ENGINEERING METHODOLOGY

RoommateSync follows the prototyping model of software development to allow the client to better understand the requirements of the system and thus ensuring the design of a user-friendly application with working functionalities. Several prototypes were developed during the project’s timespan where each new prototype is an improvement of the previous one. The first prototype was only a mobile template without any working functionality. The second prototype allowed matching between users based on common answers to the quiz without including the matching based on common interests extracted from Facebook social network. The final prototype combined between the matching based on common answers and the matching based on common interests to produce a complete system.
5. REQUIREMENTS ENGINEERING

This section includes all the functional and nonfunctional requirements for RoommateSync Project.

5.1 Functional Requirements:

a. Authentication:
   i. The user should be able to authenticate to the mobile App through Facebook.
   ii. Currently, users who do not have a Facebook account should be redirected to Facebook’s login page to create it.
   iii. Users can delete their RoommateSync account associated with their Facebook accounts.

b. Roommate Quiz:
   i. The user shall be able to answer a set of questions of “a roommate match” quiz.
      1. The user shall specify how he/she would like the other to answer.
      2. The user shall choose the level of importance of each question.
      3. The user shall submit the data.

c. User Profile:
   i. Users can view their own RoommateSync profiles.
      1. Users can see their own interests and social data extracted from Facebook:
         a. Personal Information (Profile picture, name, birthday, email, hometown, gender)
         b. Favorite music genre.
c. Favorite sport teams.

d. Favorite movies types.

e. Level of sociability.

ii. Users can view RoommateSync profiles of matched roommates.

   1. The user should be able to view common interests and social data:

      a. Overall matching percentage
      b. Quiz matching percentage
      c. Music matching percentage
      d. Movies matching percentage
      e. Sports matching percentage

   d. User Communication:

      i. The user should be able to chat with potential roommates to decide on which roommate
         he/she will choose, through our application.

5.2 Non Functional Requirements:

The mobile application should have:

- A user friendly interface that allow users to interact with the mobile application in an
  intuitive way.

- User-event response should be processed without any delay.

- Application should be lightweight in terms of both size and resources.

- The application should be compatible with both Android and IOS devices.

- The application should preserve the privacy of users and conform to the standards of
  open source software.
5.3 Use-Case Diagram:

Figure 1: RoommateSync Use-Case Diagram
6. TECHNOLOGICAL FEASIBILITY STUDY

The feasibility study will consist in identifying the viability of the project and deciding upon which sources of data, approach for profile matching as well as technologies will be used.

6.1 Front End client:

RoommateSync is a mobile application. Developing a mobile application in a native Android/IOS language and then deploying it to another platform is not often a straight-forward process and requires rebuilding the code in the target’s platform language from the beginning. In general, porting from one mobile platform to another is becoming a critical challenge for developers as they want to leverage the same design and content for different operating systems and device features. However, a viable solution to build a mobile application for all platforms is to use cross-platforms mobile development frameworks. There are many solutions for creating hybrid mobile apps that run on different mobile platforms using only the following web technologies HTML, CSS, and JavaScript. The cross platform framework that we chose is Apache Cordova [2].

6.2 Server side:

The server side is developed under Nodejs which is an open source, cross platforms JavaScript environment for building real time and scalable applications [11]. JavaScript is normally confined to run on browsers and access web pages only, but with the use of google chrome’s V8 JavaScript engine it can run on server side which allows it to listen for POST, GET,
PUT, DELETE HTTP requests. Nodejs is also an asynchronous event driven framework that performs I/O operations asynchronously while running under a single thread. It is able to execute a number of operations asynchronously with the use of a single thread instead of spawning off multiple threads that will consume a lot of memory. Nodejs derives its speed with the use of event loop and asynchronous callbacks [9]. Nodejs keeps track of an event listener and an event loop. Whenever there is a blocking I/O event, Nodejs sends it to the event loop to handle it and then continues the execution of the rest of the program. The event loop returns the event of the completed asynchronous operation to the event listener to get executed.

**6.3 Data Storage:**

Concerning the choice of the database, graph databases seem to be a better choice for such a project. For several years, relational databases have been popular. This latter infers relationship between entities by referring to other primary key attributes via foreign-keys columns. However, such a model does not seem to be a good fit for some specific use-cases such as traversing a social network in an efficient and convenient way [3]. Despite the term “relational”, relational databases do not excel when it comes to storing and expressing data connections between tables as they require the use of special properties such as foreign keys or out-of-band processing like map-reduce [4] and the use of recursive joins in hierarchies make the queries more complex. In addition, an attempt to represent a model that deals with rich structured relationships in a tabular framework does not conform to the way business stakeholders think of storing and processing data. On the other hand, representing connections as a graph is in accord with the way people intuitively model data using diagrams [4].
The graph database model that started operating in the market since 2003 [4], has the following characteristics:

- Connected entities called nodes that contain attributes (key-value pairs).
- Node can have more than one label.
- Relationships between nodes are named, directed, have a starting and ending node, and can also contain attributes.

Graph database allows for modeling very rich domains in a very flexible and intuitive way in the same way our brain stores information. Moreover, in terms of response times, graph databases get faster than relational databases when data grows in volume. Figure 2 shows a comparison between the relational and graph databases when seeking to find friends-of-friends in a social network at different depth levels. The results strongly shows that graph databases are substantially quicker for connected data [4]. As shown in Figure 2, Neo4j’s, one of the world’s leading graph databases, execution time is only 1.359 seconds against 1543.505 for relational databases in depth 4.

![Figure 2: Partner and Vukotic’s experiment on finding extended friends [4]](image-url)
The reason for this difference in performance lies in how data is stored. In fact, with the use of an index-free adjacency where nodes point directly to their inbound and outbound relationships which in turn point to other nodes, and so on, processing of large data sets become faster [4]. This way millions of records are traversed in a single second.

6.4 Client Server Data Communication: REST and JSON

There exist various web services available for data exchange between the client and server. The most common web services access protocols are REST (Representation State Transfer) and SOAP (Simple Object Access Protocol). REST is an API that relies on the HTTP protocol to interact with content on remote systems whereas SOAP is not tied up to one particular transport protocol [5].

Figure 3 outlines where REST outplays SOAP:

<table>
<thead>
<tr>
<th>No.</th>
<th>SOAP</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>SOAP is a protocol.</td>
<td>REST is an architectural style.</td>
</tr>
<tr>
<td>2)</td>
<td>SOAP stands for Simple Object Access Protocol.</td>
<td>REST stands for Representational State Transfer.</td>
</tr>
<tr>
<td>3)</td>
<td>SOAP can’t use REST because it is a protocol.</td>
<td>REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.</td>
</tr>
<tr>
<td>4)</td>
<td>SOAP uses services interfaces to expose the business logic.</td>
<td>REST uses URI to expose business logic.</td>
</tr>
<tr>
<td>5)</td>
<td>JAX-WS is the Java API for SOAP web services.</td>
<td>JAX-RS is the Java API for RESTful web services.</td>
</tr>
<tr>
<td>6)</td>
<td>SOAP defines standards to be strictly followed.</td>
<td>REST does not define too much standards like SOAP.</td>
</tr>
<tr>
<td>7)</td>
<td>SOAP requires more bandwidth and resource than REST.</td>
<td>REST requires less bandwidth and resource than SOAP.</td>
</tr>
<tr>
<td>8)</td>
<td>SOAP defines its own security.</td>
<td>RESTful web services inherits security measures from the underlying transport.</td>
</tr>
<tr>
<td>9)</td>
<td>SOAP permits XML data format only.</td>
<td>REST permits different data format such as Plain text, HTML, XML, JSON etc.</td>
</tr>
</tbody>
</table>

Figure 3: Comparison between REST and SOAP [5]
Unlike SOAP that is more suitable for exchanging documents and heavy-weight payloads, REST is a lighter weight alternative as it does not have to use complex XML nor suffer from huge XML payloads to make requests and receive responses. A valuable advantage of REST over SOAP is that it offers better performance and scalability as it can use HTTP caching mechanisms while SOAP reads cannot be cached [5]. Thus, with caching, server load is reduced and better user experience is provided with fast service calls [5].

Besides the above reasons, I have chosen REST as it allows the user to obtain the output in different forms that are easier to read and parse such as JSON and XML. JSON syntax is derived from JavaScript and it is being less verbose than XML. Also, it is frequently used in AJAX Web applications and allows high scalability and better support for browser clients.

6.5 RESSOURCES EXTRACTION

The application relies heavily on the answers provided by users in the roommate matching quiz provided by the mobile app as well as common interests acquired from data sources such as Facebook. The Graph API [1] is a library that helps with extracting information from Facebook social graph.
7. METHODOLOGICAL APPROACH

The matching algorithm is meant to provide users with the most relevant potential matches by a combination of two different matching aspects. The first part matches between users based on common answers provided in the quiz section. The second part pairs between users based on common preferences and lifestyles extracted from their Facebook profiles.

7.1 Matching users using the quiz

Regarding the first part of the algorithm, OKCupid’s approach is used as a basis for calculating the matching percentage between roommates [6] but with a slight improvement. OKCupid is one of the best American based international online dating websites.

The OKCupid matching percentage between people is based on three values:

- The user’s answer.
- How the user wants the other user to answer.
- The importance or weight of the question.

Each level of importance is assigned a numerical weight (Table 1), and then depending on each user’s answers to the three values, a percentage score will be calculated between each two users representing their compatibility. The compatibility between user A and user B is found by calculating the points that user B scored for user A and the points that A scored for B in all questions. The score from each side is calculated by looking at how each user’s expected answer satisfies the actual answer of the other user. If a user’s expected answer matches the actual answer of the other user then the score would be w/w where w is the weight of the user’s question. If they do not match then the score would be 0/w. Once both scores are calculated for
all questions, all the scores that a user scored from the perspective of another user are added and the geometric mean is applied on both users’ total score to get the compatibility between each two users. Eq.1 shows how the geometric mean combines both total scores by taking the square root of their product.

\[(\prod_{i=1}^{n} x_i)^{1/n} = \sqrt[n]{x_1 x_2 \ldots x_n}\]  (1)

<table>
<thead>
<tr>
<th>Level of importance</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Important</td>
<td>250</td>
</tr>
<tr>
<td>Important</td>
<td>50</td>
</tr>
<tr>
<td>Somewhat Important</td>
<td>10</td>
</tr>
<tr>
<td>A little Important</td>
<td>1</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: OKCupid’s scale of importance

For instance, let’s consider the following example to illustrate how the compatibility between two users is calculated.

Example Questions:

**Question 1:** How organized are you?

**Question 2:** Do you like to be the center of attention?
Table 2: Users’ answers

B’s compatibility with A:

User A indicated that the other’s answer to question 1 is very important while the answer for question 2 is a little important. By summing up the weights for all questions of user A, the maximum points a user can earn are 251 (250 + 1). Of those 251 points, user B scored 250/250 for question 1 since B’s answer matches A’s expected answer, but B scored 0/1 for question 2. Thus, B’s compatibility with A is 99.6% (250/251).

A’s compatibility with B:

User B indicated that the other’s answer to question 1 is a little important while the answer for question 2 is somewhat important. Thus, the maximum points a user can earn are 11 (10 + 1). Of those 11 points, user A scored 0/1 for question 1 since B’s expected answer does not match A’s answer, but A scored 10/10 for the second question. Thus, A’s compatibility with B is 91% (10/11).

<table>
<thead>
<tr>
<th>Question</th>
<th>Value</th>
<th>User A</th>
<th>User B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User’s answer</strong></td>
<td>User’s expected answer</td>
<td>Value of question</td>
<td></td>
</tr>
<tr>
<td>Question 1</td>
<td>Very Organized</td>
<td>Average</td>
<td>Average</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>Average</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Very important</td>
<td>A little important</td>
<td></td>
</tr>
<tr>
<td>Question 2</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A little important</td>
<td>Somewhat important</td>
<td></td>
</tr>
</tbody>
</table>
Finally, the matching percentage between both users is calculated by applying the geometric mean of the percentage scores, which is in this case \( \sqrt{99.6\% \times 91\%} \) about 95%.

OKCupid’s uses a geometric mean instead of an arithmetic mean as it can give a meaningful average when comparing between match percentages based on multiple properties. It makes also sense when two people satisfying each other 50% - 50%; this should be considered a better match than two others who satisfy 0% and 100%. If the arithmetic mean was used in this case, then it would have given the same matching score which is logically wrong.

OK cupid’s matching method is quite interesting as it tries to maximize the likelihood of satisfying the user’s preferences by collecting three values. However, OKCupid’s uses Boolean values (match, does not match) and does not allow for partial points to be earned by users when there is no perfect match between how a user wants the other one to answer and the actual answer of that user. Thus, we propose a solution to this problem by assigning partial credits to users depending on how far their answers are from the expected answer by a user. This is first achieved by dynamically assigning different weights to possible answers depending on the expected answer of the user. Most of the quiz questions have four possible answers and, thus, the possible points a user can earn out of a weight \( w \) are \( w, \frac{w}{2}, \frac{w}{4} \) or 0 where \( w \) is the weight of the question. A score of \( \frac{w}{w} \) would be achieved if a user’s expected answer matches the actual answer of another user.

The possible answers of each question are ordered from a positive answer to a negative answer. For instance, “Very organized”, “organized”, “average”, and “messy” would have 1, 2, 3, and 4 as IDs respectively.

To illustrate how partial points are distributed dynamically when a user’s expected answer does not match the actual answer of another user, we will consider the previous example. In question
1, User B’s expected answer is “average” while A’s actual answer is “very organized”. User B would have preferred that the other user answered “average” as well. Depending on the user’s expected answer, a preferred list is constructed with a full score (w) as a first value when the expected answer of a user matches another user’s actual answer (Table 3). The preferred list for B is ordered depending on how close other users’ answers are to his/her expected answer (Table 3.3). In other words, if an “average” answer is not available, then the closest answer would be an “organized” answer with w/2 as a score, else the third choice would be a “very organized” answer with a score of w/4 and so on.

<table>
<thead>
<tr>
<th>Id</th>
<th>Answer</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Very Organized</td>
<td>w</td>
</tr>
<tr>
<td>2</td>
<td>Organized</td>
<td>w/2</td>
</tr>
<tr>
<td>3</td>
<td>Average</td>
<td>w/4</td>
</tr>
<tr>
<td>4</td>
<td>Messy</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-1 Preferred list for “Very organized”
Expected answer

<table>
<thead>
<tr>
<th>Id</th>
<th>Answer</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Organized</td>
<td>w</td>
</tr>
<tr>
<td>1</td>
<td>Very Organized</td>
<td>w/2</td>
</tr>
<tr>
<td>3</td>
<td>Average</td>
<td>w/4</td>
</tr>
<tr>
<td>4</td>
<td>Messy</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-2 Preferred list for “Organized”
Expected answer
Table 3: Preferred order of answers depending on a user’s expected answer

### 7.2 Matching users using Facebook profiles

The second part of the algorithm matches between users based on common interests extracted from Facebook social network. Such extracted interests include music, movies and sports likes.

The compatibility between users is calculated by first finding matching percentages between them for each extracted interest. The matching percentage is calculated by dividing the intersection of each interest likes over the union of likes between each two users as shown by the Eq. 2:

$$\text{score} = \frac{A \cap B}{A \cup B}$$  \hspace{1cm} (2)
To illustrate this, if we consider that user A has 30 music likes, 20 movies likes, and 25 sport likes and user B has 15, 20, and 30 respectively in the same interests, and their common likes are 10, 12, 15 respectively, then the matching percentage between both users for each field would be the following:

Matching percentage in music field: \( \frac{10}{35} = 28.57\% \)

Matching percentage in movies field: \( \frac{12}{28} = 42.85\% \)

Matching percentage in sport field: \( \frac{15}{40} = 37.5\% \)

### 7.1.3. Aggregation of scores

To aggregate between these matching percentages and the one based on common answers for the quiz, we use the weighted mean (Eq. 3). The weighted mean is an average where each quantity is assigned a weight to determine its relative importance and is used also to find an average based on different percentages values [15].

\[
\bar{X} = \frac{w_1X_1 + w_2X_2 + \cdots + w_NX_N}{w_1 + w_2 + \cdots + w_N} = \frac{\sum_{j=1}^{N} w_jX_j}{\sum_{j=1}^{N} w_j}
\]  

(3)

Finally, the top highest final matching percentages will be taken to display the potential roommates for each user.

Figure 4 summarize the main parts of the algorithm.
Figure 4: Algorithm Flowchart
8. SOFTWARE ARCHITECTURE

8.1 MVC model

The RoommateSync will be developed using the Model-View-Controller paradigm. MVC model is a software design pattern that separates a software application into three main interconnected parts so as to isolate the application logic from the way information are presented to the end user and promote code reusability [7].

![MVC Diagram](image)

**Figure 5: Model-View-Control paradigm [10]**

As shown in figure 5, MVC model is made up of the following parts:

The model is responsible for managing application data and defines the business logic that manipulates the data. When a model object is updated, the controller is notified to change the view objects [8].
The view represents how the core functionality is rendered into a suitable form to the end user. It is also responsible for communicating user changes to the model through the controller.

The controller sits between the view and the model and is responsible for performing the following operations:

1. Extracting input from the end user.
2. Verifying input.
3. Sanitizing input.
4. Converting input.
5. Selecting the model to use.
6. Invoking the model.
7. Getting the results.
8. Selecting and invoking the appropriate view to use to display information back to the user.
8.2 RoommateSync Software Architecture:

Figure 6 shows the software architecture of our application.
On client side the application is developed with web technologies such as HTML5, Javascript, and CSS using Apache Cordova framework [2]. AngularJS is used as a framework to extend HTML5 for declaring dynamic views that HTML falters in. It was used also to provide the Model-View-Control on client side.

The client side that relies on Facebook graph API to get the access token for the login, issues http requests to the RESTful server to send different actions from within different controllers and parses the JSON outputs that will be sent back by the server.

On the other hand, the server-side communicates with Facebook API as an extra security feature and get the user profile information and interests from this latter. The server side is developed under Nodejs. The server side communicates with Neo4j database to get the requested data through its models that are invoked by the controller.
9. SOLUTION DESIGN

9.1 data model

Figure 7 graph represents the property graph for RoommateSync.

Figure 7: Modeling Data with Property Graph for RoommateSync
As shown in figures 7 and 8, the data is stored as nodes, attributes or edges.

The user node (blue) has the id, name, gender, birthday, picture, and sociability as attributes. The sociability defines if users are sociable or not by looking at the rate at which they comment and like in Facebook.
The question node (red) has the qid and the question as attributes.

The question node is linked to answer nodes (gray) that defines the question’s possible answers. The relationship is denoted by “OPTION”.

The “ANSWERED” relationship between the user and question nodes has the following attributes: answer, expected, and the weight. The answer is the actual answer of the user and the expected is the answer that the user would like the other user to answer with.

The user is also linked with MusicGenre (purple), MovieGenre (pink), and SportTeam (yellow) nodes that define his/her interests as “LIKES” relationships.

The “MATCH” relationship between users represent their matching percentages in different areas namely the matching percentage in questions (Q%), music (M%), movies (Mo%), and sports (S%). The final match is denoted by F%.

The message node sits between each two users and has the Message_Content and Created_Time as attributes.

10. IMPLEMENTATION

10.1 Server-side

The following code calculates Compatibility1 which represent the matching score between two users from one side based on common answers to questions of the quiz.
function CalculateCompatibility1(id) {
    defer = $.defer();
    runCypherQuery("MATCH (p:Person { name: 'Riz Benhallam' }) +
    "with p"
    "MATCH (u:Person)"
    "WHERE u <> p "
    "with u, p"
    "MATCH (p) -[r:ANSWERED]->(q:Question) where q.qid = '2' or q.qid = '3'
    "with u, p, r, toFloat(r.weight) as w, q"
    "MATCH (u) -[r2:ANSWERED]->(q:Question) where q.qid = '2' or q.qid = '3'
    "with r.exp as e, q2.answer as a, w, q, u"
    "with"
    "CASE"
    "WHEN e = '1' +
    "THEN"
    "CASE"
    "WHEN a = '1' +
    "THEN w" +
    "WHEN a = '2' +
    "THEN (w/2)"
    "WHEN a = '3' +
    "THEN (w/4)"
    "WHEN a = '4' +
    "THEN 0"
    "END"
    "WHEN e = '2' +
    "THEN"
    "CASE"
    "WHEN a = '1' +
    "THEN (w/2)"
    "WHEN a = '2' +
    "THEN w"
    "WHEN a = '3' +
    "THEN (w/4)"
    "WHEN a = '4' +
    "THEN 0"
    "END"
    "WHEN e = '3' +
    "THEN"
    "CASE"
    "WHEN a = '2' +
    "THEN (w/2)"
    "WHEN a = '3' +
    "THEN w"
    "WHEN a = '1' +
    "THEN (w/4)"
    "WHEN a = '4' +
    "THEN 0"
    "END"
}
Figure 9: Calculating compatibility between users from one side

The following code calculates the matching percentages between users based on common music likes.

```javascript
function CalculateMusicCompatibility(id) {
  let defer = $.defer();
  runCypherQuery("MATCH (p:Person { id:'"+id+'" }) "+
"MATCH (u:Person) "+
"WHERE u <> p "+
"OPTIONAL MATCH (p)-[r:LIKES]->(m:MusicGenre)<-[r:LIKES]-(u) "+
"WITH u.id as uid ,u, count(r) as c, collect(DISTINCT m.name) AS MusicGenre, p "+
"MATCH (m:MusicGenre)<-[r1:LIKES]-(p) "+
"WITH uid, u, count(r1) as b, c, MusicGenre "+
"MATCH (x:MusicGenre)<-[r2:LIKES]-(u) "+
"WITH uid, b,count(r2) as total, c, MusicGenre "+
"RETURN uid, b, total,(toFloat(c)/toFloat (b-total-c)) as MusicScore, c, MusicGenre ",function (err,result) {

  if(err) defer.reject("err");
  else defer.resolve(result);
  // console.log(result.results[0].data[0].row);

});
return defer.promise;
}
```

Figure 10: matching score for common music likes
The following code calculates the matching percentages between users based on common movies likes.

```javascript
function CalculateMovieCompatibility(id){
    defer = $.defer();
    runCypherQuery("MATCH (p:Person { id:'+id+' }) +
    "MATCH (u:Person) +
    "WHERE u <> p "+
    "OPTIONAL MATCH (p)<-[r:LIKES]-(m:MovieGenre) <-[r:LIKES]-(u) "+
    "WITH u.id as uid_u, u, count(r) as c, collect(DISTINCT (m.name)) AS MovieGenre, p "+
    "MATCH (m:MovieGenre)<-[r:LIKES]-(p) "+
    "WITH uid_u, count(r) as b, c, MovieGenre "+
    "MATCH (x:MovieGenre)<-[r:LIKES]-(u) "+
    "WITH uid, b, count(r) as total, c, MovieGenre "+
    "RETURN uid, b, total, (toFloat(c)/toFloat(b+total-c)) as MovieScore, c, MovieGenre ",function (err,result){
        if(err) defer.reject("err");
        else defer.resolve(result);
        // console.log(result.results[0].data[0].row);
    });
    return defer.promise;
}
```

**Figure 11:** matching score for common movie likes

The following code calculates the matching percentages between users based on common sport likes.

```javascript
function CalculateSportCompatibility(id){
    defer = $.defer();
    runCypherQuery("MATCH (p:Person { id:'+id+' }) +
    "MATCH (u:Person) +
    "WHERE u <> p "+
    "OPTIONAL MATCH (p)<-[r:LIKES]-(s:SportTeam) <-[r:LIKES]-(u) "+
    "WITH u.id as uid_u, u, count(r) as c, collect(DISTINCT (s.name)) AS SportTeam, p "+
    "MATCH (s:SportTeam)<-[r:LIKES]-(p) "+
    "WITH uid_u, count(r) as b, c, SportTeam "+
    "MATCH (x:SportTeam)<-[r:LIKES]-(u) "+
    "WITH uid, b, count(r) as total, c, SportTeam "+
    "RETURN uid, b, total, (toFloat(c)/toFloat(b+total-c)) as SportScore, c, SportTeam ",function (err,result){
        if(err) defer.reject("err");
        else defer.resolve(result);
        // console.log(result.results[0].data[0].row);
    });
    return defer.promise;
```

**Figure 12:** matching score for common sport likes
10.2 Client-side

Screenshots:

The login uses Facebook authentication.

Figure 13: RoommateSync Login
The following figure depicts the user’s profile extracted from Facebook.
Figure 14: User’s Profile
The following figure is the questionnaire that users need to answer.
Figure 15: RoommateSync Quiz
The following figure shows the list of a user’s potential roommates with the highest scores.

**Figure 16: User’s potentials roommates**
11. TECHNOLOGICAL ENABLERS

Apache Cordova:
Cordova is an open source platform for developing mobile applications using standards web technologies such as HTML5, CSS3, and JavaScript. Apache Cordova applications runs within different wrappers of different platforms and has access to native APIs of the device [2].

AngularJS:
AngularJS is a JavaScript framework maintained by Google and Angular community that extends HTML vocabulary to simplify the declaration of dynamic views in the application. It also provides the Model-View-Control pattern for the client side using JavaScript and HTML where the view is defined in HTML and the model and controller are defined in Javascript [12].

JQuery:
JQuery is a fast and rich cross platform Javascript library intended to simplify the client-side scripting of HTML, handling events, selecting DOM elements and developing AJAX applications [16].

Facebook Graph API:
The Facebook Graph API is a platform intended for third party developers to build applications that access data from the social network. It allows accessing users profile information and social connections from Facebook’s social graph. It is a low level HTTP based API used to perform different operations [1].
Nodejs

Nodejs is a powerful Javascript runtime that runs on Google’s V8 Javascript engine environment to allow developing server-side applications. Nodejs has an event-driven architecture that triggers a non-blocking event which makes it lightweight and efficient in handling thousands of concurrent users [11].

Neo4j

Neo4j, one of the world’s leading graph databases [4], is a highly scalable graph database management system implemented in Java and accessible from software written in other languages using the Cypher Query Language. It leverages data relationships as a core aspect of its data model to query connections in an efficient way [14].
This capstone project was a whole new experience for me as I had to deal with completely new technologies and tools I was not familiar with before. I dived into the world of graph databases in addressing challenges we face today, in terms of data complexity and size. This project has been a great means for me to learn how to develop in Node.js for server-side and implement the Model-View-Control as an architectural pattern to separate concerns and optimize the code reuse. In addition, I also learnt how to integrate the application with a very popular third party service to enhance the typical matching questionnaire with interests and lifestyles extracted from Facebook social network. I also learnt how to combine AngularJS along with HTML5, javascript, jquery and css web technologies for the front-end development and design to build native mobile apps for different platforms using Apache Cordova framework.

As a future work, the application will be hosted online using google app engine which is a platform for building scalable applications through its built-in auto scaling and offers a great support for load balancing, health checks, and security scanning to list only a few [17].
13. SOURCES


