Database Management System

Final Project Report

Team 4

Design of Friend Search System

Team Member:
Weiqing Li
Guohui Huang
Hang Yin
Dekun Geng

Supervisor:
Prof. Rodica Neamtu

December 7, 2017

Abstract

We developed an on-line freind search system where users can find their ideal mate by setting proper range of age, height, education level and select preferred locations, hobbies, movies preference, music preference and lifestyle.

The system was implemented using Django Web Tool and we chose SQLite as the database management tool. We finally achieved following functions in our demo: user log in & out, users' profile editing and management, searching the database with multiple queries, sending friend request and in-mail, display of a list of friends on users' profile.

Keywords: Database, RightOne, Django, SQLite, Cache

1 Overview

The system was called "RightOne", a web application which allows users to find their ideal mates or friends. Users are supposed to be able to sign up with their existing Google, Facebook, Github or Twitter account. After logging in, they would be allowed to create and edit the profile. The friend searching function would be sharding based on users' specific preference such as sex, hobbies (music or movie type, sports), age, location, education background, career etc. Once they got a list of users' name that satisfied all the conditions, they could view these profiles and decide whether to send the friend request or just follow them. They even could send in-mails to introduce themselves and thus increase the possibility that their friend request to be accepted.

We hope to build such a relatively complete system where users could explore and chat with their ideal friends that filtered by their specific preference because nowadays people are tired of networking and social with strangers. We suppose such website would be a platform where people sharing similar hobbies could know each other and make up the gap between strangers.

2 Background Material

2.1 Data Set We Use

Our Testing dataset comes from Kaggle community, and the details of data sets show as the following:

The data file (responses.csv) consists of 1010 rows and 150 columns (139 integer and 11 categorical).

For further convenience, the original variable names were shortened in the data file. See the columns.csv file if you want to match the data with the original names.

- The data was collected from participants in electronic and written form.
- All information were of Slovakian nationality, aged between 15-30.

The variables can be split into the following groups:

- Music preferences (19 items)
- Movie preferences (12 items)
- Hobbies & interests (32 items)
- Phobias (10 items)
- Health habits (3 items)
- Personality traits, views on life, & opinions (57 items)
- Spending habits (7 items)
- Demographics (10 items)

Some details about specific items:

MUSIC PREFERENCES

1. Slow paced music 1-2-3-4-5 Fast paced music (integer)
2. Dance, Disco, Funk: Don't enjoy at all 1-2-3-4-5 Enjoy very much (integer)
3. Folk music: Don't enjoy at all 1-2-3-4-5 Enjoy very much (integer)
4. .....

MOVIE PREFERENCES

1. I really enjoy watching movies.: Strongly disagree 1-2-3-4-5 Strongly agree (integer)
2. Horror movies: Don't enjoy at all 1-2-3-4-5 Enjoy very much (integer)
3. Thriller movies: Don't enjoy at all 1-2-3-4-5 Enjoy very much (integer)

4. .....

HOBBIES & INTERESTS

1. History: Not interested 1-2-3-4-5 Very interested (integer)
2. Psychology: Not interested 1-2-3-4-5 Very interested (integer)
3. Politics: Not interested 1-2-3-4-5 Very interested (integer)
4. .....

HEALTH HABITS

1. Smoking habits: Never smoked - Tried smoking - Former smoker - Current smoker (categorical)
2. Drinking: Never - Social drinker - Drink a lot (categorical)

SPENDING HABITS

1. I save all the money I can.: Strongly disagree 1-2-3-4-5 Strongly agree (integer)
2. I enjoy going to large shopping centres.: Strongly disagree 1-2-3-4-5 Strongly agree (integer)
3. I prefer branded clothing to non branded.: Strongly disagree 1-2-3-4-5 Strongly agree (integer)
4. .....

DEMOGRAPHICS

1. Age: (integer)
2. Height: (integer)
3. Weight: (integer)
4. Gender: Female - Male (categorical)
5. Highest education achieved: Currently a Primary school pupil - Primary school - Secondary school - College/Bachelor degree (categorical)
6. I am the only child: No - Yes (categorical)
7. Location(string)

2.2 Tools Referred To

Since we aim to build a web application, so that components of Client (Users with Browser) will send requests via HTTP, thus we need to implement a back-end server to response such queries with needed information and show with elegant view(HTML format,parsed by browser). On the other hand, in

order to handle data storage and SQL queries, we shall implement a proper database management system. Finally we decided to develop the web application with Django web tool and HTML, also we will implement the database with SQLite, because it fits well with Django and such database management system is proper for our 500k test data set.

Django takes care of much of the hassle of Web development, so we can focus on writing our application with no need to reinvent the wheel. It's also free and open source.

2.3 Django using MVC design patterns

Model-View-Controller (MVC) is a software development approach that separates the definition of code and the methods (models) of data access from the request logic (controller) and the user interface (view). This design pattern key advantage is that the various components are loosely coupled in. In this way, each Django-driven Web application has a clear purpose and can be independently changed without affecting other parts. For example, developers change the URL of an application without affecting the underlying implementation of the program. Designers can change the style of an HTML page without touching Python code. The database administrator can rename the datasheet and change only one place without having to find and replace from a bunch of files.

3 Our Solution Approaches

3.1 Data Clean

Our database system aimed at friend searching. Consequently, personal information is the main data storing in our data basement, including some basic information and preference in several fields. We first designed ER model[Figure.1] of personal information and implement it in SQLite. In term of ER model, we create users, hobbies, health habits and movie entity respectively. The specific description of each entity is given as following:

Users:
User Identity: unique subject number and primary key;
username: unique subject;

firstname: char;
lastname: char;
password: char.

Profiles:
nickname: char;
into: char;
photo: file;
Location: the geographic position when users finish their information on our website. We use the abbreviation of each state in America to normalize this attribute;
Age: age of user, integer;
Height: height of user, three-bit integer;
Weight: weight of user, three-bit integer;
Gender: gender of user, string;
onlychild: if the person is the only child of the family, the value is TRUE, otherwise, False,bool value;
Education: educational status of user, normalized by providing several selections. It's a categorical variable, we use 0,1,2,3... to save it.

Music:
genre: 18 genres of music, containing Fast Songs, Dance, Folk, Country, Classical music, Musical, Pop, Rock, Metal or Hardrock, Punk, Hiphop and Rap, Reggae and Ska, Swing and Jazz, Rock n roll, Alternative, Latino, Techno and Trance and Opera.

Movie:
genre: 11 genres of movie, including Horror, Thriller, Comedy, Romantic, Sci-fi, War, Fantasy/Fairy tales, Animated, Documentary, Western and Action.

Hobbies:
name: 42 fields are provided for user, including history, psychology, politics, mathematics, physics, internet, PC, economy management, biology, chemistry, art exhibitions, musical instruments, dancing, countryside, cars, religion, reading, foreign languages, medicine, law, geography, science and technology, fun with friends, shopping, theatre, adrenaline sports, active sport, writing, gardening, celebrities, passive sport, spiders, rats, snakes,

ageing, dangerous dogs, flying, pets, storm, darkness, heights and fear of public speaking. Yes, we try our best to include all the fields as much as we can.

Userfeatures:

user: Foreign Key referenced User. Also the primary key;

enjoymusic: int. This person's rating of if he/she like music, 1 is totally uninterested and 5 is extremely obsession;

enjoymovieint. This person's rating of if he/she like movie, 1 is totally uninterested and 5 is extremely obsession;

musicloved Many-to-many Field. It contains all the kinds of music this person rates ¿3. Django automatically generate a table to store this attribute. The auto-generate table contains user(foreign key referenced User) and music(foreign key referenced Music);

musichated Many-to-many Field. Similar to musicloved. This time we select all the kinds of music this person rates¡3;

movieloved Many-to-many Field. Similar to musicloved. This time we perform it on movie;

moviehated Many-to-many Field. Similar to musichated. This time we perform it on movie;

hobbies Many-to-many Field. Similar to musichated. It includes all the hobbies this person rates¿3.

3.2 System Design

we came up with a system design UML Diagram (Figure 1) and analyzed users' demand. In order to achieve these functions, we chose Django web tool to develop the website framework and SQLite as the database management tool.

When applying Python Django as a web developer tool, the tuples of entities' relationship will be transferred to some instance of classes, so we are supposed to create corresponded class. The objects' description of the system are as follows:

Server Object (Backend)

The function of the server class is to create links between users' different information such as profile, message mailbox, friend list, etc. It also plays

an important role as a response to browser's request.

Account Object

Users need to sign up first and then log into the website. So the account object can be used for identity validation. The information of the account includes user's email address, user name and password.

User profile Object

The user profile object will be created whenever a new account object is created. Such object contains all the information that will show up in the user's profile such as users' gender, height, weight, hobbies etc.

Message Object

The message object will be created when a user send a new message to other users/friend and the object will adds the message information to database with target user's id. When a user checks his or her inbox, a list of messages that were sent to the user will be shown in descending order.

Friend Object

This is the most important feature of our system. It will allow users to add and delete friends. Users can select any query condition such as sex, range of age, height or location in the search bar at the top of their home page. The database will be queried for users' information that satisfy all the conditions.

3.3 Database Design

As a friend search system that may serve thousands of people, we need to build a logical database management system to manage users' information, the relationship between them and the messages that they sent to each other within the website. So we designed the ER model as figure 2, figure3 and figure 4.

3.4 Problems we encountered

At first we tried to manage our database using Oracle, however when meeting with the situation that we have to build many to many relationships, there will be unacceptable redundance, so we turned to SQLite. However, when

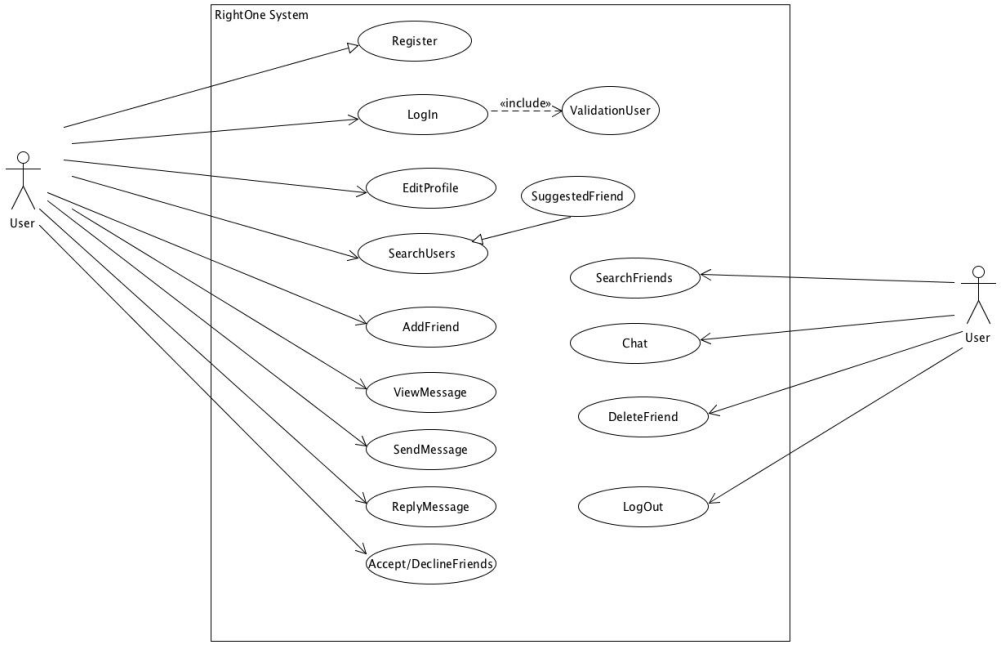


Figure 1: System Design UML Diagram.

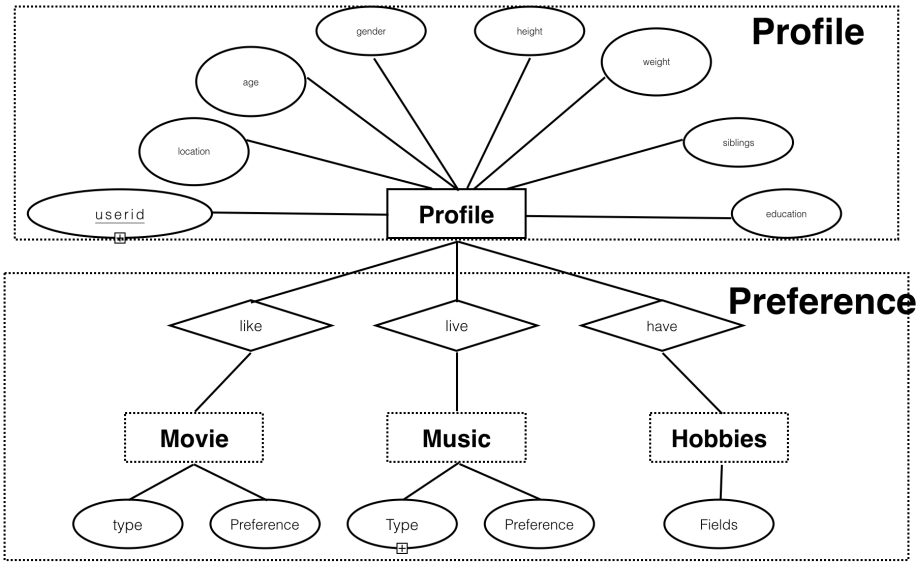


Figure 2: ER Model Diagram for Profile.

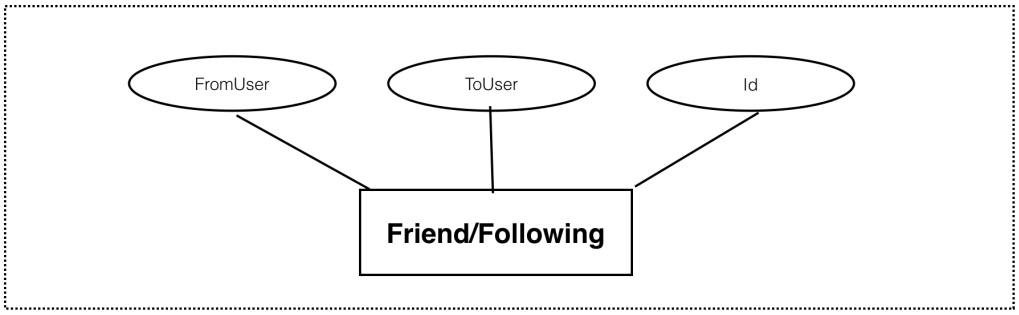


Figure 3: ER Model Diagram for Friend Relation.

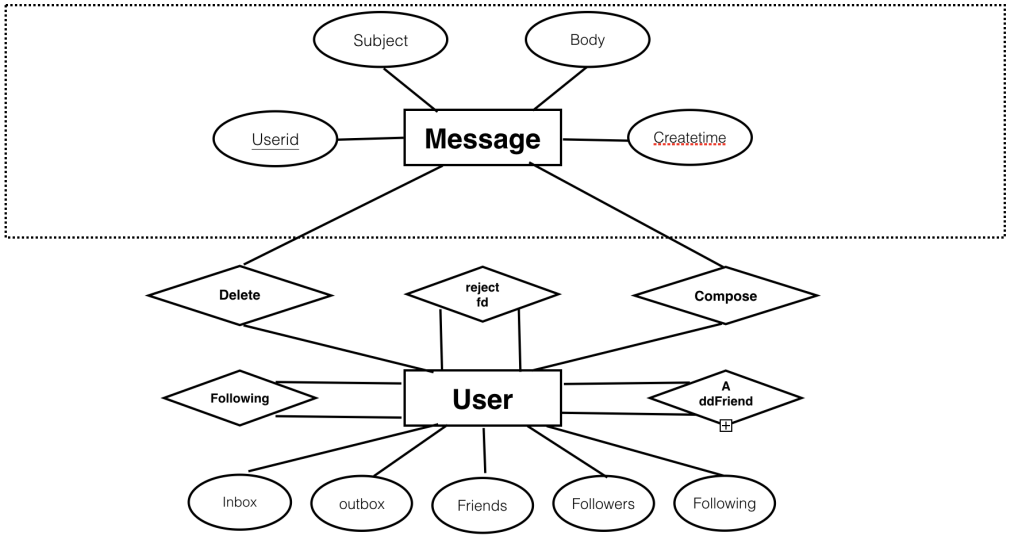


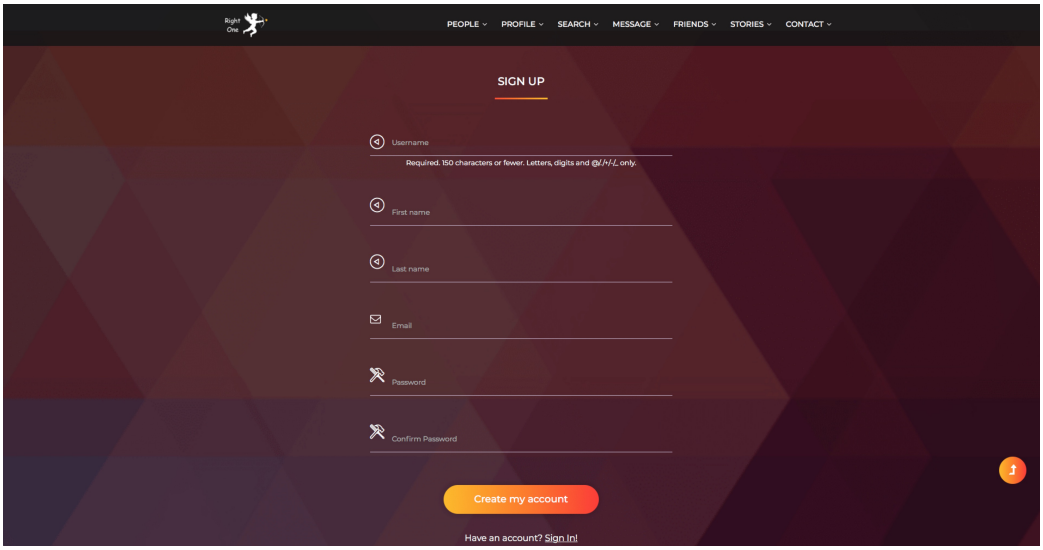
Figure 4: ER Model Diagram for Message System.

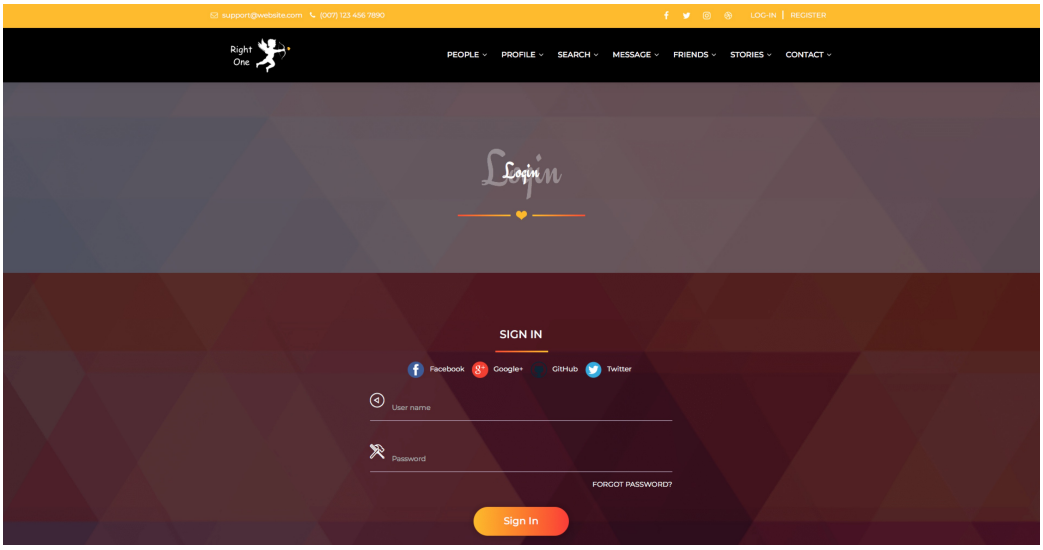
one query turned to have over 1000 results there will be an warning, so we may need to look for a database that can handle such big data load.

3.5 Validation of our approach

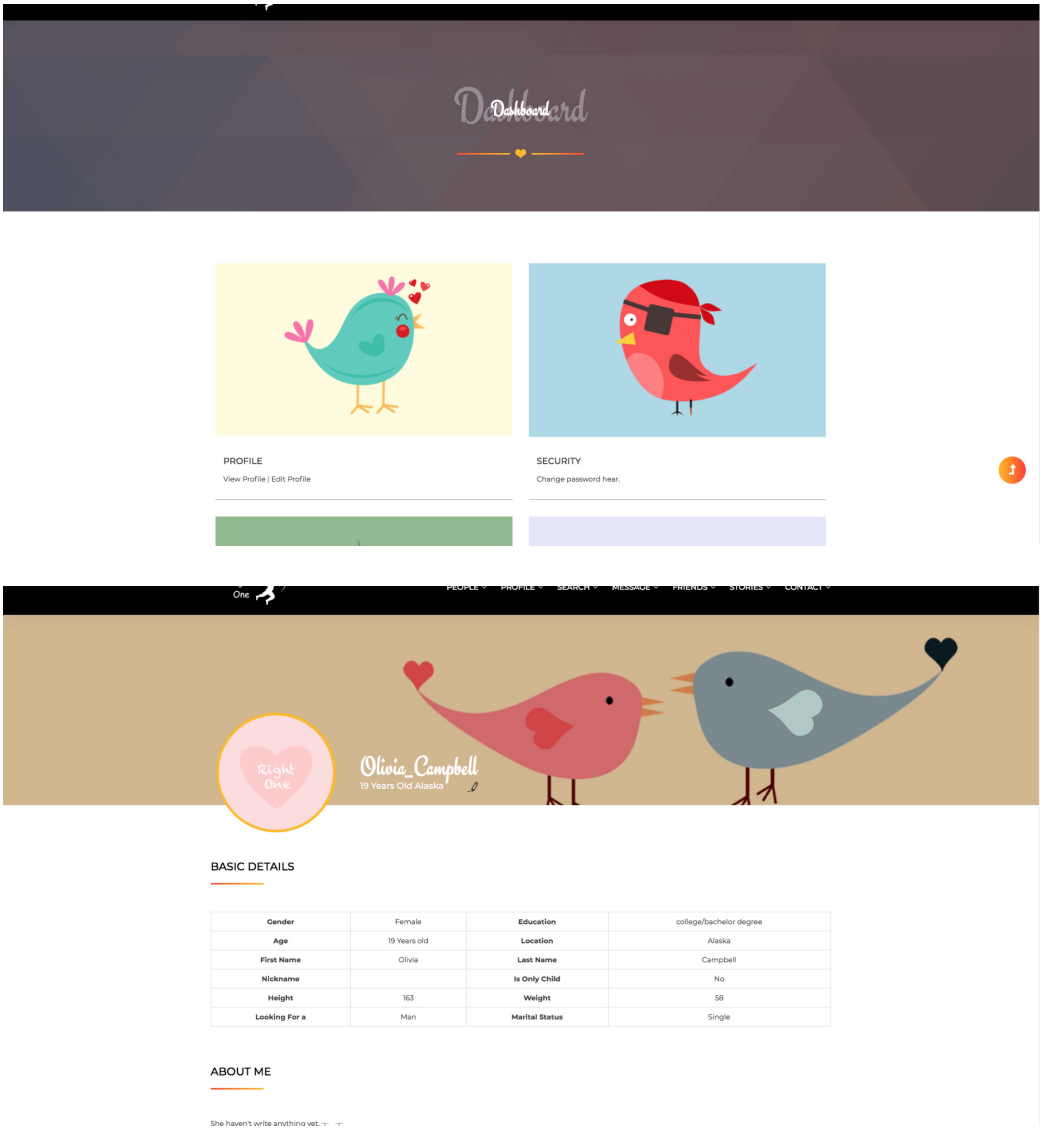
We first downloaded a 500k dataset including more than 1000 users' information and a result of a survey about their hobbies. Then we implemented data clean process and transfered the numerical values of hobbies to categorical values. Finally we added user id and location for every tuple of information manually and imported these information into our system. Now let us check if our system really works as expected.

We start from sign up and log in to the website:





Then we will see our dashboard and be able to edit our own profile:

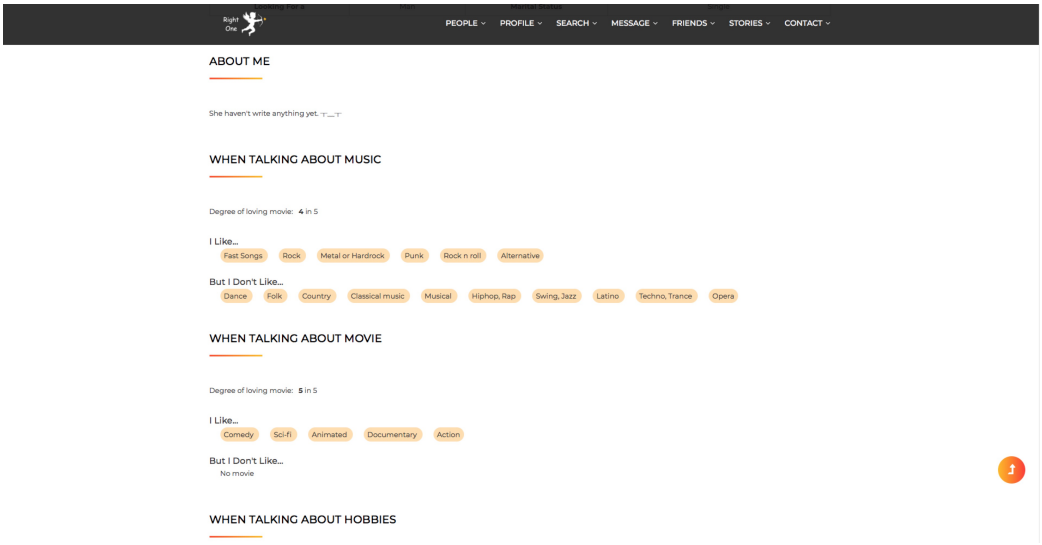


BASIC DETAILS

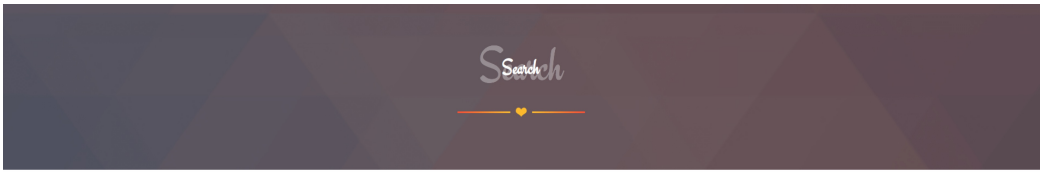
Gender	Female	**Education**	college/bachelor degree
Age	19 Years old	**Location**	Alaska
First Name	Olivia	**Last Name**	Campbell
Nickname		**Is Only Child**	No
Height	163	**Weight**	58
Looking For a	Man	**Marital Status**	Single

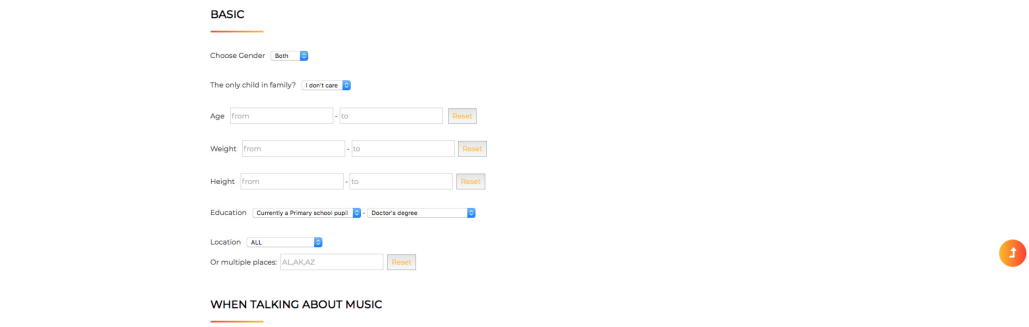
ABOUT ME

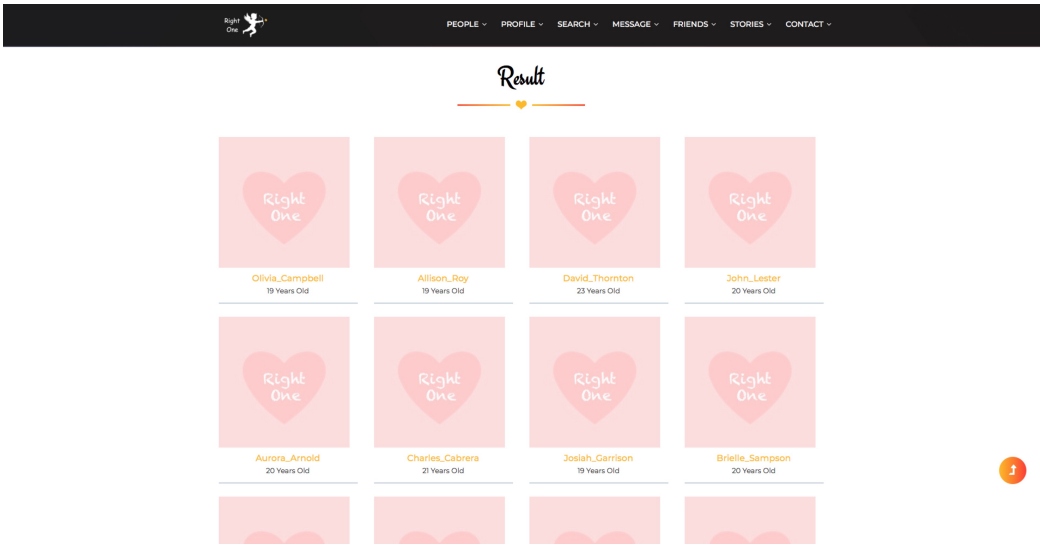
She haven't write anything yet.



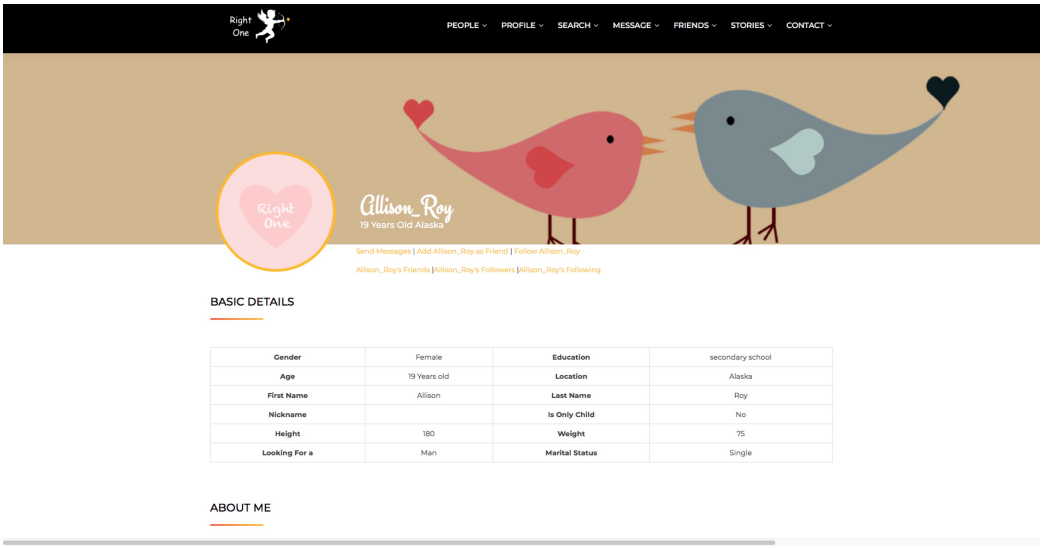
Next we can click on the friend search button and set up our preference to find the ideal mate:



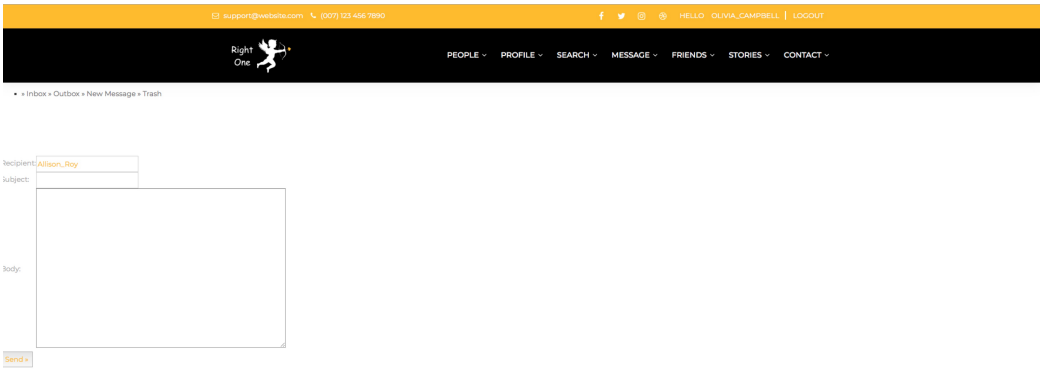




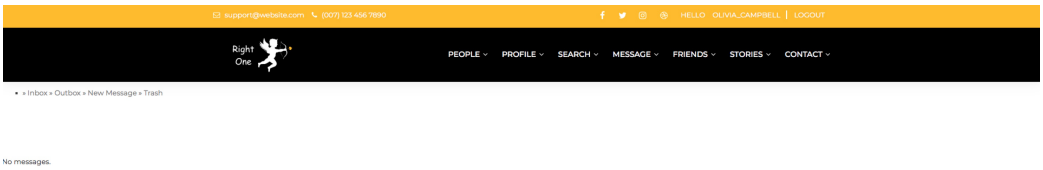
After we get a list of users that satisfy all the conditions, we will have the access to check their profile and decide whether we want to add them as friend or just follow them:

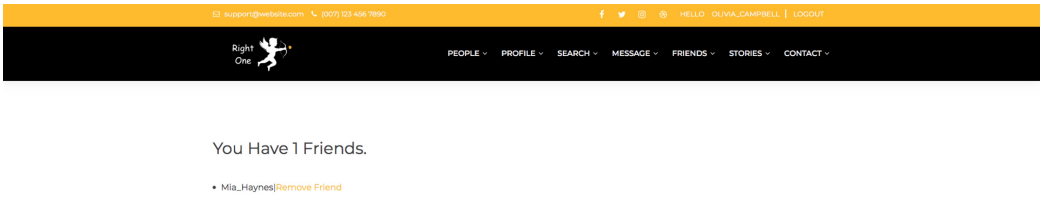


We can even send a message to our targeted users and introduce ourselves to increase the possibility that our request won't be rejected:



Ok, now let's log out the current account and log in to the account that we just sent friend request and message to and see what will happen. Let's check his mailbox and friend request list, we will see that all the requests and messages have been displayed properly, he can now decide whether to accept or reject these requests and reply to the message directly using our message system:





Awesome! We really achieved it!

4 Lessons Learned

Guohui Huang:

The development of a web application requires interdisciplinary background knowledge. We have to focus on details of implement and system design, especially when processing such large data volume. Django MVC pattern is a creative design pattern, thus decoupling the system (in this project, we decouple the view, or present of html format with the implement of querying data) reduces the workload of re- design, especially when we are dealing with frequently changing functional requirements.

Django uses class and object to present data model ,which I think is more efficient and intuitive to present the data. Using class to model data enable us to create methods where we process the Data in-depth.

Weiqing Li:

A simple idea needs a lot of effort to make it comes to a product which can be showed to public. In our mind, the DDL and DML sentence are quite simple. But when it comes to the user interaction, things could become very complicated. Django was literally a new thing for me. I worked hard on the basic knowledge of building a website with Django, including both back-

end and the front-end. A second thing is that I have learned more about the Many-to-Many Field, including how to generate, how to get many tuples that belong to a certain user. To some extent, Django is very easy to do with these "many-to-many filed". Also, as a webpage developping tool, Django makes it very easy to do with the query.

Hang Yin:

Since I was responsible for the implement of database, it would be a good chance to practice my skill learned in DBMS class. We need to deal with over 2000 tuples of data sets saved as csv format where most of the values showed as range and contain nearly 30 categories. It is really a challenge to clean the unnecessary data and import the data into database. Also I have learned that using classed data to implement the dataset is more convenient since we can create methods to operate the data.

Dekun Geng:

Due to the lack of experience in developing a real social contact website product, we started from scratch. We first applied the user requirement analysis and discussed the approach data sets to be clean and the ER diagram to be designed. Then we learned the python web developer tools to build a server for connecting the database and the front end. Also we really applied what we learned from DBMS class to our product. It was really a challenge for all of us, however fortunately, we achieved it.

5 Member Contribution

Task	Member	Efforts
Front End Implement	Weiqing Li & Guohui Huang	Developed the web page
Search Function	Guohui Huang & Weiqing Li	Achieved the friend search function
System Design	Guohui Huang & Dekun Geng	Implemented requirement analysis
Database Design	Hang Yin & Dekun Geng	Data clean and designed ER model
Database Implement	Hang Yin & Weiqing Li	Managed data with SQLite
Servers Implement	All team members	Implemented classes and objects

6 Conclusions

With the efforts of our team members, we achieved almost all the functions expected in our system including user log in & out, users' profile editing and management, searching the database with multiple queries, sending friend request and in-mail, display of a list of friends on users' profile. We even imported a 500k dataset into the system and test its validation. However, we failed when we tried to build a real-time chatting function because it was more complicated than we thought and the time for this task is really limited. To achieve such function, we need to build both push and pull services and build a cache system to store the chatting content. We may make it if more time were given.

In a nutshell, we achieved 95% of the functions when we first designed such system and they all work properly as expected.

7 Future Work

We may optimize the query time by changing the data structure and improve the searching function. Also we can try to develop a real-time chatting function if time allows.

8 Appendixes

8.1 SQL DDL Statements

Our SQL DDL are as follows:

```
CREATE TABLE friendship_follow (
    id           INTEGER  NOT NULL
                          PRIMARY KEY AUTOINCREMENT,
    created      DATETIME NOT NULL,
    followee_id INTEGER  NOT NULL
                          REFERENCES auth_user (id),
    follower_id INTEGER  NOT NULL
                          REFERENCES auth_user (id)
);
```

```
CREATE TABLE friendship_friend (
    id           INTEGER  NOT NULL
                          PRIMARY KEY AUTOINCREMENT,
    created      DATETIME NOT NULL,
    from_user_id INTEGER  NOT NULL
                          REFERENCES auth_user (id),
    to_user_id   INTEGER  NOT NULL
                          REFERENCES auth_user (id)
);
```

```
CREATE TABLE friendship_friendshiprequest (
    id           INTEGER  NOT NULL
                          PRIMARY KEY AUTOINCREMENT,
    message      TEXT     NOT NULL,
    created      DATETIME NOT NULL,
    rejected     DATETIME,
    viewed       DATETIME,
    from_user_id INTEGER  NOT NULL
                          REFERENCES auth_user (id),
    to_user_id   INTEGER  NOT NULL
                          REFERENCES auth_user (id)
);
```

```
CREATE TABLE search_hobbies (
    id          INTEGER      NOT NULL
                             PRIMARY KEY AUTOINCREMENT,
    hobby_name VARCHAR (50) NOT NULL
);
```

```
CREATE TABLE django_messages_message (
    id                  INTEGER       NOT NULL
                                      PRIMARY KEY AUTOINCREMENT,
    body                TEXT          NOT NULL,
    sent_at             DATETIME,
    read_at             DATETIME,
    replied_at          DATETIME,
    sender_deleted_at   DATETIME,
    recipient_deleted_at DATETIME,
    recipient_id        INTEGER       REFERENCES auth_user (id),
    sender_id           INTEGER       NOT NULL
                                      REFERENCES auth_user (id),
    subject             VARCHAR (140) NOT NULL,
    parent_msg_id       INTEGER       REFERENCES django_messages_message (id)
);
```

```
CREATE TABLE search_movie (
    id    INTEGER      NOT NULL
                       PRIMARY KEY AUTOINCREMENT,
    genre VARCHAR (30) NOT NULL
);
```

```
CREATE TABLE search_userfeature_musichated (
    id              INTEGER NOT NULL
                            PRIMARY KEY AUTOINCREMENT,
    userfeature_id INTEGER NOT NULL
                            REFERENCES search_userfeature (id),
    music_id        INTEGER NOT NULL
                            REFERENCES search_music (id)
);
```

```
CREATE TABLE search_music (
    id    INTEGER       NOT NULL
                        PRIMARY KEY AUTOINCREMENT,
    genre VARCHAR (30) NOT NULL
);
```

```
CREATE TABLE account_profile (
    id          INTEGER       NOT NULL
                              PRIMARY KEY AUTOINCREMENT,
    nickname    VARCHAR (50),
    intro       TEXT,
    photo       VARCHAR (100),
    age         INTEGER       NOT NULL,
    height      INTEGER       NOT NULL,
    weight      INTEGER       NOT NULL,
    gender      VARCHAR (1)   NOT NULL,
    education   INTEGER       NOT NULL,
    location    VARCHAR (2)   NOT NULL,
    user_id     INTEGER       NOT NULL
                              UNIQUE
                              REFERENCES auth_user (id),
    only_child BOOL,
    ""
);
```

```
CREATE TABLE auth_user (
    id           INTEGER       NOT NULL
                               PRIMARY KEY AUTOINCREMENT,
    password     VARCHAR (128) NOT NULL,
    last_login   DATETIME,
    is_superuser BOOL          NOT NULL,
    first_name   VARCHAR (30)  NOT NULL,
    last_name    VARCHAR (30)  NOT NULL,
    email        VARCHAR (254) NOT NULL,
    is_staff     BOOL          NOT NULL,
    is_active    BOOL          NOT NULL,
    date_joined  DATETIME,
    username     VARCHAR (150) NOT NULL
                               UNIQUE
);
```