FINAL REPORT OF MEETFUN

CSCI 3100 Project

Department of Computer Science and Engineering

The Chinese University of Hong Kong

Group 15

CONG Kai 1155061932

LI Ruizhe 1155076990

MAO Yuxuan 1155062018

QIN Hailin 1155061967

YE Boxiu 1155046907

V1.0 Printing Date: 05/04/2017

Table of Contents

Introduction 4

1. Project Overview 4
2. Objectives 4
3. Highlights 4
4. Project Statistics 5

System Architectural Design 6

1. Technology Architecture 6
2. System Architecture 6
3. DFD Diagram 7

Detailed Description of Components 9

1. Overall Database Design 9
2. Authentication System 12
3. Search Function 15
4. Account Management 17
5. Funs Module 21
6. Comment and Rating Module (Messaging) 25
7. APIs (Sharing and Google Map) 30

User Interface Design 34

1. Landing Page 34
2. Log In and Sign Up 35
3. Account Management 36
4. Browse and Search 38

5. Fun System 39

Tests 46

1. Test Overview 46
2. Case - Register 46
3. Case - Login 47
4. Case - Edit Personal Information 48
5. Case - Search 49
6. Case - Create Fun 50
7. Case - Edit Fun 52
8. Case - Delete Fun 54
9. Case - Join Fun 55
10. Case - Quit Fun 56
11. Case - Comment 57
12. Case - Rating 58

Lessons Learned 61

1. Software Engineering Skills 61
2. Programming Skills 61

Conclusion 62

Introduction

1. Project Overview

This Project aims to build an online activity organizing and sharing system, MeetFun. It provides fancy and easy ways to ***Design and Post activities*** to all potential participants. Anyone can ***Browse and Explore the website*** to look for creative or fascinating activities including but not limited to sightseeing, hiking, musical or sports events, even group studying. If a user were interested or inspired by certain activity, she could ***Register*** and enjoy the convenient way of participating or organizing offline events. Being a registered user, customers could ***create***, ***join*** and ***comment*** on the activities. Finally, the ***Sharing*** API integrated into MeetFun allows users to seamlessly inform her friends via WeChat, Facebook or other social media.

2. Objectives

The objective of this project is to connect the need between activity partners finding and exploration of new interest spots. It breaks users' existing social circle by the nature of the internet while preserving the trust among activity participants by the regulation under platform polices. MeetFun also contributes to helping users exploring their potential needs by providing intelligent recommendation for trending or featuring activities.

3. Highlights

MeetFun consists of three powerful functioning components: user authentication, activity planning, and social networking. Authentication is the basic feature to create a separate profile for each user and store their personal information as well as their activity history. Activity planning enables users to launch or register for an activity. Social networking functions include exploring, searching, and comment on activities.

Another highlight of MeetFun is the simple but elegant UI. Under the help of modern web application technology and carefully designed user interaction logic, MeetFun is friendly to new users and efficient to existing users.

4. Project Statistics

Team Members (alphabetical order):	
CONG Kai	*Chief System Architecture Designer, Back-end Programmer* 50 hours' contribution
LI Ruizhe	*Chief Tester and Optimizer, Back-end Programmer* 50 hours' contribution
MAO Yuxuan	*Chief Server-side Designer, Back-end Programmer* 50 hours' contribution
QIN Hailin	*Project Manager, Front-end Programmer* 50 hours' contribution
YE Boxiu	*Chief UI Designer, Front-end Programmer* 50 hours' contribution
Total Human Hours:	250 hours
Lines of Codes:	
Front-end	2300 Lines, Comments excluded
Back-end	1000 Lines, Comments excluded
Total Lines of Code:	3300 Lines
Lines of Codes:	
Front-end	2300 Lines, Comments excluded
Back-end	1000 Lines, Comments excluded
Total Lines of Code:	3300 Lines
Open-source Packages and Frameworks Used:	
Routing	Expressjs
Authentication	Passport
Database	Mongoose
UI script	JQuery
UI layout	Bootstrap, JQuery UI

System Architectural Design

1. Technology Architecture

The following diagram is the key technologies used in the web application

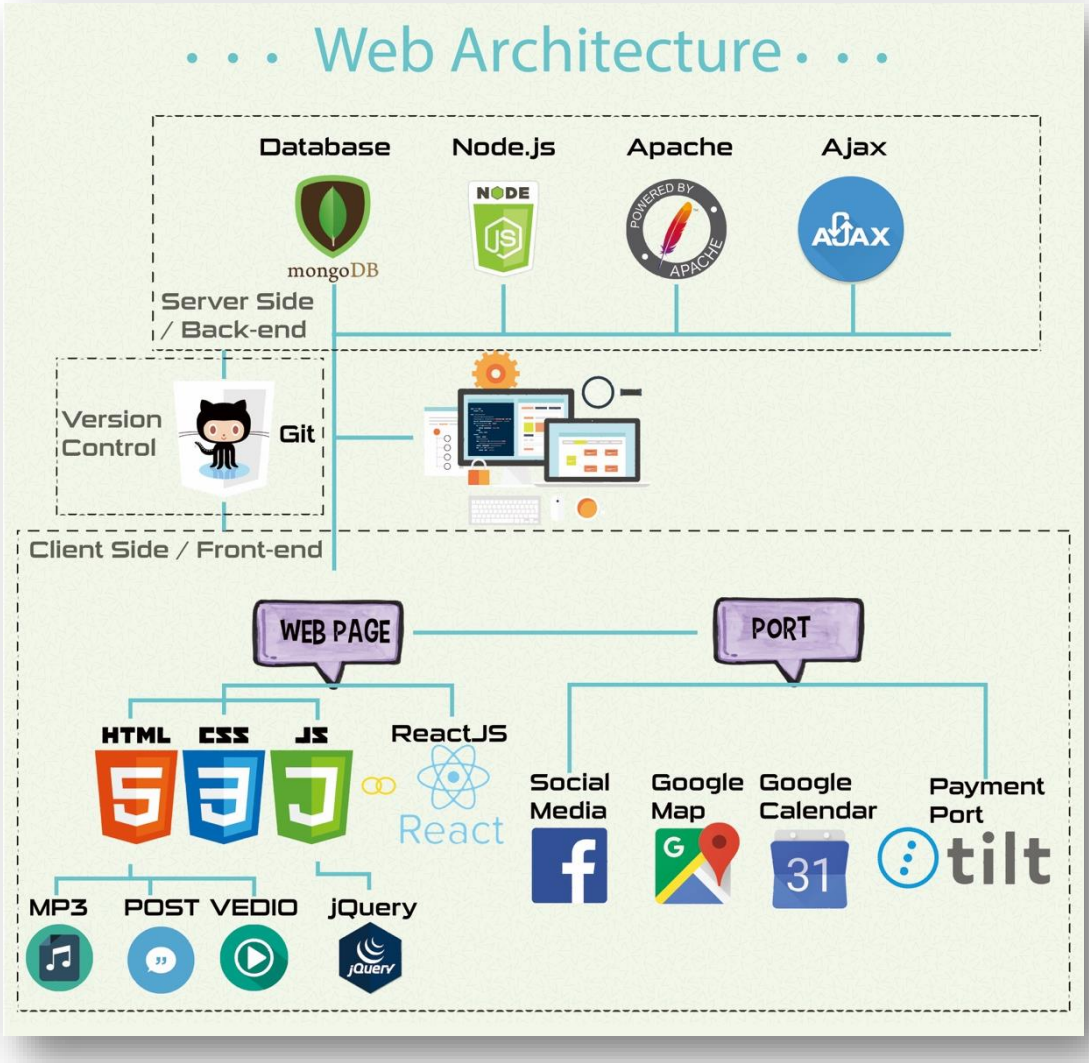


Figure 1 Technologies Used

2. System Architecture

The following GDN figure shows the whole system architecture overview. Only the whole structure of our project is shown. The details of modules are omitted. The dependency

among each part is denoted by arrows. The highlighted modules are the key components of our project.

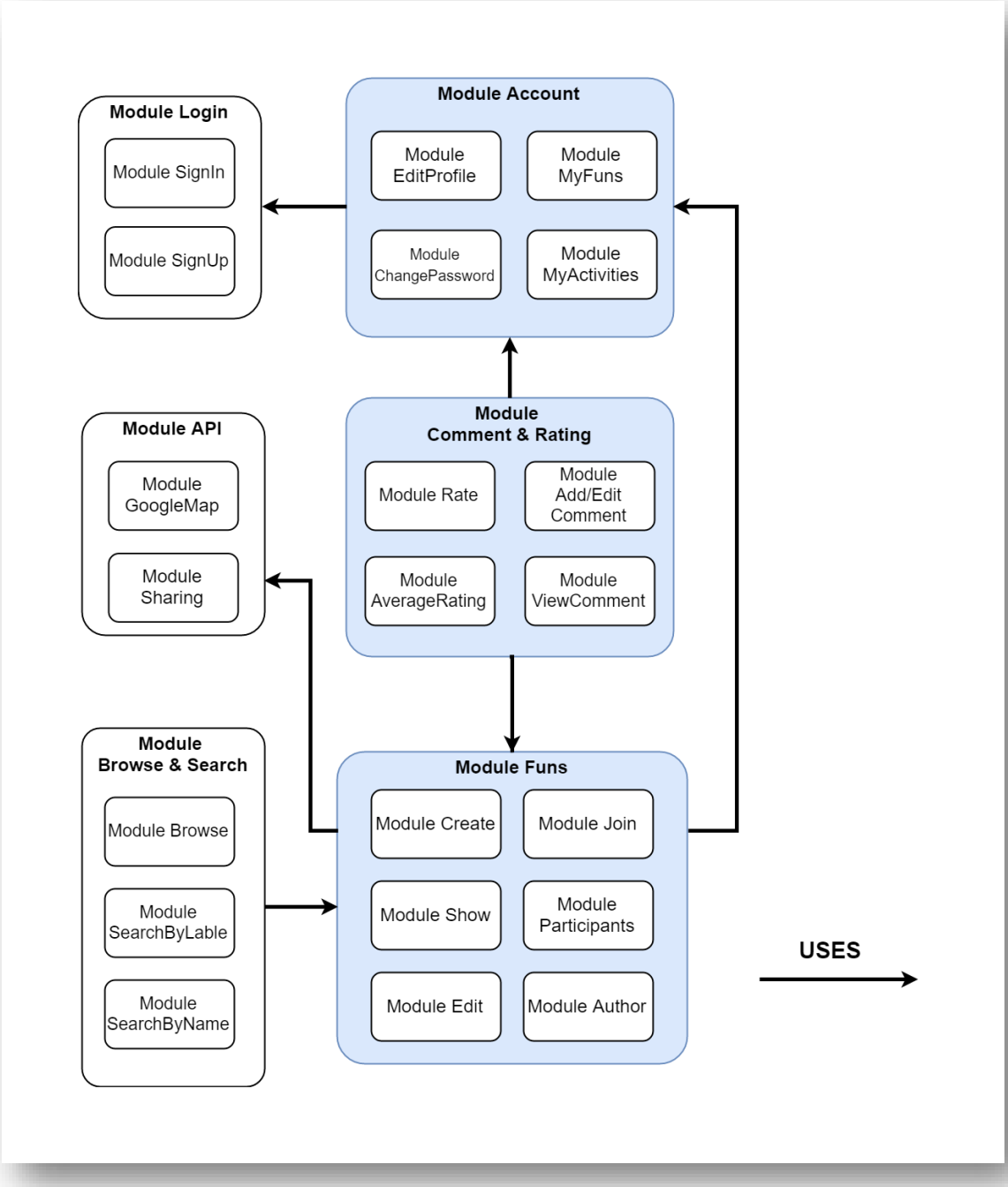


Figure 2 System GDN

3. DFD Diagram

The DFD Diagram of the Product is shown as the following:

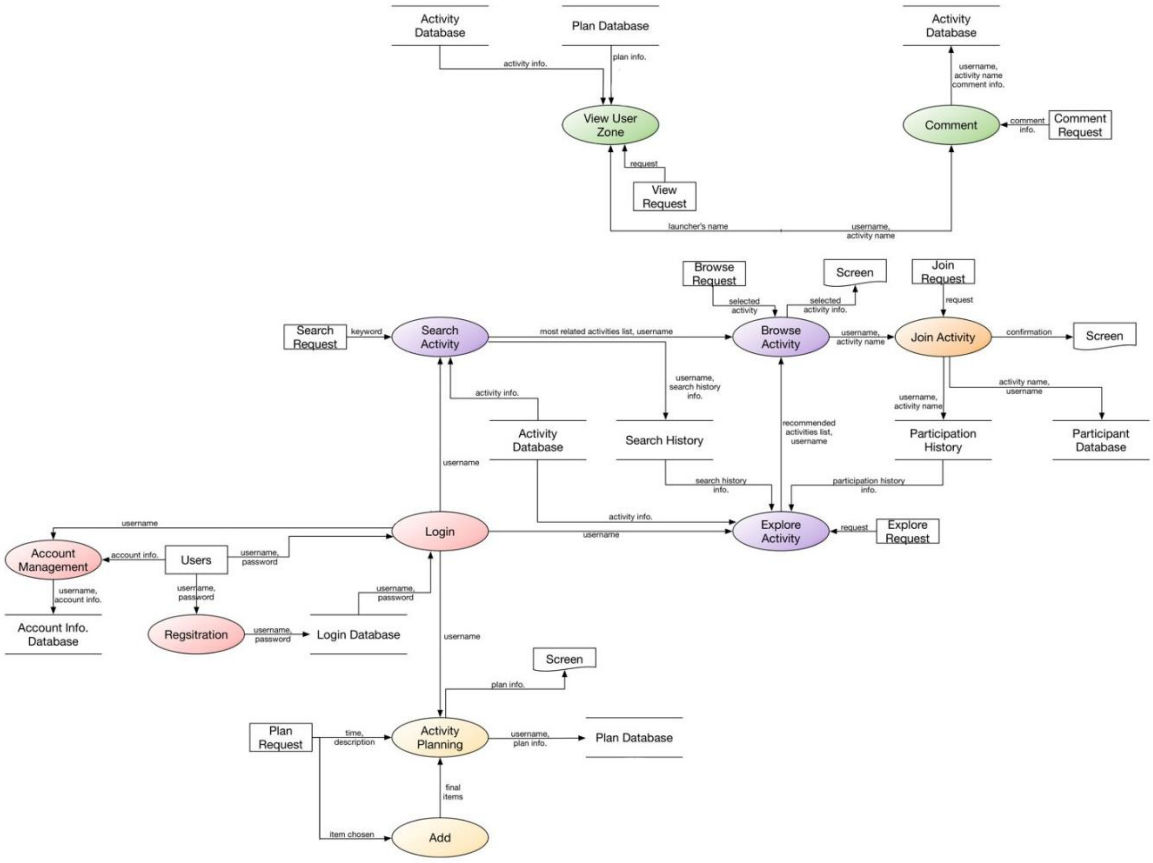


Figure 3 DFD Diagram

Detailed Description of Components

1. Overall Database Design

MongoDB is a high-performance and open-source document database. It is the most popular among the NoSQL database products. A record in MongoDB is known as a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents. Each record is denoted by a unique ObjectId generated by MongoDB.

For example, this is the data structure of one record about fun information in our platform.



Figure 4 data structure

1.1 Overview of the Database Structure

There are four major groups of data in the database:

- Users – User information
- Funs – Activity information
- Days – Daily plan for a particular fun

- Comments – Comment information.

In MongoDB, each group of data in the same format is represented as a collection.

```
switched to db meet_fun_db_v3
> show collections
comments
days
funs
system.indexes
users
```

Figure 5 collections

1.2 Definitions of the Collections in Database

In MongoDB, the definition of each collection of data is organized in the format of Schema.

In express framework, we use mongoose as our connected local database.

Users – User information

For each user, we store the username and password as a record for authentication. Also, additional personal information like gender and age is stored as well. What's more. The activities that the user create and participate in are also recorded in the history.

```
var UserSchema = new mongoose.Schema({
    username: String,
    password: String,
    gender:String,
    age: Number,
    myfun: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Fun"
        }
    ],
    activities: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Fun"
        }
    ]
});
```

Figure 6 User Information

Funs – Activity information

For each activity created as a Fun, its name, cover image, description and author will be written into the funSchema. Detailed information like days, comments, average rating, quota and participants and tag are also inserted. Noted that, the day and comment are also a record in our database.

```
var funSchema = new mongoose.Schema({
    name: String,
    image: String,
    description: String,
    author: {
        id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        },
        username: String
    },
    days: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Day"
        }
    ],
    comments: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: "Comment"
        }
    ],
    averageRating: Number,
    quota: Number,
    participants: [
          {
            id: {
               type: mongoose.Schema.Types.ObjectId,
               ref: "User"
            },
            username: String,
              rating: Number
          }
     ],
     tag: String
});
```

Figure 7 fun

Days – Daily plan for a particular fun

For each daily plan, we record the details like date, location and daily activity description.

```
var mongoose = require("mongoose");

var daySchema = mongoose.Schema({
    date: String,
    location: String,
    dailyactivity: String  // a description for daily activities
});
```

Figure 8 days

Comments – Comment information.

For each comment, we just record the content and its publisher.

```
var mongoose = require("mongoose");

var commentSchema = mongoose.Schema({
    text: String,
    author: {
        id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        },
        username: String
    }
});
```

Figure 9 comments

2. Authentication System

2.1 UML Class Diagram of Registration System (structure)

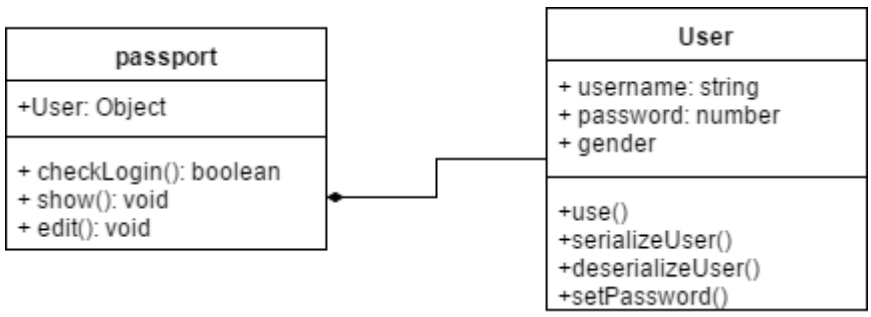


Figure 10 UML class diagram of registration

2.2 UML Sequence Diagram of Registration System

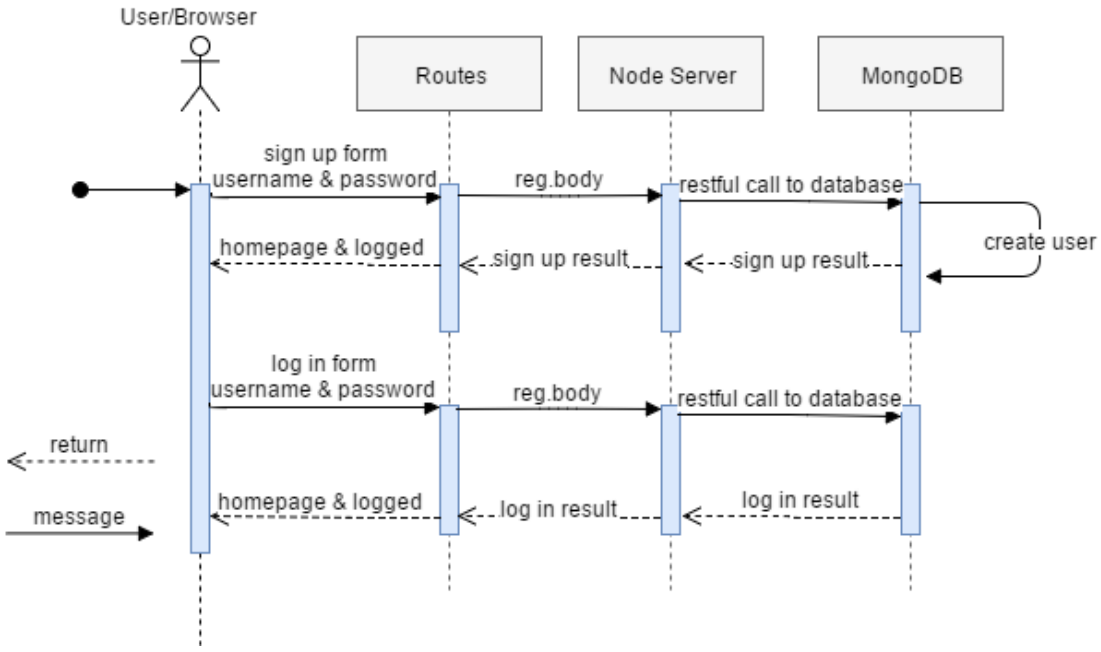


Figure 11 UML sequence diagram of registration

2.3 Functionality

All the users are required to register an account with a password in order to participate in the "fun" activities. Also, all the information of activities related to the user are stored in the user profile.

During registration, users are required to type in their username, password and personal information. The system will then check whether the username has been occupied by previous users. Once registered, the user will be automatically redirected to the home page with a logged in account and start to enjoy the fun planning. All the user information will be stored in the MongoDB database.

After registration, users can use the username and password to log into the platform. In our project, we adopt the popular Node.js middleware for authentication – Passport.js.

Passport is a Node.js middleware designed for authentication, popularly used in MEAN stack development. In order to deal with various ways of the authentication, including the

simplest account-password authentication to merging the API provided by different famous websites. Passport uses a solution named Strategies to provide an independent strategy for each authentication, and include the specific one according to the particular demand. In order to ensure the simplicity of the program, this design makes Passport popular in the Node.js community. Basically, all the mainstream authentication methods have a corresponding existing strategy

2.4 Procedures and Functions

TDN

module Authenticate

exports

var username, password: string;

implementation

is composed of Login, Register

end Authenticate

module Login

exports

var username: string;

implementation

is composed of GetUserBody, validationCheck, DislayLoginResult

//After getting the username and password of the user form the POST route, the Login module will verify them through passport.authenticate(). Then the user will be directed to the homepage and start exploration if successful, or the user will stay at the login page to input again.

end Login

module Register

exports

var username, password: string;

implementation

is composed of GetUserBody, registerCheck

//After getting the username, password from the form that the user fills in, the module Register will check whether the username has already been used. If the successful, the account will be stored in MongoDB database and the user will be directed to the home page.

end Register

3. Search Function

3.1 UML Class Diagram of Search Function (structure)

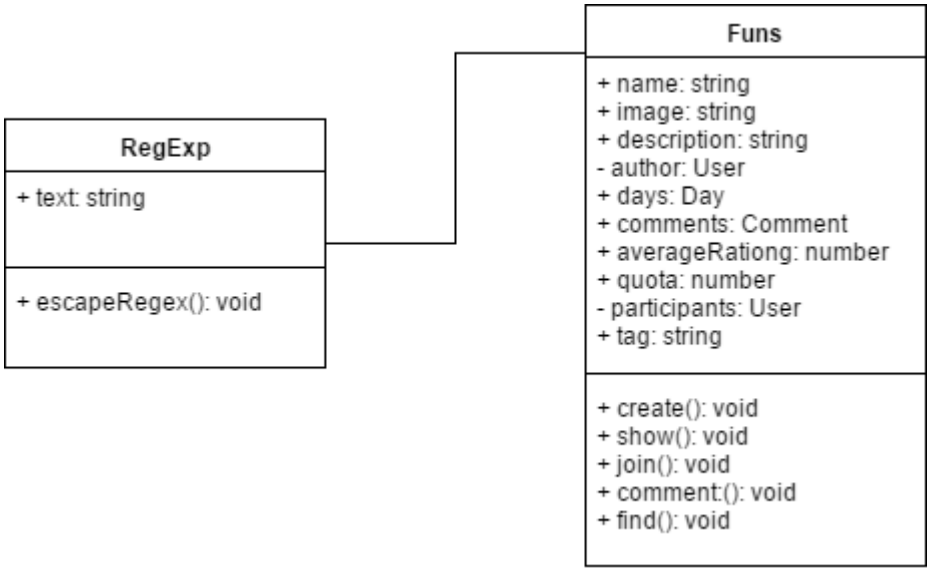


Figure 12 UML class diagram of search

3.2 UML Sequence Diagram of Registration System

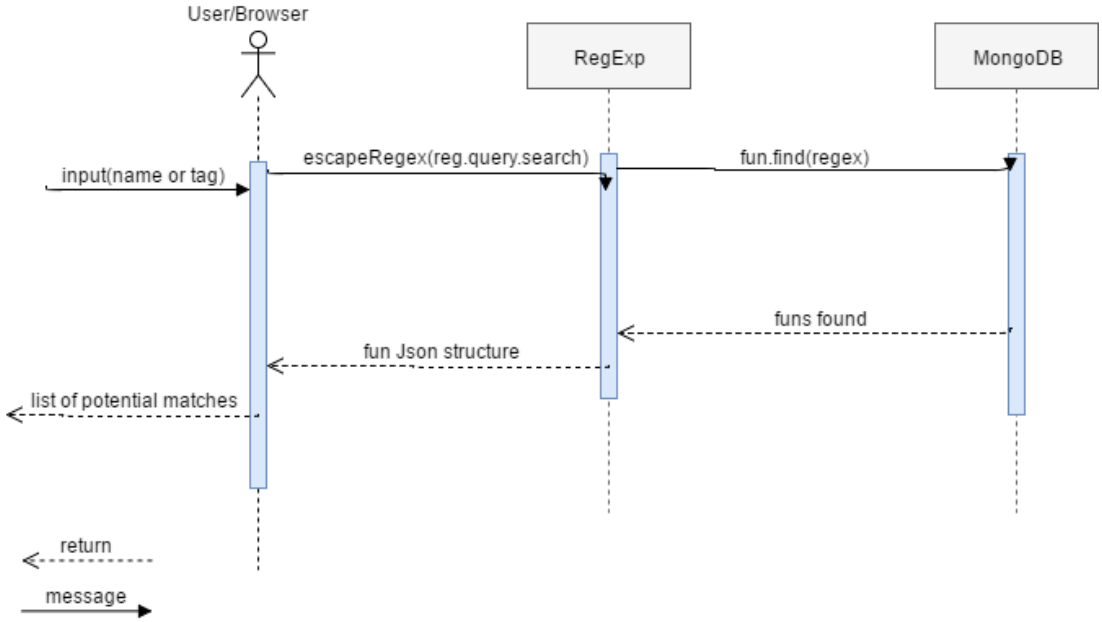


Figure 13 UML sequence diagram of search

3.3 Functionality

Users can use the search function to search for their interested activities. We support two ways of searching, one is to search by the specific activity name. The other is to search by the particular kind of activity, namely "tag". We apply the regular expression in JavaScript to implement the fuzzy search associated with express.js and MongoDB, which will return related results matched to the search input as many as possible.

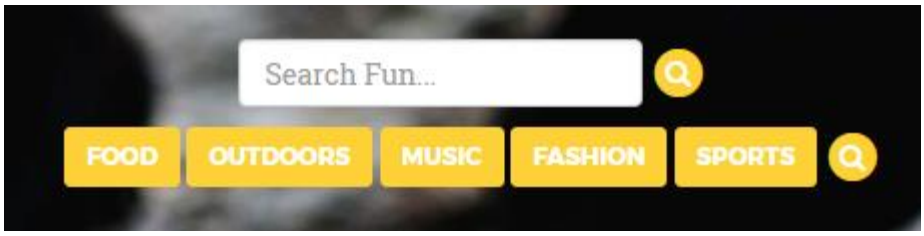


Figure 14 search interface

3.4 Procedures and Functions

TDN

module Search

exports

type funs: list of fun;

var allfuns: funs;

implementation

//We use the regular expression of the search query as the token to fine the potential results in database. By doing so, we can achieve the fuzzy search.

is composed of searchQuery, regExp, funFind

end Search

4. Account Management

4.1 Structural Diagram

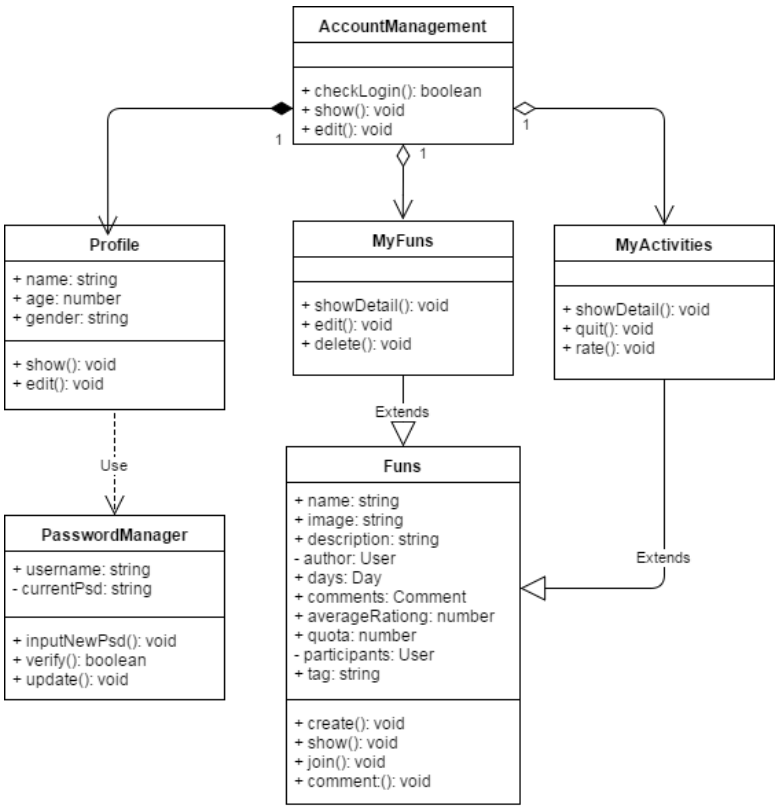


Figure 15 class diagram of account management

4.2 UMLs

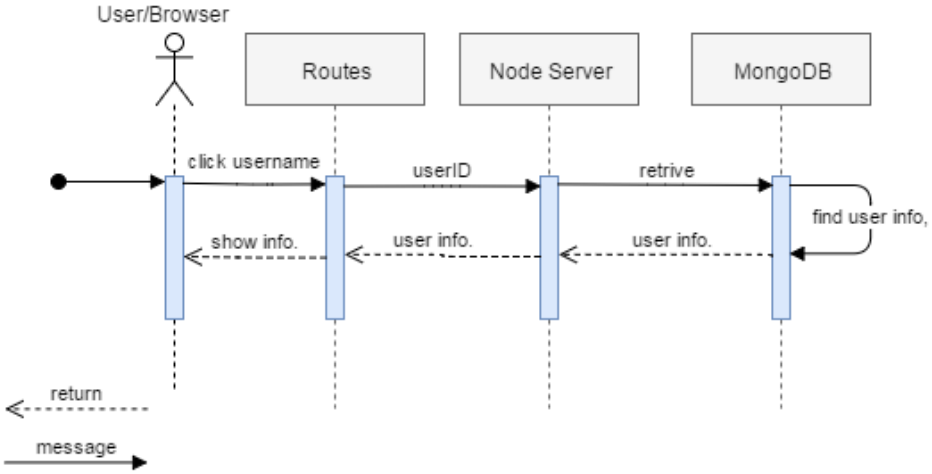


Figure 16 Sequence Diagram for Show Account

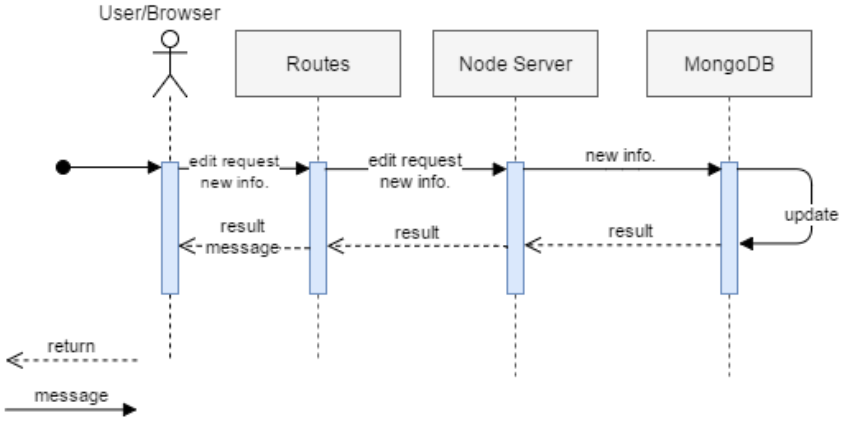


Figure 17 Sequence Diagram for Editing Account

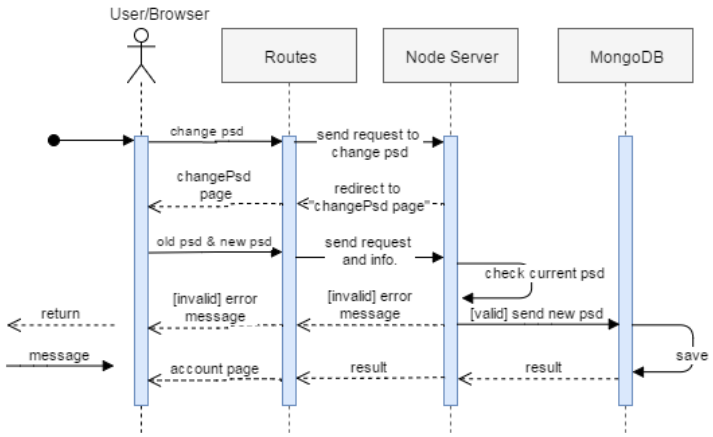


Figure 18 Sequence Diagram for Changing password

4.3 Functionality

All the registered users have an account. The users' account and information will be stored in the database. After signing in, users can get access to their account information by single clicking their username, which is located on the top right corner of the screen.

After clicking on the username, users will get in to their account page. The user's name, age, and gender will be shown in the account page. Users can edit their user information by click the "edit" button under their basic information. Under the user profile, there are two categories named "MyFuns" and "MyActivities". The funs which are created by the user will be listed in "MyFuns". And "MyActivities" category contains the funs which users have joined. User can see the detail information by clicking "MORE INFO" button located under the fun's image. Furthermore, users are able to delete and edit the funs shown in "MyFuns", or quit the funs shown in "MyAcitivites". Once the user deletes or quits an activity, that activity will be removed from the user's account, and it will disappear from the user's account page.

On clicking the edit button, users will be redirected to the edit page by the node server in the rear-end. Users can edit their username, age, and gender. Moreover, users are able to change their password. The password changing function is implemented by another user interface, which asks the user to input both his/her current password and new password in the required format. Then the server will check the current password. If it is correct, the current password will be replaced with the new one. Next time, the user has to sign in with the updated password.

4.4 Procedures and Functions

TDN

module Account

uses PasswordManager, Fun, User

exports

var username, password, gender: string;

var age: number;

var funs, activities: Fun;

procedure edit: (username, password, gender: **in** string);

procedure changePsd(currentPsd, newPsd: **in** string);

implementation

is composed of Profile, MyActivities, MyFuns

This module is called when a user wants to check his/her account and edit information. On receiving the event, the browser will send a request to the rear-end server. Then the router in the server will receive the request together its content. After that, the router will call each components according to the type of the request.

end Account

module MyActivities

uses Fun

exports

var activities: Fun;

implementation

This module is called when a user wants to check his/her joined activities.

end MyActivities

module MyFun

uses Fun

exports

var activities: Fun;

implementation

This module is called when a user wants to check his/her created activities.

end MyFun

module Profile

uses User, PasswordManager

exports

var username, password, gender: string;

var age: number;

procedure edit: (username, password, gender: **in** string);

procedure changePsd(currentPsd, newPsd: **in** string);

implementation

This module is called when a user wants to edit his/her profile or password. The system will first receive a request. The new information will be embedded in the request body. Then, the new information will be sent to the database to replace the old one. For the password changing, a authenticated process will be done. The password will be change only if the current password is correctly input.

end Profile

5. Funs Module

5.1 Structural Diagram

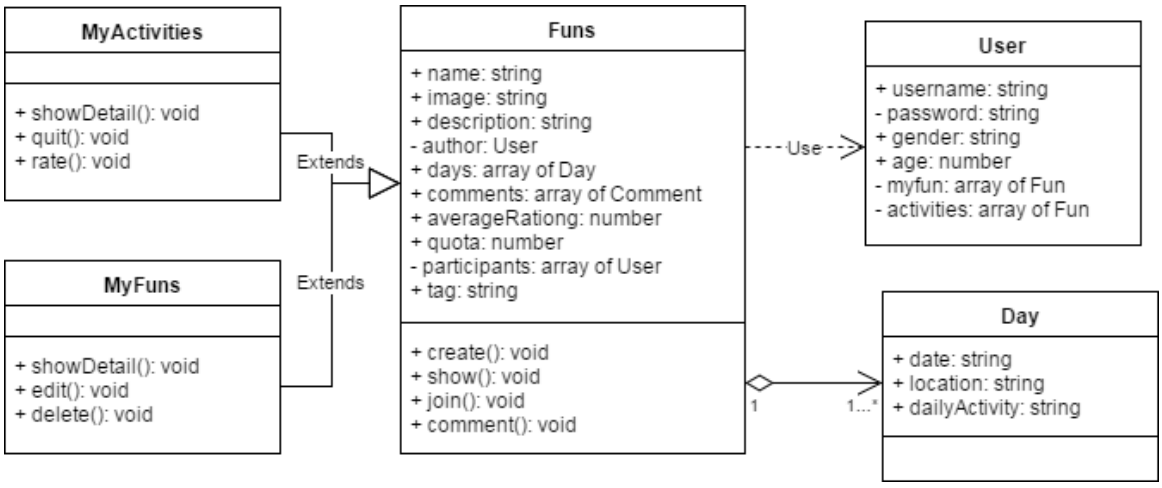


Figure 19 Structure of fun

5.2 UMLs

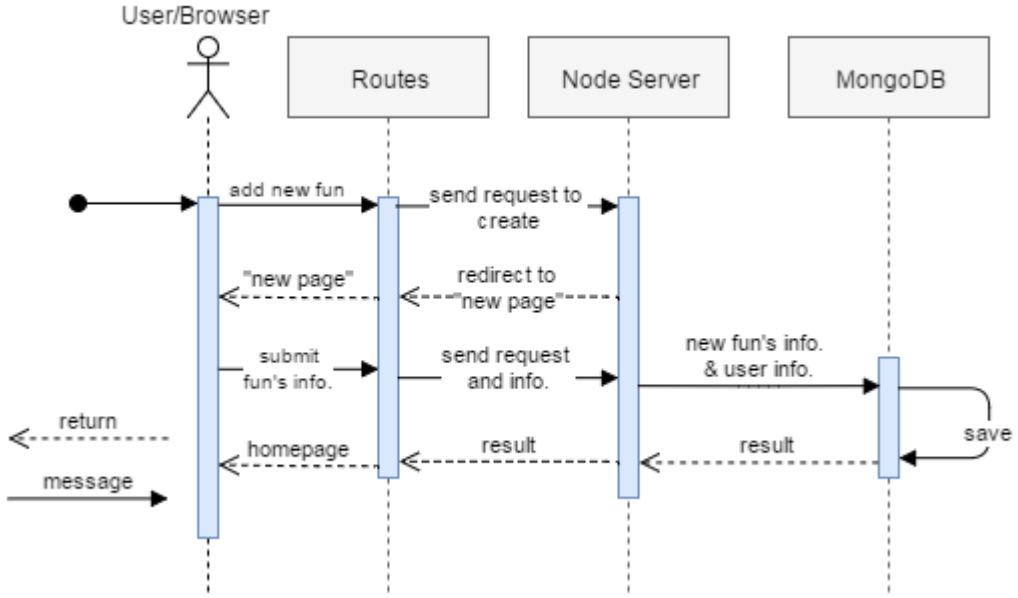


Figure 20 Sequence Diagram of Create Fun

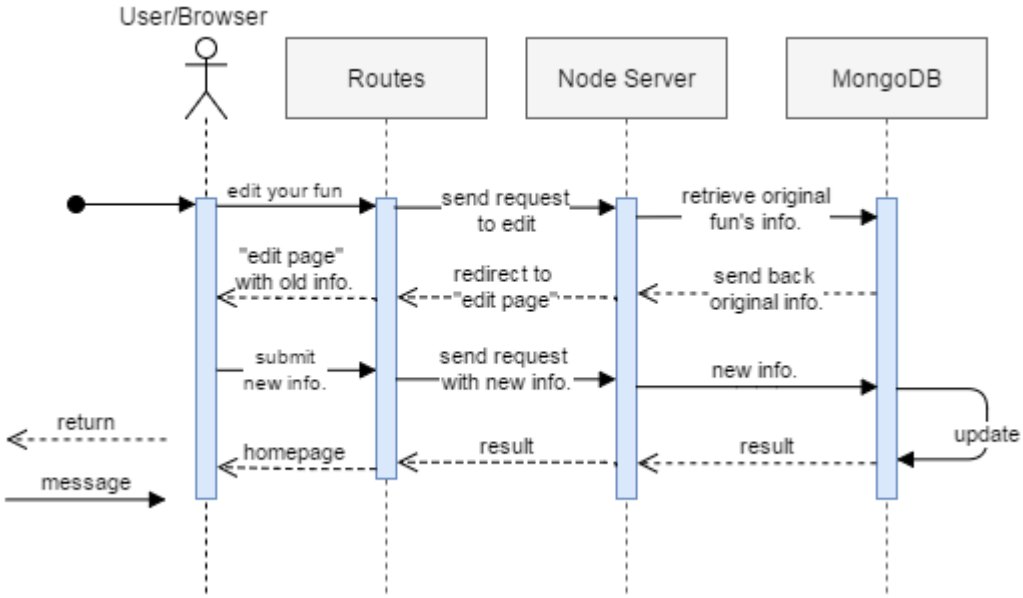


Figure 21 Sequence Diagram Edit Fun

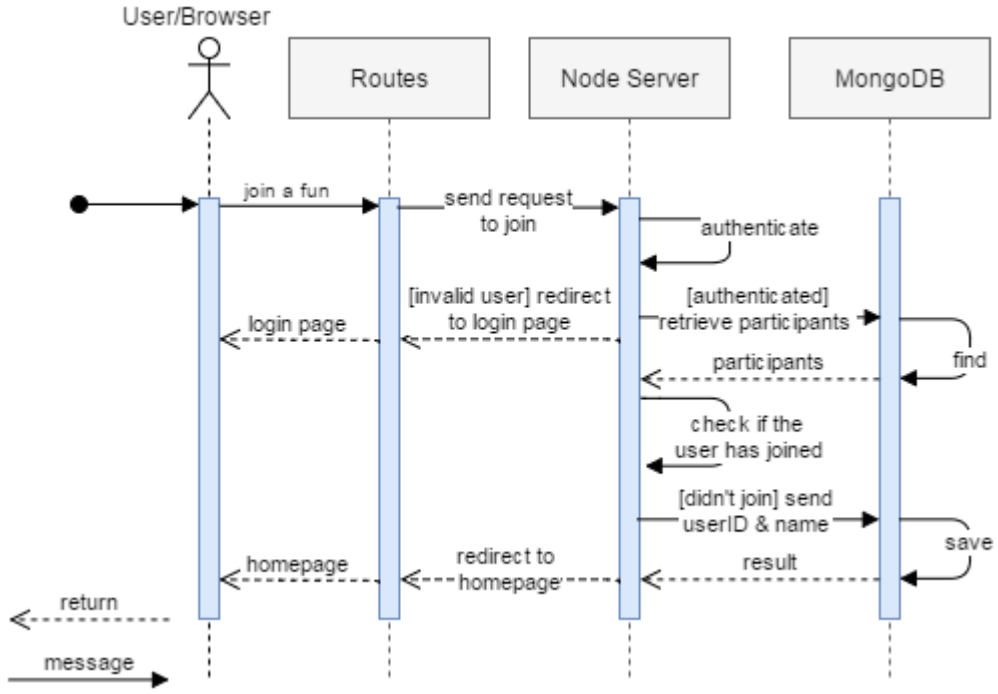


Figure 22 Sequence Diagram of Join Fun

5.3 Functionality

The funs module is the main part of our product. Through this module, users can create activities. Later, the activities can be joined by other users. The activity creator will be recorded as the author of the particular activity. And the participants will also be recorded by this module.

When a user chooses to add a new fun, he/she is required to fill an information collecting form to describe the fun to be created. The information to be entered includes the funs' name, description, quota, and a label. The quota is used to define the maximum allowed number of participants. The system will reject any application to join this activity, if the quota is full. And the label is used to classify activities into different categories. The browse and search module will make use of the label as well. Besides that, users can electively upload a related picture to attract participants. Each activity will include one or more days. Thus, after the user inputs all the basic information of the activity, he/she is required to input the days' information, which includes the date, location, and specified daily

activities. After the user submits the form, a Fun object will be generated and saved in the database.

After an activity is created, its author can get access to it through "MyFuns" category in his/her account page. And only its author has the authority to edit it. The edition process is similar to the creation process. The system first redirects the user to the editing page. And the user is required to input the new information. After submission, the old information will be replaced by the new one. The author also has the authority to delete the fun, which will cause the fun being removed from the database.

As long as the quota is not full, all of the authenticated users except for the author are allowed to join the activity. On clicking the "JOIN THIS FUN" button in the activity's information page, the system will first check if the user is authenticated. If not, a prompt window will pop up asking the users to login or sign up. After the user signs in, he/she can join the activity and become one of the participants of that activity. The participants have the right to quit a fun.

5.4 Procedures and Functions

module Fun

uses User, Day, Comment

exports

    var name, description, image, tag: string;

    var author: User;

    var days: array of Day;

    var comments: Comment;

    var quota, averageRationg: number;

    var participants: array of User;

procedure show: (fun: **out** Fun);

procedure create: (newFun: **in** fun);

procedure edit: (newName, newDescription, newTag: **in** string, newQuota: **in** number, newDays: **in** array of Day);

procedure join: (participant: **in** User);

implementation

//This module will be fired as long as the user does anything related to the activities, such as look through a fun, create a fun, edit the created fun, join or quit the fun, and delete the fun. On these receiving actions, the browser will send a request to the rear-end node server. The router will act according to the requests' type and content. Then, if necessary, the server will send a revised message to the database. And other module like User and Day will support the process.

end Fun

6. Comment and Rating Module (Messaging)

6.1 Structural diagram

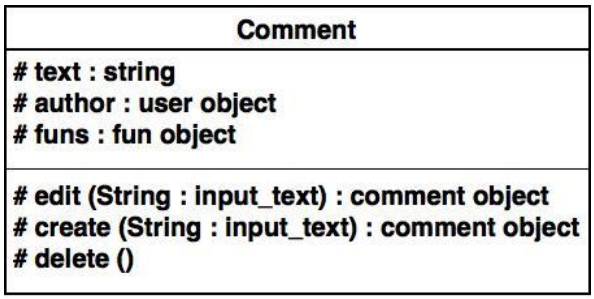


Figure 23 UML Class Diagram for Comment

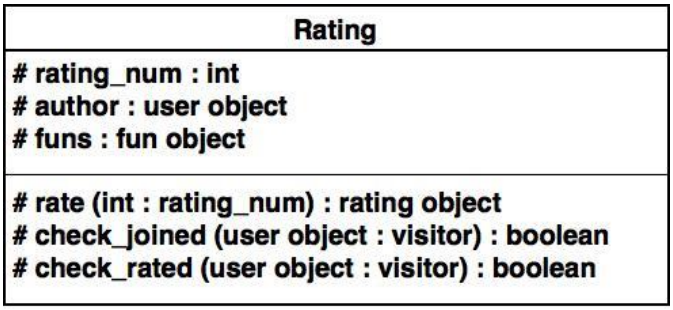


Figure 24 UML Class Diagram for Rating

6.2 UMLs

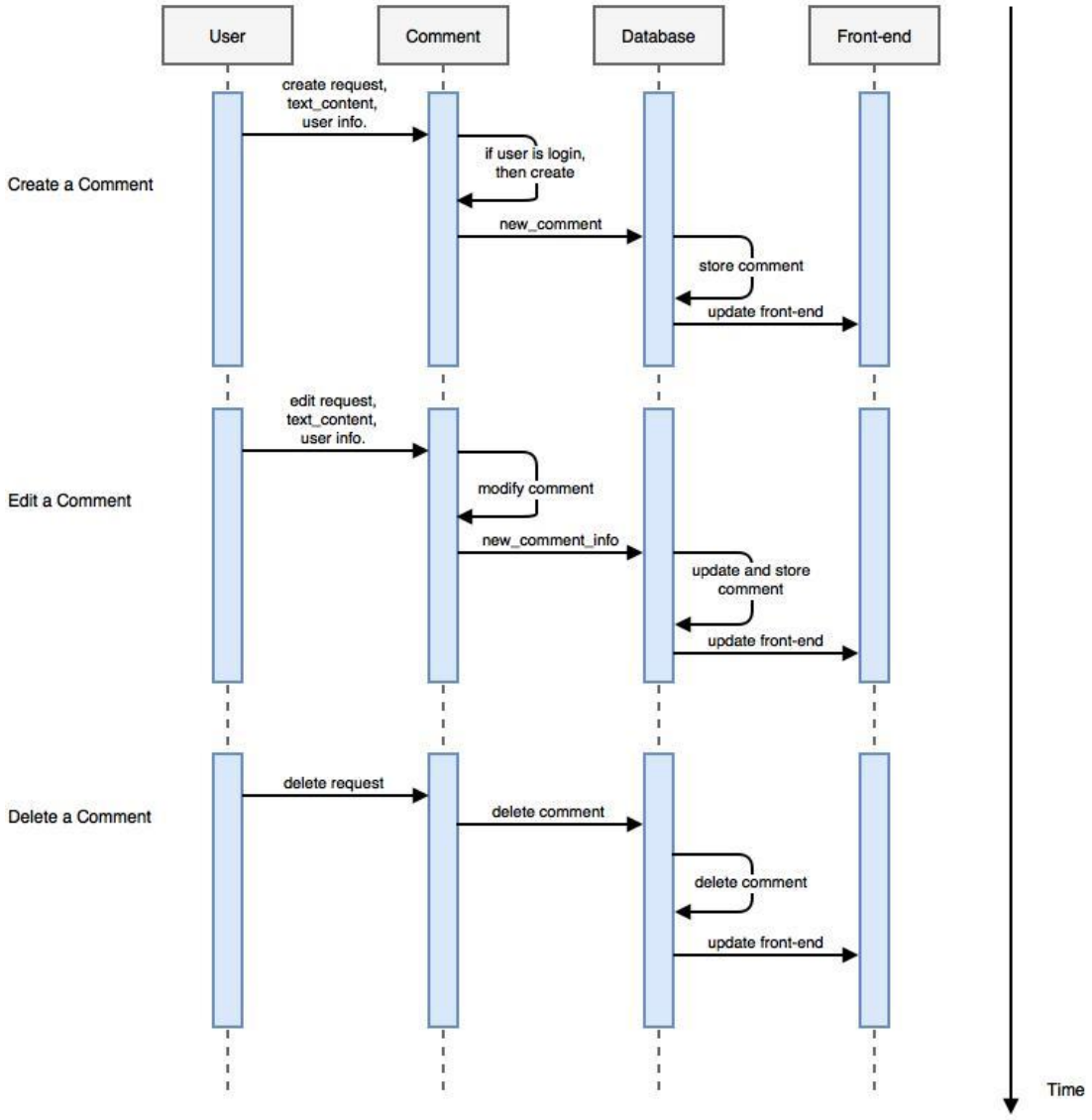


Figure 25 UML Sequence Diagram for Comment

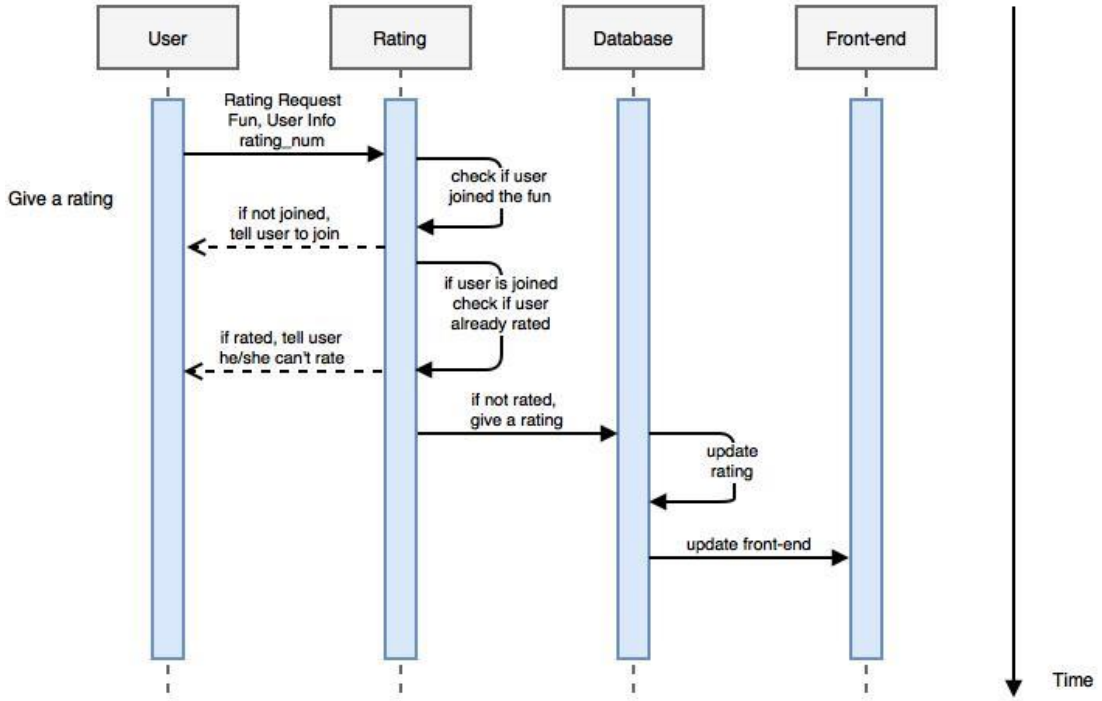


Figure 26 UML Sequence Diagram for Rating

6.3 Functionality

Under a fun post, users can do messaging sharing by giving comments and ratings, both modules are only available for log-in users.

The comment modules allow user to create, edit and delete a textual comment by using corresponding buttons. For security and privacy concern, for a particular user, only comments created by himself/ herself will show "edit" and "delete" buttons. Therefore, in our UML sequence diagram, we don't need to check the users' accessibilities on the comment, since only author of the comment can perform edit and delete of the comments.

The rating modules allows user to rate the fun activity by their preference. Note that the rating module is only available for users that have joined the particular fun and each of the joined user can only rate once. The fun object will record those ratings and give an average rating on the website.

6.4 Procedures and Functions

TDN

Module Comment

Uses Database_service, Authentication_service

// the service from database is to provide an interface for comment module to do comment object management

// the service from authentication is to provide an interface for comment module to navigate visitors to log in when necessary

Exports

Type Comment

procedure Create (String: in input_text)

procedure Edit (String: in input_text)

procedure Delete ()

implementations:

// 1. Create procedure is to write a comment: registered users of the website can leave comments for the funs they are visiting by entering the comments and press "Add New Comment" button. If a visitor (person who hasn't logged in) wants to leave a comment, the website will navigate him/her to the log-in module.

// 2. Edit procedure is to modify an existing comment: Users can edit the comments they wrote by clicking the "edit" button. The website will navigate the user to an editing page where user can type in the new comments and press "submit" button to update and replace the original comments. A user can only edit the comments wrote by himself.

// 3. Delete procedure is to delete an existing comment: Users can delete the comments they wrote by clicking the "delete" button. A comment will be removed from the webpage and the database once the user deletes it. A user can only delete the comments wrote by himself

End Comment

Module Rating

Uses Database_service, Authentication_service

// the service from database is to provide an interface for rating module to do rating object management

// the service from authentication is to provide an interface for rating module to navigate visitors to log in when necessary

Exports

Type Rating

procedure Rate (Int: in rating_num)

procedure Check_joined (User object: in rater)

procedure Check_rated (User object: in rater)

Implementations:

// 1.Rate procedure is to give a rating to a fun post: users can give a rating for the fun activity based on their preference. Each fun will record its own rating, and the rating object will record:

a). which fun it refers to;

b). who gives the rating.

This 2 fields will be essential in further operation to restrict the rating availability.

// 2. Check_joined procedure is to ensure only users who joined the fun can give a rating, otherwise a message will be given back to the user. Once the user rated the fun, website will calculate the average rating result of the fun and display in the front-end.

// 3.Check_rated procedure is for preventing multiple ratings: each user who joined the fun can only rate the fun for once. Multiple attempts of rating will be rejected. Multiple attempts of the rating will be detected by searching the fun object's rating list, and check whether the user attempting to rate is the author of one of the ratings.

End Rating

7. APIs (Sharing and Google Map)

7.1 Structural diagram

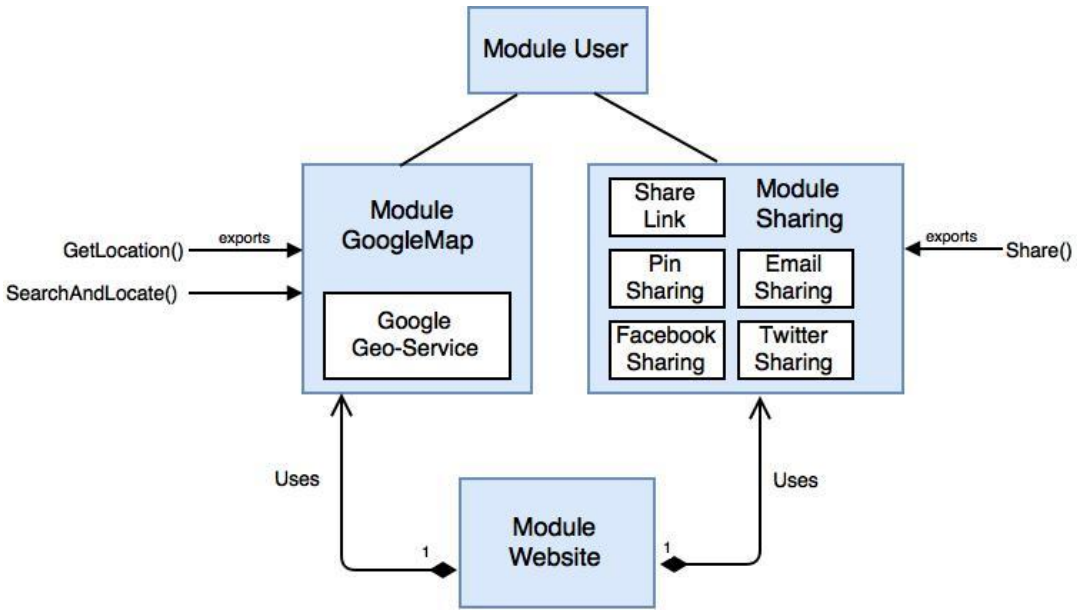


Figure 27 GDN for Google Map API Module and Sharing Module

7.2 3.5.2 UMLs

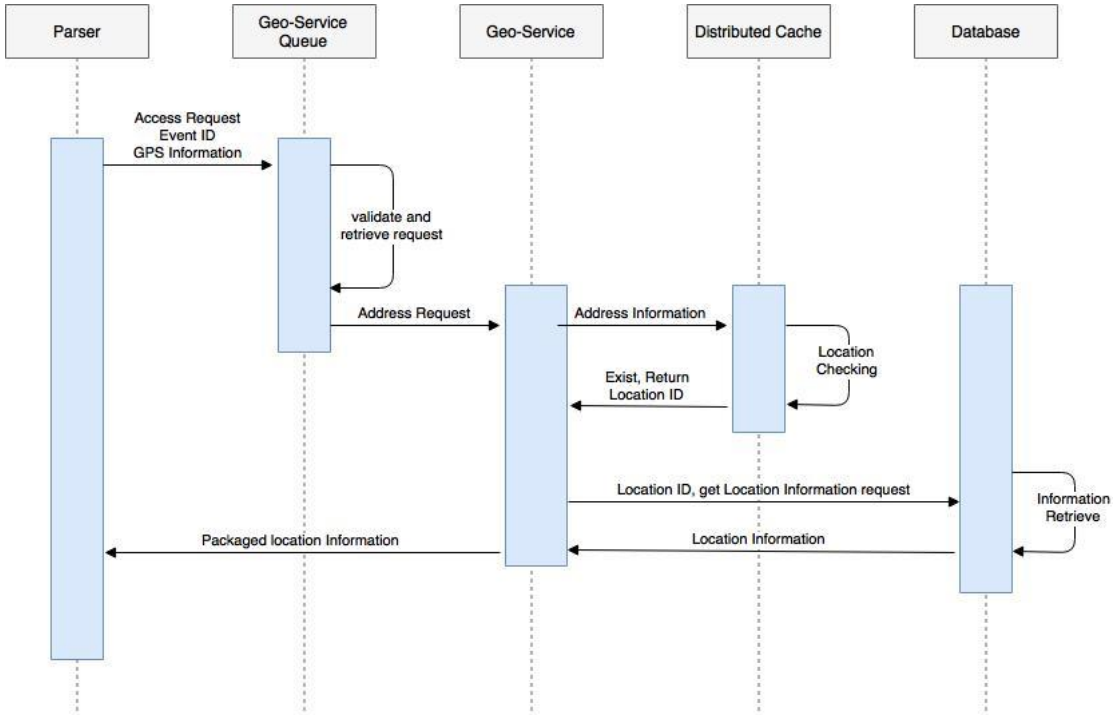


Figure 28 UML sequence diagram for Google Map API

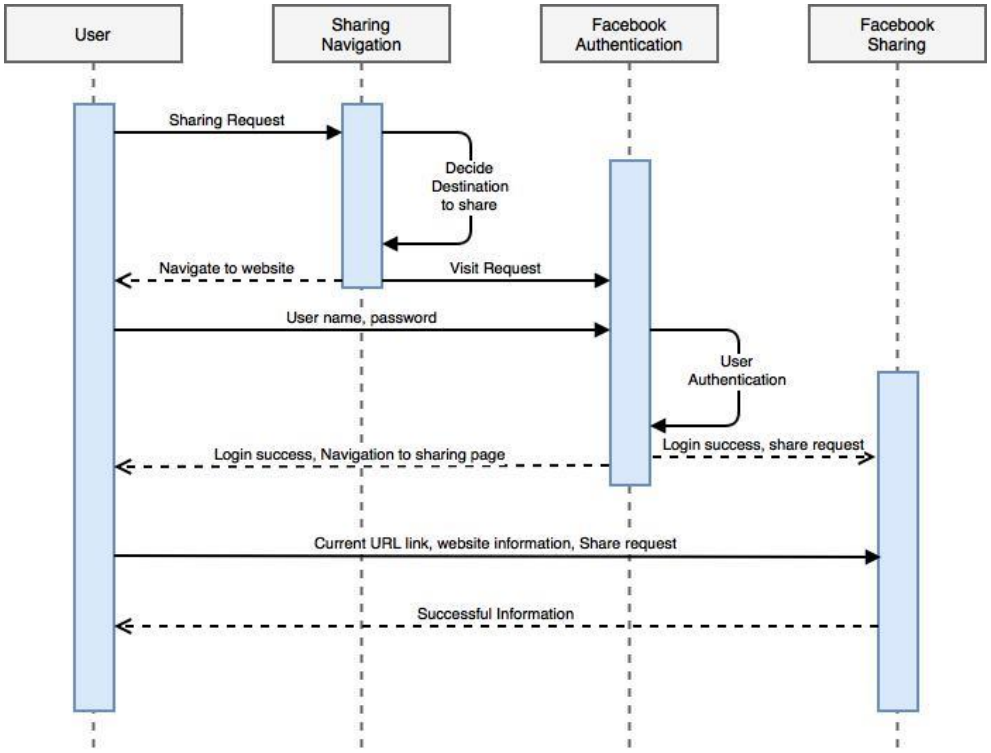


Figure 29 UML sequence diagram for "Sharing" ("share to Facebook as example")

7.3 Functionality

Google Map API: Provide position locating and destination finding service to the users. Users of the website can find their own location on the map and search for their desire destinations through map inserted on landing page and activity planning page.

Sharing Links: Provide service for users to share the website in different social platforms, including Facebook, Twitter etc.

7.4 Procedures and Functionality

TDN

Module GoogleMapAPI

Uses Geo-Service

// Geo-Service is a procedure that the API can call to handle user requests and return the result get from google map service.

Exports

procedure GetLocation (Location: in GPS_Position; MapLocation : out Map_Position)

procedure SearchAndLocate (Event: in user input request; MapLocation: out Map_Position);

Implementations:

//The module will automatically be used by the website whenever an user visit the website. By calling the GetLocation procedure, the website can make GoogleMapAPI to fetch user's GPS location and display corresponding location on the map.

//The SearchAndLocate function are implicitly used by the users. On the map interface, user can input search keyword and press search button for searching. The input string will be passed in and a list of possible results will be returned and display on the map.

//Both GetLocation function and SearchAndLocate function make use of Google's Geo-service to search desired location and obtained results.

End GoogleMapAPI

Module Sharing

Uses FacebookSharing, TwiiterSharing, PinSharing, EmailSharing, LinkSharing

Each sharing function is to get the link of the website and ask for sharing service from the particular websites.

Exports

procedure Share: (WebsiteLink : **in** String);

Implementations:

//Firstly, the user press the sharing button, the user is essentially calling the Share function. The Share function will call different website sharing function to navigate to that website for sharing. For example, if the user press the Facebook sharing button, the Facebook sharing link will be passed into the Share function.

//After receiving the string, Share function will make a decision to call the FacebookSharing function, which will navigate the user to the Facebook sharing page.

End Sharing

User Interface Design

1. Landing Page

Our landing page firstly contains an attractive interface containing a plan button and a search function (Figure 30 landing page – welcome). Scrolling down, there's a brief introduction of our functions and a list of fun activities (Figure 31 & Figure 32).

Users can log in or sign up at the top right corner of the page. A visitor can also search activities without login. But a visitor would be lead to login page when if they want to plan their own activity.

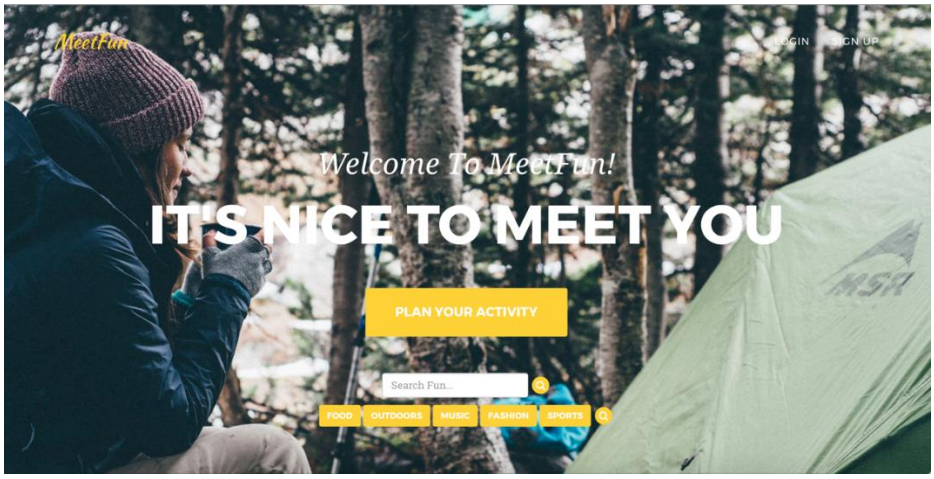


Figure 30 landing page – welcome

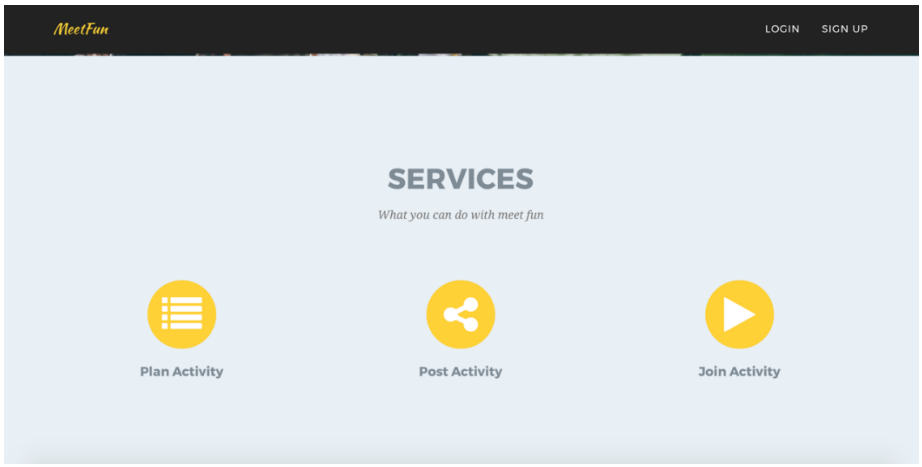


Figure 31 landing page - introduction

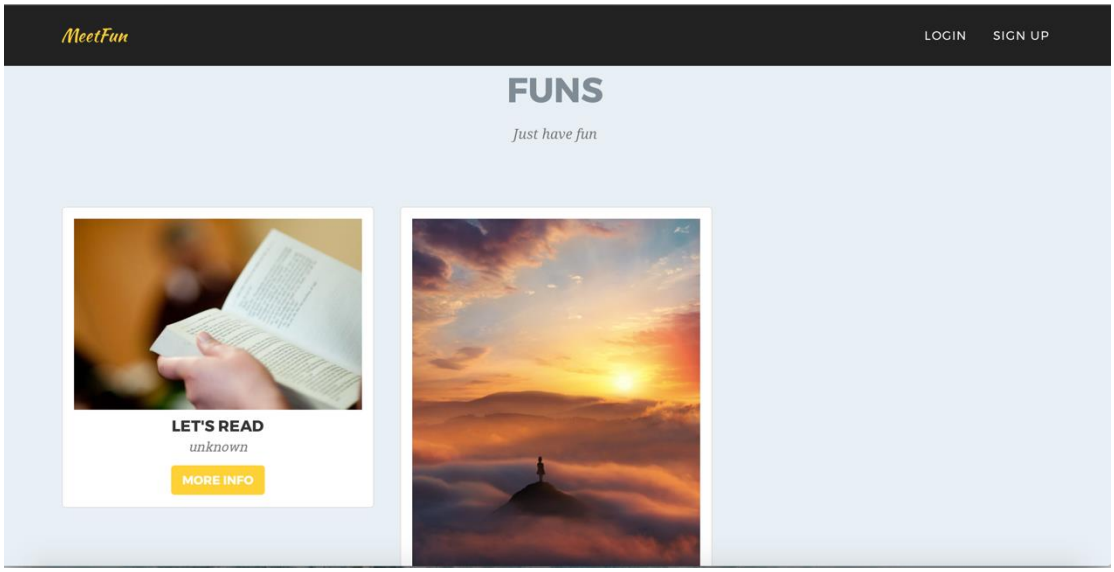


Figure 32 landing page – funs

2. Log In and Sign Up

Whenever the visitor wants to login, a login modal will jump up instead of leading to a new page (Figure 33). And if the visitor has not registered yet, he can click the register button at the bottom left corner.

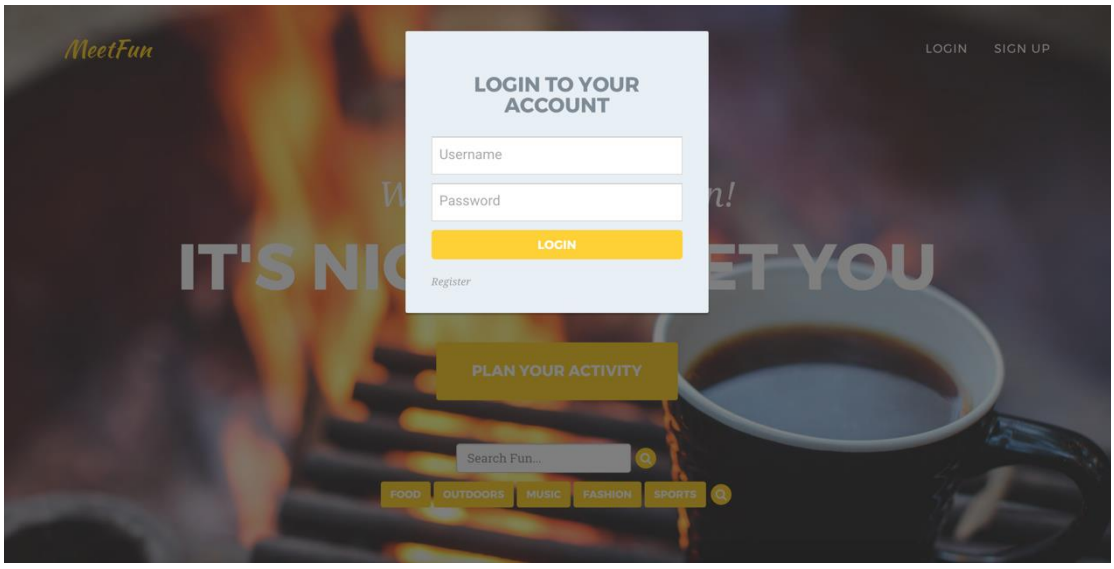


Figure 33 log in

A visitor can sign up with a username and password Figure 34). If the username already exists, the sign up would fail with an alert (Figure 35).

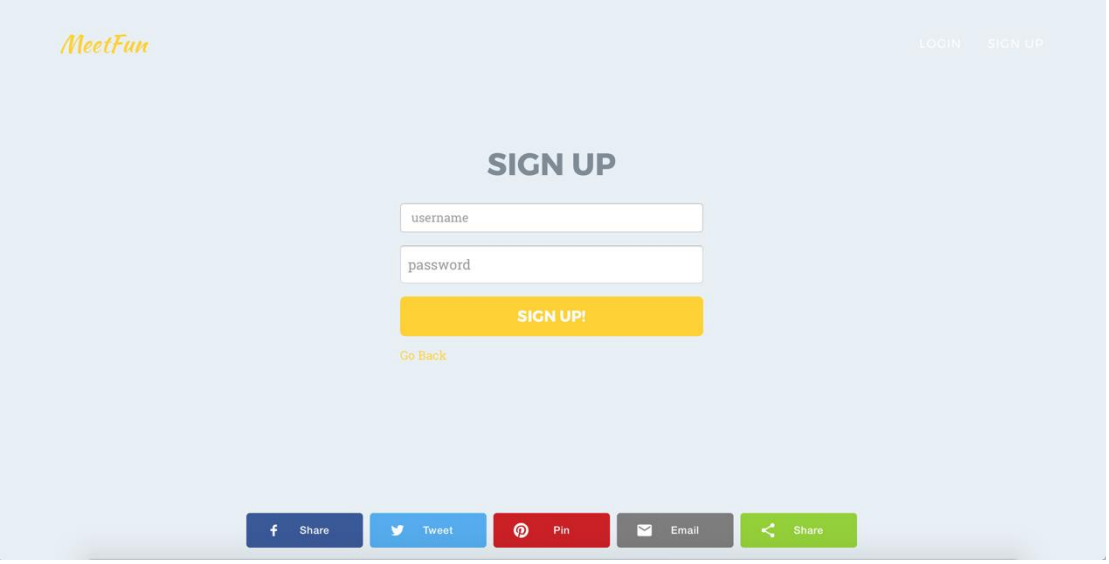


Figure 34 sign up

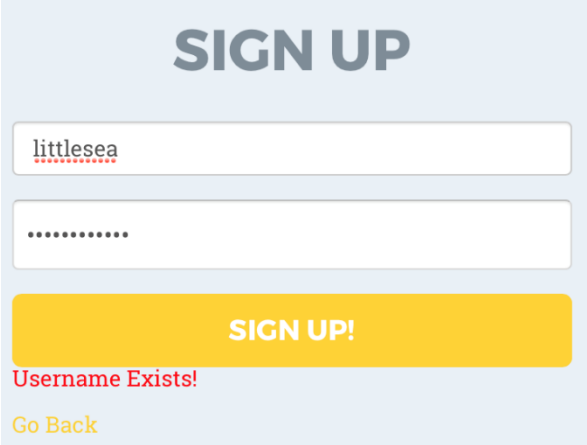


Figure 35 sign up - error

3. Account Management

After successfully log in, the user can click on their username at the top right corner of the header to go to their account management page (Figure 36).

The account management page firstly contains the basic information of the user. Scrolling down, "my funs" (activities the user created) and "my activities" (activities the user joined) would show up (Figure 37 & Figure 38).

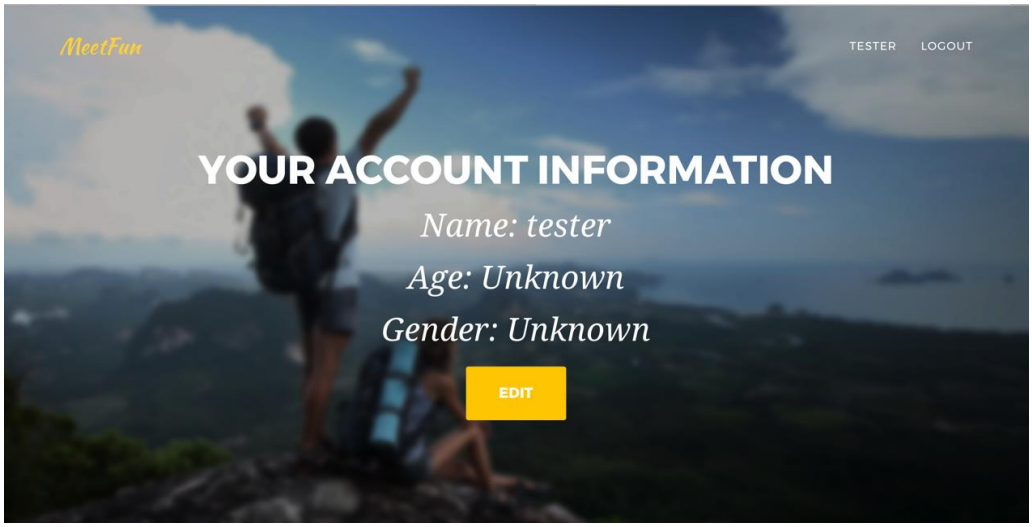


Figure 36 account information

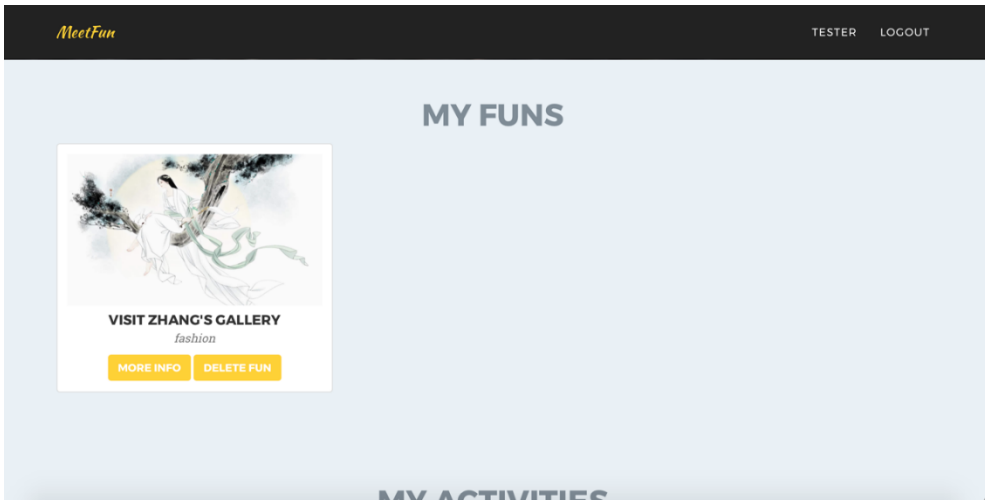


Figure 37 my funs

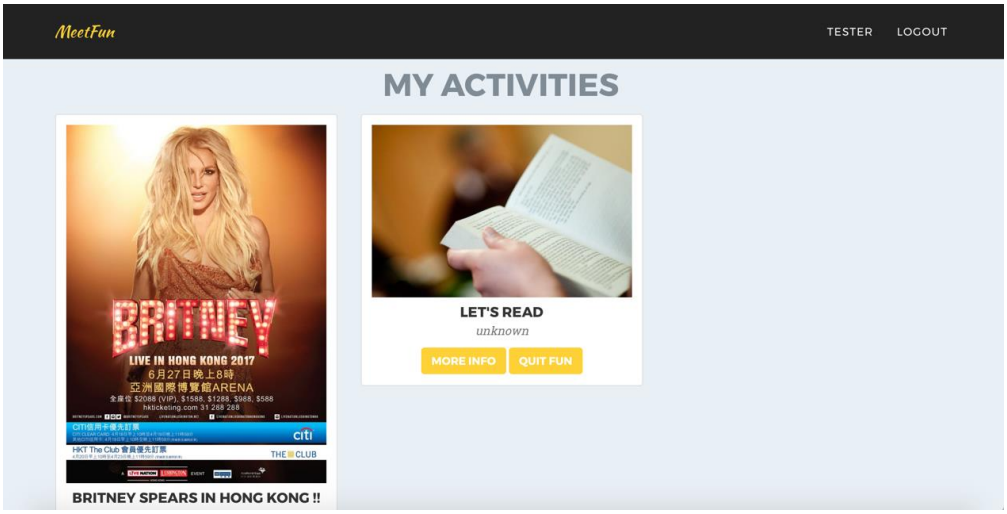


Figure 38 my activities

The user can click the "edit" button to enter the information editing page and change his basic information (Figure 39). If he wants to further change his password, he can click the "change password" button at the bottom of the page (Figure 40).

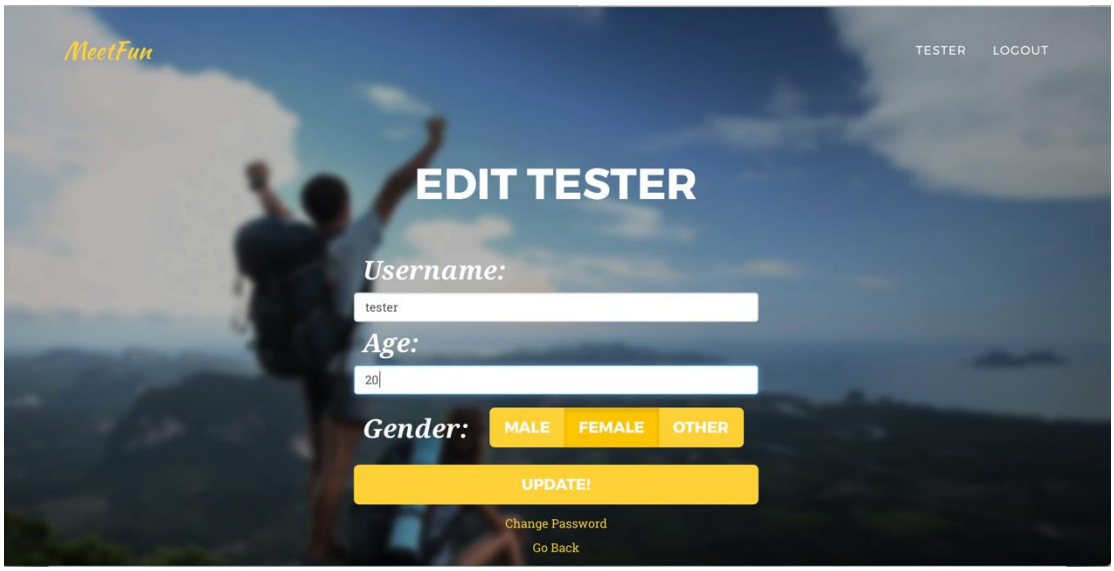


Figure 39 edit account information

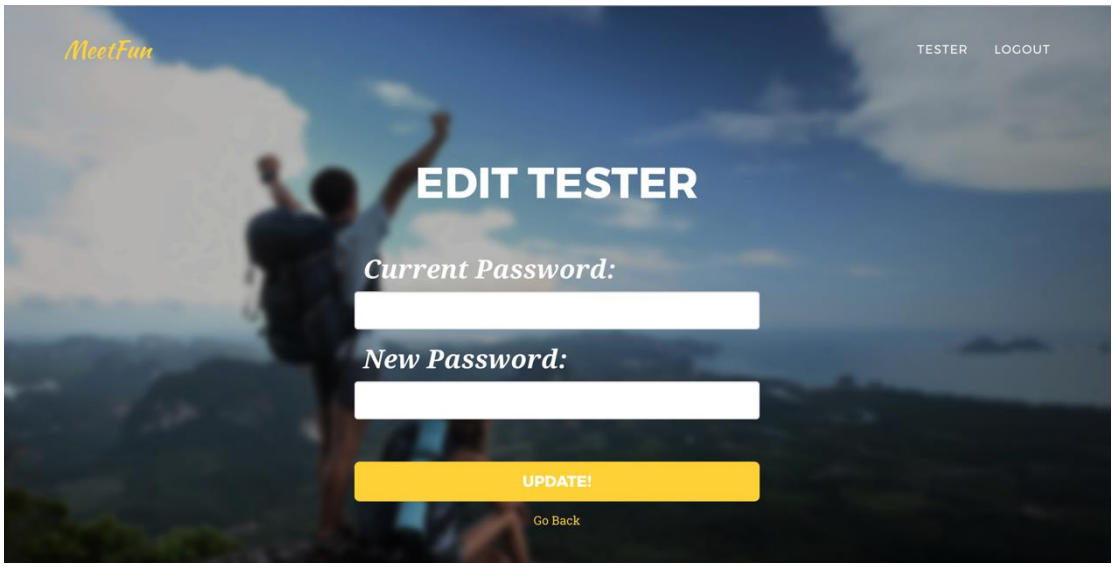


Figure 40 change password

4. Browse and Search

User can search by keyword or tag in the landing page and funs relate to the keyword or tag would be shown. For example, we search keyword "read" in the search page, and the activity "Let's read" would be shown for the user (Figure 41 & Figure 42).

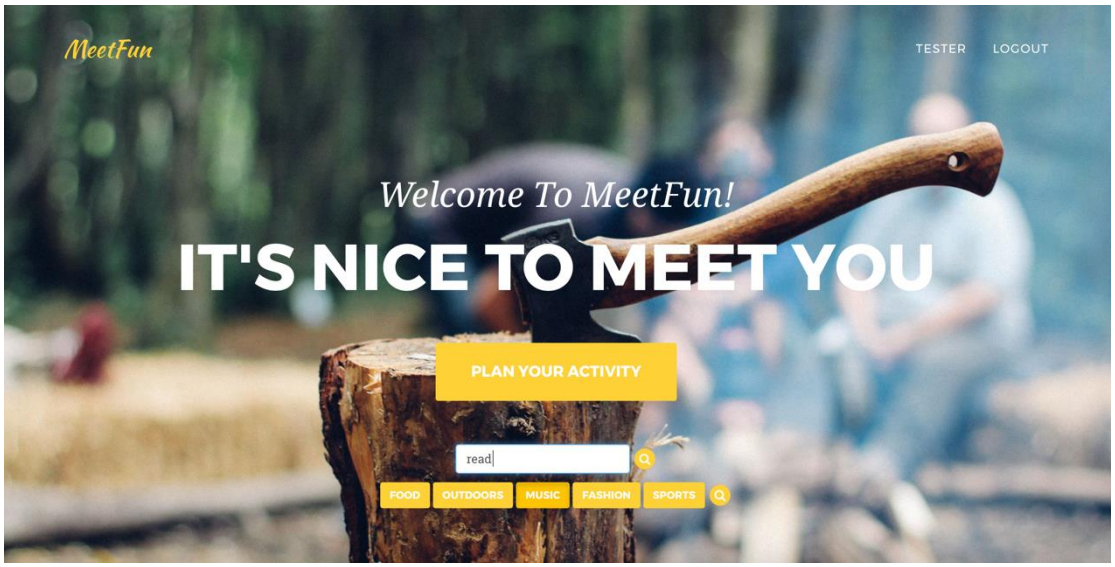


Figure 41 search keyword "read"

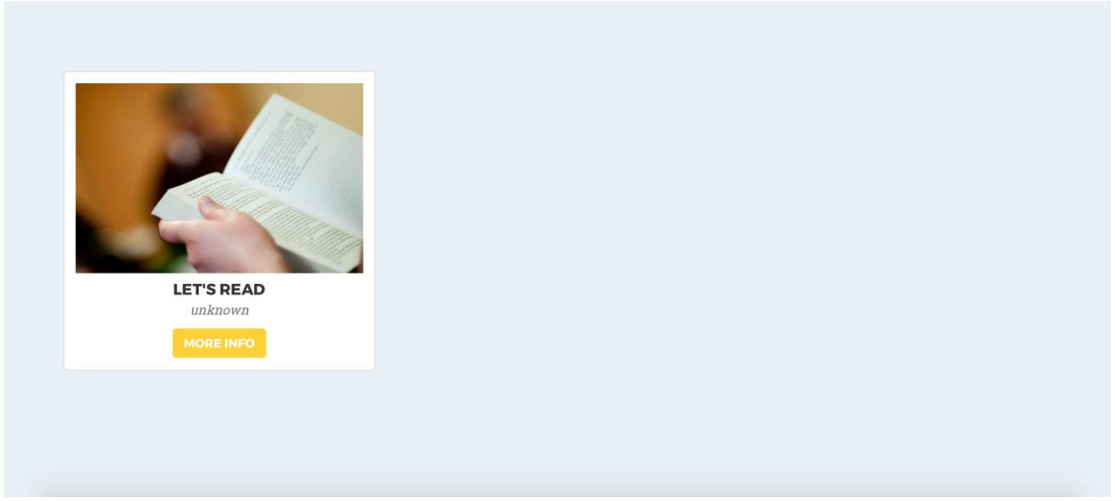


Figure 42 result of searching "read"

5. Fun System

5.1 Create new fun

User can create their own fun after login by clicking the "plan your activity" button in the landing page (Figure 30). The user need to enter some basic information and choose a tag for his fun (Figure 43).

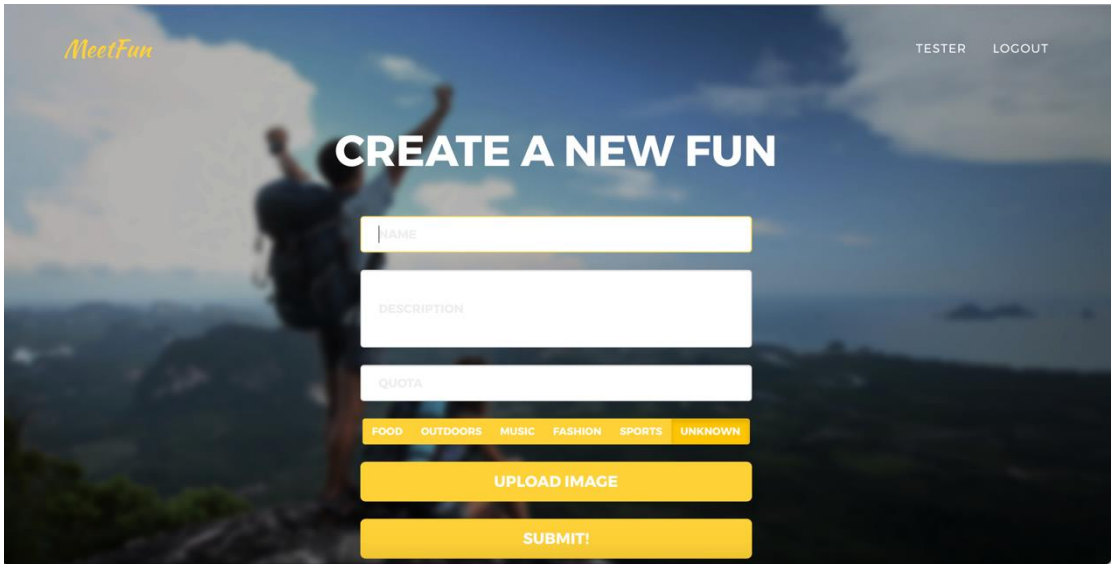


Figure 43 create new fun

The user can also upload an image to attract other users to join. Upon uploading the image, the background of the page would be automatically changed into the image to give a real-time preview for the user (Figure 44 & Figure 45).

Daily details of this activity can be edited under the basic information part (Figure 46).

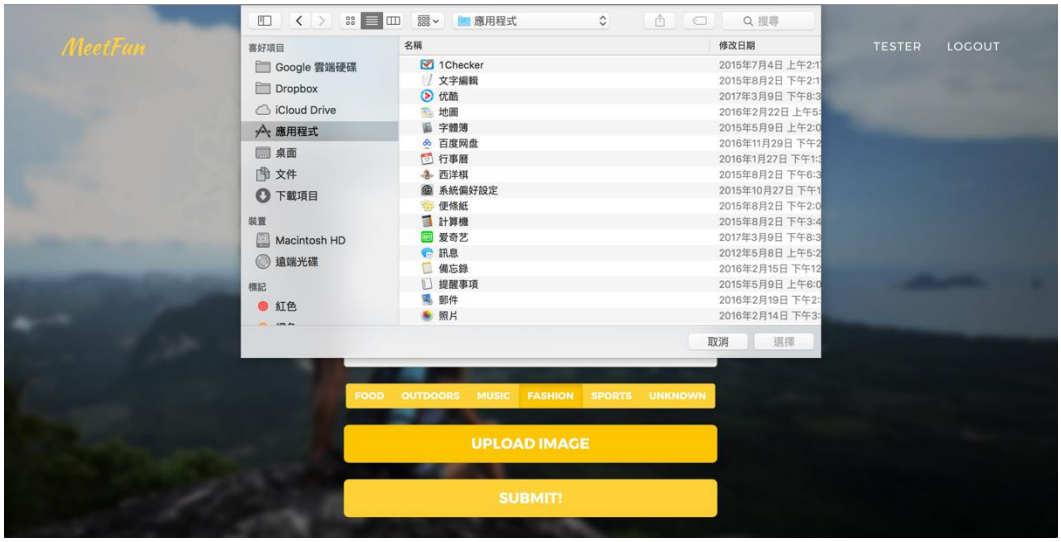


Figure 44 upload image

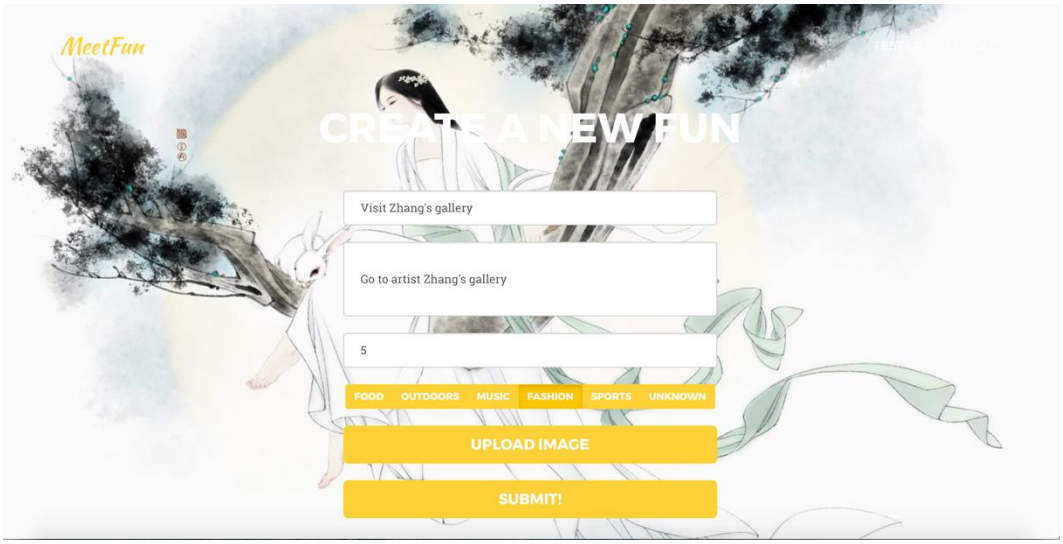


Figure 45 background preview

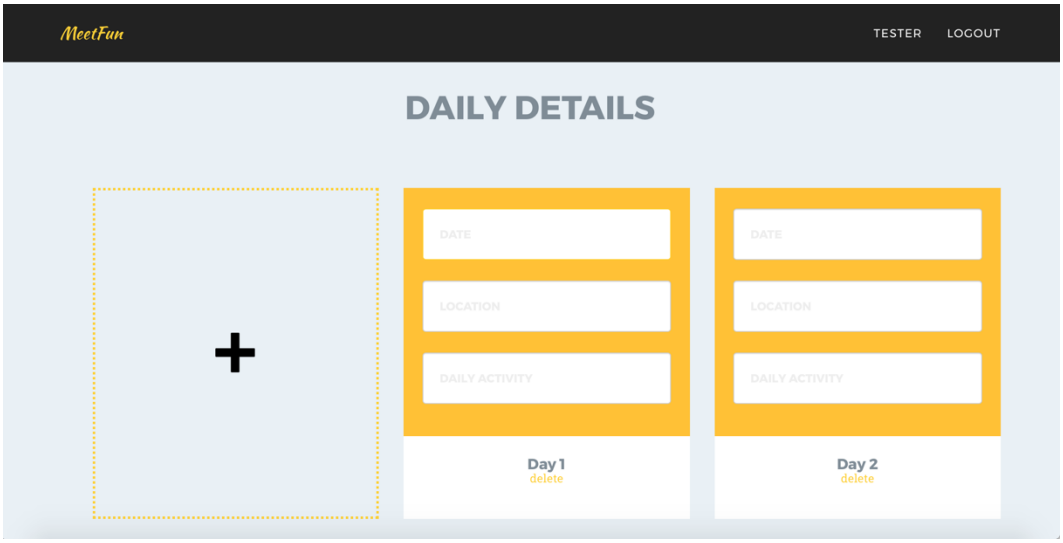


Figure 46 daily details

5.2 Edit fun

The author of the fun activity can edit the activity by clicking the "edit fun" button in the fun page or his account management page. The interface and process of editing is similar with creating a new fun. Besides, the "edit fun" button in the fun page is only visible to the author of the fun (Figure 47).

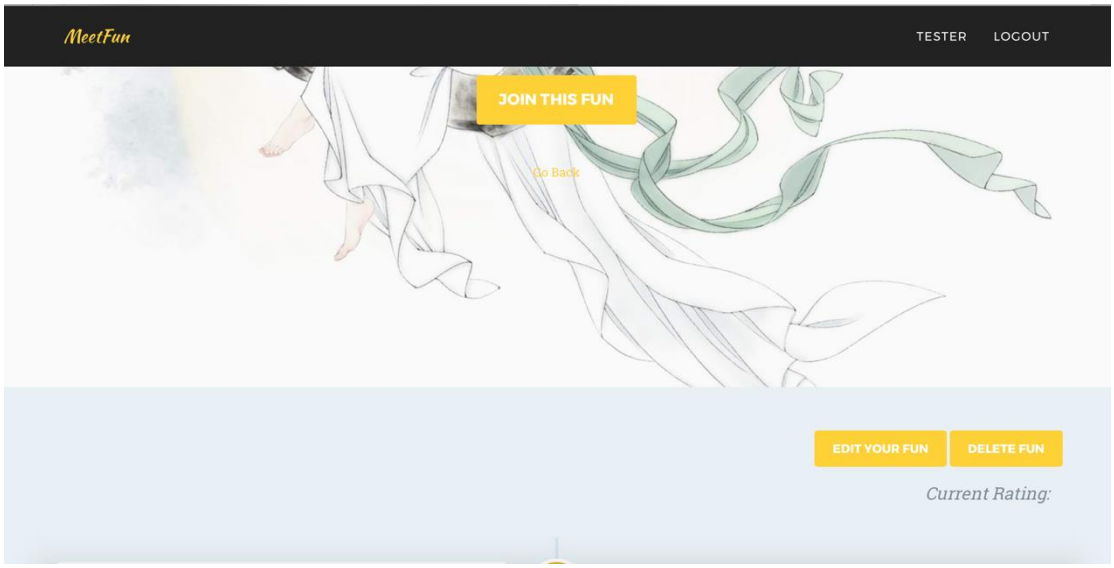


Figure 47 edit fun

5.3 View a fun

By clicking the "more info" button under the fun, user can go to the individual fun page and view the detailed information of this fun.

Entering the individual fun page, the user would firstly see a page of the basic information of this fun. There's also a "join this fun" button for the user to join this fun if he is interested in this fun. (Figure 48)



Figure 48 . fun page – basic information

Scrolling down, the user would see the daily details and current rating of this activity (Figure 49). Following the detail part, there's a comment block where the user can view the comment history, rate for the activity and add a comment (Figure 50).

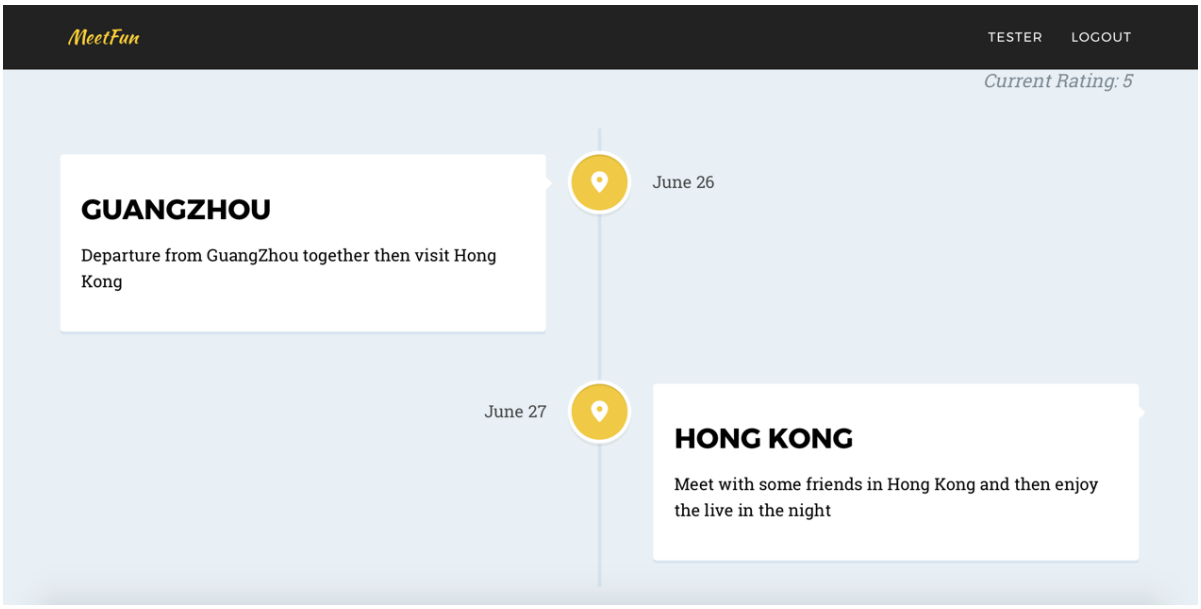


Figure 49 daily details block

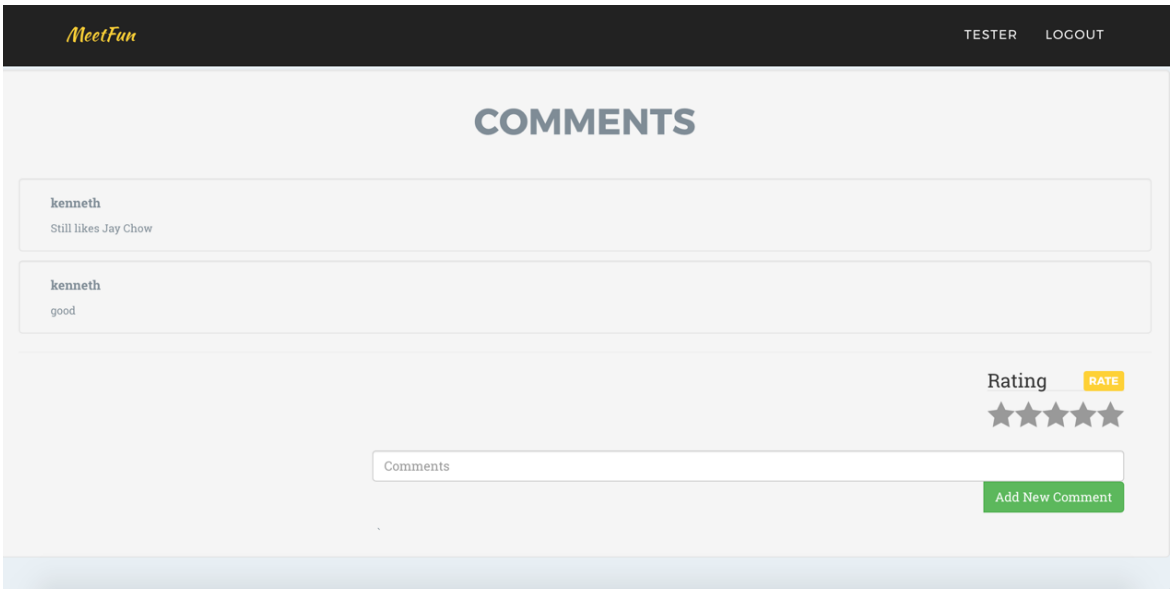


Figure 50 comment block

5.4 Join a fun

User can join the fun they are interested in by clicking the "join this fun" button in the individual page of the fun (Figure 19). However, every fun has a quota for their participants. Therefore, the join action would fail if the fun is full. Besides, alerts would occur to tell the result of the joining action (Figure 51).

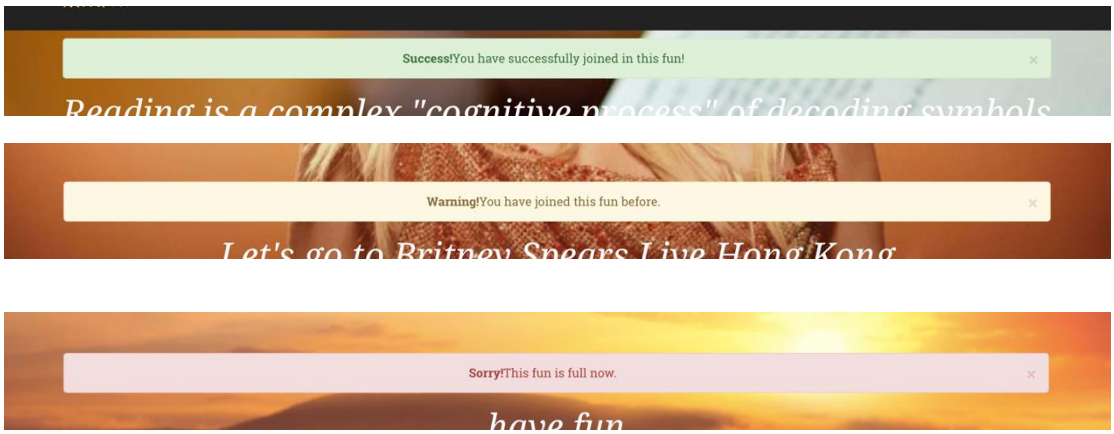


Figure 51 join activity – alerts

5.5 Quit and Delete fun

Users can quit the fun they have joined before in their account management page by clicking the "quit fun" button under the activity (Figure 38).

Similarly, users can also delete the fun they created by clicking the "edit fun" button in their account management page or the individual page of the fun (Figure 37).

5.6 Comment and Rating

Users can make comments on funs they are interested in under the fun page (Figure 21). Users can also edit or delete the comments they made before by clicking corresponding buttons next to the comment (Figure 52).

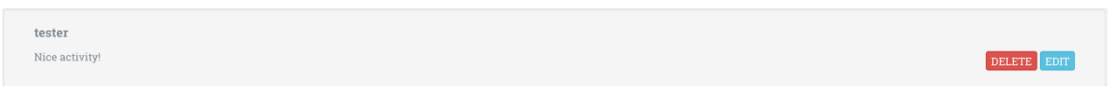


Figure 52 edit/delete comment

Participants of the fun activity can rate the fun activity based on their experience of joining this fun. The average rating of all participants would be calculated and showed in the fun page (Figure 49). Upon rating, alerts would also appear to show the result of the rating action (Figure 53).

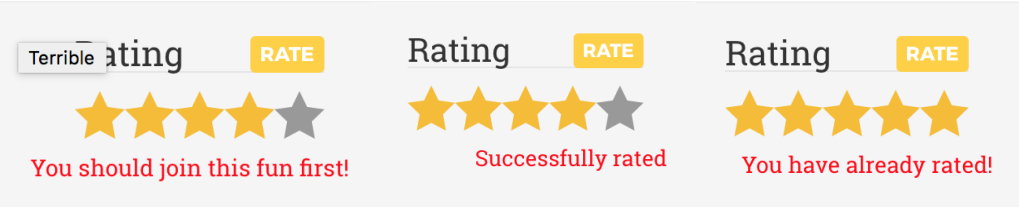


Figure 53 rating – alerts

Tests

1. Test Overview

1.1 Test Approach

Design different test sets to verify each function compared our specification. In addition, some potential or unexpected bugs could be uncovered hopefully.

1.2 Features to Be Tested

- Register
- Login
- Edit Personal Information
- Search
- Create Fun (create activity)
- Edit Fun (edit activity)
- Delete Fun (delete planning)
- Join Fun (join activity)
- Comment
- Rating

1.3 Testing Environment

- Cloud9 (Server side)
- Chrome (Client side)

1.4 Test Coverage

Statement coverage

2. Case - Register

2.1 Purpose

Test the Register function. Check whether the account can be inserted into database and whether an existing username can be avoided.

2.2 Inputs

- case 1:

  username: liruizhe666

  password: liruizhe
- case 2:

  nickname: liruizhe666

  password: liruizhe
- case 3:

  nickname: xixihaha

  password: xixihaha

2.3 Expected Outputs

- case 1: Directed to home page with an account logged as liruizhe666.
- case 2: Alert "Username Exists!".
- case 3: Directed to home page with an account logged as xixihaha.

2.4 Pass Criteria

The username should be the first time to be used for register, no duplicate username is allowed.

3. Case - Login

3.1 Purpose

Check whether a registered account can log in the system successfully.

3.2 Inputs

- case 1:

  username: tester

  password: 111111
- case 2:

  username: (blank)

  password: 222222
- case 3:

username: tester

password: 999999

3.3 Expected Outputs

- case 1: the webpage turns to welcome page
- case 2: alert 'Login failed'
- case 3: alert 'Login failed'

3.4 Pass Criteria

Find out the same username and password in the database.

4. Case - Edit Personal Information

4.1 Purpose

Check whether the user can successfully edit his personal information.

4.2 Inputs

- case 1:

  Name: liruizhe666

  Age: (type: number) 20

  Gender: (type: radio) Male

  Change Password: No
- case 2:

  Name: liruizhe666

  Age: (type: number) twenty

  Gender: (type: radio) Male

  Change Password: No
- case 3:

  Name: ET

  Age: (type: number) 100

  Gender: (type: radio) Other

  Change Password: No
- case 4:

  Name: ET

  Age: (type: number) 100

Gender: (type: radio) Other

Change Password: Yes

Current Password: 2017

New Password: 1234

4.3 Expected Outputs

- case 1: Personal information updated!
- case 2: Personal information remains the same. (Age type error)
- case 3: Personal information updated!
- case 4: Password Updated!

4.4 Pass Criteria

Age should be number type.

5. Case - Search

5.1 Purpose

Check whether the fuzzy search has been achieved.

5.2 Input

- case 1:

  Search Name: NBA
- case 2:

  //test the typo tolerance

  Name: NB
- case 3:

  Name: (blank)
- case 4:

  Tag (type: radio): MUSIC
- case 5:

  Name: UFO

5.3 Expected Outputs

- case 1: NBA activity displayed!
- case 2: NBA activity displayed!

- case 3: All activities displayed!
- case 4: Britney SPEARS IN HONG KONG!! activity displayed
- case 5: NO FUNS MATCH, PLEASE TRY AGAIN.

5.4 Pass Criteria

No particular requirement. The search name should be something related in the database.

6. Case - Create Fun

6.1 Purpose

Check whether a user can create a fun successfully. To make sure that, at least one day is needed to create a fun.

6.2 Inputs

- case 1:

  Name: "test1"

  Description: "test1"

  Quota: 13

  Label: "food"

  File: FunsCreate.jpg

  Days: [{date: "5/5/17", location: "Hong Kong", dailyActivity: "Play"}]

- Case 2:

  Name: ""

  Description: ""

  Quota: Null

  Label: ""

  File: Null

  Days: []

- Case 3:

  Name: "test3"

  Description: ""

  Quota: Null

Label: “”

File: Null

Days: []

- Case 4:

  Name: “test4”

  Description: “test4”

  Quota: 12

  Label: “outdoor”

  File: FunsCreate.jpg

  Days: []

- Case 5:

  Name: “test5”

  Description: “test5”

  Quota: Null

  Label: “”

  File: Null

  Days: [{date: “”, location: “”, dailyActivity: “”}]

- Case 6:

  Name: “test6”

  Description: “test2”

  Quota: 10

  Label: “outdoor”

  File: FunsCreate.jpg

  Days: [{date: “4/5/2017”, location: “HongKong”, dailyActivity: “study”}, {date: “5/5/2017”, location:”CUHK”, dailyActivity: “sleep”}, {date: “6/5/2017”, location: “PolyU”, dailyActivity: “PlayFootball”}]

- Case 7:

  Name: “test6”

  Description: “test2”

  Quota: 10

Label: “outdoor”

File: Null

Days: [{date: “4/5/2017”, location: “HongKong”, dailyActivity: “study”}, {date: “5/5/2017”, location:”CUHK”, dailyActivity: “sleep”}, {date: “6/5/2017”, location: “PolyU”, dailyActivity: “PlayFootball”}]

6.3 Expected Outputs

- Case 1: The activity is successfully created.
- Case 2: The activity creation is failed.
- Case 3: The activity creation is failed.
- Case 4: The activity creation is failed.
- Case 5: The activity is successfully created
- Case 6: The activity is successfully created
- Case 7: The activity is successfully created without image

6.4 Passing Criterion

To create a fun (activity) post, at least one day should be added to the schedule.”

7. Case - Edit Fun

7.1 Purpose

To make sure that an author can edit his/her own activities.

7.2 Inputs

- case 1:

  Name: “test1”

  Description: “test1”

  Quota: 20

  Label: “food”

  File: FunsCreate.jpg

  Days: [{date: “5/5/17”, location: “Hong Kong”, dailyActivity: “Play”}]

- Case 2:

  Name: “test1”

Description: “test1”

Quota: twenty

Label: “food”

File: FunsCreate.jpg

Days: [{date: “5/5/17”, location: “Hong Kong”, dailyActivity: “Play”}]

- Case 3:

  Name: “test1”

  Description: “test1”

  Quota: 13

  Label: “music”

  File: FunsCreate.jpg

  Days: [{date: “5/5/17”, location: “Hong Kong”, dailyActivity: “Play”}]

- Case 4:

  Name: “test1”

  Description: “test1”

  Quota: 13

  Label: “music”

  File: Null

  Days: [{date: “5/5/17”, location: “Hong Kong”, dailyActivity: “Play”}]

- Case 5:

  Name: “test1”

  Description: “test1”

  Quota: 13

  Label: “music”

  File: Null

  Days: []

7.3 Expected Outputs

- Case 1: The activity is successfully updated
- Case 2: The activity edition is failed (quota type error)
- Case 3: The activity is successfully updated
- Case 4: The activity is successfully updated without image

- Case 5: The activity edition is failed

7.4 Pass Criterion

The modification of the activity should make sure that:

1). There is at least one day in the schedule;

2). All other fields should meet with their designed type

8. Case - Delete Fun

8.1 Purpose

To check whether an author could delete his/her own activities.

8.2 Input

- Case 1:

  User: {name: "Ray", ID: "590c47513387fdb2173d27fe"}

  Fun: {name: "testCase1", author: {username: "Ray", userID: "590c47513387fdb2173d27fe"}}
- Case 2:

  User: {name: "Good", ID: "590c3f873387fdb2173d27f5"}

  Fun: {name: "testCase1", author: {username: "Good", userID: "590c3f873387fdb2173d27f5"}}
- Case 3:

  User: {name: "Guy", ID: "590af6e15444831824dce7c7"}

  Fun: {name: "testCase1", author: {username: "Guy", userID: "590af6e15444831824dce7c7"}}

8.3 Expected Outputs

- Case 1: Successfully delete.
- Case 2: Successfully delete.
- Case 3: Successfully delete.

8.4 Pass Criterion

Only the author of the activity has the access to delete the activity, the deletion method is not accessible to other users

9. Case - Join Fun

9.1 Purpose

To make sure, the user cannot participate in a particular activity, only if that activity's quota is full.

9.2 Inputs

- Case 1:

  User: {name: "Ray", ID: "590c47513387fdb2173d27fe"}

  Fun: {…, name: "testCase0", quota:4,

  participants: [{username: "ET", …}, {username: "Liruizhe666", …},

  {username: "Leo", …}, {username: "Mark", …}]

  }

- Case 2:

  User: {name: "Good", ID: "590c3f873387fdb2173d27f5"}

  Fun: {…, name: "testCase6", quota:20,

  participants: [{username: "ET", …}, {username: "Liruizhe666", …},

  {username: "Leo", …}, {username: "Mark", …}]

  }

- Case 3:

  User: {name: "Guy", ID: "590af6e15444831824dce7c7"}

  Fun: {name: "testCase1", author: {username: "Guy", userID:

  "590af6e15444831824dce7c7"}}

9.3 Expected Outputs

- Case 1: Fail at joining in the activity. (Quota is full)
- Case 2: Successfully join in the activity and show on personal page.
- Case 3: No action. User already joined this activity.

9.4 Pass Criterion

Users cannot join activities that are full (all the quota of the activity has been taken up).

10. Case - Quit Fun

10.1 Purpose

To make sure that a participant can successfully quit a joined activity.

10.2 Inputs

- Case 1:

  User: {name: "ET", ID: "590c47513387fdb2173d27fe"}

  Fun: {…, name: "testCase0", quota:4,

  participants: [{username: "ET", …}, {username: "Liruizhe666", …},

  {username: "Leo", …}, {username: "Mark", …}]

  }

- Case 2:

  User: {name: "Good", ID: "590c3f873387fdb2173d27f5"}

  Fun: {…, name: "testCase6", quota:20,

  participants: [{username: "ET", …}, {username: "Liruizhe666", …},

  {username: "Leo", …}, {username: "Mark", …}]

  }

10.3 Expected Outputs

- Case 1: "testCase0" removed from ET's personal page and the number of participants reduce to 3.
- Case 2: "testCase6" removed from Good's personal page and the number of participants reduce to 3.

10.4 Pass Criterion

User can quit activities they've joined at any time. Once a user quit from a particular activity, the website should display that the activity has been removed from his/her activity list. The remaining quota of the activity should also be modified.

11. Case - Comment

11.1 Purpose

Test the comment module. Check whether the comment module can provide an understandable and reliable way for users to create, edit and delete their own comments. The situation where a user or a visitor (who did not log in) wants to edit or delete other users' comments will also be covered.

11.2 Inputs

For testing purpose, we create 2 user account X and Y, and we denote a visitor who hasn't logged in as V, a fun post as F.

- Case 1 (whether a visitor who didn't log in can leave comment):
  - Subcase 1.1: V enter a fun post and enter text in comment box, click "Add New Comment".
  - Subcase 1.2: V enter a fun post, without entering text, click "Add New Comment".

- Case 2 (whether a visitor who didn't log in can edit and delete comment):
  - Subcase 2.1: V enter a fun post, click around the comments and try to edit or delete comments by other users.

- Case 3 (whether a user can create comments):
  The subcases should execute in sequential order
  - Subcase 3.1: X enter a fun post F, enter nothing in the comment box, click "Add New Comment" to create comment X1 for F.
  - Subcase 3.2: X enter a fun post F, enter 1000 symbols of random combination, click "Add New Comment" to create comment X2 for F.

- Case 4 (whether a user can edit and delete comments wrote by other users):
  - Subcase 4.1: Y enter a fun post F, click around the comments and try to edit or delete comments by user X.

- Case 5 (whether a user can edit and delete comment wrote by himself/herself):

- Subcase 5.1: X enter a fun post F, if X1 exist, try to edit it and we still name it X1; if X1 doesn't exist, Case 3 fails.
- Subcase 5.2: X enter a fun post F, if X2 exist, try to edit it and we still name it X2; if X2 doesn't exist, Case 3 fails.
- Subcase 5.3: X enter a fun post F, if X1 exist, try to delete it; if X1 doesn't exist, Case 3 fails.
- Subcase 5.4: X enter a fun post F, if X1 exist, Case 3 fails.
- Subcase 5.5: X enter a fun post F, if X2 exist, try to delete it; if X2 doesn't exist, Case 3 fails.
- Subcase 5.6: X enter a fun post F, if X2 exist, Case 3 fails.

11.3 Expected Outputs

- case 1: V are asked to log in every time
- case 2: V cannot find any button to edit or delete
- case 3: X are able to create both X1 and X2
- case 4: Y cannot find any button to edit or delete X1, X2
- case 5: X can edit X1, X2 ; X can delete X1, X2

11.4 Pass Criteria

5 cases and their subcases should be executed sequentially. The expected output is met and there is no X1, X2 remains in the comments.

12. Case - Rating

12.1 Purpose

Test the rating module. The rating function is only available to users who have joined the "funs" and every user can rate the fun exactly once. Visitors who didn't log in cannot rate the "funs". Every time the "fun" received a rating, its rating score should be updated to the average of all joined users' rating score.

12.2 Inputs

For testing purpose, we create 3 user accounts X, Y and Z, and we denote a visitor who hasn't logged in as V, a fun post as F, another fun post as E.

We consider the test cases are in sequential executing order and the result of the current test case will influence following test cases.

Note that we do not cover the case when the "fun" is out of quota, which is covered in "Case - Join Fun".

- Case 1 (test behavior for visitors who didn't log in):
  - Subcase 1.1: V enter a fun post F, select the star numbers and click "Rate"
  - Subcase 1.2: V enter a fun post F, without selecting stars, click "Rate"
- Case 2 (test behavior for users who didn't join):
  - Subcase 2.1: X enter a fun post F, select the star numbers and click "Rate"
  - Subcase 2.2: X enter a fun post F, without selecting stars, click "Rate"
- Case 3 (test behavior for users who joined):
  - Subcase 3.1: X enter a fun post F, join the fun F, select a star, click "Rate"
  - Subcase 3.2: X enter a fun post E, join the fun E, without selecting stars, click "Rate"
- Case 4 (test behavior for averaging multiple ratings):
  - Subcase 4.1: Y enter a fun post F, join the fun F, select a star, click "Rate"
  - Subcase 4.2: Z enter a fun post F, join the fun F, select a star, click "Rate"
- Case 5 (test behavior for multiple rating attempts from one user):
  - Subcase 5.1: X enter a fun post F, select a star, click "Rate"

12.3 Expected Outputs

- case 1: V are asked to log in every time
- case 2: X are asked to join the activity first every time
- case 3: X can successfully rate the fun and the rating will be display
- case 4: Y, Z can successfully rate the fun and the rating will be changed into the average value of all ratings, with limited digits behind the decimal point
- case 5: X will be alerted that he/she already rated, and the rating result won't change

12.4 Pass Criteria

5 cases and their subcases should be executed sequentially. The expected output is met and the rating of the fun is in correct number

Lessons Learned

1. Software Engineering Skills

Out team demonstrated the software engineering principles perfectly during the construction of this web application.

The specification was stated clearly for most parts of the application; however, some specifications was still ambiguous and the estimation of implementing cost was far from accurate so that we had to cut off some of the functions in our first release. Next time if our team were assigned another project, we believe we could state the requirements more clearly making programming phase easier.

We addressed modularity in our development process. Team members are assigned according to the different functionality of the system. The separation of concerns was successful but some problem still occurred. As the front and back end of the system was developed separately, it was often the case when one end is ready for debugging while the other end wasn't so that he had to either wait or wrote a driver or stub which caused excess workload and lower the efficiency. Also, the integration was sometimes messy when different team members were trying to integrate their latest work into the application simultaneously. A more efficient version control could be used next time.

The testing period was conducted smoothly without too much drawbacks. Under large number of cases of black-box testing all the functions performed as expectation under normal using environments.

2. Programming Skills

This is the first time for the team members contacting with the most fashionable web application developing frameworks – Nodejs, MogoDB, Bootstrap and JQuery. Many useful skills regarding these technologies are learnt and all the team members are equipped with basic full-stack development skills.

Conclusion

This is the first time for all the team members to experience a complete life cycle of a software – designing, programming, testing and maintaining. The application of MeetFun is still immature comparing with commercial software of websites but it is already a functioning application. Team members combined the knowledge learned in the lectures and implemented it into real world case. We are more familiar with how the theories learned like software engineering principles and tools like DFDs and UMLs are used in the software industry. New skills in web application developing are also learnt which could be useful in the future job-seeking process. This is a meaningful project and all the team members gained valuable experience during this process.