SOFTWARE DESIGN DOCUMENT

for

iSport

Release 1.0

Prepared by

1650940 Jiang Xiaohu
1650932 Xu Jingnan
1651058 Wang Yicheng

School of Software Engineering
Tongji University

January 4, 2020

Table Of Contents

Revision History

Name	Date	Reason For Changes	Version
Jiang Xiaohu	2019.11.12	Finish Introduction Part	v1.0
Wang Yicheng	2019.11.14	Finish Overview Part	v1.1
Xu Jingnan	2019.11.16	Finish External Requirements Part	v1.2
Xu Jingnan, Jiang Xiaohu, Wang Yicheng	2019.11.18	Finish Sequence Diagrams for Function Modeling Part	v1.3
Xu Jingnan	2019.11.20	Finish Function Modeling Part	v1.4
Jiang Xiaohu	2019.11.21	Finish Data Modeling Part	v1.5
Wang Yicheng	2019.11.22	Finish Behavior Requirements Part	v1.6
Xu Jingnan	2019.11.23	Finish Nonfunctional Requirements Part	v1.7
Jiang Xiaohu	2019.11.25	Finish Data Dictionary Requirements Part	v1.8

1 Introduction

1.1 Purpose

This design document describes the overall structure of iSport by outlining significant aspects of the systems architecture.

The purpose of this document is to present a detailed description of iSport. It shows how the software system will be structured to satisfy the requirements.

1.2 Scope

This software system will be a web based system for sports fans, professional athletes, and patients who are under recovering training.

This system will be designed to maximize the exercising efficiency by providing tools to assist in checking and correcting user's wrong postures and recommending training courses customized for users, which would otherwise have to be expensive, time-consuming and labor intensive. By maximizing the users training efficiency and convenience the system will meet the needs of sports fans, athletes and injured patients while remaining easy to understand and use.

More specifically, this system is designed to allow a user to imitate the standard exercising postures while observe and correct their mistakes simultaneously with the help of a website.

The software will collect some professional courses in the database, including static and dynamic trainings which means doing exercise according to

a set of images or a video and iSport will recommend suitable trainings for users on the basis of their training performance. Courses are classified into exercising courses and recovering courses, aiming to help athletes and patients respectively.

Both visual and audio notification are used in every course of the system to provide eye-catching, user-friendly and clear instructions; the feedback of one's training is proposed once the training is over and the report can be browsed in the report page.

The selection and deletion of one user's favorable course is supported in personal information webpage and one can comment training he/she has taken on comment webpage to provide suggestions to other users.

The personal information registering, changes is allowed via the application options. The system also contains a relational database containing a list of users, training images and videos.

1.3 Acronyms, Abbreviations and Definitions

Table 1.1: Definitions

Definitions, Acronyms, and Abbreviations	
Term	Definitions
User	Someone who interacts with iSport including sports fans, athletes and injured patients who need recovery training.
Sports fans	One of iSport's potential customers who love sports and want to get professional instructions when exercising. Some of them may can't afford the expense of personal coaching or don't have time to go to the gym.
Continued . . .	

Table 1.1: (continued)

Definitions, Acronyms, and Abbreviations (continued)	
Athletes	One of iSport's potential customers who want to get real-time exercising feedback to improve their performance or who want to get some relaxing training in their spare time to keep a good competitive state.
Injured Patients	One of iSport's potential customers who need recovering training after some treatments, e.g. surgeries. On the one hand, some of them may can't afford the doctor's expensive medical instructions for recovering training. on the other hand, there is no enough doctors or nurses who can instruct and supervise the patients' recovering exercising. But without professional instructions training can be useless or even leads to secondary trauma.
Admin/Administrator	System administrator who is given specific permission for managing and controlling the system, e.g. updating the user's information, uploading new training courses.
User Info	User's basic information including user's avatar, account name, tel-number and email address.
Courses	Training courses including normal exercising training and recovering training.
Exercise Courses	Training courses which serve the sports fans and athletes.
Recover Courses	Training courses which serve the injured patients.
Static Courses	Training courses which instruct the users photo by photo.
Dynamic Courses	Training courses which instruct the users according to a standard video.
Continued . . .	

Table 1.1: (continued)

Definitions, Acronyms, and Abbreviations (continued)	
Appraisal Subsystem	Remark the user's performance by using a grade from 0 -100
Comment Subsystem	User comment on the training courses they have taken to provide reference for other users.
Recommendation Subsystem	A subsystem which will provide some courses for users according to their recent performance.
Exercise Tips	There will be sports tips in the webpage of iSport to prevent users from athletic injuries.
Sport Report	A web page to feedback the user's exercising performance.
Audio Notification	An audio notification will be shown when the user is doing exercise to encourage the user to hold on or notify the user to correct their postures.
Visual Notification	A visual notification will be shown when the user is doing exercise, if the user's posture is standard, then the web-frame will turn green to suggest the user to hold on, otherwise the web-frame will be red.
DataBase	A relational database containing a list of user info, training images and videos.
Detection Subsystem	Subsystem to detect the user's postures and draw the user's skeleton. The main model of detection subsystem is PoseNet.
Comparison Subsystem	Subsystem to compare the postures of the user and that of the standard. The subsystem aims to check if the user pass the posture.
Correction Subsystem	Subsystem to calculate where the postures' wrong part are, e.g. left-arm, right-leg, head.
Continued . . .	

Table 1.1: (continued)

Definitions, Acronyms, and Abbreviations (continued)	
Clients	Group who delegate the development of iSport to the developers and will take charge of the later management of iSport.
Developers	Develop team including project managers, programmers, testers who are responsible for the development of iSport and its later mainteinance and updating.
The End	

1.4 Overview

This document is an overview of the software architecture of iSport in high detail. We start by providing all the principal components that the application is built on as well as their responsibilities to the success of the application. Next, we provide diagrams to show the hierarchy of our classes and the architectural design. Finally, we develop a general API of the major class methods used to build the functionality of our application. There are also mockups of our UI design included in our design document.

1.5 Reference Material

Standard References

The standards we have followed are as follows:

[1] T. Russell, A. Brizee, E. Angeli, and R. Keck, Mla formatting and style guide, The Purdue OWL, 2010.

[2] Barnard, H Jack and Metz, Robert F and Price, Arthur L et al., A recommended practice for describing software designs: IEEE standards project 1016, 1986.

[3] R. S. Pressman, Software engineering: a practitioners approach. Palgrave Macmillan, 2005.

Writing Tools References

The writing tools we have used are as follows:

[4] L. Lamport, LATEX: a document preparation system: users guide and reference manual. Addison-wesley, 1994.

[5] S. Wong, Staruml tutorial, Connexions Web site, Sep, 2007.

[6] P. O. Team, Process on tools, https://www.processon.com/support.

2 System Overview

2.1 Product Perspective

ISport is a web system developed by isport team, aimming at posture correcting with the assist of camera.

The user-case diagram in the folowing figure illustrates the user-case in the system. The system is expected to evolve over at least three releases, ultimately allowing for complete streamlining of the posture correcting process, fitness classes and rehabilitation classes for learning.

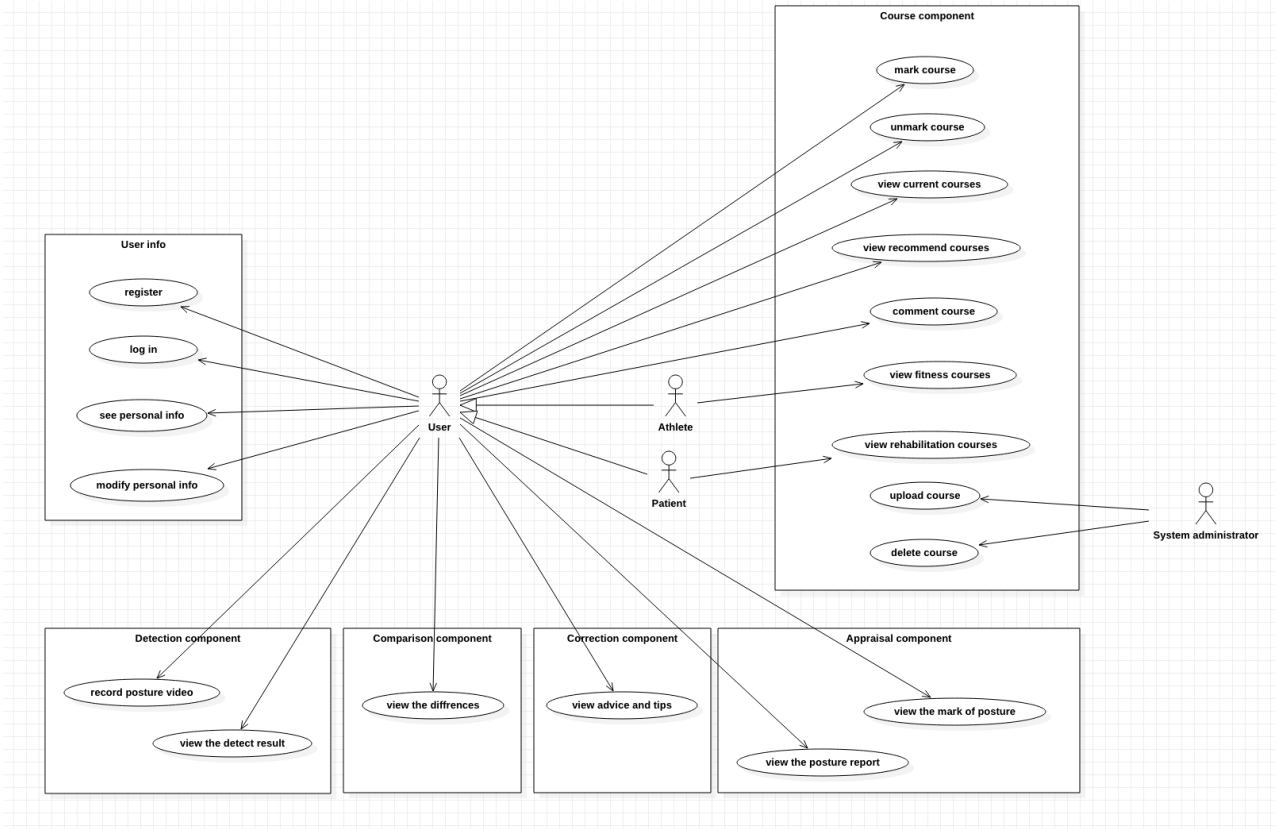


Figure 2.1: overall user-case diagram

2.2 Product Functionality

ISport contains these following key features:

Let the user register their account of the website through their mobile phone or e-mail.

Let the user log in to the system by correctly complete the log-in form. If the username is not in the database, the user will be prompted to register his/her account.

User can modify his/her account information after logging in, including the username, e-mail address, phone number, gender, age and so on.

Show all the sorted fitness courses in a list for user to choose in the fitness

courses view page.

Show a list of courses that the user has selected to learn before

Show a list of recommended courses for the user by user's previous behavior

Show the comment and evaluation of each courses in their detailed page

Provide the add button for user to add a course he/she want to learn in the future and add the course to his/her course list.

Provide the delete button for user to remove a course he/she doesn't want to learn and remove the course from his/her course list.

Generate user's exercise report for watching and analyzing in the report page

Give the user a space to leave his/her evaluation about the courses learnt before

Capture and Collect the posture data of the user in front of the computer and store the data in the server.

Compare user's posture with the standard one and calculate the similarity between the postures.

Show tips on the user's screen to notice the wrong posture of the user and help to correct them.

Rate the user's posture on a scale of zero to ten to let the user know whether his posture is standard or not

System administrator can upload new courses to the system for users to choose and learn.

2.3 Users and Characteristics

Users	Desc
People who want to correct their posture	Normal user is expected to register and log in the system, upload their posture and get the feedback advice from the system
People who want to be fitness	People who want to be fitness is expected to register and log in the system, choose fitness classes for themselves and learn it on the website
People who need rehabilitation	People who need rehabilitation is expected to register and log in the system, choose fitness classes for themselves and learn it on the website
System Administrator	System Administrator has the privilege to update posture information in the database. The Administrator does not directly interact with the website

2.4 Operating Environment

Hardware

1. Server

   CPU: Intel Dual-Core 2.4GHz

   Memory: 8GB

   External Storage: 500G SSD

   Quantity: 1

2. Client(Minimum Configuration)

   CPU: Single-Core 1GHz

Memory: 2G

External Storage: 20GB HDD

Software

1. Server

   Operating System: Ubuntu 18.04 LTS

   Database: MySQL

   Software and Library : Python, Spring, TensorFlow, Nginx

2. Client

   Operating System: Windows/macOS/Linux/Android/IOS

   Software: Browser

Recommand Browser	Version
Google Chrome	44+
Mozilla Firefox	40+
Apple Safari	7+
Microsoft Edge	12+
Microsoft Internet Explorer	11+
Opera Opera	31+

2.5 Design and Implementation Constraints

1. Memory: Server will have 500GB internal hard drive. Softwares and database cannot exceed this amount. System administrator must notice this limitation. And each user should follow the rule that the video data

uploaded each time shall not be greater than 100M

2. Language requirements: software must be multilingual, including the following languages: English, Chinese

3. Number of user: each video uploaded must be one person. Each time this system can only deal with one person's data.

2.6 User Documentation

Along with this system: iSport, a user manual need to be written to help users understand how to operate the system. It would be written for non-technical individuals and the level of content would differ considerably from a system administration guide, which is more detailed and complex. The user manual would follow common user documentation styles to be simple.

Trying to use step-by-step instructions for users who firstly log in to the website, by showing messaging structures, quick references, tips and glossary of terms.

User document can be written in HyperText Markup Language (HTML) or Portable Document Format (PDF) , which must describe the use of the software system.

2.7 Assumptions and Dependencies

It is assumed that the website will work correctly with every third-party operating system and compatible across all of the major browsers.

Assumed that the web server always runs well without down or not responding.

ISport provides two kinds of classes for user to choose: fitness classes and rehabilitation classes.

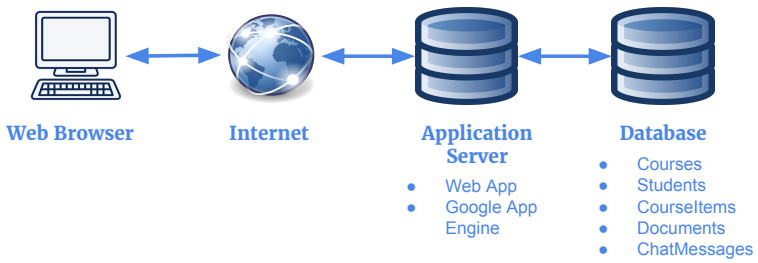
Website visitors who have not been registered are only allowed to watch some demo videos, they are not allowed to upload their posture video data before registering.

In addition to test their posture and attend into courses, members are also allowed to update their information, but they have to log in first.

3 System Architecture

3.1 Architectural Design

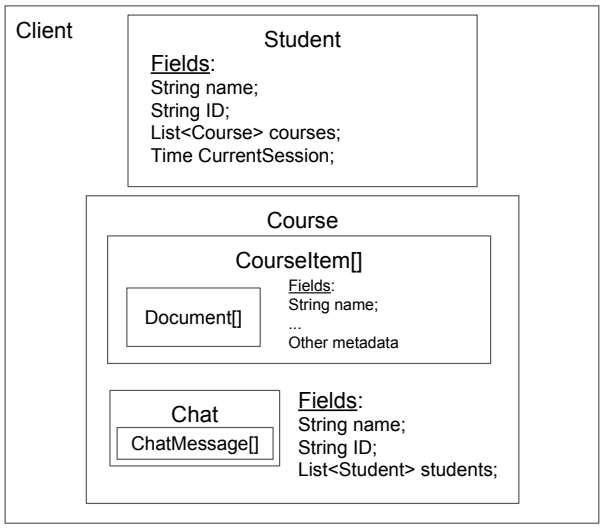
Global Overview



3.2 Description of Achitecture Goals

The Class Collaboration Application will be hosted by the Google App Engine and users will access the app via a web browser. The app will have access to a database containing lists of all the Courses created for the app, corresponding CourseItems, Documents, and Chatmessages, as well as the Students who have created accounts. The database will save the information

Class Structures



for an indefinite amount of time. Our class structures and main use cases by users are shown below.

3.3 System Components Overview

We present a component level design for iSport System and each subsystem.

For example as a subsystem of iSport, Component User Info Management has four main function: register, login, show userinfo, update userinfo, and we can detail these four function into more specific components including: validation checking, user data reading and so on. Sub components of each subsystem is structured as hierarchical relationship.

More detail designs of each subsystem and components will be specified in next chapter.

3.3.1 User Info Management Component Overview

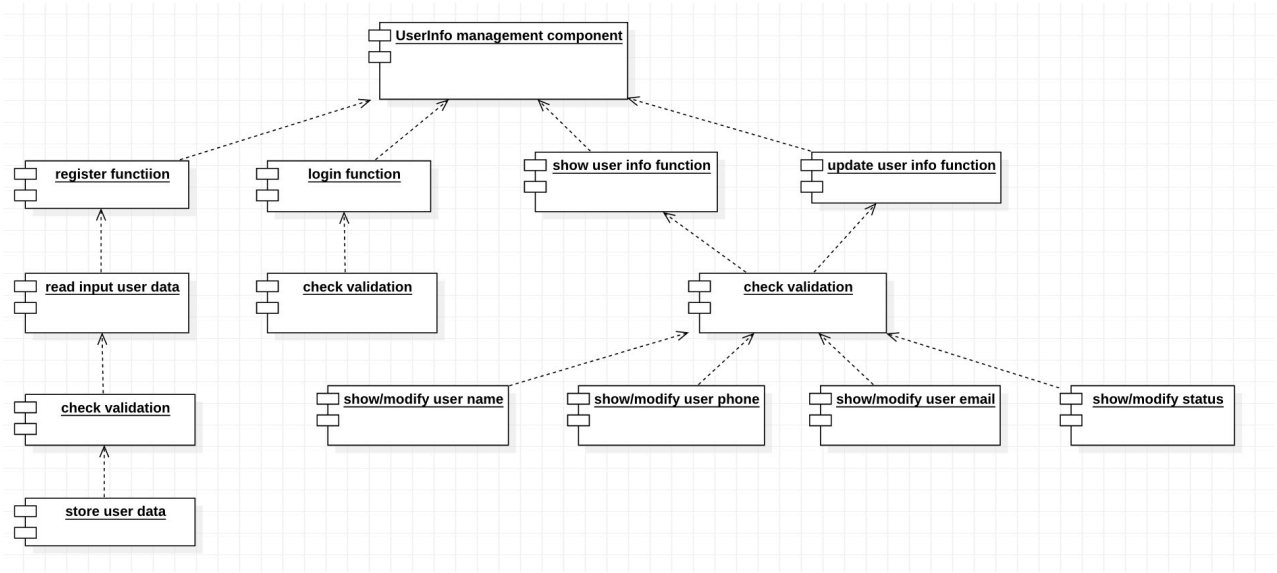


Figure 3.1: User Info Management Component Overview

3.3.2 User Posture Detection Component Overview

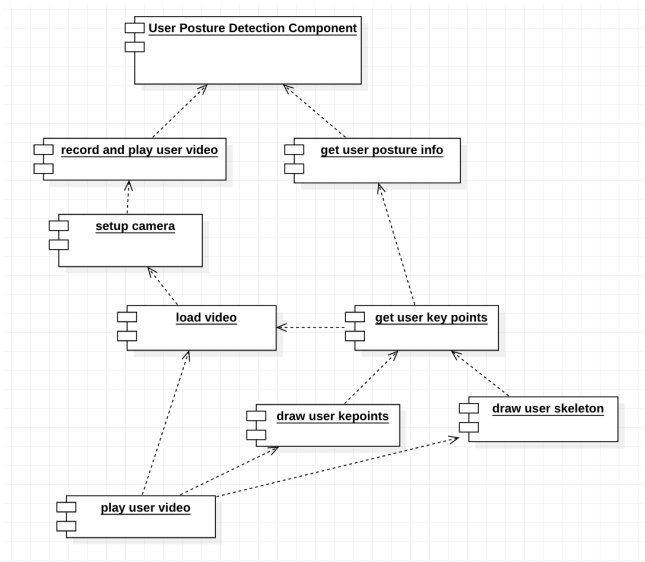


Figure 3.2: User Posture Detection Component Overview

3.3.3 Comparison Module Component Overview

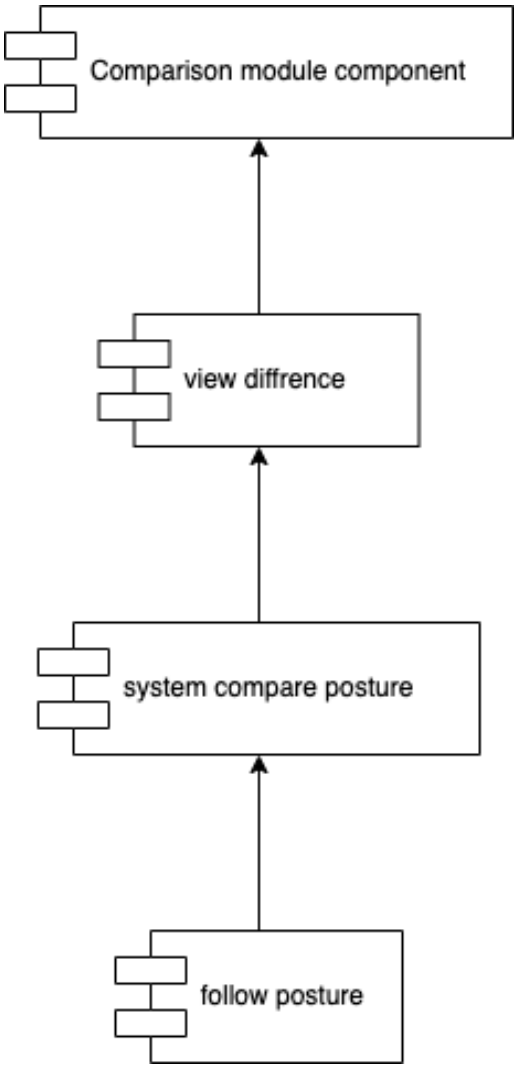


Figure 3.3: Comparison Module Component Overview

3.3.4 Correction Module Component Overview

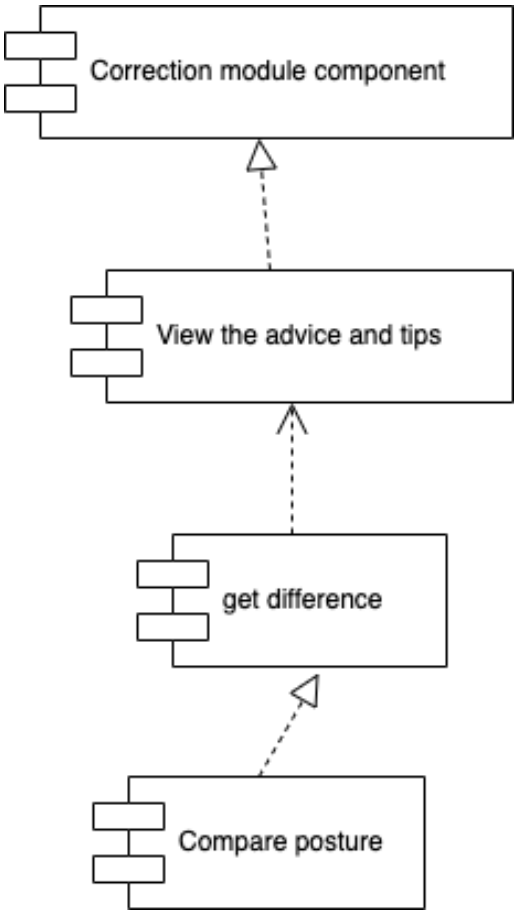


Figure 3.4: Correction Module Component Overview

3.3.5 Appraisal Module Component Overview

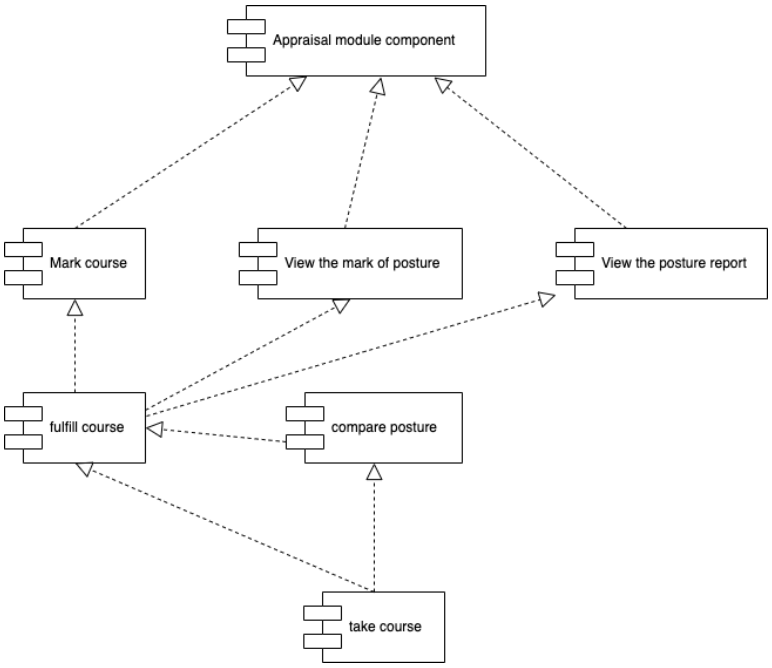


Figure 3.5: Appraisal Module Component Overview

3.3.6 Course Management Component Overview

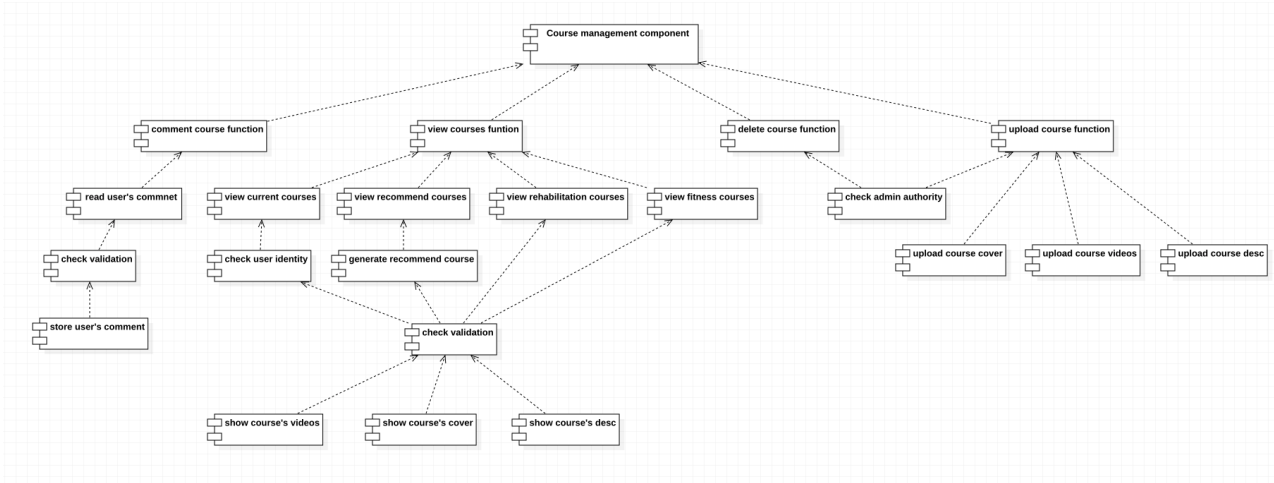


Figure 3.6: Course Management Component Overview

3.4 System Navigation Design

To combine these components, we design a navigation structure for iSport. And provide a overall navigation design for such a webApp as iSport.

When the user first enter iSport's host website, he/she has to register his/her information by UserInfoComponent, after registering, they can login to the introduction page.

At introduction page, users can view keep/rehabilitation courses through CoursesComponent. Through this component, they can view comments, add/delete their favorite course, view recommend courses too.

When course starts, user can observe their exercising video in real time with a skeleton drawn by DetectionComponent. Meanwhile, ComparisonComponent and CorrectionComponent will judge their performance and provide suggestion.

Every time after course, users can check their exercising report provided by AppraisalComponent.

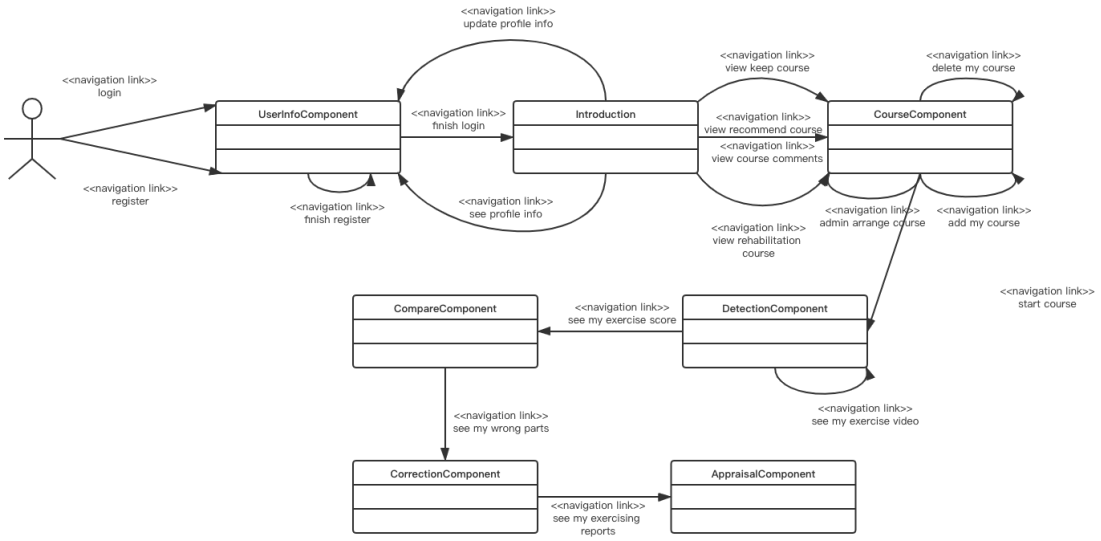


Figure 3.7: System Navigation Design

4 Principal Components

4.1 Component:User Info Management

4.1.1 Component Description

Users who use the platform for the first time need to register first. After successful registration, the system will jump to the login page. Users log in to the system through a valid user name and password, and then jump to the home page.

When users view their personal information on the personal page, they need to obtain the information saved in the database. Modifying personal information also needs to submit the modified information to the database of iSport.

All the above behaviors must interact with the database and blockchain to obtain and change the corresponding user personal information. The call relationship between classes is shown in the figure below:

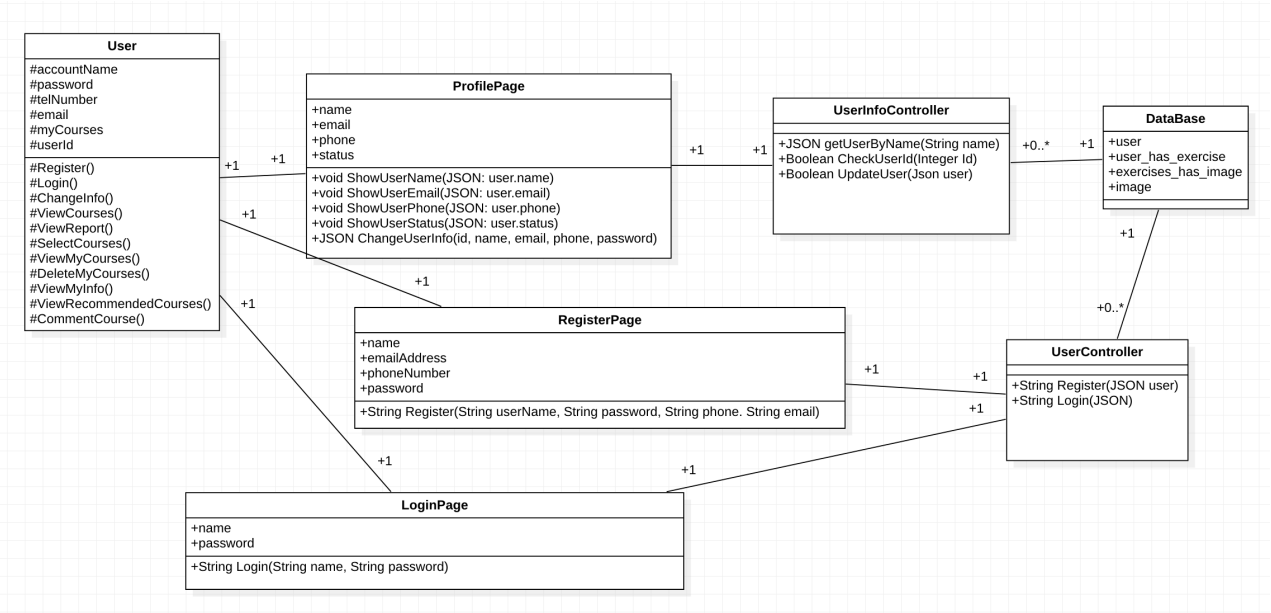


Figure 4.1: Class Diagram of UserInfo Component

4.1.2 Responsibilities

- User can register through UserInfo Component.
- User can login iSport by UserInfo Component.
- User can check his/her information through UserInfo Component on Profile Page.
- User can update his/her information through UserInfo Component on Profile Page.

4.1.3 Component Design Detail Description

Design for User Login in

- Input: User Name; User Password
- Output: Whether the user log in successfully
- Logical Flow:

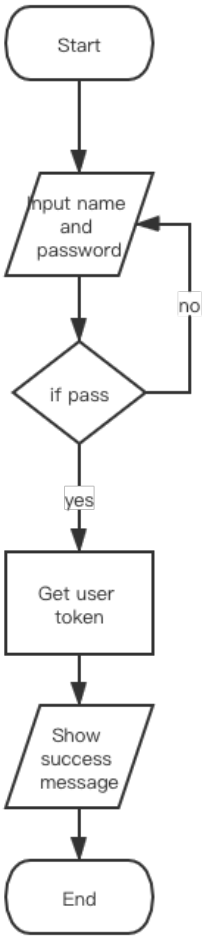


Figure 4.2: Logical Flow for User Log in

- Interface Design

Request Method	Post
Service Routing	http://[host]:6060/user/login
Params	Params1: user name; Params2: user password; Params3: request
Params Type	user name: String; user password:String; request: HttpServletRequest
Description	Check if the user's password is right according to the input user name and password.
Return Type	Json.

- Restriction and Error Handling

    1. User has registered before.

Design for User Register

- Input:

    1. User Name : Got by HttpRequest (compulsory)
    2. User Password: Got by HttpRequest (compulsory)
    3. Email Address: Got by HttpRequest (compulsory)
    4. Phone Number: Got by HttpRequest (compulsory)

- Output: Whether the user register successfully
- Logical Flow:

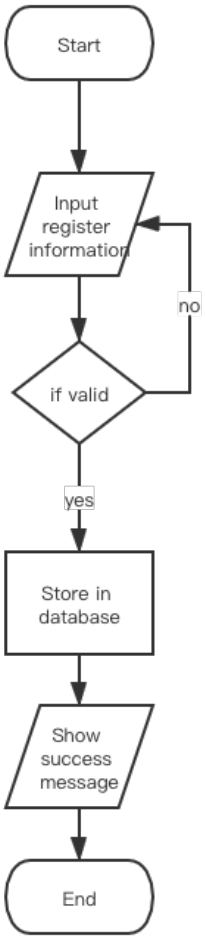


Figure 4.3: Logical Flow for User Register

- Interface Design

Request Method	Post
Service Routing	http://[host]:6060/user/register
Params	Params1: user name; Params2: user password; Params3: phone; Params4: email
Params Type	user name: String; user password:String; phone: String; email: String
Description	Help the user register iSport.
Return Type	Json.

- Restriction and Error Handling

    1. User has related information to register iSport.

Design for User Check Personal Information

- Input: null
- Output: User's profile page
- Logical Flow:

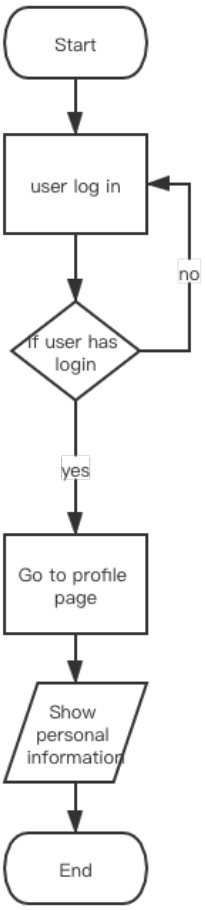


Figure 4.4: Logical Flow for User Checking Info

- Interface Design

Request Method	GET
Service Routing	http://[host]:6060/userName/info
Params	Params1: user name
Params Type	user name: String
Description	Get user's information and show on the profile page.
Return Type	Json.

- Restriction and Error Handling

1. User has personal information stored in database.

Design for User Update Personal Information

- Input:
    1. user name
    2. user password
    3. email
    4. phone
- Output: User's profile page
- Logical Flow:

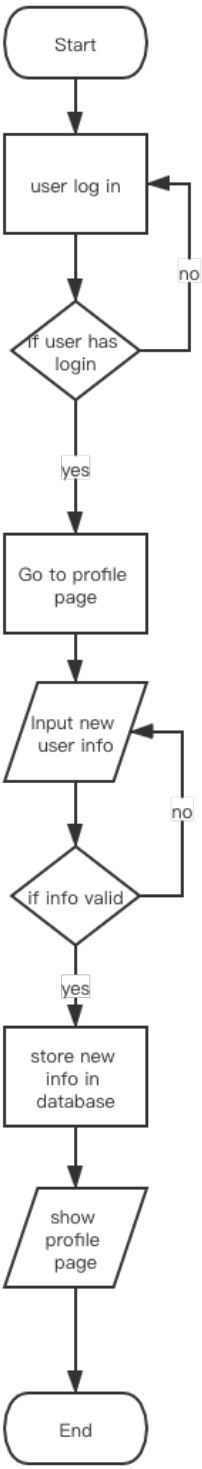


Figure 4.5: Logical Flow for User Updating Info

- Interface Design

Request Method	GET
Service Routing	http://[host]:6060/userId/updateInfo
Params	Params1: user id; Params2: user name; Params3: email; Params4: tel; Params5: password
Params Type	user id: Integer; user name: String; email: String; tel: String password: String
Description	Update user's information and refresh the profile page.
Return Type	Json.

- Restriction and Error Handling

  1. User has personal information stored in database.
  2. User provide valid new personal information.

4.2 Component:User Postures Detection

4.2.1 Component Description

When the user start to exercise, he/she have to stand before the camera and Detection component will open the camera and show the user's posture video in real time.

And then this component will get the user's key points positions by using PoseNet, and draw the user's skeleton according to theses key points.

The call relationship between classes is shown in the figure below:

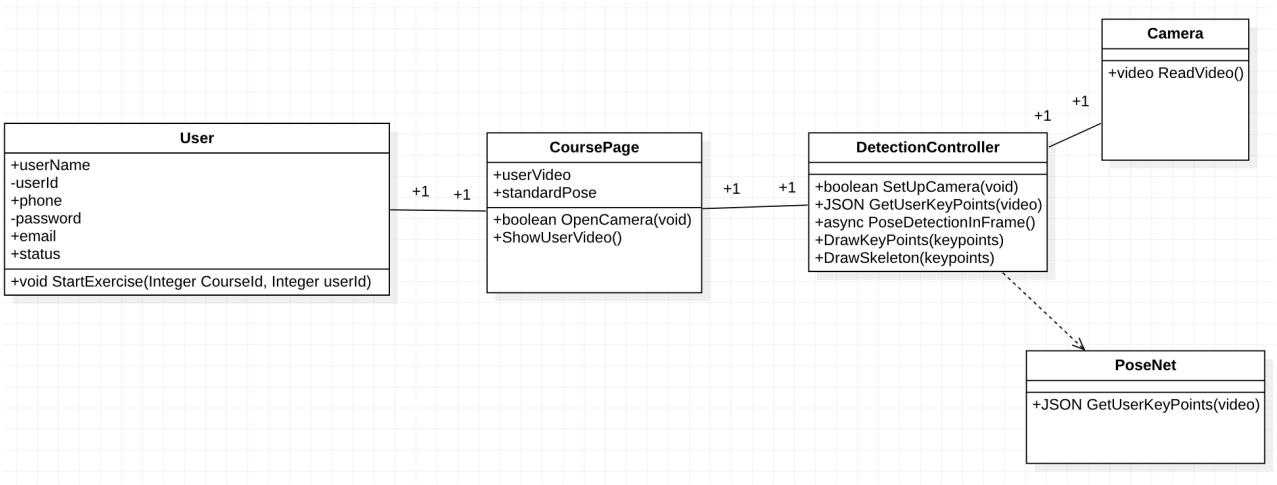


Figure 4.6: Class Diagram of Detection Component

4.2.2 Responsibilities

- Set up the camera, record and show user's video.
- Get user posture key points and draw the skeleton.

4.2.3 Component Design Detail Description

Design for Video, KeyPoints Reading and Playing

- Input: null
- Output: User's video
- Logical Flow:

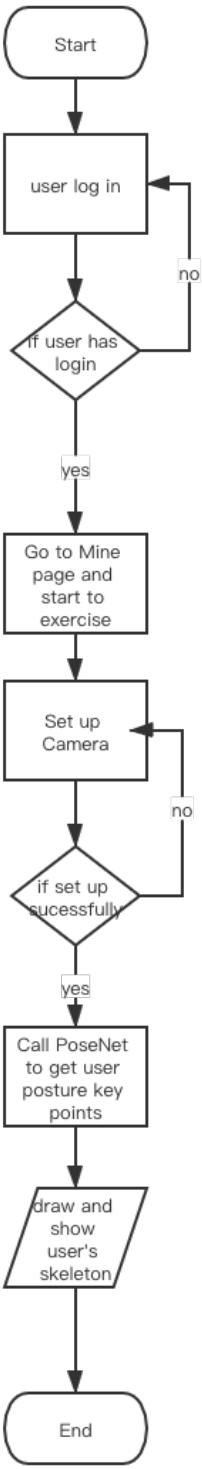


Figure 4.7: Logical Flow for Video Setup and KeyPoints Reading

- Function Design

setupCamera	Params: null; Return Type:async function; Function: Apply authority to user camera.
loadVideo	Params: null; Return Type:async function; Function: Apply authority to user camera.
detectionPoseInRealTime	Params: video; Return Type: async function; Function:play the user video
bindPage	Params: null Return Type: async function; Function:load the poseNet model
PoseDetectionFrame	Params: null Return Type: async function; Function:use the poseNet model to detect every frame's keypoints
drawKeyPoints	Params: keypoints Return Type: void; Function:to draw the keypoints which PosetNet read
drawSkeleton	Params: keypoints Return Type: void; Function:to draw the skeleton according to the keypoints

- Restriction and Error Handling

  1. User has logged in.
  2. The camera works well.

4.3 Component:Course Management

4.3.1 Component Description

This component is aiming at helping user to manage their courses, including viewing and choosing courses, add courses to user's courses and delete course from their own courses. System administrator also need this component to add new course to the database.

User is able to view all the fitness courses and rehabilitation courses in each course showing page, and the recommend course in recommend course page. When user is in these pages, they can click the "add course" button to add the course they are desired to learn to their user's courses. And they can view their own courses in user's course page. When user is in the user's course page, they can click "delete this course" button to delete the course from their learning courses.

When the system administrator is in the courses managing page, he can click the "add new course" button to add a new course to the database.

All of the behaviors need to interact with the user or course database to get, modify, add, delete the related data. The call relationship between above classes is shown in the figure below:

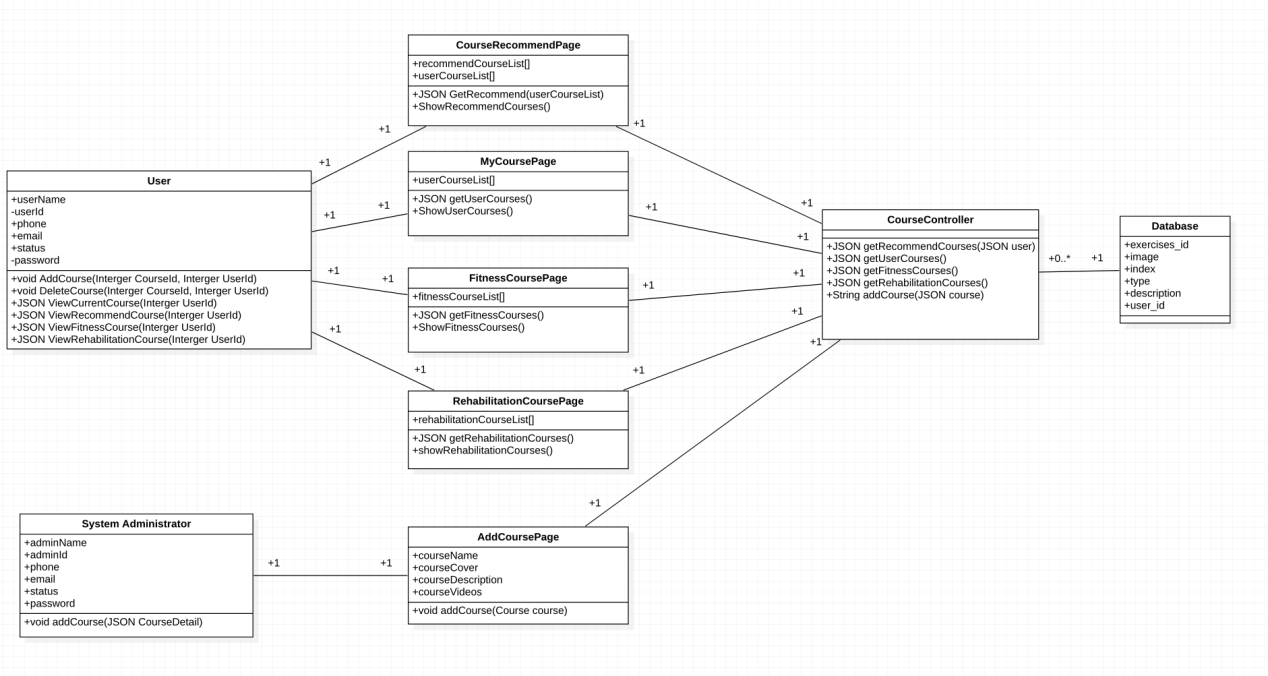


Figure 4.8: Class Diagram of course management Component

4.3.2 Responsibilities

- User can add a specific course to their course list through Course Management Component.
- User can delete a specific course from their course list through Course Management Component.
- User can view their course list through Course Management Component on user's courses page.
- User can view their recommend courses through Course Management Component on the recommend courses page.
- User can view all the fitness courses through Course Management Component on the fitness courses page.
- User can view all the rehabilitation courses through Course Management Component on the rehabilitation courses page.

- System Administrator can add a courses to the database through Course Management Component on the admin profile page.

4.3.3 Component Design Detail Description

Design for user add course

- Input: User ID, course ID;
- Output: Whether the user add the course successfully
- Logical Flow:

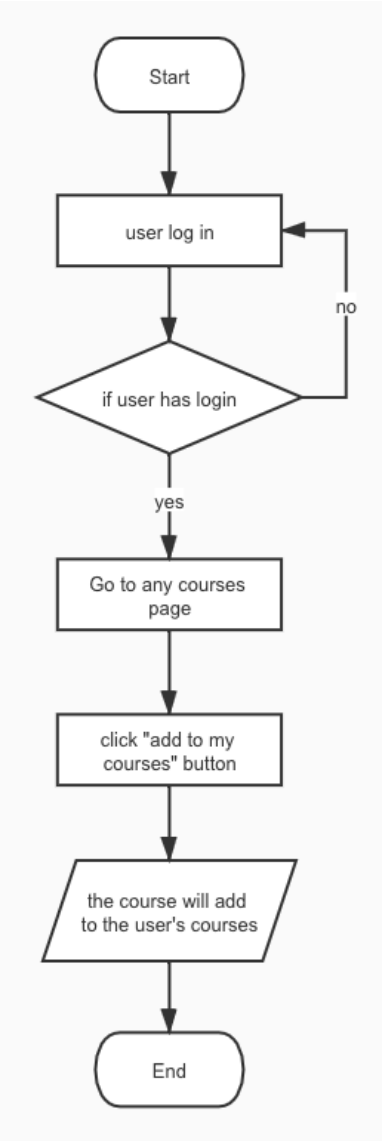


Figure 4.9: Logical Flow for User Add Course

- Interface Design

Request Method	Post
Service Routing	http://[host]:6060/addCourse
Params	Params1: user id; Params2: course id;
Params Type	user id: Integer; course id:Integer;
Description	Check if the user id and course id is exist, then add the course id to the user's course list
Return Type	String

- Restriction and Error Handling

    1. User id or course id is not in the database
    2. The course is already in user's course list

Design for user delete course

- Input: User ID, course ID;
- Output: Whether the user delete the course successfully
- Logical Flow:

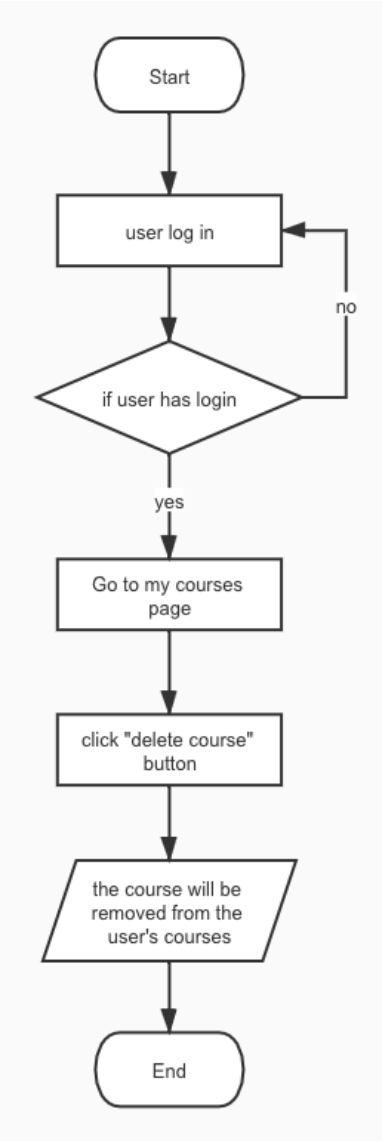


Figure 4.10: Logical Flow for User Remove Course

- Interface Design

Request Method	Post
Service Routing	http://[host]:6060/user/id/courses/courseId
Params	Params1: user id; Params2: course id;
Params Type	user id: Integer; course id:Integer;
Description	Check if the user id and course id is exist, then delete the course id from the user's course list
Return Type	String

- Restriction and Error Handling

    1. User id or course id is not in the database
    2. The course is not in user's course list

Design for user view user's course

- Input: User ID;
- Output: user's course list in JSON format
- Logical Flow:

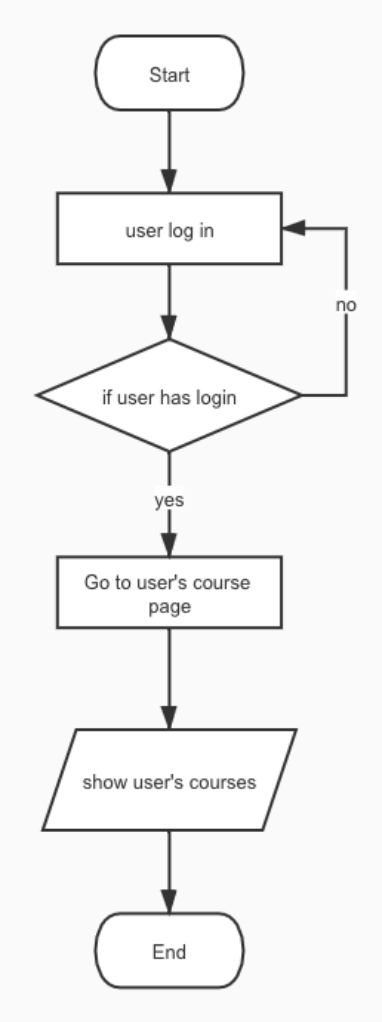


Figure 4.11: Logical Flow for User View User's Course

- Interface Design

Request Method	GET
Service Routing	http://[host]:6060/user/id/type/courses
Params	Params1: user id; Params2: course type;
Params Type	user id: Integer; course type: String;
Description	Check if the user id is exist, then show the user's course list sorted by type
Return Type	JSON

- Restriction and Error Handling

    1. User id is not in the database
    2. The user's course list is empty

Design for user view recommend courses

- Input: User ID;
- Output: user's recommend course list in JSON format
- Logical Flow:

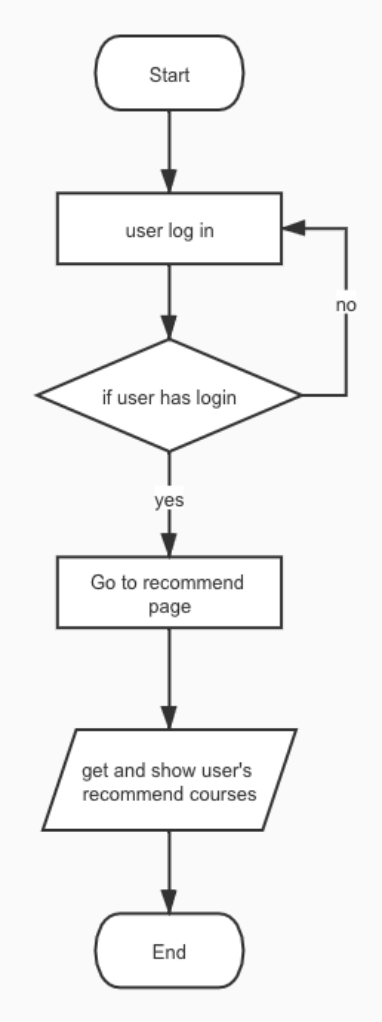


Figure 4.12: Logical Flow for User View Recommend Courses

- Interface Design

Request Method	GET
Service Routing	http://[host]:6060/user/id/recommend
Params	Params1: user id;
Params Type	user id: Integer;
Description	Check if the user id is exist, then show the user's recommend course list
Return Type	JSON

- Restriction and Error Handling

  1. User id is not in the database
  2. The user has chose all the courses

Design for user view fitness courses

- Input: None;
- Output: All the fitness course list in JSON format
- Logical Flow:

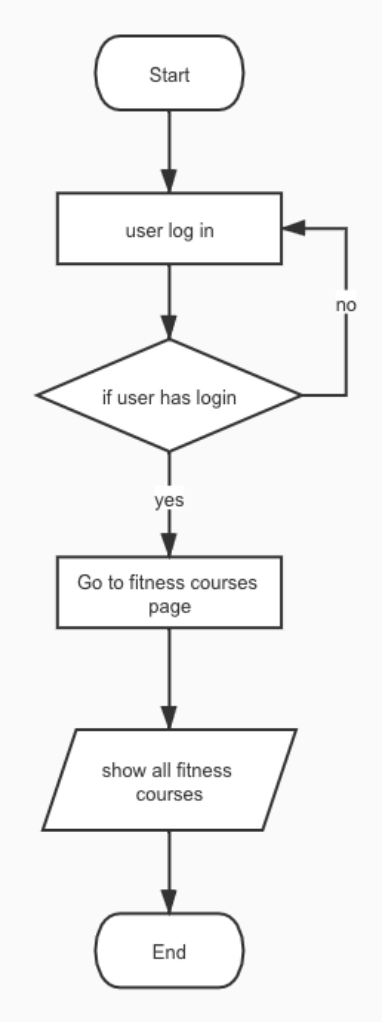


Figure 4.13: Logical Flow for User View Fitness Courses

- Interface Design

Request Method	Post
Service Routing	http://[host]:6060/courses
Params	Params1: course type("fitness");
Params Type	course type: String;
Description	Show the fitness course list
Return Type	JSON

- Restriction and Error Handling

1. No fitness courses in the database

Design for user view rehabilitation courses

- Input: None;
- Output: All the rehabilitation course list in JSON format
- Logical Flow:

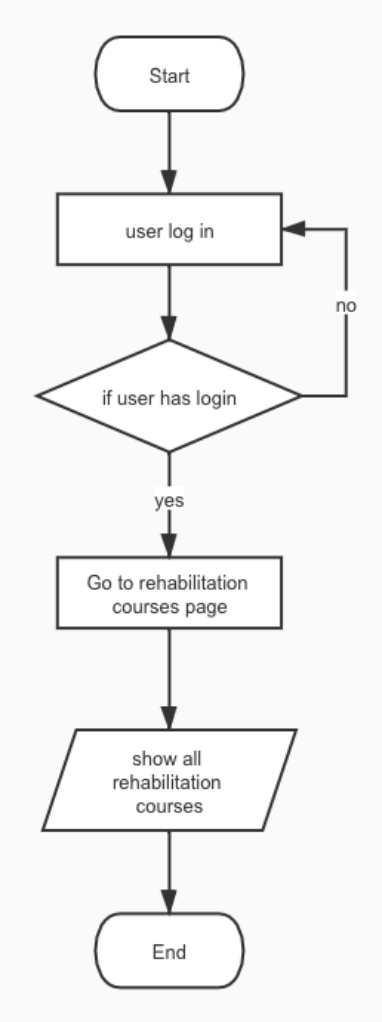


Figure 4.14: Logical Flow for User View Rehabilitation Courses

- Interface Design

Request Method	Post
Service Routing	http://[host]:6060/courses
Params	Params1: course type("rehabilitation");
Params Type	course type: String;
Description	Show the rehabilitation course list
Return Type	JSON

- Restriction and Error Handling

    1. No rehabilitation courses in the database

Design for administrator add new course

- Input: Course information in JSON format;
- Output: Whether the administrator add the course successfully
- Logical Flow:

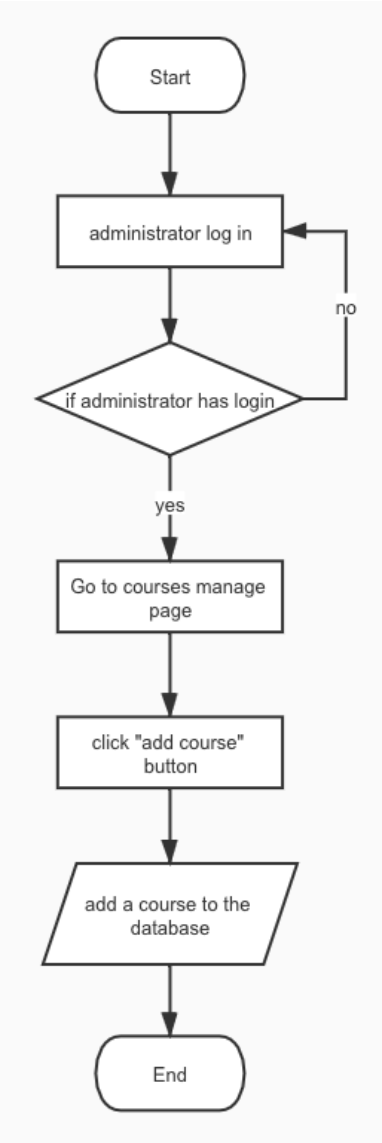


Figure 4.15: Logical Flow for Admin add Courses

- Interface Design

Request Method	Post
Service Routing	http://[host]:6060/uploadCourse
Params	Params1: course id; Params2: course name; Params3: course videos; Params4: course intro; Params5: course cover;
Params Type	course id: Integer; course name: String; course type: String; course videos: []Bytes; course intro: String; course cover: []Bytes;
Description	Add a course to the database
Return Type	String

- Restriction and Error Handling

  1. Attribution is in wrong format
  2. Course id is already in the database

4.4 Component: Posture Comparison Module

4.4.1 Component Description

The main part of this Web Service is our service correcting the posture of user. So the posture comparison is one of the key module.
When user take a course and start following the guide, the data collected by camera is uploaded and analyzed, then give the outcome to user.

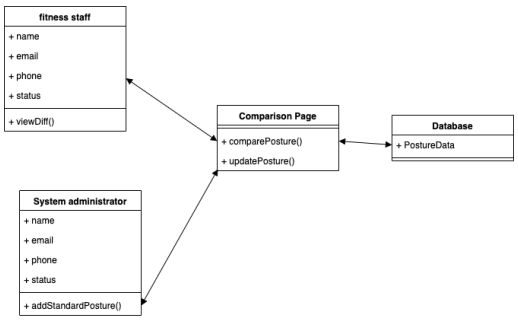


Figure 4.16: Class Diagram of Posture Comparison Component

4.4.2 Responsibilities

When users follow the postures, the video data will be uploaded to our servers, the backend compare the uploaded data with data in the database, and give the outcome to users.

4.4.3 Component Design Detail Description

Design for Comparison

- Input: user posture data
- Output: comparison outcome
- Logical Flow:

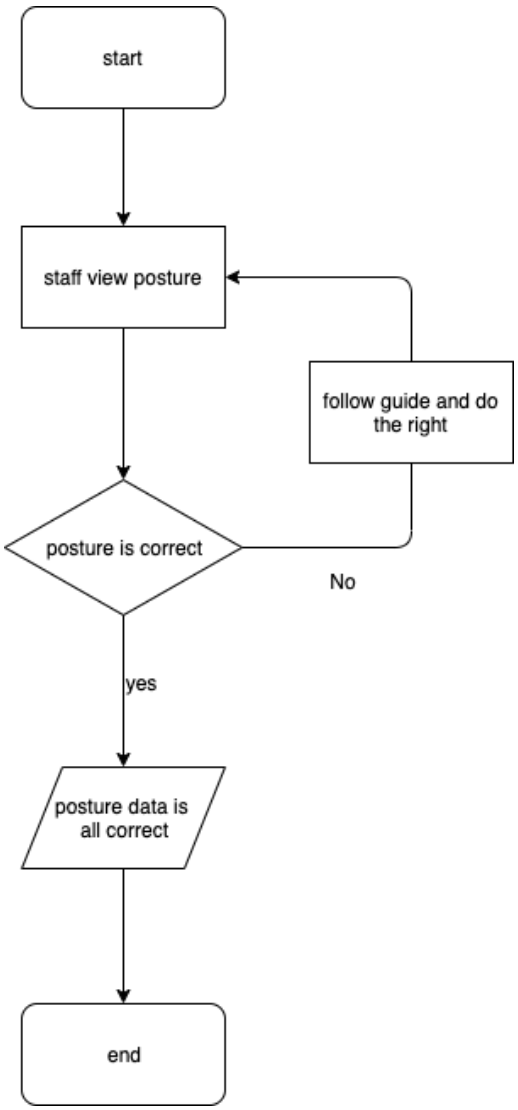


Figure 4.17: Logical Flow For Comparison

- Interface Design: null, this part is fulfill with javascript
- Restriction: user must login and take a course

4.5 Component: Correction Module

4.5.1 Component Description

When user get the outcome of comparison, he should follow the tips or advice in the browser, and correct his postures to match the right postures.

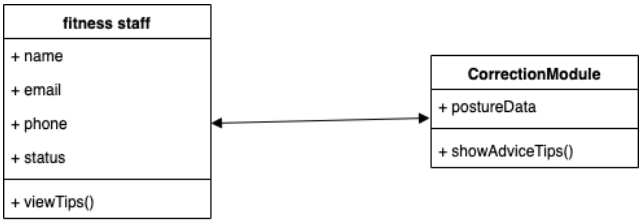


Figure 4.18: Class Diagram of Correction Component

4.5.2 Responsibilities

The Wrong posture will show as red point in the browser, and users can see right postures in the guide video. Users also get some advice or tips in the browser.
The Users should find the unmatched point or follow the guide and correct his posture until the comparison outcome shows there is no unmatched point.

4.5.3 Component Design Detail Description

Design for Correction

- Logical Flow:
- Output: advice or tips

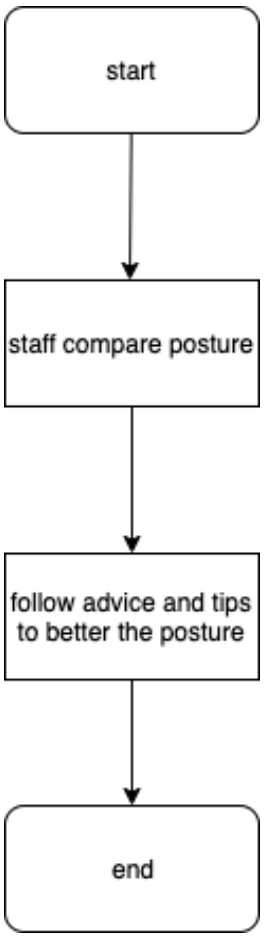


Figure 4.19: Logical Flow For Correction

- Interface Design: null, this part is fulfill with javascript
- Restriction: user must login and take a course

4.6 Component: Appraisal Module

4.6.1 Component Description

The Appraisal includes many aspects, users can appraise the course, the course can also appraise the performance of users.
The data of this module make it possible that platform and users benefit themselves.

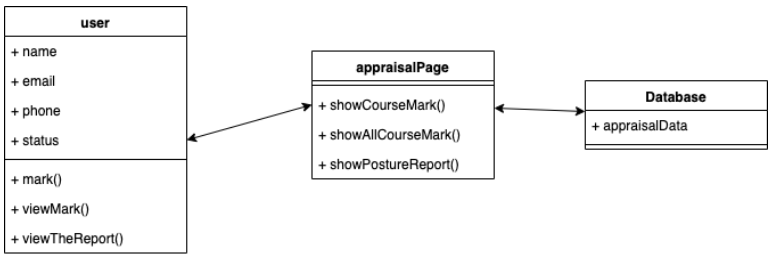


Figure 4.20: Class Diagram of Appraisal Component

4.6.2 Responsibilities

After user fulfill his course, he can appraise the course to inform other users whether this course is worthy.
In the other hand, system will appraise the usersperformance to inform user whether he did reaches the course's requirements.

4.6.3 Component Design Detail Description

Design for Commenting Course

- Input: the appraisal
- Output: appraisal outcome
- Logical Flow:

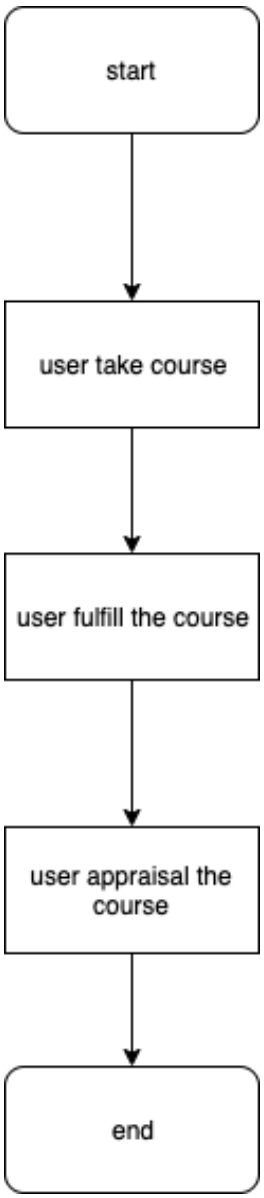


Figure 4.21: Logical Flow For Commenting Course

Design for Viewing Comment

- Input: the course id
- Output: the appraisal of this course
- Logical Flow:

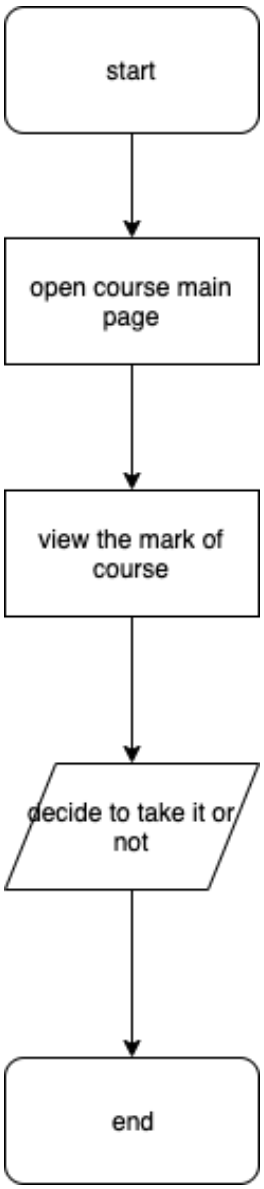


Figure 4.22: Logical Flow For Viewing Comment

- Interface Design:

Request Method	GET
Service Routing	api/course
Description	Get detail course data or comment.
Return Type	Json.

Design for Viewing Posture Report

- Input: the course id
- Output: the report of this course
- Logical Flow:

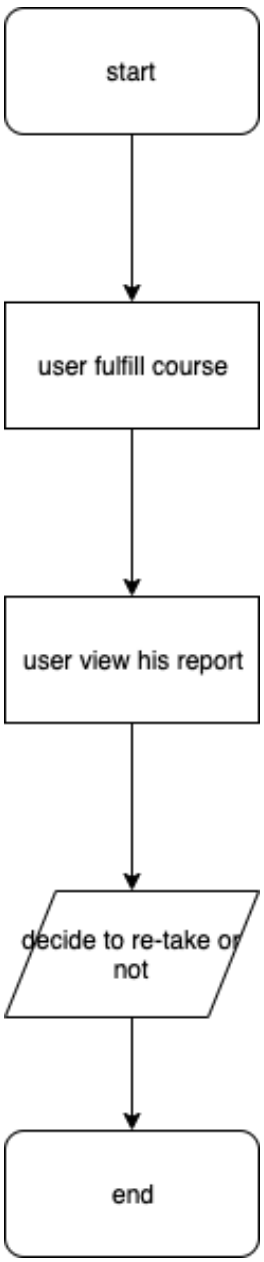


Figure 4.23: Logical Flow For Viewing Posture Report

5 Class Interfaces

5.1 Main Classes of User Info Management Component

5.1.1 Class UserController

Attribute Description

null

Methods Description

1. getUserName(userId): String
    a) Parameters: Integer userId
    b) Return Type: String
    c) Function: According to the input userId, check if the userId exists, if userId doesn't exist output "Id don't exist", otherwisereturn the name of the user with the corresponding id.
    d) Detail Description: To find the user name with corresponding userId in table user of the database.
    e) Implement:

```
public String getUserName(@PathVariable("id") int id){
    String name = userMapper.GetUserNameById(id);
    if(name != null) {
        return name;
    }
    else{
        return "Id don't exist";
    }
}
```

2. getUserInfo(userName): User
    a) Parameters: String userName
    b) Return Type: User
    c) Function: According to the input userName, check if the user-Name exists and return the corresponding user info.
    d) Detail Description: To find the user info(name, phone, email, status) with corresponding userName in table user of the database.
    e) Implement:

```
public User getUserInfo(@PathVariable("userName") String userName){
    User user = userMapper.getUserByName(userName);
    return user;
}
```

3. register
    a) Parameters: userName, password, phone, email
    b) Return Type: String
    c) Function: Register an account for user, and store the user info in database. But check if there is already a same user name first.
    d) Detail Description: To store the user info(name, phone, email, status) in database.

e) Implement:

```
public String register(@RequestParam("userName") String userName, @RequestParam("password") String password,
                       @RequestParam("phone") String phone, @RequestParam("email") String email) {
   if(userMapper.checkUserExisted(userName) < 1) {
        User user = new User();
        user.setName(userName);
        user.setPassword(password);
        user.setPhone(phone);
        user.setEmail(email);
        user.setStatus("user");
        userMapper.register(user);
        return "Register Successfully";
    }else{

        return "Username already existed";

    }
}
```

4. login

   a) Parameters: userName, password

   b) Return Type: String

   c) Function: According to the input userName and password check if the user can login iSport successfully.

   d) Detail Description: Find the record in table user with corresponding userName and password.

   e) Implement:

```
public String login(@RequestParam("userName") String userName, @RequestParam("password") String password, HttpServletRequest request) {
    //check user
    User user = userMapper.getUserByName(userName);
    if (user == null || user.getId() == null) {
        return "User doesn't exist!";
    }
    if (user.getPassword().equals(password)) {
        HttpSession session = request.getSession();
        if(session.getAttribute( s: "userId")!=null) {
            System.out.println("您已登录");
            return user.getId().toString();

        }
        else {
            session.setAttribute( s: "userId", user.getId());
            System.out.println("登录成功");
            return user.getId().toString();
        }
    } else {
        return "Password or Username incorrect!";
    }
}
```

5. updateUserInfo
    a) Parameters: userId, userName, email, tel, password
    b) Return Type: String
    c) Function: According to the input user info to update the user's information.
    d) Detail Description: Find the record in table user with corresponding userId and update the related info.
    e) Implement:

```
public String updateUserInfo(@PathVariable("id") Integer id, @RequestParam("name") String name, @RequestParam("email") String email,
                             @RequestParam("tel") String tel,@RequestParam("password") String password){

        userinfoMapper.UpdateUser(id,name,email,tel,password);
        return "update success";

}
```

5.1.2 Class User

Attribute Description

- id: Integer, every user has a unique id.
- password: String, password for every user.
- name: String, account name for the user.
- phone: String, phone number of the user.
- email: String, email of the user.
- status: String, status of the user. Check the if the user is an administration.

Methods Description

1. User(Integer id, String password, String name, String phone, String email, String status): User
    a) Parameters: Integer id, String password, String name, String phone, String email, String status
    b) Return Type: User
    c) Function: Construction function for class user
    d) Detail Description: Initialize the attributes of class user.
    e) Implement:

```
public User(Integer id, String password, String name, String phone, String email,String status) {
    this.id = id;
    this.password = password;
    this.name = name;
    this.phone = phone;
    this.email = email;
    this.status = status;
}
```

2. getId():Integer
    a) Parameters: null
    b) Return Type: Integer
    c) Function: Return the id of the user.
    d) Implement:

```
public Integer getId() { return id; }
```

3. setId(Integer id): void

a) Parameters: Integer id

b) Return Type: void

c) Function: Set the id of the user.

d) Implement:

```
public void setId(Integer id) { this.id = id; }
```

4. getName

a) Parameters: null

b) Return Type: String

c) Function: Return the account name of the user.

d) Implement:

```
public String getName() { return name; }
```

5. setName(String name):void

a) Parameters: String name

b) Return Type: void

c) Function: Set the account name of the user.

d) Implement:

```
public void setName(String name) { this.name = name == null ? null : name.trim(); }
```

5.2 Main Classes of Course Management Component

5.2.1 ExercisesController

Attribute Description

null

Methods Description

1. getAllCourses(type, [id]): CourseList
   a) Parameters: String type, Integer id(Optional)
   b) Return Type: CourseList[]
   c) Function: Query the type or id given by the parameters from all courses, if the result is null, return "The course list is blank" , otherwise, return the result as a course list.
   d) Detail Description: To find the courses with corresponding course type or id in table course of the database.
   e) Implement:

```
public ResultModel2 getAllCourses(@RequestParam("type") String type, @RequestParam("id") int id){
    List<Map<String, Object>> result = imageMapper.GetAllCourses(type);
    System.out.println(type);
    if(type.equals("all")){
        result= imageMapper.GetAllCoursesNew();
    }
    for(int i = 0;i < result.size();i++){
        System.out.println((result.get(i)).get("id") instanceof Integer);
        System.out.println((result.get(i)).get("id"));
        Integer exId = Integer.parseInt((result.get(i)).get("id").toString());
        if(imageMapper.IsUserWithCourses(id,exId)!= 0){
            result.get(i).put("Joined",1);
        }
        else{
            result.get(i).put("Joined",0);
        }
    }
    ResultModel2 resultModel2 = new ResultModel2(result);
    return resultModel2;
}
```

2. getRecommendation(id): CourseList
    a) Parameters: Integer userId
    b) Return Type: CourseList[]
    c) Function: According to the user's courses, generate and return recommend courses for the user to learn.
    d) Detail Description: Find the user with corresponding userName in table user of the database, then find the courses that the user has chosen before, generate the recommend courses according by these courses and return the result.
    e) Implement:

```
public ResultModel2 getRecommendation(@PathVariable("id") int id){
    List<Map<String,Object> >result = imageMapper.GetRecommendation(id);

    ResultModel2 resultModel2 = new ResultModel2(result);
    return resultModel2;
}
```

3. getCourseFirstImage(id): Course
    a) Parameters: Integer courseId
    b) Return Type: Course
    c) Function: According to the course id, find the course and the correspond image, return the course with a cover.
    d) Detail Description: Find the course through the course id in course database and course image in image database, set the image as the course's cover and return the course with a cover.
    e) Implement:

```
public Exercises getCourseFirstImage(@PathVariable("exercisesId") int exercisesId){
    ArrayList<Image> result =  imageMapper.selectImagesByEx(exercisesId);
    Exercises exercises = exercisesMapper.selectByPrimaryKey(exercisesId);
    if (exercises != null) {
        exercises.setImages(result);
    }
    return exercises;
}
```

4. addCourse
    a) Parameters: Integer userId, Integer courseId
    b) Return Type: String
    c) Function: According to the user id and course id, add the course to user's course list.
    d) Detail Description: Find the user in table user with corresponding userId and the course in table course with corresponding course id, if the user has not chosen the course before, add the course id to the user's courses list and return OK", otherwise, return "This course already added".
    e) Implement:

```
public String addCourse(@RequestParam("id") Integer userId, @RequestParam("exId") Integer exId) {
    UserHasExercises userHasExercises = new UserHasExercises(userId,exId);
    UserHasExercises temp = userHasExercisesMapper.selectOne(userHasExercises);
    if (temp != null && temp.getExercisesId() == userHasExercises.getExercisesId()) {
        return "This course already added";
    } else {
        userHasExercisesMapper.insert(userHasExercises);
        return "OK";

    }
}
```

5. updateComments

a) Parameters: exerciseId, commentUser, comment

b) Return Type: Integer exerciseId, String commentUser, String comment

c) Function: According to the input exerciseId, commentUser and comment, update the comment of the course.

d) Detail Description: Find the course in table user with corresponding courseId and update the comment information by the given comment.

e) Implement:

```
public String updateComments(@RequestParam("exId") Integer exId,@RequestParam("commentUser") String commentUser, @RequestParam("comment") String comment) {
    imageMapper.updateComment(exId,commentUser,comment);
    return "OK";
}
```

5.2.2 Class Exercises

Attribute Description

- id: Integer, every exercise has a unique id.
- description: String, the description of the exercise
- title: String, the title of the exercise.
- type: String, the type of the exercise.
- image: Bytes, the cover of the user.

Methods Description

1. Exercises(Integer id, String description, String title, String type): Exercise

    a) Parameters: Integer id, String description, String title, String type

b) Return Type: Exercise

c) Function: Construction function for class exercise

d) Detail Description: Initialize the attributes of class exercise.

e) Implement:

```
public Exercises(Integer id, String description, String title, String type) {
    this.id = id;
    this.description = description;
    this.title = title;
    this.type = type;
}
```

2. getId():Integer

   a) Parameters: null

   b) Return Type: Integer

   c) Function: Return the id of the exercise.

   d) Implement:

```
public Integer getId() { return id; }
```

3. setId(Integer id): void

   a) Parameters: Integer id

   b) Return Type: void

   c) Function: Set the id of the exercise.

d) Implement:

```
public void setId(Integer id) { this.id = id; }
```

4. getDescription
    a) Parameters: null
    b) Return Type: String
    c) Function: Return the description of the exercise.
    d) Implement:

```
public String getDescription() { return description; }
```

5. setName(String name):void
    a) Parameters: String description
    b) Return Type: void
    c) Function: Set the description of the exercise.
    d) Implement:

```
public void setDescription(String description) {
    this.description = description == null ? null : description.trim();
}
```

6. getType():String

a) Parameters: null
b) Return Type: String
c) Function: Get the type of the exercise
d) Implement:

```
public String getType() { return type; }
```

7. setType(String type): void
    a) Parameters: String type
    b) Return Type: void
    c) Function: Set the type of the exercise
    d) Implement:

```
public void setType(String type) { this.type = type == null ? null : type.trim(); }
```

5.2.3 Class Comparison

Attribute Description

Method Description

a) formatKeypoints(keypoints): Array
    i. Parameters: keypoints
    ii. Return: newKeys
    iii. Function: sort keypoints by name

iv. Detail Description: format keypoints, and sort keypoints by name

v. Implement:

```
/**
 * 格式化关键点，按照关键点名称排序
 */
function formatKeypoints(keypoints) {
  let newKeys = [];
  for (let i = 0; i < keypoints.length; i++) {
    newKeys.push(keypoints[i]);
  }
  newKeys.sort((a, b) => {
    let x = a.part.toLowerCase(),
      y = b.part.toLowerCase();
    return x < y ? -1 : x > y ? 1 : 0;
  });

  return newKeys;
}
```

b) PoseVector(keypoints, boudingBoxpoints):PoseVector

i. Parameters: keypoints, boudingBoxpoints

ii. Return: PoseVector

iii. Function: the vector with the same proportion regulate the final comparison

iv. Detail Description: zoom pose vector to the same proportion

v. Implement:

```
/**
 * 姿势向量放缩至统一比例
 */
function PoseVector(keypoints, boudingBoxpoints) {
  let boxWidth = boudingBoxpoints[1].x - boudingBoxpoints[0].x;
  let boxHeight = boudingBoxpoi any .y - boudingBoxpoints[0].y;
  let x0 = boudingBoxpoints[0].x;
  let y0 = boudingBoxpoints[0].y;
  let vectorTemp = [];
  for (let i = 0; i < keypoints.length; i++) {
    const {
      x,
      y
    } = keypoints[i].position;

    let x1 = ((x - x0) * 400) / boxWidth;
    vectorTemp.push(x1);
    let y1 = ((y - y0) * 400) / boxHeight;
    vectorTemp.push(y1);
  }

  let PoseVector = l2_normalize(vectorTemp);
  let sum = 0;
  for (let j = 0; j < keypoints.length; j++) {
    sum += keypoints[j].score;
    PoseVector.push(keypoints[j].score);
  }
  PoseVector.push(sum);
  return PoseVector;
}
```

```
/**
 * 归一化
 */
function l2_normalize(vectorList, dim) {
  for (let i = 0; i < vectorList.length; i++) {
    vectorList[i] = vectorList[i] * vectorList[i];
  }
  const data = tf.tensor1d(vectorList);
  const sum = tf.variable(tf.sum(data).toFloat()).get();
  // sum.print();
  for (let i = 0; i < vectorList.length; i++) {
    vectorList[i] = tf.variable(tf.sqrt(vectorList[i] / sum)).get();
  }
  return vectorList;
}
```

c) weightedDistanceMatching(poseVector1, poseVector2): Number

i. Parameters: poseVector1, poseVector2

ii. Return: summation1 * summation2

iii. Function: get similarity distance

iv. Detail Description: calculate similarity distance with weight

v. Implement:

```
/**
 * 加权距离匹配
 */
function weightedDistanceMatching(poseVector1, poseVector2) {
  let vector1PoseXY = poseVector1.slice(0, 34),
    vector2PoseXY = poseVector2.slice(0, 34),
    vector1Confidences = poseVector1.slice(34, 51),
    vector1ConfidenceSum = poseVector1.slice(51, 52);

  let summation1 = 1 / vector1ConfidenceSum,
    summation2 = 0;
  for (let i = 0; i < vector1PoseXY.length; i++) {
    let tempConf = Math.floor(i / 2),
      tempSum = vector1Confidences[tempConf] * Math.abs(vector1PoseXY[i] - vector2PoseXY[i]);
    summation2 = summation2 + tempSum;
  }

  return summation1 * summation2;
}
```

d) showResult(poseVector, imageVector, isVideo)

i. Parameters: poseVector, imageVector, isVideo

ii. Return:

iii. Function: show comparison result

iv. Detail Description: if accurate, show green, if relatively accurate, show blue, if not accurate, show red

v. Implement:

```
/**
 * 展示匹配结果（正确时为绿色）
 */
const timeout = 12,
  timeoutlong = 24;
let count = 1,
  countlong = 1;

function showResult(poseVector, imageVector, isVideo) {
  let testSimilar = weightedDistanceMatching(poseVector, imageVector),
    standard = 0.036;

  marks.push(0.3 - testSimilar);
  // console.log("result: " + testSimilar + " / " + standard);

  if (testSimilar <= standard) {
    notice = "Your action is perfect!";
    $("#judge_and_paint")[0].className = "card-body alert-success";
    $("#noticeBox")[0].className = "card-footer alert-success lh-4 align-middle";
```

5.2.4 Class: Correction

Attribute Description

Method Description

a) OnePartCorrection(poseVector1, poseVector2, standard): Bool

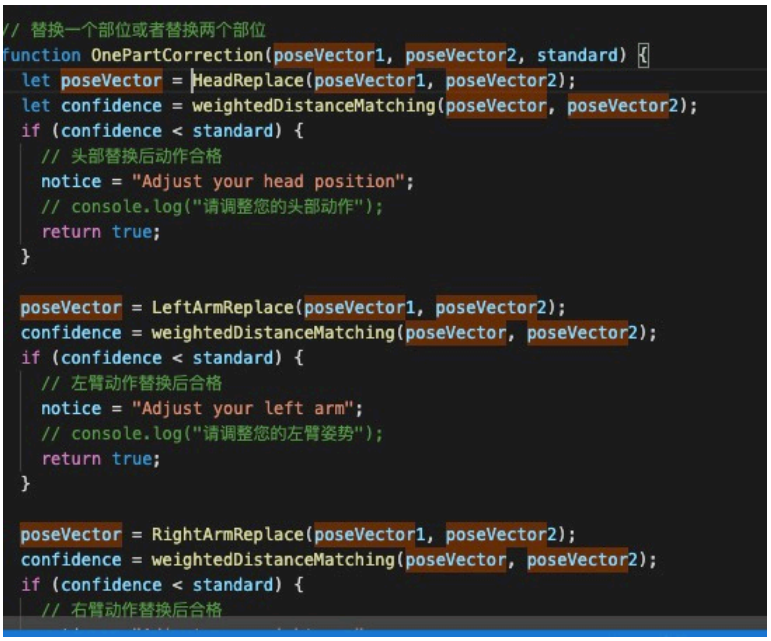
i. Parameters: poseVector1, poseVector2 standard

ii. Return Bool

iii. Function: check wrong one posture

iv. Detail Description: compare the posture and export comparison outcome

v. Implement:



```
// 替换一个部位或者替换两个部位
function OnePartCorrection(poseVector1, poseVector2, standard) {
  let poseVector = HeadReplace(poseVector1, poseVector2);
  let confidence = weightedDistanceMatching(poseVector, poseVector2);
  if (confidence < standard) {
    // 头部替换后动作合格
    notice = "Adjust your head position";
    // console.log("请调整您的头部动作");
    return true;
  }

  poseVector = LeftArmReplace(poseVector1, poseVector2);
  confidence = weightedDistanceMatching(poseVector, poseVector2);
  if (confidence < standard) {
    // 左臂动作替换后合格
    notice = "Adjust your left arm";
    // console.log("请调整您的左臂姿势");
    return true;
  }

  poseVector = RightArmReplace(poseVector1, poseVector2);
  confidence = weightedDistanceMatching(poseVector, poseVector2);
  if (confidence < standard) {
    // 右臂动作替换后合格
```

b) TwoPartCheck(poseVector1, poseVector2, standard): Bool

i. Parameters: poseVector1, poseVector2 standard

ii. Return Bool

iii. Function: check wrong posture more than one

iv. Detail Description: compare the posture and export comparison outcome

v. Implement:

```
function TwoPartCheck(poseVector1, poseVector2, standard) {
  // 对部分部位进行两两检查：左臂和右臂，左腿和右腿，左臂和左腿，右臂和右腿
  let poseVector = LeftArmReplace(poseVector1, poseVector2);
  poseVector = RightArmReplace(poseVector, poseVector2);
  let confidence = weightedDistanceMatching(poseVector, poseVector2);
  if (confidence < standard) {
    notice = "Adjust your arms!";
    // console.log("请注意您的左右臂是否协调");
    return true;
  }

  poseVector = LeftLegReplace(poseVector1, poseVector2);
  poseVector = RightLegReplace(poseVector, poseVector2);
  confidence = weightedDistanceMatching(poseVector, poseVector2);
  if (confidence < standard) {
    notice = "Notice your legs";
    // console.log("请注意您的左右腿是否协调");
    return true;
  }

  poseVector = LeftArmReplace(poseVector1, poseVector2);
  poseVector = LeftLegReplace(poseVector, poseVector2);
  confidence = weightedDistanceMatching(poseVector, poseVector2);
  if (confidence < standard) {
    notice = "Watch out your left body";
```

5.2.5 Class: Appraisal

Attribute Description

Method Description

a) getAllCourses(@RequestParam("type") String type, @RequestParam("id") int id): ResultModel2

i. Parameters: type, id

ii. Return ResultModel2

iii. Function: get course and appraisal

iv. Detail Description: get course and appraisal data from backend

v. Implement:

```
    @RequestMapping(value = "/updateComments")
    @ResponseBody
    public String updateComments(@RequestParam("exId") Integer exId,@RequestParam("commentUser") String commentUser, @RequestParam("comment") String comment) {
        imageMapper.updateComment(exId,commentUser,comment);
        return "OK";
    }
}
```

```
    @RequestMapping(value = "/courses", method = RequestMethod.GET)
    @ResponseBody
    public ResultModel2 getAllCourses(@RequestParam("type") String type, @RequestParam("id") int id){
        List<Map<String, Object>> result = imageMapper.GetAllCourses(type);
        System.out.println(type);
        if(type.equals("all")){
            result= imageMapper.GetAllCoursesNew();
        }
        for(int i = 0;i < result.size();i++){
            System.out.println((result.get(i)).get("id") instanceof Integer);
            System.out.println((result.get(i)).get("id"));
            Integer exId = Integer.parseInt((result.get(i)).get("id").toString());
            if(imageMapper.IsUserWithCourses(id,exId)!= 0){
                result.get(i).put("Joined",1);
            }
            else{
                result.get(i).put("Joined",0);
            }
        }
        ResultModel2 resultModel2 = new ResultModel2(result);
        return resultModel2;
    }
```

6 Human Interface Design

6.1 Mockup of User Interface

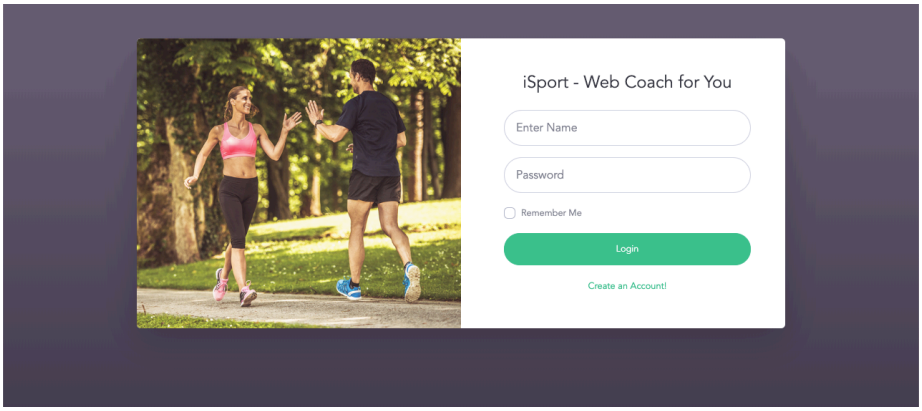


Figure 6.1: Login and Register Page

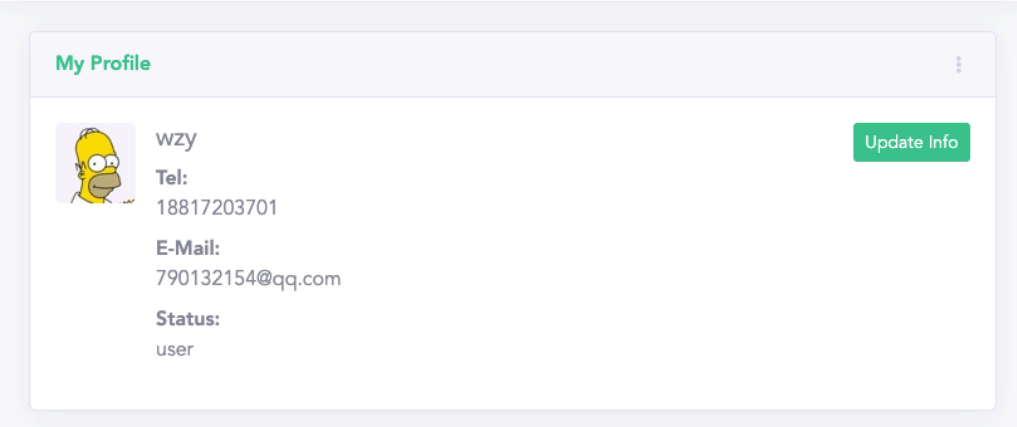


Figure 6.2: Profile Page

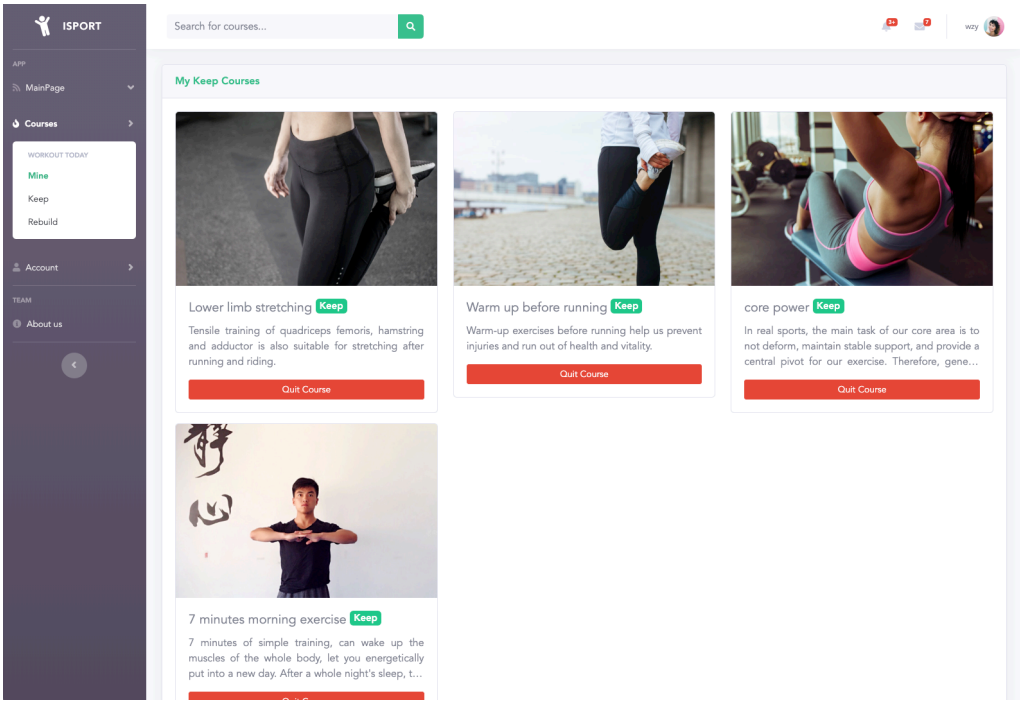


Figure 6.3: My Courses Page

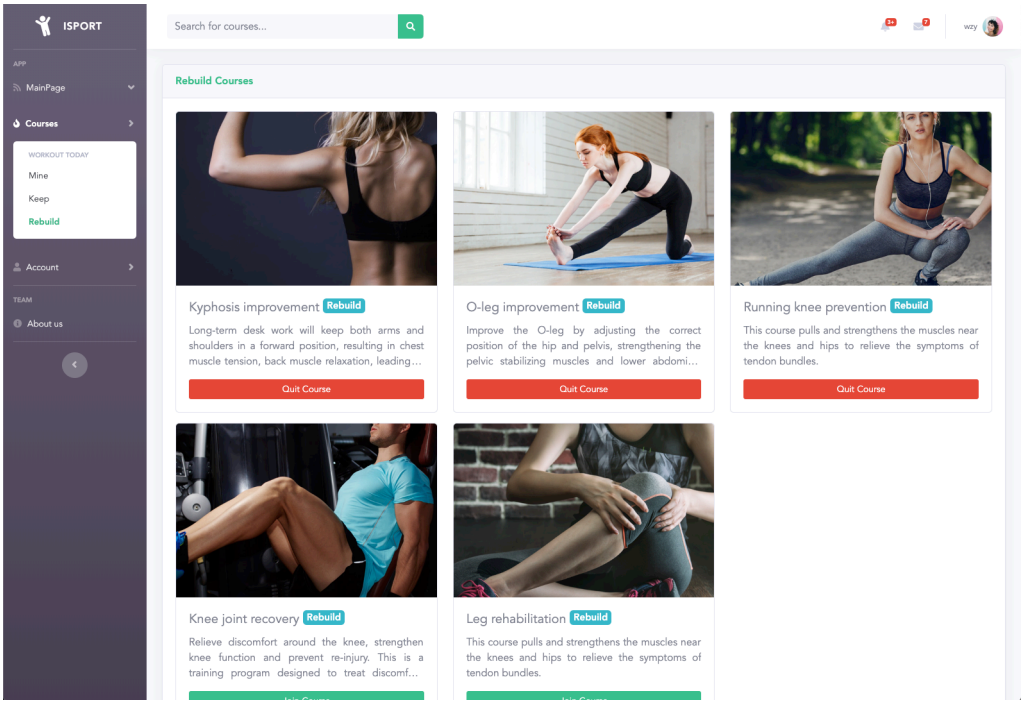


Figure 6.4: Recovering Courses Page

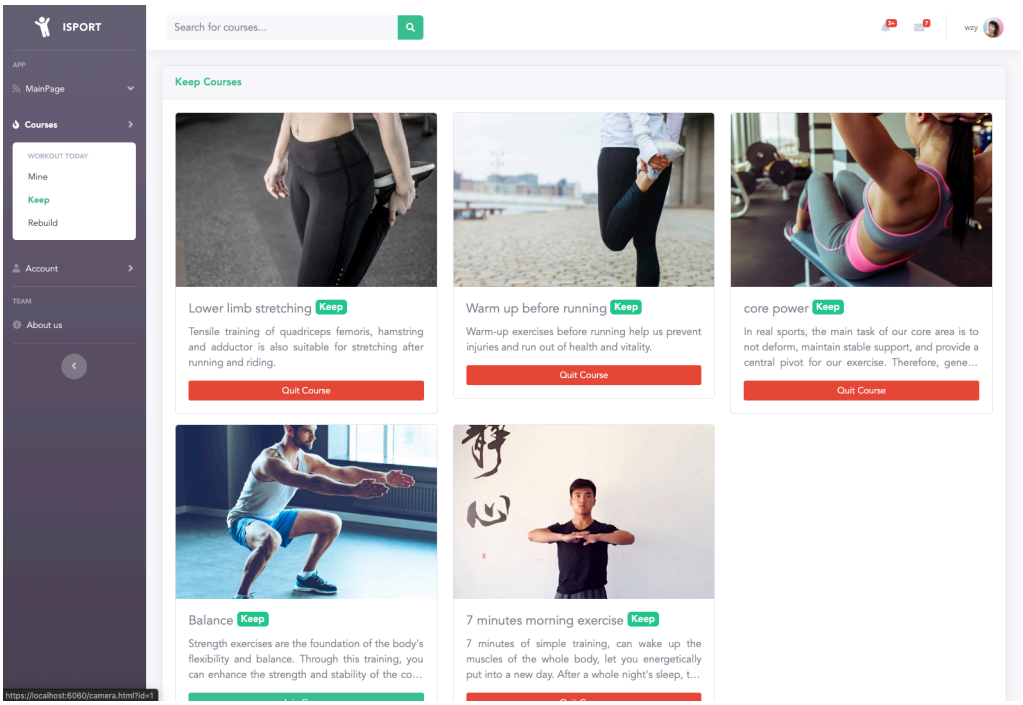


Figure 6.5: Exercise Courses Page

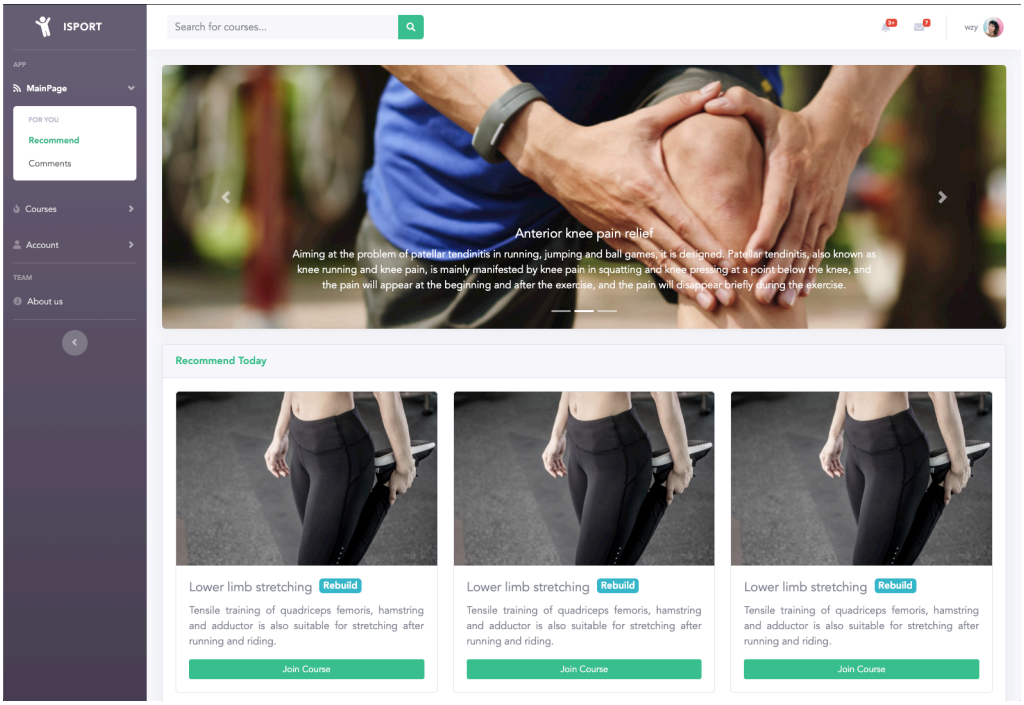


Figure 6.6: Recommend Page

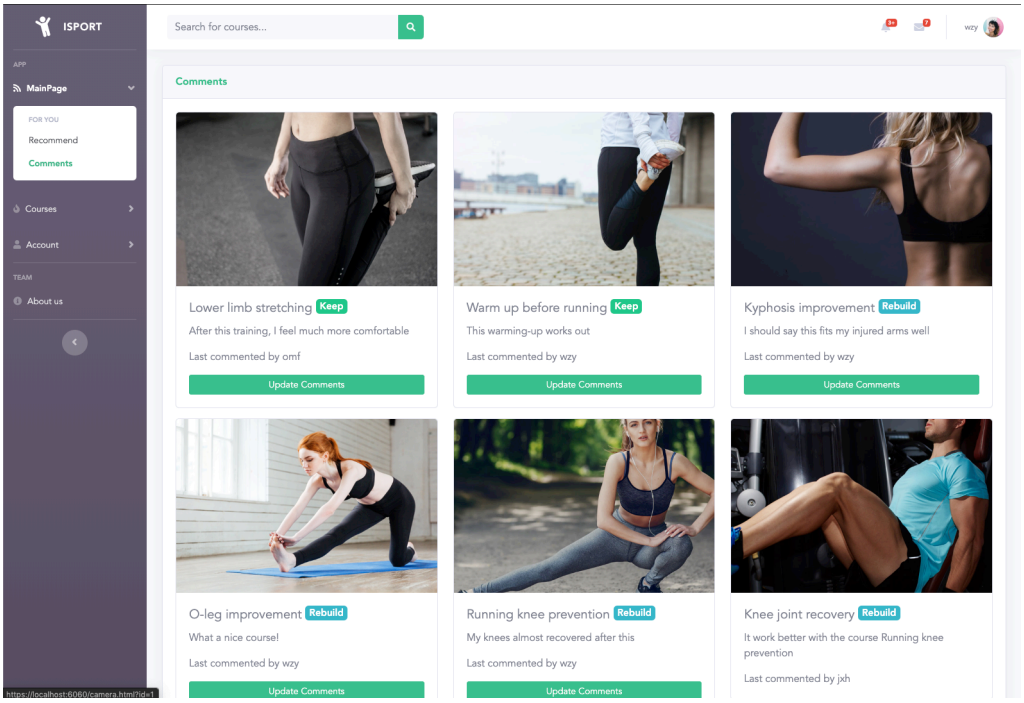


Figure 6.7: Comments Page

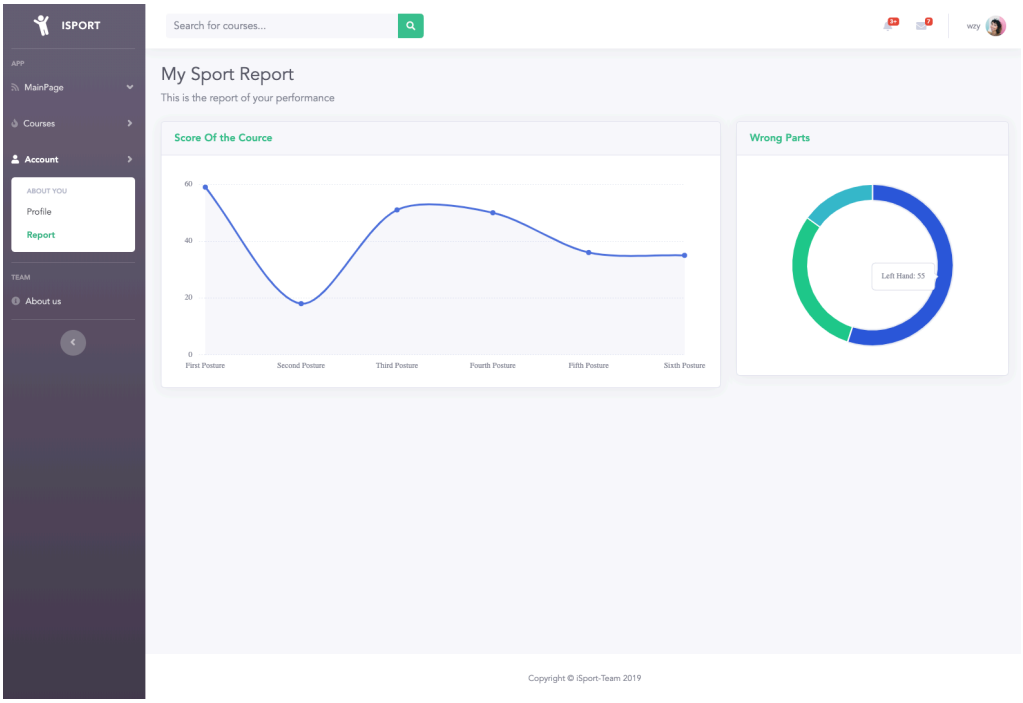


Figure 6.8: Reports Page

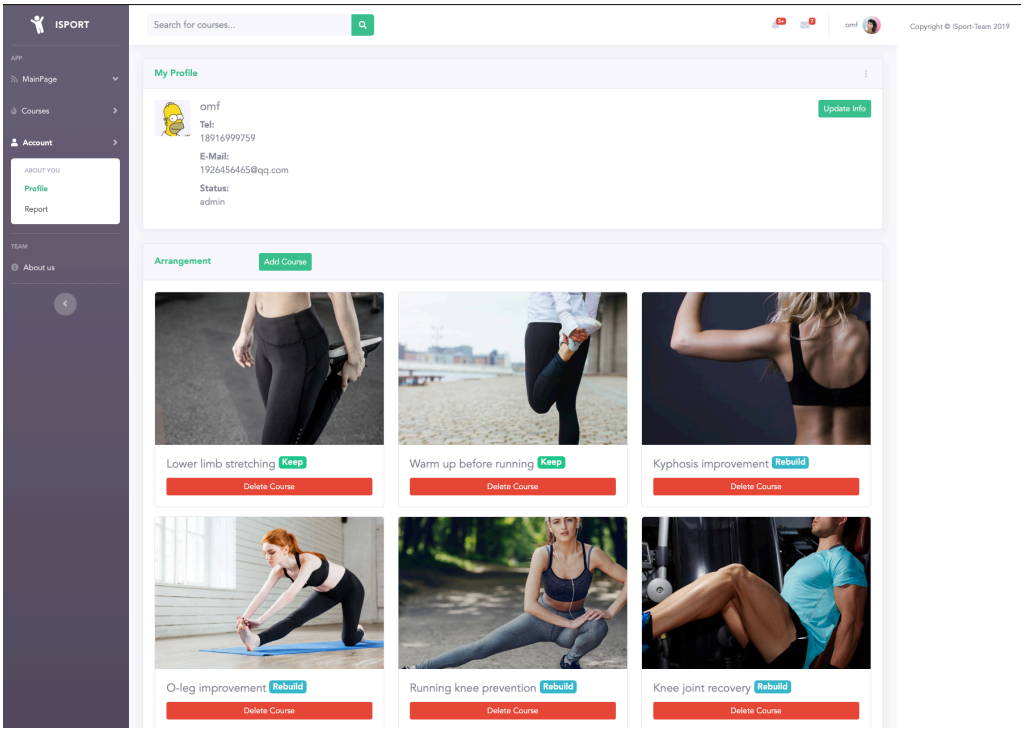


Figure 6.9: Arrangement Page

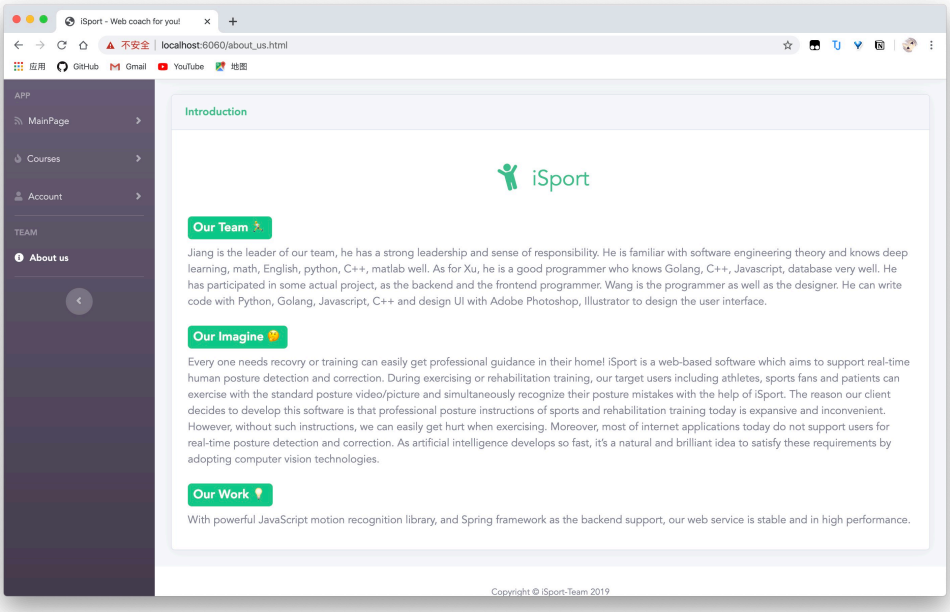


Figure 6.10: Introduction Page