Digital Gym System Software Engineering Report

Group: 62

EBU6304

Date of report: 30 May 2021

Total Number of Pages of Body Contents: 15

Table of Contents

Table of Contents ..... *Error! Bookmark not defined.*

1. Project Management ***2***

1.1 Contributions and Task Distributions **2**

1.2 Project Management Process **2**

2. Requirements ***3***

2.1 Requirements Finding Techniques **3**

2.2. User Stories **5**

2.3 Iteration Planning and Adapt to Changes **5**

3. Analysis and Design ***5***

3.1 Design Class Diagram **5**

3.1.1 Class Related to the Live Session Section 6

3.1.2 Class Related to the MyPage and Membership 7

3.1.3 Class Related to the Video Module 8

3.2 Design of the Software and Coding Evaluation **9**

3.2.1 Design of the Live Session 9

3.2.2 Design of the MyPage and the Membership 9

3.2.3 Design of the Video Module 10

3.2.4 Coding Evaluation 10

4. Implementation and Testing ***10***

4.1 Implementation Strategy and Iteration/Built Plan **10**

4.1.1 Live Session Functions Implementation and Iteration Plan 10

4.1.2 MyPage and the Membership Functions Implementation and Iteration Plan 12

4.1.3 Video Module Functions Implementation and Iteration Plan 12

4.2 Test Strategy and Test Techniques **13**

4.3 The Using of TDD **15**

Appendix ***17***

Reference **17**

Main Screenshots of the System **18**

Member Information **22**

1. Project Management

1.1 Contributions and Task Distributions

The estimated contribution percentage and member information are illustrated in the Table.1.1-1 in appendix. The task distributions for each individual are indicated in the Table.1.1-2 in appendix.

1.2 Project Management Process

We organize and manage activities by using Milestones and Deliverables shown in the Figure.1.2-1 below.

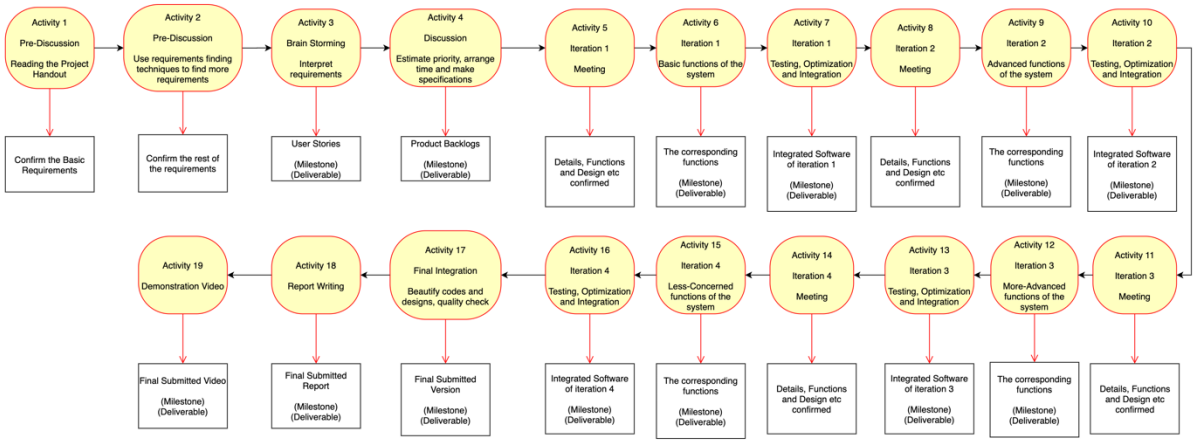


Figure.1.2-1 - Activity Organization

We arrange scheduling by using task chart in the Table.1.2-1 below. We estimate time and resources required to complete activities and organise them into a coherent sequence. Also, we split project into separate tasks, organize tasks concurrently and minimize task dependencies.

Activity	Duration(days)	Dependencies
A1	1	-
A2	1	A1
A3	4	A1,A2(M1)
A4	2	A3(M2)
A5	1	A3,A4
A6	4	A5(M3)
A7	4	A6(M4)
A8	1	A3,A4,A5
A9	4	A8(M5)
A10	4	A9(M6)
A11	1	A3,A4,A5,A8
A12	4	A11(M7)
A13	4	A12(M8)
A14	1	A3,A4,A5,A8,A11
A15	4	A14(M9)
A16	4	A15(M10)
A17	8	A7,A10,A13,A16(M11)
A18	7	M1-M11(M12)
A19	3	A17(M13)

Table.1.2-1 - Task Chart

In terms of the quality, the requirement and function should be implemented as demand, the coding should follow the coding principles, the source codes ought to be clean, readable, no-repeated, adaptable and reusable. Also, the comments and JavaDoc are required. In addition, all GUI and operation for user should be smooth, logical and friendly.
In terms of the validation, all inputs, operation, method especially for those operate on files directly need to be validated and add fault-tolerant mechanism and redundancy for exceptions.
As for collaboration, we use online meeting software for discussions, real-time synchronized document for report writing and use IntelliJ IDEA as IDE for codes writing and testing.

2. Requirements

2.1 Requirements Finding Techniques

For this online-gym project, we use background reading, interviewing, observation and questionaries as techniques.

- Background Reading

First, we read jobs descriptions on the project handout to learn more about the company and department the system is for. We noticed that this system actually has five main proportions: video watching, live session system, personal page, membership management and coach system. And each part has multiple details and functions that need to be identified.

- Interviewing

So, based on the proportions above, we established an interview with several questions to a group of people who usually hit the gym, and we structured a 1-1 meeting. The questions and several answers are as follow:

- ***Could you tell me what you do?***
  - *Student./College Student./Worker.*
- ***How often do you hit the gym?***
  - *Almost every day./4-5 times a week./1-2 times a week.*
- ***During the pandemic, would you want to go to the gym but could not due to the social-distance policy?***
  - *Yes./Absolutely./Not really.*
- ***Have you felt that you need a professional instructor to aid you during the workout?***
  - *No./Sometimes./Yes.*
- ***Why?***
  - *I'm a acquaintance with the gym and I'm familiar with all kinds of instruments and tools./Sometimes I need to exercise some particular muscles or use some new tools or the tools that need a helper and I don't know how or I don't have a partner./I'm a beginner of exercising, thus I need a professional coach to instruct me.*
- ***During the pandemic, if you need such service how would you fix this?***
  - *I searched online to check if there is a facility that provides such service./I might contact my coach privately and ask him or her if they can help me with it.*
- ***If there is a software, what kind of function would you expected most and why?***

- *It must be the function that I can book my favourite live session depending on my time schedule, coach information, content, duration and I can choose to cancel my reservation if I want./I hope that there are some videos that can help me during the exercising and I can mark the video./I hope that I can experience some privileges that can be gained by purchasing the membership.*

- Observation

We observed and experienced several current existing software including Keep by Beijing Calorie Technology Co., Ltd., Fitness by Apple Inc., etc within a week. We especially focused on all the functions that are related to the system we want to design and some of our requirements are identified through the use experience.

- Questionnaires

We designed a questionnaire with several questions and distributed it to the students in the university. Some important questions in the questionnaire and their results are as follow:

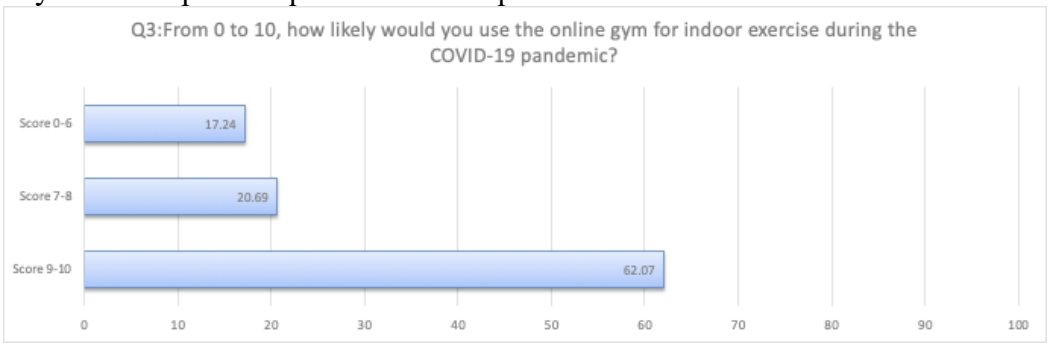


Figure2.1-1 - Question 3: 'From 0 to 10, how likely would you use the online gym for indoor exercise during the COVID-19 pandemic?' Net Promoter Score®[1] Results

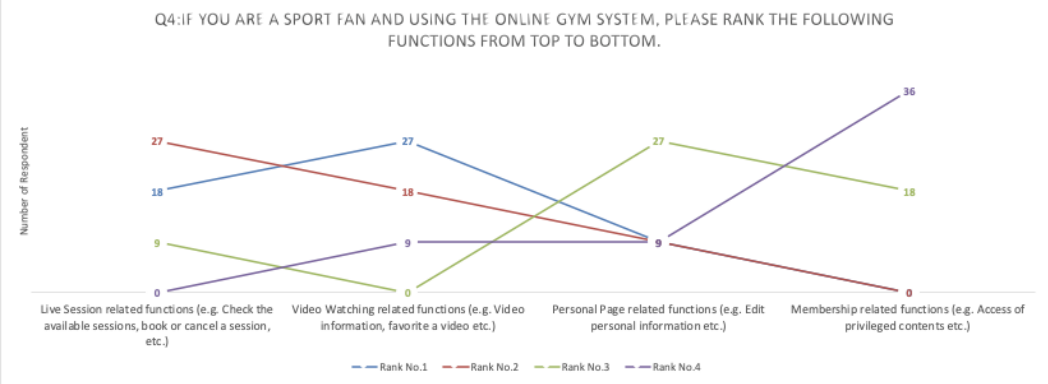


Figure2.1-2 - Question 4: 'If you are a sport fan and using the online gym system, please rank the following functions from top to bottom.' Results

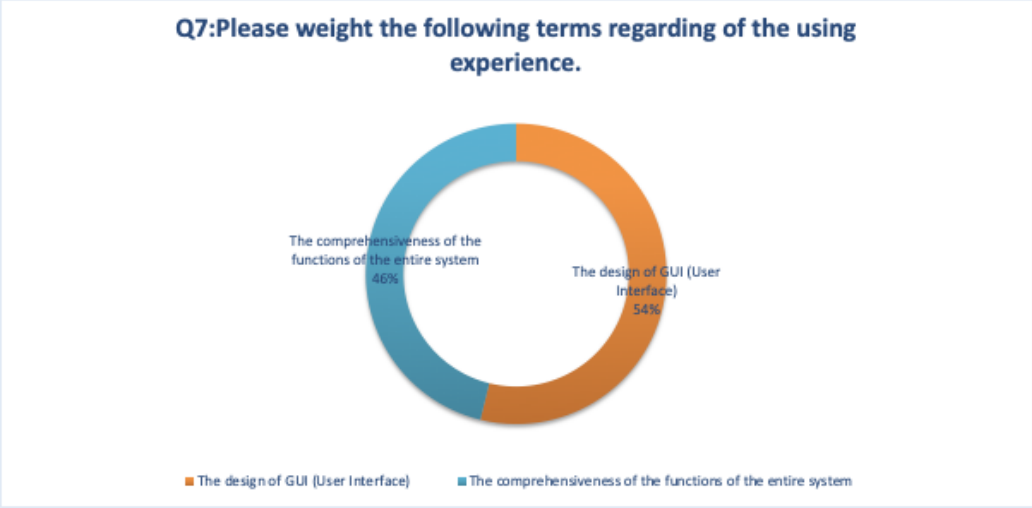


Figure2.1-3 - Question 7: 'Please weight the following terms regarding of the using experience.' Results

2.2. User Stories

The product backlogs are written in an independent Excel named as Productbacklog_Group62.xlsx. For the details, please check the corresponding file.

2.3 Iteration Planning and Adapt to Changes

We arrange our iteration timetable based on the schedule below:

- 19-23 March: set up the QM+ Hub group and discuss the project handout.
- 24-26 March: story writing workshop. Outcomes: product backlog and prototype
- 29 March-9 April: Iteration 1. Outcomes: Working Software v1
- 12-23 April: Iteration 2. Outcomes: Working Software v2
- 26 April - 7 May: Iteration 3. Outcomes: Working Software v3.
- 10-21 May: Iteration 4. Outcomes: Working Software v4.
- 24 May - 31 May: Integration, test and report writing. Outcomes: Software final delivery.
- 1 June - 3 June: Video making and submitting. Outcomes: Demonstration video.

We actually did encounter some obstacles during each iteration. For instance, we initially decided to design our GUI by using Java Swing instead of the current JavaFX which is more efficient and friendly for coding and designing. So, we self-studied the JavaFX and implemented it onto the project, and it preformed excellent. Also due to the pandemic, sometimes we were unable to communicate with each other face to face, but thanks to the online meeting system, QM+ Hub and online collaboration software and applications, we have become even more productive than before. In addition, we bumped into some accidents or miscommunications, which at that time brought us heaps of troubles. However, we managed to overcome them by adjusting our plan and improving our coding skills.

3. Analysis and Design

3.1 Design Class Diagram

We manage to separate all function into 3 sections: Live Session, Personal Information and Video Operations related sections. All the corresponding data, files, images, videos are stored under each subdirectory in ‘resources’ directory.

3.1.1 Class Related to the Live Session Section

The UML diagram of Live Session section is illustrated in Figure.3.1.1-1.

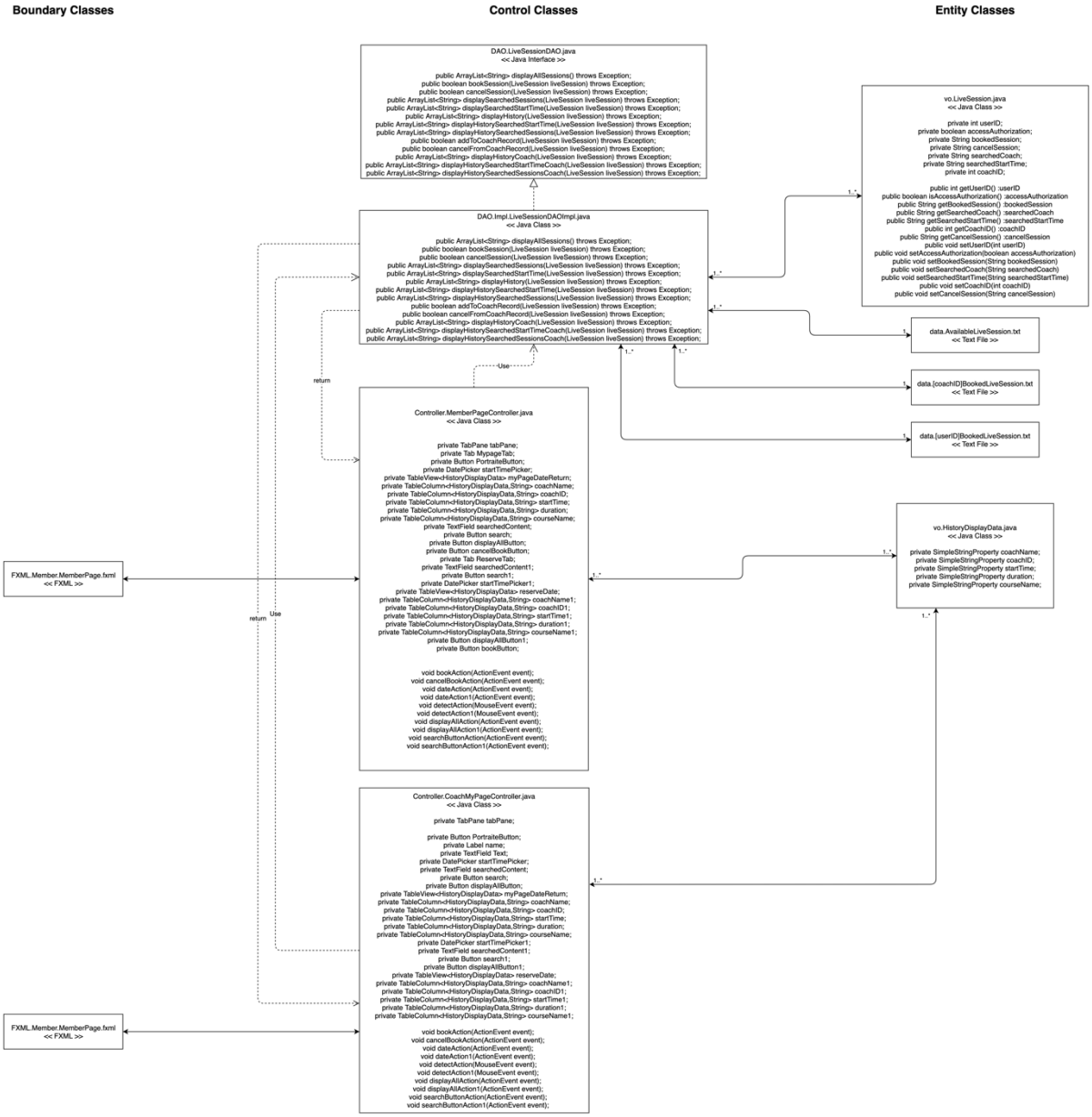


Figure.3.1.1-1 - UML Diagram of Live Session Related Classes

In terms of entity classes, LiveSession.java class contains values objects and their getters and setters which could be used in LiveSessionDAOImpl.java class whenever they are needed. Three types of text files store different information of live sessions respectively. And HistoryDisplayData.java class is created for displaying the corresponding information on TableView.

In terms of control classes, LiveSessionDAOImpl.java is a class implements the LiveSessionDAO.java interface, which contains different methods for live session functions. As the UML shows, whichever method contains keyword 'History' is related to the function that needed to be performed on user's own page. For instance, displaySearchedStartTime(LiveSession) - show personal booked live sessions, displayHistorySearchedSessions(LiveSession) - search booked live session by using entered keywords etc. And the methods without 'History' are related with operating on all available

live sessions (e.g. displayAllSessions() - display all available live session on reserve page) or with their direct functions (e.g.bookSession(LiveSession) - book a selected live session). And all methods contain keyword 'Coach' are related to the functions preformed on coach's page (e.g.displayHistorySearchedStartTimeCoach(LiveSession) - display all booked live sessions that this coach would take or have taken with a start date filter applied). And two controller classes MemberPageController.java and CoachMyPageController.java are the bridge between the backend and GUI by implementing the previous methods and gaining the return values, and then display them correspondingly on the GUI. All methods in the controllers with keyword '1' appended to the end of the names are meaning that this method will be used on the Reserve Page and without the keyword '1' means that these methods will be applied on the user's or coach's personal page.
In terms of boundary classes, we design our GUI by using Scene Builder which can automatically generate a FXML file that contains GUI structures and can be connected with the backend classes and methods by using Java.

3.1.2 Class Related to the MyPage and Membership

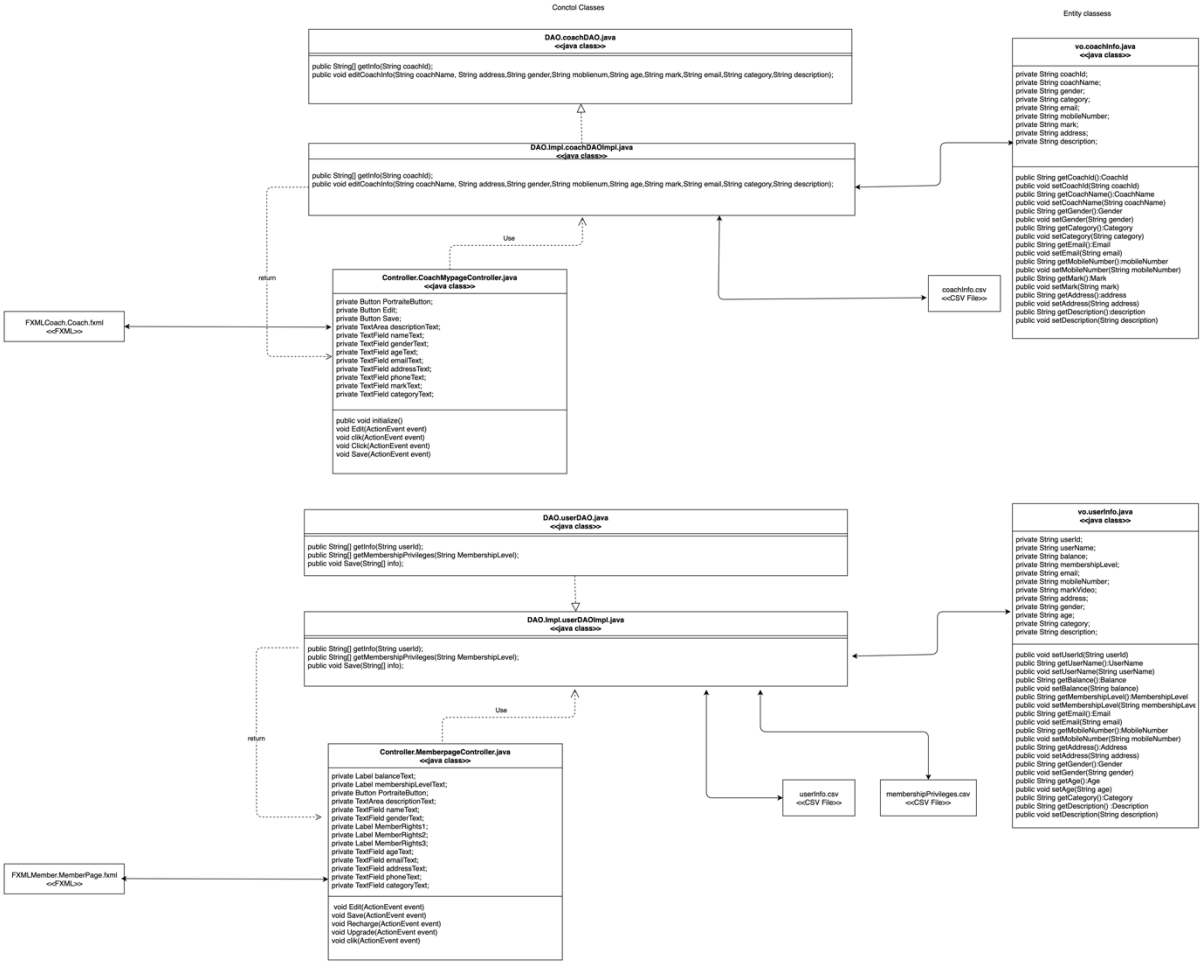


Figure.3.1.2-1 - UML Diagram of MyPage and Membership Related Classes

On the entity class side, the CoachInfo.java class and the UserInfo.java class contain value objects and their getters and setters, which can be used in the CoachDaoImpli.java class and UserDaoImpl when needed. The three CSV files store information about different people.
In terms of control classes, CoachDaoImpl. Java is a class that implements the CoachDao. Java interface, which contains the different methods applied to the trainer's personal page. UserDaoImpl is a class that implements UserDAO and contains the different methods applied

to the user's personal interface. In the control class, static methods are mainly used to pass the value of the back end to the front end, such as getInfo (String userId) : to get the user's information. Dynamic methods, on the other hand, apply to editing and recharging, such as Save (String[] info) : to store the obtained value to the back end. The controller classes CoachMyPageController and MemberPageController act as a bridge between the back end and the GUI by implementing the previous methods and getting the return values, and then displaying them accordingly on the GUI. All methods will be applied to the coaches and users' personal pages.

3.1.3 Class Related to the Video Module

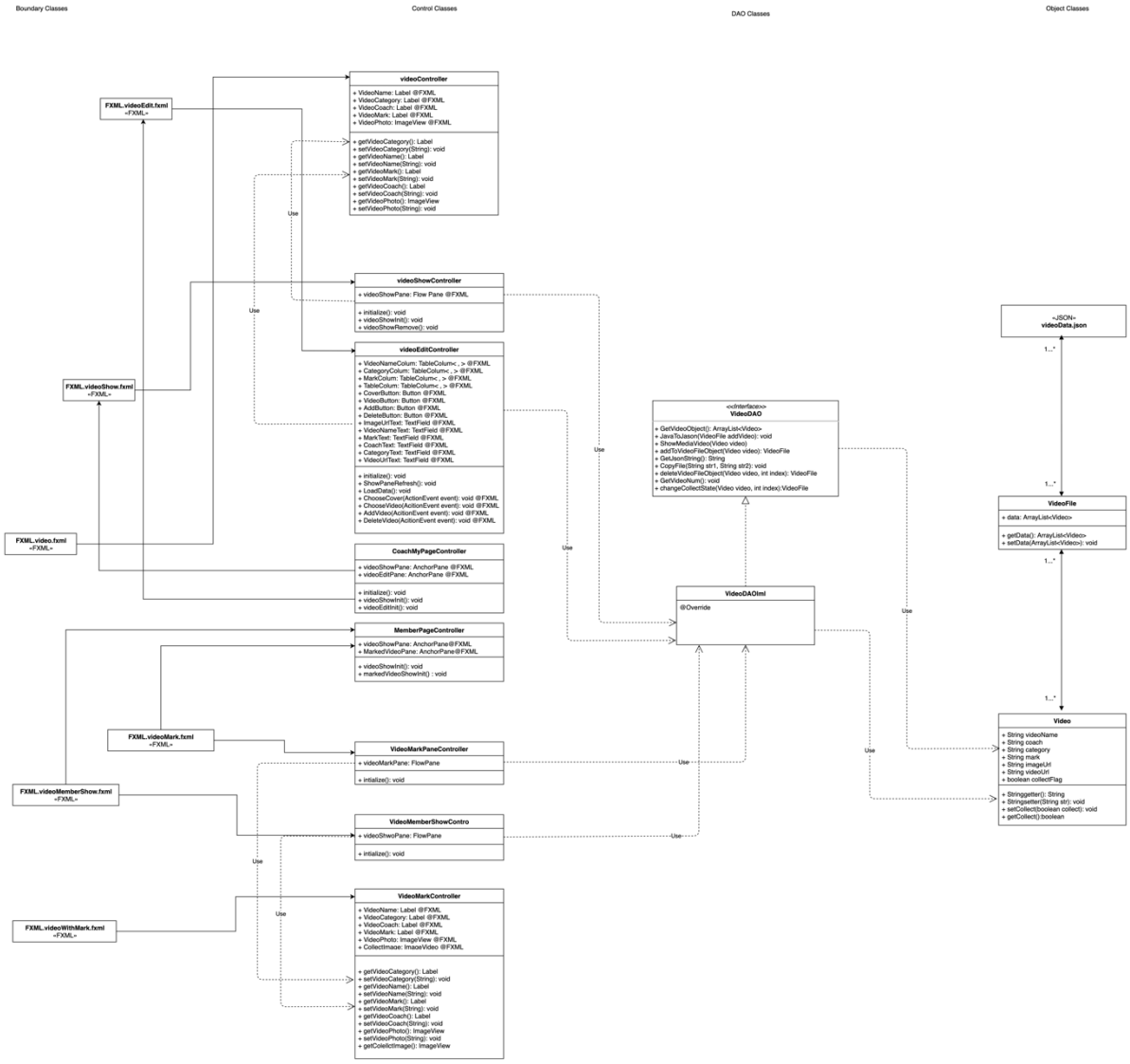


Figure.3.1.3-1 - UML Diagram of Video Module

First of all, our Video section is divided into two parts, the first is for coaches and the other is for users.

In terms of entity classes, the data of each video in these two sections comes from the object class Video.java, where each Video.java is an element of the dynamic object array in VideoFile. We use the json file videoData.json to store the video data. VideoFile and Video contain getters and setters for various properties.

We involved a total of eight Controller methods. Among them, videoContoller, videoEditController, CoachPageController, and vioShow are responsible for controlling the video functions of the coaching

section. VideoMarkController, videoMarkaneController, videoMemberShowContoller, and MemberPageController are responsible for controlling the video functions of the user section.We involved a total of eight Controller methods. Among them, videoContoller, videoEditController, CoachPageController, and vioShow are responsible for controlling the video functions of the coaching section. VideoMarkController, videoMarkaneController, videoMemberShowContoller, and MemberPageController are responsible for controlling the video functions of the user section. Then each Controller is associated with an FXML. When the Controller needs to change the video data, it will call the method in VideoDAO.

In terms of boundary classes, we disign our GUI by using SceneBuilder which can automatically generate a FXML file that contains GUI structures and can be connected with the backend classes and methods by using Java.

3.2 Design of the Software and Coding Evaluation

3.2.1 Design of the Live Session

The whole system of IO of live session is using text file (.txt) to store, add, delete, modify, extract and recognize the data. All available live sessions are stored in AvailableLiveSession.txt which contains information with the structure like '1001*Simon Smith*2021-04-01 08:00:00*2hrs*Boxing - Basic*'. '*' is the identifier for identifying each section from the entire string, and each section represents different information: coach's ID, coach's name, live session start time, live session duration and its specific descriptions respectively. Each row of the file contains one string of information only. User's booked live session will be stored in a file called [UserID]BookedLiveSession.txt where the parameter [UserID] is different for different users. The structure of the information in that file is the same as the previous one. And coach's booked live sessions will be also stored in a file called [CoachID]BookedLiveSession.txt whose structure is a little bit different - each row has an additional '[UserID]*' append to the end of each string.
Also, the system can fulfil almost all basic functions including browse all available live sessions or booked live session and their information for user and coach, query or filter them by entering contents or selecting a start date, book or cancel one particular live session for user. In addition, to increase fault tolerance and avoid exceptions occurring, we add several designs to prevent that. For instance, if only one specific live session is selected then user can cancel or book this particular live session, otherwise the system will have exceptions.

3.2.2 Design of the MyPage and the Membership

The entire system of MyPage and Membership uses CSV files to store, add, modify, extract, and identify data. The user's personal information is stored in userinfo.csv, and the coach's personal information is stored in coachinfo.csv. The user's file contains the information in the following table.

userid	username	address	gender	mobilenumber	age	category	email	description	balance	membershiplevel

Table.3.2.2-1 - User's Information File

The information in the following table is included in the coach's file.

coachid	coachname	address	gender	mobilenumber	age	mark	email	category	description

Table.3.2.2-2 - Coach's Information File

When the program reads this line, it automatically uses ", "as the delimiter. Each part represents a different piece of information. Each line of the file contains only one information string. The user and trainer click Edit and call the save method to save the newly obtained string to the corresponding file. Membership benefits are stored in a file called "membershipprivileges.csv", where membershiplevel is a different user level, each level corresponds to its level of membership benefits.

In addition, the system can also achieve basic all the basic functions. Including user and coach's personal information inquiry and modification. Users recharge the balance and check the membership rights and interests of the corresponding level.

3.2.3 Design of the Video Module

The video section is divided into coach video system and member's video system. For coach video system, first, there are two AnchorPanes in CoachController responsible for loading VideoEdit.fxml and VideoShow.fxml respectively. These two FXML files are linked to VideoShowController and VideoEditController. VideoEditControll is responsible for controlling the addition and deletion of videos. The coach needs to select the video cover and the absolute path of the video picture in FileChoose, and then the Controller will call the VideoDAO method to change the video in the json file. The data. And while adding and deleting, a folder will be generated in the project file to store video and picture files. VideoShowController replicates and controls the display of the video. First, the Controller contains a FlowPane, which implements the loading of the video. It reads each object in the dynamic array in the json file and converts it into a Viedo object, loads Video.fxml, and converts it into one A FXML component, initialized using FlowPane.getchild.add().

VideoMarkPaneController and VideoMemberShowController are responsible for the video function of the user section. The principle of VideoMemberShowController is the same as the VideoShowController of the coaching section, but there is an additional collection interface. The user can click on the favorite star. When the star turns red, it means that the user has saved the video. Click it again to cancel the collection. Each click will change the corresponding The collect (boolean) attribute in Video, the default is false and uncollected, if it is true, it means collection. After that, users can watch their favorite videos in their Reverse section. This section is controlled by VideoMarkPaneController. The initial loading of the video is also the same as before, but in the Controller method, the collect (boolean) attribute of each Video class needs to be identified. If it is True, it will be displayed and loaded.

For each Video display Pane, you need to click on the cover of the video to play the corresponding video.

3.2.4 Coding Evaluation

Speaking of coding evaluation, we broke this part of code into the smallest pieces, each class and method is only responsible for one particular thing or function. Also, we beatified our codes and reduced the duplicated codes. And we stressed the importance of comments and JavaDoc for documentation and we think we did pretty good in terms of the readability of the code.

4. Implementation and Testing

4.1 Implementation Strategy and Iteration/Built Plan

4.1.1 Live Session Functions Implementation and Iteration Plan

We use Object-Oriented programming, so I wrote codes for class definitions and method's definitions. I will illustrate the methods in detail in the following section.

- **ArrayList<String> displayAllSessions()**

Use FileReader and BuffReader to extract each row in the text file AvailableLiveSession.txt one by one, then return the ArrayList<String> that contains all of the data in the file and split them by '*' in the controllers and temporarily store split data into array info[], then display on the GUI.

- **ArrayList<String> displaySearchedSessions(LiveSession)**

Use FileReader and BuffReader to extract each row in the text file AvailableLiveSession.txt one by one, and use getter to extract the user's searched contents information stored in LiveSession then use contain() method to filter all strings which contain the searched contents, then only return the ArrayList<String> that contains the specific data in the file and split them by '*' in the controllers and temporarily store split data into array info[], then display on the GUI.

• **ArrayList<String> displaySearchedStartTime(LiveSession)**

Same as displaySearchedSessions(LiveSession), but use getter to extract the date information stored in LiveSession.

• **ArrayList<String> displayHistory(LiveSession)**

Same as displayAllSessions(), but initially use getter to extract the userID information stored in LiveSession and access to the file that belongs to the particular user. And finally display the corresponding contents.

• **ArrayList<String> displayHistorySearchedSessions(LiveSession)**

Same as displaySearchedSessions(LiveSession), but initially use getter to extract the userID information stored in LiveSession and access to the file that belongs to the particular user. And finally display the corresponding contents.

• **ArrayList<String> displayHistorySearchedStartTime(LiveSession)**

Same as displaySearchedStartTime(LiveSession), but initially use getter to extract the userID information stored in LiveSession and access to the file that belongs to the particular user. And finally display the corresponding contents.

• **boolean bookSession(LiveSession) & boolean addToCoachRecord(LiveSession)**

Firstly, use getter to extract the user's selected live session information from LiveSession. And then use FileReader and BuffReader to extract each row in the text file AvailableLiveSession.txt one by one and use contain() method to compare if the specific content of the line is match with the user's selected one, if matched then do not write this line into the temp.txt, if not write the line into the temp.txt, and after that rename the temp.txt as AvailableLiveSession.txt. And then write the deleted line into [UserID]BookedLiveSession.txt where the UserID is extracted from LiveSession Object by using the corresponding getter. In the meantime while bookSession(LiveSession) is triggered, addToCoachRecord(LiveSession) is called simultaneously to save the same information appended with '[UserID]*' at the end of each line into [CoachID]BookedLiveSession.txt.

• **boolean cancelSession(LiveSession) & boolean cancelFromCoachRecord(LiveSession)**

Same as the previous mechanism but rewrite the selected information back to AvailableLiveSession.txt and delete from [CoachID]BookedLiveSession.txt and [UserID]BookedLiveSession.txt.

• **ArrayList<String> displayHistoryCoach(LiveSession)**

Same as displayHistory(LiveSession), but access to the file that belongs to the particular coach. And finally display the corresponding contents.

• **ArrayList<String> displayHistorySearchedStartTimeCoach(LiveSession)**

Same as displayHistorySearchedStartTime(LiveSession), but access to the file that belongs to the particular coach. And finally display the corresponding contents.

• **ArrayList<String> displayHistorySearchedSessionsCoach(LiveSession)**

Same as displayHistorySearchedSessions(LiveSession), but access to the file that belongs to the particular coach. And finally display the corresponding contents.

The functions are implemented from easy and basic to difficult and advanced. The iteration plan for live session functions is as below:

• **Iteration 1**

Design the IO structure, implement the function displayAllSessions() and bookSession(LiveSession) and the corresponding controller methods and GUI.

• **Iteration 2**
Implement the function cancelSession(LiveSession), displaySearchedSessions(LiveSession), displaySearchedStartTime(LiveSession) and displayHistory(LiveSession) and the corresponding controller methods and GUI, optimize the codes in iteration 1.
• **Iteration 3**
Implement the function displayHistorySearchedStartTime(LiveSession), displayHistorySearchedSessionsCoach(LiveSession) and displayHistorySearchedSessions(LiveSession) and the corresponding controller methods and GUI, optimize the codes in iteration 1 and 2, add fault tolerance on the DAOImpl and controller.
• **Iteration 4**
Implement the function addToCoachRecord(LiveSession), cancelFromCoachRecord(LiveSession), displayHistoryCoach(LiveSession) and displayHistorySearchedStartTimeCoach(LiveSession) and the corresponding controller methods and GUI, optimize the codes in iteration 1, 2 and 3, add fault tolerance on the DAOImpl and controller, apply the test.

4.1.2 MyPage and the Membership Functions Implementation and Iteration Plan

• **ArrayList<String> getInfo(String coachId)**
According to the contents of the coach's ID query coachInfo.csv, when the ID matches Coachid, the information of the coach will be obtained and passed into the current coach class.
• **ArrayList<String> editCoachInfo(String coachName, String address,String gender,String moblienum,String age,String mark,String email,String category,String description)**
Passes the coach information obtained by the front end to the back end and writes the information to the row of the current coach ID.
• **ArrayList<String> getInfo(String userId)**
According to the contents of the user's ID query userInfo.csv, when the ID matches userid, the information of the user will be obtained and passed into the current user class.
• **ArrayList<String> MembershipPrivileges(String MemberLevel)**
According to the MemberLevel obtained from userInfo, open the membershiPrivileges.csv file, find the member rights and interests of the corresponding MemberLevel, and output it back to the front end.
• **ArrayList<String[]> save(String[] info)**
Write the modified information sent back by the front end to the corresponding user's file.
• **Iteration 1**
Determine the category of writing and implement the user and coach getInfo () method.
• **Iteration 2**
Implement other functions for users and coaches. Summary membership function, the back end part is completed.
• **Iteration 3**
Learn the GUI, complete the coach part of the GUI interface production
• **Iteration 4**
Complete the user part of the GUI interface production. Realize recharge, modify pop-up event generation, increase fault tolerance, and take verification.

4.1.3 Video Module Functions Implementation and Iteration Plan

• **ArrayList<Video> GetVideoObject()**
First call GetJsonToString to read the dynamic array of the json file, and then convert the String into a dynamic array to store the Video class.

• **VideoFile addToVideoFileObject(Video video)**
This method is called by videoEditvController, which is mainly responsible for adding new video data to the original dynamic array and returning the new dynamic array.

• **VideoFile deleteVideoFileObject(Video video, int index)**
This method is called by videoEditvController, which is mainly responsible for deleting the specified video data in the original dynamic array and returning the new dynamic array.

• **VideoFile changeCollectState(Video video, int index)**
This method is called by videoMarkPaneController and videoMemberShowController and is mainly responsible for changing the collect (boolean) property of the video after the user clicks the collection button.

• **Int GetVideosNum()**
Get the number of videos in the dynamic array.

• **void JavaToJson(VideoFile addVideo)**
Get the return value of addToVideoFileObject and deleteVideoFileObject, convert VideoFile into json object and write it into json file.

• **String GetJsonString()**
Convert json object to String from json file.

• **void CopyFile(String ori, String des)**
When the user adds a video to specify the cover path and video path, the Controller calls this method to copy the corresponding file to the resource Image and Video files for easy reading next time.

• **Void ShowMediaVideo(Video video)**
Video display method after user clicks on video cover.

• **Iteration 1**
Complete the static design of the front-end FXML components of each section.

• **Iteration 2**
Add video add, delete, check and modify method, add video editing section.

• **Iteration 3**
Complete the video presentation function of the coaching section and the user section.

• **Iteration 4**
Add the video collection function in the user section, and add a favorite section.

4.2 Test Strategy and Test Techniques

Test Strategy

- 70% of the tests will be automated.
  - The input box check will be manual.
  - This is to check if the user interface is correct.
  - Each subject use case should be tested for its normal flow and behaviour and also two alternative flows.
  - These should ensure that they cover incorrect user input.
- Success criteria – 90% of test cases passed.
- No high priority defects unresolved.

Each build we arrange specific tests which were confirmed during the integration build plan. And each build has both validation testing and defect testing. For those which need validation, we use Test Driven Development in Agile process to test the return values. All places where the user or coach can insert input needs to be validated (e.g. Personal information edition), and where the user or coach may encounter a bug needs to handle the exception or add fault tolerance (e.g. Book a live session when no live session is selected).

Test Case Design Example

1. Live Session Book/Cancel Function Fault Tolerance

– *Input*

- After displaying all data, select no live session and try to book/cancel a live session.
- After applying filters by using either contents search or date filter, try to book/cancel a live session with no course selected.
- Book/Cancel a session normally.

– *Result*

- Those tests with abnormal operations will be handled gracefully. The book/cancel button won't be able to interact while there is no course selected. The Figure.4.2-1 indicate the design.
- And the normal operations will be accepted and work as anticipated.

– *Conditions*

- No conditions exist on this test.

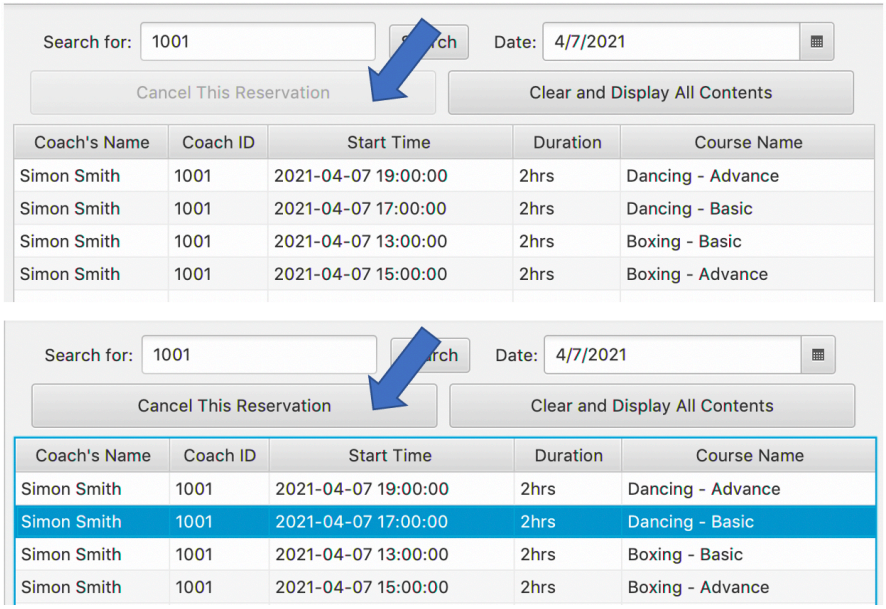


Figure.4.2-1 - Live Session Book/Cancel Function Fault Tolerance Display

2. MyPage and Membership Function Fault Tolerance

– *Input*

- Not having enough money in your wallet when upgrading
- Upgrade when you are already a Premium Member
- Normal upgrade membership

– *Result*

- When there is no money in your wallet, clicking Upgrade will prompt you that there is no money in your wallet. This method can only be performed correctly if you have enough money in your wallet
- Warning you are already a Premium member
- And the normal operations will be accepted and work as anticipated.

– *Conditions*

- Not enough balance left in the account
- Have already been a premium member

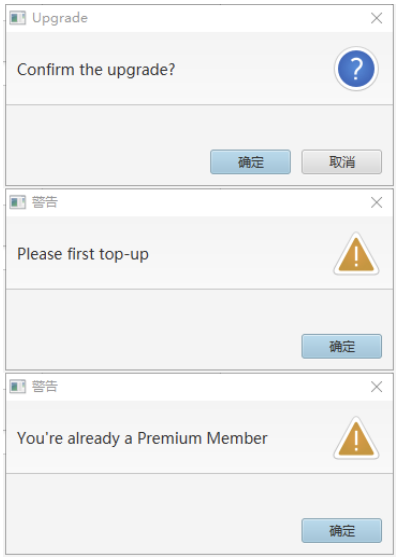


Figure.4.2-2 - MyPage and Membership Function Fault Tolerance

3. Video and Cover Upload Functions Fault Tolerance

– *Input*

- URL of video
- URL of cover
- Other information of the video

– *Result*

- Coach can upload the video and its cover simply by clicking the corresponding button and after selecting the file from the file selection window the URL will be automatically generated.

– *Conditions*

- Video should be in format *.mp4, *.mp3, *.acc, *.flac
- Cover should be in format *.jpg, *.jpeg, *.png

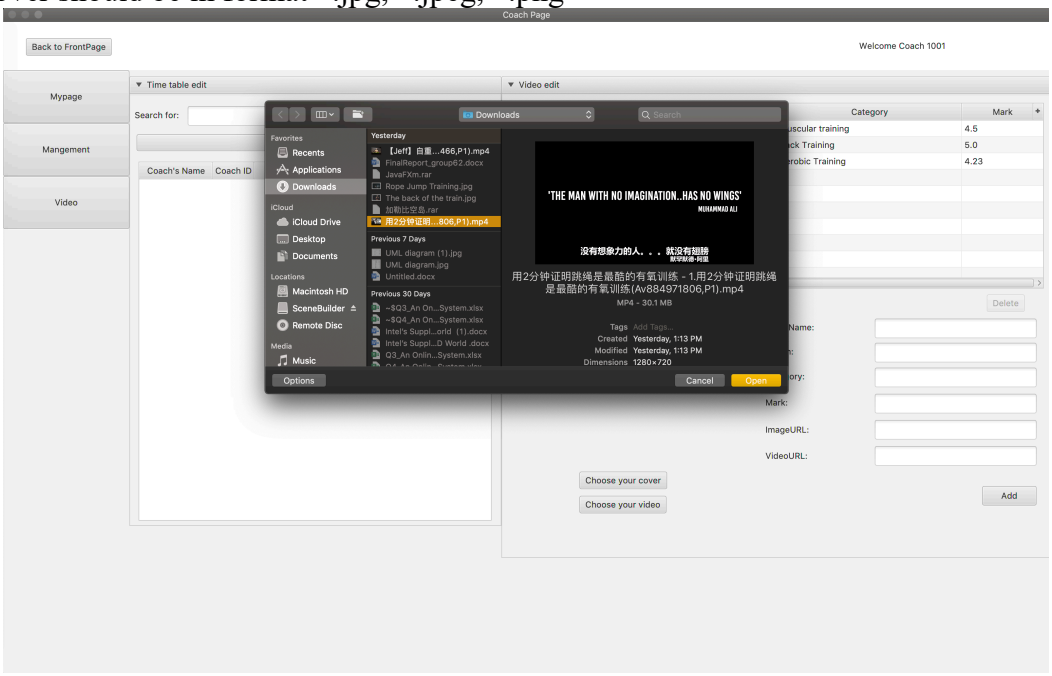


Figure.4.2-3 – Coach Video and Cover Upload Function Fault Tolerance

4.3 The Using of TDD

We mainly test the methods in vo and DAOImpl packages, the following figures show some of the results.

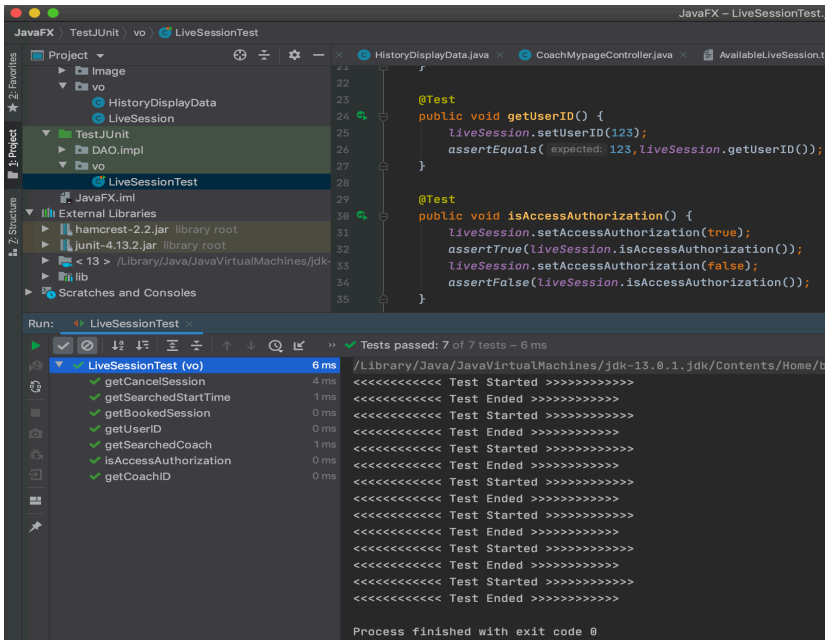


Figure.4.3-1 - Live Session vo Methods TDD Result

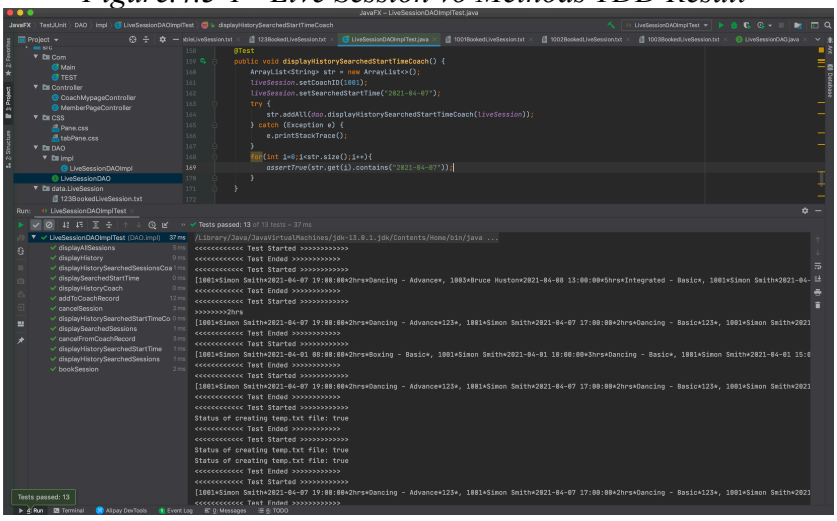


Figure.4.3-2 - Live Session DAOImpl Methods TDD Result

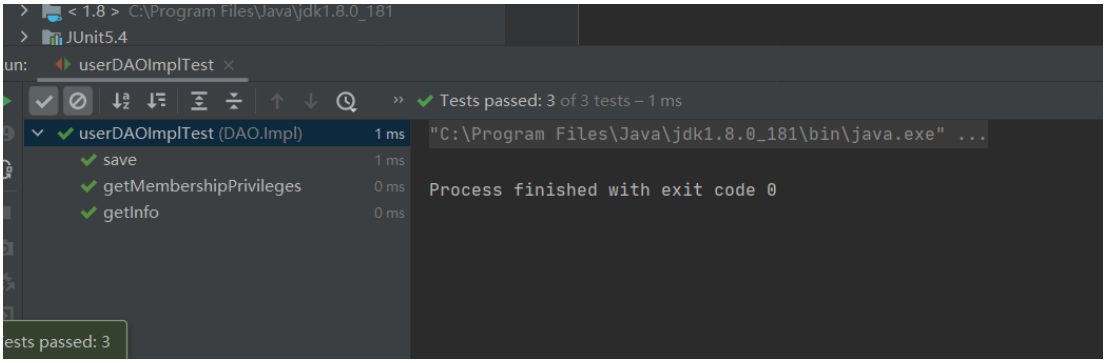


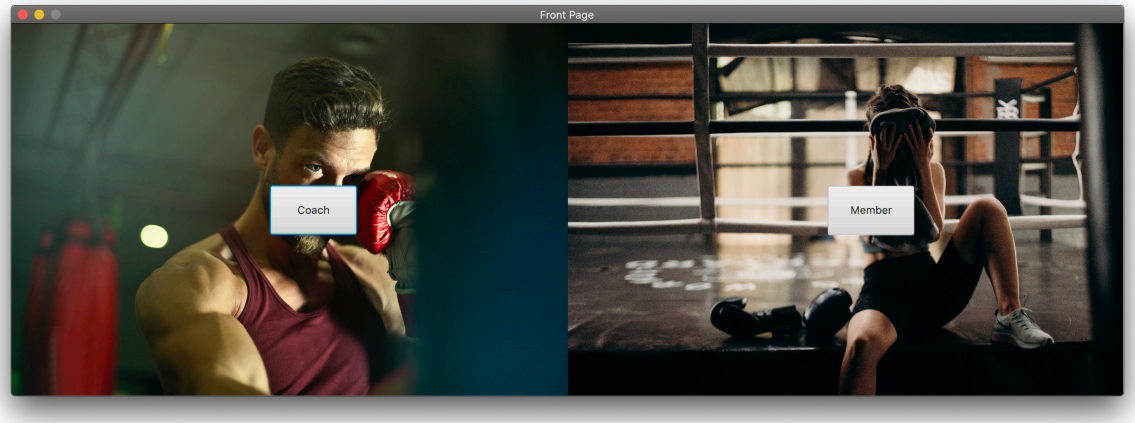
Figure.4.3-3 - MyPage DAOImpl Methods TDD Result

Appendix

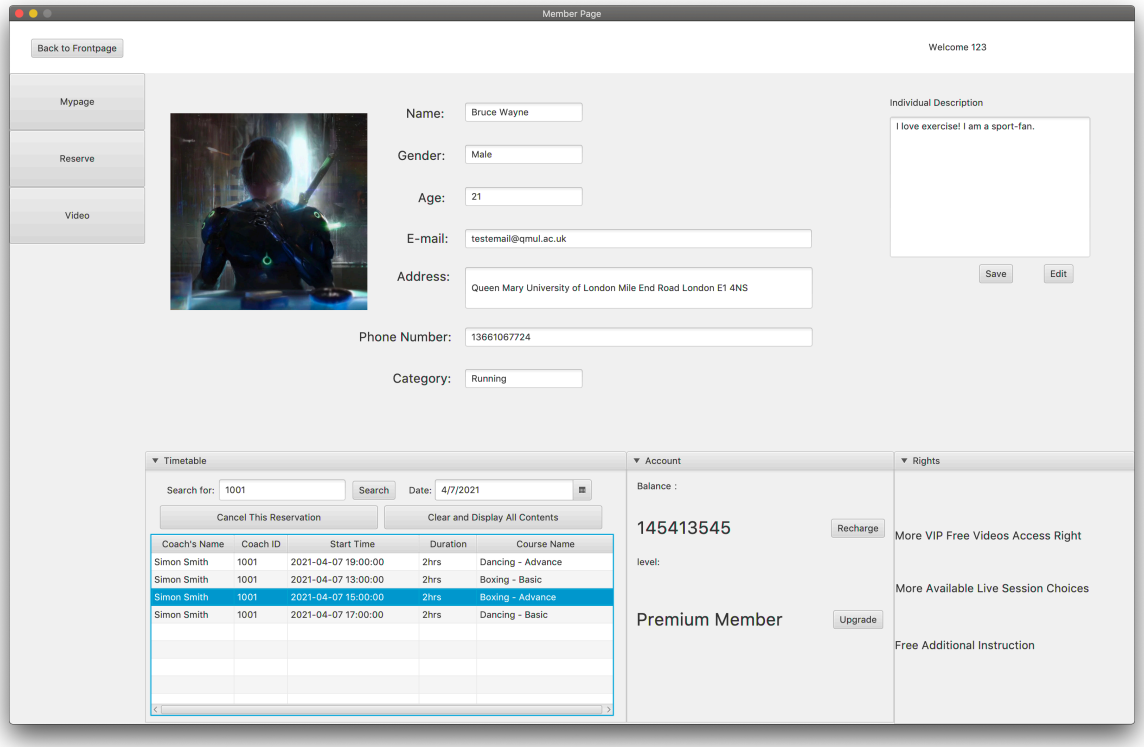
Reference

1. NICE Systems,Inc.,(2021).*What Is Net Promoter?*[online].NICE Satmetrix.[Viewed 24 May 2021].Available from: https://www.netpromoter.com/know/

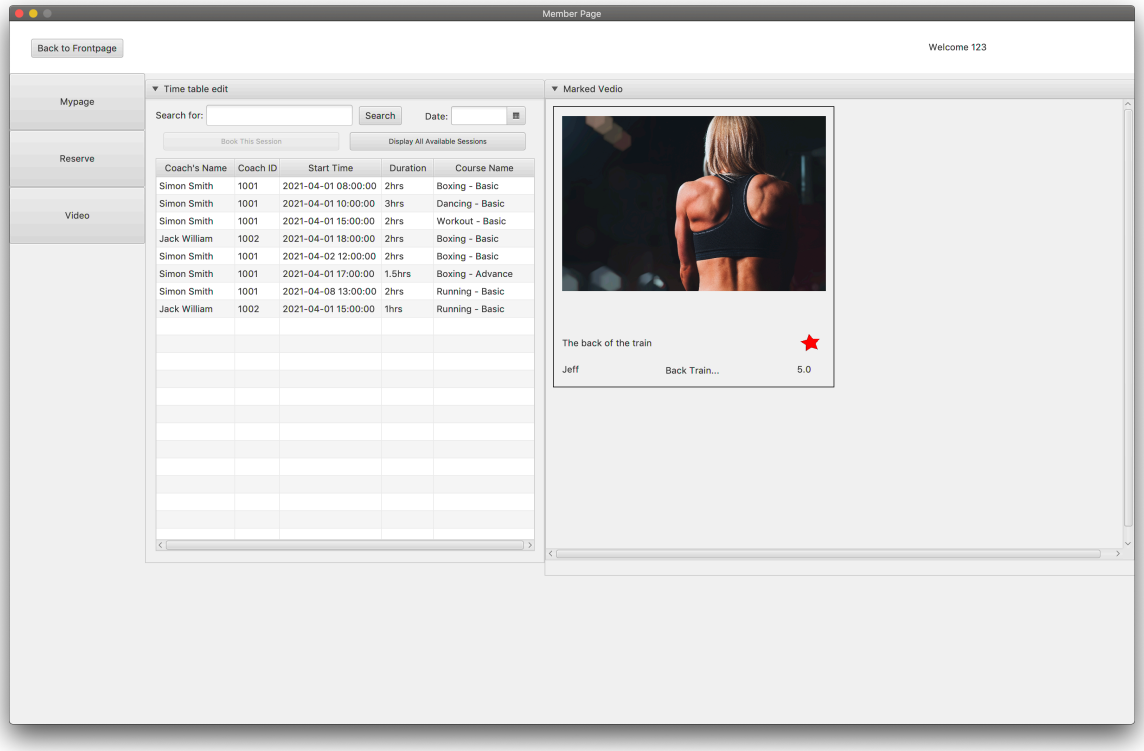
Main Screenshots of the System



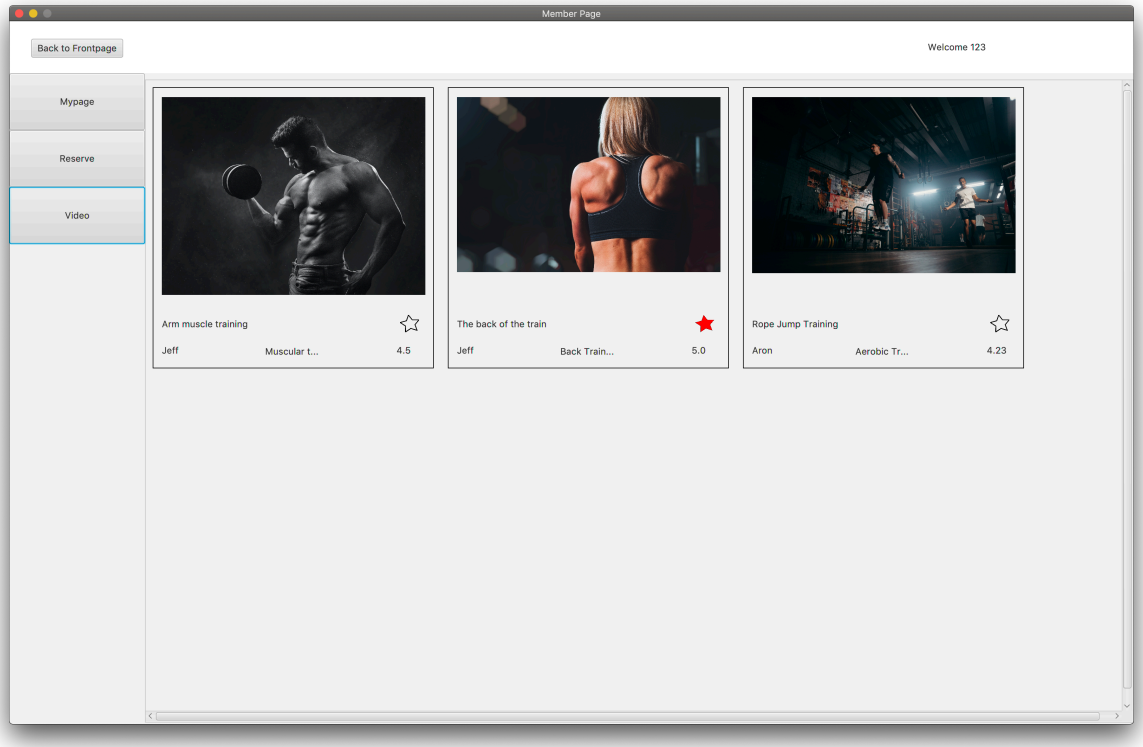
Appendix.1 – Front Page Screenshot



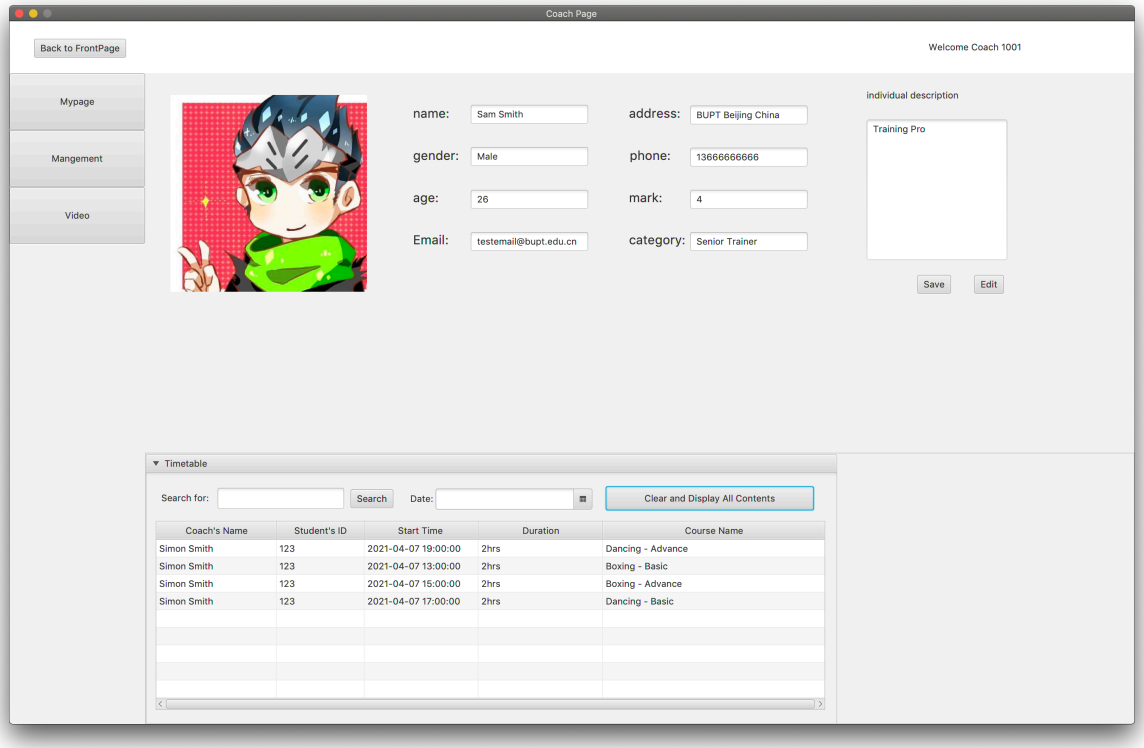
Appendix.2 – Member My Page Screenshot



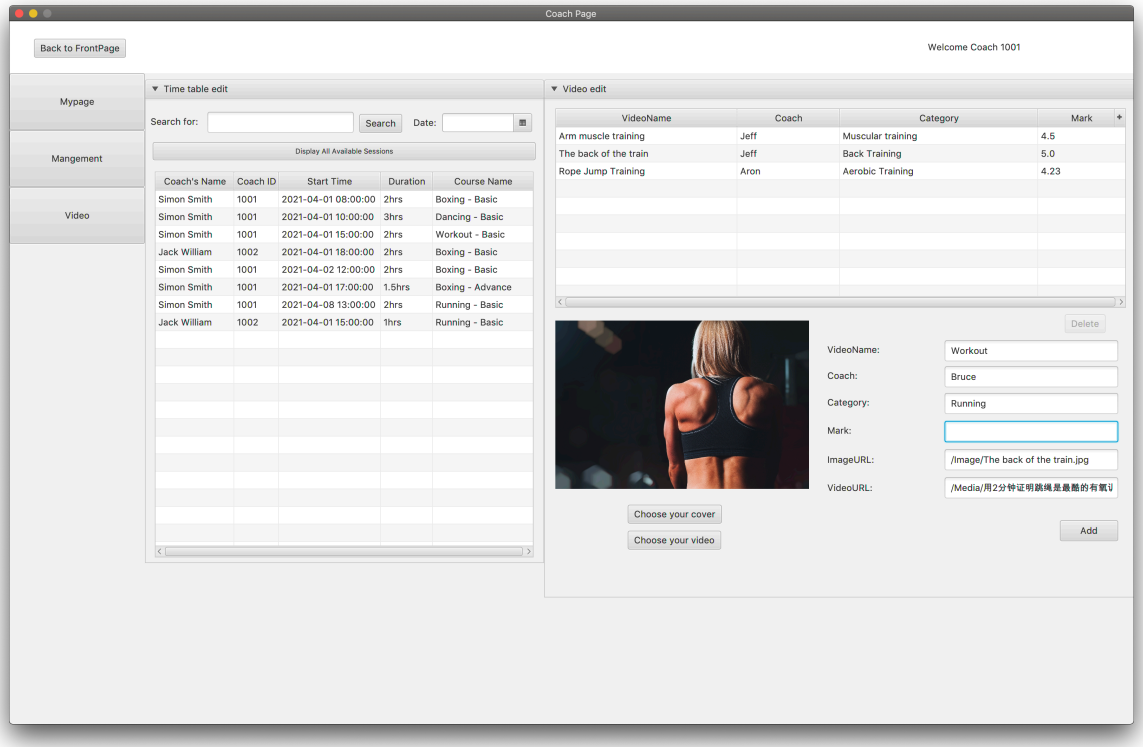
Appendix.3 – Member Reserve Page Screenshot



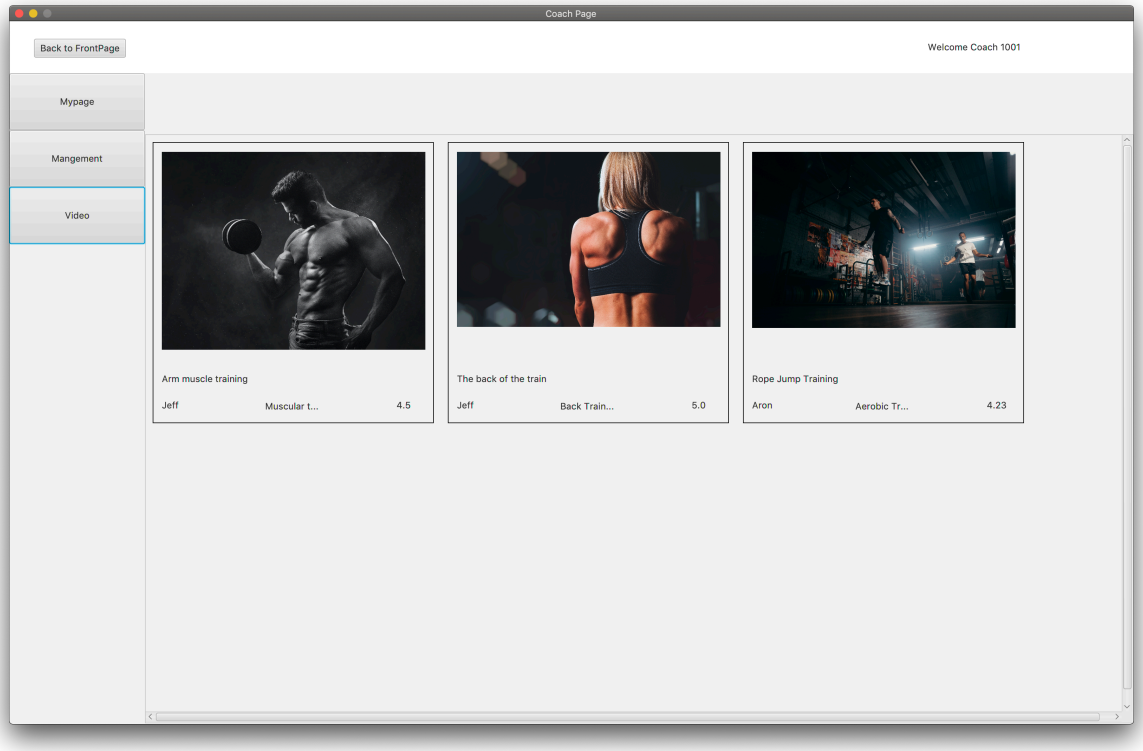
Appendix.4 – Member Video Page Screenshot



Appendix.5 – Coach My Page Screenshot



Appendix.6 – Coach Management Page Screenshot



Appendix.7 – Coach Video Page Screenshot

Appendix.8 – Video Player Screenshot

Member Information

Name	QM ID	BUPT ID	Contribution Percentage
Jiayi Zhang	190016405	2018212953	20%
Wangbo Li	190016955	2018212958	20%
Rui Ma	190016782	2018212943	15%
Yun Ge	190017549	2018212946	10%
Zhiyuan Hu	190017756	2018212963	15%
Peng He	190016737	2018212967	20%

Table.1.1-1 - Members' Information and Contribution Percentage

Name	Distributed Tasks
Jiayi Zhang	• Be responsible for the organization of every meeting, discussion and report writing, and of the corresponding QM+ Hub Journals writings. • Be responsible for all tasks and functions that related to the live session, including display all available live sessions and theirs information, search a specific available live session information by keyword(s) or start date, book a live session, display a user's booked live sessions and theirs information, search a specific booked live session information by keyword(s) or start date, cancel a booked live session and make it available again in the available live session table, display a coach's live session arrangements, search coach's booked live sessions by the keywords and date, the redesigned GUI of the above sections and their integrations. • Be responsible for the report writing of my distributed tasks in sections Analysis and Design and Implementation and Testing, and that of project management and requirements sections. • Be responsible for the final report integration. • Be responsible for the QM+ Hub individual's Pages updates.
Wangbo Li	• Be responsible for attending each meeting and make supplement and inquiry to your own part at this meeting. • Responsible for all tasks and functions related to personal pages, including displaying personal information of users and coaches, modifying personal information of users and coaches, sorting out FAQ, describing membership rights and interests, and checking user balance. Design and integrate the GUI interface for the content. Since the contents related to recharge and membership also belong to MyPage interface, I also integrated them when I integrated them. • Be responsible for the report writing of my distributed tasks in sections Analysis and Design and Implementation and Testing, and that of project management and requirements sections. • Responsible for writing the final report of my part • Be responsible for the QM+ Hub individual's Pages updates.

Zhiyuan Hu	• Be responsible for attending each meeting and making up for the part you are responsible for in the meeting. • Responsible for all the functions related to the member part and part of the functions related to the personal page, including the personal member information, recharge function, purchase member and upgrade member level function. • Be responsible for the report writing of my distributed tasks in sections Analysis and Design and Implementation and Testing, and that of project management and requirements sections. • Responsible for writing the final report of my part • Be responsible for the QM+ Hub individual's Pages updates.
Rui Ma	• Be responsible for attending each meeting and writing video watching part code. • Responsible for video watching function including display all the videos, search videos by key words,search videos by type, recommend videos to users and add video to user favourite. • Be responsible for videoEdit function including adding and deleting videos. • Be responsible for the report writing of my distributed tasks in sections Analysis and Design and Implementation and Testing, and that of project management and requirements sections. • Responsible for writing the final report of my part. • Be responsible for the QM+ Hub individual's Pages updates.
Yun Ge	• Be responsible for attending each meeting and writing video watching part code. • Be responsible for compiling the relevant function codes of coaching system including checking the class schedule by week and a special date, checking the course progress by the names of students, modifying course information, checking rating by course, checking students' rating and coach home page. • Be responsible for the report writing of my distributed tasks in sections Analysis and Design and Implementation and Testing, and that of project management and requirements sections. • Responsible for writing the final report of my part. • Be responsible for the QM+ Hub individual's Pages updates.
Peng He	• Responsible for the design of the front page with FXML file • Attend each meeting and take notes and set the direction of the project • Responsible for writing the code of the video collection function • Be responsible for adding and deleting video and modifying and testing the code of video display board • Responsible for the final overall code integration • Responsible for writing the final report of my part. • Be responsible for the QM+ Hub individual's Pages updates.

Table.1.1-2 - Task Distributions