A Digital Online Gym System

Group 105

Group member	QM number	BUPT number
Liangrun Da	190018672	2018213144
Yufan Liu	190019565	2018213129
Lingsong Feng	190018694	2018213134
Wenrui Zhao	190019082	2018213138
Zhuonan Chang	190019831	2018213132
Jinghao Lyu	190018580	2018213143

Contents

I. Introduction

We basically developed our Digital Online Gym System by using Agile Methods. We divided the project into 5 iterations and during each iteration, the process captures requirements, analysis and design, implementation, and test were repeated. We also implemented some Extreme Programming(XP) practices, such as pair programming to make our work more effective. This report shows how we applied Agile Methods into our work and finally delivered a complete Digital Gym System.

We divided the whole project into four parts: 1. Backend; 2. Frontend & UI interface + event response; 3. Control logic; 4. Test + document + video. Everyone has a clear division of tasks. Every week we would attend a group meeting to discuss the completed work and following tasks. After 5 iterations, we finally finished our project.

II. Agile Project Management

We used the Scrum approach in Agile project management to accomplish our software. There are three phases in our Agile Project Management[1].

1) Initial phase (Project planning and Architectural design)
2) Sprint cycles (Assess, Select, Develop, and Review)
3) Closure phase.

2.1First Phase: Project Planning and Architectural Design

2.1.1 Project Planning

At the beginning of our project, according to the instruction from lecturers, we established constraints, including delivery date and so on. Then we captured requirements and wrote user stories, then we defined milestones and deliverables, such as prototype, UML diagram, and programs after each iteration. During the process, we continuously draw up a schedule, initiate activities, review progress, revise the original plan, update the schedule and renegotiate constraints and deliverables.

2.1.2 Project scheduling

We scheduled the project by splitting the project into separate tasks, organizing tasks concurrently, and minimizing task dependencies. We also held group meetings frequently to adjust our schedule to adapt to change. The task table and Gantt Chart can be found in the appendix. **(Click here to see Task Table and Gantt Chart)**

2.1.3 The Design of Software Structure

All data is recorded in a CSV file including customer.csv, coach.csv, live session.csv, video.csv. The structure of our Digital Gym System is formed of seven main GUI interface:

1) Administrator Page, which is used for administrators to do membership management, video management, and Live Session Management.
2) Classes Category Interface. The user can choose a different category of fitness using the button with a picture embedded in it.
3) Contact Interface. The users can raise their opinions.
4) Login Interface, which is used for users to log in with their username and password.
5) Main Page, which is it is a card layout panel used to transfer from one panel to another panel.
6) Personal Page, which shows users' personal information.
7) Search Interface, which is used to search by keywords and filter the searching result.

2.2 Second Phase: The Sprint Cycle

2.2.1 Assess (Product backlog)

Firstly, we catch requirements by observing similar products and designing questionnaires. After the requirements capture stages, we wrote user stories based on user requirements. Then, we added Acceptance Criteria to each story and discussed them together. A list of product requirements in a prioritized order ---Product Backlog was created. We planned and estimated each stage of the whole project by using the task table and Gantt chart, which were mentioned above.

2.2.2 Select (Requirements)

In agile development, to acquire an efficient implementation of our system, we need to make our requirements as accurately as possible[1]. So at the beginning of each iteration, we always held a group meeting to select the functions implemented from the Product Backlog. We would discuss together to select a function which can meet customers' real requirements.

2.2.3 Develop (Java coding)

According to Product Backlog, we analyze and design functions. And then we use java to implement the function. we chose web-based architecture to accomplish the separation of front and back ends.

2.2.4 Review (Finish coding)

After the implementation of chosen stories, we tested the functions and then reviewed our Product Backlog in the conference to make sure the fulfillment of requirements. Then we would have a group meeting to discuss together. Once we came up with new demands or when some requirements were found impractical to implement, we will update the Product Backlog accordingly and make sure the next iteration that we developed can adapt to changes.

2.3 Third Phase: The Project Closure

After 5 iterations which cost about 2 months, we finished our Digital Gym System which can meet all requirements of users. Then the project comes to the project closure phase now, and we would complete the final report, record the video and summarize the experience learned from this project.

2.4 Risk Management

A risk is a likelihood that something that will affect the project, product and business negatively may happen during the software development process[2]. We were mainly facing three kinds of risks: project risks, product risks, and business risks. we assessed the likelihood and consequence of these risks and draw up plans to minimize the effects of the risks, we add 30% for anticipated problems and a further 20% to cover unanticipated problems. Multiple avoidance strategies are applied, e.g., be careful about using new or unfamiliar products (tools, software, hardware). So although we encountered some risks such as the absence of members due to the illness and the unexpected bugs, we finally deal with them successfully.

2.5 Quality Management

High-quality software is easy to use, easy to modify and understand the codes by other developers, and performing the most important functions. To ensure the software quality, for each iteration, after a subgroup completed their code, a member from another subgroup would check the quality of the system from the point of view of the customer, then give feedback and the developer would improve.

III. Requirements

3.1 Apply the Requirements Finding Techniques

3.1.1 Observation

After observing a few digital gym systems, such as Keep and Fitness, some general principle were found as follows:

- To earn more profits, users are encouraged to be registered as VIP members.
- To encourage more consumption, VIPs has more rights and expense discount compared with non-VIPs.
- Courses are classified into different categories for users with clear training goals to have a quick view.
- Users can book a live training course with a specific coach and select the time.
- Users can see their training courses and lives history on their homepage clearly.
- Based on our observation, we realize the importance of membership, booking, and user friendly.
- User history query and booking are essential functions.

To design our system more user-friendly. We also find some additional requirements:

(1) Managers of the gym often want to know details of the historical order in the last month to see if the profits of the online gym are still satisfying. Therefore, we add a new function that can see the monthly purchase in the form of charts.

(2) Users often input invalid content in the boxes, consequently, the validation of input plays an important role in the software.

(3) Users often have the desire to chat with their friends or gym coaches online. As a result, email sending functions are added to meet their requirements.

3.1.2 Interviewing

Some interviews were done to find out the demand for an online gym system. We interviewed three roles of people (customers, coaches, and administrators).

For customers, having an online system for watching training videos and booking live sessions is much more convenient and time-saving than the traditional training method. User-friendly UI and functions to help them train (or teach) better are two main considerations for them. The user page will be clearer and more beautiful to meet their requirements.

As for the gym administrators, they want the software to check history order as well as have a good-looking and easy-understandable UI.

3.1.3 Questionnaire

Based on the observation and interviewing content, we designed our questionnaire. **(Click here to view Questionnaire)**

Most of the users with interest should be aged between 18 and 35. Most people think that the interface is important, proving that a user-friendly interface can bring excellent interaction to the customer. What's more, a majority of users consider that the subdivision of the workout videos and live sessions is a must for users to be clear at a glance while making a selection. Easy to book live sessions and watch workout videos is also their primary demand.

Through the questionnaire, our team now has a better understanding of the importance of different features and functions. The results were critical during the prioritization of user stories.

3.1.4 Background Reading

To understand the further needs of users, there is no denying that it is important to get close to the real situation and really understand what users exactly want. We did a number of case studies, including production reports from other gyms and some online gym software. The documentation of existing systems and other real users were also analyzed.

3.2 Prototyping

In view of the gathered requirements, we made a prototype of our programming so that we could get fast feedback during the prompt stage of planning. We initially did a few low-fidelity prototyping with paper and sketch. To present our software clearly and get the feedback more precisely, we enhanced our prototype by utilizing Adobe Xd. **(Click here to view the prototype)**

3.3 Writing User Stories

3.3.1 Epics and Requirements

With the help of the application of fact-finding techniques, we converted the information we got into epics, then we summarized functional requirements and other requirements.

(1) Functional Requirements for Online Gym System

- ✓ Customers can search the videos by keywords and sort the video search result by view counts, likes, etc.
- ✓ Customers can select the videos of a particular category.
- ✓ Customers can rate the videos, give likes to some videos, and comment on the videos.
- ✓ Customers can add some videos to their favorite list.
- ✓ Customers can know the hottest video in recent days.
- ✓ Customers can have a historical record that keeps the videos that they've watched before.
- ✓ Customers can filter the live sessions based on their personal plan and sort the search result.
- ✓ Customers can book the live session that they have selected and add the book to their own class list.
- ✓ Customers can give some feedbacks or raise some questions for the live sessions.
- ✓ Customers can easily see their membership-related information.
- ✓ Customers can receive a timely reminder before the expiration of their membership.
- ✓ Customers can get some membership rights and clearly know all of them.
- ✓ Customers can give some advice to the gym.

(2) Functional Requirements for Management System

- ✓ Administrators can see all of the existing customer hierarchy.
- ✓ Administrators can look up Customers by several filters.
- ✓ Administrators can sort the result by different standards.
- ✓ Administrators can operate the information for a customer.
- ✓ Administrators can add videos to the platform provided for customers.
- ✓ Administrators can operate the information of videos.
- ✓ Administrators can see the monthly income of the membership fee.
- ✓ Administrators can operate on membership-related information such as price, level, rewards points, etc.
- ✓ Administrators can operate categories of training videos to the software.

3.3.2 Glossary

To make every member in our Agile team clear with some terms, we made a glossary. **(Click here to view the glossary)**

3.3.3 Writing User Stories

We broke down epics into user stories that can be seen in the *backlog*.

3.4 Priorities and Estimation of the Stories

We divided stories into must-have, should have, could have, and want to have. Then we used *MoSCoW rules* to do the prioritization. For the elements that must have, we put them into very high priority. Meanwhile, the elements which are put into low or very low priority are the elements users want to have.

3.4.1 Priorities the User Stories (Using *MoSCoW Rules*)

Story ID	Story Name	MoSCoW rule	Priority
L01	Search the live session based on personal plan	Could have	medium
L02	Sort the live session search result	Should have	high
L03	Book the live session	Should have	high
L04	Add the live session to class list	Could have	medium
L05	Add comments after finishing the live session	Want to have	low
M01	Check my own membership level	Should have	high
M02	View the remaining membership time	Should have	high
M03	View membership rewards points	Should have	high
M04	Membership expiration reminder	Should have	high
M05	Birthday discount	Could have	medium
M06	Exclusive live and recorded broadcasting	Want to have	very low
M07	Display of Member Benefits	Must have	very high
M08	Contact Panel	Should have	high
V01	Keyword Searching	Must have	very high
V02	Sort the video result	Should have	high
V03	Selecting Category	Must have	very high
V04	Rating Video	Could have	medium
V05	Likes Video	Could have	medium
V06	Favourite list	Could have	medium

V07	Commenting Video	Could have	medium
V08	Light off	Want to have	low
V09	Hottest Video	Want to have	very low
V10	History record	Could have	medium
A01	Check the level hierarchy	Must have	very high
A02	Check the customer's name	Should have	high
A03	Check the customer's level	Could have	medium
A04	Check the customer's gender and issue benefits	Could have	medium
A05	Sort the search result	Should have	high
A06	Modify the information of customer	Should have	high
A07	Adding Videos	Should have	high
A08	Modification of videos	Should have	high
A09	Open account	Should have	high
A10	Monthly income	Want to have	low
A11	Add new price/level name/points range	Want to have	very low
A12	Delete or change price/level name/points range	Want to have	very low
A13	Add categories	Should have	high

3.4.2 Iteration Planning

We did the iteration planning based on the priority above.

Story ID	Story Name	Iteration planning
L01	Search the live session based on personal plan	3
L02	Sort the live session search result	2
L03	Book the live session	3
L04	Add the live session to class list	3
L05	Add comments after finishing the live session	4
M01	Check my own membership level	2
M02	View the remaining membership time	2
M03	View membership rewards points	2
M04	Membership expiration reminder	3
M05	Birthday discount	4
M06	Exclusive live and recorded broadcasting	5
M07	Display of Member Benefits	1
M08	Contact Panel	3
V01	Keyword Searching	1
V02	Sort the video result	2
V03	Selecting Category	1
V04	Rating Video	3

V05	Likes Video	3
V06	Favourite list	3
V07	Commenting Video	4
V08	Light off	4
V09	Hottest Video	5
V10	History record	3
A01	Check the level hierarchy	1
A02	Check the customer’s name	2
A03	Check the customer’s level	2
A04	Check the customer’s gender and issue benefits	2
A05	Sort the search result	2
A06	Modify the information of customer	2
A07	Adding Videos	3
A08	Modification of videos	3
A09	Open account	3
A10	Monthly income	4
A11	Add new price/level name/points range	5
A12	Delete or change price/level name/points range	5
A13	Add categories	5

3.4.3 Estimation of User Stories

We did the estimation of the story by using the *story points*.

Story ID	Story Name	Story Points
L01	Search the live session based on personal plan	5
L02	Sort the live session search result	2
L03	Book the live session	2
L04	Add the live session to class list	2
L05	Add comments after finishing the live session	2
M01	Check my own membership level	1
M02	View the remaining membership time	1
M03	View membership rewards points	1
M04	Membership expiration reminder	3
M05	Birthday discount	2
M06	Exclusive live and recorded broadcasting	5
M07	Display of Member Benefits	2
M08	Contact Panel	5
V01	Keyword Searching	3
V02	Sort the video result	2

V03	Selecting Category	2
V04	Rating Video	1
V05	Likes Video	1
V06	Favourite list	3
V07	Commenting Video	1
V08	Light off	1
V09	Hottest Video	1
V10	History record	5
A01	Check the level hierarchy	2
A02	Check the customer's name	1
A03	Check the customer's level	1
A04	Check the customer's gender and issue benefits	1
A05	Sort the search result	2
A06	Modify the information of customer	3
A07	Adding Videos	2
A08	Modification of videos	3
A09	Open account	2
A10	Monthly income	8
A11	Add new price/level name/points range	2
A12	Delete or change price/level name/points range	2
A13	Add categories	2

3.4.4 Adapt to Changes

Facing the project risks during the development, some iterations adjustments were made as an adaptation.

For insurance, we planned to finish the user story Commenting Videos in iteration 3. However, due to the illness of a skillful group member, we actually accomplished it during iteration 4.

Besides, we planned to finish the story A beginner's guide for users in iteration 5. Nevertheless, as time passed by, we found that our product is user-friendly enough for the majority of actual customers that we invited for involvement to ensure the high-quality UI. Consequently, we deleted it from the user story since it volatile the Valuable metrics according to the INVEST criteria[3] and was not a good story anymore.

We also faced the requirements conflict between our incoming user story Want more discount of customers and previous user story Earn more money from coaches. After detailed considerations, we negotiated with both sides and finally agreed to keep the discount unchanged.

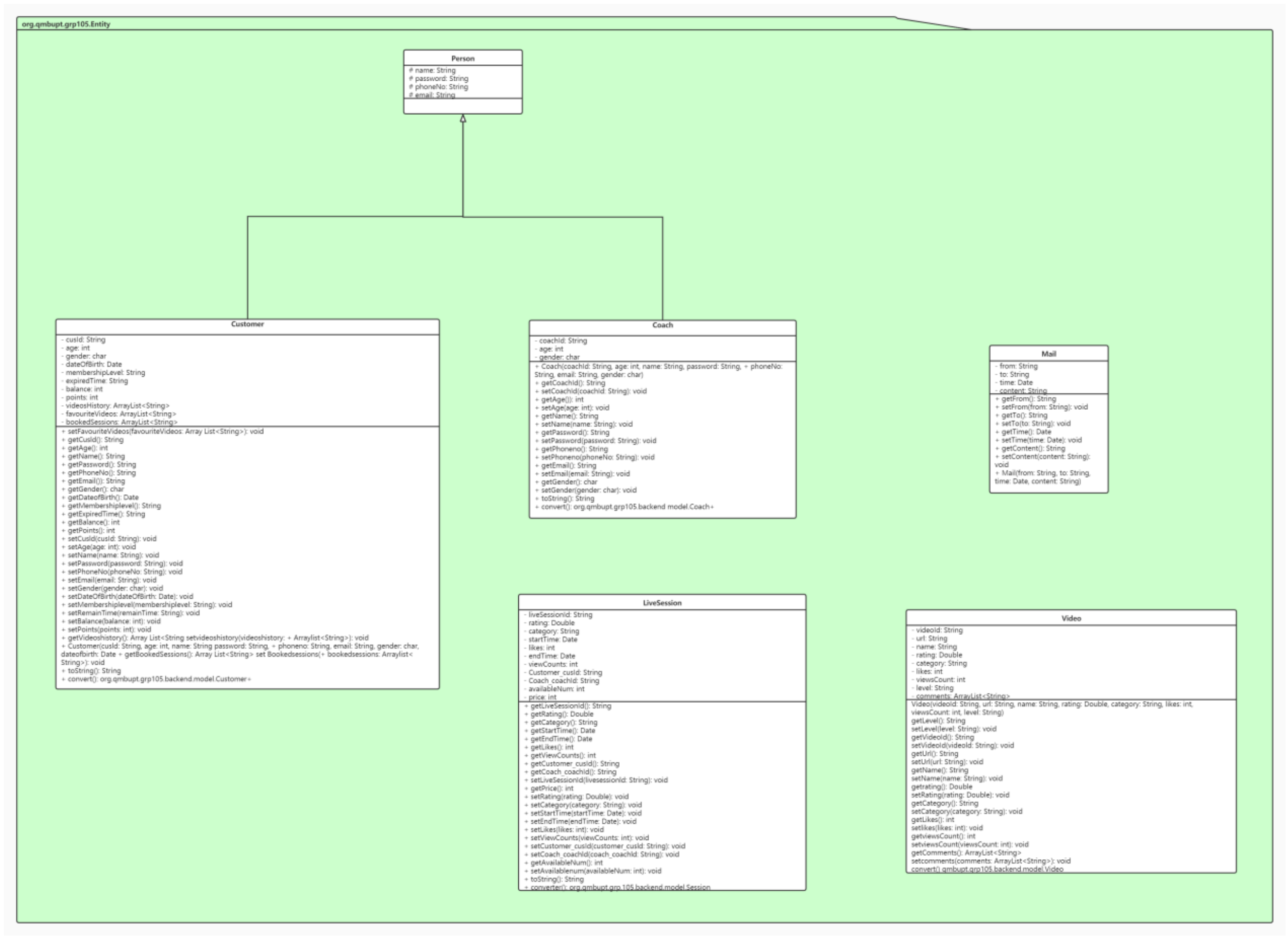
IV. Analysis

4.1 Identify Entity, Boundary, and Control Classes

We separated the source codes of the Java file into 5 parts- Entity, Boundary, Control, model, delayer classes.

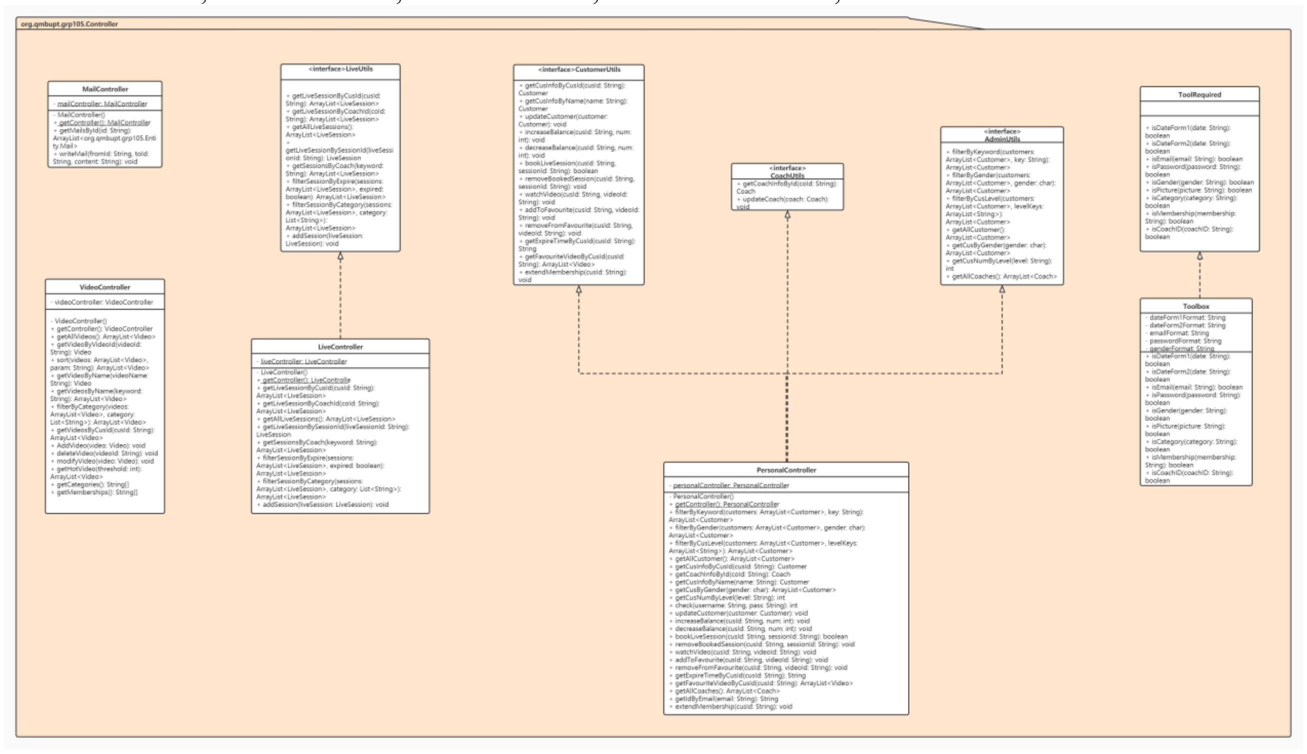
4.1.1 Entity Class

The code should follow the principle of high cohesion and loose coupling to increase the quality of code. In our system, entity classes include Customer, Coach, Video, Mail, LiveSession.



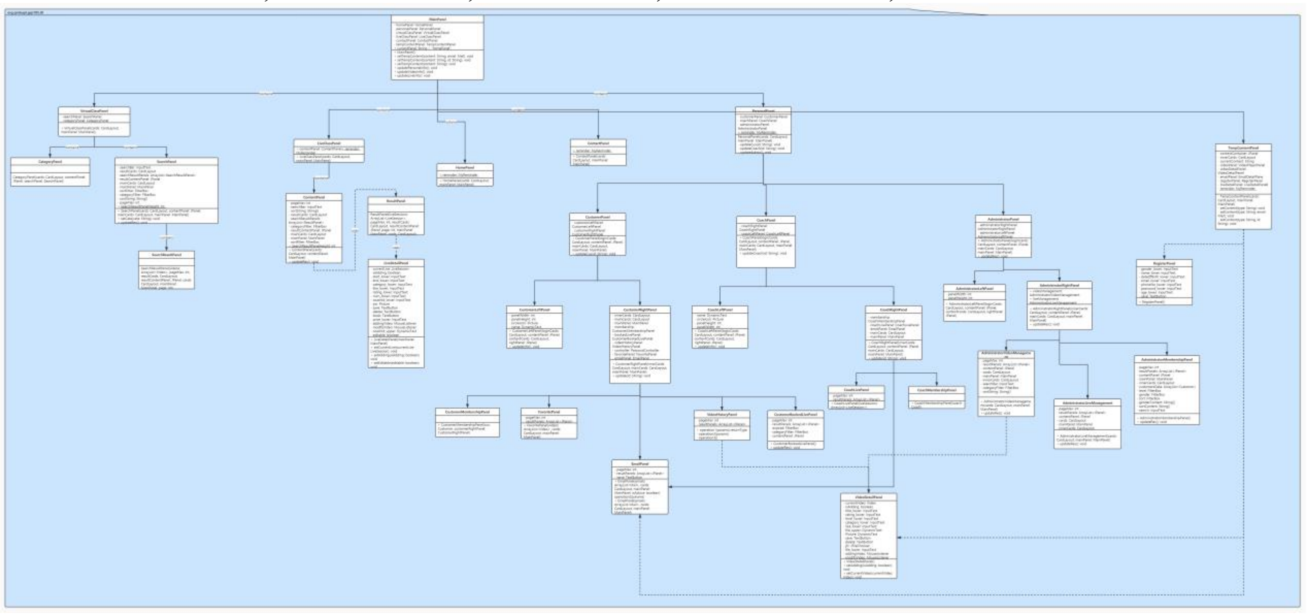
4.1.2 Control Class

We use the control class to link the Boundary class to the database and the Entity class. It is primarily responsible for processing data requests sent by the Boundary class and linking to the corresponding database based on the content of the request. After getting the data from the database, it parses the return packet and returns the specific data to the Boundary class. In our system, the control classes include Video Controller, Live Controller, Mail Controller, Personal Controller, etc.



4.1.3 Boundary Classes

The boundary class is responsible for presenting our UI to the users, listening to their actions, and interacting with them. The boundary class is at the top of our software architecture and only interacts with the user and control classes, making our software less coupled. In our system, the boundary classes include Home Panel, LiveClass Panel, Personal Panel, VirtualClass Panel, etc.



4.2 Conceptual Class Diagram

Having analyzed the relationships and associations between classes, we drew the conceptual diagram. **(Click here to view the whole Conceptual Class Diagram)**

4.3 Reusability

To make our code re-usable, we store our data in an appropriate data structure and we design our code following the design process. And we also override some important GUI components so that the layout can meet our expectations. For example, we create settings_config.json to store all categories and membership levels. Hence the code can easily adapt to changes, the system can add new categories and new membership levels by just modifying the JSON file. The main part of the code can be re-used when it comes to another online gym system.

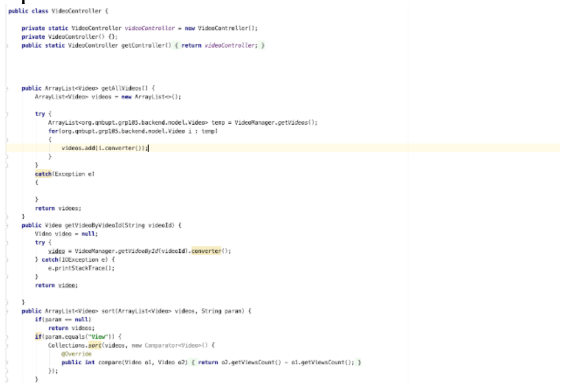
Besides, we designed some customized components for GUI to make the layout more interactive. And all these components can be reused whenever needed, **(Click here to view the UI component reuseability)**

V. Design Principles

- **SRP (Single Responsibility Principle)**

The Single Responsibility Principle is a guideline for achieving high cohesion and low coupling[4]. We discovered the different responsibilities of classes and separated them, so that one class is responsible for the corresponding responsibilities in only one functional area, giving high cohesion. The code inside a module will not be affected when other functions changed, giving loose coupling.

For example, we create a controller class called VideoController. VideoController consists of several operations which are relevant to the video. And this class is only responsible for operating videos.

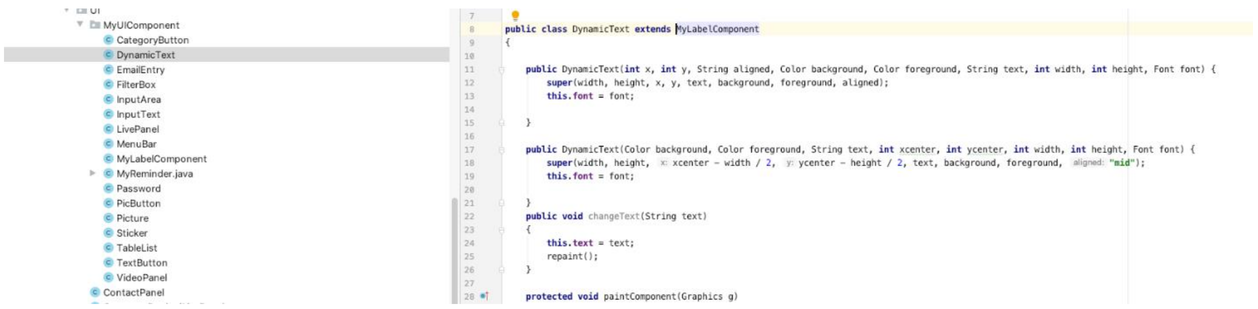


```java
public class VideoController {

    private static VideoController videoController = new VideoController();
    private VideoController() {};
    public static VideoController getController() { return videoController; }



    public ArrayList<Video> getAllVideos() {
        ArrayList<Video> videos = new ArrayList<>();

        try {
            ArrayList<org.qmbupt.grp105.backend.model.Video> temp = VideoManager.getVideos();
            for(org.qmbupt.grp105.backend.model.Video i : temp)
            {
                videos.add(i.converter());
            }
        }
        catch(Exception e)
        {

        }
        return videos;
    }
    public Video getVideoByVideoId(String videoId) {
        Video video = null;
        try {
            video = VideoManager.getVideoById(videoId).converter();
        } catch(IOException e) {
            e.printStackTrace();
        }
        return video;

    }
    public ArrayList<Video> sort(ArrayList<Video> videos, String param) {
        if(param == null)
            return videos;
        if(param.equals("View")) {
            Collections.sort(videos, new Comparator<Video>() {
                @Override
                public int compare(Video o1, Video o2) { return o2.getViewsCount() - o1.getViewsCount(); }
            });
        }
```

- **OCP (Open-Closed Principle)**

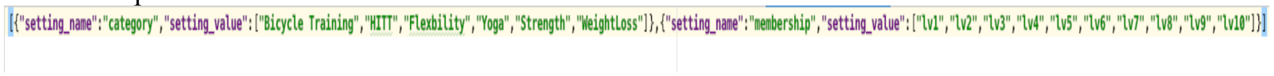
The Open-Closed Principle means that a software entity should be open for extension and closed for modification. That is, the software entity should extend as much as possible without modifying the original code.

For instance, we override many UI components to implements custom effects. We let all these UI components extend the class MyLabelComponent so that we can add some new styles without modifying the previous codes.



```java
public class DynamicText extends MyLabelComponent
{

    public DynamicText(int x, int y, String aligned, Color background, Color foreground, String text, int width, int height, Font font) {
        super(width, height, x, y, text, background, foreground, aligned);
        this.font = font;

    }

    public DynamicText(Color background, Color foreground, String text, int xcenter, int ycenter, int width, int height, Font font) {
        super(width, height,  x: xcenter - width / 2,  y: ycenter - height / 2, text, background, foreground,  aligned: "mid");
        this.font = font;

    }
    public void changeText(String text)
    {
        this.text = text;
        repaint();
    }

    protected void paintComponent(Graphics g)
```

Moreover, we create a config file called "setting_config.json" to implement the modification of category and membership in the system. That is, if we want to add some categories or membership levels, we only need to modify the setting.json file and the system will be updated with new categories and membership levels.

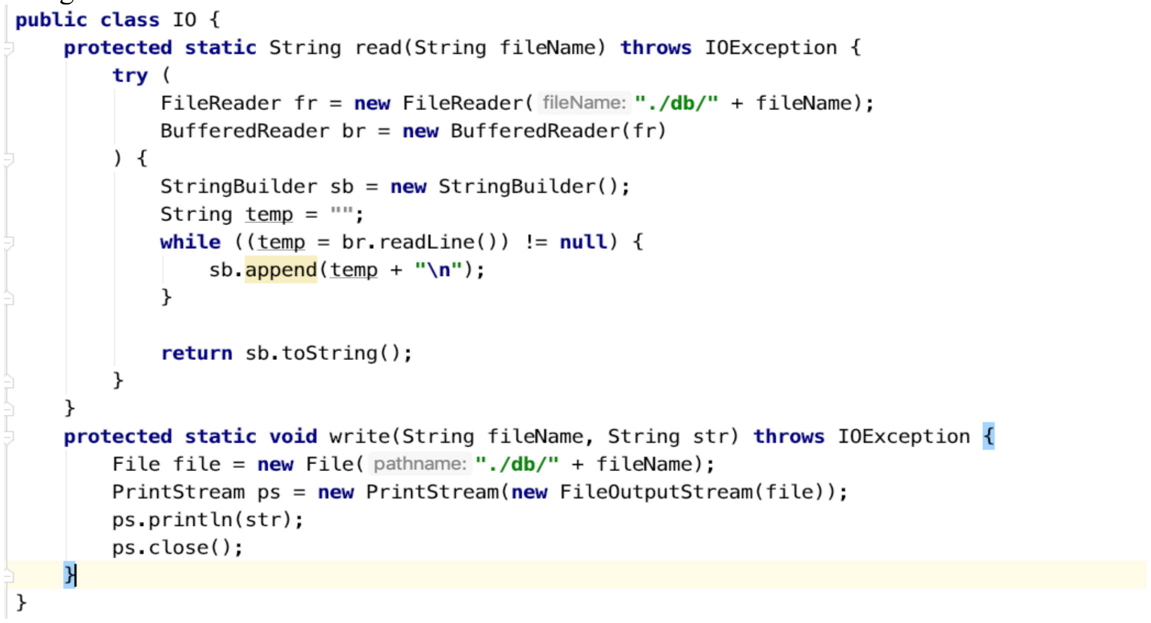


```
[{"setting_name":"category","setting_value":["Bicycle Training","HITT","Flexbility","Yoga","Strength","WeightLoss"]},{"setting_name":"membership","setting_value":["lv1","lv2","lv3","lv4","lv5","lv6","lv7","lv8","lv9","lv10"]}]
```

- **DRY (Don't repeat yourself)**

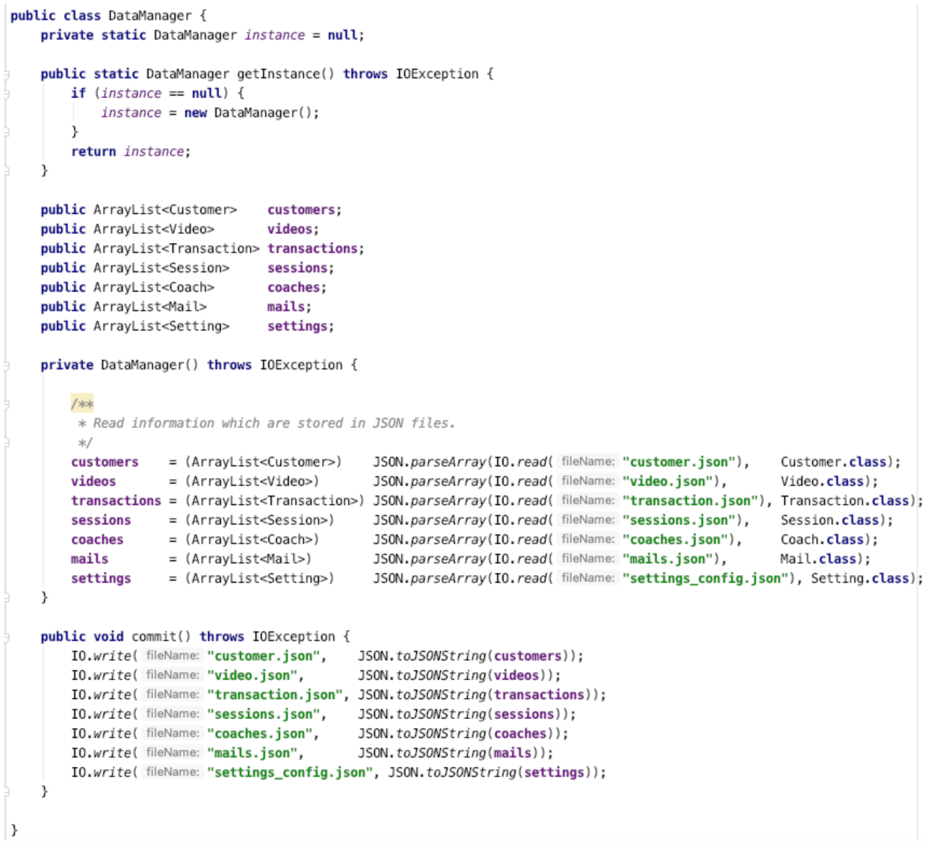
When we program, we try not to repeat our code to the greatest extent. Even though it's easy to cut and paste for where it is used each time, it's very a poor practice. If the programs have some bugs, it will be very hard to re-adjust the code if having repeated code. When some variables changed, we need to modify many places if we don't follow the don't DRY principles.

To deal with duplicated methods, we put them into a new class. We can invoke the methods whenever we need these methods. In this way, we can ensure that there is only one modification that needs to be done when there are any changes. For example, we frequently read files and write files, so we wrote an IO class containing the method of reading files and writing files. We invoke this method when we need it and we only need to adjust the code in the method correspondingly for once when the operations change in the future.



```java
public class IO {
    protected static String read(String fileName) throws IOException {
        try (
            FileReader fr = new FileReader( fileName: "./db/" + fileName);
            BufferedReader br = new BufferedReader(fr)
        ) {
            StringBuilder sb = new StringBuilder();
            String temp = "";
            while ((temp = br.readLine()) != null) {
                sb.append(temp + "\n");
            }

            return sb.toString();
        }
    }
    protected static void write(String fileName, String str) throws IOException {
        File file = new File( pathname: "./db/" + fileName);
        PrintStream ps = new PrintStream(new FileOutputStream(file));
        ps.println(str);
        ps.close();
    }
}
```

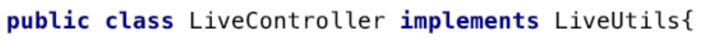
Plus, we also create a class called DataManager so that it can be invoked repeatedly to read and write an entity. And we only need to modify this class to adjust the changes.



```java
public class DataManager {
    private static DataManager instance = null;

    public static DataManager getInstance() throws IOException {
        if (instance == null) {
            instance = new DataManager();
        }
        return instance;
    }

    public ArrayList<Customer>    customers;
    public ArrayList<Video>       videos;
    public ArrayList<Transaction> transactions;
    public ArrayList<Session>     sessions;
    public ArrayList<Coach>       coaches;
    public ArrayList<Mail>        mails;
    public ArrayList<Setting>     settings;

    private DataManager() throws IOException {

        /**
         * Read information which are stored in JSON files.
         */
        customers    = (ArrayList<Customer>)    JSON.parseArray(IO.read( fileName: "customer.json"),    Customer.class);
        videos       = (ArrayList<Video>)       JSON.parseArray(IO.read( fileName: "video.json"),       Video.class);
        transactions = (ArrayList<Transaction>) JSON.parseArray(IO.read( fileName: "transaction.json"), Transaction.class);
        sessions     = (ArrayList<Session>)     JSON.parseArray(IO.read( fileName: "sessions.json"),    Session.class);
        coaches      = (ArrayList<Coach>)       JSON.parseArray(IO.read( fileName: "coaches.json"),     Coach.class);
        mails        = (ArrayList<Mail>)        JSON.parseArray(IO.read( fileName: "mails.json"),       Mail.class);
        settings     = (ArrayList<Setting>)     JSON.parseArray(IO.read( fileName: "settings_config.json"), Setting.class);
    }

    public void commit() throws IOException {
        IO.write( fileName: "customer.json",    JSON.toJSONString(customers));
        IO.write( fileName: "video.json",       JSON.toJSONString(videos));
        IO.write( fileName: "transaction.json", JSON.toJSONString(transactions));
        IO.write( fileName: "sessions.json",    JSON.toJSONString(sessions));
        IO.write( fileName: "coaches.json",     JSON.toJSONString(coaches));
        IO.write( fileName: "mails.json",       JSON.toJSONString(mails));
        IO.write( fileName: "settings_config.json", JSON.toJSONString(settings));
    }

}
```

- **DIP (Dependency Inversion Principle)**

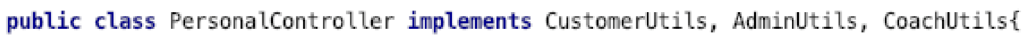
In DIP, both high-level modules and low-level modules should depend on abstractions and details should depend on abstractions. In order to follow this principle, we try to use the interface. For example, we create an interface called LiveUtils which contains all methods which are relevant to live sessions. And we let class LiveController implement this interface.



```java
public class LiveController implements LiveUtils{
```

- **ISP (Interface Segregation Principle)**

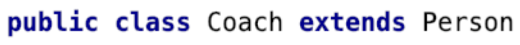
The interface segregation principle means a certain type of interface should only have a small number of methods in it so that clients are not forces to depend on the methods they do not use. During our design, we separate our interface as small as possible, we only put methods in one interface when they are highly relative. For example, the Person can be divided into three categories: customer, coach, and administrator. We create three interfaces for these categories and let the PersonalController class implement these three interfaces. Although three interfaces are invoked by the same class, we put them separately so that customers can choose the function they want and override the method.



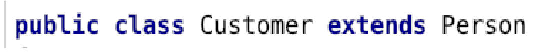
```java
public class PersonalController implements CustomerUtils, AdminUtils, CoachUtils{
```

- **LSP (Liskov Substitution Principle)**

In LSP, replacing an object of the base class with an object of its subclass in the software will not generate any errors and exceptions. The reverse is not true. If a software entity uses a subclass object, it may not be able to use the base class object. Hence, we define objects by using the base classes as much as possible and subclass types are determined at run time to replace superclass objects with subclass objects. For example, we define a superclass called Person and let all Customer, Coach to inherit it.



```java
public class Coach extends Person
```



```java
public class Customer extends Person
```

VI. Implementation

6.1 Configuration Management

We use Git for software version control and use Github as a remote server to implement parallel development. As shown in the screenshot of the appendix, (Click here to view the github commit log) it is the basic project work we did during the first iteration cycle, such as API interface implementation, Unit Test Implementation, UI component implementation, etc.

In software development, it is necessary to control the software version and requirement change management[4]. There are two main reasons: 1. We need to allow different team members to develop in parallel without interfering with each other. Never let one person's work cover another person's work, this has a disastrous impact on the development progress of the entire project, so the branch of git and its conflict detection features to support our needs. 2. Sometimes integration or adding new features may cause the system to crash, so we need to have the function of version rollback. The git reset command can meet our needs.

As shown in the figure, our Git working method, Main Branch as the main branch is a branch composed of versions that can be run. When a team member needs to add a new Feature A to the current version, he creates a Feature A development branch and then merges to the main branch after the branch test is completed, and deletes the Feature A branch. **(Click here to view the github commit log)**
Our software complies with Apache GNU General Public License v3.0. After the deadline for submission of this assignment, we will open-source it, allowing Commercial Use, Modification, Distribution, Patent use, and Private use.

6.2 Refactor

During the development process, we constantly find that there are new problems in the software architecture. This is not because there are problems with the design we started, but during the development process, there may be changes in requirements, which leads to the start of the design. The architecture is no longer suitable for new requirements.

Therefore, in the development process, once we find that the software code is becoming bloated and non-reusable, we use refactor to simplify it. Although refactor may slow down the progress of the project, we still strictly implement refactor to keep the entire project in a good structure.

As shown in the figure below, there are some commit records for refactoring during our development. **(Click here to view the refactor history)**

6.3 Pair Programming

Because we are using agile development, a lot of documents are omitted after the development is completed, and there is no code review like in traditional development. In order to make up for this shortcoming, we adopted the practice of Pair Programming when developing some complex components. The advantage of this is that, first of all, this method can reduce the probability of code errors, because during pair programming, the programmer may not find the problem from his own perspective, but other people can find the problem from another perspective. Secondly, once a member is sick or unable to continue development for other reasons, other members can also take over his work.

6.4 Implementation Strategy

The Implementation strategy we adopted is divided into two parts. The first is to carry out iterative development, each iteration produces a runnable application. And test on this basis, and release software that can be used by customers after passing the alpha test. The second is that we divide the system into multiple components. Through the pre-set Junit Test, after the component passes the test, different components can be integrated into executable files.

The advantage of this is that an entire software development process can be turned into multiple Small, manageable steps, thereby making the entire software development parallel. On the other hand, it also allows the project manager (group leader) to check the progress of the project even if it is to ensure that it is carried out as scheduled.

6.5 Integration of Build Plan

Build Plan	Functionality	Implementation
1	The interface between UI and controller classes	interface AdminUtils, interface CoachUtils, interface CustomerUtils, interface LiveUtils, interface ToolRequired
	Basic UI Component	class CategoryButton, class DynamicText, class EmailEntry, class FilterBox, class InputArea, class InputText, class MenuBar, class MyLabelComponent, class MyReminder, class Password, class PicButton,

		class Picture, class Sticker, class TableList, class TextButton, class VideoPanel
	Basic IO read & write	class IO, String read (String fileName), void write (String fileName, String str)
2	Basic UI Pages	class MainPanel, class PersonalPanel, class LiveClassPanel, class HomePanel, class ContactPanel, class VirtualClassPanel
	Entity Implementation	class Person, class Coach, class Customer, class Mail, class LiveSession, class Video
	Model Implementation and Json field design	class Coach, class Customer, class Mail, class Session, class Setting, class Transaction, class Video
3	Personal UI Page detail implementation	class AdministratorLeftPanel, class AdministratorLiveManagement, class AdministratorMembershipPanel, class AdministratorPanel, class AdministratorRightPanel, class AdministratorVideoManagement, class CoachLeftPanel, class CoachLivePanel, class CoachMembershipPanel, class CoachPanel, class CoachRightPanel, class CustomerLeftPanel, class CustomerMembershipPanel, class CustomerPanel, class CustomerRightPanel, class EmailPanel, class FavoritePanel, class PersonalPanel, class SignInPanel, class VideoHistoryPanel

VII. Testing

7.1 Testing Strategy

(1) The way we implement our test.

Half of the tests will be automated including unit testing and system testing[5]. The other half of testing will be manual which is mainly responsible for testing the GUI interface interactions. Besides, we make sure that the user interface (GUI) looks correctly during the testing. Last but not least, we try our best to make our software handle correctly under all kinds of users' incorrect input.

(2) The way to run testing.

For unit tests based on system components, such as individual interfaces or Control classes, we use regression and white-box testing to test the internal program logic. In the aspect of system testing which is based on the whole system, the black box testing method of partition testing and scenario testing is adopted to test the software requirements.

(3) The time to do the testing.

Unit testing is performed during the development of the classes, once a class is programmed, the unit testing will be done. As for system testing, it was done after each iteration of our project was completed.

(4) Testing criteria

The criteria are that all the components can pass the unit testing so that we enable to discover and correct errors as early as possible to build more robust projects. Also, checking whether we have implemented all requirements, including both functional and non-functional requirements, is the criteria for testing.

7.2 Testing Techniques

1) White-box testing

We use White-box Testing to test the internal program logic. we use a basis path testing method to test core functions that consist of multiple components. **(Click here to view Basis Path Test diagram)**

2) Black-box testing

1. Partition testing

Partition testing is used when practically it is impossible to test a large pool of test cases individually[5]. Since our online gym is a highly interactive system where users can enter and query a large amount of information, we used a partitioned testing method to verify that the input validation was robust enough.

2. Scenario testing

It's a tough job to test all the scenarios. So, we divided scenario testing into manual testing and automated testing. For automated testing, we assume some scenarios and write testing code using

TDD based on these scenarios. For manual testing, we just assume ourselves as different users to test the software. We also invite so people who are not in our Agile team to test the code.

In our scenarios testing, we have tested the following scenarios: When a user enters our system, he/she may directly view the classes video or book the live session without login, he/she may try to login to the system directly without registering. During the registration or login, the user may input invalid content or entering the wrong account number or password. After login, the user wants to watch the class video that is not available to her current membership level. During the process of watching the video and live, the user may want to add a video into or remove a video from their favorite list, he/she may want to like or dislike a video, he/she may want to subscribe or unsubscribe from a live session.

If the user is a manager, he/she may want to delete a video that contains wrong information, or he/she may want to modify the title of the video, or he/she may want to add a new video into the system.

We tested all scenarios mentioned above. **(Click here to view Scenarios test code)**

3. Regression testing

We use regression testing during each stage of iterations to see whether the functions perform correctly if having incremental development. For example, when we finish developing the logic function of "consumer register and login" in the early stage, we need to further test this function after adding the GUI classes. **(Click here to view Regression test example diagram)**

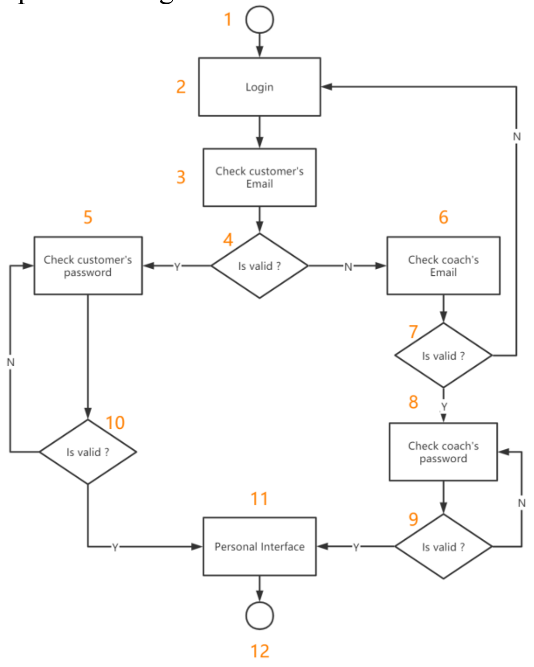
7.3 Test case Design

We carried out the basis path test method for every aspect of our system, and analyzed each test result, and produced the corresponding test matrix. Due to the limitations of the report, we now show one demonstration as follows:

1) Code under test

(Click here to view the code for white box test)

2) Basis path test diagram.



3) Cyclomatic complexity calculation

As can be seen from the figure, there are four simple decisions in the figure, so the complexity of the loops is five.

Cyclomatic Complexity = 5

4) Basic path set design

The basic path set is:

Path A: 1, 2, 3, 4, 5, 10, 11, 12

Path B: 1, 2, 3, 4, 6, 7, 8, 9, 11, 12

Path C: 1, 2, 3, 4, 5, 10, 5, 10, 11, 12

Path D: 1, 2, 3, 4, 6, 7, 2, 3, 4, 6, 7, 8, 9, 11, 12

Path E: 1, 2, 3, 4, 6, 7, 8, 9, 8, 9, 11, 12

5) Test case set design

The test case set is:

- Path A

  Input: andrecruz@gmail.com, Andre757

  Expected output: Welcome, customer!
- Path B

  Input: matt_morsia@hotmail.com, Matt0119

  Expected output: Welcome, Coach!
- Path C

  Input: andrecruz@gmail.com, Andre787, Andre757

  Expected output: Password Wrong, Welcome, Costumer!
- Path D

  Input: maTT_morsia@hotmail.com, matt_morsia@hotmail.com, Matt0119

  Expected output: Email doesn't exist, Welcome, Coach!
- Path E

  Input: matt_morsia@hotmail.com, Matt9119, Matt0119

  Expected output: Password Wrong, Welcome, Coach!

6) Test result

(Click here to view Test Result)

7.4 Test Procedure with Test Matrix

After each test, we will produce a Test Matrix. Now, due to the limitations of the report, we only show the Test Matrix of login and registration as an example.

- Test Matrix for User to log in

TestCase	#Test Description	Cases	P/F	Bug.No	Bug#	Comments
N/A	User login	Setup	--	--	--	Added to system for testing
1.1	Test the correct input	1.1	P	0	0	See the correct result
1.2	Test the input cannot be empty	1.2	P	0	0	See the correct result
1.3	Test the Email address is not exist	1.3	P	0	0	See the correct result
1.4	Test the user password is wrong	1.4	P	0	0	See the correct result

- Test Matrix for User to register

TestCase	#Test Description	Cases	P/F	Bug.No	Bug#	Comments
2.1	Test the correct input	2.1	P	0	0	See the correct result
2.2	Test the input cannot be empty	2.2	P	0	0	See the correct result
2.3	Test the invalid input for date of birth	2.3	P	0	0	See the correct result
2.4	Test the invalid input for gender	2.4	P	0	0	See the correct result
2.5	Test the invalid input for PhoneNo	2.5	P	0	0	See the correct result
2.6	Test the invalid input for Email	2.6	P	0	0	See the correct result
2.7	Test the invalid input for password	2.7	P	0	0	See the correct result

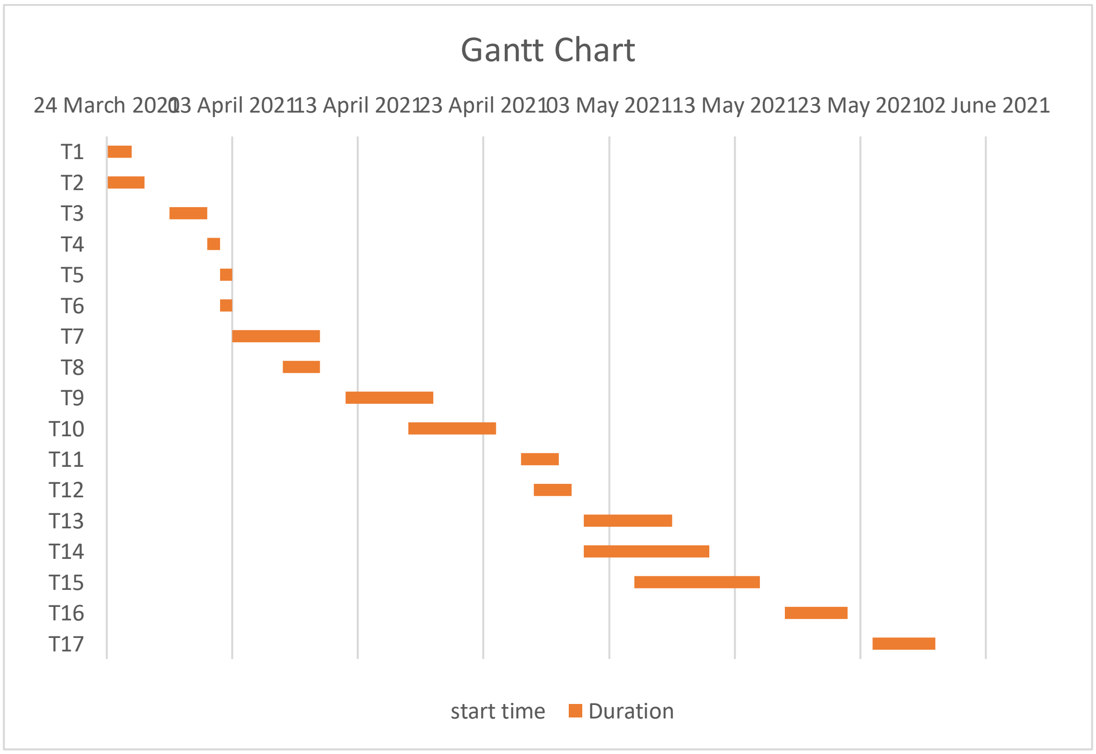
VIII. Conclusion

Our group used Agile Methods and completed our Digital Gym System as well as relative report and video. Through this project, we have experienced a lot and learned a lot. We worked together and formed an agile team to develop the system which let us have a better understanding of Agile Methods. During the process, although many problems occurred, we overcame them together and achieved success. It's really an unforgettable experience for us.

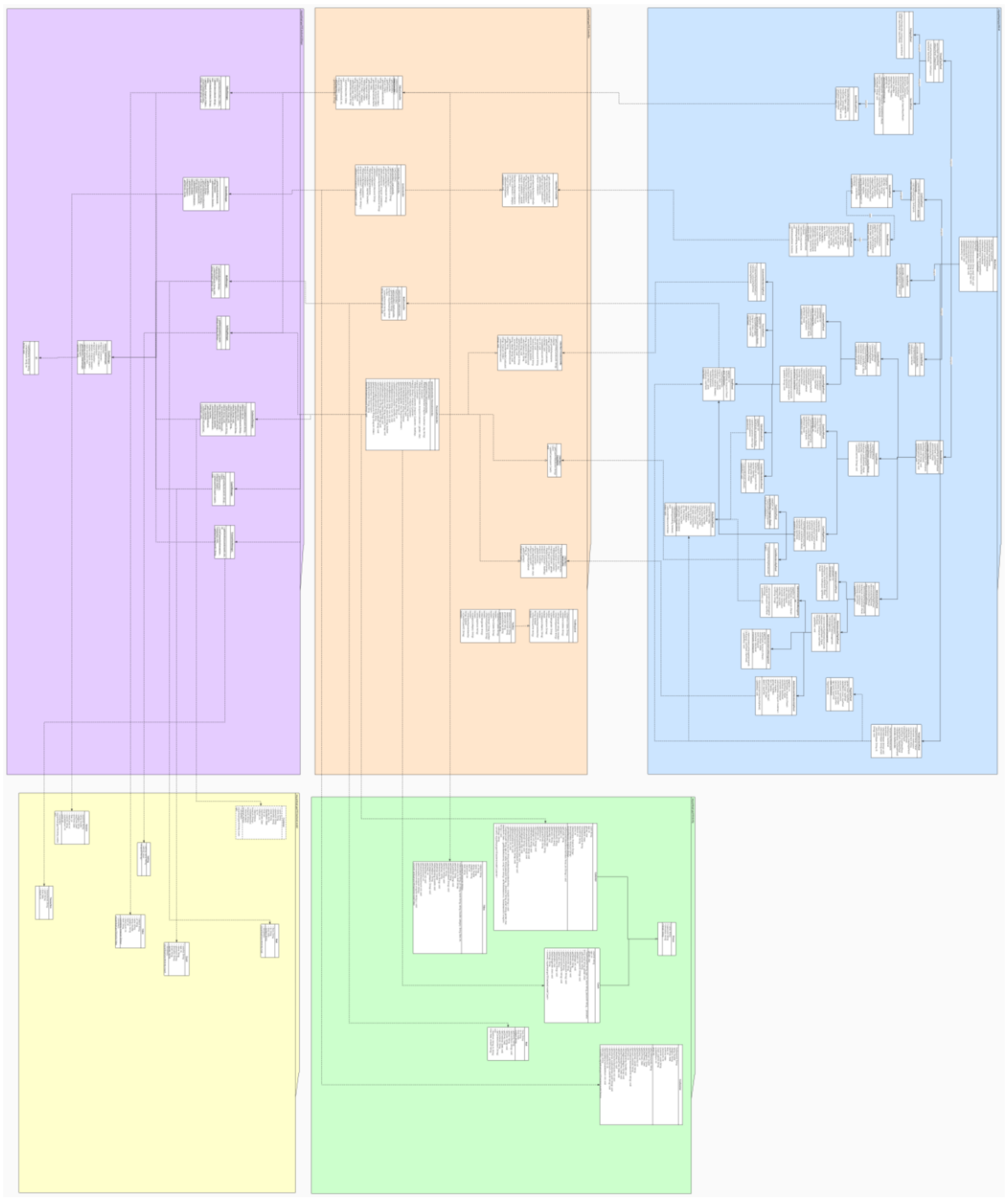
Appendix

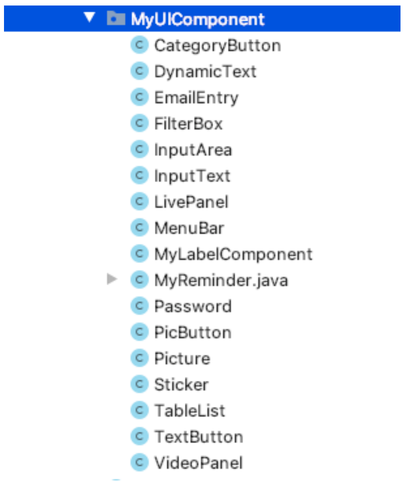
Task	Description	Duration	Dependency
T1	Requirement gathering	2	
T2	Prototype design	3	T1(M1)
T3	Software architecture design	3	T1(M1)
T4	Class design	1	T3(M2)
T5	Interface design	1	T4(M3)
T6	Implementation of entity	1	T4(M3)
T7	Implementation of basic UI component	7	T2(M4)
T8	Implementation of Layout and route function	3	T2(M4)
T9	Implementation of five basic pages	7	T7(M5)
T10	Implementation of the detail of every page	7	T9(M6)
T11	Implementation of the video play	3	T7(M5)
T12	Implementation of json File IO	3	T5(M7)
T13	Implementation of json array encapsulation and read & write	7	T12(M8)
T14	Implementation of practical function of every model	10	T13M9)
T15	Implementation of controllers based on the interface designed before	10	T5(M7)
T16	Implementation of stub function in controller for frontend to test	5	T15(M10)
T17	Implementation of the communication between con-trollers and backend	5	T15,T9,T13(M11)

Task Table

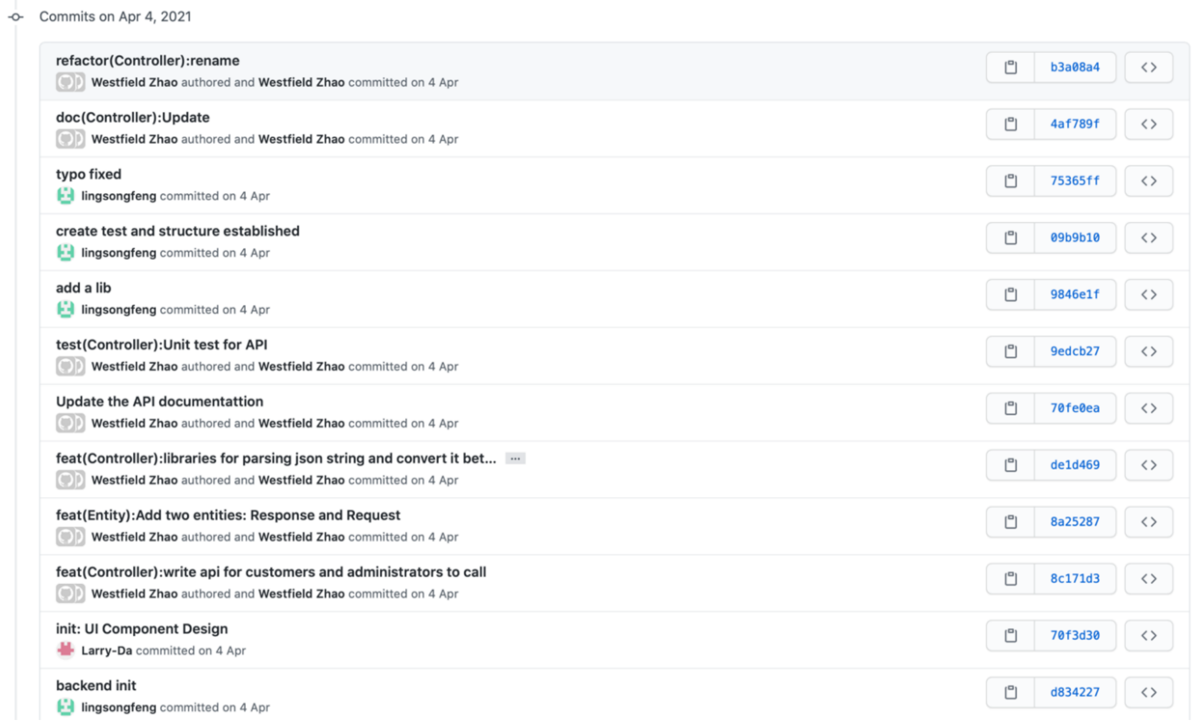


Gantt Chart


Conceptual Class Diagram


▼ MyUIComponent
- CategoryButton
- DynamicText
- EmailEntry
- FilterBox
- InputArea
- InputText
- LivePanel
- MenuBar
- MyLabelComponent
- ▶ MyReminder.java
- Password
- PicButton
- Picture
- Sticker
- TableList
- TextButton
- VideoPanel

UI Component we designed


Commits on Apr 4, 2021

refactor(Controller):rename
Westfield Zhao authored and Westfield Zhao committed on 4 Apr — b3a08a4

doc(Controller):Update
Westfield Zhao authored and Westfield Zhao committed on 4 Apr — 4af789f

typo fixed
lingsongfeng committed on 4 Apr — 75365ff

create test and structure established
lingsongfeng committed on 4 Apr — 09b9b10

add a lib
lingsongfeng committed on 4 Apr — 9846e1f

test(Controller):Unit test for API
Westfield Zhao authored and Westfield Zhao committed on 4 Apr — 9edcb27

Update the API documentattion
Westfield Zhao authored and Westfield Zhao committed on 4 Apr — 70fe0ea

feat(Controller):libraries for parsing json string and convert it bet... ...
Westfield Zhao authored and Westfield Zhao committed on 4 Apr — de1d469

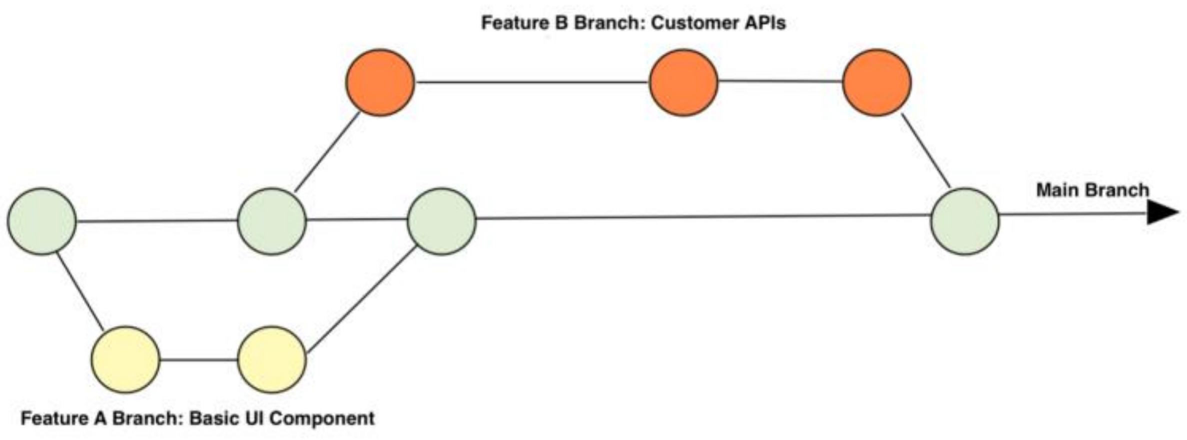
feat(Entity):Add two entities: Response and Request
Westfield Zhao authored and Westfield Zhao committed on 4 Apr — 8a25287

feat(Controller):write api for customers and administrators to call
Westfield Zhao authored and Westfield Zhao committed on 4 Apr — 8c171d3

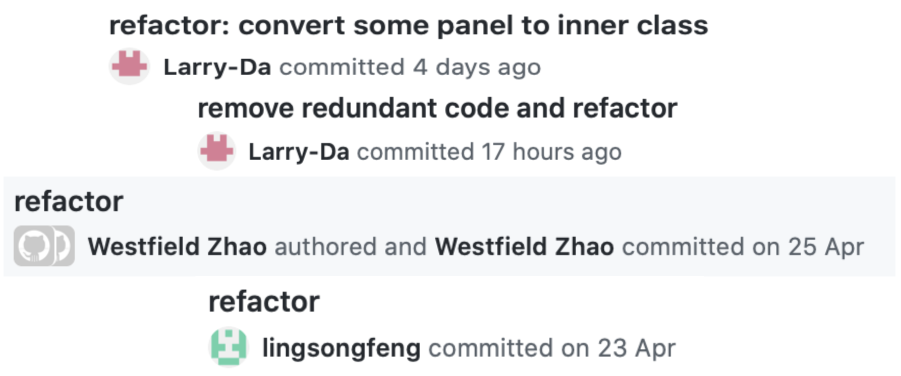
init: UI Component Design
Larry-Da committed on 4 Apr — 70f3d30

backend init
lingsongfeng committed on 4 Apr — d834227

Github Commit Log


Feature B Branch: Customer APIs
Main Branch
Feature A Branch: Basic UI Component

Github branch management


refactor: convert some panel to inner class
Larry-Da committed 4 days ago

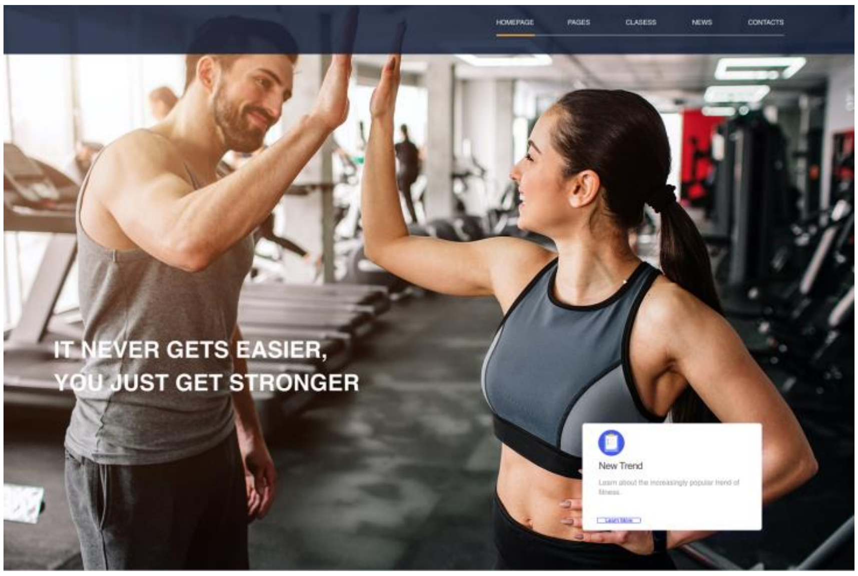
remove redundant code and refactor
Larry-Da committed 17 hours ago

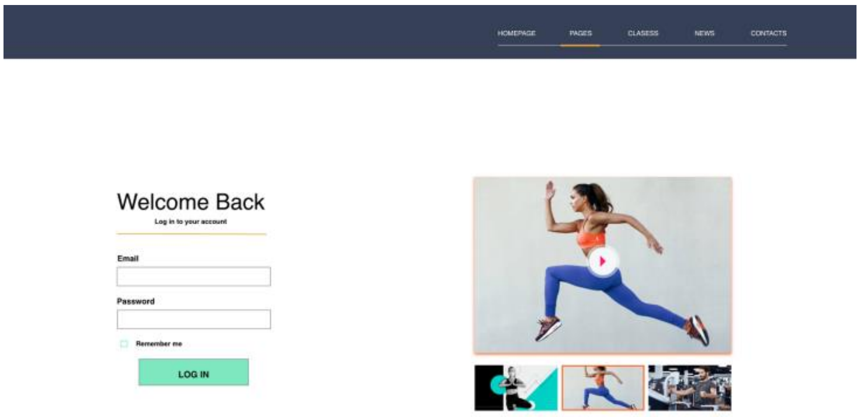
refactor
Westfield Zhao authored and Westfield Zhao committed on 25 Apr

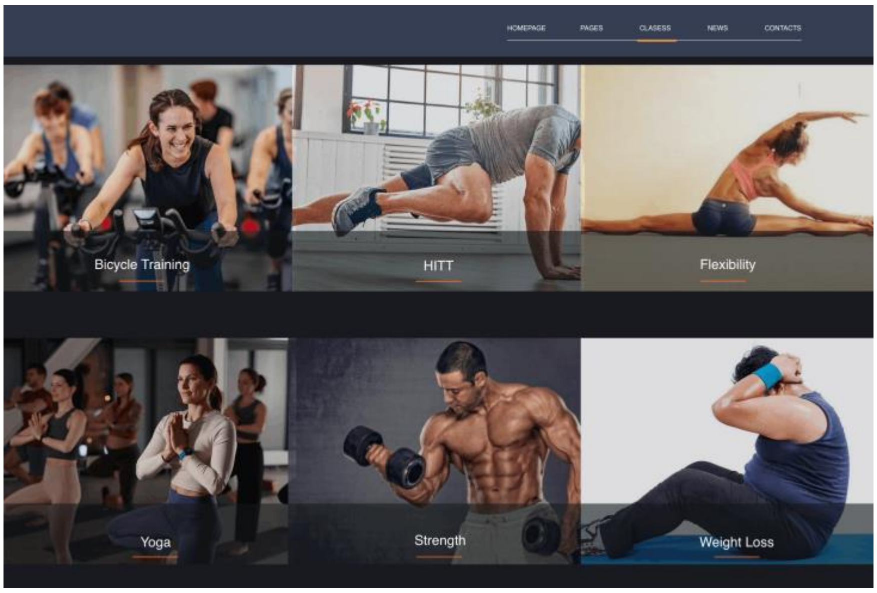
refactor
lingsongfeng committed on 23 Apr

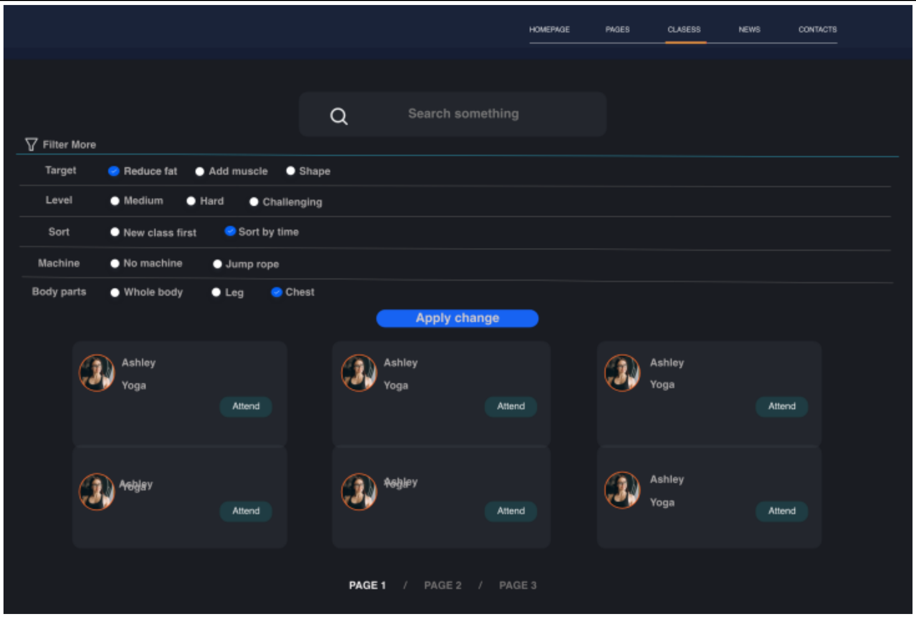
Refactor history

Prototype

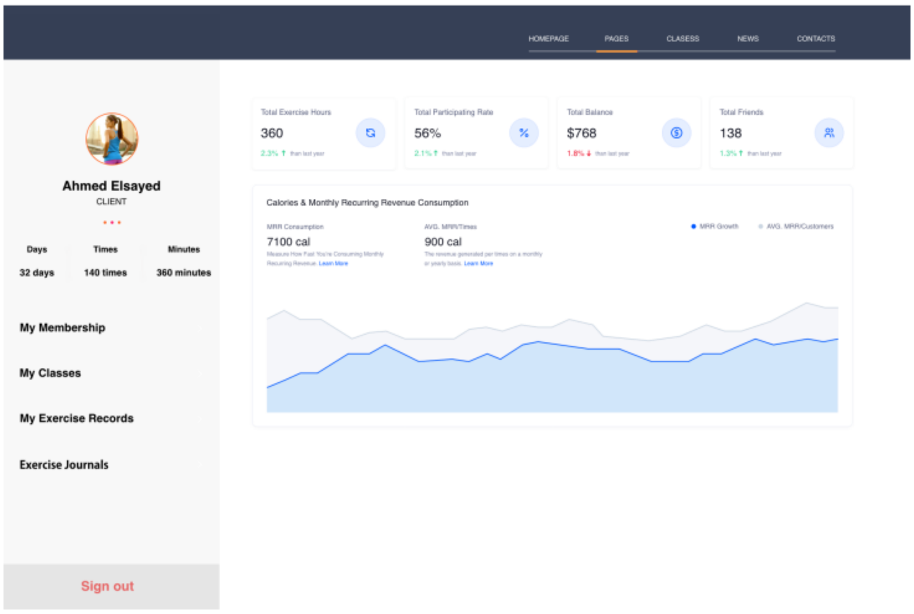

HOMEPAGE
PAGES
CLASESS
NEWS
CONTACTS
IT NEVER GETS EASIER,
YOU JUST GET STRONGER
New Trend
Learn about the increasingly popular trend of fitness.
Learn More


HOMEPAGE
PAGES
CLASESS
NEWS
CONTACTS
Welcome Back
Log in to your account
Email
Password
Remember me
LOG IN


HOMEPAGE
PAGES
CLASESS
NEWS
CONTACTS
Bicycle Training
HITT
Flexibility
Yoga
Strength
Weight Loss



Target	Reduce fat	Add muscle	Shape
Level	Medium	Hard	Challenging
Sort	New class first	Sort by time	
Machine	No machine	Jump rope	
Body parts	Whole body	Leg	Chest



Ahmed Elsayed
CLIENT

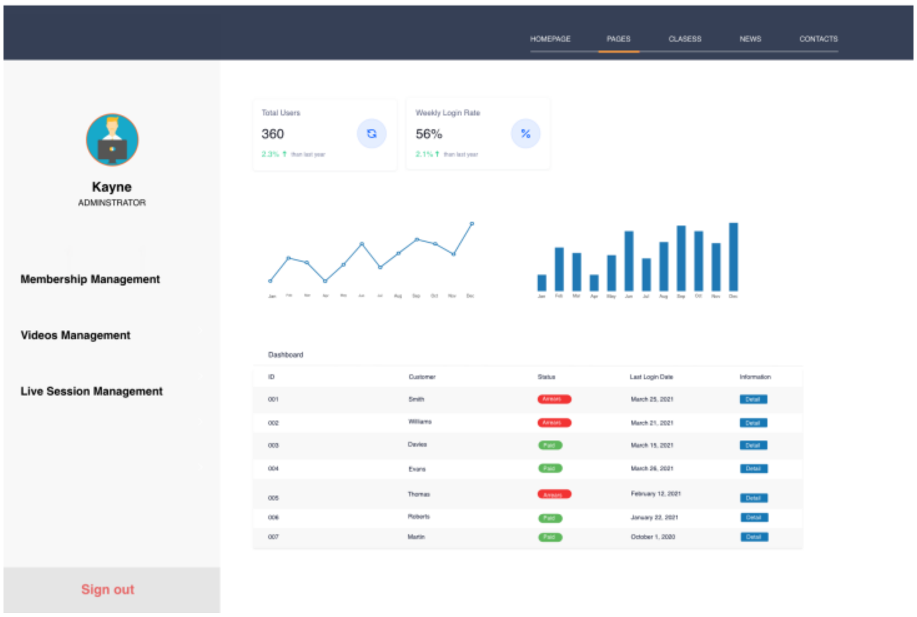
Days	Times	Minutes
32 days	140 times	360 minutes

Total Exercise Hours	Total Participating Rate	Total Balance	Total Friends
360	56%	$768	138
2.3% ↑ than last year	2.1% ↑ than last year	1.8% ↓ than last year	1.3% ↑ than last year

Calories & Monthly Recurring Revenue Consumption

MRR Consumption
7100 cal
Measure How Fast You're Consuming Monthly Recurring Revenue. Learn More

AVG. MRR/Times
900 cal
The revenue generated per times on a monthly or yearly basis. Learn More



Kayne
ADMINSTRATOR

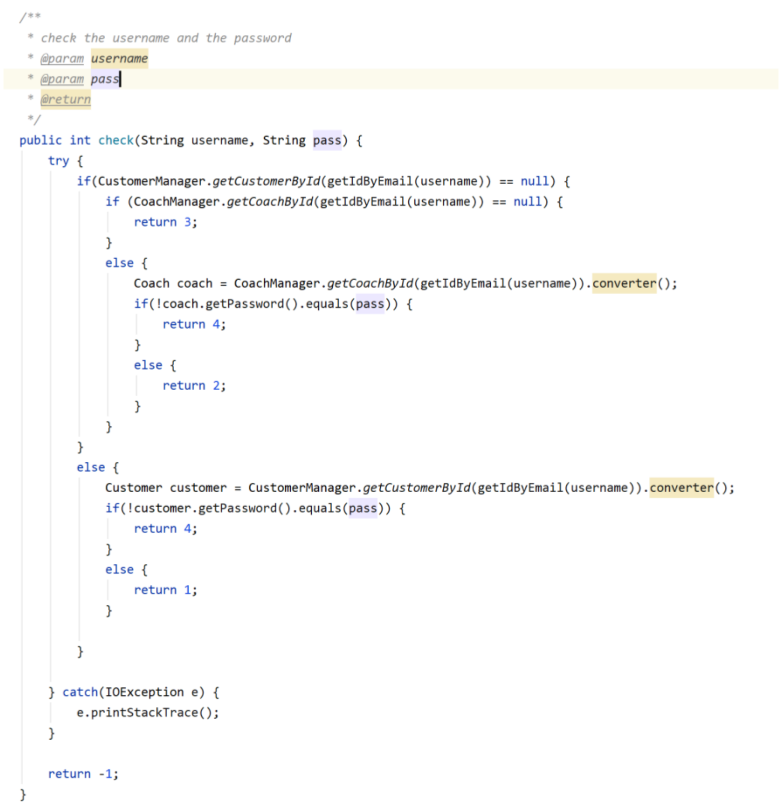
Total Users	Weekly Login Rate
360	56%
2.3% ↑ than last year	2.1% ↑ than last year

Dashboard

ID	Customer	Status	Last Login Date	Information
001	Smith	Arrears	March 25, 2021	Detail
002	Williams	Arrears	March 21, 2021	Detail
003	Davies	Paid	March 15, 2021	Detail
004	Evans	Paid	March 26, 2021	Detail
005	Thomas	Arrears	February 12, 2021	Detail
006	Roberts	Paid	January 22, 2021	Detail
007	Martin	Paid	October 1, 2020	Detail

Term	Abbreviation	Definition
Customer	Cus	People who use the system to watch videos or book live sessions
Administrator	Admin	People who are responsible for backend management including user management and class management and so forth
Category	——	Category refers to different kinds of training video such as YOGA, HITT
Likes	——	It means that the customer likes this video and send this message mixed up with others' likes to the video producer
Premier	——	A kind of membership provided by the system requires customers to spend an amount of money to upgrade
Class list	——	A list that contains a specific customer's classes, including virtual classes and live session
Membership rewards points	——	A kind of virtual currency that can be used in this software. It can be used to replace money or upgrade membership level
Membership levels	——	A level that reflects membership and its high or low status. The higher the level, the more active the user is in the software or the more amount of recharge

Glossary

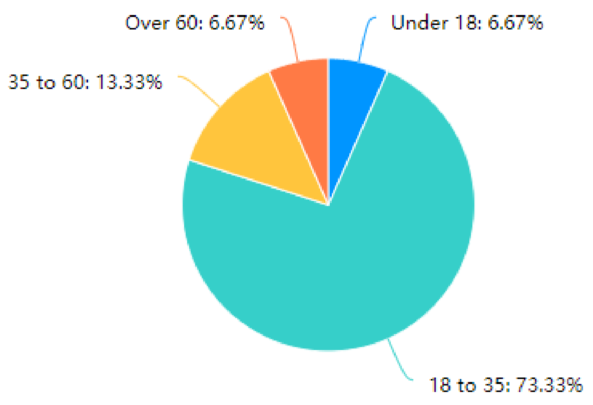


```java
/**
 * check the username and the password
 * @param username
 * @param pass
 * @return
 */
public int check(String username, String pass) {
    try {
        if(CustomerManager.getCustomerById(getIdByEmail(username)) == null) {
            if (CoachManager.getCoachById(getIdByEmail(username)) == null) {
                return 3;
            }
            else {
                Coach coach = CoachManager.getCoachById(getIdByEmail(username)).converter();
                if(!coach.getPassword().equals(pass)) {
                    return 4;
                }
                else {
                    return 2;
                }
            }
        }
        else {
            Customer customer = CustomerManager.getCustomerById(getIdByEmail(username)).converter();
            if(!customer.getPassword().equals(pass)) {
                return 4;
            }
            else {
                return 1;
            }

        }

    } catch(IOException e) {
        e.printStackTrace();
    }

    return -1;
}
```

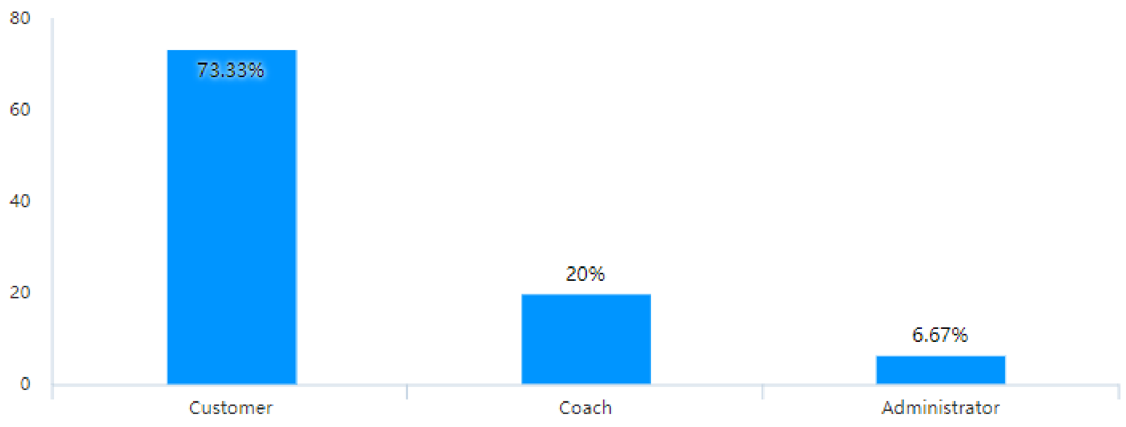
Code for white box test

Questionnaire and Result

1. What is the age of you? [Multiple choice]

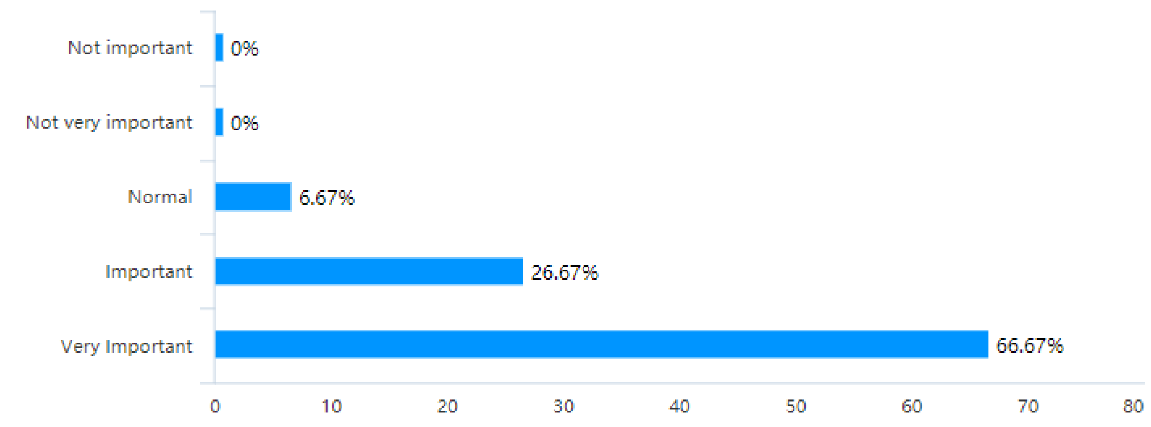


2. What is your identity for the online gym system? [Multiple choice]

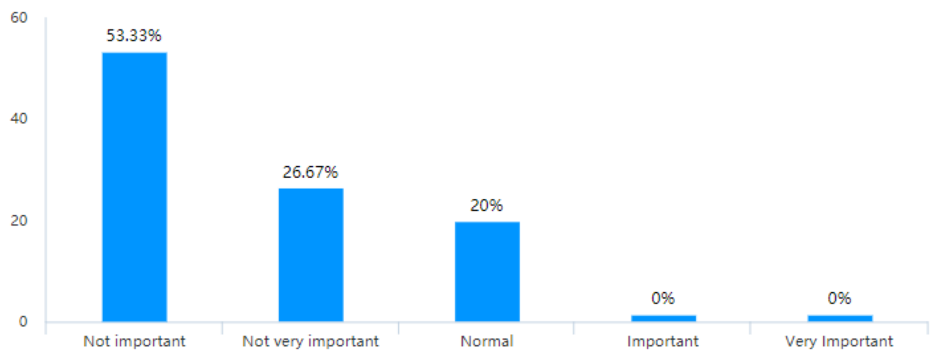


For each of the following features, please indicate on a scale from 1 to 5 to show how important it is for you when it comes to an online gym system.

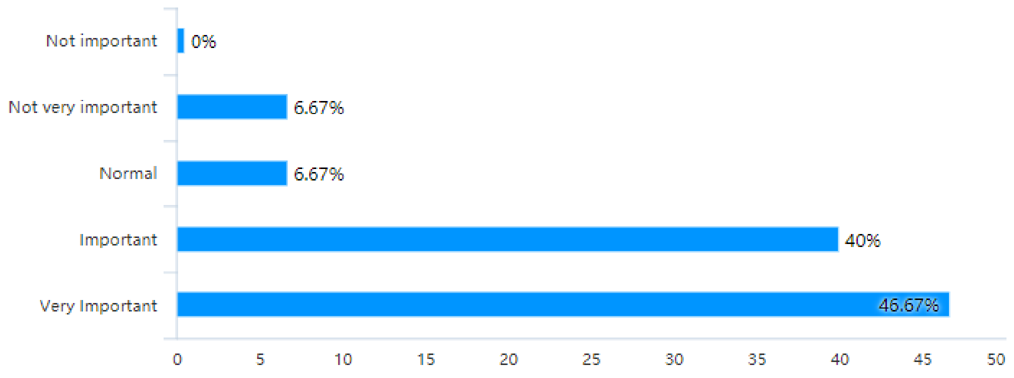
3. The software has a beautiful and understandable UI. [Multiple choice]



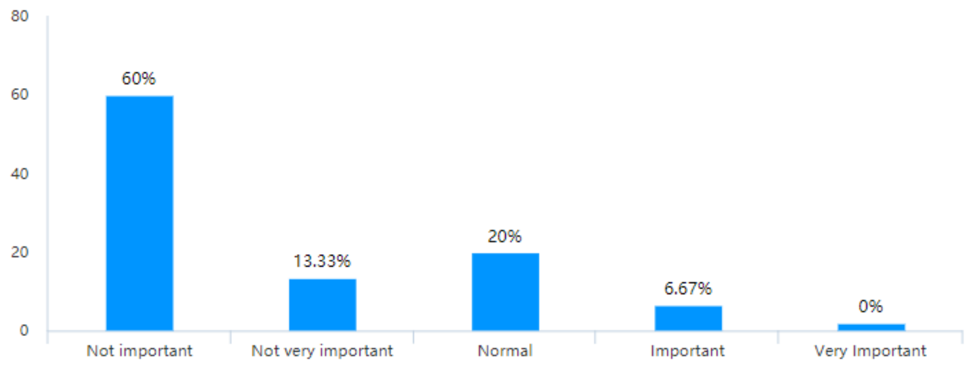
4.The information of coaches is detailed. [Multiple choice]



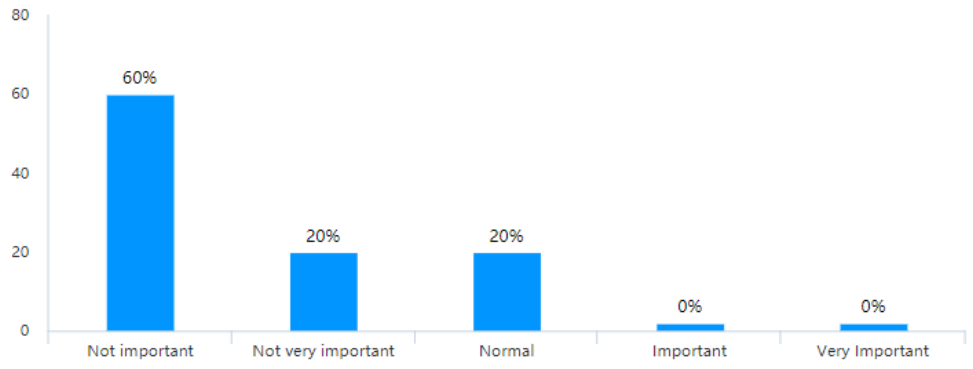
5.It is simple to book live sessions or watch workout videos. [Multiple choice]

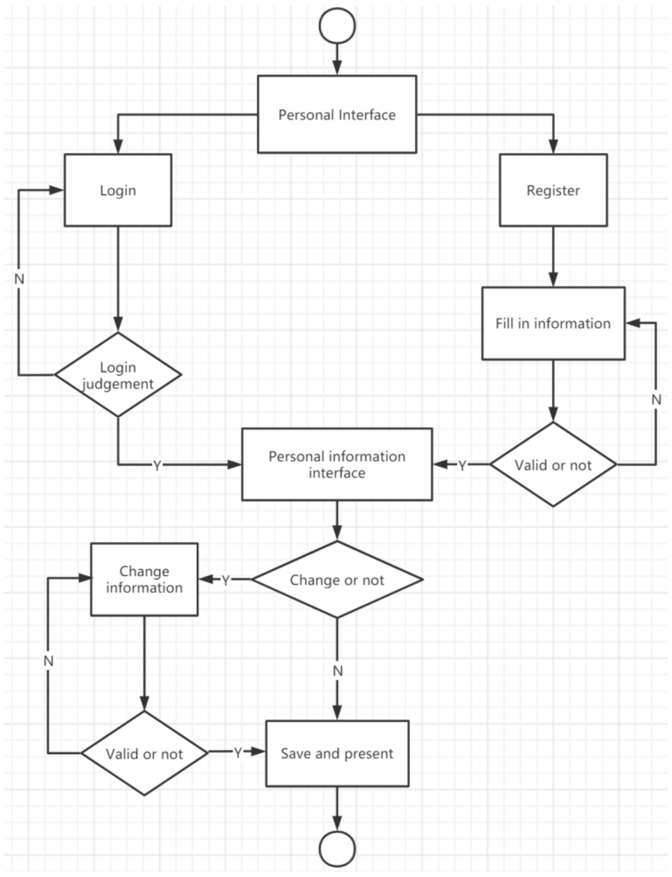


6.If you do not perform any operation within several minutes on the ordering interface, the software will automatically exit the ordering page. [Multiple choice]

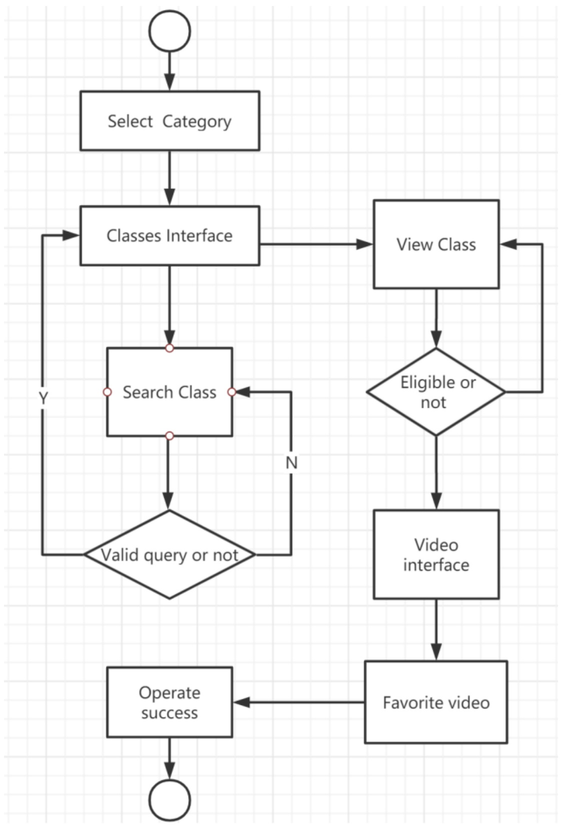


7.There should be a beginner's guide for user. [Multiple choice]

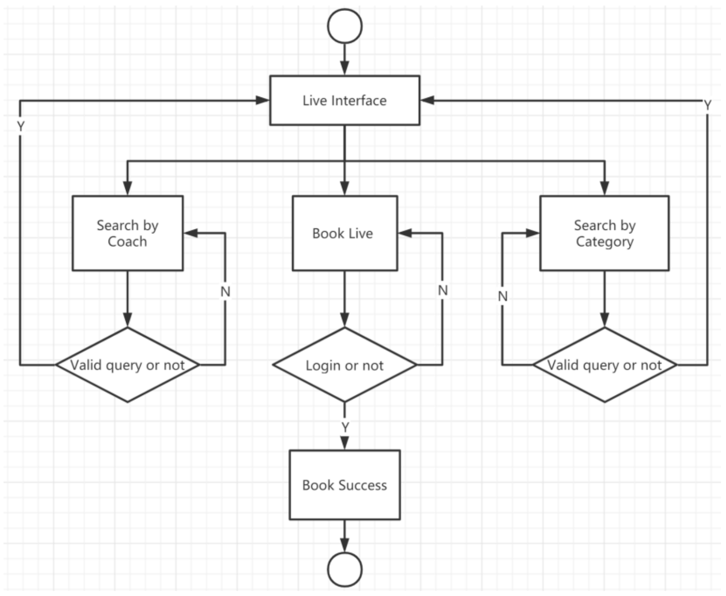




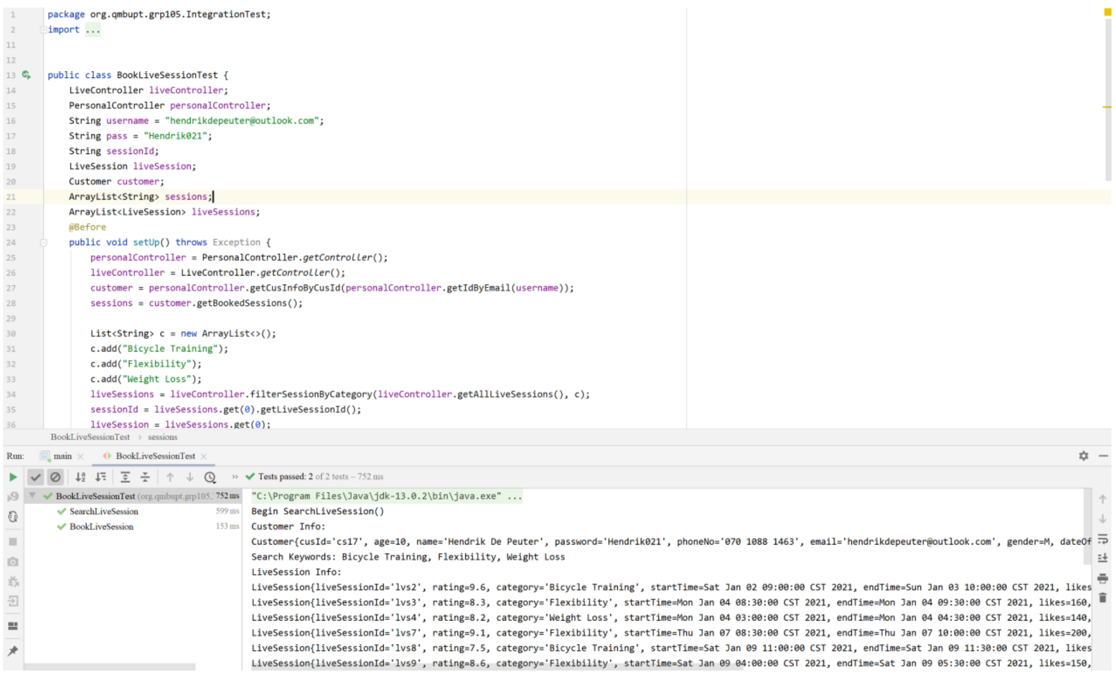
Basis path test 1



Basis path test 2



Basis path test 3



```
package org.qmbupt.grp105.IntegrationTest;
import ...


public class BookLiveSessionTest {
    LiveController liveController;
    PersonalController personalController;
    String username = "hendrikdepeuter@outlook.com";
    String pass = "Hendrik021";
    String sessionId;
    LiveSession liveSession;
    Customer customer;
    ArrayList<String> sessions;
    ArrayList<LiveSession> liveSessions;
    @Before
    public void setUp() throws Exception {
        personalController = PersonalController.getController();
        liveController = LiveController.getController();
        customer = personalController.getCusInfoByCusId(personalController.getIdByEmail(username));
        sessions = customer.getBookedSessions();

        List<String> c = new ArrayList<>();
        c.add("Bicycle Training");
        c.add("Flexibility");
        c.add("Weight Loss");
        liveSessions = liveController.filterSessionByCategory(liveController.getAllLiveSessions(), c);
        sessionId = liveSessions.get(0).getLiveSessionId();
        liveSession = liveSessions.get(0);
```

BookLiveSessionTest › sessions

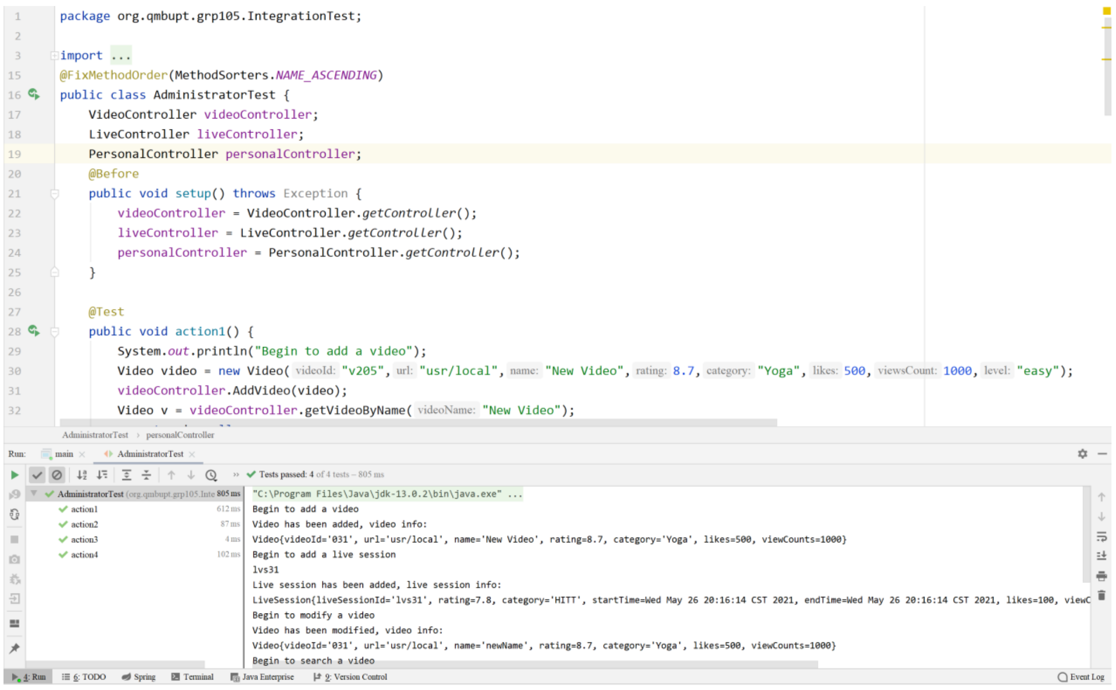
Run: main BookLiveSessionTest

Tests passed: 2 of 2 tests – 752 ms

BookLiveSessionTest (org.qmbupt.grp105.: 752 ms
SearchLiveSession 599 ms
BookLiveSession 153 ms

```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" ...
Begin SearchLiveSession()
Customer Info:
Customer{cusId='cs17', age=10, name='Hendrik De Peuter', password='Hendrik021', phoneNo='070 1088 1463', email='hendrikdepeuter@outlook.com', gender=M, dateOf
Search Keywords: Bicycle Training, Flexibility, Weight Loss
LiveSession Info:
LiveSession{liveSessionId='lvs2', rating=9.6, category='Bicycle Training', startTime=Sat Jan 02 09:00:00 CST 2021, endTime=Sun Jan 03 10:00:00 CST 2021, likes
LiveSession{liveSessionId='lvs3', rating=8.3, category='Flexibility', startTime=Mon Jan 04 08:30:00 CST 2021, endTime=Mon Jan 04 09:30:00 CST 2021, likes=160,
LiveSession{liveSessionId='lvs4', rating=8.2, category='Weight Loss', startTime=Mon Jan 04 03:00:00 CST 2021, endTime=Mon Jan 04 04:30:00 CST 2021, likes=140,
LiveSession{liveSessionId='lvs7', rating=9.1, category='Flexibility', startTime=Thu Jan 07 08:30:00 CST 2021, endTime=Thu Jan 07 10:00:00 CST 2021, likes=200,
LiveSession{liveSessionId='lvs8', rating=7.5, category='Bicycle Training', startTime=Sat Jan 09 11:00:00 CST 2021, endTime=Sat Jan 09 11:30:00 CST 2021, likes
LiveSession{liveSessionId='lvs9', rating=8.6, category='Flexibility', startTime=Sat Jan 09 04:00:00 CST 2021, endTime=Sat Jan 09 05:30:00 CST 2021, likes=150,
```

Scenarios test code 1



```
package org.qmbupt.grp105.IntegrationTest;

import ...
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class AdministratorTest {
    VideoController videoController;
    LiveController liveController;
    PersonalController personalController;
    @Before
    public void setup() throws Exception {
        videoController = VideoController.getController();
        liveController = LiveController.getController();
        personalController = PersonalController.getController();
    }

    @Test
    public void action1() {
        System.out.println("Begin to add a video");
        Video video = new Video( videoId: "v205", url: "usr/local", name: "New Video", rating: 8.7, category: "Yoga", likes: 500, viewsCount: 1000, level: "easy");
        videoController.AddVideo(video);
        Video v = videoController.getVideoByName( videoName: "New Video");
```

AdministratorTest › personalController

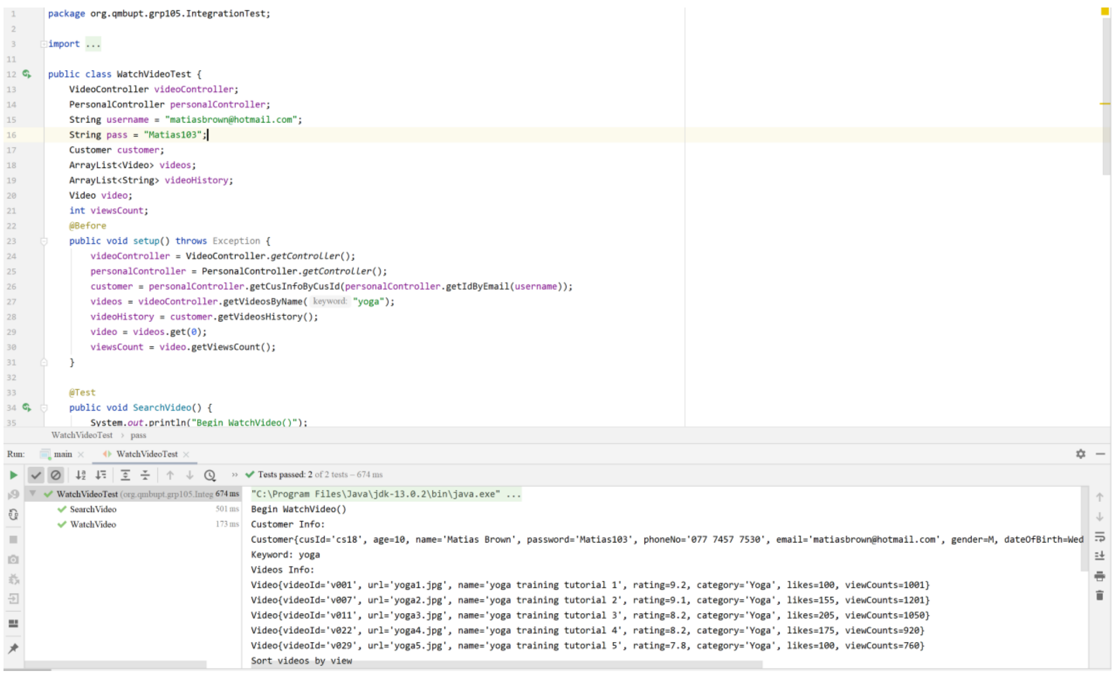
Run: main AdministratorTest

Tests passed: 4 of 4 tests – 805 ms

AdministratorTest (org.qmbupt.grp105.Inte 805 ms
action1 612 ms
action2 87 ms
action3 4 ms
action4 102 ms

```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" ...
Begin to add a video
Video has been added, video info:
Video{videoId='031', url='usr/local', name='New Video', rating=8.7, category='Yoga', likes=500, viewCounts=1000}
Begin to add a live session
lvs31
Live session has been added, live session info:
LiveSession{liveSessionId='lvs31', rating=7.8, category='HITT', startTime=Wed May 26 20:16:14 CST 2021, endTime=Wed May 26 20:16:14 CST 2021, likes=100, viewC
Begin to modify a video
Video has been modified, video info:
Video{videoId='031', url='usr/local', name='newName', rating=8.7, category='Yoga', likes=500, viewCounts=1000}
Begin to search a video
```

Scenarios test code 2

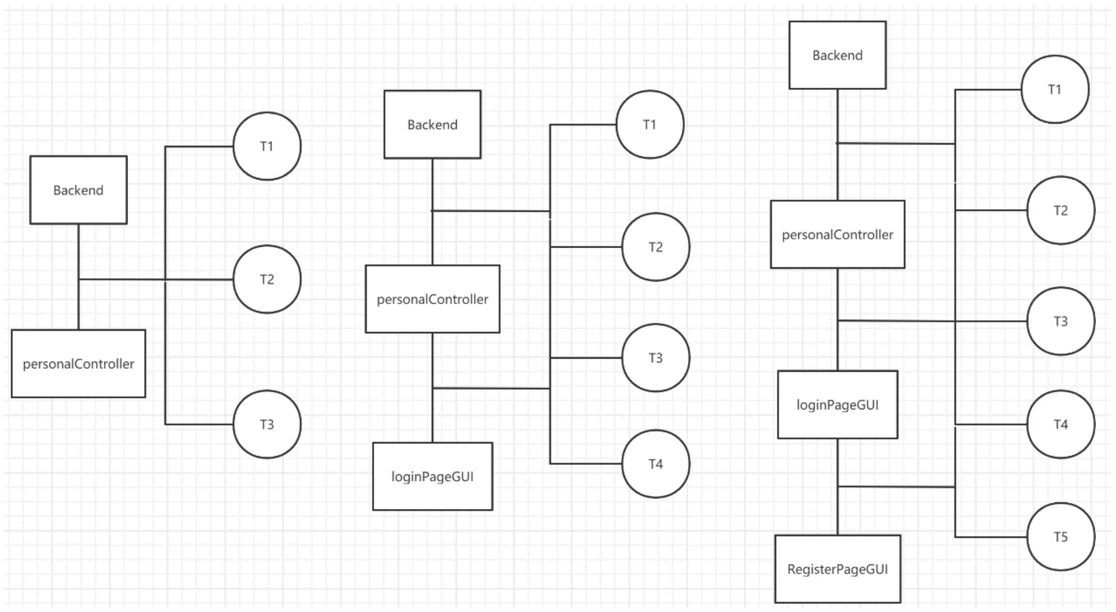


```java
package org.qmbupt.grp105.IntegrationTest;

import ...

public class WatchVideoTest {
    VideoController videoController;
    PersonalController personalController;
    String username = "matiasbrown@hotmail.com";
    String pass = "Matias103";
    Customer customer;
    ArrayList<Video> videos;
    ArrayList<String> videoHistory;
    Video video;
    int viewsCount;
    @Before
    public void setup() throws Exception {
        videoController = VideoController.getController();
        personalController = PersonalController.getController();
        customer = personalController.getCusInfoByCusId(personalController.getIdByEmail(username));
        videos = videoController.getVideosByName( keyword: "yoga");
        videoHistory = customer.getVideosHistory();
        video = videos.get(0);
        viewsCount = video.getViewsCount();
    }

    @Test
    public void SearchVideo() {
        System.out.println("Begin WatchVideo()");
```

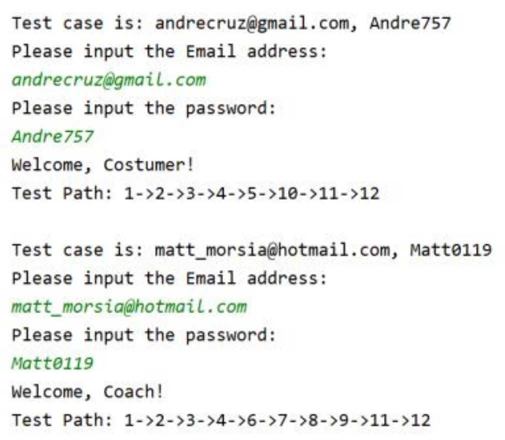
WatchVideoTest › pass
Run: main WatchVideoTest
Tests passed: 2 of 2 tests – 674 ms
WatchVideoTest (org.qmbupt.grp105.Integ 674 ms
SearchVideo 501 ms
WatchVideo 173 ms

```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" ...
Begin WatchVideo()
Customer Info:
Customer{cusId='cs18', age=10, name='Matias Brown', password='Matias103', phoneNo='077 7457 7530', email='matiasbrown@hotmail.com', gender=M, dateOfBirth=Wed
Keyword: yoga
Videos Info:
Video{videoId='v001', url='yoga1.jpg', name='yoga training tutorial 1', rating=9.2, category='Yoga', likes=100, viewCounts=1001}
Video{videoId='v007', url='yoga2.jpg', name='yoga training tutorial 2', rating=9.1, category='Yoga', likes=155, viewCounts=1201}
Video{videoId='v011', url='yoga3.jpg', name='yoga training tutorial 3', rating=8.2, category='Yoga', likes=205, viewCounts=1050}
Video{videoId='v022', url='yoga4.jpg', name='yoga training tutorial 4', rating=8.2, category='Yoga', likes=175, viewCounts=920}
Video{videoId='v029', url='yoga5.jpg', name='yoga training tutorial 5', rating=7.8, category='Yoga', likes=100, viewCounts=760}
Sort videos by view
```

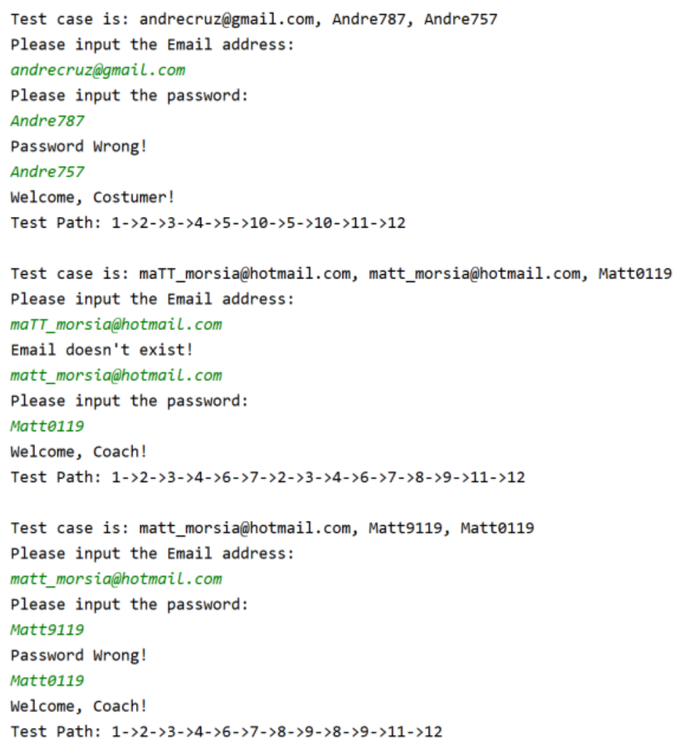
Scenarios test code 3



Regression testing example



```
Test case is: andrecruz@gmail.com, Andre757
Please input the Email address:
andrecruz@gmail.com
Please input the password:
Andre757
Welcome, Costumer!
Test Path: 1->2->3->4->5->10->11->12

Test case is: matt_morsia@hotmail.com, Matt0119
Please input the Email address:
matt_morsia@hotmail.com
Please input the password:
Matt0119
Welcome, Coach!
Test Path: 1->2->3->4->6->7->8->9->11->12
```



```
Test case is: andrecruz@gmail.com, Andre787, Andre757
Please input the Email address:
andrecruz@gmail.com
Please input the password:
Andre787
Password Wrong!
Andre757
Welcome, Costumer!
Test Path: 1->2->3->4->5->10->5->10->11->12

Test case is: maTT_morsia@hotmail.com, matt_morsia@hotmail.com, Matt0119
Please input the Email address:
maTT_morsia@hotmail.com
Email doesn't exist!
matt_morsia@hotmail.com
Please input the password:
Matt0119
Welcome, Coach!
Test Path: 1->2->3->4->6->7->2->3->4->6->7->8->9->11->12

Test case is: matt_morsia@hotmail.com, Matt9119, Matt0119
Please input the Email address:
matt_morsia@hotmail.com
Please input the password:
Matt9119
Password Wrong!
Matt0119
Welcome, Coach!
Test Path: 1->2->3->4->6->7->8->9->8->9->11->12
```

Test Result

References

[1] Runyan, K., Ashmore, S. (2014). *Introduction to Agile Methods.* (n.p.): Pearson Education.

[2] EBU6304_W8_Live-Project_Management_Part_2, Page 3.

[3] EBU6304_W3_Live-a_Summary_Exercises_Product Backlog_Prototyping, Page 19.

[4] Sommerville, I. (2016). *Software Engineering.* United Kingdom: Pearson.

[5] Beck, K. (2003). *Test Driven Development.* Germany: Addison-Wesley.