SE250 Software Engineering Final Report

QB Weather Assistant

Group Member

Fu, Zhanchao	*1909823M-I011-0046*
Yu, Mengke	*1909853W-I011-0021*
Liao, Weiyu	*1909853G-I011-0138*
Li, Yiran	*1909853D-I011-156*

Catalog

1 Requirement Document

1.1 Problem

In the modern society where the pace of life is accelerating, people's outdoor exercise time is becoming more and more compact, and many people are discouraged from exercising and going out because of uncertainty about the weather. In another situation, people do not know what exercise is appropriate in the weather conditions of the day, so they are confused about whether to exercise and what exercise to do. o modern people need a more convenient and user-friendly service for weather prediction, and it also has the function of exercise recommendation.

1.2 Background

There are a lot of software like Weather Assistant on the App market (such as ink weather and JiKe weather). They all focus on showing details of recent weather and aspects of people's lives, especially for travel and clothing. Our project is more focused on urging users to exercise. It also focuses on the assessment of fitness for exercise, and the user's exercise attendance record. Users are encouraged to do more outdoor sports by recommending them to exercise today and recording their exercise process.

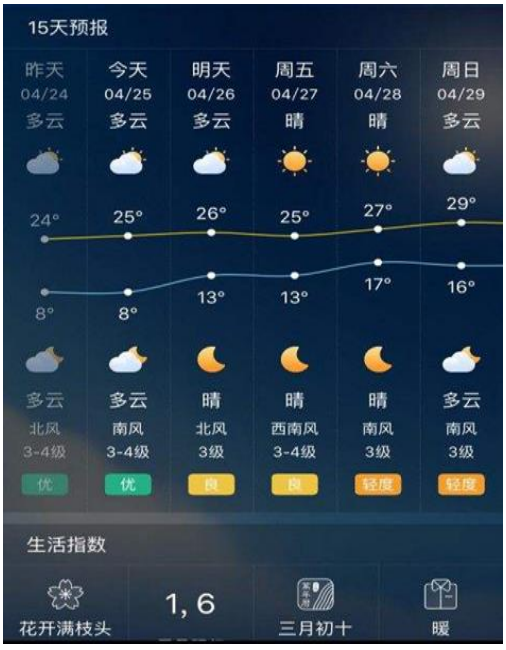


Figure 1 Ink Weather

1.3 Domain Analysis Document

We conducted a brief survey of the existing weather assistants on the market and found that the existing weather assistant software is already very complete, including not only the basic weather forecast function, but also clothing recommendations, weather reminders and other helpful features.

Here is some information about ink weather:
Moji Weather, a weather information query software, user-friendly design, simple to use, is the most supported city weather forecast software in China. The weather APP, which is used by about 500 million people around the world, supports weather query of more than 700,000 cities and regions in 196 countries, accurate positioning and timely push, minute-level and kilometer-level weather forecast, and real-time monitoring of rain, rain and snow. Provide 15-day weather forecast, 5-day air quality forecast, real-time air quality and air quality grade forecast. Special weather send early warning information, help users make better life decisions, calmly deal with all kinds of weather conditions.

In June 2016, Moji registered users exceeded 500 million and monthly active users exceeded 100 million, accounting for 53.9% of the market share of similar software. Practice "Weather +", based on meteorological big data, take weather as the entrance, and provide users with more life-oriented and scenario-based services.

As a weather assistant software, we naturally need to achieve some basic functions, which are similar to the common weather assistant software mentioned above in the market, such as forecasting the weather in seven days, extracting weather conditions according to the location of the user, etc..

However, we found that for people who want to exercise, but the weather in their area varies greatly during the morning and evening, or the air quality is poor, they cannot accurately determine whether they can do the outdoor exercise they want to do based on only a few weather indices and naked eye observation.

So, we added exercise recommendations on the basis of clothing recommendations, this allows QB to meet the diverse needs of most people's lives, and it can be used well in a variety of scenarios such as commuting, travelling, and exercising.

Therefore, although QB is one app for all people, it can make the lives of the following three groups of people more convenient and avoid the hassle of downloading a separate app for each function they need, specifically:
1. People who want to check the daily detailed weather conditions.
2. People who want to do outdoor exercise but don't know if the weather is suitable.
3. People who want to record the daily exercise status but don't want to download an extra fitness app.

1.4 Environment and System Models

1.4.1 Environment

We mainly use java to develop this project, and IDE will also be used.
The final product will be able to run smoothly in Windows and macOS.

1.4.2 Use Case Diagram

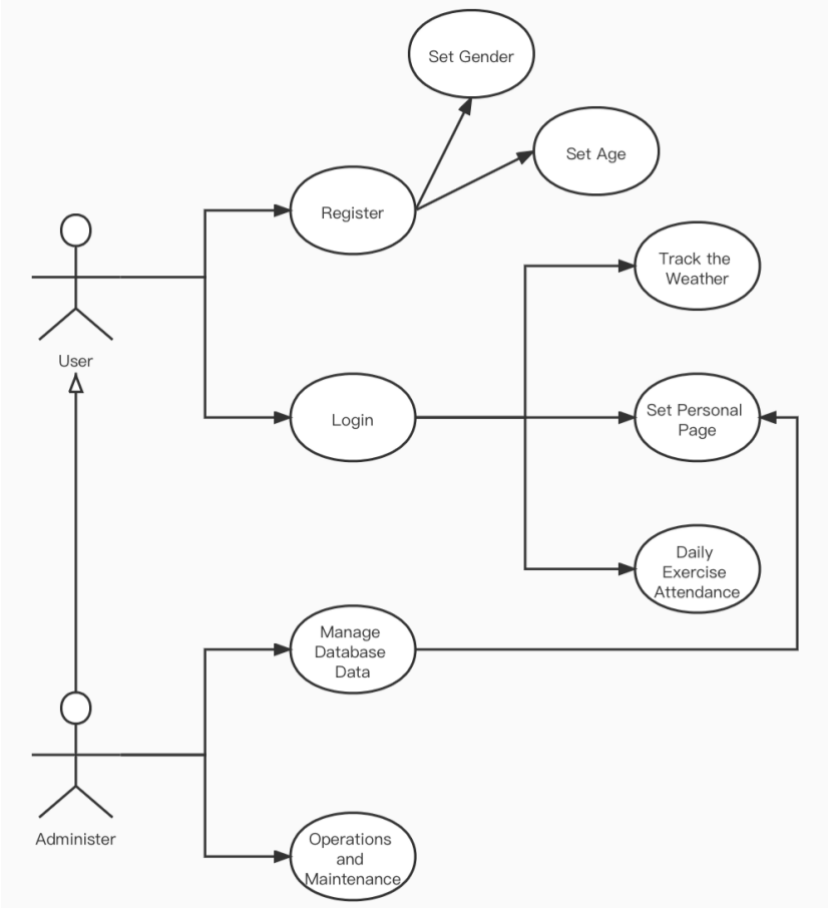


Figure 2 Use Case Diagram

We mainly divide the functions of QB into two parts, users and administrators, where users can achieve:

- Register
- Login
- Track the weather
- Set personal page
- Daily exercise attendance

Administrators can achieve:

- Manage database data
- Operations and maintenance

For each use case, we project for the actor actions and the corresponding system responses:

Use Case: Register	**Related Use Case:**
Actor Actions: 1. Choose *register* command 3. Input username, password, gender and age 4. Return result	**System Responses:** 2. Response *register* page 5. Dialog disappears

Use Case: Login	**Related Use Case:**
Actor Actions: 1. Open the software 3. Input username and password 4. Return result	**System Responses:** 2. Response *login* page 5. Dialog disappears

Use Case: Edit Age and Gender	**Related Use Case:** Register
Actor Actions: 1. Choose *personal home* page 3. Select related information 4. Change the information 5. Confrim information	**System Responses:** 2. Response *personal home* page 6. Dialog disappear

Use Case: Weather Forecast	**Related Use Case:** Login
Actor Actions: 1. Select the city	**System Responses:** 2. Pop-up city list 3. Display local weather condition

Use Case:	**Related Use Case:**
Set Sports Reference	Login
Actor Actions:	**System Responses:**
1. Choose *personal home* page	2. Reponse *personal home* page
3. Click *sports reference*	
4. Choose sports	
5. Confrim information	6. Dialog disappear

Use Case:	**Related Use Case:**
Daily Exercise Record	Login
Actor Actions:	**System Responses:**
1. Click *record attendance*	2. Response *record attendance* page
3. Click the day and select sports	
4. Confrim information	5. Dialog disappear

1.4.3 Evolutionary Model

Because our project is a small and medium-sized system that is short of demand and short development cycles, the life cycle model we chose was the evolutionary model. The evolutionary model advocates "double development", namely experimental development and product development. Each development phase follows the waterfall model for specific development activities. The benefits of choosing an evolution model are to clarify user needs, improve system quality, and reduce development risks.

First trial development

This phase includes requirements analysis, software design, software implementation, software testing, and software deployment.

At this stage, we roughly estimated the user needs, developed an executable version with the fastest speed, and realized the basic popularization of functions. In the development mode, the method of batch cyclic development is adopted, and each cycle develops a part of the function, which becomes the new function of the prototype of the product. As a result, the design constantly evolves into new systems.

Second product development

After the end of the first part, user reviews and feedback from the prototypes were collected to revise and add requirements and start the development of the second deliverable. Lessons learned in the first development can be fed back and applied to the next cycle of the product, greatly improving quality and efficiency.

After the second development, the delivery phase is over.

However, depending on the principle of flexibility, a third or fourth finished product development phase may be added depending on the actual situation and software complexity.

1.5 Functional Requirement

According to the basic requirements of QB, the specific functions of this project are as follows:

- Weather Forecast

Display the weather data of the users' city. Weather data can only be obtained through the call of the interface to get the corresponding data. After getting the geographic location according to IP automatically, QB will pass it to the interface to pull the weather data and then display them to the users, so that the weather forecast function can be realized.

- User Personalized Settings

According to the personalized data selected by the user, combined with the current city weather data, we set a built-in algorithm to provide the corresponding sports or life advice. Each user has his own account and keeps his personal preferences.

- Daily Exercise Attendance

The user is urged to exercise and can share the record on social media platforms. It is possible to do this by reading the computer calendar. For the exercise attendance function, we need to design the database to save storage space and ensure the integrity of the data. The idea is to first determine whether the item exists, then determine whether the attendance has been checked today, and insert the data if not, then calculate the number of consecutive days and the total number of days. After today's check, QB will check whether the attendance missed yesterday, if so, the number of consecutive days is set to one, and the total number of days is added to one. Here is mainly written using SQL statements.

- Keyword Search

For example, if you enter the corresponding sport or outdoor activity in the search bar, QB will launch the activity with the corresponding suitability level, and our expectation is that the information will appear within 2s. The search function is then implemented through a chain table. For the search of keywords, the index is first established to classify the keywords and pages correspondingly well. In response to

the user's search needs, the keywords entered by the user are first decomposed and all relevant contents matching the keywords are found from the index.

- Calculate sports suitability index

Based on our analysis of the limitations of the existing weather assistant, we require QB to calculate the suitability index for specific sports and make recommendations based on the weather of the city selected by the user. The specific algorithm to implement this function will be analyzed and designed in more detail later.

1.6 Non-Functional Requirement

- Response Time

At 95% of cases, the response time should not exceed 2 seconds during normal hours and 5 seconds during peak hours.
When the network is available, the time required for the localization system to display the first screen from the click to the first screen shall not exceed 300 ms.
In the recommended configuration environment: the response time for login is within 2 seconds, the response time for refreshing the page is within 2 seconds, the response time for opening each function entry is within 2 seconds, and the response time for refreshing personal data is within 2 seconds.
Searching for cities and campaigns based on number and name during off-peak hours can yield search results in less than 3 seconds.

- Throughput

The estimated number of users is 10,000, the number of logged-in users per day is about 1,000, and the bandwidth of the network is 100M.
The system can accommodate 10,000 simultaneous user requests and provide browsing capabilities for 20,000 concurrent users.

- Resource Usage

CPU occupancy is no more than 40% and memory occupancy is no more than 40%.

- Environmental requirements

Operating system: Windows 7/ MacOS Sierra - 10.12.6 or better
Processor: 1.7+ GHz or better
Network: Broadband Internet connection
Storage space: 200 MB free space required

- Reliability

For user information (username, password, etc.), there are prompts for input and checks are performed to prevent data exceptions.
The system should be able to handle basic abnormal situations that occur during system operation, such as: operation errors, inputting illegal data, etc. For these problems, the system should be able to handle them correctly and make proper evasion.
The probability that the user cannot complete the operation due to the failure of the software system should be less than 5‰.

- Availability

Ensure the availability of the basic functions of the system under the conditions of poor network quality, small bandwidth and other poor network environment.
Provide data backup and recovery function for the database to prevent the loss of user data.

- Usability

The interface is simple and aesthetic, without redundant components, and each function entrance can be easily found by users without misleading them.

- Allowances for Maintainability and Enhancement

90% of the bugs will take no more than 1 working day to modify, and the others will take no more than 2 working days.
The original database content and all personal settings must remain unchanged after installing a new version.

2 Design Document

2.1 Purpose

QB is expected to be a very user-friendly software that calculates the daily weather data, displays the recommended index for various outdoor activities in percentages and provides exercise recommendations.

We also expect QB to be able to let users log in the system and personalize their own homepage. After logging in, users may record their daily exercise status and share the record to the social media if they want.

In addition, QB will also have a good human-machine interface, making it easy to operate and more convenient to help users plan their daily trips.

2.2 General Priorities

QB has the following objectives, starting with top priority:

- Security: Considering that querying weather requires access to users' location information, we store users' IP addresses separately to prevent users' privacy from being leaked.
- Maintainability: We require QB's system to be stable, reduce the number of system crashes, and enable administrators to maintain the system in a timely manner.
- Portability: To meet the needs of more users, we require QB to run smoothly on Windows and macOS systems.
- CPU efficiency: Must respond to the user within two second when running on a 3GHz Intel processor.
- Network bandwidth efficiency: Must not require transmission of more than 1MB of data per transaction.
- Memory efficiency: Must not consume over 200MB of RAM.

2.3 Outline of the Design

- Login and Registration

In the process of realizing login and registration functions, we choose event-driven architecture. An Event-driven architecture is a software architecture that communicates through events.
Event Queue: An entry point to receive events
Event Mediators: Distribute different events to different business logic units
Event Channels: Communication channels between dispensers and processors
Event Processor: Implements the service logic. After the processing is complete, an event is emitted to trigger the next operation

Advantages:
Distributed asynchronous architecture, high decoupling between event processors, good software expansibility.
Wide applicability, all kinds of projects can be used.
Performance is good because the software is less prone to clogging due to the asynchronous nature of events.
Event handlers can be loaded and unloaded independently and are easy to deploy.

Disadvantages:

Development is relatively complex when asynchronous programming is involved (consider remote communication, loss of response, and so on).

Atomic operations are difficult to support because event passage involves multiple processors and is difficult to roll back.

The distributed and asynchronous nature of this architecture makes it difficult to test.

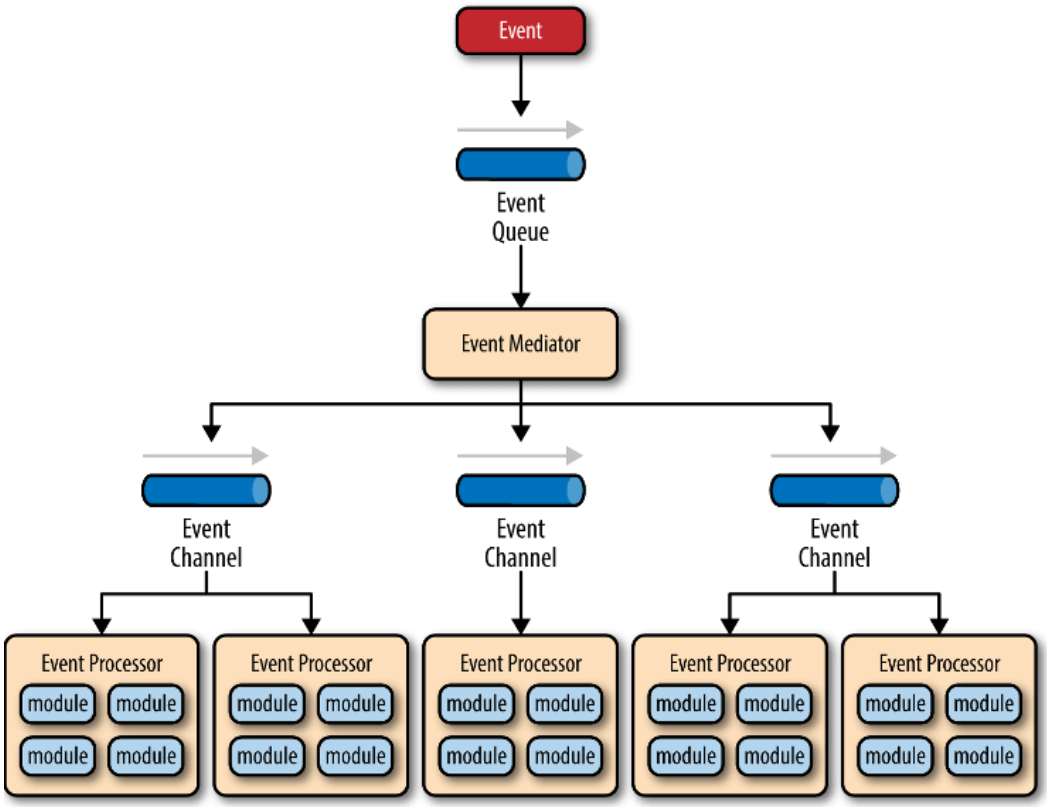


Figure 3 Event-Driven Architecture

In the implementation of login and registration functions, the event-driven architecture can be used to pass different user operations as events to the distributor, which can be passed to different event handlers. For example, the user password input format error can be passed to the core code as an event, so as to return a hint. Event-driven architecture can be more flexible and organized to achieve the existence of a variety of events, which is very suitable for login and registration.

- Database

In terms of database, we chose microcore architecture. Microkernel Architecture, also known as "plug-in architecture", refers to the relatively small core of software, and the main functions and business logic are implemented through plug-ins. The core

usually contains only minimal functionality for the system to run. Plug-ins are independent of each other, and communication between plug-ins should be kept to a minimum to avoid interdependence problems.

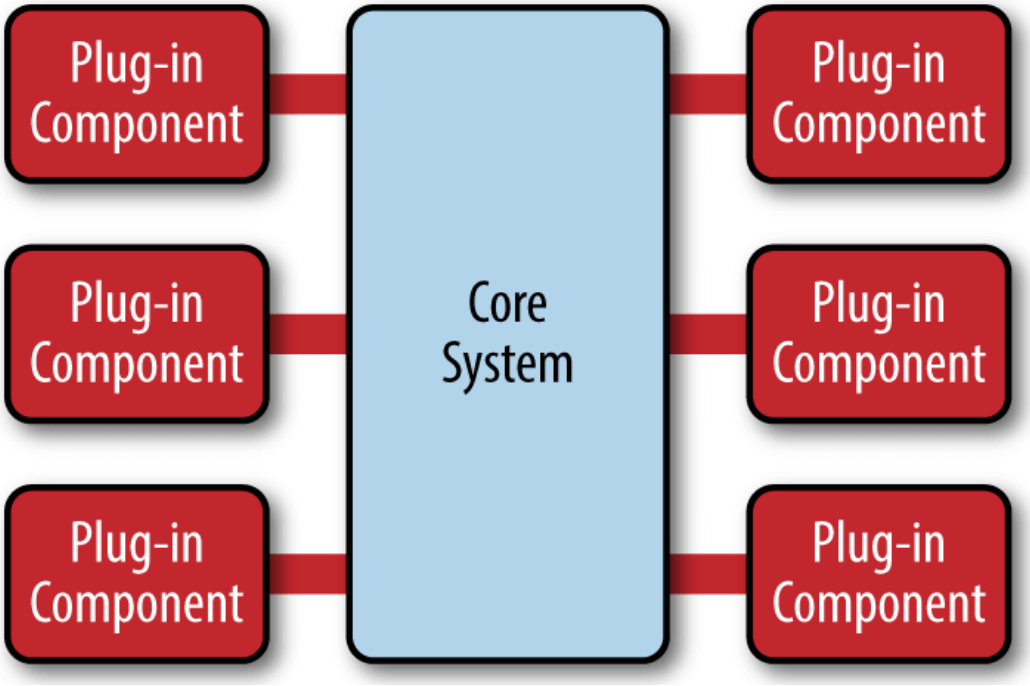


Figure 4 Microcore Architecture

Advantages:
Good function extensibility (extensibility), what function do you need to develop a plug-in.
Functions are isolated, plug-ins can be loaded and unloaded independently, making it easier to deploy.
High customizability, adapt to different development needs.
It can be developed incrementally, gradually adding functionality.

Disadvantages:
The kernel is usually an independent unit and cannot be easily distributed.
Development is relatively difficult because of the communication between plug-ins and the kernel, as well as the internal plug-in registration mechanism.

In the process of implementation, we found that the database controller kernel is relatively simple. Major functions and business logic are implemented through plug-ins, such as user creation, data control, identity authentication, etc. The kernel only contains the ability to accept and schedule tasks, and the implementation of tasks depends on plug-ins.

Throughout the development process, we used the following design principles:

- Divide and conquer: The website of our second-hand trading platform adopts a hierarchical mode, so the division of labor among team members can be better carried out.

- Loose Coupling: In the process of writing the relevant function implementation function of the campus second-hand trading platform, we have been paying attention to minimize the dependence on each other.

- Service Abstraction: The internal code of our website is implemented in a highly abstract way.

- Service Reusability: Logic is divided into services with the intent of maximizing reuse.

- Service Autonomy: Services control the logic they encapsulate.

- Service Statelessness: Services are stateless.

- Service Discoverability: Services can be discovered.

- Service Interoperability: Services should use standards that allow diverse subscribers to use the service. This is considered so obvious these days that it is often dropped as a principle.

Besides that, the overall system of our software is designed with a combination of top-down and bottom-up design.

We used top-down design to divide the entire Weather Assistant development into several parts: personalized user settings, weather data acquisition and display, outdoor activity rating based on weather conditions and preferences, and checking the daily outdoor activity attendance. After refining the problems in these parts, we decided to use a web interface and design a separate algorithm and database to solve the problem after discussion.

The bottom-up design ensures the usability and reusability of the user registration and login panels, the personalization panel, the search dialog, and the various buttons on the weather display screen of our system.

2.4 Major Design Issues

- Weather data acquisition

Due to our team's development costs, production methods, development tools and information acquisition channels are far inferior to some professional teams, coupled with the professional and special nature of weather data, we are unable to generate the necessary data for this software.

Also, considering the cost issues of our team and the available tools, we mainly use the interface for weather data acquisition.

The weather data we end up displaying is mainly divided into real-time data and forecast data for the next seven days, and also includes the geographical location of the user.

We will first use the URL resource locator to pull the IP resource, and then use regular expressions to match the content we want so that we can get our local public IP address and use an interface to query the geographic location of the corresponding IP. Once we get the geographic location, we upload it to the weather interface to pull the weather data we need for our software. According to the information that our team members have found, we need to use Jdom to parse the data obtained from the interface. Some professionals have tested a similar approach to this weather data fetching Some professionals have tested a similar approach to this weather data acquisition with an accuracy rate of 90%. The feasibility of this design solution is still very high at the moment.

- Independent algorithm

Our built-in algorithm is positioned to calculate the recommended index of a major category of sports based on the weather conditions of the user's city, such as temperature, humidity, and other factors, i.e., the degree of suitability for that sport.

Based on the knowledge of mathematical modeling, we can conclude that the implementation of this function belongs to the evaluation class of problems and can use hierarchical analysis. First we should review the literature to summarize the main weather and environmental factors affecting outdoor sports, and then determine the weight of each factor based on expert analysis and a large amount of data, which can be based on the actual situation by introducing mathematical methods such as sampling surveys and judgment matrices to calculate the weights, and the second step is to give an evaluation. Our design is to divide each factor into two to three

numerical intervals, divided into three levels of good and bad, and then assign a score from high to low, with the maximum value set at 1, and multiply it with the weights, and the final result is the recommendation index.

In the subsequent design process, we optimized the algorithm to make its structure more scientific and influence factors more comprehensive and refined.

We choose factor multiplication, taking weather phenomenon, temperature, relative humidity, wind speed and air quality as five factors A, B, C, D and E.[1]
Its overall motion index K can be calculated by:

$$K = A \times B \times C \times D \times E$$

For specific sports recommendation indexes, we choose to add weight coefficients in front of relevant factors, so as to reflect the recommendation indexes of different sports in the positive direction.

2.5 Other Details of the Design

According to Design Principle 11: Defensive Design, we have pre-defined the following centralized measures to deal with possible system failures caused by users operating too fast.

When there is a network demand:
After the user interacts with the component, temporarily turn off the interaction capability of the component (e.g., the button turns gray and cannot be operated) until the network data returns or timeout.

When there is no network demand:
Redundant requests sent by users are put into a queue and executed one by one, setting queue limits to prevent software crashes.

2.6 Class Diagram

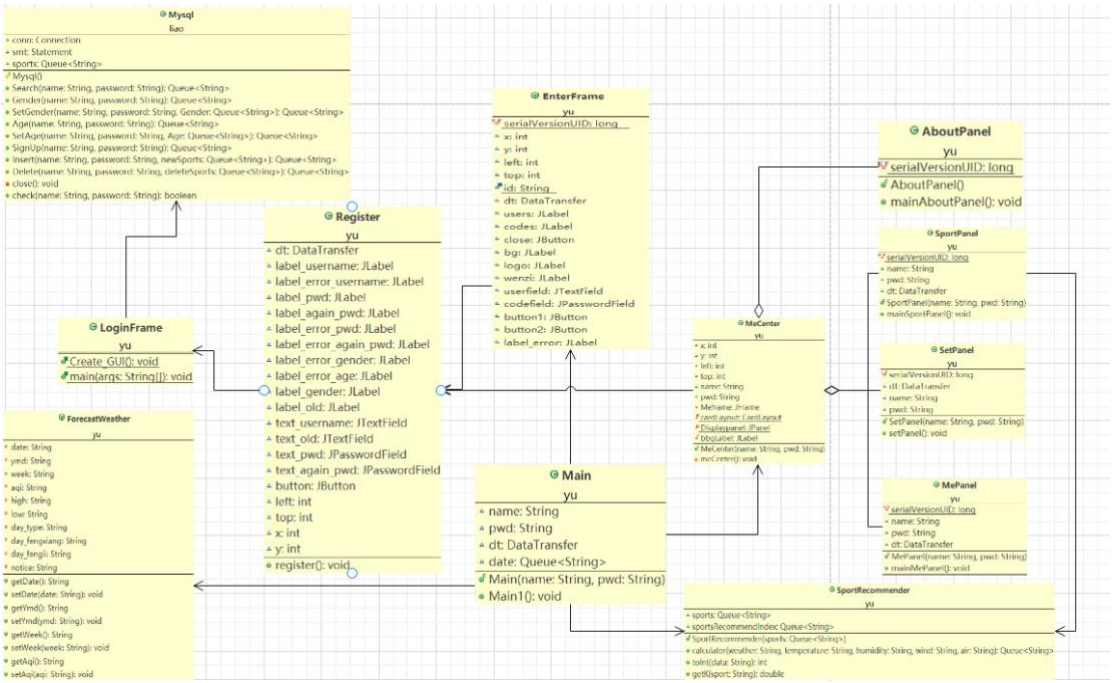


Figure 5 Class Diagram

2.7 Sequence Diagram

For easy viewing, we divide the sequence diagram into the following parts:

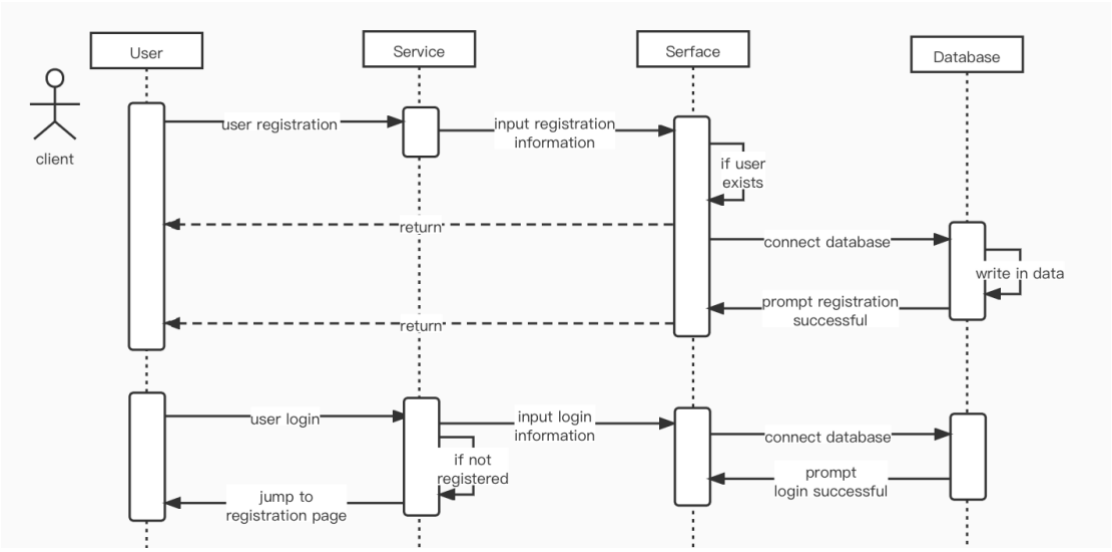


Figure 6.1 Sequence Diagram – Login and Registration

First, we will show the basic function of QB: login and registration, all the function are to be operated on the basis of user login.

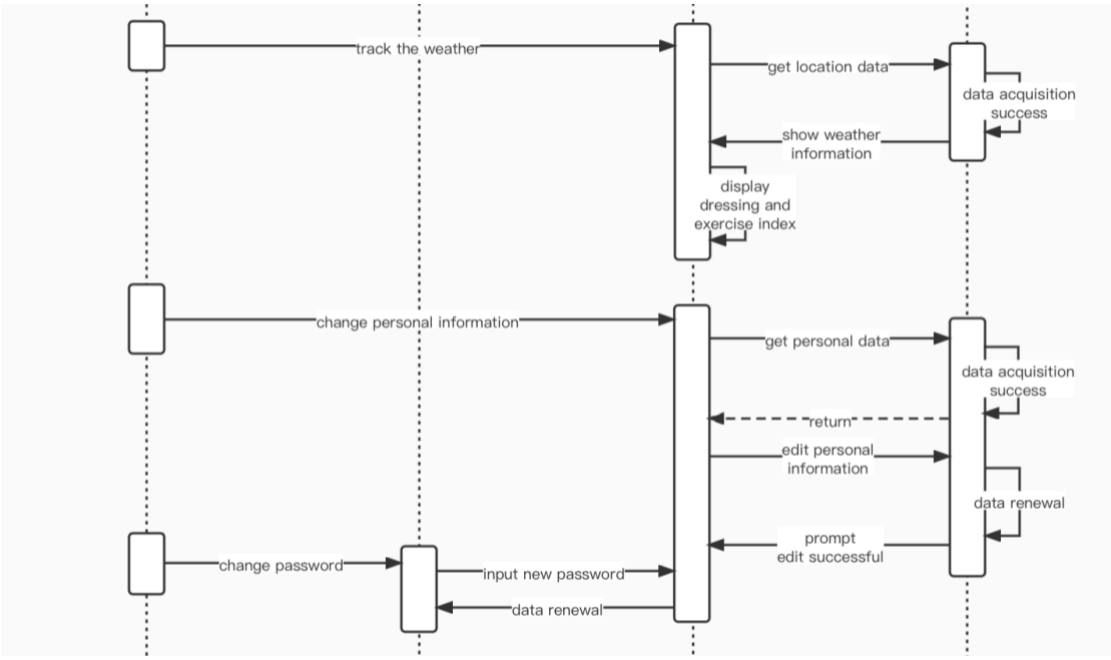


Figure 6.2 Sequence Diagram –Simple Functions

Then, there are three simple functions of QB, which are weather forecast, editing personal information and rewriting password.

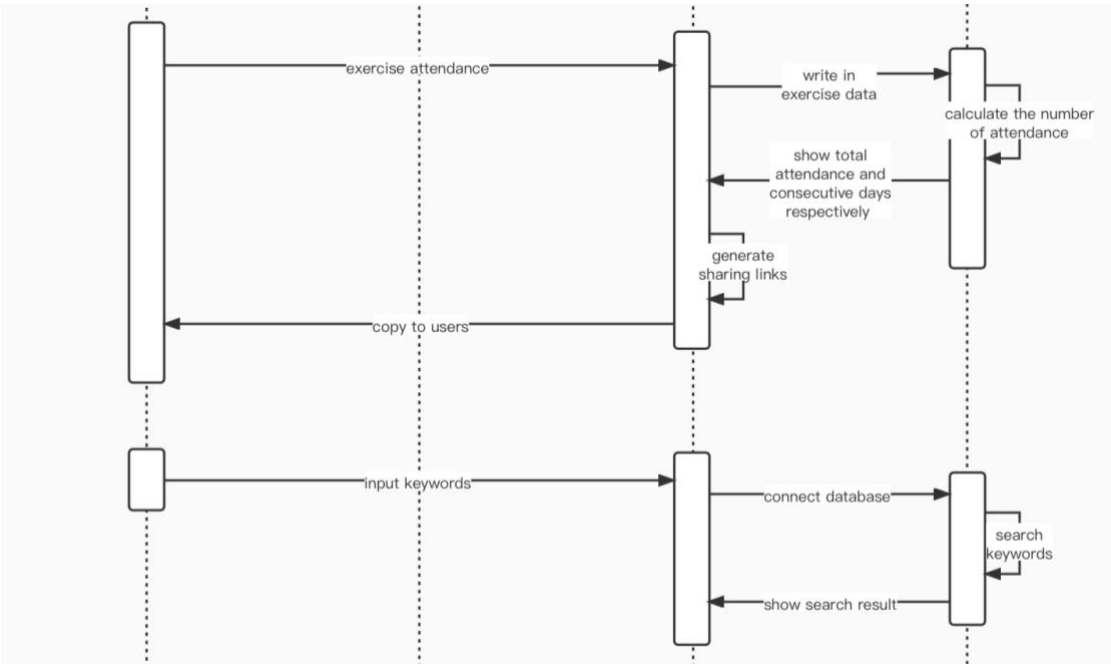


Figure 6.3 Sequence Diagram – Advanced Functions

To increase the usability of QB and facilitate our users, we also develop some more advanced functions, where one is daily exercise attendance and the other is searching for sports by keywords.

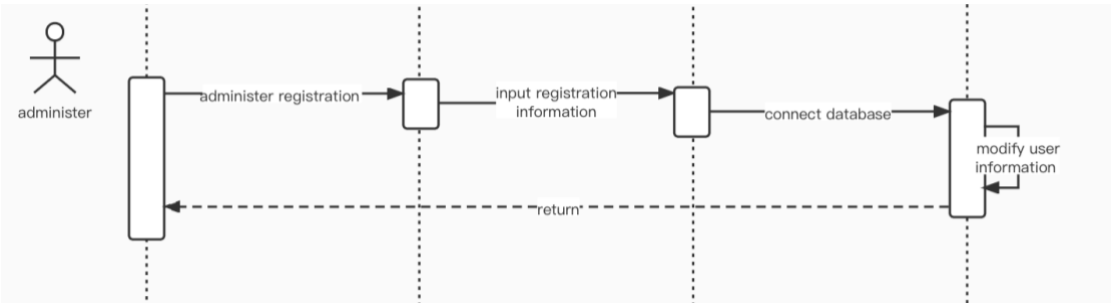


Figure 6.4 Sequence Diagram – Administer

Also, to better organize the data of our users, the administer is indispensable. The administer will have an account with higher privileges to change and delete registered user account information, and will be able to add new accounts directly instead of users registering themselves.

3 Development Document

3.1 Scrum Framework

In each iteration, we organize four meetings, sprint planning, daily stand-up meeting, sprint review, and sprint retrospective, to make our agile development process more efficient.

Figure 7 Scrum Framework

- Sprint planning

We set up a total of three iterations, and at the beginning of each agile iteration, the Product Owner would first explain the requirements, and then the Team members would plan the work hours. During the meeting, we prioritized the requirements, analyzed and evaluated the PBL and determined the goals for this iteration, and then developed the iteration plan in turn, including: creating iteration tasks based on the PBL, making estimates for those tasks, and letting Team members pick the entries

they would complete from the PBL. Finally, we determine the completion time and the final review meeting and retrospective meeting.

- Daily Stand-up Meeting

As the iteration progresses, each day the team members will hold a short meeting in which we will give a narrative of what we have accomplished and what we plan to do next, and bring up the problems we encountered and work together to find a solution.

Also known as a Daily Scrum, a 15-minute mini-meeting for the software team to sync. At the meeting, each team member is asked three questions:

What did you get through yesterday?
What will you get done today?
Are there barriers to achieving your goals?

It must be emphasized that the purpose of standing meetings is to update the status of the team, not to manage it.

- Sprint Review

Before the end of the iteration, we will have a review meeting for the Product Owner to give a demo and come up with a new Product Backlog based on the feedback.
The main purpose of the meeting is to check the results of this iteration, to check if all the goals in the iteration plan are accomplished, and to get a few users to participate in the testing process, so that we can get user approval of the product or find any problems missed before.

In this part, we totally have five stages:
Stage 1: Preparation

- The moderator of the retrospective meeting will discuss a topic of common interest with the Scrum Master at the beginning of the meeting, which is of common interest to everyone in the Sprint.
- The moderator of the retrospective organized the improvements decided by the retro team last time.
- Review the PPT required for the meeting.

Stage 2: Data collection
Stage 3: Generating insights

- The good backlog was sorted and compared to the previous good and bad backlog to encourage the team to keep it.
- For the bad ones, we also sorted out and selected no more than 2 for improvement. If too many improvement items were selected, the team would not be able to follow suit, and ultimately all the improvements would fail

Stage 4: Identify improvement items

- For each improvement point, we should analyze the root cause of the problem in detail, and then propose improvement measures based on the root cause.

Stage 5: End

- Sprint Retrospective

The retrospective is one of the most important meetings in agile development. It is stated in the Agile lesson plan that this meeting is held at the end of the Sprint and is mainly for summary and induction.

In the last retrospective session, we summarized what we did well in this session and what needs to be improved, in time to reflect on our recent work and to better make continuous improvements. This also matched with the 12th principle of agile development.

3.2 Team Member Roles

Product Owner – Fu, Zhanchao

- Creates a prioritized wish list called a product backlog.
- Closely partners with the business and the team to ensure everyone under-stands the work items in the product backlog.
- Gives the team clear guidance on which features to deliver next.
- Decides when to ship the product with a preference towards more frequent delivery.

Scrum Master – Li, Yiran

- Coach the team, the product owner, and the business on the scrum process and look for ways to fine-tune their practice of it.
- Deeply understands the work being done by the team and can help the team optimize their delivery flow.
- Schedule the needed resources (both human and logistical) for sprint planning, standup, sprint review, and the sprint retrospective.
- Look to resolve impediments and distractions for the development team, insulating them from external disruptions whenever possible.

Team – Yu, Mengke; Liao, Weiyu

- Cooperate to complete the sprint development work.
- Ensure the success of every sprint.

- Actively participate sprint planning, standup, sprint review, and the sprint retrospective.
- Ensure file stability and basic implementation of functions.
- After each meeting to improve the code, is the smooth progress of the program.

3.3 Development Process

3.3.1 Epic Story

Large user stories are often called "epic stories". Epic stories typically take 1 to 2 sprints to develop and test. They are usually large in scope and small in detail, and often need to be broken down into smaller stories before the team can develop them.

First, we broadly categorized the user requirements based on several key functions, and after a period of discussion, we arrived at the following user requirements and grouped them into three milestones:

We designed the main interface, personal information interface, and login interface so that users can clearly understand the functions of QB and quickly get started using it.

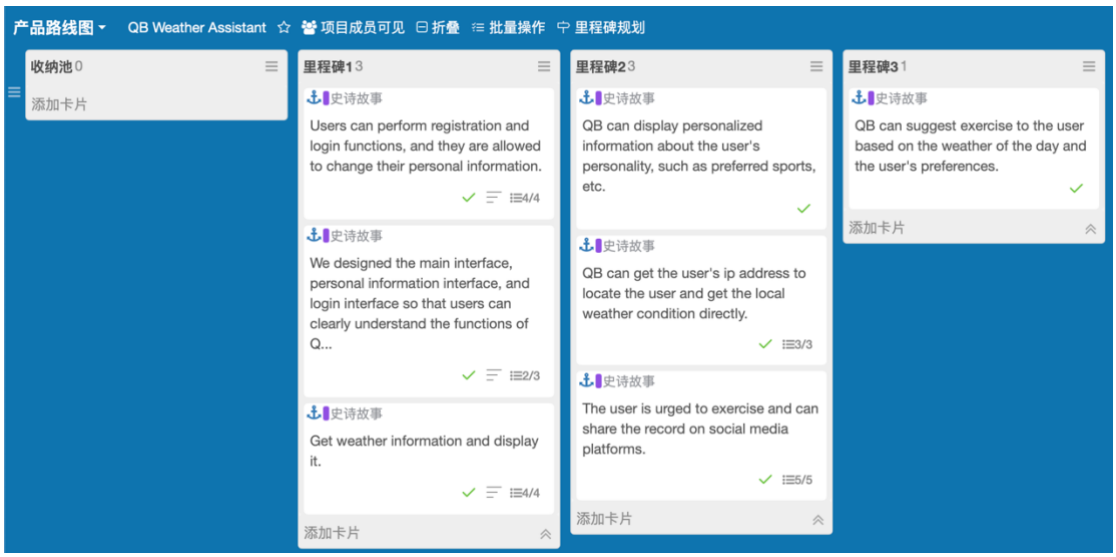


Figure 8 Product Roadmap

Besides, users also can perform registration and login functions, change their personal information, get weather information, personalize their information about their personality, such as preferred sports, etc., get the local weather condition directly, be urged to exercise and can share the record on social media platforms, and be suggested exercise to them based on the weather.

3.3.2 User Story

User stories describe features from the perspective of people (usually users or customers of the system) who crave new features. User stories are usually described in a simple format like this: "As [certain user] I [want/can/need...]. In order to [satisfy what user value] ". Despite the advantages of this description format, user stories can be described in any form as long as they are communicated around them.

We then split the user requirements into smaller modules of user stories based on the epic story:

- Registration and Login

As a user, I wish I could register as a new user and login to QB with the account and password I have set up; I expect QB to be able to display the correct information such as my username, gender, etc. about myself; I expect to be able to edit my username, gender, etc., and change my password to protect the security of my account.

- Interfaces

As a user, I need a friendly user panel to guide me through the login and registration process.

- Get Weather

As a user, I want to be able to get the weather information for the whole country and display it; I would also like to be able to check the weather in a specific city.

- Display Personalized Information

As a user, I would like QB to allow me to have a personal homepage that I can show my personal information in it.

- Get Local Weather

As a user, I would like QB to automatically get weather information based on my location.

- Exercise

As a user, I want QB to urge me to exercise and also allow me to share my daily punch card to social media platforms; QB can also remind me to clock in on time, and if I forget to clock in, the number of consecutive clock-in days will be reset and the cumulative number of days will remain the same.

- Suggest Exercises

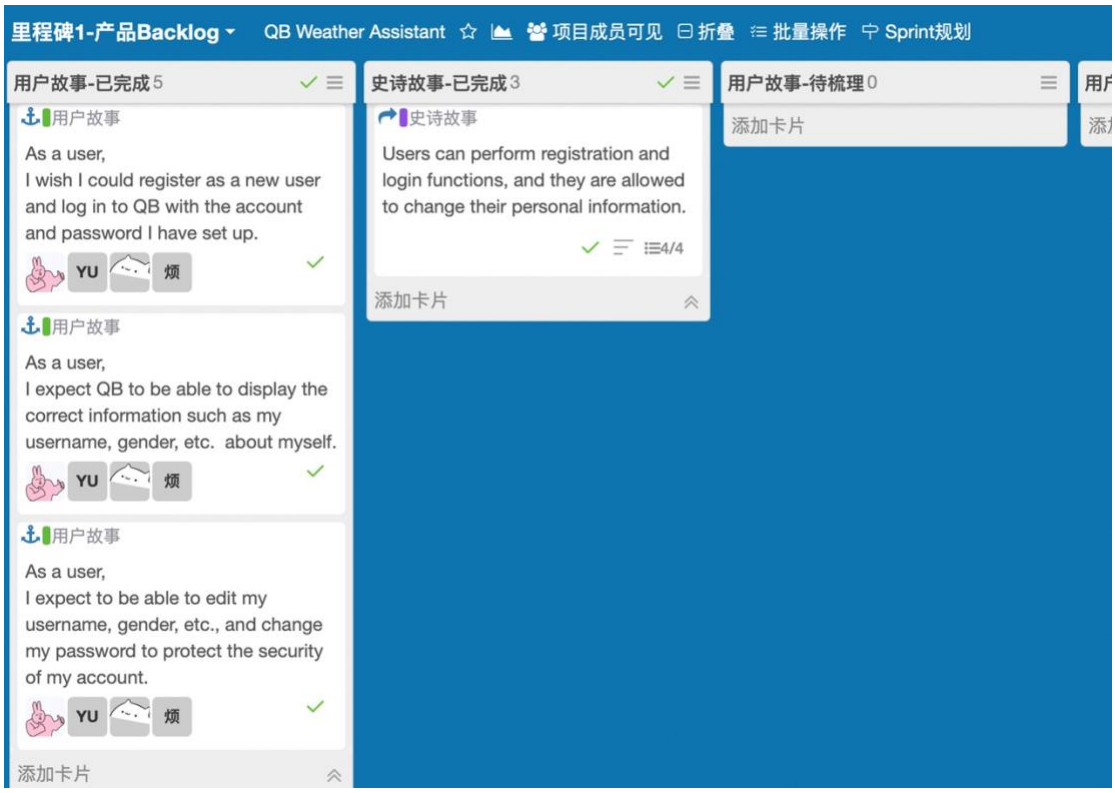
As a user, I would like Weather Manager to provide recommended indices and information for my preferred sports based on the day's weather information.

3.3.3 Product Backlog

The Product Owner is responsible for the Product Backlog and has the right to determine the contents of the Product Backlog, such as what needs to be added to the Product Backlog, which needs to be modified, which needs to be deleted, and which Product Backlog Items need to be adjusted. Most Pbis have real business value to the customer, and some may have no direct business value to the customer. In principle, the PO considers that PBI is valuable to the overall delivery of the product, so it can also be put into PBL.

The most common form of expression for PBI is user stories, but this is not absolute. User stories themselves are not part of the Scrum framework, and Scrum does not require a representation of PBI. Users can use stories, use cases, or other meaningful formats. A good Product Backlog follows the DEEP principle.

We divided all the epic stories in three PBLs, the first PBL mainly implements the three functions of registration and login, panel design and gaining basic weather information.



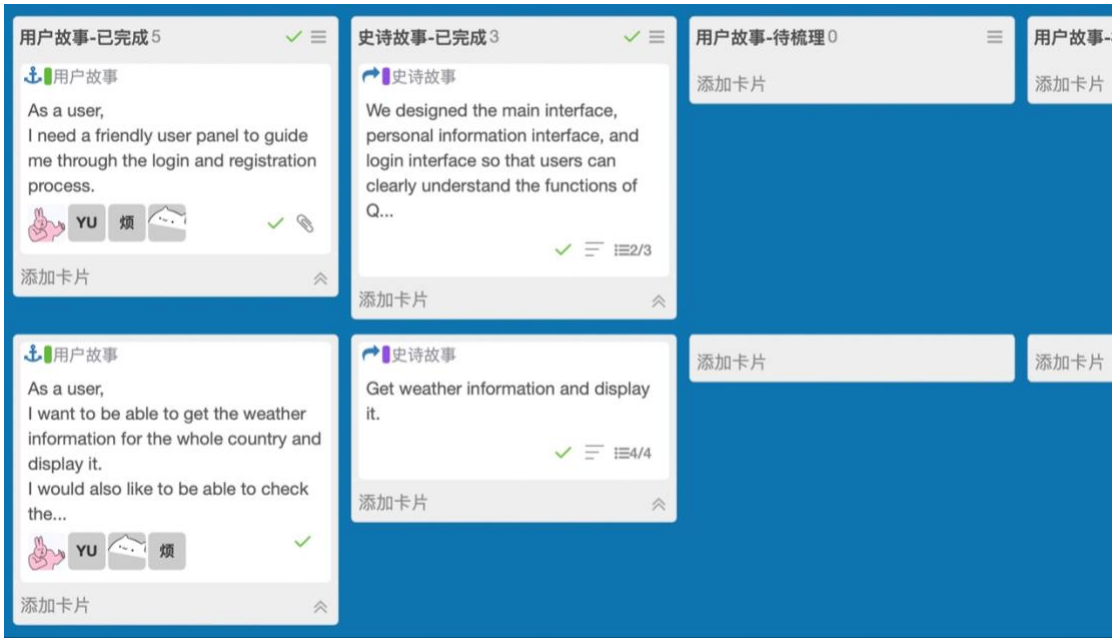


Figure 9 PBL1

The second PBL mainly implements the setting of user's personal preference, the acquisition of ip address and the function of exercise reminder.

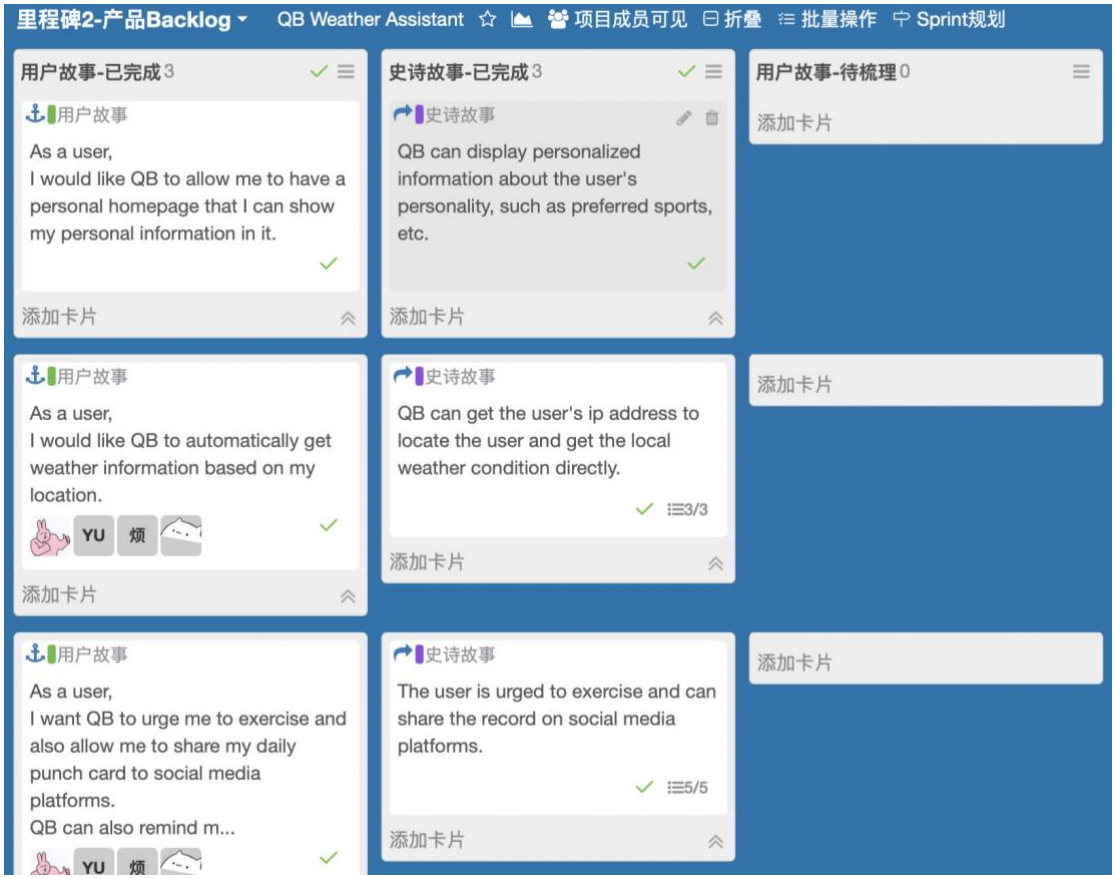


Figure 10 PBL2

The third PBL focuses on designing a simple recommendation algorithm to suggest exercise to users.

Figure 11 PBL3

3.3.4 Sprint 1

(Oct 14th – Oct 31st)

For the first sprint, we have mainly completed these five user stories corresponding to register, login, edit profile and display a user-friendly interface.

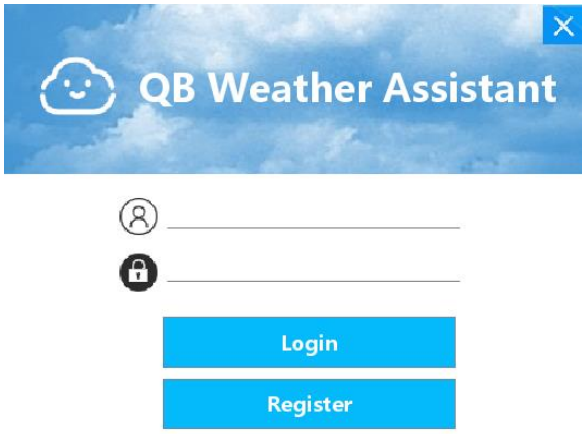


Figure 12 Login Interface

First of all, we provide a simple and easily understandable login interface. This interface can provide at least two functions: login and registration. Then we design the organization of the components of QB. With responsive buttons and keys, users can quickly switch and select what they need in the interface. We also design the user's home page panel, which can display the user's username and other publicly available personal information.

Figure 13.1-2 Homepage

In QB's home page, we rationalize the layout of the main interface so that it can clearly display all the required weather data, such as temperature, weather, humidity, wind, etc. In this page, users can use the interface to get weather information for areas at or below the municipal level. To get the weather information, users need to enter a city code so that QB can show the corresponding weather conditions for the next seven days.

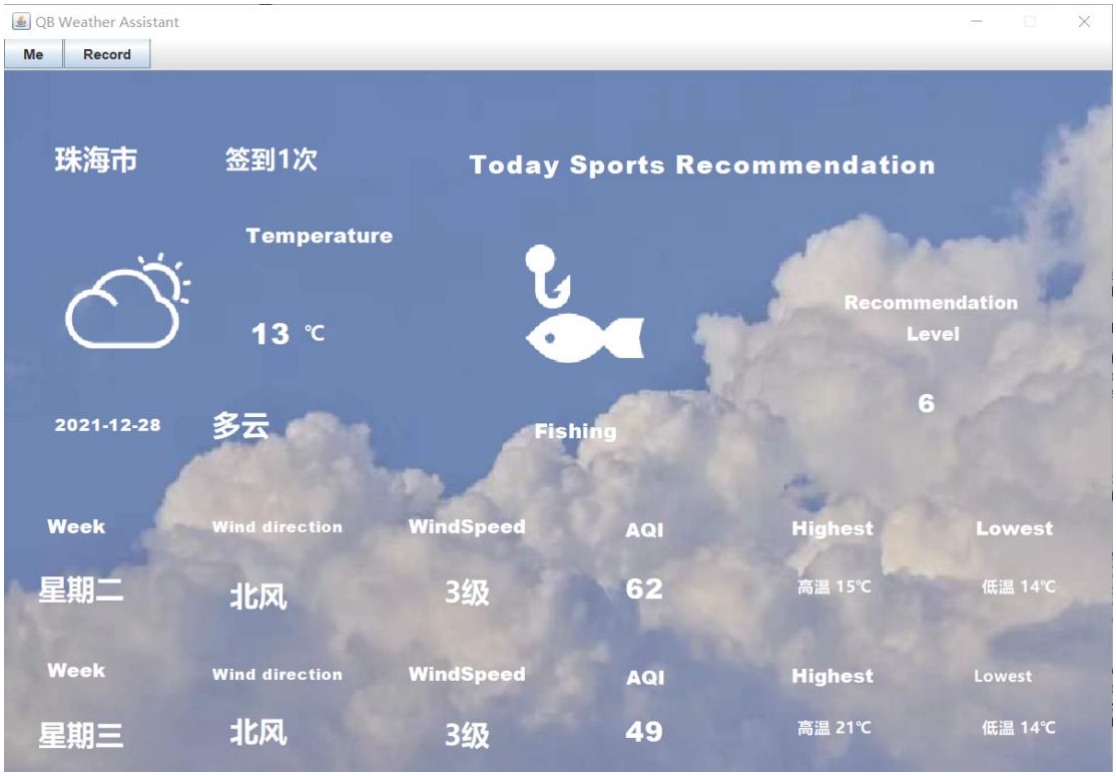
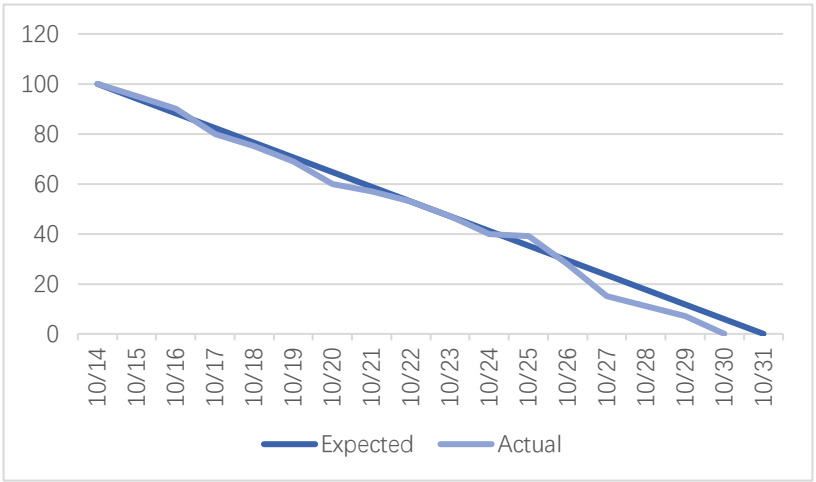


Figure 14 Weather Page

Also, we design a database in which users can change their username and password. After changing their personal data, we expect that the new data can be updated and saved timely.

Here is the burn down diagram:



3.3.5 Sprint 2

(Nov 1st – Nov 14th)

In sprint 2, we mainly implement the function of automatically getting the location. After getting the IP address successfully, QB will analysis the IP address so that the user's location can be easily gotten. With this function, QB's major function, weather forecast, can be used more easily by users. We also add a function that can store the acquired IP address according to the user's needs for the next query.

Apart from that, QB also support recording user's daily exercise attendance data. After recording the attendance, QB can calculate the number of consecutive exercise days and cumulative exercise days for that user and display them to the user. For those who want to maintain a daily exercise routine, attendance reminder is also available for QB. After the users turn on the reminder, QB can remind them to exercise at the set time and record the attendance.

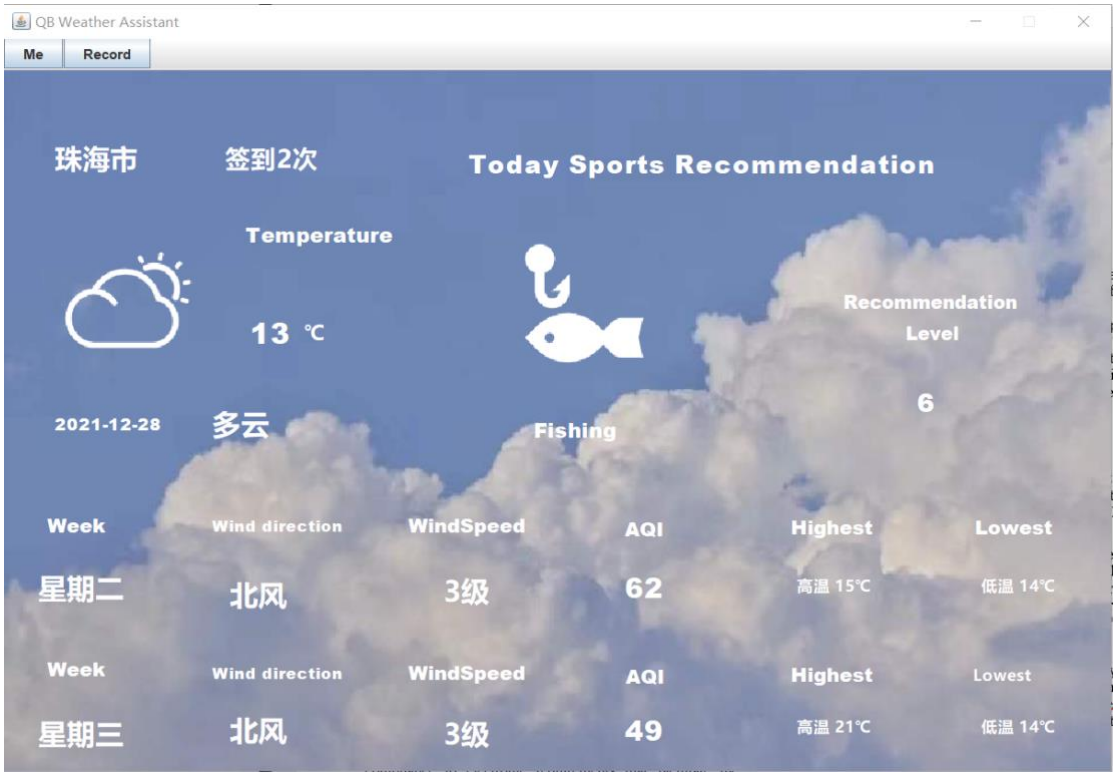
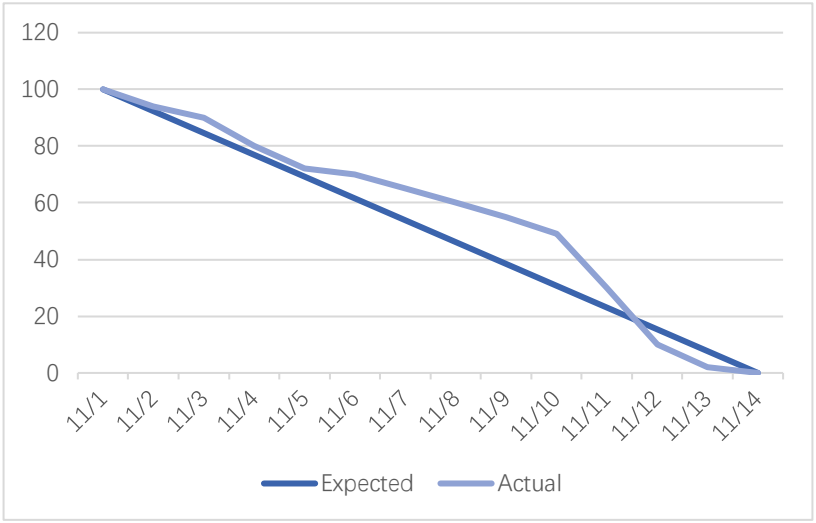


Figure 15 Exercise Attendance

In this iteration, the sharing feature has also been implemented. Users are able to share their attendance information to WeChat, Weibo or the other social media so that not only more people can be aware of their exercise status, but it also can play a role in urging users. Users can also search for sport or outdoor activity in the search bar, QB will launch the activity with the corresponding suitability level.

Here is the burn down diagram:



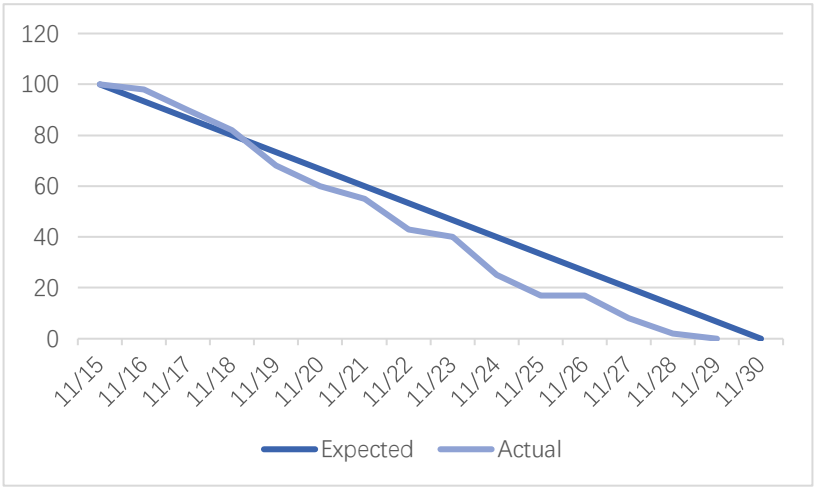
3.3.6 Sprint 3

(Nov 15th – Nov 30th)

In this loop, we mainly implement the exercise index calculation algorithm.

In the initial design, although it has a high accuracy for sunny days, rainy days, cloudy days and other unusual weather, but when it comes to snowy days or skiing and other sports that require a certain amount of snow such complex conditions, it often encounters larger errors (such as directly returning to 0, etc.), so we also consulted some literature to make some improvements in this aspect, and finally came up with the final results, the specific calculation process can be referred to the previous article.[2][3][4][5]

Since the code implementation part is relatively well implemented in this aspect, our progress is still faster than expected after the overall design of the algorithm is completed, which can be referred to our burnout diagram as follows:



4 Testing Document

4.1 Test Case

After our discussion, we decided to test mainly three functions in black-box and white-box testing, namely login and registration and weather forecast city selection, since other functions are difficult to test manually.

For these three functions, we first write three brief test cases:

Test Case 1: Register Test

Input	Output
Valid username and password	Register successfully!
Invalid username	Username is not required, please try again.
Password is too short/long	Password is too short/long, please try again.
Invalid password	Password is not required, please try again.

Test Case 2: Login Test

Input	Output
Username and correct password	Login Successfully!
Username and incorrect password	The password is wrong, please try again.
Invalid username	Username is wrong, please try again
Unregistered username	User doesn't exist, please check again.

Test Case 3: Weather Forecast Test

Input	Output
Correct City Name	Corresponding weather data
Unrecorded City	Error, please try again.

4.2 Black-box Testing

For the black-box testing part, we mainly used the equivalence partition method, the cause-effect graph method, and the decision table method to test all three functions.

Although the basic concept of boundary value analysis is reflected in these functions, the boundary value analysis is covered by the equivalence class division method in this test, so it will not be discussed separately.

4.2.1 Equivalence Partition

We used the equivalence partition to test the normality of the input values (length of input characters, format, etc.) in the three functions mentioned above in a more comprehensive way.

The following are the equivalence classes and test cases that we tested:

Test 1: Register Test

i. Equivalence Class

Input	Valid Equivalence	No.	Invalid Equivalence	No.
Username	1-14 letters and numbers	1	Blank	6
			More than 14 characters	7
			Contains only letters	8
			Contains only numbers	9
			Contains characters other than letters and numbers	10
			Registered username	11
Password	8-20 letters and numbers	2	Blank	12
			Less than 8 characters	13
			More than 20 characters	14
			Contains only letters	15
			Contains only numbers	16
			Contains other special characters	17
Re-enter Password	Same as the first input	3	Blank	18
			Different from the first input	19
Gender	Male or Female	4	Blank	20
Age	1-150 integer numbers	5	Blank	21
			Smaller than 1	22
			Bigger than 150	23
			Not an integer	24
			Contains characters other than numbers	25

ii. Test Case

No.	Input					Expected Output	Coverage Equivalence Class
	Username	Password	Re-enter Password	Gender	Age		
1	Test1	TTtest001	TTtest001	Female	99	Valid	1, 2, 3, 4, 5
2	Blank	TTtest002	TTtest002	Male	100	Invalid	6
3	Teeeeest3333333	TTtest003	TTtest003	Female	38	Invalid	7
4	Test	TTtest004	TTtest004	Male	1	Invalid	8
5	5555	TTtest005	TTtest005	Female	21	Invalid	9
6	Test#6	TTtest006	TTtest006	Male	19	Invalid	10
7	Test1	TTtest007	TTtest007	Female	21	Invalid	11
8	Test8	Blank	TTtest008	Male	19	Invalid	12
9	Test9	test009	test009	Female	21	Invalid	13
10	Test10	TTeeeeeeeeeest000010	TTeeeeeeeeeest000010	Male	19	Invalid	14
11	Test11	Tttestttt	Tttestttt	Female	21	Invalid	15
12	Test12	012345678	012345678	Male	19	Invalid	16
13	Test13	T_e_st##013!	T_e_st##013!	Female	21	Invalid	17
14	Test14	TTtest014	Blank	Male	19	Invalid	18
15	Test15	TTtest015	TTtest0015	Female	21	Invalid	19
16	Test16	TTtest016	TTtest016	Blank	19	Invalid	20
17	Test17	TTtest017	TTtest017	Female	Blank	Invalid	21
18	Test18	TTtest018	TTtest018	Male	-1	Invalid	22
19	Test19	TTtest019	TTtest019	Female	1244	Invalid	23
20	Test20	TTtest020	TTtest020	Male	4.8	Invalid	24
21	Test21	TTtest021	TTtest021	Female	age	Invalid	25

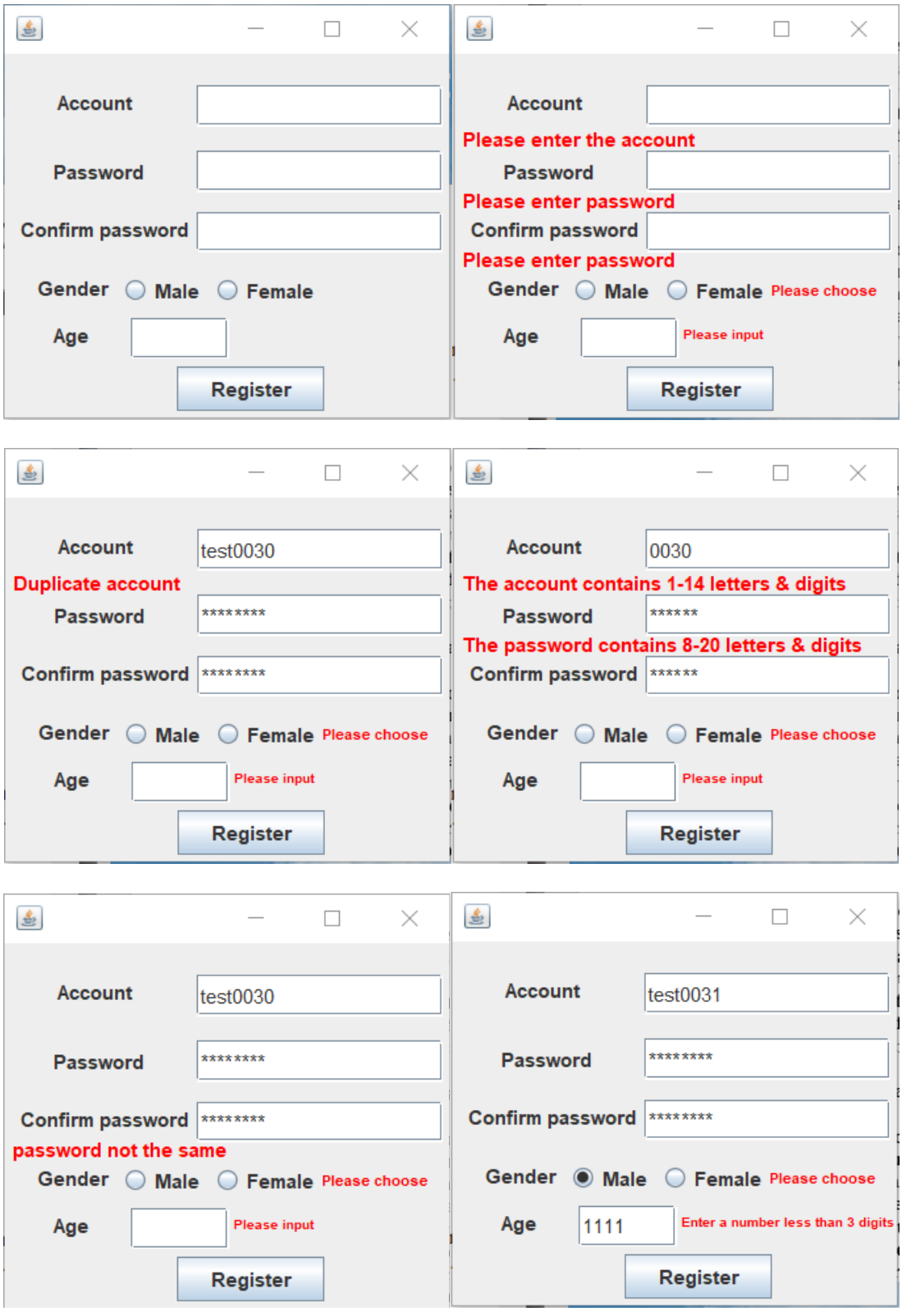


Figure 16.1-6 Register Test

Test 2: Login Test

i. Equivalence Class

Input	Valid Equivalence	No.	Invalid Equivalence	No.
Username	1-14 registered letters and numbers	1	Blank	3
			More than 14 characters	4
			Contains only letters	5
			Contains only numbers	6
			Contains characters other than letters and numbers	7
			Unregistered username	8
Password	8-20 letters and numbers which match the username	2	Blank	9
			Less than 8 characters	10
			More than 20 characters	11
			Contains only letters	12
			Contains only numbers	13
			Contains other special characters	14
			Mismatch with the entered username	15

ii. Test Case

Test Case No.	Input		Expected Output	Coverage Equivalence Class
	Username	Password		
1	Test1	TTtest001	Valid	1, 2
2	Blank	TTtest002	Invalid	3
3	Teeeeest3333333	TTtest003	Invalid	4
4	Test	TTtest004	Invalid	5
5	5555	TTtest005	Invalid	6
6	Test#6	TTtest006	Invalid	7
7	TEST1	TTtest001	Invalid	8
8	Test1	Blank	Invalid	9
9	Test1	Test00001	Invalid	10

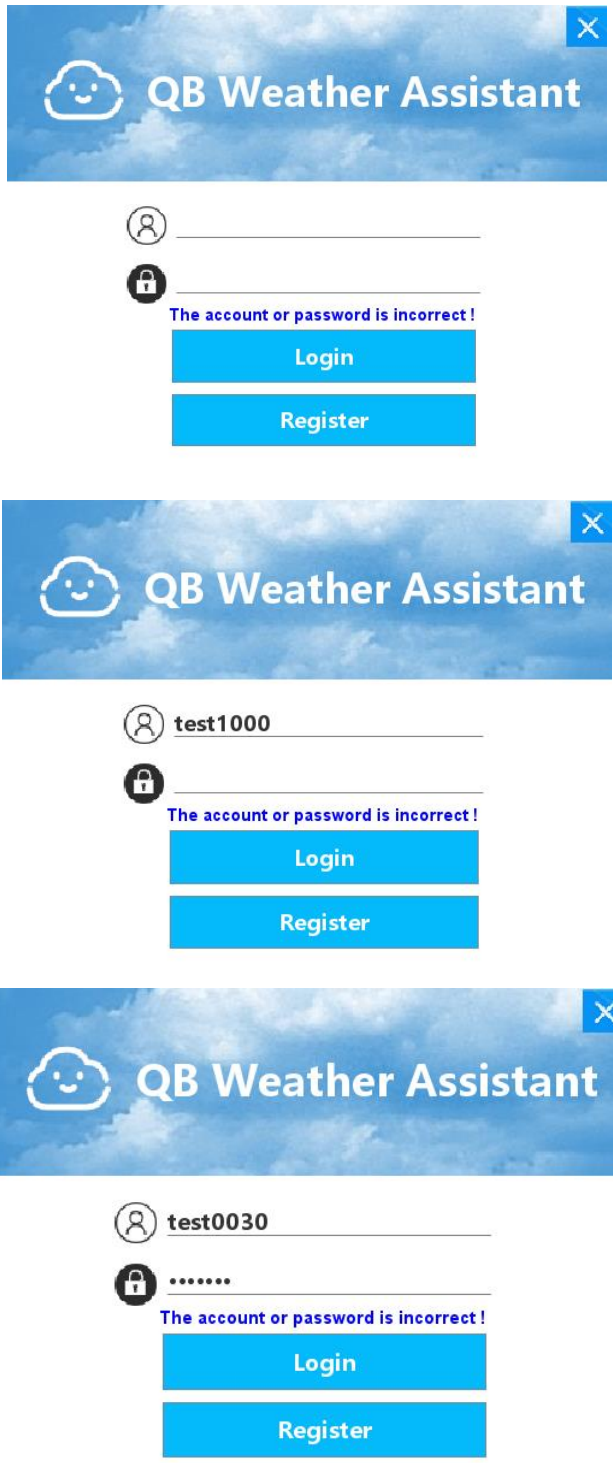


Figure 17.1-3 Login Test

4.2.2 Cause Effect Graph

After testing the equivalence classes, we considered that although most of the cases were covered, the logical relationships between the components were not yet complete, so we conducted cause-and-effect graphs tests for login and registration, and the following are the test cases.

Test Case 1: Register Test

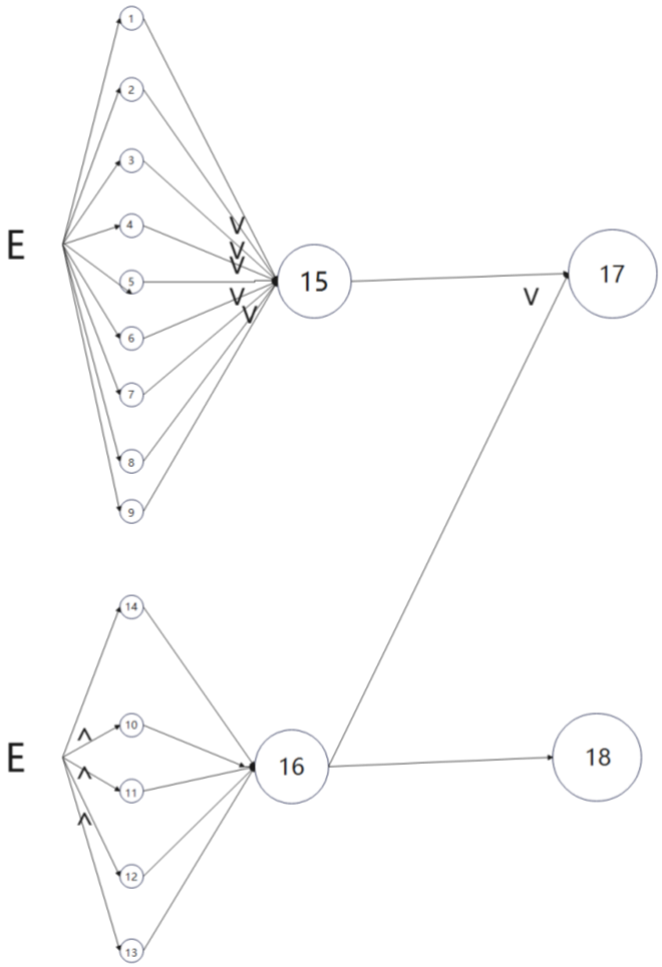


Figure 18 Register Test

1. Account is empty
2. Password is empty
3. Confirm password is empty
4. gender is empty
5. Age is empty
6. Account number format is wrong
7. Wrong password format
8. Two password is not the same
9. Wrong age format
10. Account entered correctly
11. Password correctly entered
12. Two password is the same
13. Gender is entered
14. Age format is correct
15. Click to register
16. Click to register
17. Registration failed
18. Registration is successful

Test Case 2: Login Test

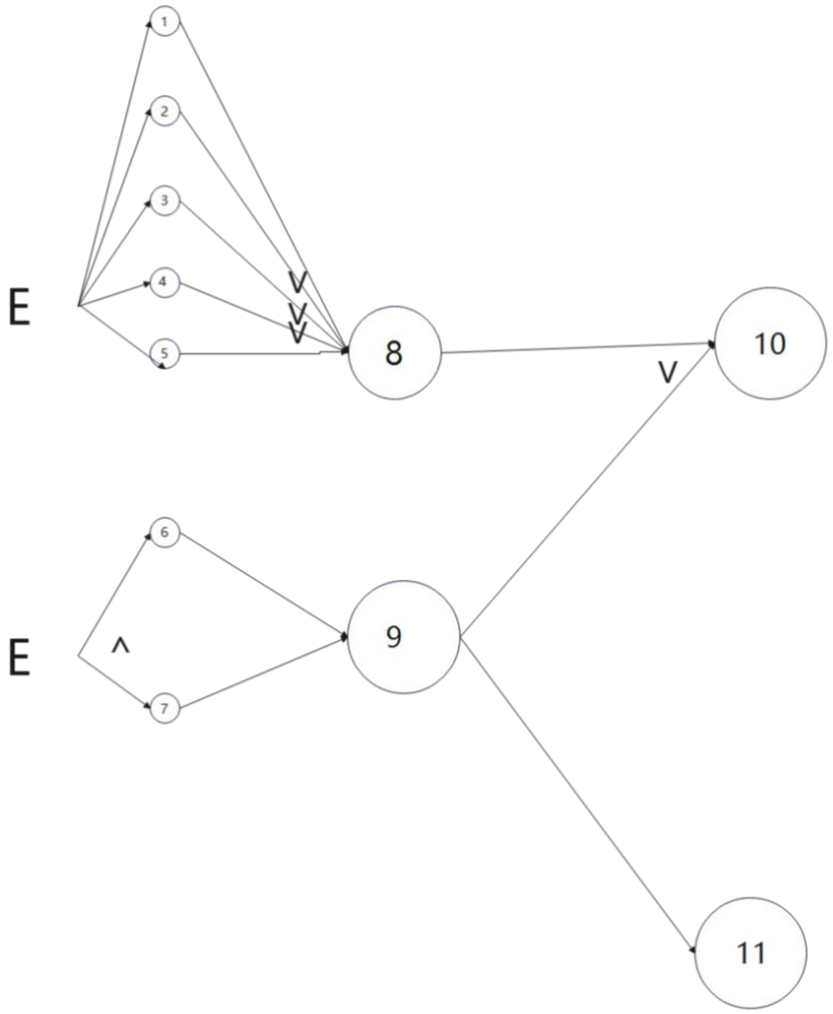


Figure 19 Login Test

1. Account number is empty
2. Password is empty
3. Account not registered
4. Wrong account number
5. Wrong password
6. Account number is correct
7. Password is correct
8. Click to login
9. Click Login
10. Login failed
11. Login successfully

4.2.3 Decision Table

After finishing the cause-and-effect graphs, we also plotted decision tables to have a better understanding of the relationship between the various constraints of the input string.

Test Case 1: Register Test

	1	2	3	4	5	6	7	8	9	10	11	12
C1	F	T	T	T	T	T	T	T	T	T	T	T
C2	-	F	F	F	F	F	F	T	T	T	T	T
C3	-	-	-	-	-	-	-	F	T	T	T	T
C4	-	F	F	F	F	T	T	T	F	T	T	T
C5	-	F	F	F	T	T	T	T	T	F	T	T
C6	-	F	T	T	T	T	T	T	T	T	F	T
A1												√
A2	√											
A3		√	√	√	√	√	√					
A4								√				
A5		√	√	√	√				√			
A6		√	√	√						√		
A7		√									√	

C1: Whether the information is not completed?

C2: Whether the username is entered correctly?

C3: Whether the username is not registered?

C4: Whether the password is entered correctly?

C5: Whether the two times of password input is the same?

C6: Whether the age is entered correctly?

A1: Register successfully.

A2: Information is not completed.

A3: The username is invalid.

A4: The username has been registered.

A5: The password is invalid.

A6: The two password is not the same.

A7: The age is invalid.

Test Case 2: Login Test

	1	2	3	4	5
C1	F	T	T	T	T
C2	-	F	T	T	T
C3	-	-	F	T	T
C4	-	-	-	F	T
A1					√
A2	√				
A3		√	√		
A4				√	

C1: Whether the information is not completed?

C2: Whether the username is entered correctly?

C3: Whether the password is entered correctly?

C4: Whether the username and the password are matched?

A1: Login successfully.

A2: Information is not completed.

A3: The username or password is invalid.

A4: The password is not matched.

4.3 White-box Testing

In the white-box testing part, we mainly test the statement coverage of the password and username part of the registration, draw a control flow diagram based on the code, then generate independent paths, design test cases, and test them according to the designed cases, so that each statement in the core code part is run once to get a complete test result.

Test 1: Password Test

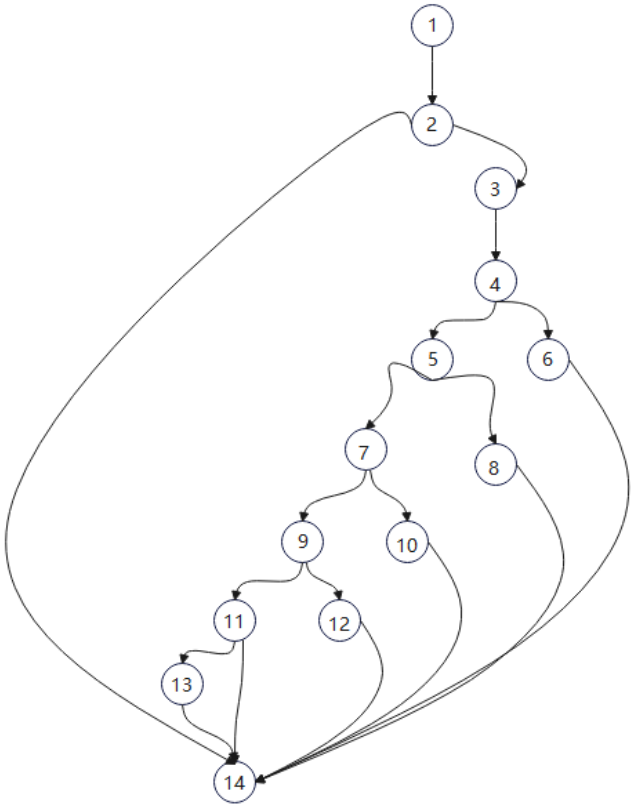


Figure 20 Control Flow Diagram1

From the diagram, we obtain the following independent paths:

1. 1→2→14
2. 1→2→3→4→6→14
3. 1→2→3→4→5→8→14
4. 1→2→3→4→5→7→10→14
5. 1→2→3→4→5→7→9→12→14
6. 1→2→3→4→5→7→9→11→14
7. 1→2→3→4→5→7→9→11→13→14

The complexity is **7** since there are 14 nodes and 19 sides, so the ring complexity is:

$$(19 - 14) + 2 = 7$$

```
if((idisMatch == true||user_name.isEmpty())&&(pisMatch == true||pwd.isEmpty())) {
        label_error_username.setText(null);
        label_error_pwd.setText(null);
        ids.offer("13");
        String dbid = null;

        //检测密码
        if (pwd.isEmpty()&&again_pwd.isEmpty()) {
                label_error_pwd.setText("Please enter password");
                label_error_again_pwd.setText("Please enter password");
        }else if(!pwd.isEmpty()&&again_pwd.isEmpty()) {
                label_error_pwd.setText(null);
                label_error_again_pwd.setText("Please enter password");
        }else if(pwd.isEmpty()&&!again_pwd.isEmpty()) {
                label_error_again_pwd.setText(null);
                label_error_pwd.setText("Please enter password");
        }else if (!pwd.equals(again_pwd)) {
                label_error_again_pwd.setText("password not the same");
                System.out.println("密码:"+pwd+"and"+again_pwd);
        }else if(pwd.equals(again_pwd)&&!again_pwd.isEmpty()&&!pwd.isEmpty()&&!user_name.isEmpty()){
                label_error_pwd.setText(null);
                label_error_again_pwd.setText(null);
                System.out.println(user_name);
                        System.out.println(pwd);
                dbid = dt.Send(user_name, pwd, "SignUp", ids);
        }
```

Figure 21 Code for the Password Part

Therefore, combining the independent path and the code, we designed the following test cases:

Test Case	Input	Overlay path	Expected Output
1	idisMatch = true，user_name=test0001，pwd=1	1→2→14	No output
2	idisMatch = true，user_name=test0001，pwd=null	1→2→14	No output
3	idisMatch = true，user_name=test0001，pwd=null，again_pwd=null	1→2→3→4→6→14	UI dispaly "Please enter password"
4	idisMatch = true，user_name=test0001，pwd=pwd00001，again_pwd=null	1→2→3→4→5→8→14	UI dispaly "Please enter password"under the again_password text label
5	idisMatch = true，user_name=test0001，pwd=null，again_pwd=pwd000001	1→2→3→4→5→7→10→14	UI dispaly "Please enter password"under the password text label

6	idisMatch = true， user_name=test0001， pwd=pwd00001， again_pwd=pwd00002	1→2→3→4→5→7→ 9→12→14	UI dispaly "password not the same"under the again_password text label
7	idisMatch = true， user_name=test0001， pwd=pwd00001， again_pwd=pwd00001	1→2→3→4→5→7→ 9→11→14	Database return “agree”

Test 2: Username Test

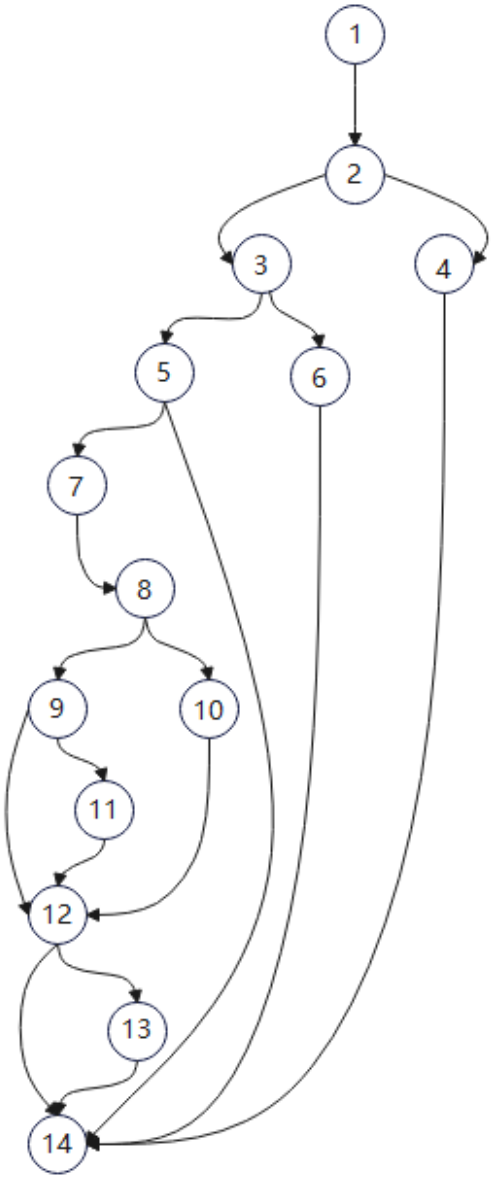


Figure 22 Control Flow Diagram2

```
//检测用户名是否正确
if (user_name.isEmpty()) {
        label_error_username.setText("Please enter the account");
}else if (dbid.trim().equals("reject")) {
        label_error_username.setText("Duplicate account");
}else if (dbid.trim().equals("agree")&&pwd.equals(again_pwd)&&(female.isSelected()
                ||male.isSelected())&&(!user_age.isEmpty()&&ageisMatch == true)) {
        label_error_username.setText(null);
        if(male.isSelected()) {
                label_error_gender.setText(null);
                sex.offer("Male");
                String m = dt.Send(user_name, pwd, "SetGender", sex);
                System.out.println(m);
        }else if(female.isSelected()) {
                label_error_gender.setText(null);
                sex.offer("Female");
                String f = dt.Send(user_name, pwd, "SetGender", sex);
                System.out.println(f);
        }
        if(!user_age.isEmpty()&&ageisMatch == true) {
                label_error_age.setText(null);
                age.offer(user_age);
                String a = dt.Send(user_name, pwd, "SetAge", age);
                System.out.println(a);
                sex.offer("14");
                age.offer("14");
                System.out.println(dt.Send(user_name, pwd, "Gender", sex)+dt.Send(user_name, pwd, "Age", age));
                rf.dispose();
        }

}
```

Figure 23 Code for the Username Part

From the diagram, we obtain the following independent paths:

1. 1→2→4→14
2. 1→2→3→5 →14
3. 1→2 →3→6 →14
4. 1→2→3→5 →7→8→10→12→14
5. 1→2→3→5→7→8 →10→12→13→14
6. 1→2 →3→5→7→ 8→ 9→11→12→13→14
7. 1→2 →3→5→7→ 8→ 9→12→13→14

The complexity is **7** since there are 14 nodes and 19 sides, so the ring complexity is:

$$(19 - 14) + 2 = 7$$

Therefore, combining the independent path and the code, we designed the following test cases:

Test Case	Input	Overlay path	Expected Output
1	user_name=null	1→2→4→14	UI displays "Please enter the account" under the text label
2	user_name is not in the database，pwd=pwd0001 (again_pwd=pwd0002/ gender=null/age=null)	1→2→3→5 →14	UI displays "Please enter password" "Please choose" "Please input"

3	user_name is already in the database	1→2 →3→6 →14	UI displays "Duplicate account" under the text label
4	user_name is not in the database, pwd=pwd0001, again_pwd=pwd0001, gender =male, age=null	1→2→3→5 →7→8→10 →12→14	Database return “agree” UI displays "Please input"
5	user_name is not in the database, pwd=pwd0001, again_pwd=pwd0001, gender =male, age=21	1→2→3→5 →7→8 →10→12→1 3→14	Database return “agree” “agree”
6	user_name is not in the database, pwd=pwd0001, again_pwd=pwd0001, gender =female, age=21	1→2→3→5 →7→8→9→ 11→12→13 →14	Database return “agree” “agree”

4.4 Unit Test

In this part, we first write a simple driver that can call the methods that communicate with the server, then test each of the methods provided by the server, and finally finish the test by checking if the return value is correct.

Here is a part of the code in our driver:

```
import java.util.LinkedList;

public class UnitTestOfDatabase {
    static Queue<String> data = new LinkedList<String>();
    public static void main(String[] args) {
        DataTransfer dt = new DataTransfer();
        UnitTestOfDatabase ut = new UnitTestOfDatabase();
        String name = "UnitTest01";
        String password = "UnitTest01";
        String operation;
        //data-necessary method
        operation="Insert";
        ut.isData("mahjong");
        System.out.println(dt.Send(name, password, operation, data));
        operation="Delete";
        ut.isData("mahjong");
        System.out.println(dt.Send(name, password, operation, data));
        operation="SetAge";
        ut.isData("24");
        System.out.println(dt.Send(name, password, operation, data));
        operation="SetGender";
        ut.isData("man");
        System.out.println(dt.Send(name, password, operation, data));
        operation="DailyCheck";
        ut.isData("12");
        System.out.println(dt.Send(name, password, operation, data));
```

Figure 24 Unit Test

The design of unit test was originally inspired by Junit and other languages with common features of the unit framework. It supports automated testing, using setup and shutdown operations in tests, organizing test cases into suites, and separating tests from reports.

4.5 Automated Testing

Automated testing is a process of converting human-driven testing behavior into machine execution. Usually, after the test cases are designed and reviewed, the testers execute the tests step by step according to the protocols described in the test cases and get a comparison of the actual results with the desired results.

In this process, the concept of automated testing is introduced in order to save manpower, time or hardware resources and improve testing efficiency. And the common tools for the automated testing are QTP, WinRunner, etc.

```
class MysqlTest {
    String name = "Junittest";
    String password = "Junittest";
    Queue<String> test = new LinkedList<String>();
    @Test
    void testMysql() throws ClassNotFoundException, SQLException {
        Mysql Mysql = new Mysql();
        test.offer("10");
    }

    @Test
    void testSearch() throws ClassNotFoundException, SQLException {
        Mysql Mysql = new Mysql();
        Mysql.Search(name,password);
    }

    @Test
    void testGender() throws ClassNotFoundException, SQLException {
        Mysql Mysql = new Mysql();
        Mysql.Gender(name, password);
    }

    @Test
    void testSetGender() throws ClassNotFoundException, SQLException  {
        Mysql Mysql = new Mysql();
        Mysql.SetGender(name, password, test);
    }

    @Test
    void testAge() throws ClassNotFoundException, SQLException  {
        Mysql Mysql = new Mysql();
        Mysql.SetGender(name, password, test);
    }

    @Test
    void testSetAge() throws ClassNotFoundException, SQLException  {
        Mysql Mysql = new Mysql();
        Mysql.SetAge(name, password, test);
    }

    @Test
    void testSignUp() throws ClassNotFoundException, SQLException  {
        Mysql Mysql = new Mysql();
        Mysql.SignUp(name, password);
    }

    @Test
    void testInsert() throws ClassNotFoundException, SQLException  {
        Mysql Mysql = new Mysql();
        Mysql.Insert(name, password, test);
    }

    @Test
```

Figure 25 Code for Automated Test

For our current project, we still use our own written drivers for automation testing, which is equivalent to an extension and expansion of the unit tests we have done

before, to ensure that fewer errors will appear in QB, so that it can be more in line with our expectations. And the main part of the code is shown above.

After getting the results, we found that all the methods passed the test fluently, i.e. they did not have any major bugs in this part.

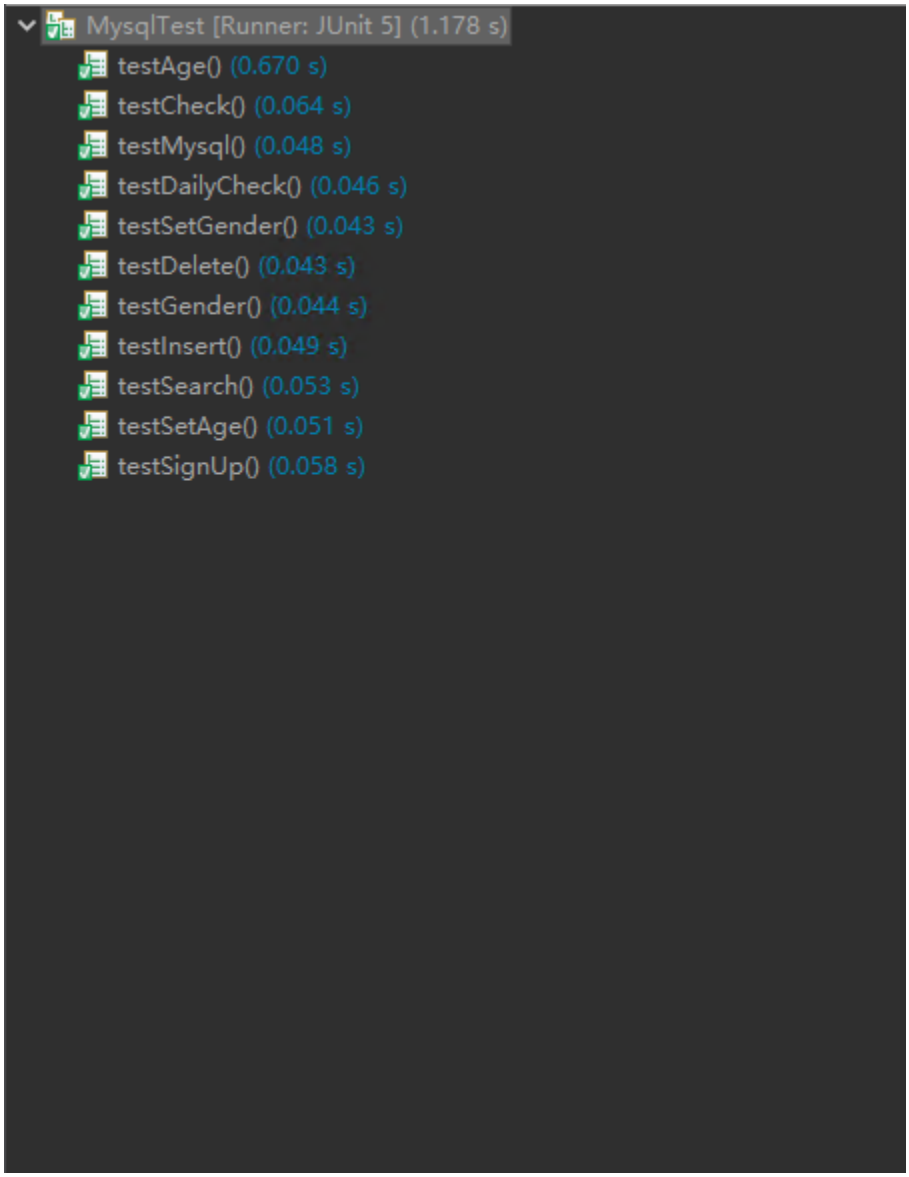


Figure 26 Automated Test

And after that, each of us acted as a user and operated the QB ourselves to troubleshoot any problems we might encounter along the way. In addition, we also randomly selected a few users to try it out and collect their feedback for further improvement, which we will not go into here.

5 Management and Maintenance Document

Software management involves all aspects of software development, and its targets include software developers, project funds, and the software itself.

Software maintenance refers to modifying software by adding new features after the software system has been delivered to correct operational errors in the software or to meet new user requirements.

Our team manages and maintains the software after development and testing, hoping to provide a better system environment for our customers.

5.1 Software Management

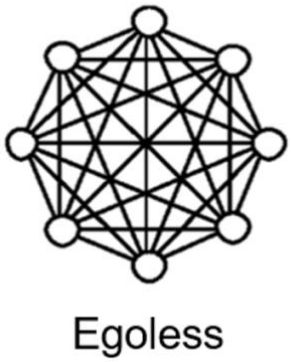


Figure 27 Egoless Model

In this development process, we used egoless team model, which means in our team everybody is equal, and the team works together to achieve a common goal. Meanwhile, our decisions are made by consensus. This model is the most suited one to our projects that we can think about, we believed that we could deal with all the challenges we may meet, and we really did it!

Also, we draw a Gantt chart present the start and end dates of each software engineering task.

According to the Gantt chart, we can easily get out development progress such as: The project lasted 14 weeks in total. We have 3 Sprints in total, each of them took two weeks. Our testing started from the 10th week throughout, and the final maintenance

started from the 12nd week, our review, presentation preparing and report writing all took three weeks each.

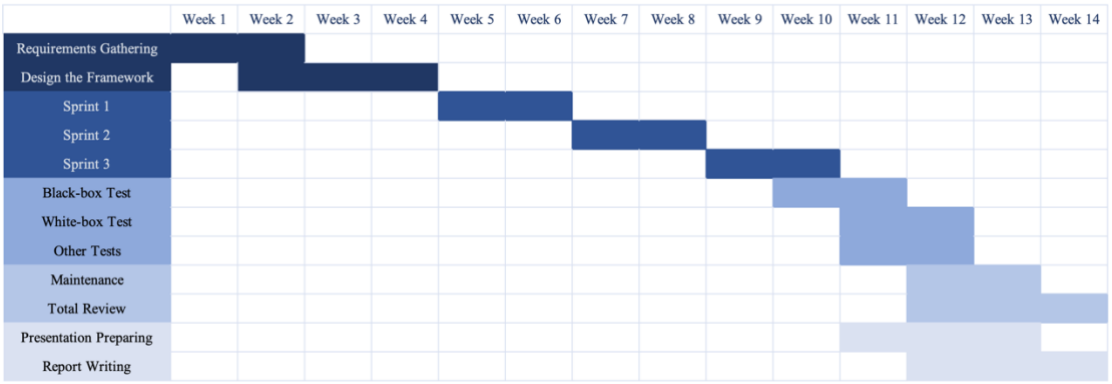


Figure 28 Gantt Chart

5.2 Software Maintenance

During the maintenance process, we always make sure that the code and Leangoo are updated and consistent at the same time. Ensure that change control processes and version control processes are in place and that product changes during maintenance are managed according to these processes.

In addition to this, we make every effort to ensure that changes to the code still meet coding specifications, pass code reviews, and do not result in junk or redundant code, making the code clear and easy to understand for other team members to perform other tasks such as testing and more efficient collaboration.

At the same time, given that the CMMI assessment uses the SCAMPI methodology, which helps an organization to have a comprehensive understanding of its own process capability or organizational maturity, we have reviewed the relevant information and conducted a suggested, approximate assessment of our own small team, and believe that we are now roughly at CMMI Level I.[6]

6 Reference

[1]. Xiao, G. (2008, January 14). *Shooting Weather Index*. Sina Blog. http://blog.sina.cn/dpool/blog/s/blog_5061b97101008b5n.html

[2]. Man, Y., & Yuxiang, A. (2015). Effect of different temperature and humidity environments on physical exertion of athletes. *Contemporary Sports Technology (33)*, 2.

[3]. Jingshu, Z., & Yuxiang, A. (2012). The effect of environmental temperature and humidity on exercise capacity. *Science and Technology Innovation Herald (10)*, 154.

[4]. Lan, Q. (2017). Effects of cold and windy weather on outdoor physical education classes in colleges and universities and countermeasures. *Contemporary Sports Technology*, *7*(25), 2.

[5]. Li, H., Ju, C., Wang, Y., Jia, Z., & Feng, Y. J. (2015). An analysis of the effects of physical exercise on physical health in hazy weather. *Sports Time*, *000*(009), 31.

[6]. Zheng D. Tan L. Gu Q. Chen Thu C. (2006). A small software process self-assessment tool based on CMMI. *Computer Science*, *33*(2), 263-265.