Testing Reports

Revision History:

|  |  |  |
| --- | --- | --- |
| Date | Author | Description |
| 2021-05-15 | Egbert Ding | Database Testing case |
| 2021-05-15 | Bryce Shi | Testing plan |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Contents

[1. Introduction 3](#_Toc41054770)

[1.1. Intended Audience and Purpose 3](#_Toc41054771)

[1.2    How to use the document 3](#_Toc41054772)

[2. Testing Cases 3](#_Toc41054773)

[2.1. Server 3](#_Toc41054774)

[2.2. Client 3](#_Toc41054775)

[2.3. Map 3](#_Toc41054776)

[2.4. Wi-Fi Fingerprint 3](#_Toc41054777)

[2.5. Database 3](#_Toc41054778)

[2.6. Algorithm 3](#_Toc41054779)

[3. Testing Plan 3](#_Toc41054780)

[3.1. Register 3](#_Toc41054781)

[3.2. Upload Map 3](#_Toc41054782)

[3.3. … 3](#_Toc41054783)

[4. Testing Results 4](#_Toc41054784)

## Introduction

## Intended Audience and Purpose

This document provides the testing method and results, corresponding to the requirement from the customer. It consists of 3 parts, the testing cases, the test plan, and the testing results.

## 1.2    How to use the document

You may refer to the content section for the structure of the document, in which Sec. Testing Cases collect the unit and module test information from each team; Sec. Testing Plan shows the steps and expected results of the integration test; Sec. Results describes the real world data out of the test, and the correspondence to the requirements.

## Testing Cases

In this section, each team propose their testing cases on unit and module testing.

## Server

## Client

## Map

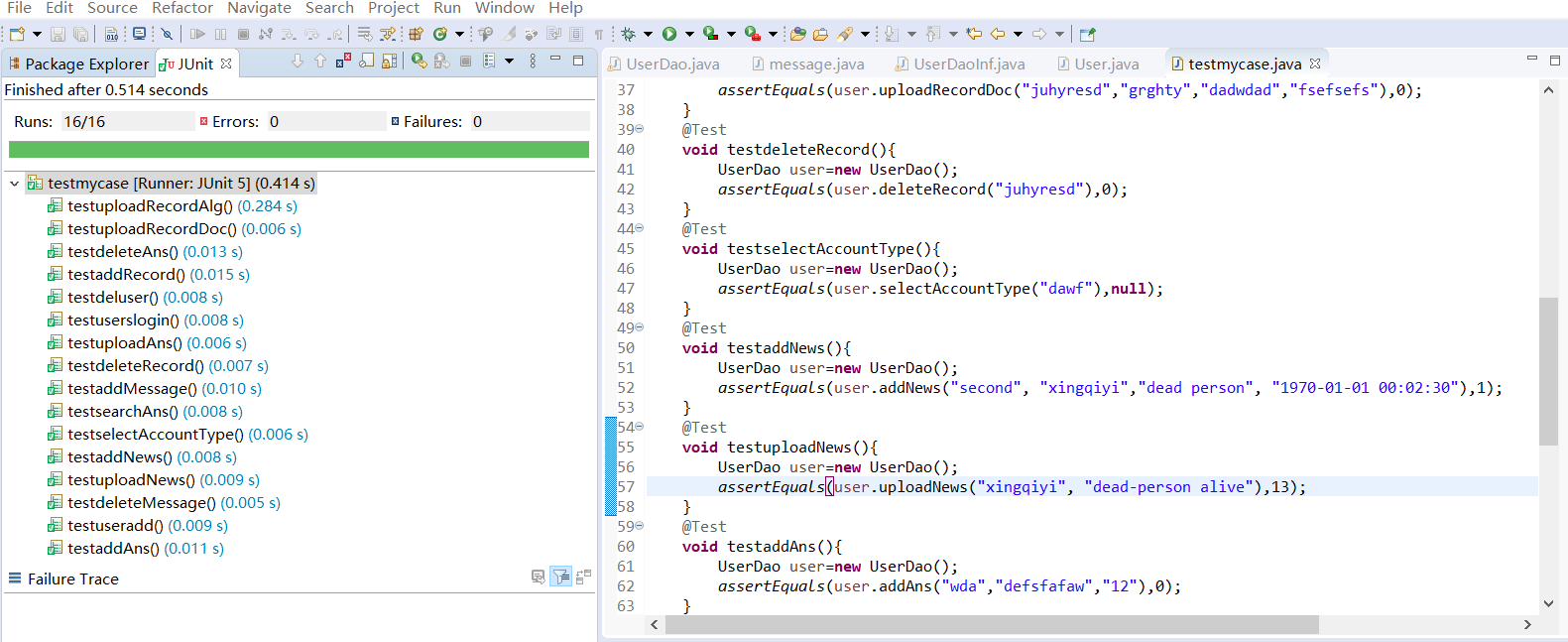
## Wi-Fi Fingerprint

## Database

Import the dependency packages needed for JUnit testing. First, create a java project; Import JUnit package and hamcrest-core package into the project. Then, create a resource folder to store resource files and create a source folder resource folder. Write java file corresponding to the test class, the general naming method: test + test class or test class name + test. Finally, Run test methods or test classes.

JUnit can't support concurrent testing, but it can do so with a grobautils core. First of all, the dependency is introduced for single test, and then the concurrent test array is established for concurrent test.

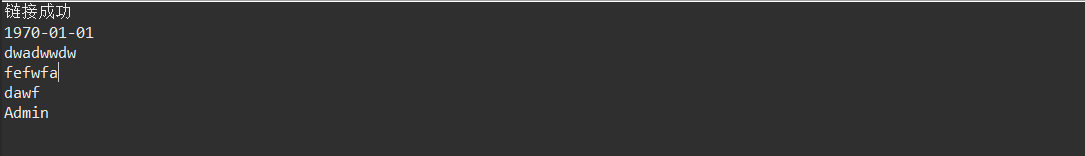
Junit testing result:



1. addUser

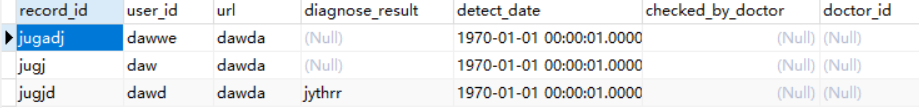
Create the test class named testadduser. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, name,birthday, phone\_number, email\_address, credential, user\_password, and

account\_type, and the expected output is the success of add a user in the table.



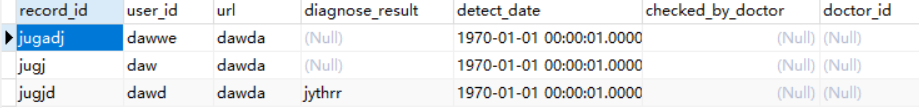
1. addRecord

Create the test class named testaddrecord. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, user\_id, url, detect\_date, and the expected output is the success of add a record in the table.



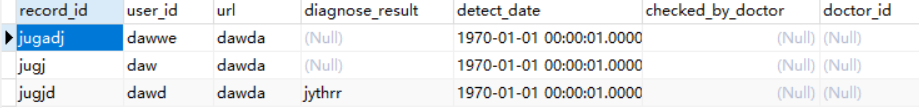
1. uploadRecordAlg

Create the test class named testuploadRecordAIg. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, upload\_record\_id, result, url, and the expected output is the success of adding an AIrecord in the table.



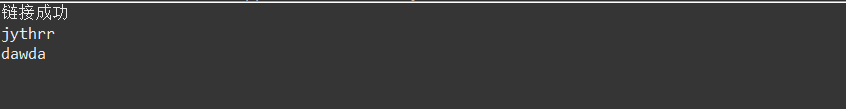
1. uploadRecordDoc

Create the test class named testuploadRecordDoc. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, upload\_record\_id, doctor\_id, url, result, and the expected output is the success of uploading a recorddoc in the table.



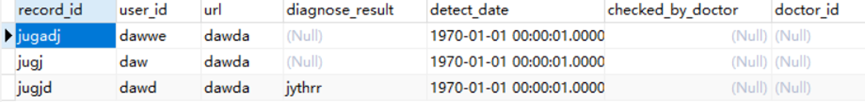
1. selectRecord

Create the test class named testselectRecord. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, start\_date, end\_date, and the expected output is the success of seleting record in the table,



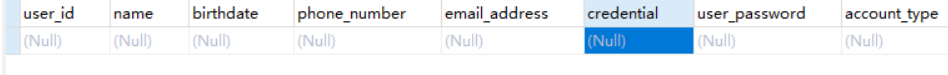
1. deleteRecord

Create the test class named testdeleteRecord. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, upload\_record\_id, and the expected output is the success of deleteing record in the table.



1. deleteUser

Create the test class named testdeleteUser. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, user\_id, and the expected output is the success of deleteing user in the table.



1. addNews

Create the test class named testaddNews. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, title, content, time, and the expected output is the success of adding news in the table.



1. deleteNews

Create the test class named testdeleteNews. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, title, and the expected output is the success of deleteing news in the table.



1. uploadNews

Create the test class named testuploadNews. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, title, and the expected output is the success of uploading news in the table.



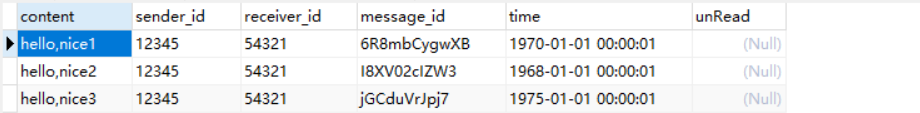
1. selectNews

Create the test class named testselectNews. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, sender\_id, receiver\_id, content, time, and the expected output is the success of selecting news in the table.



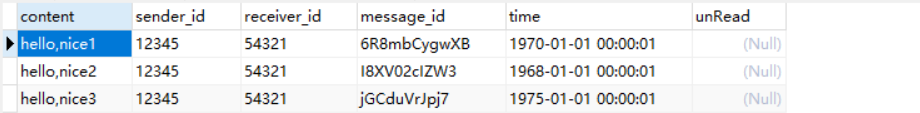
1. addMessage

Create the test class named testaddMessage. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, sender\_id, receiver\_id, content, time, and the expected output is the success of selecting messages in the table.



1. deleteMessage

Create the test class named testdeleteMessage. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, mess\_id , and the expected output is the success of deleteing messages in the table.



1. searchMessagebyTime

Create the test class named testsearchMessagebyTime. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, user\_id1 , user\_id2, start-date, end-date, and the expected output is the success of selecting messages by time in the table.



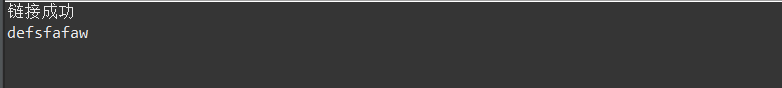
1. searchMessagebyNum

Create the test class named testsearchMessagebyNum. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, user\_id1, user\_id2, n, and the expected output is the success of selecting messages by time in the table.



1. searchAns

Create the test class named testsearchAns. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, keyword, and the expected output is the success of selecting answer in the table.



1. addAns

Create the test class named testaddAns. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, keyword, ans, and the expected output is the success of adding answer in the table.



1. deleteAns

Create the test class named testdeleteAns. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, ans\_id , and the expected output is the success of deleteing answer in the table. 

1. uploadAns

Create the test class named testuploadAns. For each method in the test class, use@ before@after Test each module one by one. The input is the same as described in the IS document, keyword, ans, and the expected output is the success of uploading answer in the table.



## Algorithm

## Testing Plan

Here comes the complete testing plan for integration, referring to the workflows in the system design document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case No.** | **Test item** | **Desired outcom** | **Data flow** |
| 1 | Register an account | The user’s account can be creat and the data be saved in the database | Client(String name, String birthdate, String phone\_number, String email\_address, String credential, String user password)>Server(String user\_id, String name, String birthdate, String phone\_number, String email\_address, String credential, String user password, String account\_type)>DB(String user\_id, String name, datetime birthdate, String phone\_number, String email\_address, String credential, String user password, String account\_type)>Server(int i)>Client(int i) |
| 2 | User login | The user can login the system | Client(String info)>Server(String info)>DB(String info)  >Server(String info)>Cient |
| 3 | Change password | Account’s password be changed successfully | Client(String user\_id, String new\_password)>Server(String user\_id,String new\_password)>DB(String user\_id,String new\_password) |
| 4 | (AD)Search user | Return the user’s information | Client(String info, String password)>Server(String info, String password)>DB(String info, String password)  >Server(List<User>)>Cient(List<User>) |
| 5 | (AD) Modify User Profile | User’s profile be changed | Client(String user\_id, String name, String birthdate, String phone\_number, String email\_address, String credential, String user\_password, String account\_type)>Server(String user\_id, String name, String birthdate,String phone\_number,String email\_address,String credential,String user\_password,String account\_type)>DB(String user\_id, String name, datetime birthdate,String phone\_number,String email\_address,String credential,String user\_password,String account\_type) |
| 6 | (AD)delete specific users | User be deleted | Client(String user\_id)>Server(String user\_id)>DB(String user\_id)  >Server(int i)>Cient(int i) |
| 7 | Upload Image | Images through the server be uploaded to the database | Client(image)>Server(String record\_id, String user\_id, String url, String detect\_date)>DB(String record\_id, String user\_id, String url, datetime detect\_date)  >Server(int i)>Cient(int i) |
| 8 | (AD) Manage Algorithms | Manage Algorithms successfully | Client(String)>Server(String)> Algorithms (String) |
| 9 | Find Upload Records | Return the records | Client(String user\_id, String start\_date, String end\_date)>Server(String user\_id, String start\_date, String end\_date)>DB(String user\_id, datetime start\_date, datetime end\_date)  >Server(List<record>)>Cient(List<record>) |
| 10 | Revise Treatment | The doctor can revise the treatment | Client(String record\_id ,String doctor\_id, String url, String doctor\_result)>Server(String record\_id, String doctor\_id, String url, String doctor\_result)>DB(String record\_id, String doctor\_id, String url, String doctor\_result)  >Server(int i)>Cient(int i) |
| 11 | (AD)Find Upload Records | Return the records | Client(String user\_id, String start\_date, String end\_date)>Server(String user\_id, String start\_date, String end\_date)>DB(String user\_id, datetime start\_date, datetime end\_date)  >Server(List<record>)>Cient(List<record>) |
| 12 | (AD)Modify Records | The records be modified | Client(String record\_id,String diagnose\_result,String url)>Server(String record\_id,String diagnose\_result,String url)>DB(String record\_id,String diagnose\_result,String url)  >Server(int i)>Cient(int i) |
| 13 | (AD)Delete records | The records be deleted | Client(String record\_id)>Server(String record\_id)>DB(String record\_id)  >Server(int i)>Cient(int i) |
| 14 | Search Doctors | Find the doctor | Client(String info, String password)>Server(String info, String password)>DB(String info, String password)>Server(List<User>)>Cient(List<User>) |
| 15 | Send Message | Send message successfully | Client(String content)>Server(String sender\_id,String receiver\_id,String content,String time)>DB(String sender\_id,String receiver\_id,String content,datetime time)  >Server(int i)>Cient(int i) |
| 16 | Find Messages | Return the messages | Bytime：Client(String user1\_id,String user2\_id,String start\_date,String end\_date)>Server(String user1\_id,String user2\_id,String start\_date,String end\_date)>DB(String user1\_id,String user2\_id,datetime start\_date, datetime end\_date)>Server(List<message>)>Cient(List<message>)  Bynum:  Client(String user1\_id,String user2\_id,int n)>Server(String user1\_id,String user2\_id,int n)>DB(String user1\_id,String user2\_id,int n)  >Server(List<message>)>Cient(List<message>) |
| 17 | (AD) Find Messages | Return the messages | Bytime：Client(String user1\_id,String user2\_id,String start\_date,String end\_date)>Server(String user1\_id,String user2\_id,String start\_date,String end\_date)>DB(String user1\_id,String user2\_id,datetime start\_date, datetime end\_date)>Server(List<message>)>Cient(List<message>)  Bynum:  Client(String user1\_id,String user2\_id,int n)>Server(String user1\_id,String user2\_id,int n)>DB(String user1\_id,String user2\_id,int n)  >Server(List<message>)>Cient(List<message>) |
| 18 | (AD) Delete Messages | Delete messages successfully | Client(String message\_id)>Server(String message\_id)>DB(String message\_id)  >Server(String message\_id)>Cient(String message\_id) |
| 19 | (AD)Add News | News added successfully | Client(String news\_id,String news\_title, String news\_content, String news\_time)>Server(String news\_id,String news\_title, String news\_content, String news\_time)>DB(String news\_id,String news\_title, String news\_content, datetime news\_time)  > Server(int i)>Cient(int i) |
| 20 | (AD)Delete News | News be deleted | Client(String news\_title)>Server(String news\_title)>DB(String news\_title)  > Server(int i)>Cient(int i) |
| 21 | (AD)Update News | News be updated | Client(String news\_title,String news\_content)>Server(String news\_title,String news\_content)>DB(String news\_title,String news\_content)> Server(int i)>Cient(int i) |
| 22 | Search News | Search news by time | Client(String time)>Server(String time)>DB(String time)>Server(List <news>)>Cient(List <news>) |
| 23 | (AD)Create Backup | Make a backup of all data | Client(String address)>Server(String address)>DB(String address)> Server(int i)>Cient(int i) |
| 24 | (AD)Find Backups | Return the backup | Client(String address)>Server(String address)>DB(String address) |
| 25 | (AD)Restore Backups | Restore the backup successfully | Client(String address)>Server(String address)>DB(String address)  > Server(all data)>Cient(all data) |

## Testing Results

The results of the integration are listed here and you may find the correspondence to the requirements in the requirement analysist document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case No.** | **Module** | **Result** | **Corresponding Requirement** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |