Online Flight Ticket System

Test report

Text

Description automatically generated

**Table of Content:**

1. **Hierarchy Diagram**
2. **Testing Strategy**
3. **Test Cases**
   1. **Unit Testing**
   2. **Integration Testing**
   3. **System Testing**
4. **Testing coverage and Analysis**
5. **Hierarchy Diagram**

Diagram

Description automatically generated

1. **Testing Strategy**

Bottom-up test approach is mainly adopted in our system. The advantages are obvious as our class structure is quite complex overall, it would be a tough task to stub loads of dependent classes or methods. Therefore, breaking it down and starting from the bottom can make our test procedure as simple and systematic as possible. Moreover, it is easy to find out problems if any since bottom-up is a gradual approach and test cases are organized in adjacent layers and are closely related.

However, disadvantages also exist. With the Database module and the Station-Edge-Line module being mutually related, it is hard to tell which one should be at the bottom layer. After deliberation, we put Database the top layer as the concept of having a Database class is to gain a high-level control of all the other data.

So, for the modules mentioned above, we used the sandwich approach. Therefore, we create a unique testing method for our system with a local sandwich approach in a global bottom-up strategy.

1. **Test Cases (Total: 62 test cases)**
   1. **Unit Testing**

**1)FlightTest.java**

Table

Description automatically generated

**Test analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testQueryAllFail** | Test query all flights when sql exception happen in method executeQuery of statement |
| **testQueryAllFail1** | Test query all flights when sql exception happen in method next of resultset |
| **testQueryAllPass1** | Test valid query all flights with no data |
| **testQueryAllPass2** | Test valid query all flights with exist data |
| **testQueryByIdFail** | Test query flight by id when sql exception happen in method executeQuery of statement |
| **testQueryByDeptPass1** | Test query flight by dept with no data |
| **testQueryByDeptPass2** | Test query flight by dept with exist data |
| **testQueryByDeptFail** | Test query flight by dept when sql exception happen in method executeQuery of statement |
| **testQueryByDestPass1** | Test query flight by dest with no data |
| **testQueryByDestPass2** | Test query flight by dest with exist data |
| **testQueryByDestFail** | Test query flight by dest when sql exception happen in method executeQuery of statement |
| **testQueryByDeptAndDestPass1** | Test query flight by dept and dest with no data |
| **testQueryByDeptAndDestPass2** | Test query flight by dept and dest with exist data |
| **testQueryByDeptAndDestFail** | Test query flight by dept and dest when sql exception happen in method executeQuery of statement |
| **testQueryByIdPass1** | Test query flight by id with no data |
| **testQueryByIdPass2** | Test query flight by id with exist data |
| **testDeleteFail** | Test delete flight when sql exception happen in method executeQuery of statement |
| **testDeletePass1** | Test delete flight which is not exist in db |
| **testDeletePass2** | Test delete flight which is exist in db |
| **testUpdateFail1** | Test update flight which is not exist in db |
| **testUpdateFail2** | Test update flight which is available seat is not enough |
| **testUpdateFail3** | Test update flight when sql exception happen in method executeQuery of statement |
| **testUpdatePass** | Test valid update flight |

**2)AdminTest.java**

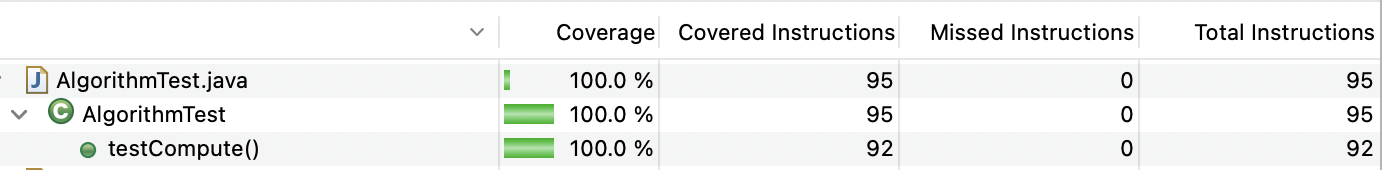
Table

Description automatically generated

**Test case analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testLoginFailed** | Test invalid admin name or password |
| **testGetUserOrderFail** | Test sql exception happened when get user order |
| **testGetUserOrderPass** | Test valid order get for user |
| **testGetAllOrderFail** | Test sql exception happened when get all order |
| **testGetAllOrderPass** | Test valid order get for all orders |

**3)AlgorithmTest.java**



**Test case analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testCompute** | Test the algorithm of route compute for flights |

* 1. **Integration Testing**

**1)ComparatorTest.java**

Table

Description automatically generated

**Test case analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testGetComparator** | Test get comparator with different type |
| **testPriceComparator** | Test compare flight with price |
| **testTakeoffTimeComparator** | Test compare flight with takeoff time |

**2)PeopleTest.java**

Table

Description automatically generated

**Test analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testRegisterFailed1** | Test register user which already exist |
| **testRegisterFailed2** | Test sql exception happened when register |
| **testRegisterUserPass** | Test valid register |

**3) SearchTest.java**

Table

Description automatically generated

**Test analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testSearchRoute1** | Test only single search |
| **testSearchRoute1** | Test no single search |

**4)UserTest.java**

Table

Description automatically generated

**Test analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testAddOrderFailed1** | Test add order with invalid flight id |
| **testAddOrderFailed2** | Test add order when sql exception happen |
| **testAddOrderFailed3** | Test add order with flight which is not exist |
| **testAddOrderFailed4** | Test add order with flight which available seat is not enough |
| **testAddOrderPass** | Test add valid order |
| **testCancelOrderFailed1** | Test cancel order which is not exist |
| **testCancelOrderFailed2** | Test cancel order with flight which available seat is invalid |
| **testCancelOrderPassed** | Test valid cancel order |
| **testDeleteFailed1** | Test delete user which is not exist |
| **testDeleteFailed2** | Test delete user when sql exception happen |
| **testDeleteUserPass** | Test valid delete user |
| **testLoginFailed1** | Test login with user does not exist |
| **testLoginFailed2** | Test login when sql exception happen |
| **testLoginPass** | Test valid login |
| **testChangePwdFailed1** | Test change password which user does not exist |
| **testChangePwdFailed2** | Test change password when sql exception happen |
| **testChangePwdPass** | Test valid change password |
| **testGetOrderFailed** | Test get order when sql exception happen |
| **testGetOrderPass** | Test valid get order |

* 1. **System Tesing**

**1)TicketSystemTest.java**

Table

Description automatically generated

**Test analysis**

|  |  |
| --- | --- |
| **Test Name** | **Purpose** |
| **testRegister** | Test new user register |
| **testLogin1** | Test admin login |
| **testLogin2** | Test normal user login |
| **testSearch** | Test search with dept and dest |
| **testTerminate** | Test valid order get for all orders |

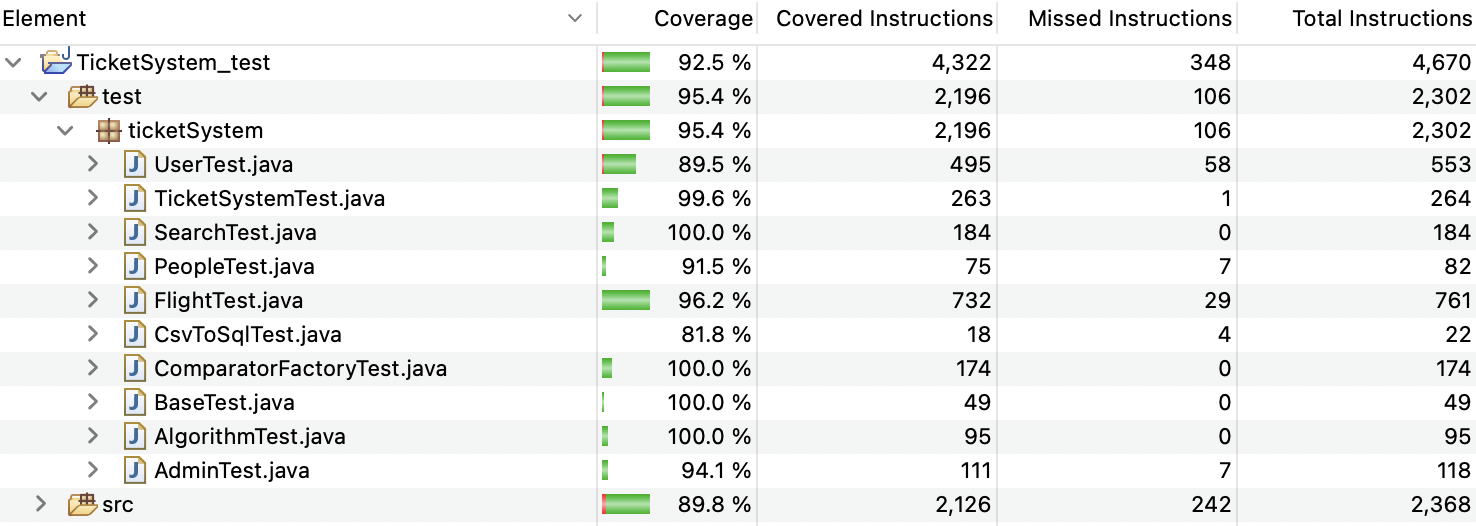
1. **Testing coverage and Analysis**

**1)Coverage of source files**

**Table

Description automatically generated**

**2)Coverage of testing files**



We can see the testing coverage results of our project is 92.5% in total, which combines coverage of src(89.8%) and test(95.4%). Overall, this result lives up to our expectations and can be considered as a successful output. By analyzing the detailed data, we can find out the main not-covered part is Database, including Database.java and CsvToSql.java. The reason why they are not fully covered is we project needs to be connected to outside database and import data from it, therefore the testing for database cannot be shown in the source file, except for this reason, I believe our coverage results will hit a higher percentage.