Assignment T5 - Second Iteration

Team: JVM
Members:

Chengchen	cl4021
Qianhui Yu	qy2226
Gu Jin	gj2325
Yicheng Li	yl4251
Zixuan Zhang	zz2777

Part 1 User Stories

All the user stories are implemented, and we will show these stories in the final demo. No changes.

1. As a user, I want to be able to log in the app so I can access my history from different devices if I am logged in.

   My conditions of satisfaction are:
   a. No other user has the same username as I do.
   b. My account is protected by a password that only I know.
   c. The website remembers that I have logged into the device for some period. During this period, I do not need to log in again if I visit the website again.
   d. If I do not visit the website for a long time (e.g. a day), the website should automatically log me out, so that other users that access the same device will not be able to use my account.
2. As a user, I want to record my daily diet in the app so that I can see my diet history and get some statistics about my past diet records.
   My conditions of satisfaction are:
   a. The app supports at least 5 types of food for me to choose from for each record
   b. I can modify or delete the created diet record in case I made a mistake
   c. The records are available on any device as long as I am logged in to the account.
   d. The app can provide me with accurate statistics on the food I eat
   e. I can see my history in a different time period (1 week, 1 month, 1 year)
3. As a user, I want to record my weight and on a daily basis so that I can track the changes in my weight throughout any period.
   My conditions of satisfaction are:
   a. I can record my weight in either pound or kilograms.

    b. I can view my weight in either pound or kilograms no matter which unit I used when entering it.
    c. I can modify or delete the created weight record in case I made a mistake
    d. The records are available on any device as long as I am logged in to the account.
4. As a user, I want to record my workout history so that I can keep track of the time and intensity of my past workouts.
   My conditions of satisfaction are:
    a. The app can recognize different types of workout from the records I input
    b. The app can provide me with statistics of my workout based on workout type and duration.
    c. I can modify or delete the created workout record in case I made a mistake
    d. I should receive a warning when I am trying to delete a record or modify sensitive attributes of the record like date
5. As a user, I want to see a visualized report of my past records so that I can understand the statistics better
   My conditions of satisfaction are:
    a. I should be able to select different durations to visualize (1 week, 1 month, 3 months, 6 months, 1 year)
    b. The display of data should be easy to understand. e.g. I can see the change of my weights in a curve chart.
    c. The overall display should be visually appealing.
    d. The statistics should always be up-to-date and reflect the most recent input
6. As an intensive user, I want to get some suggestions from the app about my life habit so that I can become more healthy by following them.
   My conditions of satisfaction are:
    a. The suggestions are based specifically on the different records I input to the application, instead of just general health advice
    b. The suggestions should be easy to understand and follow
    c. The suggestions should be beneficial to my health instead of damaging it
    d. I can obtain suggestions based on my different personal needs (reduce fat, increase fitness, build shape)

Part 2 Test Plan

Automated test suite is located in
wehealth/src/test/java/com/jvm/coms4156/columbia/wehealth.

In our system, after excluding all getter, setter, and constructors of our data models and classes, all major functions and subroutines are located in folder **service** and **utility** which handle the real business logics and requests dispatched from controllers and frontend.

The **service** folder contains the following modules

Service Module Name	**Functions**	**Number of Methods will occur branches**
AppUserService	Handling user signup & signin requests	4
DietService	Handing diet record related requests	5
ExerciseService	Handing exercise record related requests	5
WeightService	Handing weight record related requests	4
AdviceService	Handing health advice related requests	1
JwtService	Handling user authentication	0

The **utility** folder contains functions for conversion between DateTime and specific String format and calculating wanted DateTime by users' requests, which are not related to the business logic but useful in every service module.

Module Name	**Functions**	**Number of Methods will occur branches**
Utility	Handling time related calculation and format conversion	1

Therefore, our test plan is focusing on the unit-test for methods and branches in services and utilities.

Tips: The following tables will ONLY show the parameters that occur branches in methods, and discuss boundary conditions about those parameters and corresponding unit-test cases.

- # DietService

➢ **Function 1:**
addDietRecordToDb(AuthenticatedUser au, DietRecordDto dietRecordDto)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
dietTypeId	Exists in the dietType Table or not?	dietTypeId exists	dietTypeId not exists
weight	> 0 or not?	weight > 0	weight <= 0
unit	In [gram, pound] or not	unit = 'gram' || unit = 'pound'	unit != 'gram' && unit != 'pound'

❖ **Valid equivalence class (already covered in the codebase)**

1. **public void addDietRecordValidTest()**
   {
   "userId" = "1",
   "dietTypeId" = "1",
   "weight" = "10.0",
   "unit" = "gram"
   }
2. **public void addDietRecordValid2Test()**
   {
   "userId" = "1",
   "dietTypeId" = "1",
   "weight" = "10.0",
   "unit" = "pound"
   }

❖ **Invalid equivalence class (already covered in the codebase)**

1. **public void addDietRecordInvalidUserIdTest()**
   {
   "userId" = "-1",
   "dietTypeId" = "1",
   "weight" = "10.0",
   "unit" = "gram"
   }
2. **public void addDietRecordInvalidDietTypeIdTest()**
   {
   "userId" = "1",
   "dietTypeId" = "-1",

"weight" = "10.0",
"unit" = "gram"
}

3. public void addDietRecordInvalidWeightTest()
{
"userId" = "1",
"dietTypeId" = "1",
"weight" = "-10.0",
"unit" = "gram"
}
4. public void addDietRecordInvalidUnitTest()
{
"userId" = "-1",
"dietTypeId" = "1",
"weight" = "10.0",
"unit" = "random"
}

➢ **Function 2:**
addAllNutrientsInfoToDietNutrientMapping(DietRecordDto dietRecordDto)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
dietTypeId	Exists in the dietType Table or not?	dietTypeId exists	dietTypeId not exists

❖ **Valid equivalence class (already covered in the codebase)**
public void addDietNutrientMappingValidTest()
{
"dietTypeId" = "1"
}

❖ **Invalid equivalence class (already covered in the codebase)**
1. public void addDietNutrientMappingInvalidTest()
{
"dietTypeId" = "-1"
}

➢ **Function 3:**
getDietHistory(AuthenticatedUser au, Optional<String> unit, Optional<Integer> length)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
timeLength	> 0 or not?	length > 0	length <= 0

timeUnit	In [all, week, month, year] or not	timeUnit IN [all, week, month, year]	timeUnit NOT IN [all, week, month, year]

- **Valid equivalence class (already covered in the codebase)**
  1. **public void getDietHistoryValidTest()**
     {
     "userId" = "1",
     "timeLength" = "1",
     "timeUnit" = "all"
     }
  2. **public void getDietHistoryValid2Test()**
     {
     "userId" = "1",
     "timeLength" = "2",
     "timeUnit" = "month"
     }

- **Invalid equivalence class (already covered in the codebase)**
  1. **public void getDietHistoryInvalidUserIdTest()**
     {
     "userId" = "-1",
     "timeLength" = "1",
     "timeUnit" = "all"
     }
  2. **public void getDietHistoryInvalidTimeLengthTest()**
     {
     "userId" = "1",
     "timeLength" = "-1",
     "timeUnit" = "all"
     }
  3. **public void getDietHistoryInvalidTimeUnitTest()**
     {
     "userId" = "1",
     "timeLength" = "1",
     "timeUnit" = "quarter"
     }

➢ **Function 4:**
updateDietHistory(AuthenticatedUser au, Integer recordId, DietRecordDto dietRecordDto)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
recordId	Exists in the dietHistory Table or not?	Exists && Belongs to	NOT Exists || NOT Belongs to

	Belongs to the user who made the request or not?		
weight	> 0 or not?	weight > 0	weight <= 0
unit	In [gram, pound] or not	unit = ‘gram’ || unit = ‘pound’	unit != ‘gram’ && unit != ‘pound’

- **Valid equivalence class (already covered in the codebase)**
  1. **public void updateDietHistoryValidTest()**
     {
     “userId” = “1”,
     “recordId” = “1”,
     “weight” = “10.0”,
     “unit” = “gram”
     }
  2. **public void updateDietHistoryValidPOUNDTest()**
     {
     “userId” = “1”,
     “recordId” = “1”,
     “weight” = “10.0”
     “unit” = “pound”
     }
- **Invalid equivalence class (already covered in the codebase)**
  1. **public void updateDietHistoryUserNotFoundTest()**
     {
     “userId” = “-1”,
     “recordId” = “1”,
     “weight” = “10.0”,
     “unit” = “pound”
     }
  2. **public void updateDietHistoryInvalidRecordIdTest()**
     {
     “userId” = “2”,
     “recordId” = “1”, (record 1 belongs to userId = 1)
     “weight” = “10.0”,
     “unit” = “gram”
     }
  3. **public void updateDietHistoryInvalidWeightTest()**
     {
     “userId” = “1”,
     “recordId” = “1”,
     “weight” = “-10.0”,
     “unit” = “pound”
     }
  4. **public void updateDietHistoryInvalidWeightUnitTest()**
     {
     “userId” = “1”,

“recordId” = “1”,
“weight” = “10.0”,
“unit” = “kilogram”
}

5. public void updateDietHistoryRecordIdNotFoundTest()
{
“userId” = “1”,
“recordId” = “-1”,
“weight” = “10.0”,
“unit” = “kilogram”
}

➢ **Function 5:**
deleteDietHistory(AuthenticatedUser au, Integer recordId)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
recordId	Exists in the dietHistory Table or not? Belongs to the user who made the request or not?	Exists && Belongs to	NOT Exists || NOT Belongs to

❖ **Valid equivalence class (already covered in the codebase)**

1. public void deleteDietHistoryValidTest()
{
“userId” = “1”,
“recordId” = “1”,
}

❖ **Invalid equivalence class (already covered in the codebase)**

1. public void deleteDietHistoryUserNotFoundTest()
{
“userId” = “-1”,
“recordId” = “1”,
}
2. public void deleteDietHistoryInvalidRecordIdTest()
{
“userId” = “2”,
“recordId” = “1”, (record 1 belongs to userId = 1)
}
3. public void deleteDietHistoryRecordIdNotFoundTest()
{
“userId” = “1”,
“recordId” = “-1”,
}

- # ExerciseService

➢ **Function 1:**

validateUser(Long userId, Optional<Long> requestUserId)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
requestUserId	Is Present or not? Equals to userId or not?	NOT Present || (Present && Equals to userId)	Present && NOT Equals to userId

❖ **Valid equivalence class (already covered in the codebase)**

```
public void validateUserTest()
{
        "userId" = "1",
        "requestUserId" = "1",
}
```

❖ **Invalid equivalence class (already covered in the codebase)**

```
public void validateUserInvalidUserTest()
{
        "userId" = "-1",
        "requestUserId" = "1",
}
public void validateUserDifferentUserTest()
{
        "userId" = "-1",
        "requestUserId" = "2",
}
```

➢ **Function 2:**

addExerciseRecordToDb(ExerciseRecordDto exerciseRecordDto, AuthenticatedUser au)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
duration	> 0 or not?	Duration > 0	Duration <= 0
exerciseType Name	Exists in the exerciseType table or not?	Exists	NOT Exists

❖ **Valid equivalence class (already covered in the codebase)**

```
public void addExerciseRecordToDBValidTest()
{
        "duration" = "1000.0",
        "exerciseTypeName" = "TestExerciseType"
```

}

- Invalid equivalence class (already covered in the codebase)

```
public void addExerciseRecordToDBInvalidTypeTest()
{
        "duration" = "1000.0",
        "exerciseTypeName" = "InvalidTestExerciseType"
}
public void addExerciseRecordToDBNonPositiveDurationTest()
{
        "duration" = "-1000.0",
        "exerciseTypeName" = "InvalidTestExerciseType"
}
```

➢ **Function 3:**

`getExerciseHistory(Optional<String> unit, Optional<Integer> length, AuthenticatedUser au)`

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
length	> 0 or not?	length > 0	length <= 0

- Valid equivalence class (already covered in the codebase)

1.
```
public void getExerciseHistoryAllTest()
{
        "length" = "1"
}
```

- Invalid equivalence class (already covered in the codebase)

1.
```
public void getExerciseHistoryNegativeDurationTest()
{
        "length" = "-100"
}
```

➢ **Function 4:**

`editExerciseRecordAtDb(Optional<Integer> recordId, ExerciseRecordDto exerciseRecordDto, AuthenticatedUser au)`

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
recordId	Is Present or not? Exists in the exerciseHistory Table or not?	Present && Exists	NOT Present || NOT Exists
duration	> 0 or not?	Duration > 0	Duration <= 0
exerciseType	Exists in the exerciseType	Exists	NOT Exists

Name	table or not?		

- **Valid equivalence class (already covered in the codebase)**
  1. **public void editExerciseRecordValidTest()**
     {
     "recordId" = "1",
     "duration" = "10.0",
     "exerciseTypeName" = "TestExerciseType"
     }

- **Invalid equivalence class (already covered in the codebase)**
  1. **public void editExerciseRecordInvalidExerciseIdTest()**
     {
     "recordId" = "-1",
     "duration" = "10.0",
     "exerciseTypeName" = "TestExerciseType"
     }
  2. **public void editExerciseRecordInvalidTypeNameTest()**
     {
     "recordId" = "1",
     "duration" = "10.0",
     "exerciseTypeName" = "InvalidTestExerciseType"
     }
  3. **public void editExerciseRecordNonPositiveDurationTest()**
     {
     "recordId" = "-1",
     "duration" = "10.0",
     "exerciseTypeName" = "TestExerciseType"
     }

➢ **Function 5:**
deleteExerciseHistory(AuthenticatedUser au, Integer recordId)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
recordId	Exists in the dietHistory Table or not? Belongs to the user who made the request or not?	Exists && Belongs to	NOT Exists || NOT Belongs to

- **Valid equivalence class (already covered in the codebase)**
  2. **public void deleteExerciseHistoryValidTest()**
     {
     "userId" = "1",
     "recordId" = "1",

}

- Invalid equivalence class (already covered in the codebase)

4. public void deleteExerciseHistoryUserNotFoundTest()
   {
   “userId” = “-1”,
   “recordId” = “1”,
   }
5. public void deleteExerciseHistoryInvalidRecordIdTest()
   {
   “userId” = “2”,
   “recordId” = “1”, (record 1 belongs to userId = 1)
   }
6. public void deleteExerciseHistoryRecordIdNotFoundTest()
   {
   “userId” = “1”,
   “recordId” = “-1”,
   }

• WeightService

➢ **Function 1:**
addWeightRecordToDb(AuthenticatedUser au, WeightRecordDto WeightRecordDto)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
weight	> 0 or not?	weight > 0	weight <= 0
unit	In [gram, pound] or not	unit = ‘gram’ || unit = ‘pound’	unit != ‘gram’ && unit != ‘pound’

- Valid equivalence class (already covered in the codebase)

1. public void addWeightRecordToDBGramTest()
   {
   “userId” = “1”,
   “weight” = “60000.0”,
   “unit” = “gram”
   }
2. public void addWeightRecordToDBPoundTest()
   {
   “userId” = “1”,
   “weight” = “60000.0”,
   “unit” = “pound”
   }

- Invalid equivalence class (already covered in the codebase)

1. public void addWeightRecordToDBInvalidUserIdTest()

{
“userId” = “-1”,
“weight” = “60000.0”,
“unit” = “gram”
}

2. public void addWeightRecordToDBInvalidWeightTest()
{
“userId” = “1”,
“weight” = “-60000.0”,
“unit” = “gram”
}
3. public void addWeightRecordToDBInvalidUnitTest()
{
“userId” = “-1”,
“weight” = “10.0”,
“unit” = “random”
}

➢ **Function 2:**
getWeightHistory(AuthenticatedUser au, Optional<String> unit, Optional<Integer> length)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
timeLength	> 0 or not?	length > 0	length <= 0
timeUnit	In [all, week, month, year] or not	timeUnit IN [all, week, month, year]	timeUnit NOT IN [all, week, month, year]

❖ Valid equivalence class (already covered in the codebase)

1. public void getWeightHistoryValidTest()
{
“userId” = “1”,
“timeLength” = “1”,
“timeUnit” = “all”
}
2. public void getWeightHistoryValid2Test()
{
“userId” = “1”,
“timeLength” = “2”,
“timeUnit” = “month”
}

❖ Invalid equivalence class (already covered in the codebase)

1. public void getWeightHistoryInvalidUserIdTest()
{
“userId” = “-1”,

“timeLength” = “1”,
“timeUnit” = “all”
}

2. public void getWeightHistoryInvalidTimeLengthTest()
{
“userId” = “1”,
“timeLength” = “-1”,
“timeUnit” = “all”
}
3. public void getWeightHistoryInvalidTimeUnitTest()
{
“userId” = “1”,
“timeLength” = “1”,
“timeUnit” = “quarter”
}

➢ **Function 3:**
updateWeightHistory(AuthenticatedUser au, Integer recordId, WeightRecordDto weightRecordDto)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
recordId	Exists in the dietHistory Table or not? Belongs to the user who made the request or not?	Exists && Belongs to	NOT Exists || NOT Belongs to
weight	> 0 or not?	weight > 0	weight <= 0
unit	In [gram, pound] or not	unit = ‘gram’ || unit = ‘pound’	unit != ‘gram’ && unit != ‘pound’

❖ **Valid equivalence class (already covered in the codebase)**

1. public void updateWeightHistoryValidTest()
{
“userId” = “1”,
“recordId” = “1”,
“weight” = “60000.0”,
“unit” = “gram”
}
2. public void updateWeightHistoryValidPOUNDTest()
{
“userId” = “1”,
“recordId” = “1”,
“weight” = “60000.0”
“unit” = “pound”

}

- Invalid equivalence class (already covered in the codebase)

1. public void updateWeightHistoryUserNotFoundTest()
   {
   "userId" = "-1",
   "recordId" = "1",
   "weight" = "10.0",
   "unit" = "pound"
   }
2. public void updateWeightHistoryInvalidRecordIdTest()
   {
   "userId" = "2",
   "recordId" = "1", (record 1 belongs to userId = 1)
   "weight" = "10.0",
   "unit" = "gram"
   }
3. public void updateWeightHistoryInvalidWeightTest()
   {
   "userId" = "1",
   "recordId" = "1",
   "weight" = "-60000.0",
   "unit" = "pound"
   }
4. public void updateWeightHistoryInvalidWeightUnitTest()
   {
   "userId" = "1",
   "recordId" = "1",
   "weight" = "10.0",
   "unit" = "kilogram"
   }
5. public void updateWeightHistoryRecordIdNotFoundTest()
   {
   "userId" = "1",
   "recordId" = "-1",
   "weight" = "10.0",
   "unit" = "kilogram"
   }

➢ **Function 4:**
deleteWeightHistory(AuthenticatedUser au, Integer recordId)

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
recordId	Exists in the dietHistory Table or not?	Exists && Belongs to	NOT Exists || NOT Belongs to

	Belongs to the user who made the request or not?		

- **Valid equivalence class (already covered in the codebase)**
  1. `public void deleteWeightHistoryValidTest()`
     {
     “userId” = “1”,
     “recordId” = “1”,
     }
- **Invalid equivalence class (already covered in the codebase)**
  1. `public void deleteWeightHistoryUserNotFoundTest()`
     {
     “userId” = “-1”,
     “recordId” = “1”,
     }
  2. `public void deleteWeightHistoryInvalidRecordIdTest()`
     {
     “userId” = “2”,
     “recordId” = “1”, (record 1 belongs to userId = 1)
     }
  3. `public void deleteWeightHistoryRecordIdNotFoundTest()`
     {
     “userId” = “1”,
     “recordId” = “-1”,
     }

• AdviceService

➢ **Function 1:**

```
AdviceDto getAdvice(AuthenticatedUser user,
                    Optional<Integer> length,
                    Optional<String> unit)
```

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
userId	Exists in the user Table or not?	userId exists	userId not exists
length	Valid number or not	Length > 0	Length <= 0
unit	Valid unit or not	Must be week, month or all	else
dietByDayDtos	User has diet history or no?	Length > 0	Length = 0

The two parameters *length* and *unit* are not used directly by *AdviceService.* Instead, they are passed to other services like *DietService* or *WeightService* to get *DietHistory* and *WeightHistory.* Therefore, We did not test the two parameters but using mock to simulate the history response from other services for various *length* and *unit* values

❖ **Valid equivalence class (already covered in the codebase)**

1. **1public void getAdviceValidLongRecordTest()**
   {
   "userId" = "1L",
   "dietByDayDtos.size()" = 100,
   "exerciseByDayDtos.size()" = 100
   }
2. **public void getAdviceValidShortRecordTest()**
   {
   "userId" = "1L",
   "dietByDayDtos.size()" = 1,
   "exerciseByDayDtos.size()" = 1
   }

❖ **Invalid equivalence class (already covered in the codebase)**

3. **public void getAdviceInvalidDietTest()**
   {
   "userId" = "1L",
   "dietByDayDtos.size()" = 0,
   "exerciseByDayDtos.size()" = 1
   }
4. **public void getAdviceInvalidExerciseTest()**
   {
   "userId" = "1L",
   "dietByDayDtos.size()" = 1,
   "exerciseByDayDtos.size()" = 0
   }
5. **public void getAdviceInvalidBothest()**
   {
   "userId" = "1L",
   "dietByDayDtos.size()" = 0,
   "exerciseByDayDtos.size()" = 0
   }
6. **public void getAdviceInvalidUserTest()**
   {
   "userId" = "-1L",
   }

● AppUserService

➢ **Function 1:**
AppUserService register( )

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
username	Exist in DB or not	Username not exist	Username exist
currentPassword	Password length	Length > 0	Length <= 0
newPassword	Password length	Length > 0	Length <= 0

- **Valid equivalence class (already covered in the codebase)**
  1. **public void registerTest()**
     {
     "username" = "fakeName",
     "currentPassword" = "123456",
     "newPassword" = "123456"
     }
- **Invalid equivalence class (already covered in the codebase or missing field)**
  2. **public void registerExistingUser()**
     {
     "username" = "TestUsername",
     "currentPassword" = "1",
     "newPassword" = "1"
     }
  3. **public void registerMissingUsername()**
     {
     "currentPassword" = "1",
     "newPassword" = "1"
     }
  4. **public void registerMissingPassword()**
     {
     "username" = "TestUsername"
     }

➢ **Function 2:**

➢

Parameter	Boundary Conditions	Valid Partitions	Invalid Partitions
username	Username	Username exist	Username not exist
password	Match hashed password in DB after encode	jwt encode match	Jwt encode mismatch

AppUserService login( )

- **Valid equivalence class**
  - **public void loginTest()**
    {

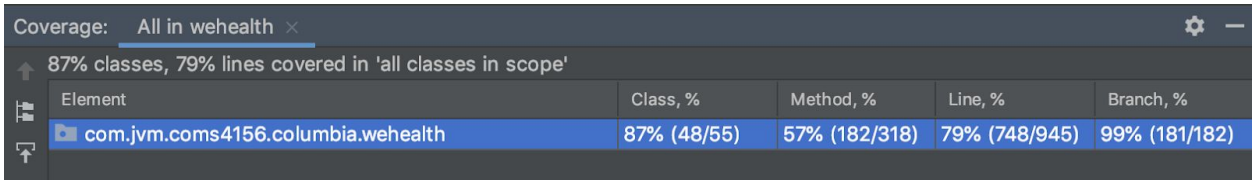
“username” = “Test1”,
“password” = “123456”,
}

- InValid equivalence class (invalid authentication)
  - public void loginTestWithWrongPassword()
  {
  “username” = “Test1”,
  “password” = “1”,
  }

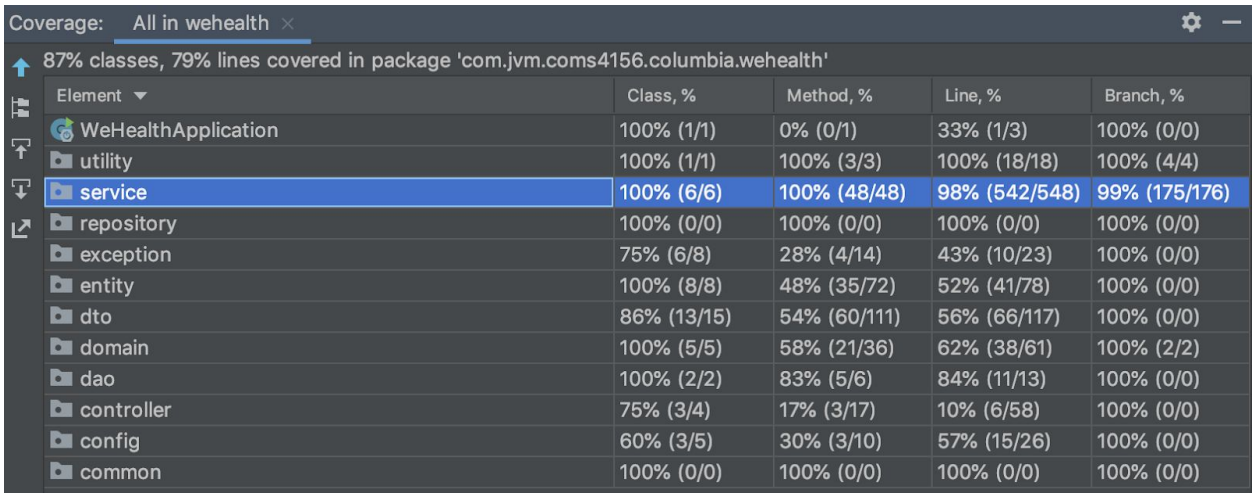
Part 3 Branch Coverage

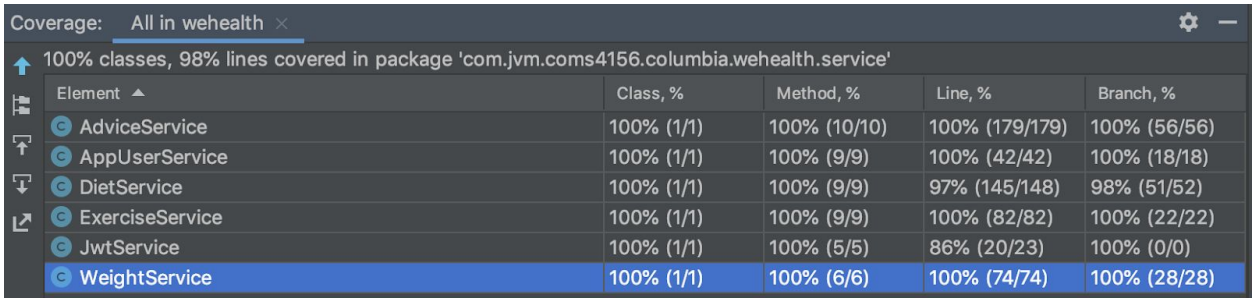
Jacoco coverage report is located in
wehealth/reports/SecondIteration/code coverage/index.html

We used **Jacoco** to test our branch coverage.
Finally, we achieved **99% branch coverage**. We covered 181 branches among 182 all branches.



As we mentioned in part 2, in our system, after excluding all getter, setter, and constructors of our data models and classes, all major functions and subroutines are located in folder **service** and **utility** which handle the real business logics and requests dispatched from controllers and frontend.
We covered 100% on utility and 99% on service.





Why not 100%

One missing branch is located at service/DietService/UpdateDietHistory.
The reason is that we are using Mockito to mock the result of Database interaction.
If we want to test the branch: If newDietType is empty, we need let Mockito return empty when we query the dietType table by dietTypeId. And at the same time, when the newDietType is empty, our service will firstly add the new diet type into the database and then add this new diet type's all nutrient type information to the database.
However, in the method addAllNutrientsInfoToDietNutrientMapping, we need to double check if the input dietType exists. At this time, the dietType is supposed to be present in the database, but Mockito will return empty as we set in unit-test, thus the system will throw BadRequestException but not continue.
Thus, we can't cover this branch by Mockito. **But we tried our best to cover all the other branches and achieved 99%.**

```
// Check if need to update diet type
DietType dietType = dietHistory.get().getDietType();
if (dietType.getDietTypeId() != dietRecordDto.getDietTypeId()) {
  // Update diet type for this record
  // Check if the new diet type exists
  Optional<DietType> newDietType = dietTypeRepo.findByDietTypeId(dietRecordDto.getDietTypeId());
  if (newDietType.isEmpty()) {
    // add new diet type to diet_type table
    addDietType(dietRecordDto.getDietTypeId(), dietRecordDto.getDietTypeName());
    // add 4 nutrients' info to diet_nutrient_mapping table
    addAllNutrientsInfoToDietNutrientMapping(dietRecordDto);
  }
  newDietType = dietTypeRepo.findByDietTypeId(dietRecordDto.getDietTypeId());
  dietHistory.get().setDietType(newDietType.get());
}
```

Part 4 Continuous Integration

We use **Github Action** as our CI tool.
Configuration file is located at **.github/workflows/maven.yml**

CI reports are located at
wehealth/reports/SecondIteration

Screenshots of working CI actions of our project.