HealthHub

TEST SPECIFICATION REPORT

Course Name	Software Engineering
Course Code	BLG411E
Course CRN	13330
Team Number	21
Team Members	Hakan SANDER - 150140146 Rıdvan SIRMA - 150120133 Görkem TOPPEKER – 150140139 Adil Furkan EKİCİ – 150140112
Date	17.12.2017

Table of Contents

1. Introduction

This document provides the information for the test specification contents of the HealthHub which is an Android project. The process of running a program to find the errors is called as testing. Test specification is very important for the success of a software project since many projects fail due to insufficient test planning and face the severe consequences and those consequences might lead to even deadly results.

1.1. Goal

By the test specification, the bugs of a software project are determined and solved. Test specification document provides to fix the bugs, to observe the errors in the project and prevents the project failure and negative results. In many important software projects, test specification takes a long time to deliver a high quality software product in the end of the progress. Due to all these benefits of the test specification, necessary tests are applied on the HealthHub project and test specification document is prepared.

1.2. Contents and Organization

This document is compromised of three main parts which are test plan, unit tests and additional tests. The first part which is the test plan includes testing strategy, test subjects and equivalence partitioning. In the testing strategy, the chosen strategy and the motivation for choosing it are explained. In the test subjects, the component diagram in the design specification document is considered and the data that is the subject to integration testing is explained. The second part which is the unit tests includes the test cases and the scripts. In the test cases, the test cases are listed and mapped to the user stories. In the scripts, the script for unit testing table is given for each scenario. The last part which is the additional tests includes the security testing, performance testing, device testing and acceptance testing. For all these testing parts, the goal of the tests and the techniques that are used are explained in detail.

2. Test Plan

2.1. Testing Strategy

In the HealthHub project, the Big Bang Approach is chosen as the testing strategy instead of the Incremental Approach.

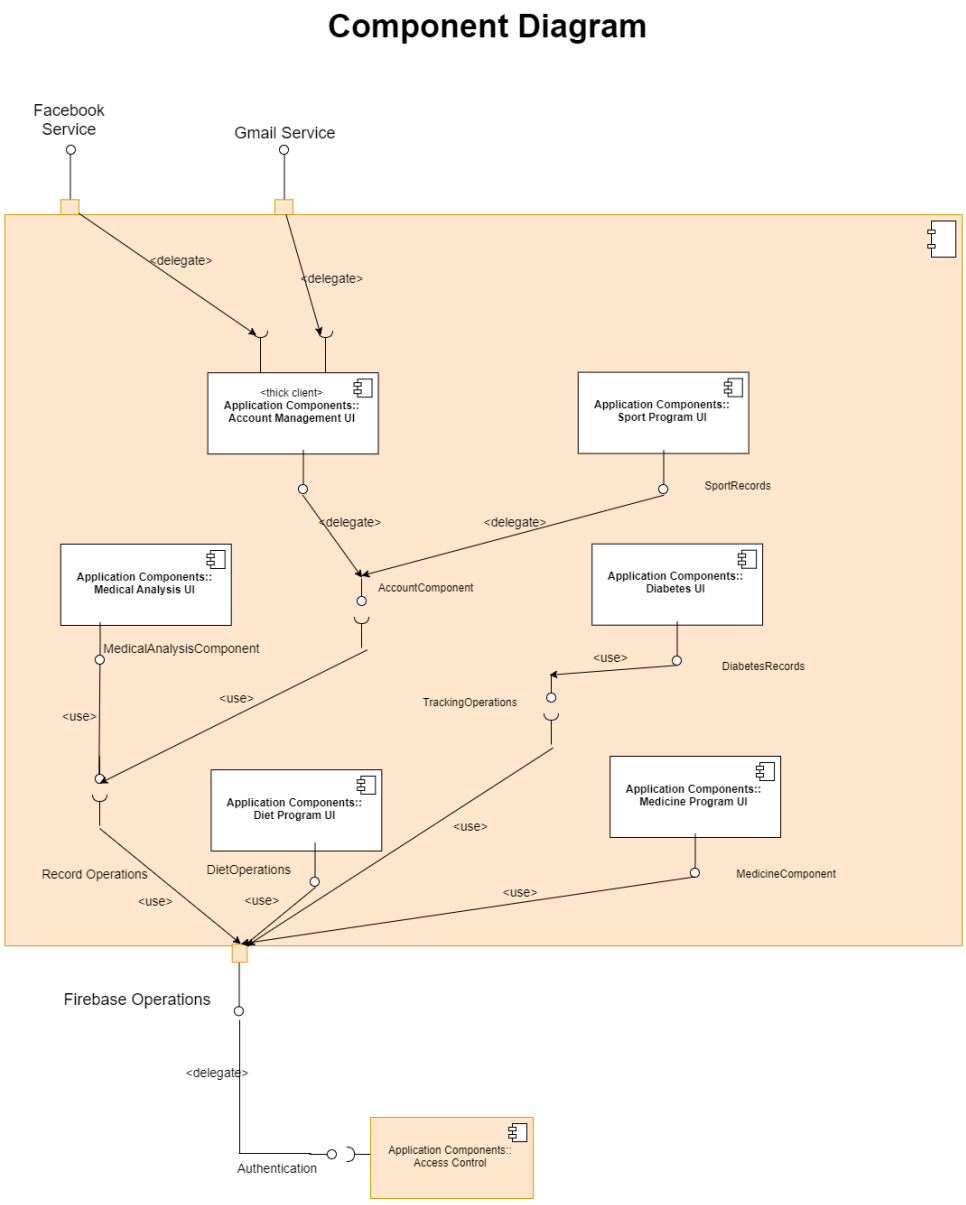
Motivations of Using the Big Bang Approach Strategy:

- The Big Bang Approach is suitable for the start-up projects and small teams.
- The approach is suitable for the teams that consist of members which have multiple areas to work(both back-end and front-end). In the HealthHub project, all the team members are worked as full-stack developers.
- The Big Bang Approach is appropriate for the teams which has no management hierarchy. In the HealthHub project, we had no management hierarchy.
- The model is very simple and easy to apply to a project.
- Nearly no planning is required for the Big Bang Approach.
- The test report assignment is given when the project is nearly completed. Therefore, we had to apply integration tests after the all modules are fully implemented and combined as a whole.

The Big Bang Approach	
Advantages	Disadvantages
Suitable for start-up projects and small teams.	Not suitable for the complex projects.
Suitable for the teams whose members have multi-duties on both designing and programming.	Contains very high risks that may lead to project failure.
Simple and easy to be applied.	If the requirements are not understood correctly, may lead to costly consequences.
Test management is easy.	Not suitable for the projects that lasts for a long time.
Suitable for the teams which does not have a hierarchical management system.	Since the integration tests are applied as a whole after all the components are fully implemented and combined as a whole, critical errors may appear very lately and finding the source of the errors might be very difficult.
Provides flexibility to make changes to developers.	Detaching the components might be needed to detect the errors, very time consuming.
Planning is not required.	

2.2. Test Subjects

- The component diagram of the project is given below, which was the topic of Design Specification document (part 3.2).
- As seen from this diagram, all the components are subject to integration testing.
- Since the aim of integration testing is to see whether do we get the expected result or not when the parts of a component are put together, we have implemented integration testing to all these components.



The data that are subject to integration testing for each component are explained in detail below.

1) Account Management Component Integration Tests:

- Test the login functionality via Google and Facebook Accounts.
- Test whether user type functionality works correctly.
- Test whether user relations between "Supervisor" and "HealthMan" accounts work correctly.

2) Medical Analysis Component Integration Tests:

- Test the information that is supplied by user (e.g. test photo, explanation) are saved successfully.
- Test the camera can take the photo of the analysis report and that photo is correctly saved to firebase.
- Test whether user can reach previously saved test records successfully.

3) Diet Program Component Integration Tests:

- Test the information supplied by user (e.g. meal name, date, time) are saved successfully.
- Test the nutritional values of the meals are calculated and saved correctly.
- Test whether user can reach previously saved meal records successfully.

4) Sport Program Component Integration Tests:

- Test the information supplied by user (e.g. training time, cardio information) are saved successfully.
- Test the step counter functionality whether the previously measured step counts can be reached and program can continue counting from that value.
- Test whether user can reach previously saved sport program records successfully.

5) Diabetes Component Integration Tests:

- Test the information supplied by user (e.g. meal information, carbohydrate count) are saved successfully.
- Test the insulin dose values that is supplied by user is correctly added with the time.
- Test whether user can reach previously saved insulin, carbohydrate count records successfully.

6) Medicine Program Component Integration Tests:

- Test the information supplied by user (e.g. medicine name, time, date) are saved successfully.
- Test the notification dates and times are correctly generated according to user inputs via form and notifications are alerted in expected times.
- Test whether user can reach previously saved medicine and prospectus records successfully.

2.3. Equivalence Partitioning

Insulin Dose:

Valid Equivalence Class Partition: An integer value between 1-99.
Non-valid Equivalence Class Partition: Values 'over 99' or 'below 1' are restricted.

Blood sugar value:

Valid Equivalence Class Partition: Integer values between 30 and 600.
Non-valid Equivalence Class Partition: Values 'below 30' or 'above 600' are restricted.

Portion Size:

Valid Equivalence Class Partition: Integer values between 1-40.
Non-valid Equivalence Class Partition: Values 'below 1' and 'above 40' are restricted.

Medicine date:

Valid Equivalence Class Partition: Start date is earlier than end date in yyyy-mm-dd form. (yyyy is 4 digit number, mm is between 1-12, dd limit is determined by the selected month in calendar)
Non-valid Equivalence Class Partition: Start date is later than end date or form of date input does not hold above restrictions.

Medicine time:

Valid Equivalence Class Partition: Times are in hh:mm format. (hh is between 0-23, mm is between 0-59)
Non-valid Equivalence Class Partition: Time input does not hold above restrictions.

Medicine name:

Valid Equivalence Class Partition: Name must contain alpha-numeric characters, empty strings are restricted.
Non-valid Equivalence Class Partition: Empty strings.

3. Unit Tests

3.1. Test Cases

- The below table lists the test cases and map them to user stories.

Test Case	User Story
Add medicine information via form.	Medicine Information
Show medicine record pastly	Medicine History
Add prospectus of a medicine via photo	Prospectus Information
Show prospectus of a medicine	Prospectus History
Add new sport record	Sport Info
Show sport records	Sport Records
Add new sport program	Future Sport Info
Show sport programs	Future Sport Records
Add new medical analysis	Medical Analysis Info
Show medical analysis	Medical Analysis Record
Count and show step	Step Counter
Sign up with Facebook And Google	Account Creation
Login to account with Facebook and Google	Facebook and Google Login

3.2. Scripts

Scenario:	Adding medicine Information			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click add medicine page button in main medicine page	Move to adding medicine page	Move to adding medicine page	-
2	Fill the required places	Get the necessary info	Get the necessary info	-
3	Click submit button	Added new medicine record	Added new medicine record	-

Script of Test code for adding medicine Information:

```
Instrumentation.ActivityMonitor monitorForMedicineActivity = getInstrumentation().addMonitor(MedicineActivity.class.getName(),null,false);
    @SmallTest
    public void testAddButton(){
        assertNotNull(mActivity.findViewById(R.id.addButton));
    }
    @SmallTest
    public void testmedicineActivity(Activity medicineActivity){
        assertNotNull(medicineActivity);
    }
    @Test
    public void testLaunchOfMedicineActivityOnButtonClick(){
        testAddButton();
        onView(withId(R.id.addButton)).perform(click());
        Activity medicineActivity = getInstrumentation().waitForMonitorWithTimeout(monitorForMedicineActivity,5000);
        testmedicineActivity(medicineActivity);
        medicineActivity.finish();
}
```

Scenario:	Show medicine record pastly			
Test Number	Test Description / Input	Expected Result	Actual Result	Fix Action
1	Press list medicine record page button in main medicine page	Move to list medicine record page	Move to list medicine record page	-
2	See medicine record in page	List medicine record correctly	List medicine record correctly	-

Script of Test code for show medicine record pastly:

```
Instrumentation.ActivityMonitor monitorForMedicineListActivity = getInstrumentation().addMonitor(MedicineListActivity.class.getName(),null,false);
    @SmallTest
    public void testListButton(){
        assertNotNull(mActivity.findViewById(R.id.listButton));
    }
    @SmallTest
    public void testmedicineListActivity(Activity medicineListActivity){
        assertNotNull(medicineListActivity);
    }
    @Test
    public void testLaunchOfMedicineListActivityOnButtonClick(){
        testListButton();
        onView(withId(R.id.listButton)).perform(click());
        Activity medicineListActivity = getInstrumentation().waitForMonitorWithTimeout(monitorForMedicineListActivity,5000);
        testmedicineListActivity(medicineListActivity);
        medicineListActivity.finish();
}
```

Scenario:	Add prospectus of a medicine			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click add prospectus page button in main medicine page	Move to adding prospectus page	Move to adding prospectus page	-
2	Take prospectus photo	Get a smooth and correct photo of prospectus	Get a smooth and correct photo of prospectus	-
3	Click submit button	Added new prospectus photo to database	Added new prospectus photo to database	-

Script of Test code for add prospectus of a medicine:

```
Instrumentation.ActivityMonitor monitorForAddProspectusActivity = getInstrumentation().addMonitor(AddProspectusActivity.class.getName(),null,false);
    @SmallTest
    public void testProspectusAddButton(){
        assertNotNull(mActivity.findViewById(R.id.prospectusAddButton));
    }
    @SmallTest
    public void testaddProspectusActivity(Activity addProspectusActivity){
        assertNotNull(addProspectusActivity);
    }
    @Test
    public void testLaunchOfAddProspectusActivityOnButtonClick(){
        testProspectusAddButton();
        onView(withId(R.id.prospectusAddButton)).perform(click());
        Activity addProspectusActivity = getInstrumentation().waitForMonitorWithTimeout(monitorForAddProspectusActivity,5000);
        testaddProspectusActivity(addProspectusActivity);
        addProspectusActivity.finish();
}
```

Scenario:	Show prospectus of a medicine			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click show prospectus page button in main medicine page	Move to show prospectus page	Move to show prospectus page	-
2	List prospectus photo	Show prospectus images correctly from database	Show prospectus images correctly from database	-

Script of Test code for show prospectus of a medicine:

```
Instrumentation.ActivityMonitor monitorForListProspecturActivity = getInstrumentation().addMonitor(ListProspectusActivity.class.getName(),null,false);
    @SmallTest
    public void testProspectusInfoButton(){
        assertNotNull(mActivity.findViewById(R.id.prospectusInfoButton));
    }
    @SmallTest
    public void testlistProspectusActivity(Activity listProspectusActivity){
        assertNotNull(listProspectusActivity);
    }
    @Test
    public void testLaunchOfListProspectusActivityOnButtonClick(){
        testProspectusAddButton();
        onView(withId(R.id.prospectusInfoButton)).perform(click());
        Activity listProspectusActivity = getInstrumentation().waitForMonitorWithTimeout(monitorForListProspecturActivity,5000);
        testlistProspectusActivity(listProspectusActivity);
        listProspectusActivity.finish();
}
```

Scenario:	Add new sport record to database			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click add new sport record page button in main sport page	Move to adding new sport record page	Move to adding new sport record page	-
2	Fill the required places	Get the necessary info for sport record	Get the necessary info for sport record	-
3	Click submit button	Added new sport record to database	Added new sport record to database	-

Script of Test code for add new sport record to database:

```
Instrumentation.ActivityMonitor monitorAddNewSportRecordActivity = getInstrumentation().addMonitor(addNewSportRecordActivity.class.getName(),null,false);
    @SmallTest
    public void testAddSportRecordButton(){
        assertNotNull(mActivity.findViewById(R.id.addSportRecord));
    }
    @SmallTest
    public void testAddNewSportRecordActivity(Activity addNewSportRecordActivity){
        assertNotNull(addNewSportRecordActivity);
    }
    @Test
    public void testLaunchOfAddNewSportRecordActivityOnButtonClick(){
        testAddSportRecordButton();
        onView(withId(R.id.addSportRecord)).perform(click());
        Activity addNewSportRecordActivity = getInstrumentation().waitForMonitorWithTimeout(monitorAddNewSportRecordActivity,5000);
        testAddNewSportRecordActivity(addNewSportRecordActivity);
        addNewSportRecordActivity.finish();
}
```

Scenario:	Show sport records			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click show sport record page button in main sport page	Move to show sport records page	Move to show sport records page	-
2	List sport records	List medicine record correctly from database	List medicine record correctly from database	-

Script of Test code for show sport records:

```
Instrumentation.ActivityMonitor monitorListSportRecordActivity = getInstrumentation().addMonitor(ListSportRecordActivity.class.getName(),null,false);

    @SmallTest
    public void testSportRecordButton(){
        assertNotNull(mActivity.findViewById(R.id.sportRecord_Button));
    }
    @SmallTest
    public void testListSportActivity(Activity listSportRecordActivity){
        assertNotNull(listSportRecordActivity);
    }
    @Test
    public void testLaunchOfListSportRecordActivityOnButtonClick(){
        testSportRecordButton();
        onView(withId(R.id.sportRecord_Button)).perform(click());
        Activity listSportRecordActivity = getInstrumentation().waitForMonitorWithTimeout(monitorListSportRecordActivity,5000);
        testListSportActivity(listSportRecordActivity);
        listSportRecordActivity.finish();
}
```

Scenario:	Add new sport program to database			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click add new sport program page button in main sport page	Move to adding new sport program page	Move to adding new sport program page	-
2	Fill the required places	Get the necessary info for sport program	Get the necessary info for sport program	-
3	Click submit button	Added new sport program to database	Added new sport program to database	-

Script of Test code for add new sport program to database:

```
Instrumentation.ActivityMonitor monitorAddNewSportProgramActivity = getInstrumentation().addMonitor(addNewSportProgram.class.getName(),null,false);
    @SmallTest
    public void testCreateSportProgramButton(){
        assertNotNull(mActivity.findViewById(R.id.createSportProgram));
    }
    @SmallTest
    public void testAddNewSportProgramActivity(Activity addNewSportProgramActivity){
        assertNotNull(addNewSportProgramActivity);
    }
    @Test
    public void testLaunchOfAddNewSportProgramActivityOnButtonClick(){
        testCreateSportProgramButton();
        onView(withId(R.id.createSportProgram)).perform(click());
        Activity addNewSportProgramActivity = getInstrumentation().waitForMonitorWithTimeout(monitorAddNewSportProgramActivity,5000);
        testAddNewSportProgramActivity(addNewSportProgramActivity);
        addNewSportProgramActivity.finish();
}
```

Scenario:	Show sport program			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click show sport program page button in main sport page	Move to show sport program page	Move to show sport program page	-
2	List sport program	List medicine program correctly from database	List medicine program correctly from database	-

Script of Test code for show sport program from database:

```
Instrumentation.ActivityMonitor monitorListSportProgramActivity = getInstrumentation().addMonitor(ListSportProgramActivity.class.getName(),null,false);
    @SmallTest
    public void testShowSportProgramButton(){
        assertNotNull(mActivity.findViewById(R.id.showSportProgram));
    }
    @SmallTest
    public void testListSportProgramActivity(Activity listSportProgramActivity){
        assertNotNull(listSportProgramActivity);
    }
    @Test
    public void testLaunchOfListSportProgramActivityOnButtonClick(){
        testShowSportProgramButton();
        onView(withId(R.id.showSportProgram)).perform(click());
        Activity listSportProgramActivity = getInstrumentation().waitForMonitorWithTimeout(monitorListSportProgramActivity,5000);
        testListSportProgramActivity(listSportProgramActivity);
        listSportProgramActivity.finish();
}
```

Scenario:	Add new medical analysis			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click add medical analysis page button in main medical analysis page	Move to adding medical analysis page	Move to adding medical analysis page	-
2	Take medical analysis photo	Get a smooth and correct photo of medical analysis	Get a smooth and correct photo of medical analysis	-
3	Click submit button	Added new medical analysis record	Added new medical analysis record	-

Script of Test code for add new medical analysis:

```
Instrumentation.ActivityMonitor monitoraddAnalysisActivity = getInstrumentation().addMonitor(AddAnalysisActivity.class.getName(),null,false);
    @SmallTest
    public void testaddAnalysis_2(){
        assertNotNull(mActivity.findViewById(R.id.addAnalysis_2));
    }
    @SmallTest
    public void testmedicalAnalysisActivity(Activity medicalAnalysisActivity){
        assertNotNull(medicalAnalysisActivity);
    }
    @Test
    public void testLaunchOfStepCounterActivityOnButtonClickWithId2(){
        testaddAnalysis_2();
        onView(withId(R.id.addAnalysis_2)).perform(click());
        Activity medicalAnalysisActivity = getInstrumentation().waitForMonitorWithTimeout(monitoraddAnalysisActivity,5000);
        testmedicalAnalysisActivity(medicalAnalysisActivity);
        medicalAnalysisActivity.finish();
    }
    @SmallTest
    public void testaddAnalysis_1(){
        assertNotNull(mActivity.findViewById(R.id.addAnalysis_1));
    }
    @Test
    public void testLaunchOfAddAnalysisActivityOnButtonClickwithId1(){
        testaddAnalysis_1();
        onView(withId(R.id.addAnalysis_1)).perform(click());
        Activity medicalAnalysisActivity = getInstrumentation().waitForMonitorWithTimeout(monitoraddAnalysisActivity,5000);
        assertNotNull(medicalAnalysisActivity);
        medicalAnalysisActivity.finish();
}
```

Scenario:	Show medical analysis			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click show medical analysis button in main medical analysis page	Move to show medical analysis page	Move to show medical analysis page	-
2	List medical analysis	List medical analysis correctly from database	List medical analysis correctly from database	-

Script of Test code for show medical analysis:

```
Instrumentation.ActivityMonitor monitorMedicalAnalysisReportActivity = getInstrumentation().addMonitor(MedicalAnalysisReportsActivity.class.getName(),null,false);

    @SmallTest
    public void testshowAnalysis_2(){
        assertNotNull(mActivity.findViewById(R.id.showAnalysis_2));
    }
    @SmallTest
    public void testmedicalAnalysisReportActivity(Activity medicalAnalysisReportActivity){
        assertNotNull(medicalAnalysisReportActivity);
    }
    @Test
    public void testLaunchOfActivityOnButtonClickWithId2(){
        testshowAnalysis_2();
        onView(withId(R.id.showAnalysis_2)).perform(click());
        Activity medicalAnalysisReportActivity = getInstrumentation().waitForMonitorWithTimeout(monitorMedicalAnalysisReportActivity,5000);
        testmedicalAnalysisReportActivity(medicalAnalysisReportActivity);
        medicalAnalysisReportActivity.finish();
    }
    @SmallTest
    public void testshowAnalysis_1(){
        assertNotNull(mActivity.findViewById(R.id.showAnalysis_1));
    }
    @Test
    public void testLaunchOfActivityOnButtonClickWithId1(){
        assertNotNull(mActivity.findViewById(R.id.showAnalysis_1));
        onView(withId(R.id.showAnalysis_1)).perform(click());
        Activity medicalAnalysisReportActivity = getInstrumentation().waitForMonitorWithTimeout(monitorMedicalAnalysisReportActivity,5000);
        assertNotNull(medicalAnalysisReportActivity);
        medicalAnalysisReportActivity.finish();
    }

    @After
    public void tearDown() throws Exception {
        mActivity=null;
}
```

Scenario:	Count and show step			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Click step counter button	Application start to count step with sensors	Application start to count step with sensors	-
2	See count of steps with text after step counter running	Show number of steps correctly after step counter running	Show number of steps correctly after step counter running	-

Script of Test code for count and show step:

```
@Rule
    public ActivityTestRule<StepCounterActivity> mActivityTestRule = new ActivityTestRule<>(StepCounterActivity.class);
    private StepCounterActivity mActivity=null;
    @Before
    public void setUp() throws Exception {
        mActivity=mActivityTestRule.getActivity();
    }
    @Test
    public void testStartButton(){
        assertNotNull(mActivity.findViewById(R.id.start_button));
    }
    @Test
    public void testimageView2ImageView(){
        assertNotNull(mActivity.findViewById(R.id.imageView2));
    }
    @Test
    public void testCountTextView(){
        assertNotNull(mActivity.findViewById(R.id.count));
    }

    @After
    public void tearDown() throws Exception {
        mActivity=null;
}
```

Scenario:	Account Creation			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Sign Up with Facebook	Account creation with Facebook account	Account creation with Facebook account	-
2	Sign Up with Google	Account creation with Google account	Account creation with Google account	-

Script of Test code for account Creation:

```
@Rule
    public ActivityTestRule<LoginActivity> mActivityTestRule = new ActivityTestRule<>(LoginActivity.class);
    private LoginActivity mActivity=null;
    @Before
    public void setUp() throws Exception {
        mActivity=mActivityTestRule.getActivity();
}
@Test
    public void testsignInButton(){
        assertNotNull(mActivity.findViewById(R.id.signInButton));
    }

    @After
    public void tearDown() throws Exception {
        mActivity=null;
}
```

Scenario:	Facebook and Google Login			
Test Number	**Test Description / Input**	**Expected Result**	**Actual Result**	**Fix Action**
1	Login with Facebook	Login with Facebook account	Login with Facebook account	-
2	Login with Google	Login with Google account	Login with Google account	-

Script of Test code for Facebook and Google login:

```
@Rule
    public ActivityTestRule<LoginActivity> mActivityTestRule = new ActivityTestRule<>(LoginActivity.class);
    private LoginActivity mActivity=null;
    @Before
    public void setUp() throws Exception {
        mActivity=mActivityTestRule.getActivity();
}
@Test
    public void testlogin_button(){
        assertNotNull(mActivity.findViewById(R.id.login_button));
}

    @After
    public void tearDown() throws Exception {
        mActivity=null;
}
```

4. Additional Tests

4.1. Security Testing

CASE	TEST DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	TOOLS
1	Buffer Overflow	There should be enough memory on the device for the operations in the app.	Some devices which have not enough memory are restricted. So, there is no problem.	Google Play Store
2	Path Manipulation	Any input from user should not be used for path creating.	All paths are created by the devices without using any information from users.	Manual
3	Reverse Engineering	Debugging mode should be disabled.	In the app, debugging mode is disabled.	Manual
4	Input Abusing	The inputs from users should be checked whether the information in correct form.	All inputs are checked.	Manual
5	Authentication Misusing	The information about identification should be protected by permission and encryption	The security of authentication are ensured by Facebook and Google SDK	Firebase Authentication
6	Brute Force Login	The malware should not log in without any registering.	The app supports only Facebook and Google login. So, there is no problem	Manual
7	Database Privacy	Firebase is used as database.	The privacy satisfied by contract with Google.	Firebase Database

Tools for security testing:

- **Google Play Store**
  The store helps us to restrict the unwanted devices.
- **Firebase Authentication**
  Firebase provides us Facebook and Google SDK to encrypt and store the user information.
- **Firebase Database**
  The *Firebase* Realtime *Database* is a cloud-hosted *database*. Data is stored as JSON and synchronized in realtime to every connected client.

4.2. Performance Testing

CASE	TEST DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	TOOLS
1	Disk usage	It should not exceed 20 MB	The average value of disk usage is 14,09 MB	Amazon Device Farm
2	Starting time	The app starts in 0.1s	The average value of starting time 73ms	Amazon Device Farm
3	Network usage	The expected network usage should be under 128Kb/s in any operation	The average value of network usage is 57.65Kb/s in sending and retrieving operation	Android Profiler
4	Memory usage of the program	The memory usage should not exceed 256mb	The average value of memory usage is 66.79mb in 10 minutes.	Android Profiler

Tools for performance testing:

- **Amazon Device Farm**
  AWS Device Farm is an app testing service that lets you test and interact with your Android, iOS, and web apps on many devices at once, or reproduce issues on a device in real time. View video, screenshots, logs, and performance data to pinpoint and fix issues and increase quality before shipping your app.

- **Android Profiler**
  The Android Profiler tools provide realtime data for your app's CPU, memory, and network activity.

4.3. Device Testing

- Since HealthHub is an Android Application, it is not suitable for Load/Stress testing. Instead of these tests, we have implemented device testing.

CASE	TEST DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	TOOLS
1	Phone Test	The app should work on as many phones as possible	The number of running phones / the number of all phones = 141/161	Amazon Device Farm
2	Tablet Test	The app should work on as many tablets as possible	The number of running tablets / the number of all tablets =27/32	Amazon Device Farm
3	SDK Test	The app should not work on devices with an SDK lower than 15	The app cannot install into devices with an SDK lower than 15	Amazon Device Farm
4	Multiple Screen	There should not be any differences in interfaces on different screens	There are not any differences in interfaces on different screens	Amazon Device Farm

Tools for device testing:

- **Amazon Device Farm**
  AWS Device Farm is an app testing service that lets you test and interact with your Android, iOS, and web apps on many devices at once, or reproduce issues on a device in real time. View video, screenshots, logs, and performance data to pinpoint and fix issues and increase quality before shipping your app.

4.4. Acceptance Testing

CASE	TEST DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	TOOLS
1	Verify that users can login using their Google or Facebook account	The user can perfectly authenticate in the app.	This situation are tested by team members and customer. There is no problem.	Manual
2	Verify that healthmans can store and show information about medicine.	The inputs from healthman should be stored in database. These are retrieved from database and displayed in the screen.	This situation are tested by team members and customer. There is no problem.	Manual
3	Verify that healthmans can store and show information about sports.	The inputs from healthman should be stored in database. These inputs are retrieved from database and displayed in the screen.	This situation are tested by team members and customer. There is no problem.	Manual
4	Verify that healthmans can store and show information about medical analysis.	The inputs from healthman should be stored in database. These inputs are retrieved from database and displayed in the screen.	This situation are tested by team members and customer. There is no problem.	Manual
5	Verify that ‘step counter’ works well	The step counter should counts well as possible as	The accuracy rate of ‘step counter’ calculated by team members is 93%.	Manual
6	Verify that ‘health rate monitor’ works well	The ‘health rate monitor’ works well as possible as	The accuracy rate of ‘health rate monitor’ calculated by team members is 91%.	Manual
7	Verify that supervisor can perfectly comment	The comments from supervisor should be matched the correct healthman and be stored in database. The healthman displays these comments perfectly.	This situation are tested by team members and customer. There is no problem.	Manual
8	Verify that supervisor can follow healthman	The request from the supervisor should be matched the correct healthman.	This situation are tested by team members and customer. There is no problem.	Manual

Tools for acceptance testing:

- No testing tools are required for acceptance testing as it is a test conducted by the customer or the development team to check whether all earlier demands are satisfied or not.