Weather 24/7

Software Testing Document

Submitted by:

SH-01, Tahmid Mosaddek

FH-97, Kazi Shadman Sakib

Submitted to:

Dr. Saifuddin Md. Tareeq
Professor & Chairperson
Computer Science and Engineering, University of Dhaka

Dr. Sarker Tanveer Ahmed
Assistant Professor
Computer Science and Engineering, University of Dhaka

Table of contents

1. Unit Testing

1.1 Static Testing

1.1.1 Walkthrough

1.1.1.1 UI

1.1.1.1.1 Home

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.1.2 CurrentWeather

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.1.3 WeatherPrediction

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.1.4 Settings

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.1.5 LocationCardAdapter

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.1.6 LocationCardModel

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.1.7 PredictionCardAdapter

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.1.8 PredictionCardModel

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.2 Data

1.1.1.2.1 Cache

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.2.2 DataController

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.2.3 VolleyListener

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this interface. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this interface. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this interface.
Code guideline violations	This interface is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper method names were used when constructing this interface. The interface name was also given on proper usage of this interface. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.2.4 WeatherApiController

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.1.3 Notification

1.1.1.3.1 MemoBroadcast

List of concerns	Outcome
Uninitialized Variables	No uninitialized variables were found in this class. As all the uninitialized variables were removed from the codebase.
Undocumented empty block	No undocumented empty blocks were found in this class. All the methods that were used in this class were well documented and appropriately tutored.
No effect assignment	No useless or unsuccessful variables and methods were found in this class.
Code guideline violations	This class is well constructed and follows all the code guidelines, thus making it easier to detect errors in the early phases of building this software system. Proper attribute names and proper method names were used when constructing this class. The class name was also given on proper usage of this class. Thus no code guideline violations were found.
Code anomalies	No code anomalies were found when the software system was in its initial testing phase.
Structural anomalies	As the class is well constructed in a fairly scalable way using the object-oriented design and concept, structural anomalies were minimized to the greatest extent possible.

1.1.2 Code review

1.1.2.1 Reviewer: Abdullah Ibne Masud

Remarks:

***Backend*:** This is a simple weather application that provides the user with current weather information and weather prediction of any place.

I have looked through the backend part of the implementation. The project structure follows an MVVM architecture pattern. This project structure makes it easier for the developers to debug or further improve the application.

The code is free of uninitialized variables and undocumented blocks. The method functionalities are properly documented in the comments for every method in every class. The variable naming convention is intuitive, and therefore the code is easy to read. No obvious anomalies were found. The code is free of compilation errors.

***Frontend*:** The user interface of this application is very simple and user-friendly.

The layouts of the screens are designed using XML. Some sliding animations are also present in the codebase, which are stored in a different folder. The resources used to design the screen layout and the backgrounds are stored in a folder named drawable. The division of frontend files makes it easier for the developers to redesign or enhance the UI of the application.

1.1.2.2 Reviewer: Nayeem Uzzaman

Remarks:

***Backend*:** The backend part of the codebase is clean and concise. The implementation has followed every coding convention. The classes are organized in different folders according to their use. The backend implemented all the frontend functionalities. The frontend and the backend of the application have been integrated successfully.

***Frontend*:** The XML files are separated from the backend part. The resources are stored in the drawable folder. Layout files are stored in the layout folder. There is also an animation folder that contains the animation XML files that animate the transition between different activities.

1.2 Dynamic Testing

1.2.1 Black box testing

1.2.1.1 Range partitioning

1.2.1.1.1 CalculateAQI

This method calculates the value of the air quality index based on the parameters: $PM_{2.5}$, $PM_{10,}$ and O_3. The ranges of valid input for these parameters are 0-500 for $PM_{2.5}$, 0-604 for PM_{10}, and 0-604 for O_3. This method throws an exception if negative values are provided as parameters. If the value of the parameters is above the range, this method normally executes but returns the maximum valid output for AQI which is 500.

Test case - Below the range
Input:

$PM_{2.5}$	-8.0
PM_{10}	15.0
O_3	-70.0

Test stub:

```
@Test
public void belowRange() {
   int result1 = DataController.calculateAQI(
           -8.0f,
           15.0f,
           -71.0f
   );
   Assert.assertTrue(result1 > 0 && result1 < 150);
}
```

Obtained output:

```
Arguments can not be negative
java.lang.IllegalArgumentException: Arguments can not be negative
```

Desired output: Exception

Test case - Within the range
Input:

$PM_{2.5}$	32.3
PM_{10}	53.4
O_3	25.0

Test stub:

```
@Test
public void belowRange() {
   int result1 = DataController.calculateAQI(
           32.3f,
           53.4f,
           25.0f
   );
   Assert.assertTrue(result1 == 50);
}
```

Obtained output: 50
Desired output: 50

Test case - Above the range
Input:

$PM_{2.5}$	1000.6
PM_{10}	2000.6
O_3	1276.4

Test stub:

```
@Test
public void aboveRange() {
   int result3 = DataController.calculateAQI(
           1000.6f,
           2000.6f,
           1276.4f
   );
   Assert.assertTrue(result3 > 0 && result3 <= 500);
}
```

Obtained output: AQI value is between 0 and 500
Desired output: 500

1.2.1.2 Set partitioning

1.2.1.2.1 getDayOfTheWeek

This method parses a date and returns the name of the day of the week, e.g. Saturday. The set of all valid discrete input for this method is the set of all dates with this format, `yyyy-MM-dd`. If this method fails to parse a date it will return “The day after tomorrow” and throw an exception.

Testcase - Valid discrete value

Input:

Date	“2022-04-04”

Test stub:

```
@Test
public void test1GetDayOfTheWeek() {
   String result = DataController.getDayOfTheWeek("2022-04-04");
   Assert.assertEquals("Monday", result);
}
```

Obtained output: Monday
Desired output: Monday

Testcase - Invalid discrete value

Input:

Date	“gibberish”

Test stub:

```
@Test
public void test2GetDayOfTheWeek() {
   String result = DataController.getDayOfTheWeek("gibberish");
   Assert.assertEquals("The day after tomorrow", result);
}
```

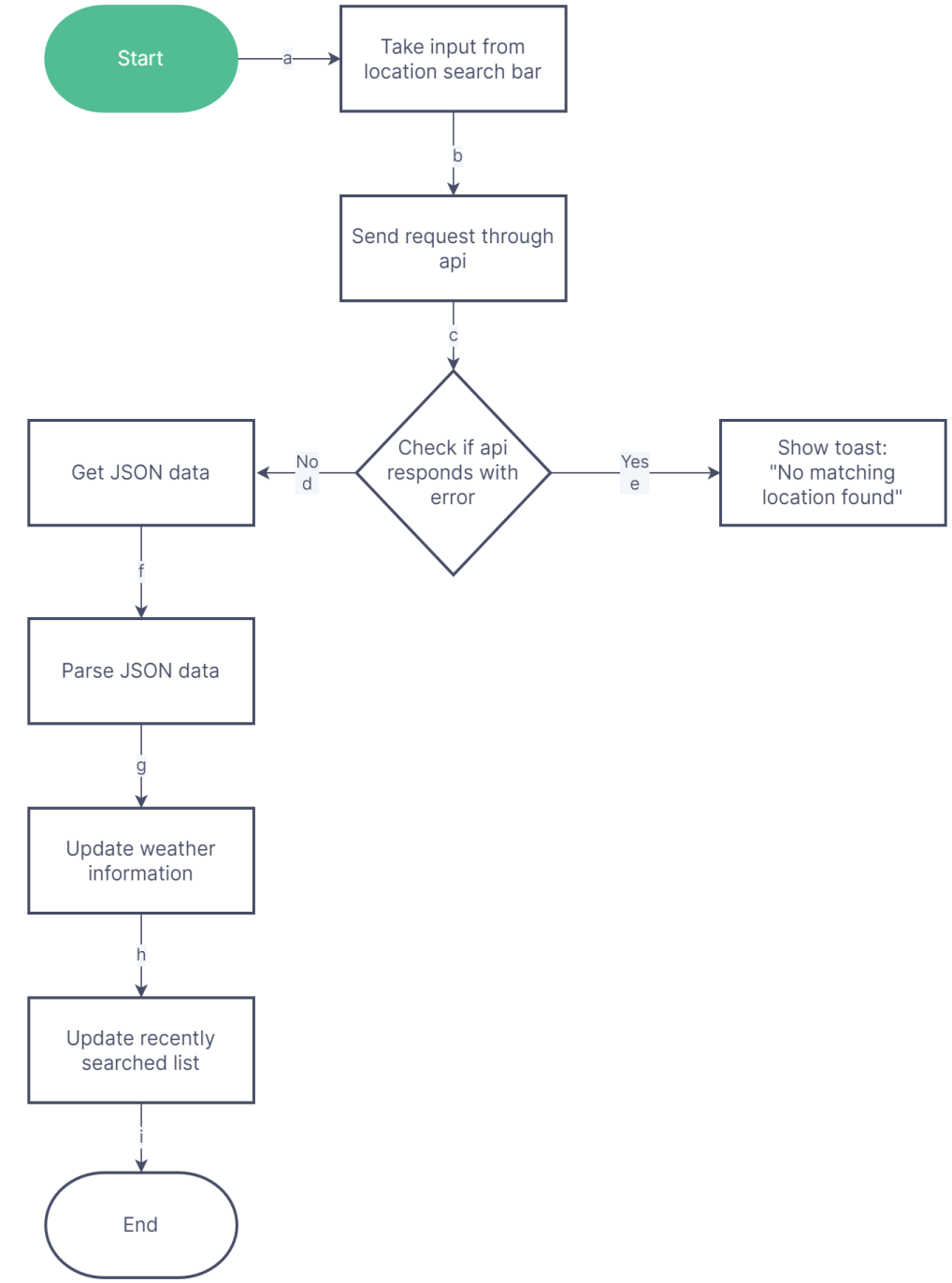
Obtained output:
parseException occurred
“The day after tomorrow”
Desired output: “The day after tomorrow”

1.2.2 White box testing

1.2.2.1 Statement/Code coverage

1.2.2.1.1 Home

Control flow diagram:



Test case - 1:

Input:
“Karachi”

Test stub:

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class HomeTest {

   @Before
   public void setUp() {
       ActivityScenario<Home> activityScenario =
ActivityScenario.launch(Home.class);
       activityScenario.moveToState(Lifecycle.State.RESUMED);
   }

   @Test
   public void inputValidLocation() {
       String validLocation = "Karachi";

Espresso.onView(withId(R.id.searchLocationBar)).perform(ViewActions.click());

Espresso.onView(isAssignableFrom(EditText.class)).perform(ViewActions.typeText(vali
dLocation),
               ViewActions.pressKey(KeyEvent.KEYCODE_ENTER));
   }

   @After
   public void tearDown() {
   }
}
```

Statement coverage:
a→b→c→d→f→g→h→i

Obtained output:
Updated weather information and updated recently searched card list.

Test case - 2:

Input:
“abcd”

Test stub:

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class HomeTest {

   @Before
   public void setUp() {
       ActivityScenario<Home> activityScenario =
ActivityScenario.launch(Home.class);
       activityScenario.moveToState(Lifecycle.State.RESUMED);
   }

  @Test
public void inputInvalidLocation() {
   String invalidLocation = "abcd";
   Espresso.onView(withId(R.id.searchLocationBar)).perform(ViewActions.click());

Espresso.onView(isAssignableFrom(EditText.class)).perform(ViewActions.typeText(inva
lidLocation),
           ViewActions.pressKey(KeyEvent.KEYCODE_ENTER));
}


   @After
   public void tearDown() {
   }
}
```

Path:
a→b→c→e

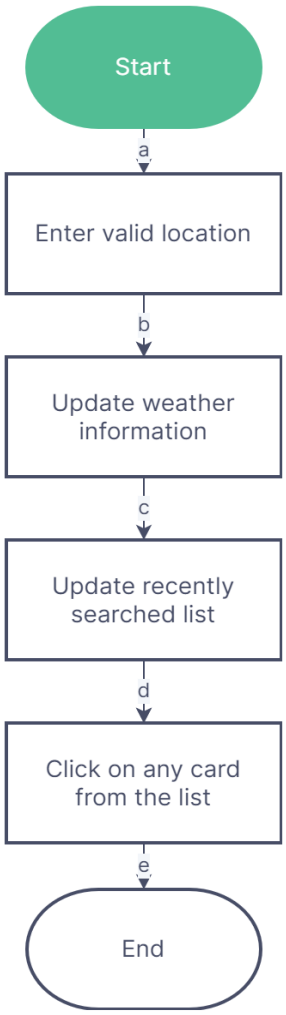
Obtained output:
A toast showing the message “No matching location”.

Coverage

Process link	a	b	c	d	e	f	g	h	i
abcdfghi	✔	✔	✔	✔		✔	✔	✔	✔
abce	✔	✔	✔		✔				

1.2.2.1.1 Home (Continued)

Control flow diagram



Test case - 1

Test stub:

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class HomeTest {

   @Before
   public void setUp() {
       ActivityScenario<Home> activityScenario =
ActivityScenario.launch(Home.class);
       activityScenario.moveToState(Lifecycle.State.RESUMED);
   }

@Test
public void selectRecentlySearchedLocation() {
   String firstLocation = "Delhi";
   String secondLocation = "London";

   Espresso.onView(withId(R.id.searchLocationBar)).perform(ViewActions.click());

Espresso.onView(isAssignableFrom(EditText.class)).perform(ViewActions.typeText(firs
tLocation),
           ViewActions.pressKey(KeyEvent.KEYCODE_ENTER));

   Espresso.onView(withId(R.id.searchLocationBar)).perform(ViewActions.click());

Espresso.onView(isAssignableFrom(EditText.class)).perform(ViewActions.typeText(seco
ndLocation),
           ViewActions.pressKey(KeyEvent.KEYCODE_ENTER));


Espresso.onView(withId(R.id.recycler_view)).perform(RecyclerViewActions.actionOnIte
mAtPosition(1, ViewActions.click()));

}


   @After
   public void tearDown() {
   }
}
```

Path:

a→b→c→d→e

Obtained output:
Updated weather information with the selected card location.

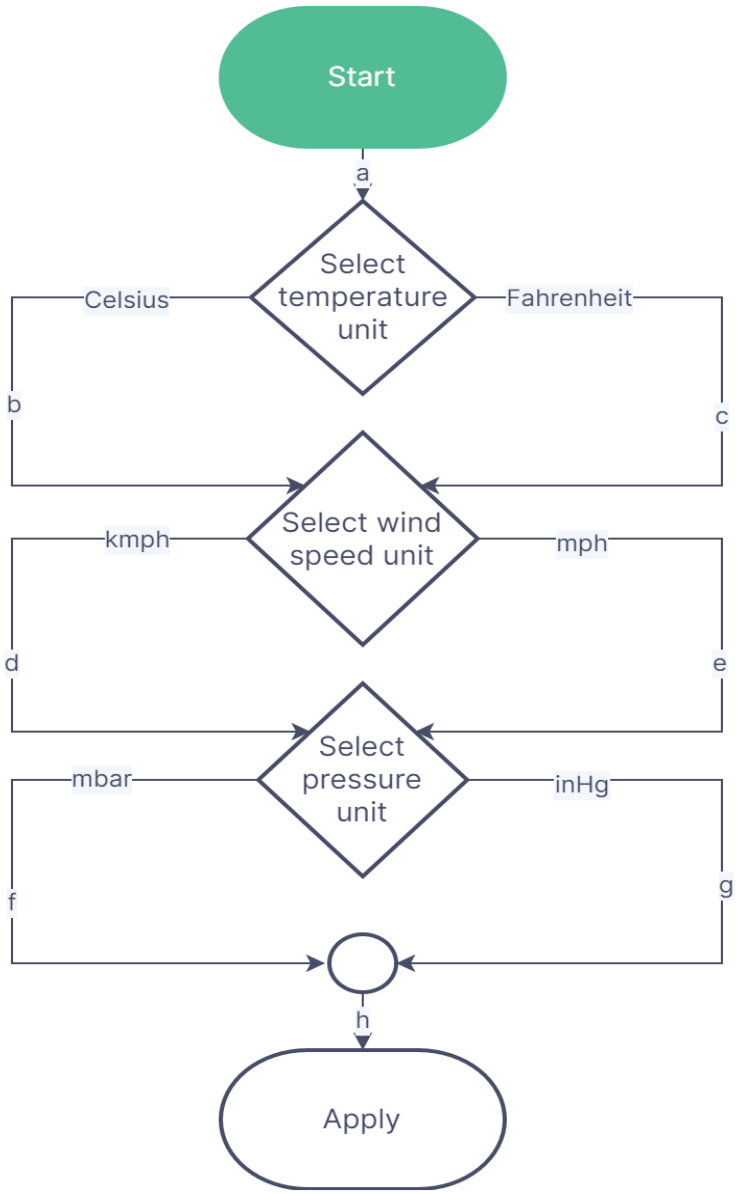
Coverage

Process link	a	b	c	d	e
abcde	✔	✔	✔	✔	✔

1.2.2.2 Branch and conditional coverage

1.2.2.2.1 Settings

Control flow diagram



Test case - 1
Input:

Temperature unit	Celsius
Wind speed unit	mph
Pressure unit	mbar

Test stub:

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class SettingsTest {

   @Before
   public void setUp() {
       ActivityScenario<Settings> activityScenario =
ActivityScenario.launch(Settings.class);
       activityScenario.moveToState(Lifecycle.State.RESUMED);
   }

   @Test
   public void test1Settings() {
       Espresso.onView(withId(R.id.temperatureUnits)).perform(click());
       Espresso.onData(anything()).atPosition(1).perform(click());

       Espresso.onView(withId(R.id.windSpeedUnits)).perform(click());
       Espresso.onData(anything()).atPosition(0).perform(click());

       Espresso.onView(withId(R.id.pressureUnits)).perform(click());
       Espresso.onData(anything()).atPosition(1).perform(click());

       Espresso.onView(withId(R.id.settingsApply)).perform(click());
   }

   @After
   public void tearDown() {
   }
}
```

Path:
a→b→e→f→h

Obtained output:
Weather information updated with selected units.

Test case - 2

Input:

Temperature unit	Fahrenheit
Wind speed unit	kmph
Pressure unit	inHg

Test stub:

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class SettingsTest {

   @Before
   public void setUp() {
       ActivityScenario<Settings> activityScenario =
ActivityScenario.launch(Settings.class);
       activityScenario.moveToState(Lifecycle.State.RESUMED);
   }

 @Test
public void test2Settings() {
   Espresso.onView(withId(R.id.temperatureUnits)).perform(click());
   Espresso.onData(anything()).atPosition(0).perform(click());

   Espresso.onView(withId(R.id.windSpeedUnits)).perform(click());
   Espresso.onData(anything()).atPosition(1).perform(click());

   Espresso.onView(withId(R.id.pressureUnits)).perform(click());
   Espresso.onData(anything()).atPosition(0).perform(click());

   Espresso.onView(withId(R.id.settingsApply)).perform(click());
}


   @After
   public void tearDown() {
   }
}
```

Path:
a→c→d→g→h

Obtained output:
Weather information updated with selected units.

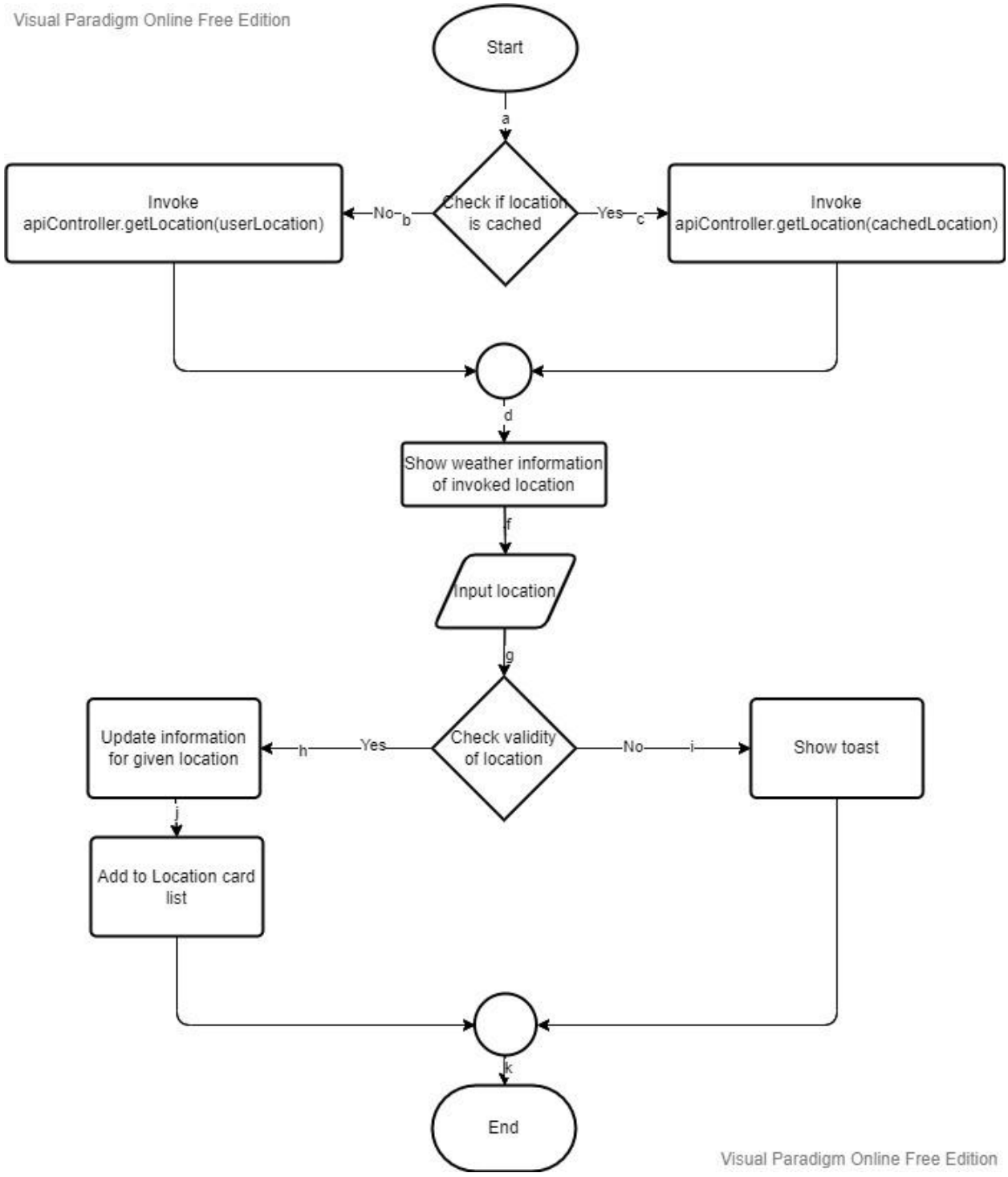
Coverage

Process links	a	b	c	d	e	f	g	h
abefh	✔	✔			✔	✔		✔
acdgh	✔		✔	✔			✔	✔

1.2.2.3 Path coverage

1.2.2.3.1 Home

Control Flow Diagram



Test case - 1

Input:

Cache	empty
Location search bar	“Mumbai”

Expected output:
Before search- Weather information of the local area.
After search- Weather information of Mumbai.

Obtained output:
Before search- Weather information of Dhaka.
After search- Weather information of Mumbai.

Path:
a→b→d→f→g→h→j→k

Test case - 2

Input:

Cache	empty
Location search bar	“blabla”

Expected output:
Before searching- Weather information of the local area.
After searching- Pop up toast.

Obtained output:
Before searching- Weather information of Dhaka.
After searching- A toast shows “No matching location found”.

Path:
a→b→d→f→g→i→k

Test case - 3

Input:

Cache	“Shanghai”
Location search bar	“Kabul”

Expected output:
Before searching- Weather information of Shanghai.
After search- Weather information of Kabul.

Obtained output:
Before searching- Weather information of Shanghai.
After searching- Weather information of Kabul.

Path:
a→c→d→f→g→h→j→k

Test case - 3

Input:

Cache	“Shanghai”
Location search bar	“xyz”

Expected output:
Before searching- Weather information of Shanghai.
After search- Pop up toast.

Obtained output:
Before searching- Weather information of Shanghai.
After searching- A toast shows “No matching location found”.

Path:
a→c→d→f→g→i→k

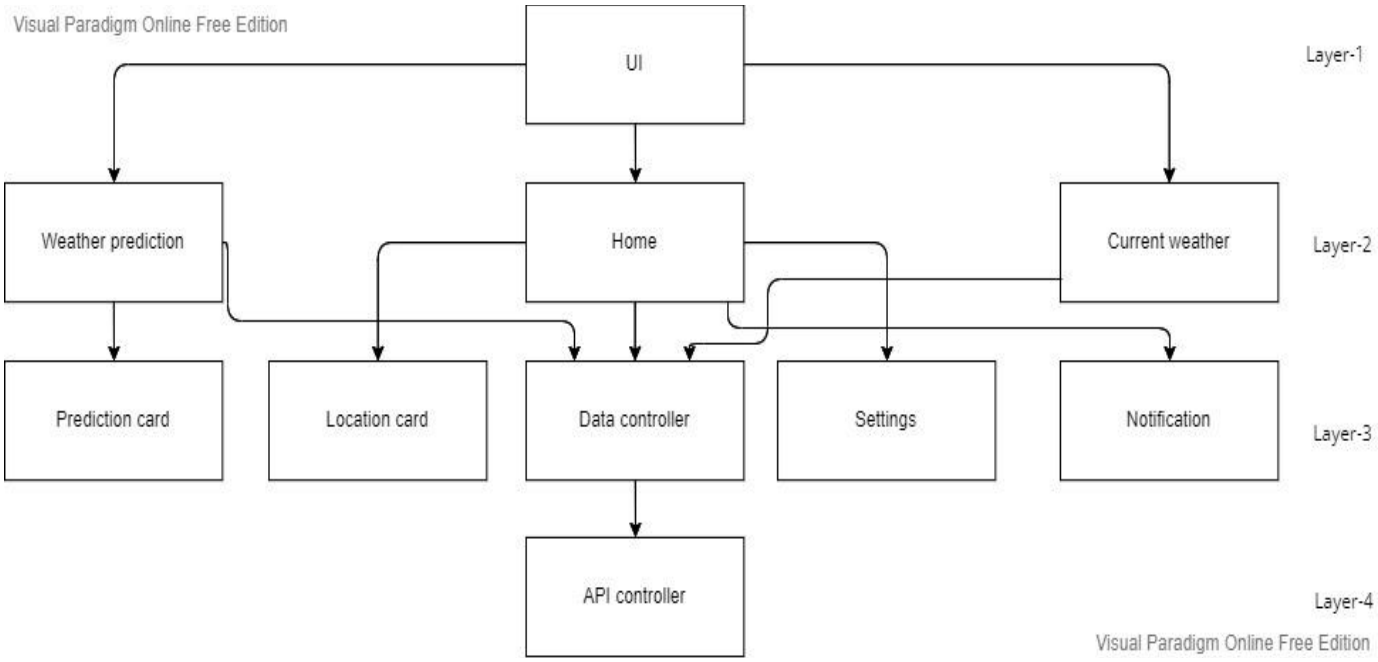
Path coverage

Process links	a	b	c	d	f	g	h	i	j	k
Tc-1 abdfghjk	✔	✔		✔	✔	✔	✔		✔	✔
Tc-2 abdfgik	✔	✔		✔	✔	✔		✔		✔
Tc-3 acdfghjk	✔		✔	✔	✔	✔	✔		✔	✔
Tc-3 acdfgik	✔		✔	✔	✔	✔		✔		✔

2. Integration Testing

2.1 Bottom-up testing

Diagram of system decomposition into three layers:



We chose to use the bottom-up approach for integration testing. At first, the subsystems of the bottom layer of the call hierarchy are tested individually. Then the subsystems of the immediately upper layer were tested, that call the previously tested subsystems. This process is repeated until we reach the top of the call hierarchy, which is the Home subsystem.

Layers	Selected modules	Interaction and interface testing	Subsystem dependencies testing
Layer-4	API controller	1. Sends requests to API 2. Receives JSON data.	Independent subsystem
Layer-3	Data controller, API controller	1. Parses JSON data. 2. Calculates AQI and day of the week.	Receives raw JSON data from API controller to parse.
	Notification	1. Sends notification at certain times.	Independent subsystem
	Location card	1. Contains the name, time, and local time of the place that was recently searched. 2. Clicking a card will show the weather information of that place.	Independent subsystem
	Prediction card	1. Shows the correct layout. 2. Expands when a collapsed Cardview is selected. 3. Collapses when an expanded Cardview is selected.	Independent subsystem
	Settings	1. Consists of three spinners of weather parameter units. 2. Clicking on ‘apply’ will update corresponding information in the whole system.	Independent subsystem
Layer-2	Home, Location card, Data controller, API controller, Settings.	1. Shows basic weather information fetched from the data controller. 2. Entering a valid location in the location search bar shows the weather	Depends on all the subsystems of layer-3 except the prediction card subsystem.

		information of that place and adds a card to the recently searched card list. 3. Pressing a location card from a recently searched location shows weather information of that place. 4. Pressing the settings icon navigates to the settings screen. 5. Basic weather information is used to send notifications at certain times.	
	Current weather, Data controller, API controller	1. Shows current weather information.	Depends on the data controller. Data controller provides the required parameter values.
	Weather prediction, Prediction card, Data controller, API controller	1. Shows three days forecast. 2. Presents prediction information with accurate data and units. 3. Pressing a prediction card shows detailed prediction data.	Show weather prediction data contained in the prediction card view. Depends on the prediction card and data controller.
Layer-1	UI, Home, Current weather, Weather prediction, Settings, Data Controller, API controller, Notification, Prediction card, Location card	1. Swiping left on the home page navigates to the current weather page. 2. Swiping right on the home page navigates to the weather prediction page. 3. Swiping left on the weather prediction page navigates to the home page.	Depends on all the subsystem of the bottom layers

		4. Swiping right on the current weather page navigates to the home page. 5. Swiping down on the home page invokes API controller methods to refresh the home page layout with new weather information 6. Swiping up on the home page, the current weather page and the weather prediction page scroll down and show the bottom part of the pages.	

3. System Testing

System testing is concerned with testing the behavior of the entire system of Weather247.

3.1. Functional Testing

The functional testing validates the functional requirements of Weather247. Test cases are designed from the requirement analysis document of Weather247. This section is centered around requirements and key functions of Weather247, and if the test cases validates the functional requirements of Weather247.

3.1.1 Home

Functional Requirement Test Case Number	User Activity	Result	Details
01	User opens the Weather247 software system to know about the current basic weather.	The text fields of the location and the weather information changes according to the user-specific location.	Input: User specific location. (Automatically) Results: Weather247 software system opens and fetches all weather information along with the weather status icon and changes it according to users location, and changes all the home information accordingly. Thus displays it. Status: OK.
02	User clicks on the search bar in Home and user inputs valid location and clicks on	The text fields of the location and the weather information	User Searched Location in Home: London.

	search to change all the software system's weather information according to that area's weather information.	changes according to the user searched inputs shown on the display. And a recently searched location card appears below the search bar.	Results: 1) Changes all the current basic weather information along with the weather status icon and changes it according to London's weather information on the home activity or page along with all other weather information. And thus displays it. 2) A recently searched location card appears below the search box. That shows the details of when the search was done along with the local time of that location. Status: OK.
03	Users click on the recently searched location cards to change all the weather information according to the clicked location card.	The text fields of location and the weather information changes according to the clicked location card. This action also updates the recently searched location card.	Input: Recently searched location Results: 1) Changes all the current basic weather information along with the weather status icon and changes it according to recently searched location's weather information in home activity or page along with all other weather information. And thus displays it. 2) Recently searched location cards keep appearing below the search box. That shows the details of when the search was done along with the local time of that location.

			Status: OK.
04	User clicks on the settings icon.	A settings popup window appears.	Input: Settings icon clicked. Results: A settings popup window appears that gives the user options to change the parameters of weather information. Status: OK.
05	Users swipe down to refresh all the weather information.	All the weather information refreshes.	Input: User specific location Results: All the weather information refreshes according to the user specific location, and displays accordingly. Status: OK.
06	User inputs an invalid location in the search bar of Home activity or page.	A toast pop up appears with the message saying invalid location.	Input: User Searched Invalid Location in Home: abcdef. Results : Weather 247 acknowledges the invalid location and shows a toast message/pop up message in the Home activity or page with the message “No matching location found”. Status: OK.
07	User opens Weather247 for the first time.	Notifications are setted for alerts according to the weather information.	Input : User gives Weather247 recently fetched information for the first time.

			Results: User gets 3 notification alerts daily on his/her device from the Weather247 software system. 1) Everyday at 8 AM the user gets a notification, alerting the user about today's weather prediction with an icon describing the weather condition. 2) Everyday at 8 PM the user gets a notification, alerting the user about tomorrow's weather prediction with an icon describing the weather condition. 3) Everyday the user gets alert about the current day weather prediction of rain or snow. Status: OK.

3.1.2 CurrentWeather

Functional Requirement Test Case Number	User Activity	Result	Details
01	User opens the Weather247 software system. And swipes	A new activity or page appears which	Inputs : User specific location (Automatically) / User Searched

	left to go to the current weather information page.	has current weather information.	Location / Recently searched location. Results: A CurrentWeather new activity or page appears, fetches all current weather information according to users location / user searched location or recently searched location and changes all the detailed current weather information accordingly. Thus displays it. Status: OK.

3.1.3 WeatherPrediction

Functional Requirement Test Case Number	User Activity	Result	Details
01	User opens the Weather247 software system. And swipes right to go to the weather information prediction information page.	A new activity or page appears which has weather prediction information of 3 days.	Inputs : User specific location (Automatically) / User Searched Location / Recently searched location. Results: 1) A WeatherPrediction new activity or page appears, fetches all predicted weather information according to users location / user searched location or recently searched location and changes all the detailed

			predicted weather information accordingly. Thus displays 3 predicted weather cards of today, tomorrow and the next day. Status: OK.
02	User opens the Weather247 software system. And swipes right to go to the weather information prediction information page and clicks on one of the 3 predicted weather cards of today, tomorrow or the next day.	The clicked predicted weather card expands and shows detailed predicted weather information.	Inputs : User specific location (Automatically) / User Searched Location / Recently searched location. Results: 1) The clicked predicted weather card of Today, Tomorrow or the next day expands beautifully and shows detailed weather information of the next 3 days. 2) By clicking the predicted weather card again the card contracts again. Status: OK.

3.1.4 Settings

Functional Requirement Test Case Number	User Activity	Result	Details
01	User opens the Weather247 software system. And clicks on the settings icon in the	All the weather information	Input: Parameters selected by the user.

	home. And changes the parameters shown on the settings activity and clicks on apply button.	parameters change according to the user's inputs.	Results: All the weather information parameters change according to the user's choice of parameter when the apply button is clicked. The parameter changes occur on the whole Weather247 system. Thus displaying the results on the Weather247 system. Status: OK.

3.2. Performance Testing

The performance testing validates the non-functional requirements of Weather247. Test cases are designed from violating the non-functional requirements of Weather247. This section is centered around non-functional requirements of Weather247, and if the test cases validates the non-functional requirements of Weather247.

3.2.1 Security Testing

1. Test Case (Security based on User Location):

- The system will use the user's device location using the Geolocation API that returns a location and accuracy radius based on information about cell towers and WiFi nodes that the mobile client can detect. The system will use this information only when the application is being used.

Details: As the system will use the user's device location using the Geolocation API only when the application is being used, and communications between the application and the weather APIs are

done over HTTPS using the POST method. As both the request and response messages will be encrypted, the user's location will not be leaked if someone eavesdrops on the data packets. Thus, the security of user location is ensured.

2. Test Case (Security based on User Authentication) :

- In the Weather247 android system, the user does not need to login or register using his/her personal information.

Details : As the user does not need to login or register using his/her personal information, it ensures that the users' data are not accessible anywhere on the internet.

3. Test Case (Security based on Code Quality) :

- No deprecated codes, methods or APIs were used for security and stability issues of the software system.

Details: Deprecated codes, methods or APIs mean that in the future this code, method or API will no longer be supported. This method might not be in the code/methods/APIs but in one of the libraries that one uses. As the deprecated codes, methods or APIs will no longer be supported in the future, security issues and stability issues may rise up. Thus, for these reasons, all the deprecated codes, methods or APIs were removed and newer code/methods/APIs were used to secure our software system data and for securing our stability of the software system.

4. Test Case (Security based on Improper Platform Usage):

- The software system only allows the users to search locations, so any sort of misuse is not possible through our user interface.

5. Test Case (Security based on Data Storage):

- The retrieval of the recently saved data from the private data storage or cache is only done when there is no wifi/internet connection in the users' device or the user opened the Weather247 recently.

Details: Android allows our Weather247 android application to save or cache private data or files on the device memory, which is called internal storage. Data saved or cached in internal storage are private and can be accessed only within Weather247, other applications cannot access them.

The retrieval of the recently saved data from the private data storage or cache is only done when there is no wifi/internet connection in the users' device or the user opened the Weather247 recently. Thus making sure that the data is saved or recovered only when it is needed.

3.2.2 Timing Testing

The testing was conducted in the Android Studio. The specifications of the computer that is used to run the tests are given below:

RAM: 16 GB
Memory: 1 TB(HDD)
GPU: NVidia Geforce GTX 1050

3.2.2.1 Home:

- The following method stub was used to test the execution time of detecting the user location and fetch data of location in Home Activity module:

```
@Test
public void timingTestOnDataControllerParseLocation(){

    long startTime = System.nanoTime();

    String result = DataController.getRegion();

    System.out.println("Time Taken to get Location " + result + "
: " + (System.nanoTime() - startTime) * 0.001 + " mS\n");

}
```

Input	Timing	Output
DataController.getRegion();	16.4260 mS	Dhaka

- The following method stub was used to test the execution time of parsing and fetching data of current temperature in Home Activity module :

```
@Test
public void timingTestOnDataControllerHome(){

    long startTime = System.nanoTime();

    String result = DataController.getCurrentTemperatureHome();

    System.out.println("Time Taken to get Home Temperature " +
result + " : " + (System.nanoTime() - startTime) * 0.001 + "
mS\n");

}
```

Input	Timing	Output
`DataController.getCurrentTemperatureHome()`	`11.82 mS`	`30`

- <u>If device location is turned off :</u>

  If the device of the User is turned off, the system will detect it and will ask the user to turn on the location in less than 1 seconds.

- <u>If device wifi or internet connection is turned off :</u>

The retrieval of the recently saved data from the private data storage or cache is only done when there is no wifi/internet connection in the users' device or the user opened the Weather247 recently.

Input	Timing	Output
`weatherApiController.getSavedJsonData(this, readSavedData());`	~1s	Displays recently saved data when there is no wifi/internet connection in the users' device or the user opened the Weather247 recently.

- <u>Home backgrounds change timing :</u>

Input	Timing	Output
`if(nowTimeInt >= currentSunriseInt && nowTimeInt < currentSunsetInt)`	~1s	Displays the day background.
`else if(nowTimeInt >= currentSunsetInt && nowTimeInt < currentSunsetFinishTimeInt)`	~1s	Displays the sunset background.

3.2.2.2 Current Weather :

- The following method stub was used to test the execution time of parsing and fetching data of CurrentAQI in Current Weather Activity module :

```
@Test
public void timingTestOnDataControllerCurrentWeather(){

    long startTime = System.nanoTime();

    String result = DataController.getCurrentAQI();

    System.out.println("Time Taken to get Current AQI " + result
+ " : " + (System.nanoTime() - startTime) * 0.001 + " mS\n");

}
```

Input	Timing	Output
`DataController.getCurrentAQI();`	`16.4260`	`7.3020 mS`

- Current Weather backgrounds change timing :

Input	Timing	Output
`if(nowTimeInt >= currentSunriseInt && nowTimeInt < currentSunsetInt)`	~1s	Displays the day background.
`else if(nowTimeInt >= currentSunsetInt && nowTimeInt < currentSunsetFinishTimeInt)`	~1s	Displays the sunset background.

3.2.2.3 Weather Prediction :

- The following method stub was used to test the execution time of parsing and fetching data of PredictedAvgTemp in Weather Prediction Activity module :

```
@Test
public void timingTestOnDataControllerWeatherPrediction(){

    long startTime = System.nanoTime();

    String result = DataController.getPredictedAvgTemp()[0];

    System.out.println("Time Taken to get Predicted Average
Temperature " + result + " : " + (System.nanoTime() - startTime)
* 0.001 + " mS\n");

}
```

Input	Timing	Output
`DataController.getPredictedAvgTemp()[0];`	`6.992 mS`	`27.5°C`

- Weather Prediction backgrounds change timing :

Input	Timing	Output
`if(nowTimeInt >= currentSunriseInt && nowTimeInt < currentSunsetInt)`	~1s	Displays the day background.
`else if(nowTimeInt >= currentSunsetInt && nowTimeInt < currentSunsetFinishTimeInt)`	~1s	Displays the sunset background.

3.2.3 Volume Testing

The testing was conducted in the Android Studio. The specifications of the computer that is used to run the tests are given below:

RAM: 16 GB
Memory: 1 TB(HDD)
GPU: NVidia Geforce GTX 1050

In the case of volume testing, the only data we are handling in Weather247 is the API response and the searched location that the user inputs.

So, as for the API response it has to be precise for DataController to parse the data according to its structure. The cache storage or the internal storage where this API response is stored only costs ~ 50 kB.

In the case of user searched location inputs, we inserted a significant number of inputs in the search bar, and repeated this action after 1-2 seconds for some significant amount of time. We always got the correct and precise outputs for the valid inputs of the searched location. And the time taken to get all the outputs was less than 1 second. Results were displayed before the user's in a blink of an eye. Hence, we can safely assume that our web application works certainly well under a big amount of user data.

3.3 Acceptance Testing

3.3.1 Alpha Test

For the Alpha test, we took feedback from mock clients (i.e our classmates) in the development environment. As a result, we discovered some small bugs and inconsistencies in Weather247. Thus, we had to make corrections and modifications based on those peer reviews of Weather247.

The feedback and modification/corrections of the alpha testing phase are described below:

Serial No.	Feedback	Modification
01	All the java classes were under one package, which was looking really messy to look at. And it was taking some time to find the proper specific java class when searching for one.	All the java classes were divided into 3 main packages (ui, notification and data). The UI package has a sub-package named cards, which has 2 more sub-packages named, locationcard and predictioncard.
02	There was no way of handling errors if the user device had no wifi or internet connection.	A method was introduced, (`isConnectedToInternet(this)`), which checks if there is an internet connection in users' device. If there is no internet connection then Weather247 displays all the weather information according to recently saved information from the internal storage.
03	There was no way of handling errors if the user searched an	A method was introduced `invalidSearchRequestFinished()`,

	invalid location in the search bar.	Which triggers with the message “No matching location found”, when the user searches for an invalid location.
04	There was no way of handling errors if the location of the user device was turned off.	A simple logical check was done here. If Weather247 finds that the user location == null, then a toast will be displayed to the user saying “ Please turn on your location to get updated information!”.
05	Some methods were deprecated / outdated by newer methods.	We searched for the newer methods which correspond to that particular deprecated method and implemented it. Thus, this made our software system more efficient and stable.

3.3.2 Beta Test

For the Beta test, we took feedback from mock clients (i.e our classmates) after finishing our system. As a result, we discovered some small bugs and inconsistencies in Weather247. Thus, we had to make corrections and modifications based on those peer reviews of Weather247.

The feedbacks and modification/corrections of the beta testing phase are described below:

Serial No.	Feedback	Modification
01	A mock client wanted to know about the local time of the searched location.	We added this new feature, where the user can know about the local time of the searched location, in the recently searched location card of Home.
02	Notification alert was not very descriptive.	We added more descriptions about today/tomorrow's weather in the notification.
03	The location textview does not align properly in Home UI.	Adjusted the textview programmatically to align properly in Home UI.
04	In Current Weather Activity, the 4 timings were not showing future 4 hours prediction beforehand.	Proper logic was implemented in the DataController method for showing the future 4 hours prediction beforehand in the current weather activity.
05	Weather status textview does not align properly in Home UI.	Adjusted the textview programmatically to align properly in Home UI.
06	Default android icon was not matching with the theme of Weather247.	By matching with the purpose of Weather247, changed the Weather247 application icon.