كلية علوم الحاسب والمعلومات
College of Computer and Information Sciences

مصرف الراجحي
Al Rajhi Bank

Student name :	Student ID :	Student Email :
Aliah abdulrhman alhameed	440020584	ahameed@imamu.edu.sa
Arwa nasser alqahtani	440018921	Anhalqahtani21@imamu.edu.sa
Gamra Abdulrahman Almutairi	440019252	qaaalmutairi@sm.imamu.edu.sa
Rahaf meshaal Alotibi	440019590	rmmalotibi@sm.imamu.edu.sa

Dr. Lames Alhazza

Table of content :

Purpose of the workshop :

This report will define Alrajhi bank application's quality in different aspects including source code quality, documents, design and further than that will be described in the report. The main purpose of this workshop is to master and study the chosen banking app and evaluate it. In addition to proposing solutions to any issues that might be found.

The goal of workshop :

Our goal in this workshop is to provide a report that will solve quality issues found in the banking app. One of the main objectives that we aim to achieve is to make this banking app's software face the desired quality standards in order to help the developers of this app and make their work much easier.

Learned lessons :

Many valuable lessons were learned during the process of evaluating the general quality of this banking app. Such as the importance of quality attributes and following them helps the code to be highly readable, understandable, maintainable and more.
In addition to the fact that it's almost impossible for a software system to achieve all of the quality attributes as some of them might conflict.

MOBILE APP ANALYSIS :

In the first step, we downloaded docker to activate the MobSF tool, because the normal way to activate the tool did not work with us, and this is the first problem we encountered in analyzing the program. Second step, we activate the docker tool through terminal, we used the comment "sudo docker pull" and the rest of the static analysis to run it. Next, after running the Docker tool, we moved to the Google Play program to copy the link of the program “Alrajhi bank” Android version to download the Apk. Lastly, we had a problem downloading the Apk file using “Apk pure”, it did not recognize alrajhi link. Instead of it we used “Apk downloader” site, After downloading the alrajhi apk file , we finally moved to the "mobSF" tool as a last step to analyze the file and extract the source code.

1.STRUCTURE :

1.1 Meaningful identifiers :

Most of the identifiers used in the source code are neither well identified nor clear enough. this could be an issue for other developers to understand the code. Here are some examples:

```
boolean z2;
boolean z3;
int width;
int i3;
int i4;
int i5;
int height;
int i6;
int i7;
int i8;
```

1.2 Vertical alignment :

Unfortunately most of the source code hasn't taken vertical alignment into consideration. The code would be easier to read with them.

```
public final class f {
    public static final int abc_action_bar_title_item = 0;
    public static final int abc_action_bar_up_container = 0;
    public static final int abc_action_menu_item_layout = 0;
    public static final int abc_action_menu_layout = 0;
```

1.3 Standard code structure :

The overall structure is good and well taken care of . Code follows a certain and only one structure standard such as the curly braces along with the class's header in the same line and including curly braces with any if statement. However a tiny issue with the structure was detected which is adding simicolon after a single parenthesis as shown in the next figure:

```
            @Override // androidx.lifecycle.e
            public void c(androidx.lifecycle.g gVar, d.a aVar) {
                View view;
                if (aVar == d.a.ON_STOP && (view = Fragment.this.mView) != null) {
                    view.cancelPendingInputEvents();
                }
            }
        });
    }
}
```

1.4 Classes names :

A number of classes names were not as clear as it supposed to be some classes have names with a single letter. The main purpose of the class is not explicit.

```
public static class b {
    public final int a;
    public final Method b;

    public b(int i, Method method) {
        this.a = i;
        this.b = method;
        method.setAccessible(true);
    }
```

1.5 Proper data structure :

As shown in following figure the issue here is nested if-else block, as the number of conditions increases the complexity also increases ,readability reduced, and it may be an issue that makes the program testing a challenging process however, there's no logical issue.

The other figure shows a multiple of related attributes that were defined which may lead to a readability reduction.

```
    }
} else if (obj instanceof Float) {
    createMap.putString(r3, r5);
    createMap.putDouble(r2, ((Float) obj).doubleValue());
    return createMap;
} else if (obj instanceof Long) {
    createMap.putString(r3, r5);
    createMap.putDouble(r2, ((Long) obj).doubleValue());
    return createMap;
} else if (obj instanceof String) {
    createMap.putString(r3, C0201.m82(12436));
    createMap.putString(r2, (String) obj);
    return createMap;
} else if (obj instanceof Date) {
    createMap.putString(r3, C0201.m82(12437));
    createMap.putDouble(r2, (double) ((Date) obj).getTime
    return createMap;
} else if (Map.class.isAssignableFrom(obj.getClass())) {
    createMap.putString(r3, C0201.m82(12438));
    createMap.putMap(r2, e((Map) obj));
    return createMap;
} else if (List.class.isAssignableFrom(obj.getClass())) {
    createMap.putString(r3, C0201.m82(12439));
    List list = (List) obj;
    createMap.putArray(r2, d(list.toArray(new Object[list.size()])));
    return createMap;
} else if (obj instanceof DocumentReference) {
    createMap.putString(r3, C0201.m82(12440));
    createMap.putString(r2, ((DocumentReference) obj).getPath());
    return createMap;
} else if (obj instanceof Timestamp) {
    WritableMap createMap2 = Arguments.createMap();
    Timestamp timestamp = (Timestamp) obj;
    createMap2.putDouble(C0201.m82(12441), (double) timestamp.getSeconds());
    createMap2.putInt(C0201.m82(12442), timestamp.getNanoseconds());
    createMap.putString(r3, C0201.m82(12443));
    createMap.putMap(r2, createMap2);
```

```
public static final int TextAppearance_Compat_Notification = 0;
public static final int TextAppearance_Compat_Notification_Info = 0;
public static final int TextAppearance_Compat_Notification_Line2 = 0;
public static final int TextAppearance_Compat_Notification_Time = 0;
public static final int TextAppearance_Compat_Notification_Title = 0;
public static final int Widget_Compat_NotificationActionContainer = 0;
public static final int Widget_Compat_NotificationActionText = 0;
public static final int Widget_Support_CoordinatorLayout = 0;
```

1.6 Coherent Identifiers :

Only one variable style should be chosen and stick to it only. As shown in the following figures some of the classes used camelCaseIdentifiers and some others used underscores_identifiers.

```
public ImmLeaksCleaner(Activity activity) {
    this.a = activity;
}

private static void h() {
    try {
        b = 2;
        Field declaredField = InputMethodManager.class.getDeclaredField(C0201.m82(1001));
        d = declaredField;
        declaredField.setAccessible(true);
        Field declaredField2 = InputMethodManager.class.getDeclaredField(C0201.m82(1002));
        e = declaredField2;
        declaredField2.setAccessible(true);
        Field declaredField3 = InputMethodManager.class.getDeclaredField(C0201.m82(1003));
        c = declaredField3;
        declaredField3.setAccessible(true);
        b = 1;
    } catch (NoSuchFieldException unused) {
    }
}
```

```
public static final int TextAppearance_Compat_Notification = 0;
public static final int TextAppearance_Compat_Notification_Info = 0;
public static final int TextAppearance_Compat_Notification_Line2 = 0;
public static final int TextAppearance_Compat_Notification_Time = 0;
public static final int TextAppearance_Compat_Notification_Title = 0;
public static final int Widget_Compat_NotificationActionContainer = 0;
public static final int Widget_Compat_NotificationActionText = 0;
public static final int Widget_Support_CoordinatorLayout = 0;
```

1.7 Dependency injection :

There are some classes that are dependent to others and have dependency. Dealing with these classes should be very careful because any change in the class that has dependency may lead to an issue in the dependent class.

```
ckage android.support.v4.media;

port android.os.Bundle;

 compiled from: MediaBrowserCompat */
blic abstract class a {
    public abstract void a(String str, Bundle bundle, Bundle bundle2);

    public abstract void b(String str, Bundle bundle, Bundle bundle2);

    public abstract void c(String str, Bundle bundle, Bundle bundle2);
```

```
package android.support.v4.media;

import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Parcel;
import android.os.Parcelable;
import android.support.v4.media.d;
import android.support.v4.media.e;
import android.support.v4.media.session.MediaSessionCompat;
import android.text.TextUtils;

public final class MediaDescriptionCompat implements Parcelable {
    public static final Parcelable.Creator<MediaDescriptionCompat> CREATOR = n
a();
```

```
public class a52 extends r42 {
    public List<y42> f() {
        List<o42> e = d();
        ArrayList arrayList = new ArrayList();
        Iterator<o42> it = e.iterator();
        while (it.hasNext()) {
            arrayList.add((y42) it.next());
        }
        return arrayList;
    }
}
```

2. Readability :

In general the source code's readability is average since some quality measures have been violated such as consistent identifiers, meaning less naming, deep nesting , not enough comments and documentations. File and folder organization was helpful and good however the naming wasn't quit obvious what the purpose of each file is.

3. Reliability :

In the code there are many things that support reliability , ensure the quality and reliability of the program.
There are a number of class Exception functions that are used in the code , such as :
NullPointerException that deals with an entry that is null or does not contain an initial value.
and RuntimeException which deals with running problems.
Also, try and catch are used to catch and handle errors.

This supports reliability and ensures that the application is used continuously without any failure.

4. Reusability :

Overall reusability is satisfied and quite acceptable. There are a number of abstract classes and interfaces that has been reused like Android context, Android ActionMode and more.
Dependency injection is considered as a reusability measure which is found in the code source, creating more methods and overwriting codes have been avoided using dependency injection and depending on other classes.

5. Documentation and commenting in code :

Some of the keys that should be included in any documentations have not been found which could be very helpful such as mentioning any restrictions of the input values, returned value types, describing the main purpose of the methods and classes that their names are not quit obvious and describing what the code does.
However there's an advantage and a very well thought point of documenting , it is avoiding obvious comments when the text and code are well written and obvious, considering that overwriting obvious comments in this case may lead to redundancy and repeating your self, recall one of the quality measures is DRY principle.

6. Clean code :

The cleanliness of the code has standards that were not followed in many parts of this code, such as: not simplifying the code and making it easy to read in class a93 and also crowding the definitions of objects and variables in the ab2 method, therefore it would be much better to use spaces between each object and variable and make a space between each line. It is also verified that they do not follow the standards in creating and defining methods and Variable is not used (aa2, b, c, d and f) (variable like ac E , nb P ...) and this distracts the reader's understanding from developers and team members , this code also fulfills many principles, including DRY and single responsibility principle.

7. Maintainability

Maintainability is moderate since some quality metrics are not followed. For example meaningless identifiers and no comments. These may lead to an issue while maintaining the software product as developers might spend more time trying to understand the purpose of the code.

8.DESIGN :

8.1 Coupling :

Achieving low coupling is one of the main characteristics for a good design. The maximum coupling found between classes is almost equal to 4. Taken into consideration that this source code is for a banking app the coupling is pretty low which is a good thing!

8.2 Balancing the workload :

Unfortunately some of the files contain classes that have a higher workload than others, thus the workload is unbalanced across them. Nevertheless balancing work across the object is useful and important.
as shown in example below the workload is unbalanced between these two classes in the same file :

```
package android.support.v4.media;

import android.media.MediaDescription;
import android.net.Uri;

/* compiled from: MediaDescriptionCompatApi23 */
public class e {

    /* access modifiers changed from: package-private */
    /* compiled from: MediaDescriptionCompatApi23 */
    public static class a {
        public static void a(Object obj, Uri uri) {
            ((MediaDescription.Builder) obj).setMediaUri(uri);
        }
    }

    public static Uri a(Object obj) {
        return ((MediaDescription) obj).getMediaUri();
    }
}
```

```
    }

    public static void c(Object obj, CharSequence charSequence) {
        ((MediaDescription.Builder) obj).setDescription(charSequence);
    }

    public static void d(Object obj, Bundle bundle) {
        ((MediaDescription.Builder) obj).setExtras(bundle);
    }

    public static void e(Object obj, Bitmap bitmap) {
        ((MediaDescription.Builder) obj).setIconBitmap(bitmap);
    }

    public static void f(Object obj, Uri uri) {
        ((MediaDescription.Builder) obj).setIconUri(uri);
    }

    public static void g(Object obj, String str) {
        ((MediaDescription.Builder) obj).setMediaId(str);
    }

    public static void h(Object obj, CharSequence charSequence) {
        ((MediaDescription.Builder) obj).setSubtitle(charSequence);
    }

    public static void i(Object obj, CharSequence charSequence) {
        ((MediaDescription.Builder) obj).setTitle(charSequence);
    }
}
public static Object a(Parcel parcel) {
    return MediaDescription.CREATOR.createFromParcel(parcel);
}

public static CharSequence b(Object obj) {
    return ((MediaDescription) obj).getDescription();
}

public static Bundle c(Object obj) {
    return ((MediaDescription) obj).getExtras();
}

public static Bitmap d(Object obj) {
    return ((MediaDescription) obj).getIconBitmap();
}

public static Uri e(Object obj) {
    return ((MediaDescription) obj).getIconUri();
}

public static String f(Object obj) {
    return ((MediaDescription) obj).getMediaId();
}

public static CharSequence g(Object obj) {
    return ((MediaDescription) obj).getSubtitle();
}

public static CharSequence h(Object obj) {
    return ((MediaDescription) obj).getTitle();
}

public static void i(Object obj, Parcel parcel, int i) {
    ((MediaDescription) obj).writeToParcel(parcel, i);
}
}
```

8.3 Information Expert :

In this code there are many methods that contain information and share it to other methods or classes (when needed), not only sharing information, but also preserving this information and its inputs and storing it locally.. For example, class b0 shares the information of the object e in class a40 and this achieves the principle expert

8.4 Knowing responsability :

Object Responsibilities were achieved by achieving the first and second branches of it. the first branch was achieved due to the presence of objects in many classes to which values were assigned and stored , such as bArr in class a.

8.5 Doing responsability :

As for the second branch (Doing responsibility), this code contained many methods that performed arithmetic operations such as (getInterpolation, evaluate in class a6), and Methods performed assignments to variables such as (method a in class a9) and many operations.

9.SECURITY :

9.1 Access to apps :

The permission “android.permission.READ_PROFILE “ allows reading the user profile. It is dangerous because unauthorized pepole may enter and steal person's information. It is dangerous to read contact data because malicious apps can use it to send your data to other people.

9.2 Providing a handler for exceptions :

Exception handling is one of the best practices for a secured dependable code. As mentioned earlier at point 3 this source code handles and catches some exceptions therefore making it more dependable and securable.

9.3 Androids export activity:

Multiple activities were found not to be protected due to setting the value of the Android activity [android:exported] to true and some others due to the presence of intent-filter therefore the exported Android activity is set to true by default either way these activities are exposed and shared with other apps on the device making them accessible by other apps. This issue leads to lack of confidentiality.

9.4 Limit the visibility of information :

We previously talked about the lack of clarity of code symbols and its impact on the quality of the code from the perspective of quality and its standards (mentioned in point 1.1 and 1.4)
As for security, the ambiguity in the names of the codes is considered encryption of sensitive data, and this supports the security of the system and protects it from attacks, since the program cares greatly about security and protection thus , there is a balance between security and readability.

9.5 Cleartext network traffic :

One of the security issues detected is that Android clear text traffic attribute is set to “true” which indicates the app is using clear text traffic therefore, there’s a threat to data integrity as some information could be corrupted. Confidentiality is lacking as well, as there might be third parties that can leak users’ information.

9.6 External storage :

The permission “android.permission.WRITE_EXTERNAL_STORAGE” allows an application to read/modify/or even delete any external storage content, which is dangerous.

9.7 Launch mode activity:

The following activities (com.alrajhiretailapp.MainActivitylight) , (com.alrajhiretailapp.MainActivitydark) , and (com.alrajhiretailapp.MainActivity) were found causing a security issue. These activities should not have their launch mode attributes set to neither “single task” nor “single instance”. They should be set to “Standard” to avoid this issue.

DISCUSSION :

Through the whole process of evaluating the banking app quality and writing this report we have came to a conclusion of the importance and impact of quality on code, and how some of these quality attributes might conflict leading to a software product that achieves some quality attributes but on the other hand other properties violate its quality therefore the most important and needed quality attributes are what should be achieved.
The structure quality is not bad, there was a number of attributes, classes and methods with meaningless names due to attempting to limit the visibility of information which is indeed a higher priority and taking the security into consideration, nevertheless vertical alignment is an important key that facilitates code's readability unfortunately it wasn't paid any attention. Nested if-else blocks and the repetition of creating related data could be implemented in a better advanced way using appropriate data structure or any other solution. One thing that has a great impact on readability is commenting and documentation, they were barely included in the source code this issue leads to lack of code readability ,clarity, and maintainability.

In addition quality design is good, there is some minor violation in balancing the workload we found that the load is not well distributed, workload should be distributed between files and methods equally so that they have approximately the same workload for a balanced workflow.

As for security quality was not bad, but some security measures were violated and this leads to a security weakness. android.permission.WRITE_EXTERNAL_STORAGE was found that it leads to overwriting in external storage, this is dangerous and against security. To maintain the security of application, care must be taken to meet security requirements such as identification and authorization requirements. As well to ensure that people reliably access the application.

Another issue is setting the value of Cleartext network traffic to true, to prevent this issue we could set it “false” if it does not cause any conflicts, along with Android export activity issue, a possible solution is to set the Android export activity to "false" so that apps and information can't be shared with other apps on the device except for those activities that have an intent-filter since the Android export activity is set to “true” by default.

In a brief the quality is somehow satisfying and quite acceptable, although there was some quality measures weren’t followed due to a conflict and inconsistency between them therefore achieving the most important quality attributes and that has a greater impact on the software product is the main priority.

The proposed solutions and the quality assessment of the software product is from our simple point of view and understanding.

CONCLUSION :

The purpose of this report is to identify a number of different aspects of quality. In conclusion, we have covered the following aspects :

1. Code structure
2. Readability
3. Reliability
4. Reusability
5. Documentation and commenting
6. Cleanliness of code
7. Maintainability
8. Design
9. Security

Some of these aspects have been further described with different points that either violates its quality or doesn't.
In the code structure multiple points have been discussed such as meaningful identifiers, vertical alignment, classes names and identifiers , nested if-else blocks, dependency injection. Different design characteristics and principles have been followed and were described in the design aspect as well.

Also, from our point of view, we have suggested some possible solutions to solve the problems of structure , design and security.