Oum El Bouaghi University
Faculty of Exact Sciences and Natural and Life Sciences
Mathematics and Computer Sciences Department

Mobile Applications

Course, 3rd year Licence's degree in Computer Science, Speciality: Information Systems (SI)

Dr. Sofiane Zaidi, MCA

2026

Content

Chapter 1: Mobile application

2.1 Introduction

With the rapid evolution of mobile technologies, smartphones and portable devices have become an integral part of everyday life. At the core of these devices lies the mobile operating system, which plays a fundamental role in ensuring their functionality, usability, and performance. A mobile operating system not only manages hardware resources and wireless communication but also provides a platform for running applications and delivering services to users.

Over the years, mobile operating systems have evolved significantly, transitioning from early systems designed for Personal Digital Assistants (PDAs) to sophisticated platforms capable of supporting complex applications while efficiently managing limited resources such as battery life and processing power. This evolution has led to the emergence of a wide range of operating systems, which can be broadly classified into proprietary and open-source systems, each with its own advantages and ecosystem.

Today, the mobile operating system market is largely dominated by a few key players, shaping the direction of mobile computing and application development. Understanding these systems is essential for grasping how modern mobile applications are designed, deployed, and optimized.

2.2 Mobile operating systems

A mobile operating system (mobile OS) can be defined as software that enables a mobile device (smartphone, PDA, notebook, tablet, smartwatch, etc.) to function. It allows users to manage wireless connectivity (mobile network, Wi-Fi, Bluetooth, GPS, etc.), make phone calls, download applications, and configure and personalize their devices.

Since mobile operating systems are designed to run on small devices with limited battery life, they include advanced energy management and the ability to operate with constrained resources.

The first mobile devices equipped with an operating system were PDAs, invented in 1990. Since then, there has been a proliferation of mobile operating systems, including Blackberry, Symbian, Bada, RIM, iOS, Windows Phone, Ubuntu Touch, Firefox OS, Tizen, and Android.

In fact, mobile operating systems can be classified into two categories:

- Proprietary systems: these are systems designed to run on specific hardware; generally, both the operating system and the hardware are developed by the same

manufacturer. The source code of these systems is accessible only to their creators.

- Open-source systems: these are systems whose source code is available. Each manufacturer selects a version of the operating system and integrates it into their mobile device after adding their own software layer.

The global mobile operating systems market is currently dominated by three major companies: Google, Apple, and Microsoft. They respectively develop the operating systems Android, iOS, and Windows Phone. Figure 1.1 illustrates this dominance in global smartphone sales.

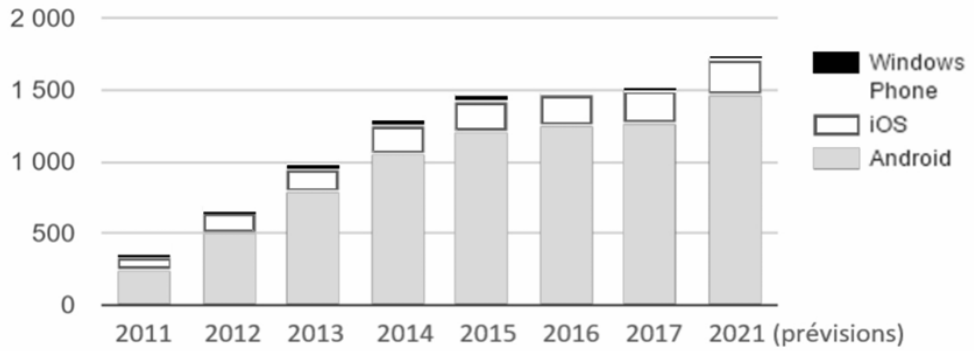


Figure 1.1: Distribution of global smartphone shipments by operating system.

This figure also illustrates the dominance of the Android OS over its competitors, with a market share exceeding 80%.

2.3 Types of mobile applications

1.3.1 iOS

iOS is the operating system developed by Apple for its mobile devices: iPhone, iPad, and iPod Touch. Founded in 1976, Apple released the first version of its mobile operating system in 2008, called iPhone OS (iOS 1). The transition to the second version (iOS 2) was accompanied by the release of the iPhone 3G and the iPod Touch; it also marked the first appearance of the App Store.

iOS 3, released in 2009, introduced several improvements such as cut, copy/paste functionality and MMS integration. In 2010, iOS 4 was launched, offering new features including multitasking and FaceTime. The fifth version was marked by the introduction of a notification center, followed by iOS 6, which did not bring major changes. iOS 7, released in 2013, introduced a new graphical interface design characterized by simple shapes, brighter colors, and adherence to responsive design principles. Among the new features of iOS 8 (2014) are Family Sharing and a redesigned App Store.

In 2015, iOS 9 introduced updated versions of existing applications (Notes, Maps,

Passbook, etc.), new multitasking modes, improved speed, and better battery life. iOS 10 (2016) introduced, among other things, a development kit for Siri and a new design. The first fully 64-bit version was iOS 11, which included a file management application allowing direct access to locally stored files and those in iCloud. The latest version mentioned here is iOS 12, announced in June 2018.

2.3.1 Windows Phone

Windows Phone is the operating system developed by Microsoft for its mobile devices: smartphones (Lumia, Asha, etc.), tablets (Microsoft Surface), and PDAs (Cortana). Microsoft, founded in 1975, specializes in developing operating systems (Windows 3, Windows 95, Windows 10, etc.) and software such as MS Office. In the early 2000s, Microsoft introduced its first mobile operating system called Windows Mobile, which was later replaced in 2010 by Windows Phone. Since 2011, Microsoft has adopted a strategy to unify its PC operating systems with Windows Phone. Its subsidiary "Microsoft Mobile" was created following the acquisition of Nokia in 2014.

During this period, Windows Phone 8 appeared in several versions (Apollo, GDR1 Portico, GDR2, GDR3, etc.), introducing new features such as:

- Support for multi-core processors, Full HD screens, and dual-SIM
- A new start screen displaying applications as tiles
- Integration of SkyDrive
- Addition of a notification center

It was not until November 2015 that Microsoft released a unified operating system, Windows 10, for both PC and mobile platforms.

3.3.1 Android

Android is an open-source operating system for mobile devices, originally created by a startup of the same name and acquired by Google in August 2005. The name comes from the term "android," referring to a human-like robot.

In November 2007, Google created the Open Handset Alliance (OHA), a group of more than fifty companies from various sectors (mobile operators, phone manufacturers, semiconductor companies, software publishers, distributors, etc.), including HTC, Sony, Dell, Intel, Samsung, and LG. Its goal was to promote and develop open-source standards for mobile devices, leading to the creation of Android.

Android has gone through multiple versions throughout its development. Starting from version 1.5, Google adopted a naming convention where each version is assigned a codename in alphabetical order, referring to desserts (cakes, candies, etc.). Versions 1.0 and 1.1 were not

officially named, although version 1.1 internally had the codename “Petit Four.”

The official list of Android versions along with their main features are summarized as following:

- Android 1.5 (Cupcake): introduction of virtual keyboard, video recording and sharing.
- Android 1.6 (Donut): new interface, support for different screen resolutions.
- Android 2.x (Eclair, Froyo, Gingerbread): performance improvements, hotspot sharing, Bluetooth, front camera support.
- Android 3.x (Honeycomb): tablet-optimized version, improved multitasking.
- Android 4.x (Ice Cream Sandwich, Jelly Bean, KitKat): UI improvements, Google Now integration.
- Android 5.x (Lollipop): redesigned interface, enhanced security, ART runtime.
- Android 6.x (Marshmallow): permission management, fingerprint support.
- Android 7.x (Nougat): multi-window support, VR platform (Daydream).
- Android 8.x (Oreo): adaptive icons, improved notifications, Google Play Protect.
- Android 9 (Pie): gesture-based interface, adaptive battery, digital wellbeing features

Chapter 2: Android Platform

2.1 Presentation of Android platform

Android is an open-source operating system (OS) developed by Google and the Open Handset Alliance (OHA) for mobile devices such as smartphones, tablets, televisions, car radios, smartwatches, and even smart helmets. The OHA is a coalition of telecommunications operators and mobile device manufacturers such as Samsung, Intel, HTC, SFR, Orange, Asus, Qualcomm, and others.

Android is based on Linux and is distributed under the Apache License v2. The separation between the hardware and software layers of Android allows the same application to run on any mobile device.

Since Android is an open-source framework, developers can access its source code to understand how it works, create customized versions of Android, and develop mobile applications.

The idea of Android emerged in 2003, when Andy Rubin, Rich Miner, Nick Sears, and Chris White founded Android Inc. in Palo Alto, California. Two years later, Google acquired the company. The first version of Android (1.0) was released in 2008 to support mobile applications on the HTC Dream phone. Table 1.1 presents the different versions of Android.

Version	Version Number	Release Date
Apple pie	1.0	23/09/2008
Banana bread	1.1	09/02/2009
Cupcake	1.5	30/04/2009
Donut	1.6	15/09/2009
Eclair	2.0, 2.1	26/10/2009
Froyo	2.2	20/05/2010
Gingerbread	2.3	06/12/2010
Honeycomb	3.0, 3.1, 3.2	22/02/2011
Ice Cream Sandwich	4.0	08/10/2011
Jelly Bean	4.1, 4.2, 4.3	09/07/2012
KitKat	4.4	31/10/2013
Lollipop	5.0, 5.1	17/10/2014
Marshmallow	6.0	05/10/2015

Nougat	7.0, 7.1	22/08/2016
Oreo	8.0, 8.1	21/08/2017
Pie	9.0	09/08/2018

Table 2.1 : Android Versions.

2.2 Fundamental components of Android application

The basic software architecture of Android is composed of several layers (as shown in Figure 2.1): the Linux kernel, libraries, Android runtime, application framework, and the application layer.

The first layer, the Linux kernel, acts as an interface between the hardware and the software. It provides essential services such as process management, memory management, and interaction with hardware through device drivers.

Above the Linux kernel is the libraries layer, which consists of a set of core libraries of the framework (often written in C/C++) as well as various open-source tools. Within this layer, we also find the Android runtime, which includes the Android virtual machine (Dalvik Virtual Machine) along with a set of Java libraries and ART (Android Runtime).

The application framework layer provides a set of interfaces that simplify the use of these libraries for developers.

Finally, the top layer, the application layer, contains all Android applications.

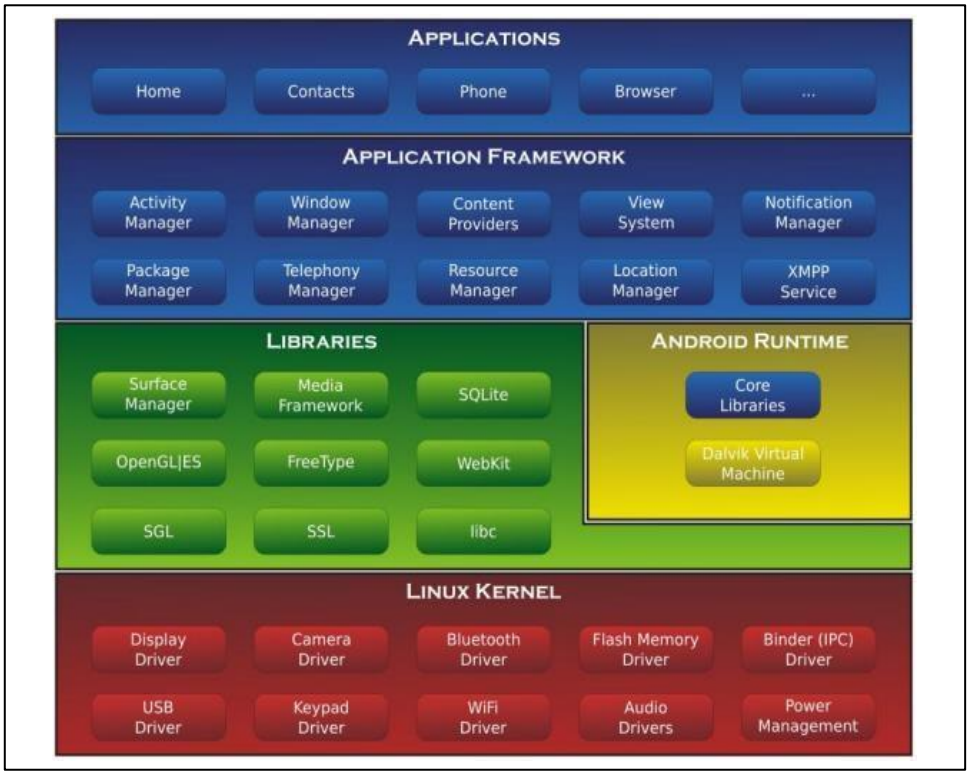


Figure 2.1: Android architecture.

Various mobile operating systems have been developed to run on mobile devices. Examples of these systems include Symbian OS by Nokia, iOS by Apple, BlackBerry OS by RIM, Windows Phone by Microsoft, and Android by Google, among others. Figure 2.2 shows the usage percentage of different mobile operating systems in 2017.

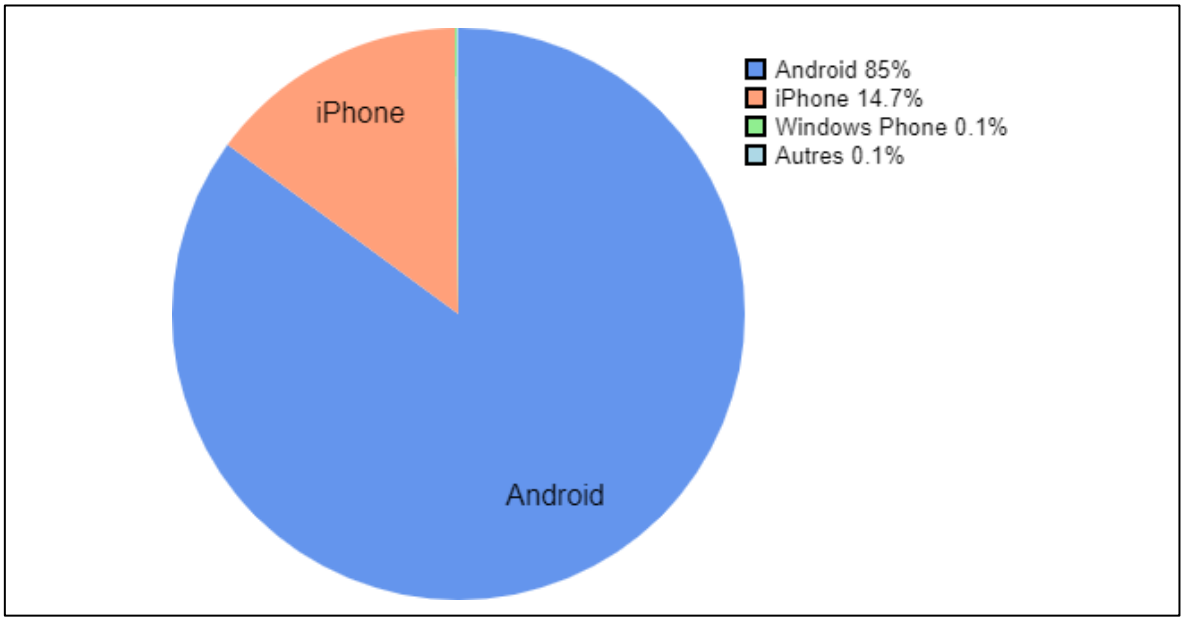


Figure 2.2 : Percentage of Mobile Operating System Usage in 2017.

A mobile application is a software program designed and developed to be installed and executed on a mobile device, such as a smartphone, tablet, or portable media player.

Android uses Google Play to allow users to access a wide range of mobile applications. Google Play enables users to download applications, movies, music, magazines, books, and more. It also allows users to rate, uninstall, and update Android applications. Figure 2.3 shows the Google Play website (https://play.google.com/store).

Some mobile applications available on Google Play are free, while others are paid.

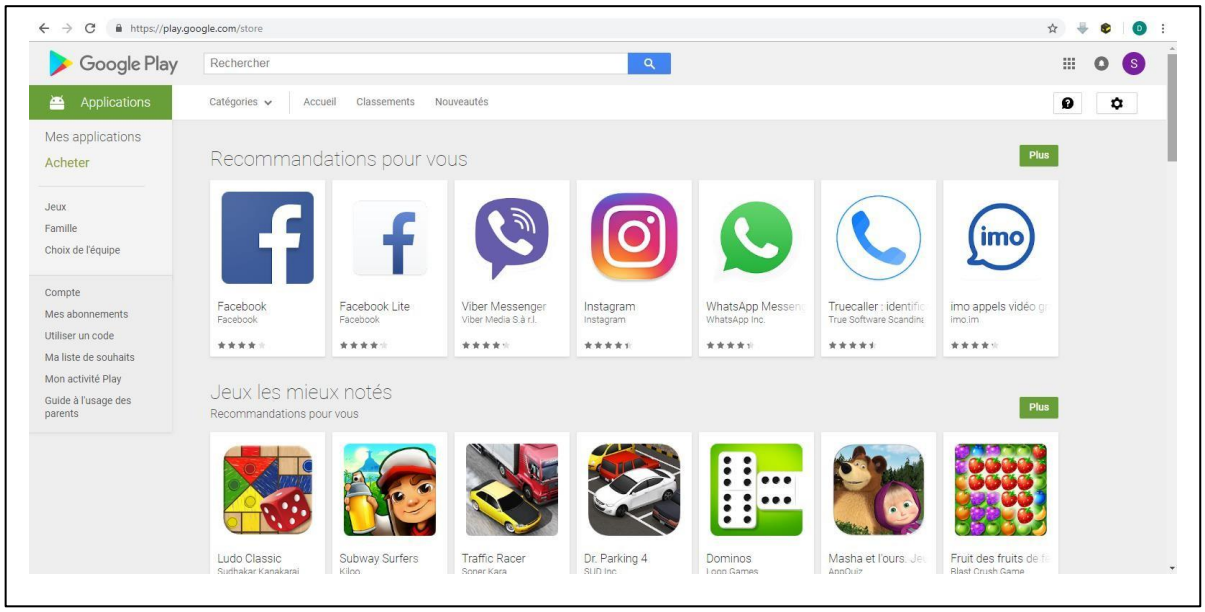


Figure 2.3 : Web site https://play.google.com/store.

We use development environments and platforms such as Java Eclipse and Android Studio to develop mobile applications on Android.

In the case of using Java Eclipse, it is necessary to integrate the SDK (Software Development Kit) with the Eclipse plugin in order to be able to develop these mobile applications.

2.3 Android SDK

The use of the Android SDK with the Eclipse plugin requires having a JDK (Java Development Kit) and a JRE (Java Runtime Environment). To download the JDK and JRE, you need to access the following web page: http://java.sun.com/javase/downloads/index.jsp. Figure 2.4 shows this web page.

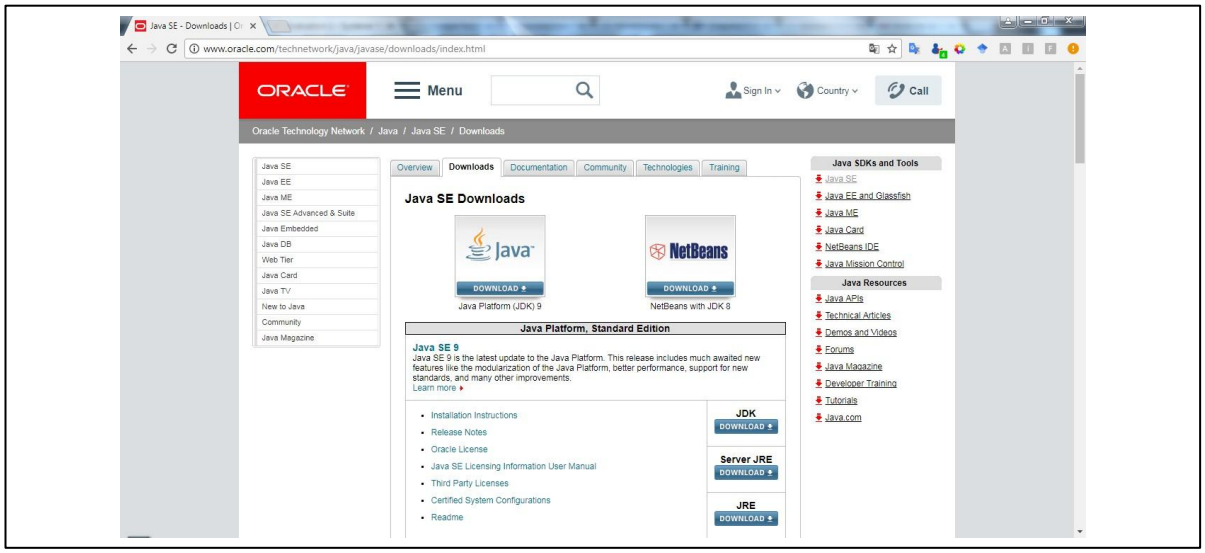


Figure 2.4: The web page http://java.sun.com/javase/downloads/index.jsp

After installing the JDK and JRE, the next step is to install and configure the SDK, which contains all the tools required for Android mobile application development. Each SDK version includes the following tools:

- aapt – Android Asset Packaging Tool: used to create and analyze .apk files that contain the mobile application program.
- adb – Android Debug Bridge: used to establish connections with a device or emulator to transfer applications or files.
- dx – Dalvik Cross-Assembler: used to merge and convert standard Java class files (.class) into binary files (.dex).
- ddms – Dalvik Debug Monitor Service: allows monitoring running threads, simulating incoming calls and messages, simulating location, etc.

In addition, the SDK also includes emulators, documentation for each version, and examples for different APIs.

2.4 Installation and configuration of tools

To download the SDK, access the following web address: https://developer.android.com/sdk/download.html

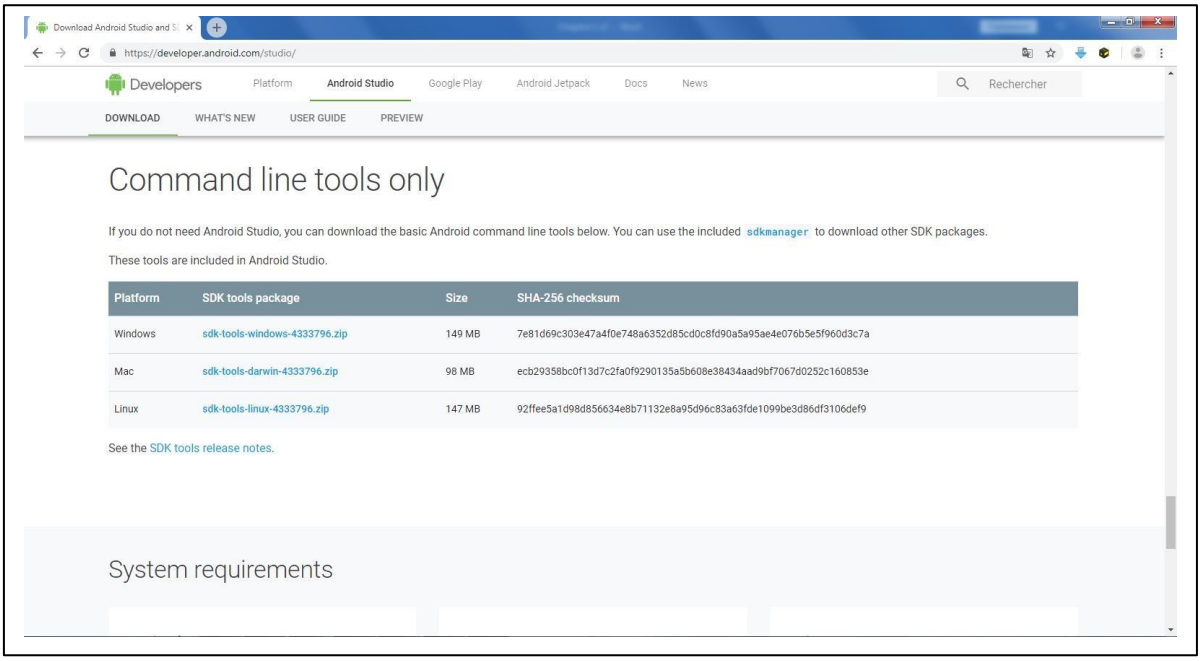


Figure 2.5: SDK download web page.

After installing the SDK, the Android SDK Manager can be used to update the SDK, install new Android versions, and update installed versions.

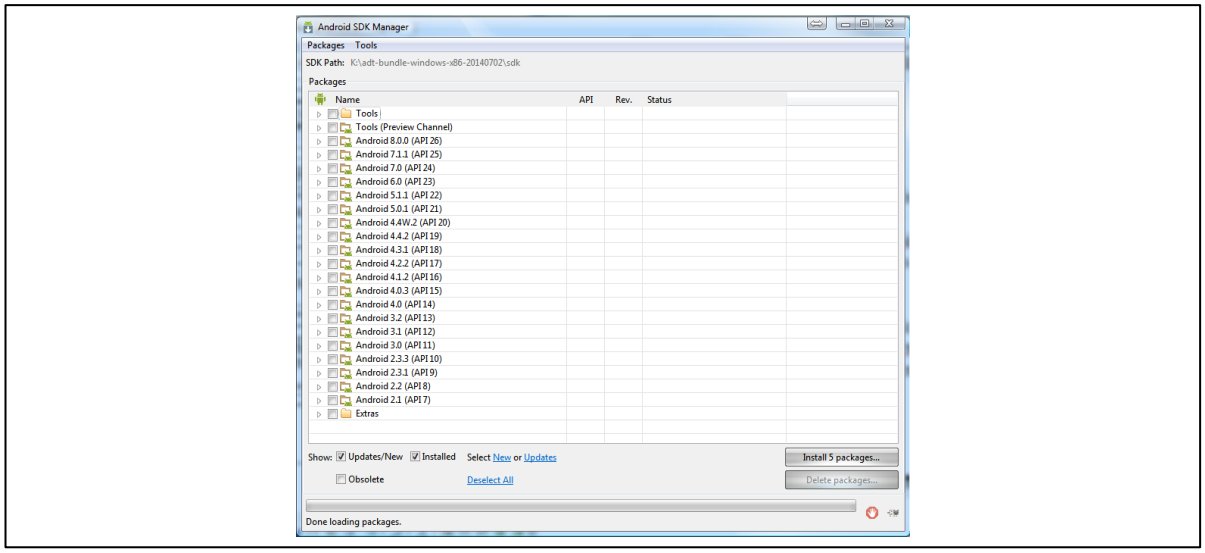


Figure 2.6: Android SDK Manager.

The next step is to download and run the Eclipse IDE (Integrated Development Environment), which simplifies application development. It can be downloaded from:
https://www.eclipse.org/downloads/

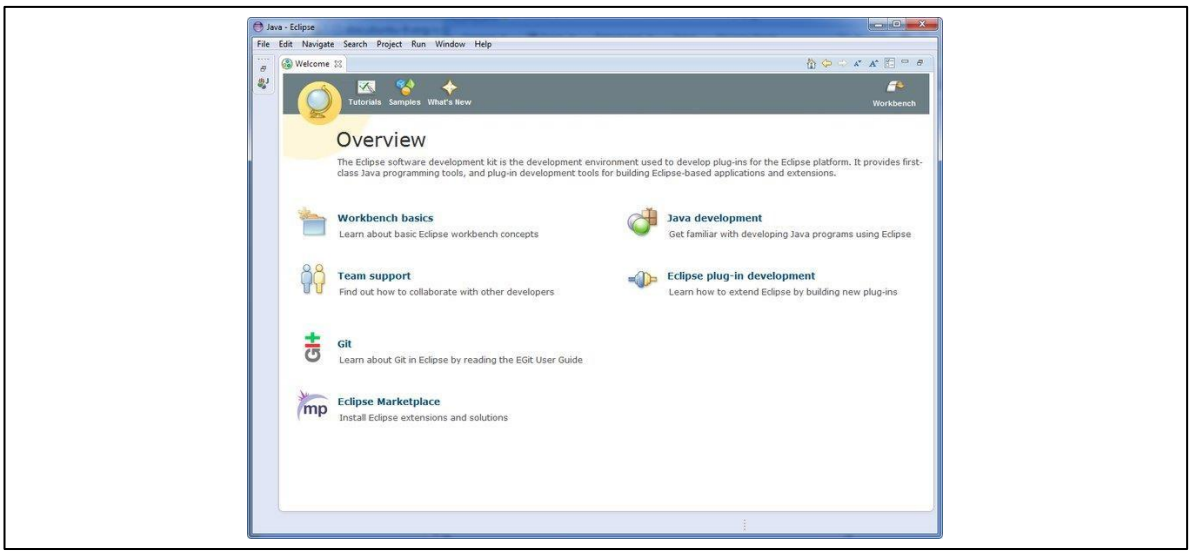


Figure 2.7: First launch of Eclipse IDE.

After launching Eclipse, the ADT plugin (Android Development Tools) must be installed to enable Android development within Eclipse. To install it:

- Click on Help → Install New Software
- Click Add
- Enter a name (e.g., “ADT Plugin”)
- Use the following location: https://dl-ssl.google.com/android/eclipse/
- Click OK

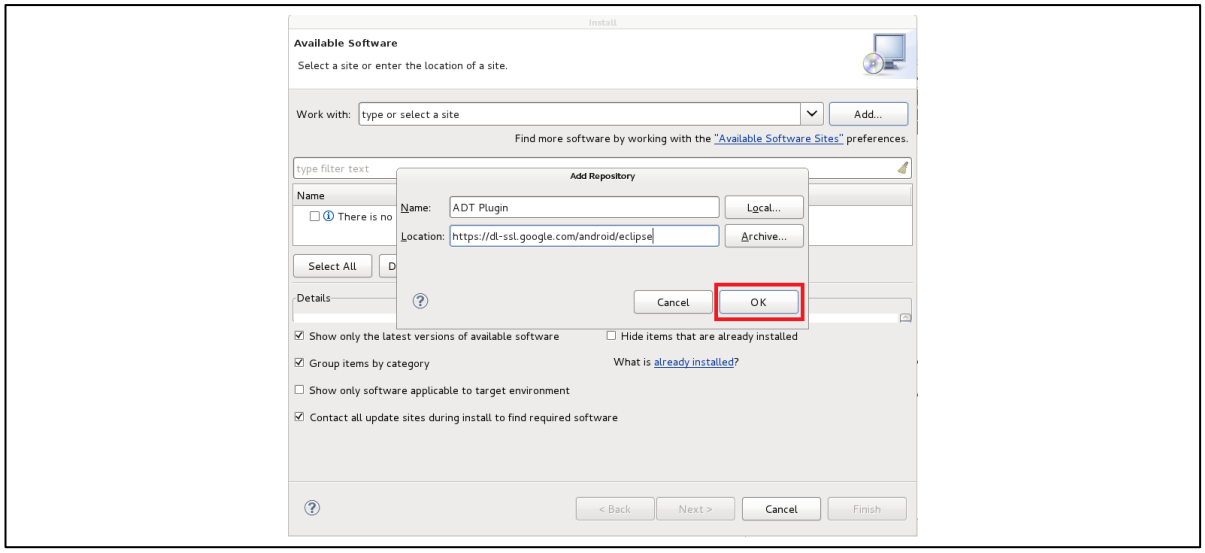


Figure 2.8: Integration of the ADT plugin into Eclipse IDE.

2.5 Creation of Android emulator

The emulator is a virtual Android device (AVD – Android Virtual Device) used to simulate a smartphone or tablet. It allows running and testing Android applications.

In Eclipse, there is a specific icon for managing emulators (Android Virtual Device Manager), which allows you to:

- View the list of emulators
- Create a new emulator
- Modify or delete an existing emulator
- View emulator details
- Launch an emulator

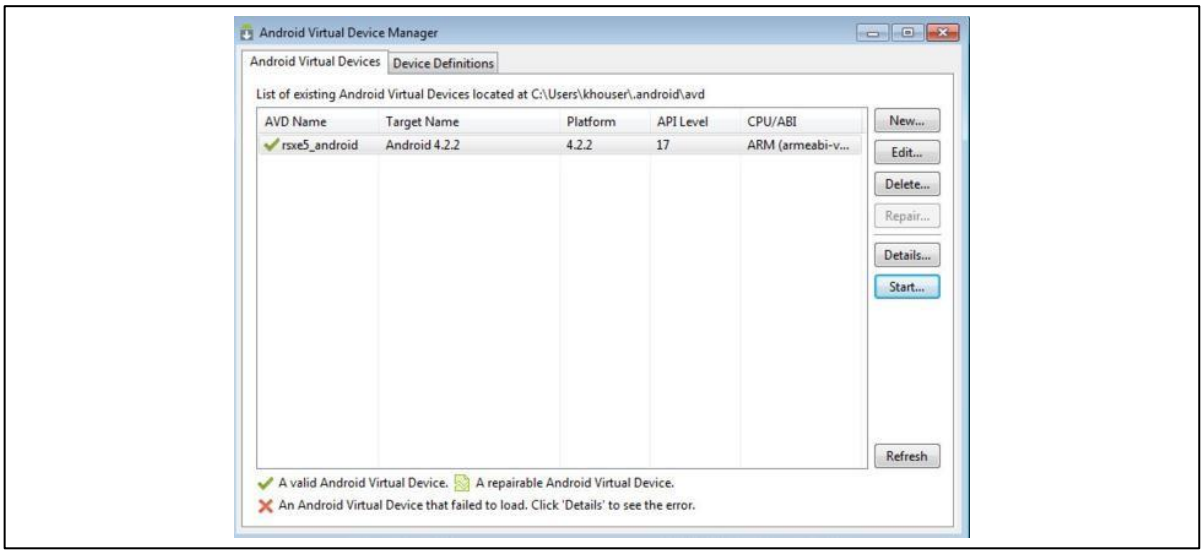


Figure 2.9: Android Virtual Device Manager.

To create a new emulator, the following information must be provided:

- Name: emulator name
- Target: Android version
- SD Card: storage size (optional)
- Snapshot: enable saving emulator states (optional)
- Skin: emulator resolution (predefined or custom)
- Hardware: additional features (GPS, accelerometer, etc.)

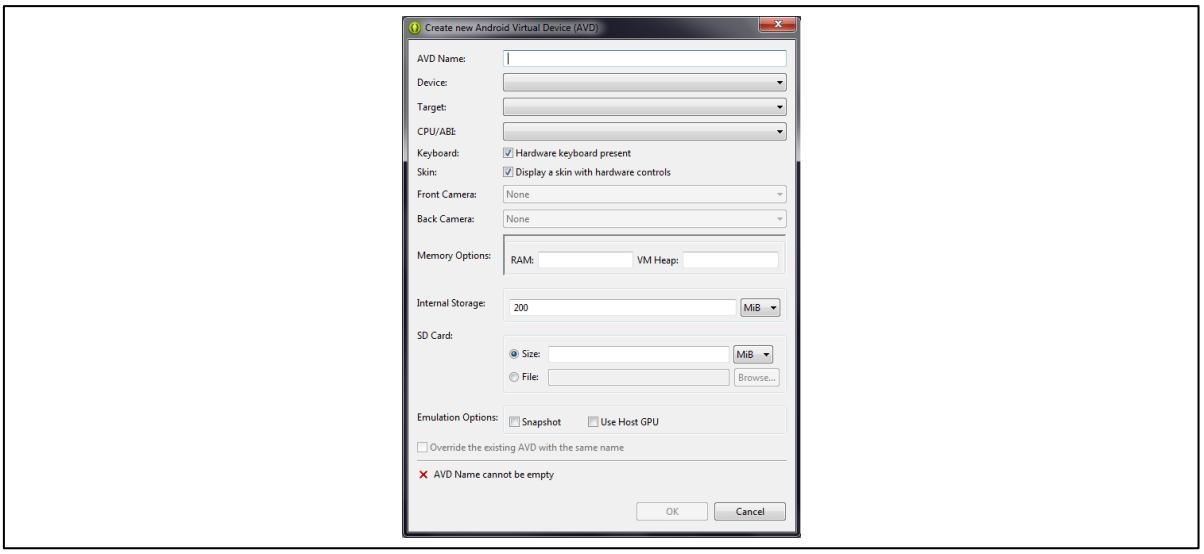


Figure 2.10: Creating a new Android emulator (AVD).

To start the emulator, select it in the AVD Manager and click Start.

Figure 2.11: Android emulator (AVD).

2.6 The first Android application

The objective of this exercise is to follow the steps below to create and run a simple Android application:

- From the menu File → New, select Android Application Project, and fill in the required information as shown in Figure 2.12.
- Click Next and keep the default settings. You can change the project location by unchecking Create Project in Workspace.
- Click Next; the next window allows you to define an application icon (default or custom).
- Click Next; the next window is used to create an activity (a screen with a graphical interface) as shown in Figure 2.13. Select Blank Activity.
- Click Next; specify the name of the activity and its layout as shown in Figure 2.14.
- Click Finish; the Android application is created.
- Run the application on the emulator (AVD) by clicking the run button as shown in Figure 2.15, and select Android Application.

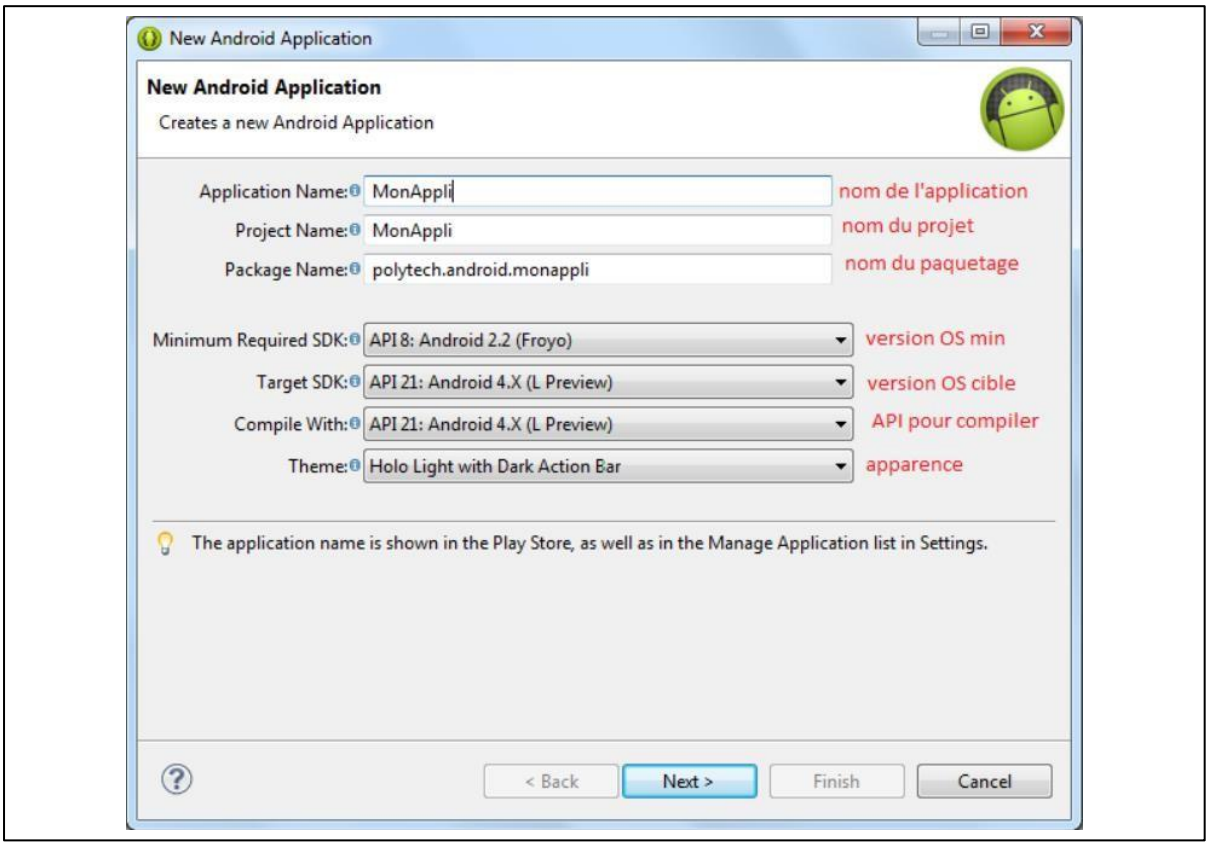


Figure 2.12: Creating a new Android application.

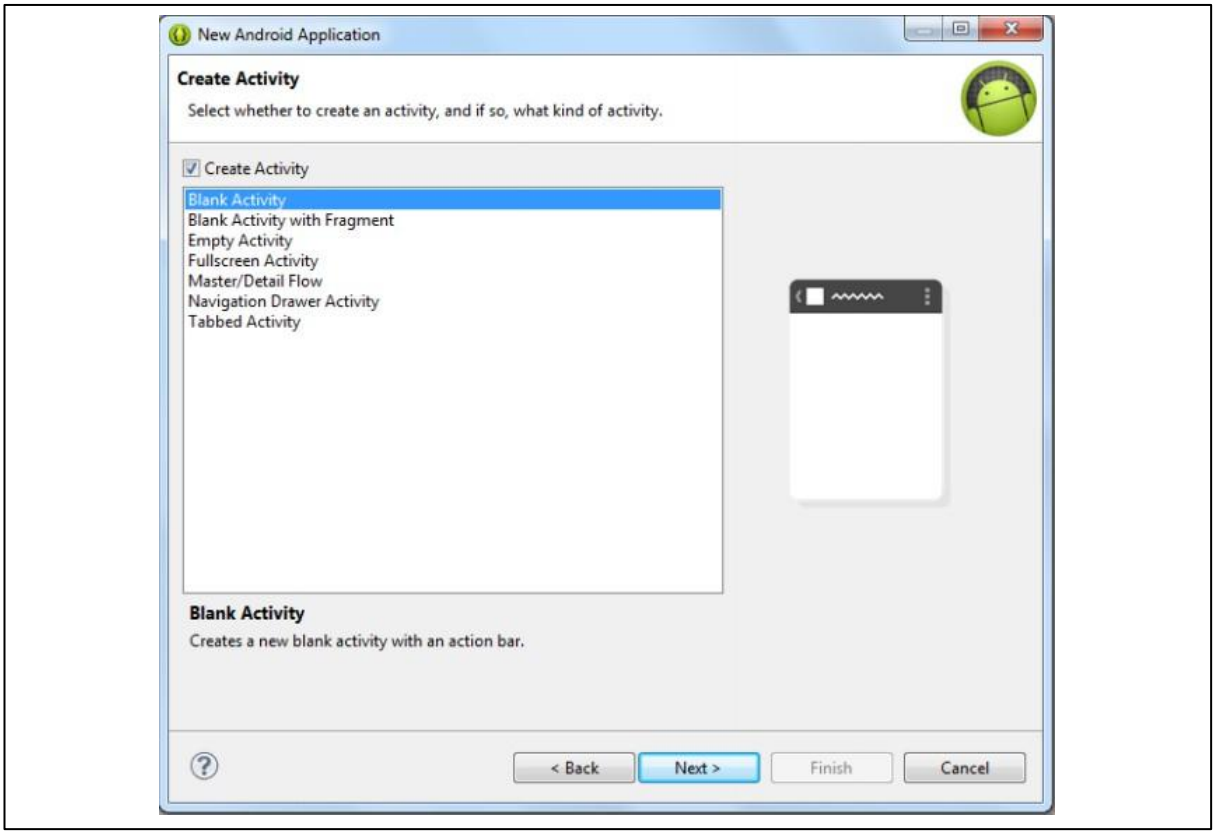


Figure 2.13: Creating an activity.

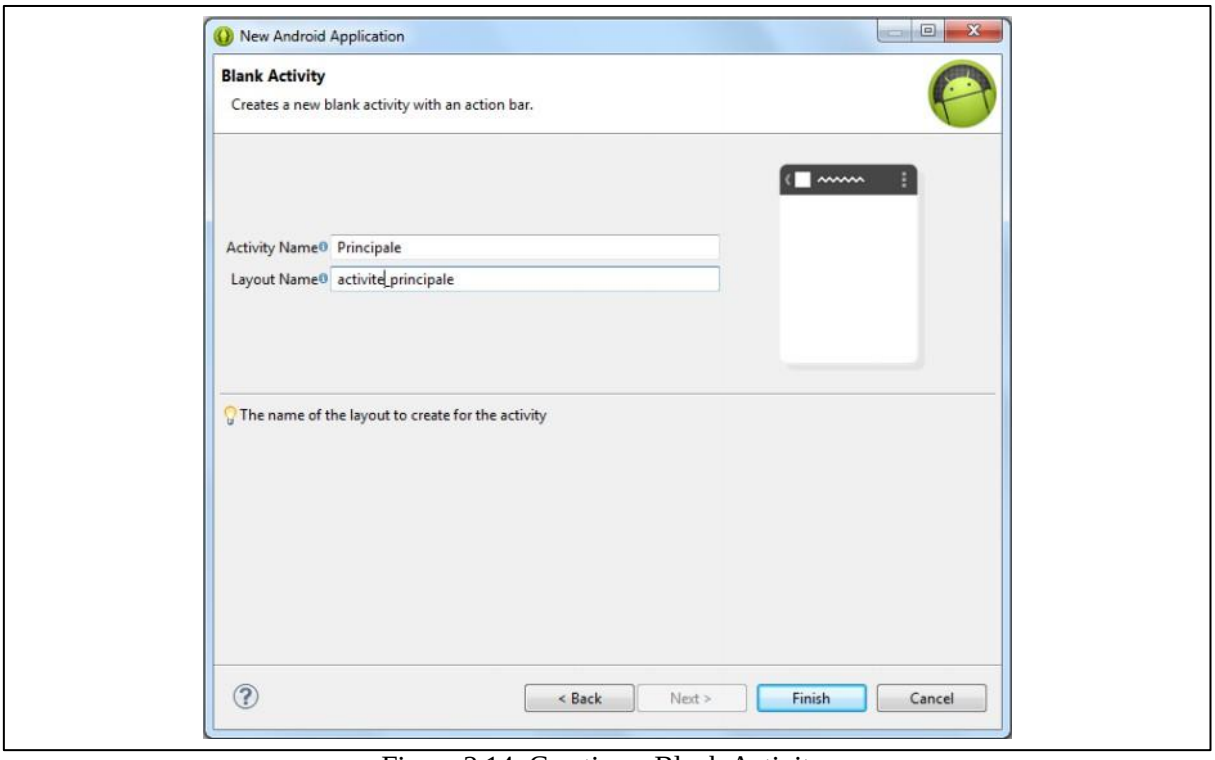


Figure 2.14: Creating a Blank Activity.

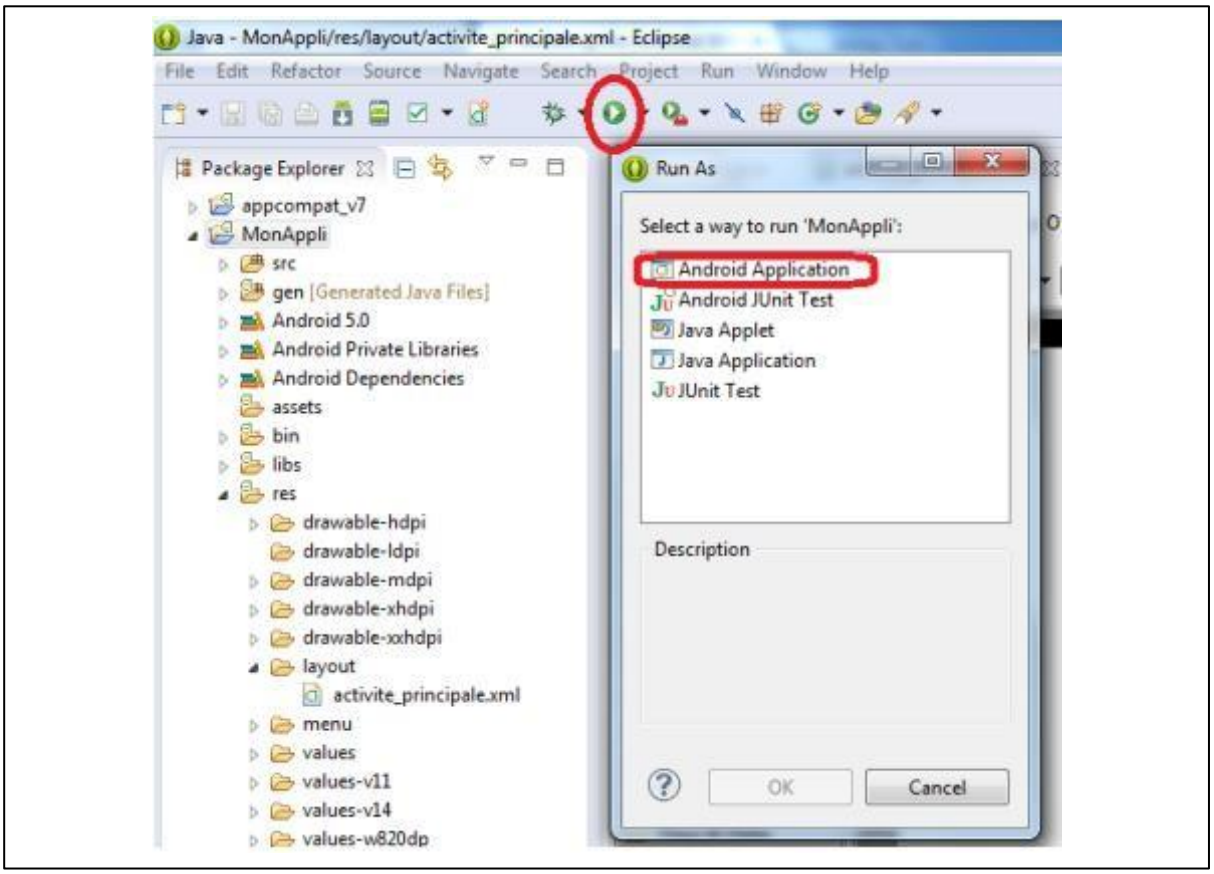


Figure 2.15: Running the Android application.

Chapter 3: Activities and resources

3.1 Introduction

An Android mobile application is composed of one or more activities. Each activity is defined by a Java class, which specifies the processing logic and user interactions for a particular screen of the application's graphical interface.

An Android mobile application may also include services, content providers, and broadcast receivers. A service represents background processing within the application. A content provider represents the database used to store the application's data. A broadcast receiver, on the other hand, is used to handle events received from outside the application.

3.2 Concept of an Activity

An activity in an application represents the processing and interactions performed on a screen (window) of the application's graphical user interface. To create a new activity in Java, the following requirements must be:

- The activity must be declared in the application manifest.
- The activity must extend the base Activity class.
- At a minimum, the onCreate() method of the Activity class must be overridden in the new class (see Figure 3.2: Activity skeleton).
- The setContentView() method must be called to link the activity with its graphical user interface.

Figure 3.1 shows an example of an activity (MainActivity) that implements the processing on a screen (window of a graphical user interface) defined in the file module.xml.

```
public class MainActivity extends Activity
{
@ Override
public void onCreate (Bundle savedInstanceState) {
super.onCreate (savedInstanceState) ;
setContentView (R.layout.module) ;
}
}
```

Figure 3.1: Example of an activity.

3.3 Activity Lifecycle

An activity can be in one of the following states (see Figure 3.2):

- **Created:** The activity is being created.
- **Started:** The activity is started (resources are allocated).
- **Resumed:** The activity is in the foreground, visible, and interacting with the user.
- **Paused:** The activity is still visible but no longer in the foreground because another activity is on top.
- **Stopped:** The activity is no longer visible and is not in the foreground.
- **Destroyed:** The activity is terminated and its resources are released.

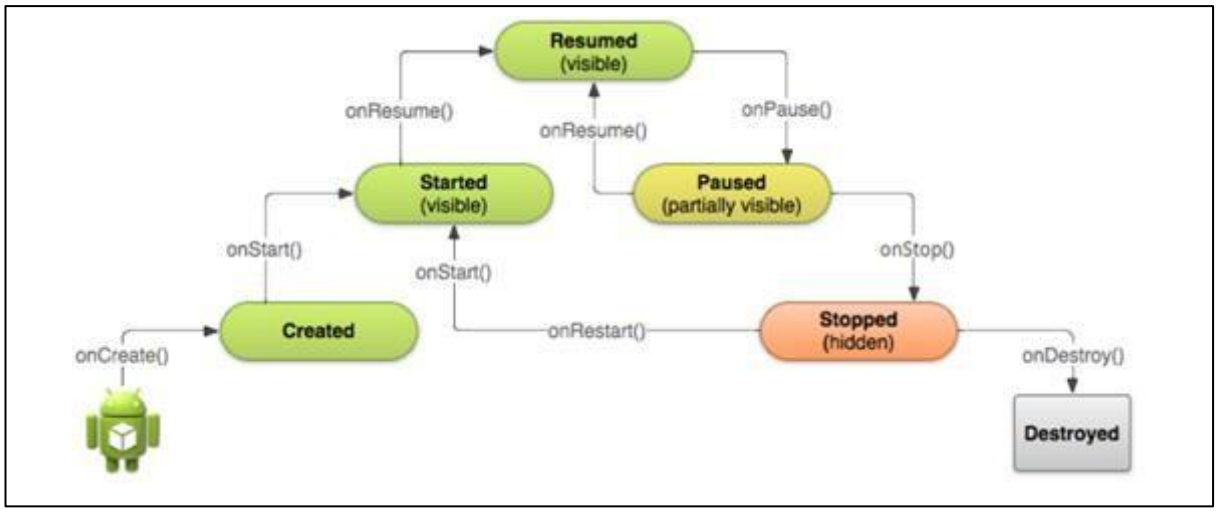


Figure 3.2 : Activity Lifecycle.

To move from one state to another, the activity executes a corresponding method (onCreate(), onStart(), etc.). Table 3.1 presents these methods along with a description and the source code of each method.

Method	Description	Redefining the method (Overloading)
onCreate()	Creation of an activity	public void onCreate (Bundle savedInstanceState) { super.onCreate (savedInstanceState) ; setContentView (R.layout.module) ; … }

onStart()	Launch the activity (allocate resources)	public void onStart(){
		super.onStart();
		…
		}

onResume()	Activate the activity (visible and in the foreground)	public void onResume(){
		super.onResume();
		…
		}

onPause()	Put the activity in the background, but it's still visible	public void onPause(){
		super.onPause();
		…
		}

onStop()	Stop the activity	public void onStop(){
		super.onStop();
		…
		}

onDestroy()	Destroy the activity (free up resources)	public void onDestroy(){
		super.onDestroy();
		…
		}

Table 3.1 : Methods for changing the states of an activity.

3.4 Resources

Resources represent the elements that make up an application, such as images, videos, text, and others. They also include XML file descriptions of application components.

3.5 Resources organization

The res directory of an Android mobile application contains all the application resources. Figure 3.3 shows the res directory of an Android mobile application.

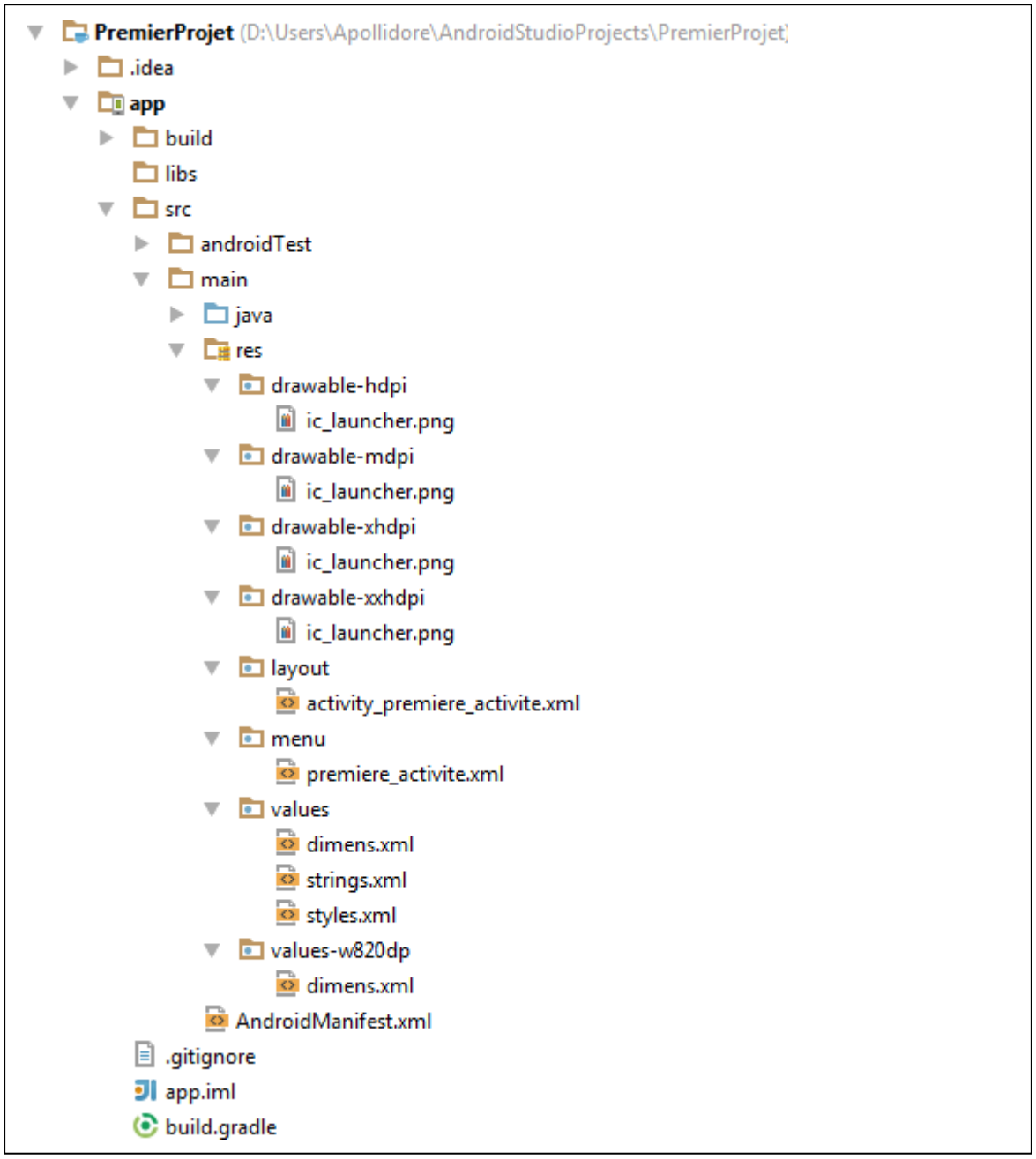


Figure 3.3 : *res* Directory of an Android mobile application.

Table 3.2 presents a description of the different types of resources in the *res* directory.

Resource	Description
drawable-hdpi	High-definition images
drawable-ldpi	Low-definition images
drawable-mdpi	Medium-definition images
layout	XML description of the interfaces
values	XML definitions of constants: strings, arrays, numeric values …
anim	XML description of animations
menus	XML description of menus for the application
xml	XML files used directly by the application
raw	All other types of resources: sounds, videos, …

Table 3.2 : Different types of resources in the *res* directory.

3.6 Utilization of resources

The res/values directory contains XML files (e.g., colors.xml, strings.xml, styles.xml). Value-type resources are declared in these files as shown in Figure 3.4. Figure 3.5 presents examples of colors.xml, strings.xml, and styles.xml files.

```
<resources>
…
<ressource_type name=" ressource_identifier"> ressource_value </ressource_type>
…
</resources>
```

Figure 3.4: Declaration of a resource in XML

colors.xml File

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

strings.xml File

```
<resources>
    <string name="app_name">My Application</string>
</resources>
```

styles.xml File

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

Figure 3.5: colors.xml, strings.xml, and styles.xml files of an Android mobile application.

Access to application resources from Java code is performed using two classes: the R class and the Resources class.

The R class is used to reference resources. It is defined in Java (file R.java) and contains inner classes. Each inner class corresponds to a specific type of resource. Usually, the name of the inner class matches the type of resource it references (e.g., id, drawable, layout, etc.). Figure 3.6 shows an example of the source code of the R class. For example, the inner class layout allows access to the XML files of the application's user interface. A resource can be referenced in an XML file using the syntax: @type/identifier. For example: @string/app_name. A resource can also be referenced in Java code using: R.type.identifier. For example: R.string.app_name.

```
public final class R {
  public static final class drawable {
  public static final int ic_action_search=0x7f020000;
  public static final int ic_launcher=0x7f020001;
  }
  public static final class id {
  public static final int menu_settings=0x7f080000;
  }
  public static final class layout {
  public static final int main=0x7f030000;
  }
  public static final class menu {
    public static final int activity_main=0x7f070000;
  }
  public static final class string {
    public static final int app_name=0x7f050000;
    public static final int hello_world=0x7f050001;
    public static final int menu_settings=0x7f050002;
    public static final int title_activity_main=0x7f050003;
  }
  public static final class style {
    public static final int AppTheme=0x7f060000;
  }
}
```

Figure 3.6: Example of the R class.

The methods of the resources class allow access to resources referenced by the R class. An instance of the Resources class must be obtained using the method getResources() in order to use its methods, such as:

- boolean getBoolean(int)
- int getInteger(int)
- int[] getArray(int)
- String getString(int)
- String[] getStringArray(int)
- int getColor(int)
- float getDimension(int)
- Drawable getDrawable(int)

The parameter of these methods is a reference to a resource. The following example shows how to access a string resource (defined in an XML file) from Java code:

```
String titre = getResources().getString(R.string.app_name);
```

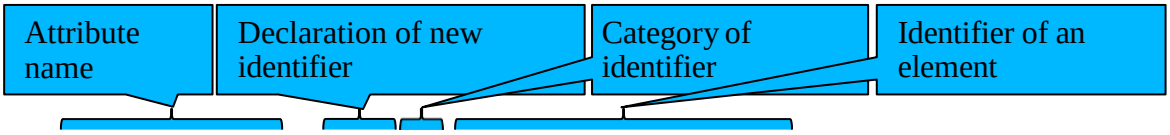
The access to the main graphical interface (layout) of the mobile application from Java code is done using the function: *setContentView(R.layout.nom_du_fichier_xml)*.

Chapter 4 : Graphic Interfaces and Widgets

4.2 Creation of Graphic Interface

An Android mobile application consists of two parts: a static part and a dynamic part. The first part (static) represents the graphical user interface (GUI) of the application, which is implemented using the XML language, while the second part (dynamic) represents the application activities, implemented using the Java language.

The graphical interface of an Android mobile application is composed of various graphical elements (e.g., button, text, input field, etc.). All graphical element classes inherit from the base class View. Each element of the GUI must have a unique identifier. Thanks to this identifier, we can access the graphical element in order to implement interactions and processing. The syntax below allows declaring a new identifier in an XML file:



android:id = @+id/name_identifier

The following syntax allows accessing the identifier of an element from a Java file: *R.id.identifier_name,* or from an XML file: *@id/ identifier_name.*

Application activities consist of processing and interactions performed on the graphical elements of the application. To create a new activity in Java, the following points must be respected:

- The activity must be declared in the application manifest.
- The activity must be a subclass of the Activity class.
- At minimum, the onCreate method of the Activity class must be overridden.
- Call the setContentView method to link the activity to the graphical interface.

For example, if an interface is created in the file module.xml, the onCreate method should contain at least the following code:

```
public void onCreate (Bundle savedInstanceState) {
super.onCreate          (savedInstanceState)          ;
setContentView          (R.layout.module)             ;
```

4.1.1 Layouts

A layout is a group of views (ViewGroup) used to size and position views (TextView, ImageView, EditText, Button, CheckBox, etc.) on the screen (GUI). There are several types of layouts, each following a specific strategy.

a) LinearLayout

This layout arranges views in a single direction: vertically (column) or horizontally (row).

- android:orientation: specifies direction ('vertical', 'horizontal')
- android:layout_width: defines width ('fill_parent', 'wrap_content')
- android:layout_height: defines height ('fill_parent', 'wrap_content')
- fill_parent: takes maximum available size
- wrap_content: takes minimum required size

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <Button   android:id="@+id/first"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:text="First
    button" />
  <Button
    android:id="@+id/second"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Second button" />
</LinearLayout>
```

Figure 4.1 : Example of an XML code for a LinearLayout layout.

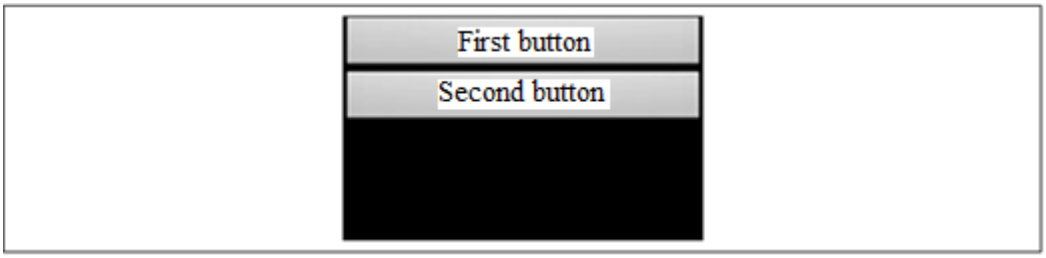


Figure 4.2: The graphical interface corresponding to the XML code in Figure 4.1.

b) RelativeLayout

Views in a RelativeLayout can be positioned relative to their parent or to other views. Examples of positioning relative to parent:

- android:layout_centerInParent="true"
- android:layout_centerHorizontal="true"
- android:layout_centerVertical="true"
- android:layout_alignParentLeft="true"
- android:layout_alignParentRight="true"
- android:layout_alignParentTop="true"
- android:layout_alignParentBottom="true"

Positioning relative to other views: android:layout_alignLeft

- android:layout_alignRight
- android:layout_alignBottom
- android:layout_alignTop

Figure 4.3 presents an example of XML code defining a RelativeLayout layout. This layout consists of four views (EditText, two Buttons, and a ProgressBar). Figure 4.4 shows the graphical interface of this XML code.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_height="match_parent"
android:layout_width="match_parent">
<EditText android:id="@+id/name"
…"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"
/>
<Button android:id="@+id/send"
android:layout_toRightOf="@+id/name"
… />
<ProgressBar android:id="@+id/wait"
android:layout_below="@+id/name"
… />
<Button android:id="@+id/cancel"
android:layout_toRightOf="@+id/ wait"
android:layout_below="@+id/name"
… />
</RelativeLayout>
```

Figure 4.3: Example of an XML code for a *RelativeLayout* layout.

Figure 4.4: The graphical interface corresponding to the XML code in Figure 4.3.

c) TableLayout

This layout arranges views in the form of a table. Figure 4.6 shows graphical interface de of this XML code.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:stretchColumns="1">
   <TableRow>
      <TextView
         android:text="@string/table_layout_4_open"
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_open_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>

   <TableRow>
      <TextView
         android:text="@string/table_layout_4_save"
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_save_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>
</TableLayout>
```

Figure 4.5: Example of an XML code for a *TableLayout* layout.

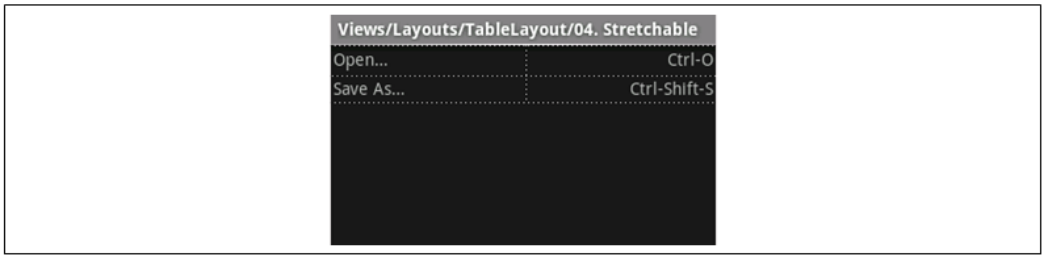
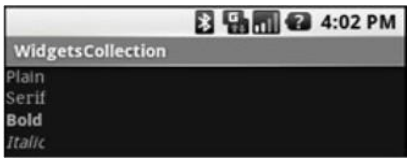


Figure 4.6: The graphical interface corresponding to the XML code in Figure 4.5.

4.1.2 Interface Components

a) TextView

Displays simple text. Figure 4.7 presents an example of four *TextViews*.




```
<?xml version= "1.0" encoding= "utf-8" ?>
<LinearLayout
xmlns:android= "http://schemas.android.com/apk/res/android"  android:orientation= "vertical"  an
droid:layout_width= "fill_parent" android:layout_height= "fill_parent" >
<TextView android:id="@+id/text1"
android:text= "Plain" android:layout_width= "wrap_content" android:layout_height= "wrap_conte
nt"  />
<TextView
android:text= "Serif" android:layout_width= "wrap_content" android:layout_height= "wrap_conte
nt" android:typeface= "serif" />
<TextView
android:text= "Bold" android:layout_width= "wrap_content" android:layout_height= "wrap_conte
nt" android:textStyle= "bold" />
<TextView
android:text= "Italic" android:layout_width= "wrap_content" android:layout_height= "wrap_conte
nt" android:textStyle= "italic"/> </LinearLayout>
```

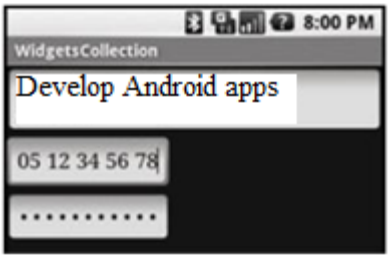
Figure 4.7 : Example of 4 *TextViews*.

The content of a TextView can be changed in Java code as follows:

```
TextView tx = (TextView) findViewById(R.id.text1);

tx.setText("bbbbbb");
```

b) EditText

Allows text input. Figure 4.8 presents an example of 3 *EditTexts*. The attribute *android:inputType* specifies the type (address, phone, etc.).

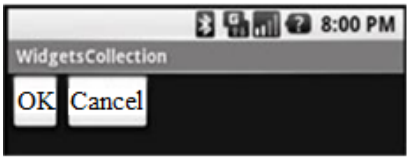



```
<?xml version= "1.0" encoding= "utf-8" ?> <LinearLayout
xmlns:android= "http://schemas.android.com/apk/res/android" android:orientation= "vertical"
andr oid:layout_width= "fill_parent" android:layout_height= "fill_parent" >

<EditText android:id= "@+id/title" android:text= "Develop Android apps"
android:layout_width= "wrap_content" android:layout_height= "wrap_content" />

<EditText android:id= "@+id/phoneNumber" android:text= "05 12 34 56
78" android:layout_width= "wrap_content" android:layout_height= "wrap_content"
android:typefa ce= "serif" android:phoneNumber= "true" />

<EditText
android:id= "@+id/password" android:text= "monPassword" android:layout_width=
"wrap_conten t" android:layout_height= "wrap_content" android:password= "true" />
</LinearLayout>
```

Figure 4.8 : Example of 3 *EditTexts*.

c) Button

Represents a button that the user can click to trigger an action. Figure 4.9 presents an example of 2 *Buttons.*

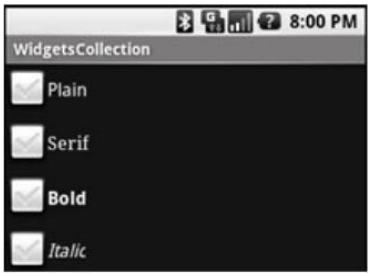



```
<?xml version= "1.0" encoding= "utf-8" ?> <LinearLayout
xmlns:android= "http://schemas.android.com/apk/res/android" android:orientation= "horizontal"
an droid:layout_width= "fill_parent" android:layout_height= "fill_parent" >

<Button
android:id= "@+id/okButton" android:text= "OK" android:layout_width= "wrap_content"
android: layout_height= "wrap_content" />

 <Button
android:id= "@+id/CancelButton" android:text= "Cancel" android:layout_width= "wrap_content"
 android:layout_height= "wrap_content" /> </LinearLayout>
```

Figure 4.9: Example of 2 *Buttons*.

d) CheckBox

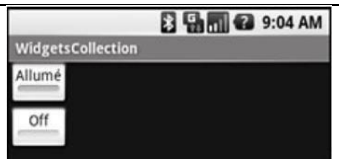
Represents a selectable checkbox. Figure 4.10 presents an example of 4 *CheckBoxs*.

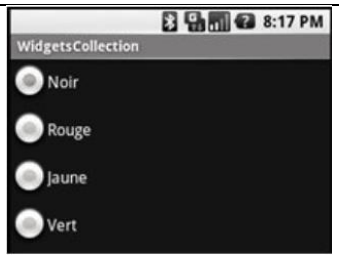
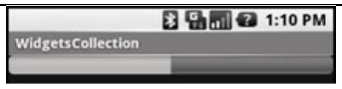



```
<?xml version= "1.0" encoding= "utf-8" ?> <LinearLayout
xmlns:android= "http://schemas.android.com/apk/res/android" android:orientation= "vertical" andr
oid:layout_width= "fill_parent" android:layout_height= "fill_parent" >
<CheckBox
android:id= "@+id/plain_cb" android:text= "Plain" android:layout_width= "wrap_content" android
:layout_height= "wrap_content" />
<CheckBox
android:id= "@+id/serif_cb" android:text= "Serif" android:layout_width= "wrap_content" android:
layout_height= "wrap_content" android:typeface= "serif" />
<CheckBox
android:id= "@+id/bold_cb" android:text= "Bold" android:layout_width= "wrap_content" android:
layout_height= "wrap_content" android:textStyle= "bold" />
<CheckBox
android:id= "@+id/italic_cb" android:text= "Italic" android:layout_width= "wrap_content" android
:layout_height= "wrap_content" android:textStyle= "italic" /> </LinearLayout>
```

Figure 4.10: Example of 4 *CheckBoxs*.

Table 4.2 presents other types of graphical components, with a portion of XML code used to create it.

Composant graphique	**L'interface graphique**	**Portion du code XML**
ToggleButton		<ToggleButton android:id="@+id/toggle1" android:layout_width= "wrap_content" android:layout_height= "wrap_content" android:textOn= "Allumé" android:textOff= "Eteint" />

RadioGroup		<RadioGroup <RadioButton android:id="@id/option1" android:layout_width= "wrap_content" android:layout_height= "wrap_content" android:text= "Noir" android:checked="false" android:layout_gravity= "left" > </RadioButton>
		…. </RadioGroup>
ImageView		<ImageView android:id="@+id/image" android:src="@drawable/logo" android:layout_width= "wrap_content" android:layout_height= "wrap_content" android:layout_gravity= "center" />
ImageButton		<ImageButton android:id= "@+id/image" android:src="@drawable/logo" android:layout_width= "wrap_content" android:layout_height= "wrap_content" android:layout_gravity= "center" />
ProgressBar		<ProgressBar android:id= "@+id/progress" style= "?android:attr/progressBarStyleHorizontal" android:layout_width="fill_parent" android:layout_height= "wrap_content" />
RatingBar		<RatingBar android:id= "@+id/gallery" android:layout_width="wrap_content" android:layout_height= "wrap_content" />
Spinner		<Spinner android:id="@+id/saisons" android:layout_width= "wrap_content" android:prompt="@string/prompt" android:layout_height= "wrap_content" android:entries= "@array/saisons" />

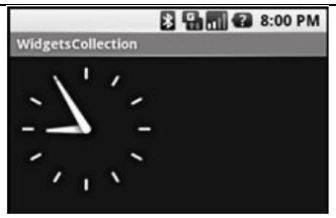
DatePicker		<DatePicker android:layout_width="wrap_content" android:layout_height= "wrap_content" />
TimePicker		<TimePicker android:id= "@+id/time" android:layout_width="wrap_content" android:layout_height= "wrap_content" />
AnalogClock		<AnalogClock android:layout_width="wrap_content" android:layout_height= "wrap_content" />
DigitalClock		<DigitalClock android:layout_width="wrap_content" android:layout_height= "wrap_content" />

Table 4.2: Different types of graphical components.

4.1.1 Styles

A style allows modifying the appearance properties of a view, such as font, text size, color, images, spacing, etc.

To define a new style, it was necessary to add a style resource in the styles.xml file located in the res/values directory. For example, the XML code below allows the creation of two style resources named "MyTheme" and "MySubTheme".

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="MyTheme">
    <item name="android:textSize">12sp</item>
    <item name="android:textColor">#111</item>
  </style>
  <style name="MySubTheme" parent="MyTheme"; >
    <item name="android:textSize">8sp</item>
  </style>
</resources>
```

The parent attribute allows the "MySubTheme" style to inherit the properties of the

"MyTheme" style. The use of the "MySubTheme" style for a view of type TextView in the XML layout file (layout.xml) is done as follows:

```
<TextView
  style="@style/MySubTheme"
  android:text="@string/hello" />
```

A theme is a style applied throughout the entire application. The theme must be specified in the application manifest (AndroidManifest.xml), as shown in the following example:

```
<application
android:theme="@style/MyTheme"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name" … /application>
```

4.2 Manage events on widgets

A view represents a graphical user interface element (Button, text field, etc.). An activity can access a layout (a group of views) or an individual view using the following methods:

- setContentView: to load a layout onto the screen.
- findViewById: to access a view (graphical component).La figure 4.10 présente un exemple d'accès aux vues (*EditText* et *TextView*) à partir de l'activité (*SecondActivity*).

Figure 4.11 shows an example of accessing views (EditText and TextView) from an activity (SecondActivity).

We can define actions and listeners on a view. For example, when the view is a button, you can define the method that should be triggered when the button is clicked in two ways:

- **First method: Define an action**
  - Specify the method name in the android:onClick attribute when creating the button in the XML layout file. For example:

    ```
    <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick=" nomFonction" />
    ```
  - Define this method in the corresponding activity (Java file). The general form of this method is:

```
public void nomFonction(View view){
//code de la fonction ici}
```

- **Second method: Define a listener**
  - Create a listener in the activity as an instance of the OnClickListener class.
  - Override the onClick(View v) method of the OnClickListener class.
  - Use the button's setOnClickListener method directly in the activity, where the created listener is passed as a parameter.
  - If the same listener is used for multiple views, use the getId() method of the view to distinguish between them.

Figure 4.12 shows an example of creating a listener for two buttons.

```
Button bt1= (Button) findViewById(R.id.button1) ;
Button bt2= (Button) findViewById(R.id.button2) ;
View.OnClickListener listener = new View.OnClickListener(){
public void onClick(View view){
switch (view.getId()){
case R.id.button1 :
… // Code for the onClick method of button1
Break;
case R.id.button2 :
… // Code for the onClick method of button2
Break;
}
}
}
bt1.setOnClickListener(listener);
bt2.setOnClickListener(listener);
```

Figure 4.12: Example of creating a listener for two buttons.

Listeners can also be defined for other views (such as Spinner, RatingBar, etc.). Similar to the case of a button (overriding onClick), the corresponding methods are overridden, such as onItemSelected when selecting an item in a Spinner, or onRatingBarChange when changing

the rating in a RatingBar, etc.

4.3 2D Drawing

To create an Android application with 2D graphics, it is necessary to define a subclass (named View2D, for example) that extends the View class and overrides its onDraw() method. The 2D rendering of a view is performed through calls to this method.

To invoke the onDraw() method from an activity, a new view of type View2D must be declared in the activity's layout. Figure 4.13 shows an example of the definition of the View2D class in Java, while Figure 4.14 presents an example of the declaration of a View2D view in XML (the activity layout).

```
package fr.iutlan.dessin;
public class View2D extends View {
public View2D(Context context){
super(context) ;
}
public void onDraw(Canvas canvas) {
super.onDraw(canvas)
Rect Rectangle = new Rect();
Rectangle.set(0, 0, 100, 50);
Paint Peinture = new Paint();
Peinture.setColor(Color.BLUE);
Peinture.setStyle(Paint.Style.FILL);
canvas.drawRect(Rectangle, Peinture);
}
}
```

Figure 4.13: Example of a View2D Class Definition in Java.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<fr.iutlan.dessin.View2D
android:id="@+id/drawing2d"
android:layout_width="match_parent"
android:layout_height="match_parent" />
</LinearLayout>
```

Figure 4.14: Example of a View2D View Definition in XML within the Activity Layout.

As shown in Figure 4.13, the drawRect() method of the Canvas class is used to draw a rectangle, while the setColor() and setStyle() methods of the Paint class are used to specify

the rectangle's color and style.

Table 4.3 lists the different classes and their methods commonly used for creating 2D graphics in an Android mobile application.

Class	Method	Description
Canvas	void drawCircle (float cx, float cy, float radius, Paint paint)	Draw a circle using a Paint object
	void drawRect (Rect r, Paint paint)	Draw a rectangle using a Paint object
	void drawText (String text, float x, float y, Paint paint)	Draw text using a Paint object
Paint	void setColor (int color)	Set the paint color
	void setStyle (Paint.Style style)	Set the paint style
	void setTextSize(float textSize)	Set the paint text size
Rect	Rect(int left, int top, int right, int bottom)	Create a new rectangle with specific coordinates
	boolean equals(Object o)	Check whether another Rect object is equal to this Rect object
	boolean contains(int x, int y)	Return true if (x, y) is inside the rectangle

Table 4.3: Some Classes and Methods Used for 2D Drawing in Android

The user can interact with the application's 2D drawing, for example by touching the drawing area, performing gestures on the 2D drawing, and so on. Android provides the onTouchEvent() method of the View class to handle user input events.

4.4 List Application

This section introduces the concept of an item list (ListView) in Android mobile applications. A ListView is a view that displays a collection of items, where each row in the list represents a single item (ListItem). A ListView can be scrolled vertically. Figure 4.15 shows an example of a ListView.

Figure 4.15: Example of a ListView (Item List).

The data displayed in a ListView (the items) must be stored in a data structure (such as an array, matrix, etc.) or in a database. In this chapter, we use a dynamic array (ArrayList) to store the data for the ListView.

In an Android mobile application, the intermediary between a ListView, which displays the data, and an ArrayList, which stores the data, is the Adapter. Figure 4.16 illustrates the general relationship between a ListView, an Adapter, and an ArrayList.

As shown in the figure, the Adapter retrieves the item data stored in the ArrayList and displays it in the ListView.

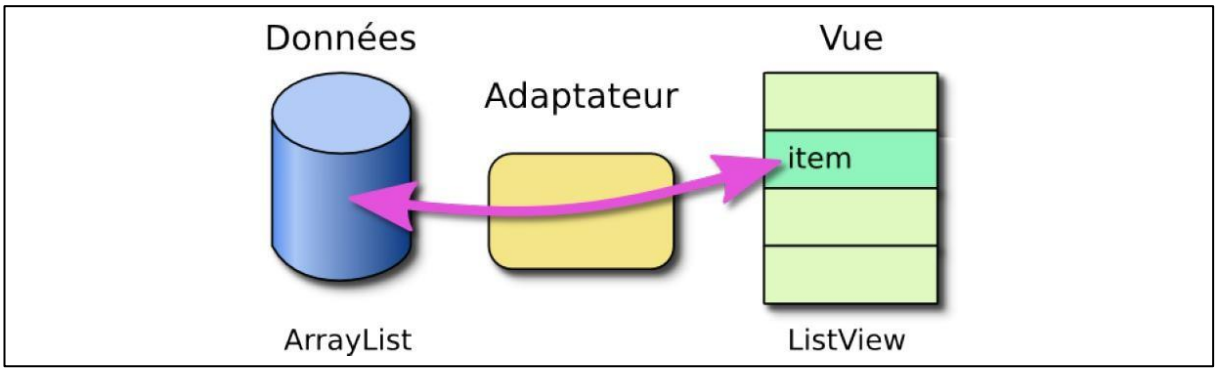


Figure 4.16: General Diagram of the Relationship between a ListView, an Adapter, and an ArrayList.

To display a set of item data in a ListView, the following steps must be followed:

- Store the item data in an ArrayList.
- Create an Adapter and a ListView.

- Associate the Adapter with the ArrayList.
- Associate the Adapter with the ListView.
- Display the item data in the ListView.

There are two ways to store item data in an ArrayList. The first approach consists of creating a Java array and then copying its contents into an ArrayList. The second approach consists of defining arrays in the res/values/arrays.xml file and retrieving these data in the Java code (through resource access) to populate an ArrayList.

Figure 4.17 presents an example of the first data storage approach, while Figure 4.18 presents an example of the second approach.

Creating a class that represents an item.

```
public class Module {
public String name;
public int coff;
public int credit;
….
};
```

Creating a Java array that contains objects of the Module class.

```
final Modules[] datainit = {
new Module (“Mobile applications”, 3, 5) ,
new Module (“Security”, 3, 5) ,
                                new Module (“Artificial Intelligence”, 3, 5) ,
…
} ;
```

Copy the Java array into an ArrayList.

```
Protected ArrayList<Module> List ;
void oncreate (…)
{
…
List = new ArrayList<Module>();
for (Module module: datainit){
List.add(module);
}
}
```

Figure 4.17: Storing ListView data in an ArrayList from a Java array.

Creating a resource consisting of three arrays in the res/values/arrays.xml file.

```
<resources>
<string-array name=”names”>
<item> Mobile applications </item>
<item> Security </item>
<item> Artificial Intelligence </item>
…
</string-array>
<integer-array name=”coffs”>
<item>3</item>
<item>3</item>
<item>3</item>
…
<\integer-array>
<integer-array name=”credits”>
<item>6</item>
<item>6</item>
<item>6</item>
…
<\integer-array>
</resources >
```

Accessing resources in Java code to retrieve the three arrays.

```
Resources res = getResources() ;
final String[] names=
res.getStringArray(R.array.names); final int[] coffs=
res.getIntegerArray(R.array.coffs); final int[] credits=
res.getIntegerArray(R.array.credits);
```

Copy the three arrays into an ArrayList.

```
Protected ArrayList<Module> List;
void oncreate (…)
{
…
List = new ArrayList<Module>(); for
(int i=0; i<noms.length; ++i){
List.add(new Module(noms[i], coffs[i] , credits[i]));
}
}
```

Figure 4.18: Storing ListView data in an ArrayList from XML arrays.

4.4.1 Displaying the List

To display item data in a ListView, it is necessary to create the graphical user interface (layout) for the list and the application activity that contains it. The <ListView> view is used in the XML layout file to create a list of items. This view can then be accessed from the application's activity. Figure 4.19 shows an example of creating a ListView in the application's layout and accessing this ListView from the application's activity.

XML layout file of the activity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >

   <ListView
      android:id="@+id/modulesList"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
    />

</LinearLayout>
```

Java file of the activity.

```
public class MainActivity extends Activity
{
private ListView myModulesList;

@ Override
public void onCreate (Bundle savedInstanceState) {
super.onCreate (savedInstanceState) ;
setContentView (R.layout.module) ;
myModulesList = (ListView) findViewById(R.id. modulesList);
}
}
```

Figure 4.19 : Example of creating a ListView and accessing it.

The graphical user interface (layout) of a ListView item can be defined in the res/layout/item_module.xml file. Figure 4.20 presents an example of the XML code used to define the graphical interface of an item.


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content" >

   <ImageView
              android:id="@+id/img"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_gravity="center_vertical"
              android:padding="10px" />

            <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
               android:orientation="vertical"
               android:layout_width="fill_parent"
               android:layout_height="wrap_content"
               android:layout_gravity="center_vertical"
               android:paddingLeft="10px"
               android:layout_weight="1  >

               <TextView android:id="@+id/title"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:textSize="16px"
                android:textStyle="bold" />

               <TextView android:id="@+id/description"
                 android:layout_width="fill_parent"
                 android:layout_height="fill_parent" />
   </LinearLayout>
</LinearLayout>
```

Figure 4.20: Example of the graphical user interface of a ListView item.

4.4.2 Adapter

The Adapter provides access to the data of ListView items stored in a data structure (ArrayList) in order to display them in the list. The adapter also serves as a link between the

layout of an item and the layout of the ListView.

Android provides several predefined adapters, such as ArrayAdapter, which can be used to retrieve item data stored in an ArrayList and display it in a ListView.

The constructor signature of the ArrayAdapter class is as follows: *ArrayAdapter(Context context, int item_layout_id, int textview_id, List<T> data)*, where:

- context represents the activity that creates the adapter.
- item_layout_id represents the identifier of the item layout.
- textview_id represents the identifier of the TextView within that layout.
- data represents the list containing the data (an instance of ArrayList).

The ArrayAdapter can display only one TextView per item. To associate an adapter with a ListView, the setAdapter(adapter) method is used, where adapter represents an adapter instance (an object of the ArrayAdapter class). Figure 4.21 presents an example of creating an ArrayAdapter and associating a ListView with this adapter.

```
ArrayAdapter<Module> adapter = new ArrayAdapter<Module>(this,
android.R.layout.item_module, android.R.id.nom_module, List) ;
ListView lm = (ListeView) findViewById(R.id.modulesList);
Lm.setAdapter(adapter);
```

Figure 4.21: Creating an ArrayAdapter and Associating It with a ListView.

ArrayAdapter can display only a single TextView in each ListView item. To display multiple views for each item (TextView, ImageView, Button, etc.), it is necessary to create a custom adapter.

In addition, when using ArrayAdapter, the item layout must contain at least one TextView. The Module class must also provide a toString() method to return the module name. With ArrayAdapter, it is possible to use either a standard Android item layout (such as R.layout.simple_list_item_1) or a custom item layout.

Since ArrayAdapter displays only one TextView, a custom adapter can be created to display a custom item layout containing multiple views. To do so, the following steps should be followed:

- Create a custom item layout (such as the example shown in Figure 4.20).
- Create a layout for the item list (see Figure 4.19).

- Define a custom adapter class that extends ArrayAdapter<T> (or BaseAdapter). This new class must include a constructor and override the methods required for item management, such as getCount(), getItem(), getView(), etc. Figure 4.22 presents an example of a custom adapter class.
- In the activity that manages the item list, create an instance of the custom adapter class. Figure 4.23 presents an example of creating a custom adapter of type ModuleAdapter and associating a ListView with this adapter.

One of the most commonly used methods in a custom adapter is the getView(...) method. This method is defined in the ArrayAdapter class and provides access to the item layout in order to create the views displayed in the ListView. The getView() method returns the layout corresponding to a ListView item. Its signature is: public View getView(int position, View convertView, ViewGroup parent), where:

- position is the position of the item in the ListView.
- convertView is an old, invisible view from the ListView that can be reused and replaced by the current view in order to reduce memory allocation and improve performance.

- parent represents the ListView to which the returned view will be attached.

```
public class ModuleAdapter extends ArrayAdapter <Module> {
private ArrayList<Module> List;
private Context context;
private int res;

public ModuleAdapter(Context context, int resource, ArrayList<Module> objects) {
super(context, resource, objects);
List = objects;
res = resource;
this.context = context;
}
@Override
public int getCount() {
if (data != null)
return Liste.size;
return 0; }

@Override
public Object getItem(int position){
return Liste.get(position);
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
LayoutInflater inflater = getLayoutInflater();
View rowView = inflater.inflate(R.layout.item_module, parent, false);
TextView textView1 = (TextView) rowView.findViewById(R.id.title);
TextView textView2 = (TextView) rowView.findViewById(R.id.description);
ImageView imageView = (ImageView) rowView.findViewById(R.id.img);

Module temp= Liste.get(position);
textView1.setText(temp.name);
textView2.setText(temp.coff);
imageView.setImageResource (R.drawable.module);
return rowView;
}
}
```

Figure 4.22: Definition of a Custom Adapter Class.

```
ModuleAdapter adapter = new ModuleAdapter (this, android.R.layout.item_module, List) ;
ListView lm = (ListeView) findViewById(R.id.modulesList);
Lm.setAdapter(adapter);
```

Figure 4.23: Creating a Custom Adapter and Associating It with a ListView.

Chapter 5 : Menus and Dialog boxes

5.1 Application Menus management

An action bar is located at the top of an activity's graphical interface. It allows displaying the activity title, application icons, some menu items, a button leading to another menu, and other elements. Figure 5.1 shows an example of an action bar.

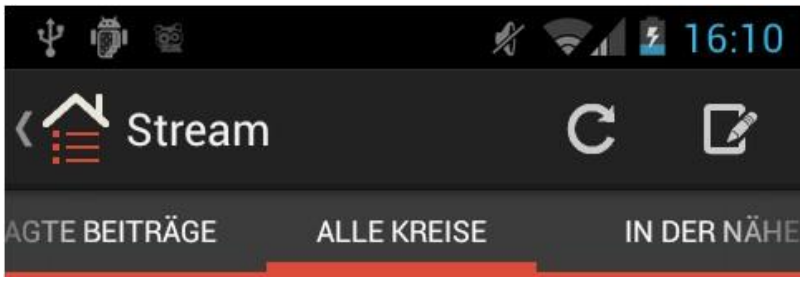


Figure 5.1: Example of an action bar.

A menu is a list of items that appears when clicking on a menu button or on the action bar. Menu items are defined using an XML file (e.g., menu.xml) placed in the res/menu directory. Figure 5.2 shows an example of a mainmenu.xml file.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
   <item
      android:id="@+id/Units"
      android:showAsAction="always"
      android:icon="@drawable/ic_Units"
      android:title="Units"/>
   <item
      android:id="@+id/Modules"
       …
      android:title="Modules">
   </item>
</menu>
```

Figure 5.2: Example of a menu definition containing two items.

The attribute showAsAction specifies the visibility of an item in the action bar:

- always: the item must always be displayed in the action bar
- ifRoom: the item is displayed only if there is enough space
- never: the item is not displayed in the action bar

a) Options menu

To create a menu in an activity (Java file) and display it, an instance of the MenuInflater class is used (via the method getMenuInflater()). It is mandatory to override the method onCreateOptionsMenu to create the menu. Figure 5.3 shows an example of accessing an XML menu file from an activity to create the menu.

```
@Override
 public boolean onCreateOptionsMenu(Menu menu) {
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.mainmenu, menu);
   return true;
 }
```

Figure 5.3: Creating a menu in an activity.

b) Contextuel menu

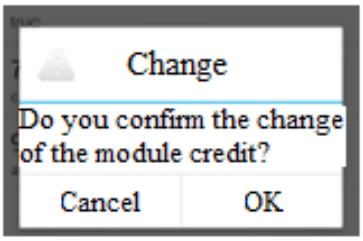
To handle actions when menu items are selected, the method onOptionsItemSelected is used. Figure 5.4 shows an example of defining actions for menu items upon selection.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
switch (item.getItemId()) {
case R.id.Units:
...
return true;
case R.id.Modules:
...
return true;
...
default: return super.onOptionsItemSelected(item);
}
}
```

Figure 5.4: Handling menu item selection actions.

5.2 Dialog Boxes

A dialog is a graphical window that appears over (typically at the bottom or center of) the interface to display information or request user interaction. Figure 5.6 shows an example of an alert dialog.




```
Builder confirm = new AlertDialog.Builder(this);
confirm.setTitle("Change");
confirm.setIcon(android.R.drawable.ic_dialog_alert);
confirm.setMessage("Do you confirm the change of the module credit?");
```

Figure 5.6: Example of an alert dialog.

As with custom notifications, it is also possible to create custom dialogs. Figure 5.7 shows an example of creating and displaying a custom dialog, where the dialog's interface is defined in the file dialog_module.xml.

```
final Dialog dialog = new Dialog(this);
dialog.setContentView(R.dialog_module);
dialog.setTitle("Create a personal dialogue");
dialog.show();
```

Figure 5.7: Example of a custom dialog.

Event listeners can be added to the views (such as buttons) within a custom notification or a custom dialog interface.

5.3 Specific Dialog Box

In an Android mobile application that contains a 2D drawing, a specific dialog box can be used to perform specific tasks, such as selecting the color of the 2D drawing, choosing the file in which the drawing is saved, and so on. Figure 5.8 presents an example of creating a custom dialog box for selecting the color of a 2D drawing.

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
Context context = getActivity();
Builder builder = new AlertDialog.Builder(context);
ColorPickerView cpv = new ColorPickerView(context);
builder.setView(cpv);
… // Creating a listener to change the color.
return builder.create();
}
```

Figure 5.8: Example of Creating a Custom Dialog Box for Selecting the Color of a 2D Drawing in Android

Chapter 6: AndroidManifest.xml and communication between components

6.1 AndroidManifest.xml file

To create a new Android mobile application, the first step is to declare the application in the application manifest file (AndroidManifest.xml). All activities of the application must also be declared in this manifest.

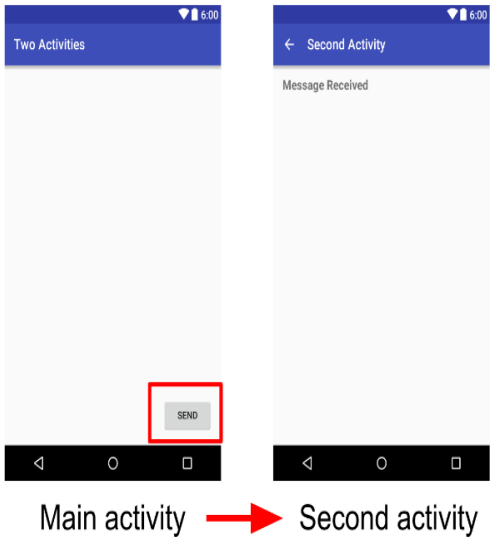
Figure 6.1 shows an example of an XML manifest file for an application composed of two activities (MainActivity and SecondActivity).

We can define application properties in the manifest such as:

- android:sharedUserId: This property represents the Unix user ID (UID) of the application. It helps secure the application. To allow two applications to access each other, they must share the same UID.
- <uses-permission ...>: This property grants the application the permissions it needs, such as access to the internet, camera, GPS, etc.

We can use <intent-filter> in the manifest to define the conditions under which an activity is launched, such as:

- android.intent.action.MAIN: Specifies that the activity is the main entry point.
- android.intent.category.LAUNCHER: Specifies that the activity can be launched from outside (e.g., from the app launcher).




```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.android.twoactivities">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_la
    uncher"
    android:label="@string/app_na
    me"
    android:supportsRtl="true"
    android:theme="@style/AppTh
    eme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".SecondActivity"
      android:label="@string/activity2_name"
      android:parentActivityName=".MainActivity">

</activit
      y>
</applicat
      ion>
</manifest>
```

Figure 6.1: Example of a manifest file for an application with two activities.

6.2 Communication between components

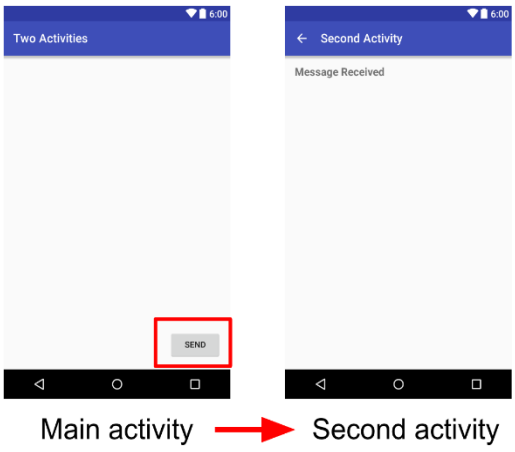
Navigation between activities is achieved using Intent. An Intent is an object that allows one activity to start another. It sends a request to perform an action, and it may also carry data or additional information required by the target activity.

An Intent consists of three main attributes:

- Action: Represents the operation to be performed. Examples include:
  - Intent.ACTION_WEB_SEARCH: to perform a web search
  - Intent.ACTION_CALL: to make a phone call
- Data: Represents the data required to perform the action (e.g., a phone number for a call).
- Category: Represents a classification of the action. Examples include:
  - Intent.CATEGORY_LAUNCHER: specifies that the activity should be launched when the application starts
  - Intent.CATEGORY_HOME: specifies that the activity should be displayed when the phone starts.

a) Explicit Intents

An activity can be launched either at application startup or from another activity. In both cases, an Intent is used. Figure 6.2 shows an example of launching SecondActivity from MainActivity within the same application.

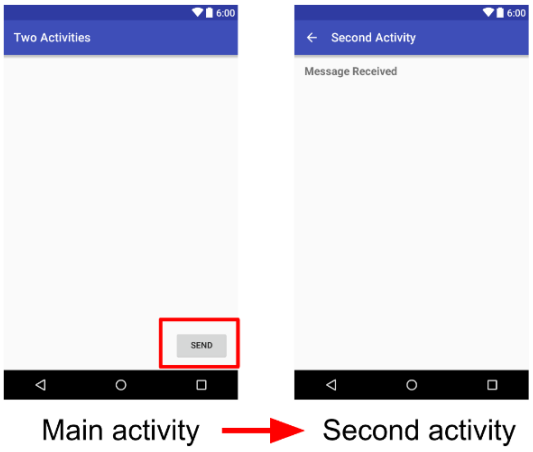



```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

public void launchSecondActivity(View view) {

    Intent intent = new Intent(this,
    SecondActivity.class); startActivity(intent);
}
}
```

Figure 6.2: Example of launching SecondActivity from MainActivity.

Launching Activity 2 from Activity 1 can be done in two ways:

- Without waiting (no result expected)
- Activity 1 is finished (using the finish() method) immediately after launching Activity 2, without waiting for any result. The user cannot navigate back to Activity 1. Figure 6.3 shows an example of launching without waiting.

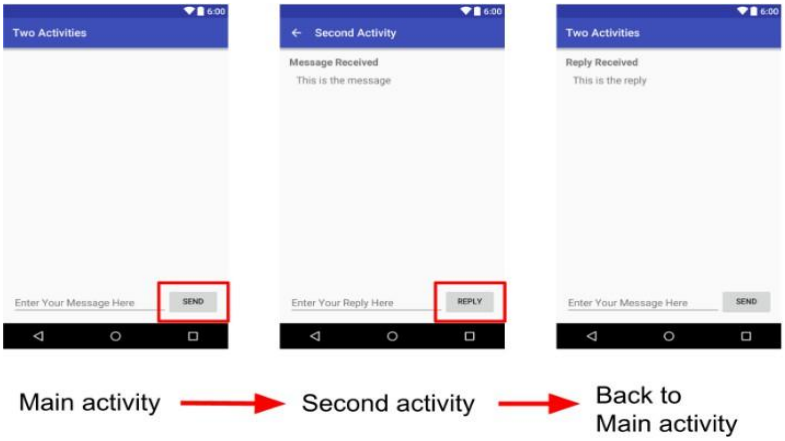



```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

public void launchSecondActivity(View view) {

    Intent intent = new Intent(this, SecondActivity.class);
    startActivity(intent);
    finish();
}
```

Figure 6.3: Launching Activity 2 without waiting.

b) Implicit Intents

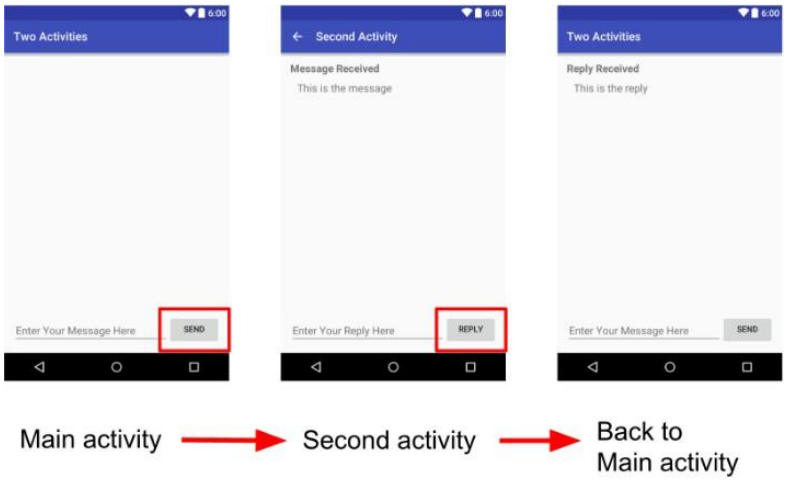
Launching with waiting means that Activity 1 starts Activity 2 and waits for a result. The user can navigate back to Activity 1. In this case, the method startActivityForResult() is used instead of startActivity(). Figures 6.4 and 6.5 show an example of launching Activity 2 (SecondActivity) from Activity 1 (MainActivity) and receiving a result.



Activité 1 (*MainActivity*)

```
public static final int TEXT_REQUEST = 1;
@Override
  protected void onCreate(Bundle savedInstanceState) {

     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
  }

  public void launchSecondActivity(View view) {

     Intent intent = new Intent(this, SecondActivity.class);
     String message = mMessageEditText.getText().toString();

     intent.putExtra(EXTRA_MESSAGE, message);
     startActivityForResult(intent, TEXT_REQUEST);
  }
}
  public void onActivityResult(int requestCode, int resultCode,
                        Intent data) {
     super.onActivityResult(requestCode, resultCode, data);

     if (requestCode == TEXT_REQUEST) {
        if (resultCode == RESULT_OK) {
        String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);

           mReplyHeadTextView.setVisibility(View.VISIBLE);
           mReplyTextView.setText(reply);
           mReplyTextView.setVisibility(View.VISIBLE);
        }
     }
```

Figure 6.4: Launch with result (MainActivity).



Activité 2 (*SecondActivity*)

```
@Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_second);
     mReply = (EditText) findViewById(R.id.editText_second);

     Intent intent = getIntent();
     String message =
           intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
     TextView textView = (TextView) findViewById(R.id.text_message);
     textView.setText(message);
  }

  public void returnReply(View view) {
     String reply = mReply.getText().toString();

     Intent replyIntent = new Intent();
     replyIntent.putExtra(EXTRA_REPLY, reply);
     setResult(RESULT_OK, replyIntent);
     finish();
  }
```

Figure 6.5: Launch with result (SecondActivity).

c) Resolution of implicit Intents

An Intent can carry data from one activity to another. To send data from Activity 1, use: putExtra(name, value), where name is the key and value is the data (String, Integer, etc.). To retrieve data in Activity 2, use: getTypeExtra(name), where Type corresponds to the data type (e.g., getStringExtra, getIntExtra, etc.). Before retrieving the data, you must first obtain the Intent using: getIntent().

Chapter 7: Data base with SQLIte

7.1 SQlite

SQLite is a relational database management system implemented as a library written in the C language to manage a database using SQL.

The main feature of SQLite is its ease of integration into programs without requiring a connection to a server, unlike traditional database management systems such as MySQL and PostgreSQL. SQLite is used in many systems such as Firefox, Skype, Android, Google Gears, etc.

SQLite uses most SQL commands such as: CREATE, INSERT, UPDATE, DELETE, and SELECT. Figure 7.1 shows an example of SQL queries used to create and manage a database.

```
sqlite> CREATE TABLE Modules (
_id INTEGER PRIMARY KEY AUTOINCREMENT,
nom_module TEXT NOT NULL,
nom_unit TEXT NOT NULL,
coff_module INTEGER,
credit_module INTEGER);

sqlite> INSERT INTO Modules VALUES (1, "Mobile applications", "UEF 6.11", 3, 5);
sqlite> INSERT INTO Modules VALUES (2, " Security", "UEF 6.12", 3, 5);
sqlite> INSERT INTO Modules VALUES (3, "Artificial  Intelligence", "UEF 6.21", 3, 5);

sqlite> SELECT * FROM Modules;

1| Mobile applications |UEF 6.11|3|5
2| Security |UEF 6.12|3|5
3| Artificial  Intelligence |UEF 6.21|3|5

sqlite> SELECT COUNT(*) FROM Modules WHERE unit_name LIKE ' UEF%';
sqlite> DELETE FROM Modules WHERE _id = 1;
sqlite> UPDATE Modules SET coff_module = 3 WHERE _id=2;
sqlite> DROP TABLE Modules;
```

Figure 7.1: Example of SQLite queries.

7.2 SQLite in an Android Application

Android uses SQLite to store and manage complex data. The two main Android classes used to create and manage a database are: SQLiteDatabase and Cursor.

To manipulate a database in Android, the following steps are performed. Figure 7.2 shows an example of database manipulation using SQLite in Android:

- Opening or creating the database (using the SQLiteDatabase class)
- Executing different SQL queries (using SQLiteDatabase and Cursor classes)
- Closing the database

```
class ModuleActivite extends Activity
{
private SQLiteDatabase data_base;
String path = this.getFilesDir().getPath().concat("/modules.db");
void onCreate(...) {
...
data_base = SQLiteDatabase.openOrCreateDatabase(path, null);
data_base.execSQL("CREATE TABLE IF NOT EXISTS Modules(_id integer primary key
autoincrement, nom_module VARCHAR, nom_unit VARCHAR, coff_module integer,
credit_module integer);");
…
Cursor curseur = data_base.rawQuery("SELECT * FROM Modules WHERE...");
…
curseur.close();
}
void onDestroy() {
...
base_de_donnees.close();
}
}
```

Figure 7.2: Example of database manipulation using SQLite in Android.

The execSQL method of the SQLiteDatabase class allows executing SQL queries such as CREATE, SELECT, INSERT, DELETE, UPDATE, and DROP without returning any value. There are two variants of this method:

- void execSQL(String sql_query): executes a query as a string.

  Example: database.execSQL("DROP TABLE Modules");

- void execSQL(String sql_query, Object[] parameters): executes a query with parameters.

Example:

db.execSQL("DELETE FROM Modules WHERE _id BETWEEN ? AND ?", new Object[] {3, 5});

The rawQuery method of the Cursor class allows executing a SELECT query and retrieving all selected elements from the database into a cursor. Table 7.1 presents some methods of the Cursor class.

Method	Description
moveToFirst ()	Moves the cursor to the first result
moveToNext ()	Moves the cursor to the next element
isAfterLast()	Checks if the end of the result set is reached
getCount()	Returns the number of selected elements
getColumnCount()	Returns the number of columns
getColumnName(n)	Retrieves the name of column n
getInt(n), getLong(n), getFloat(n), getString(n),	Retrieves the value of column n
isNull(n)	Checks if column n is null
close()	Closes the cursor

Table 7.1: Some methods of the Cursor class.

Figure 7.3 shows an example of a method used to retrieve the name of a module (based on its number) from the database.

```
public static String getNomModule(SQLiteDatabase data_base, long n)
{
Cursor curseur = data_base.rawQuery( "SELECT
nom FROM Modules WHERE _id=?", new String[]
{Long.toString(n)});
if (curseur == null) return null;
try {
if (curseur.moveToFirst() && ! curseur.isNull(1)) {
return curseur.getString(1);
} else
return null;
} finally {
curseur.close();
}
}
```

Table 7.3: Example of a method to retrieve the name of module number n.

Android also provides another class for database management called SQLiteOpenHelper. To manipulate a database, two abstract methods must be overridden:

- onCreate(SQLiteDatabase database)
- onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion)

The first method is used to create the database, while the second is used to update it (oldVersion is the previous version number and newVersion is the new version number). Figure 7.4 shows an example of using SQLiteOpenHelper to create and manage a database.

The methods getReadableDatabase() and getWritableDatabase() open or create the database and automatically call onCreate and onUpgrade.

- getReadableDatabase() is used to read data
- getWritableDatabase() is used to read and write data

```
public class ModulesSQLiteHelper extends SQLiteOpenHelper
{
private static final String data_base_name= "test.db"; private
static final int data_base_version = 1;

public ModulesSQLiteHelper(Context context)
{
super(context, data_base_name, null, data_base_version);
}
@Override
public void onCreate(SQLiteDatabase data_base)
{
data_base.execSQL("CREATE TABLE IF NOT EXISTS Modules(_id integer primary key autoincrement, nom_module VARCHAR, unit_name VARCHAR, coff_module integer, credit_module integer);");
}
@Override
public void onUpgrade(SQLiteDatabase data_base, int oldVersion, int newVersion)
{
bdd.execSQL("DROP TABLE IF EXISTS Modules");
onCreate(data_base);
}
}
public class MainActivity extends Activity {
private ModulesSQLiteHelper helper;
private SQLiteDatabase data_base;
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
...
helper = new ModulesSQLiteHelper(this);
data_base= helper.getReadableDatabase();
…
data_base = helper.getWritableDatabase();
...
}
}
@Override
protected void onDestroy()
{
super.onDestroy();
...
helper.close();
...
}
```

Figure 7.4: Example of using SQLiteOpenHelper in Android..

7.3 CursorAdapter and Loader

CursorAdapter is used to populate a list of items (ListView) from a database. To avoid blocking the user interface while executing SQL queries, Android uses a mechanism called a Loader.

The Loader class loads data asynchronously into an activity or fragment.

To create a ListView connected to a database using a CursorAdapter, follow these steps:

- Define a subclass of CursorLoader to perform SQL queries. Override:
  - onCreateLoader
  - onLoadFinished
  - onLoaderReset
- Define the CursorAdapter in the onCreate() method of the activity
- Open or create the database in onCreate()
- Create and start a CursorLoader in onCreate()

Figure 7.5 shows an example of creating a ListView connected to a database.

```
static private class ModulesCursorLoader extends CursorLoader
{
private SQLiteDatabase data_base;
public ModulesCursorLoader(Context context, SQLiteDatabase data_base) { super(context);
this. data_base = data_base;}
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) { adapter.changeCursor(cursor);
}
@Override
public void onLoaderReset(Loader<Cursor> loader) { adapter.changeCursor(null);
        }
public class MainActivity extends Activity implements
LoaderManager.LoaderCallbacks<Cursor>
{
private ModulesSQLiteHelper helper; private SQLiteDatabase data_base; private
SimpleCursorAdapter adapter;
private static final int LOADER_LISTE_Units_Modules = 1; @Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState); setContentView(R.layout.main);
        ...
 adapter = new SimpleCursorAdapter(this, android.R.layout.simple_list_item_2, null, new
 String[]{"nom_module", "coff_module"}, new int[]{android.R.id.text1, android.R.id.text2}, 0);
 ListView lm = (ListView) findViewById(R.id.listeModules); Lm.setAdapter(adapter);
 helper = new MySQLiteHelper(this); data_base = helper.getReadableDatabase();
 getLoaderManager().initLoader(LOADER_LISTE_Units_Modules,null,this); @Override
 public Loader<Cursor> onCreateLoader(int loaderID, Bundle bundle)
 {
 return new ModulesCursorLoader(getApplicationContext(), data_base);
 }
 }
```

Figure 7.5: Example of a ListView linked to a database.

7.4 Web Service

The objective of a Web Service is to implement an Android mobile application connected to a database on a remote server.

Access to the database (on the server) from a client is done through HTTP requests. As shown in Figure 7.6, the HttpURLConnection class (on the client side) is used to send HTTP requests to the server and receive responses.

There are two types of HTTP requests:

- GET → used for SELECT operations
- POST → used for INSERT, UPDATE, DELETE operations

The execution of the request on the server is handled by a PHP script.

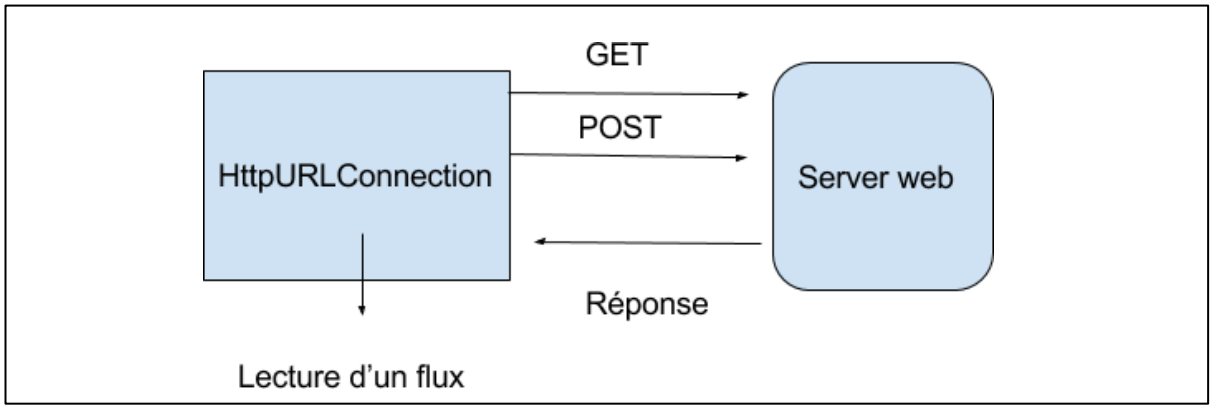


Figure 7.6: Accessing a remote database.

Steps of communication:

- The client sends an HTTP request (GET or POST)
- The PHP script executes the request and returns a response encoded in JSON
- The client receives and decodes the response

Figure 7.7 shows an example of a PHP script (get_module.php) executing a GET request.

```
// connexion au serveur SQL par PDO
$db = new PDO("pgsql:host=$hostname;dbname=$dbname", $login, $passwd);

// paramètres de la requête (TODO: tester la présence)
$id = $_GET['_id'];

// requête SQL
$query = $db->prepare("SELECT * FROM Modules WHERE _id=?");
$query->execute(array($id));

// encodage JSON de la réponse
echo json_encode($query->fetch(PDO::FETCH_NUM));
```

Figure 7.7: Example of a PHP script executing a GET request.

To allow an Android app to access the internet, the following permission must be added:

<uses-permission android:name="android.permission.INTERNET"/>
The PHP script is called within the Android activity to access the remote database.

```
public static Cursor getAll(RemoteDatabase data_base)
{
// requête Get à l'aide de la classe RemoteDatabase
String jsondata = data_base.get("get_module.php", 3);
// décoder les n-uplets et en faire un curseur
return data_base.cursorFromJSON(jsondata, new
String[] { "_id", " module_name", " unit_name" });
}
```

Figure 7.8: Example of calling a PHP script in an Android activity.

7.5 jQuery Mobile Framework

jQuery Mobile is a JavaScript library that simplifies the development of mobile web applications. It makes it easier to write JavaScript code within the HTML of these applications. jQuery Mobile is based on jQuery, HTML5, and CSS3.

The body of an HTML5 page using the jQuery Mobile framework includes attributes such as: page, header (page header), content (page content), and footer (text at the bottom of the page). Figure 7.9 shows the structure of the body of an HTML5 page using jQuery Mobile. Figure 7.10 presents an example of an HTML5 page using jQuery Mobile.

```
<body>
<div data-role= ‘page’ >
<div data-role=’header’>…>/div>
<div data-role=’content’>…>/div>
<div data-role=’footer’>…>/div>
</div>
</body>
```

Figure 7.9: Structure of the body of an HTML5 page using jQuery Mobile.

```
<!doctype html>
<html>
  <head>
  <title> Web page for managing modules </title>
  <link rel="stylesheet" href="jquery.mobile.css" />
  <script src="jquery.js"></script>
  <script src="jquery.mobile.js"></script>
</head>
<body>
<div data-role="page">
  <div data-role="header">
    <h1>Page Title</h1>
  </div>
  <div data-role="content">
    <p> Web page content (modules).</p>
  </div>
  <div data-role="footer">
    <h4>3 year Licence Computer Sciences (SI)</h4>
  </div>
</div>
```

Figure 7.10: Example of HTML5 code using the jQuery Mobile framework.

7.6 Mobile Mobile-Compatible WebApp Using the jQuery Mobile Framework

jQuery Mobile can be used to develop mobile web applications compatible with all mobile operating systems such as: Android, iOS, BlackBerry, Windows Phone, etc.

Using jQuery Mobile, it is possible to develop a mobile web application containing a single web page or multiple HTML5 pages. To create a multi-page mobile web application, each page must be included in a element and assigned the attribute data-role="page" with a unique id (e.g., id="page_identifier"). Navigation between pages is achieved using links such as: <a href="#page_identifier">.

We can also add graphical components to the pages of the mobile application, such as: buttons, lists, headers, footers, icons, dialog boxes, etc. Figure 7.11 shows an example of HTML5 code for an Android mobile web application developed using jQuery Mobile.

```
<!DOCTYPE html> <html lang="fr">
<head>
<title> Mobile web application for managing jQuery-based mobile modules </title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"  href="jquery.mobile-1.4.2/jquery.mobile-1.4.2.css">
<script  src="jquery.mobile-1.4.2/jquery-1.10.2.js"></script>
<script  src="jquery.mobile-1.4.2/jquery.mobile-1.4.2.js"></script>
</head>
<body>
<div data-role="page" id="main">
<div data-role="header">
<h1> Module management </h1>
</div>
<div data-role="content" id=" main">
<p><a href="#unit" data-transition="slidedown">Units</a></p>
<p><a href="#modules" data-transition="turn">Modules</a></p>
</div>
<div data-role="footer" data-position="fixed">
<h4>Copyright www.univ-oeb.dz 2019</h4>
</div> </div>
<div data-role="page" id="unit">
<div data-role="header">
<h1> Teaching unit </h1>
<p><a href="# main" data-transition="flip"> Main </a></p>
</div>
<div data-role="content">
<h4> Fundamental teaching unit</h4>
</div>
<div data-role="footer" data-position="fixed">
<h4>Copyright www.univ-oeb.dz 2026</h4>
</div> </div>
<div data-role="page" id="modules">
<div data-role="header">
<h1> Teaching modules </h1>
<p><a href="# main" data-transition="flip"> Main </a></p>
</div>
<div data-role="content">
<h4>Mobile applications</h4>
<h4>Security</h4>
<h4>Artificial Intelligence</h4>
</div>
<div data-role="footer" data-position="fixed">
<h4>Copyright www.univ-oeb.dz 2026</h4>
</div> </div>  </body>  </html>
```

Figure 7.11: Example of HTML5 code for a mobile web application developed using the jQuery Mobile framework.

Chapter 8: Development of simple application

8.1 Part I of a Mobile Application for Grade Management

The main objective of this exercise is to build a Human-Machine Interface (HMI) that includes several Android graphical components (Views). This interface allows the calculation of averages for some modules (within a fundamental teaching unit) of a third-year Licence's student (Licence 3 / Computer Science / Information Systems) by requesting the marks of tutorials (TD), practical work (TP), and exams for each module.

The application must also calculate the average of the fundamental teaching unit, as well as the number of credits obtained for this unit.
Mobile application for Grade Management (3-Licence Computer Science SI)
For now, we will focus only on creating the graphical interface similar to Figure 8.1.

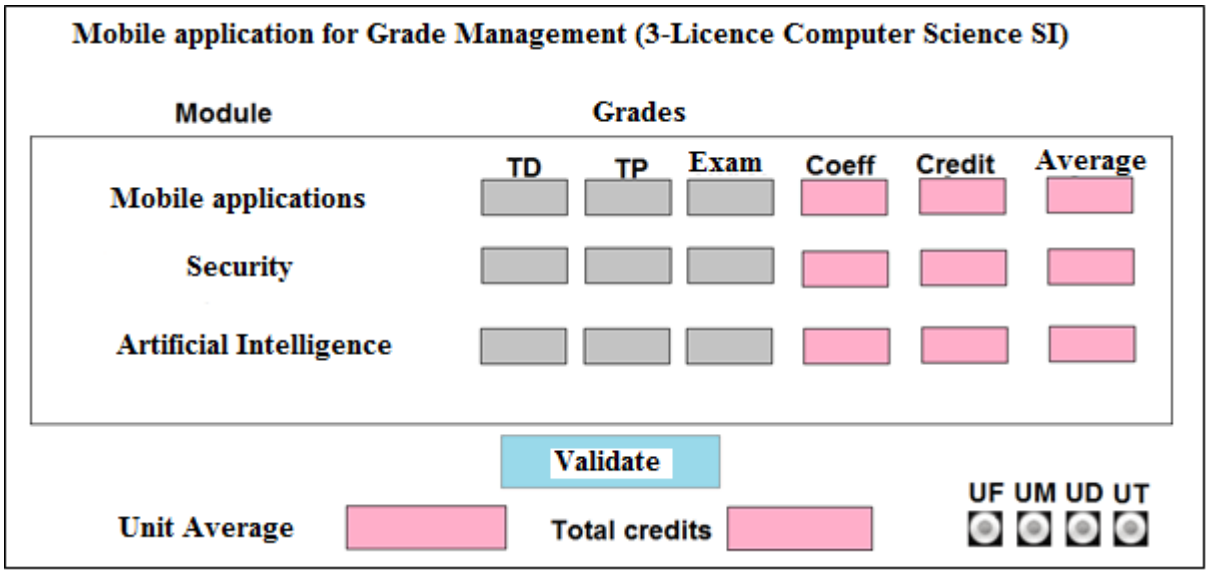


Figure 8.1: Initial version of the graphical interface of the mobile application.

1) Identify the Android graphical components involved in this HMI.
2) Build this HMI using Eclipse with the Android plugin (or Android Studio). It is required to write as little Java code as possible. Strings should preferably be defined in the strings.xml file.

8.2 Part II of a Mobile Application for Grade Management

The main objective of this exercise is to build an Android mobile application composed of multiple activities (i.e., multiple screens of the graphical interface). This application allows the calculation of averages for all modules (from all teaching units: fundamental,

methodology, discovery, and transversal) of a third-year Licence's student (Licence 3 / Computer Science / Information Systems) by requesting the marks of tutorials (TD), practical work (TP), and exams for each module of each unit.

The application must also calculate the average of each teaching unit, as well as the number of credits obtained for each unit. In addition, it must compute the overall average and the total number of credits obtained by the student during the first semester.

For now, we will focus on building an application similar to Figure 8.2.

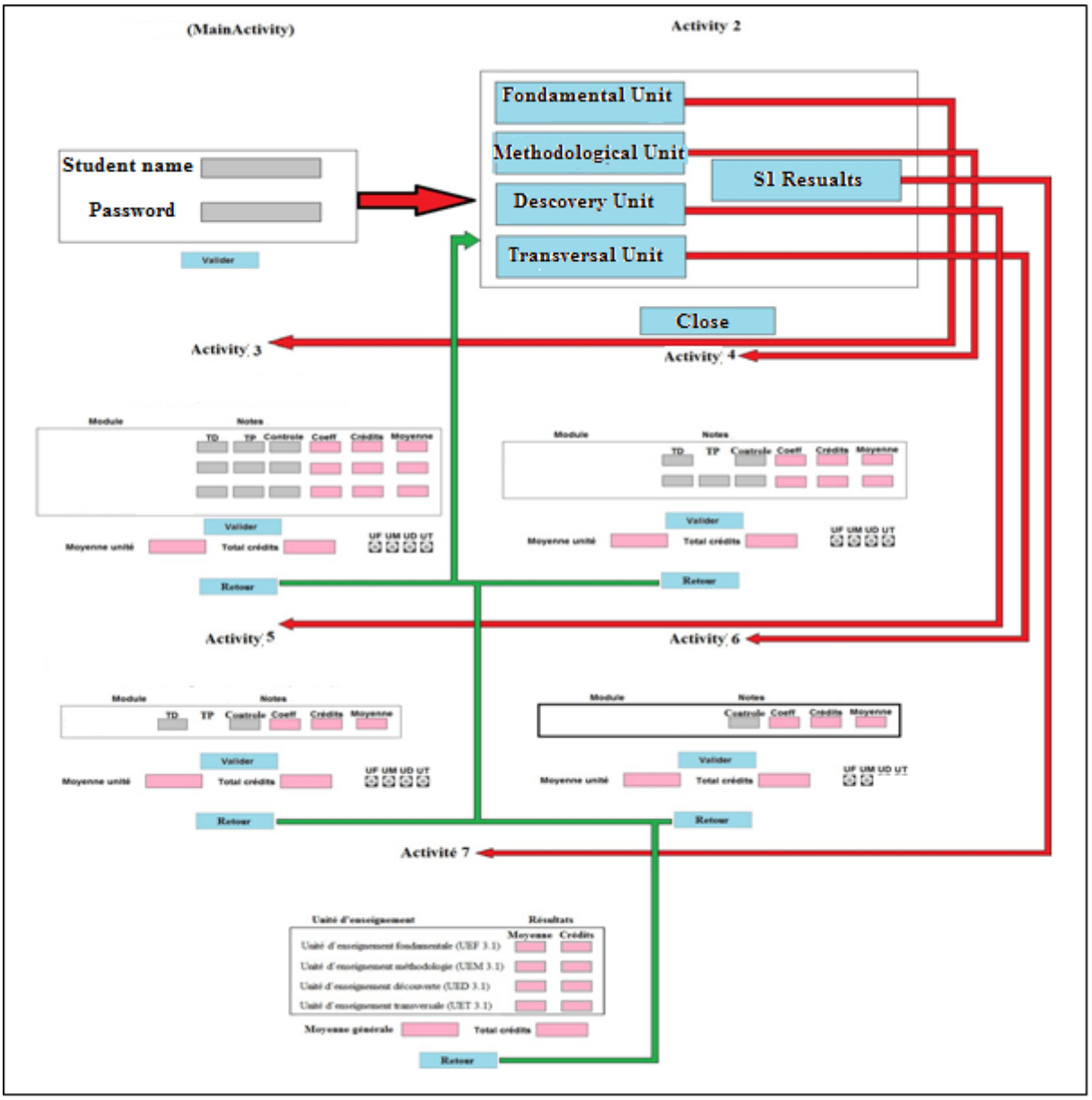


Figure 8.2: Initial version of the mobile application for TP 3.

8.3 Part III of a Mobile Application for Grade Management

The main objective of this exercise is to build an Android mobile application composed of multiple activities (i.e., multiple screens of the graphical interface), knowing that one of these activities manages a ListView.

This application allows the calculation of averages for all modules (from all teaching units: fundamental, methodology, discovery, and transversal) of a third-year Licence's student (Licence 3 / Computer Science / Information Systems) by requesting the marks of tutorials (TD), practical work (TP), and exams for each module of each unit.

The application must also calculate the average of each teaching unit, as well as the number of credits obtained for each unit. In addition, it must compute the overall average and the total number of credits obtained by the student during the first semester.

8.4 Part IV of a Mobile Application for Grade Management

The main objective of this exercise is to improve the usability of the mobile application implemented in Android in TP 4 by adding custom notifications and dialogs, a menu, an action bar, and dynamic fragments.

- A custom notification must confirm the addition of module grades (in activities 4, 5, 6, and 7).
- A custom dialog must be displayed if a grade is incorrect (less than 0 or greater than 20) to ask the user to correct the entered grade (in activities 4, 5, 6, and 7).
- An Edit menu (in activity 2) must display module grades by unit.
- An action bar (in activity 2) must display all menu items.
- One of the application activities must display two dynamic fragments (activity 3 must display one fragment for the list of units and another fragment to display and manage the different module layouts).

8.5 Part V of a Mobile Application for Grade Management

The objective of this TP exercise is to integrate a local database into the Android mobile application developed in TP 5 (Chapter 5). The different activities of the application must create and manage the database using SQLite queries.

- For the activities (or fragments) of the different unit modules, the classes SQLiteDatabase, Cursor, and SQLiteOpenHelper must be used to create the database and its tables, as well as to add and update module data such as: module name, exam

grade, TD grade, TP grade, coefficient, credits, etc.

- For the list of units (ListView), a CursorLoader and a custom CursorAdapter must be created to manage the data of this list, which must be stored in the same database.

8.6 Part VI of a Mobile Application for Grade Management

In this TP, it is required to use the concept of jQuery Mobile in order to convert the Android mobile application developed in TP 7 into a mobile web application. The objective is to implement a web-based system for managing the grades of different modules.

Bibliography

[1] Android 4 : Développement d'applications avancées, Auteur : Reto Meier. Édition : Pearson Education, 2012.

[2] jQuery Mobile, Auteur : Eric Sarrion, Editeur : Eyrolles, 2012.

[3] Développer avec les API Google Maps : Applications web, iPhone/iPad et Android, Auteur(s) : Fabien Goblet, Michel Dirix, Loïc Goblet, Jean-Philippe Moreux. Editeur : Dunod, 2010.

[4] Android 4 Les fondements du développement d'applications JAVA, Auteur : Nazim Benbourahla.

[5] Android, Concevoir et développer des applications mobiles et tactiles, Auteur : Florent Garin.

[6] http://socialcompare.com/fr/comparison/android-versions-comparison

[7] https://www.geeksforgeeks.org/android-system-architecture/

[8] https://www.scriptol.fr/mobile/statistiques.php

[9] https://www.instructables.com/id/How-To-Setup-Eclipse-for-Android-App-Development/

[10] https://www.embarcadero.com/starthere/xe5/mobdevsetup/android/fr/creating_an_android_emulator.html

[11] https://hal.archives-ouvertes.fr/cel-01082588v2/document

[12] Android : Apprenez à développer efficacement pour le leader des OS mobiles, Auteur : Florent Garin. Editeur : Dunod, 2012.

[13] Créez des applications Android : Le développement pour appareils mobiles Android à la portée de tous, Auteur : Frédéric Espiau. Editeur : OpenClassrooms, 2013.

[14] https://openclassrooms.com/fr/courses/2023346-creez-des-applications-pour-android/2024892-organiser-son-interface-avec-des-layouts

[15] https://www.supinfo.com/articles/single/1331-developpement-android-linearlayout-relativelayout

[16] http://www.iutbayonne.univ-pau.fr/~dalmau/documents/cours/android/Cours%20Android.pdf

[17] https://developer.android.com/guide/topics/ui/layout/grid

[18] http://cedric.cnam.fr/~farinone/RSX116/FOD/enonceTPAndroid.pdf

[19] https://google-developer-training.gitbooks.io/android-developer-fundamentals-course-practicals/content/en/Unit%201/21_p_create_and_start_activities.html#task1intro
[20] https://mathias-seguy.developpez.com/tutoriels/android/comprendre-cyclevie-activite/

[21] https://o7planning.org/fr/10435/tutoriel-android-listview

[22] https://a-renouard.developpez.com/tutoriels/android/personnaliser-listview/

[23] http://www.tutos-android.com/MTI/2017/Cours%202.pdf

[24] http://javamind-fr.blogspot.com/2012/08/creer-des-listes-personnalisees-en.html

[25] https://vogella.developpez.com/tutoriels/android/utiliser-barre-actions/

[26] http://www-igm.univ-mlv.fr/~forax/ens/java-avance/cours/pdf/Android3-Fragment-MVC.pdf

[27] http://www.lifl.fr/~dumoulin/enseign/pje/cours/6.fragments/1.fragments.pdf

[28] https://sql.sh/sgbd/sqlite

[29] http://projet.eu.org/pedago/sin/1ere/4-sqlite.pdf

[30] https://openclassrooms.com/fr/courses/2023346-creez-des-applications-pour-android/2027250-les-bases-de-donnees

[31] https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper

[32] https://vogella.developpez.com/tutoriels/android/background-processing-handlers-asynctask-loaders/

[33] https://openclassrooms.com/fr/courses/4428411-developpez-des-applications-android-connectees/4433916-lisez-des-objets-json-depuis-une-requete-http

[34] https://www.supinfo.com/articles/single/1035-creer-une-tache-asynchrone-android

[35] https://mathias-seguy.developpez.com/cours/android/oldch3_1_thread/

[36] http://supertos.free.fr/supertos.php?page=1249

[37] https://medium.com/@trishantsharma1997/asynctask-in-android-f594a565d676

[38] https://ensweb.users.info.unicaen.fr/isn4/cours/02-reseau.pdf

[39] https://www.openstreetmap.org

[40] https://github.com/osmdroid/osmdroid/wiki/How-to-use-the-osmdroid-library

[41] https://www.emse.fr/~picard/cours/2A/devsi/android.html#sec-7-1

[42] https://www.iut-info.univ-lille1.fr/~casiez/M2105/TP/TP2Evenements/tp2.pdf

[43] https://programmation-facile.developpez.com/tutoriels/javascript/comment-simplifier-creation-vos-applications-mobiles-avec-jquery-mobile/

[44] https://www.alsacreations.com/tuto/lire/1459-jQuery-Mobile-par-lexemple--la-Kiwiparty.html

[45] http://markdalgleish.com/presentations/jquerymobile/

[46] https://jquerymobile.com/