Mobile Application

Unit 1 – Introduction to Mobile Applications

Introduction:

- Mobile devices have become the main gateway to the Internet (more than desktops).
- They are replacing traditional channels to access information.
- Enterprises now design digital applications for a wide range of devices & platforms.
- Native apps are built for a specific OS & hardware.

Mobile Applications in Business:

- Mobile apps are now part of B2C digital strategy.
- Many companies follow "mobile-first" strategy – apps are first designed, developed & tested for mobile.
- Mobility disruption has a huge impact on enterprise revenue.
- Apps provide real-time info and create engaging user experiences.

Digital strategy considers:

1. User experience, performance, interactivity
2. Device limitations, form factors
3. Personalization & location needs

Key Drivers for Mobile Apps:

- Innovation
- Consumer behavior
- Personalized content delivery
- Mobile ecosystem
- Social networking

Mobile Apps in Various Domains:

- Retail & Consumer Goods
- Banking industry
- Logistics
- Healthcare
- Social Media
- Gaming
- Video Streaming

Mobile Application

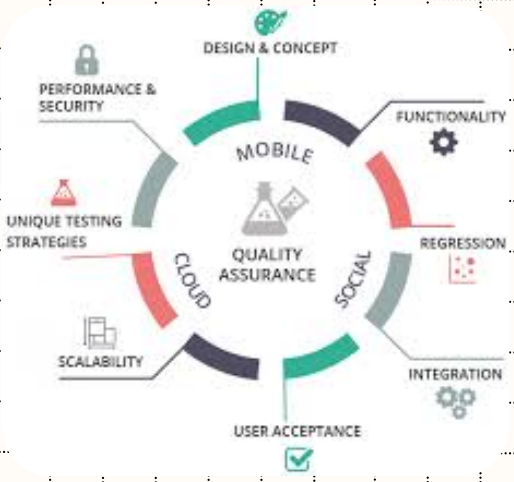
Unit 1 – Introduction to Mobile Applications

Considerations & Challenges

- Utility of app.
- Consumer engagement (rich experience).
- Productivity (efficient workflows).
- Revenue growth via user stickiness.
- Customer conversion & loyalty (personalized offers).
- Architecture – native vs hybrid vs web.
- Middleware for centralized config.
- Offline vs online data handling.
- App development principles.
- Compatibility, performance, security, target users.

Attributes of Mobile Apps

- Ubiquity – access info anytime, anywhere.
- User friendliness – responsive & interactive UI.
- Location awareness – uses GPS & sensors.
- Minimalistic – only essential features included.



Main Challenges in Mobile Apps:

- Device diversity & heterogeneous technologies – app must work smoothly across platforms.
- Security – protect data in storage & transmission.
- User experience – maximum engagement using device capabilities.
- Network issues – handle latency & bandwidth problems.
- Compliance – to standards, OS, platforms.

Mobile Application

Unit 1 – Introduction to Mobile Applications

PC-Based Applications

Standalone Applications:

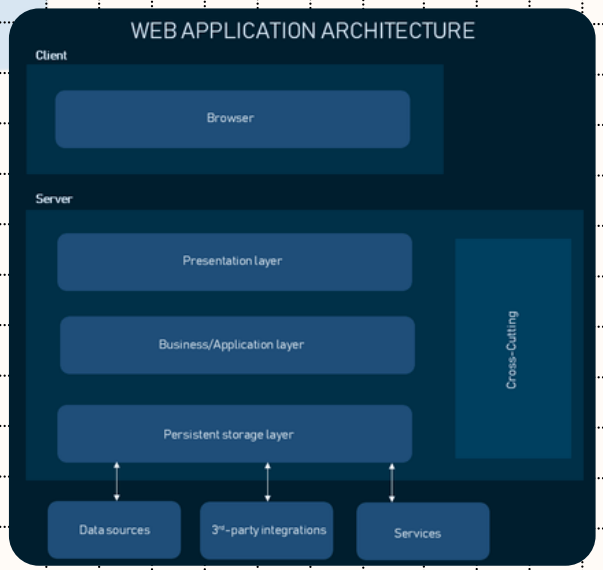
- Independent programs running on an OS.
- Don't need network resources.
- Examples: Word processors, media players, calculators.

Client-Server Applications:

- Client PC connected to a central server.
- Known as "thick clients".
- Examples: Database software, banking systems, network games.

Web-Based Applications

- Run on Internet browsers (desktop-focused).
- Use MVC architecture → separates View, Business, Data layers.



Layers:

- Presentation layer (View) → UI components (pages, forms, buttons).
- Business layer → business rules, workflows, caching, search.
- Data layer (Model) → DB handling, ORM, queries.
- Content Management → authoring, tagging, publishing, asset management.
- E-commerce layer → catalog, orders, shopping cart, promotions.
- Integration layer → Service Bus connects multiple apps.
- Security layer → authentication & authorization.

Mobile Application

Unit 1 – Introduction to Mobile Applications

Evolution of Mobile Applications

- **Embedded systems** – limited apps (e.g., calculator).
- **Device-specific apps** – built for particular devices.
- **OS-specific apps** – run on certain OS (e.g., games, media players).
- **Cloud-based apps** – always connected, real-time access.

Mobile Apps vs Web Apps

Feature	Mobile Apps	Web Apps
Installation	Installed via app stores (e.g., Apple App Store, Google Play)	Accessed via web browsers, no installation required
Platform Dependency	Platform-specific (iOS, Android)	Platform-independent, run on any device with a web browser
Performance	Generally faster and more responsive	Dependent on internet speed, can be slower
Offline Access	Can work offline once installed	Require internet connection, limited offline capabilities
Development Cost	Higher, separate versions needed for different platforms	Lower, a single version runs on multiple platforms
User Experience (UX)	Can leverage device-specific features (camera, GPS, etc.)	Limited access to device features
Updates	Users need to download updates from app stores	Automatically updated on the server side
Distribution	Distributed via app stores	Accessible via URLs
Maintenance	More complex, updates re ed for each platform	Easier, single update on the server

Mobile Application

Unit 1 – Kotlin Basics

Introduction:

- Kotlin → A cross-platform, statically typed, general-purpose language with type inference.
- Fully interoperable with Java → You can call Java from Kotlin & Kotlin from Java.
- Sponsored by JetBrains & Google → Through the Kotlin Foundation.

Why Kotlin for Android?:

- Reduces code length → fewer lines, more convenience.
- Everything possible in Java can also be done in Kotlin.
- Both languages are 100% interoperable → Can use Java & Kotlin in the same project.

Features of Kotlin:

- Modern & concise → Write less code, achieve same functionality.
- Safe & stable apps → Less chance of crashes.
- Faster development → Build apps quickly & efficiently.
- Industry trend → Most Android developers prefer Kotlin today.

Sample Code:

```
fun main() {
    println("Hello, world!")
}
```

- Function names → camelCase & should be verbs.
- Braces {} → Opening brace on same line, closing brace on a new line.
- Indentation → Keep body indented properly.

Mobile Application

Unit 1 – Kotlin Basics

Variables in Kotlin

```
val count: Int = 2    // immutable variable
var number = 10       // mutable variable
```

- val → Immutable (cannot change).
- var → Mutable (can change).

Can use string templates:

```
fun main() {
println("You have ${unreadCount + readCount} messages")}
```

Comments:

- Single-line → // comment
- Multi-line → /* comment */

Classes & Objects

Class syntax:

```
class ClassName(param1: Type, param2: Type) {

}
```

- val property → Immutable (unchangeable).
- var property → Mutable (changeable).
- Object = Instance of a class.

Example:

```
class SmartDevice {
    fun turnOn() = println("Device ON")
    fun turnOff() = println("Device OFF")
}

fun main() {
    val tv = SmartDevice()
    tv.turnOn()
}
```

Mobile Application

Unit 1 – Kotlin Basics

Default return type = Unit (like void in Java).

```
fun greet(): Unit {
    println("Hello!")
}
```

Returning values:

```
fun add(a: Int, b: Int): Int {
    return a + b
}
```

Parameters in Functions

Single parameter:

```
fun greet(name: String) = "Hello, $name!"
```

Multiple parameters:

```
fun greet(name: String, age: Int) = "Happy $age, $name!"
```

Default arguments:

```
fun greet(name: String = "Rover", age: Int) = "Happy $age, $name!"
```

Inheritance in Kotlin:

- Inheritance → New class inherits properties of another class.
- Superclass / Base class → Parent class.
- Subclass / Derived class → Child class.
- open keyword → Used to allow inheritance (by default, classes are final

Example:

```
open class Vehicle(val brand: String) {
    fun start() = println("$brand started")
}

class Car(brand: String): Vehicle(brand)
```

Mobile Application

Unit 1 – Kotlin Basics

Constructors

Primary Constructor → Declared in class header.

```
class Add(val a: Int, val b: Int)
```

Init block → Executes code during object creation.

```
init {
   println("Object created")
}
```

Secondary Constructor → Allows multiple ways to create an object.

```
class Add {
   constructor(a: Int, b: Int) {
      println(a + b)
   }
}
```

Extension Functions

Used to add new functions to existing classes without inheritance.

Example:

```
class Circle(val radius: Double) {
   fun area() = Math.PI * radius * radius
}

// Extension function
fun Circle.perimeter() = 2 * Math.PI * radius

val c = Circle(2.5)
println(c.perimeter())
```

Mobile Application

Unit – Layouts in Android

Introduction to Layouts:

- Layout = A container used to hold and arrange Views (buttons, text, images, etc.).
- Layouts decide the size, position, and organization of views.
- Defined in XML files (cannot be changed directly at runtime).
- In Android, layouts are also called ViewGroups (because they group views).
- Common layouts: LinearLayout, RelativeLayout, AbsoluteLayout, ConstraintLayout, FrameLayout, TableLayout, GridLayout.

LinearLayout:

- Arranges elements sequentially → either Horizontal or Vertical.

Important attributes:

- android:orientation → horizontal / vertical.
- android:layout_width & layout_height → size of the control.
- android:layout_weight → distribute extra space among views.
- android:gravity → aligns content inside a view.
- android:layout_gravity → aligns the view inside container.
- android:padding → space between content & boundary.
- Example use: Form with TextViews and Buttons stacked vertically.

Example (Vertical LinearLayout):

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/andr
oid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a TextView" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is a Button" />
</LinearLayout>
```

Mobile Application

Unit – Layouts in Android

Gravity Attribute

- Used to align content inside a control.

Options:

- center, left, right, top, bottom
- center_horizontal, center_vertical, fill_horizontal, fill_vertical

Example: android:gravity="center" → places content at center of container.

Example (Horizontal with Weight):

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.0" />

    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1.0" />

    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.0" />
</LinearLayout>
```

Mobile Application

Unit – Layouts in Android

RelativeLayout

- Places each view relative to other views or parent container.
- Useful for flexible positioning.

Key attributes:

- android:layout_alignParentTop/Bottom/Left/Right → align with parent.
- android:layout_centerHorizontal / centerVertical / centerInParent.
- android:layout_above / below / toLeftOf / toRightOf → relative to another view.
- android:layout_alignTop / Bottom / Left / Right / Baseline → align with another control.

Spacing attributes:

- android:padding → space inside control.
- android:layout_margin → space outside control.

Example

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="15dp"
        android:layout_marginLeft="20dp" />

    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="28dp"
        android:layout_toRightOf="@id/Apple"
        android:layout_alignParentTop="true" />

    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="200dp"
        android:layout_height="50dp"
        android:layout_below="@id/Apple"
        android:layout_alignParentLeft="true" />
</RelativeLayout>
```

Mobile Application

Unit – Layouts in Android

Constraint Layout:

- Most powerful & flexible layout.
- Designed using drag-and-drop editor in Android Studio.

Features:

- Efficient for complex UIs (better than Linear & Relative).
- Can manage position & size without nesting layouts.
- Supports animations & group control.
- Improves performance of UI.

Dependency:

```
implementation "androidx.constraintlayout:constraintlayout:2.1.4"
```

Example

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World"
        android:textSize="20sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Mobile Application

Unit – Layouts in Android

Absolute Layout

- Each child is placed at an exact X, Y coordinate.
- 0,0 coordinate = top-left corner of screen.
- Not recommended → UI breaks on different screen sizes & resolutions.

Example

```
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Product Code:"
        android:layout_x="5dp"
        android:layout_y="40dp"/>

    <EditText
        android:id="@+id/product_code"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_x="110dp"
        android:layout_y="30dp"/>
</AbsoluteLayout>
```

ImageView

- Used to display images.

Example

```
<ImageView
    android:id="@+id/first_image"
    android:src="@drawable/samplepic"
    android:layout_width="250dp"
    android:layout_height="100dp"
    android:scaleType="fitXY"
    android:adjustViewBounds="true"/>
```

Mobile Application

Unit – Layouts in Android

Frame Layout

- Used to display a single View.
- Additional views overlap (stacked on top of each other).
- Common for image switchers or overlapping views.

Visibility options:

- View.VISIBLE → shown.
- View.INVISIBLE → hidden but takes space.
- View.GONE → hidden and takes no space.

Example

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/first_image"
        android:src="@drawable/pic1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"/>

    <ImageView
        android:id="@+id/second_image"
        android:src="@drawable/pic2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click the image to switch"
        android:layout_gravity="center_horizontal|bottom"
        android:textColor="#ffffff"
        android:background="#333333"/>
</FrameLayout>
```

Mobile Application

Unit – Layouts in Android

Kotlin Code for Switching Images:

Example

```
val firstImage = findViewById<ImageView>(R.id.first_image)
val secondImage = findViewById<ImageView>(R.id.second_image)

firstImage.setOnClickListener {
    secondImage.visibility = View.VISIBLE
    it.visibility = View.GONE
}

secondImage.setOnClickListener {
    firstImage.visibility = View.VISIBLE
    it.visibility = View.GONE
}
```

Mobile Application

Unit – 2 : Activity in Android

Components of an Android Application

Activities:

- Presentation layer of the app.
- Built using Activity class (UI + user interaction).
- Uses Fragments & Views to display layout.

Example

```
class MainActivity : AppCompatActivity() { }
```

Services:

- Work in the background (invisible workers).
- Update data, trigger notifications, run tasks even when app is inactive.

Example

```
class MyService : Service() { }
```

Content Providers:

- Manage & store app data (often with SQLite database).
- Share data between apps.

Example

```
class MyProvider : ContentProvider() {
    override fun onCreate(): Boolean { return true }
}
```

Broadcast Receivers:

- Listen to system-wide events (e.g., SMS received, WiFi change).
- Make apps event-driven.

Mobile Application

Unit – 2 : Activity in Android

Components of an Android Application

Intents:

- Message-passing framework.
- Used to start Activities, Services, Broadcast Receivers.
- Example: Starting new Activity.

Example

```
val intent = Intent(this, SecondActivity::class.java)
startActivity(intent)
```

Widgets:

- Small visual components (on Home Screen).
- Special type of Broadcast Receivers.
- Example: Weather widget, Clock widget.

Notifications:

- App alerts to draw user attention without interrupting.
- Example: Email popup, Messenger notification.

Activity in Android

- Activity = Entry point for user interaction.
- Every app (small or big) has at least one Activity.
- Unlike normal apps (with main()), Android starts code via Lifecycle methods.
- Every Activity has a layout set by setContentView().

Example

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Mobile Application

Unit – 2 : Activity in Android

Activity Lifecycle Methods

1) onCreate():

- Called when Activity starts.
- Used to initialize UI & variables.
- Loads layout using setContentView().

Example

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}
```

2) onStart()

- Called when Activity becomes visible.

Example

```
override fun onStart() { super.onStart() }
```

3) onResume():

- Called when Activity is in foreground & interactive.

Example

```
override fun onResume() { super.onResume() }
```

4) onPause()

- Called when Activity is partially hidden (another Activity comes in front).

Example

```
override fun onPause() { super.onPause() }
```

Mobile Application

Unit – 2 : Activity in Android

Activity Lifecycle Methods

5) onStop():

- Called when Activity is no longer visible.

Example

```
override fun onStop() { super.onStop() }
```

6) onDestroy():

- Called before Activity is destroyed (final cleanup).

Example

```
override fun onDestroy() { super.onDestroy() }
```

7) onRestart():

- Called when Activity restarts after being stopped.

Example

```
override fun onRestart() { super.onRestart() }
```

Declaring Activity in Manifest

- Every Activity must be declared in AndroidManifest.xml.

Example

```
<application ...>
    <activity
android:name=".SecondActivity"
android:exported="true" />
</application>
```

- MainActivity must have:

Example

```
<intent-filter>
   <action
android:name="android.intent.action.M
AIN" />
   <category
android:name="android.intent.category
.LAUNCHER" />
</intent-filter>
```

Mobile Application

Unit – 2 : Activity in Android

Activity Lifecycle Summary

- Created → onCreate()
- Started → onStart()
- Resumed → onResume() (interactive)
- Paused → onPause() (partially hidden)
- Stopped → onStop() (not visible)
- Destroyed → onDestroy()
- Restarted → onRestart()

Mobile Application

Unit – 2 : Intents in Android

What is an Intent?

- Intent → A messaging object used to request an action from another app component.
- Used for communication between components.

3 main uses of Intent:

- Start an Activity → Open a new screen.
- Start a Service → Run tasks in background (e.g., downloading a file).
- Deliver a Broadcast → Send a system-wide message (e.g., device charging, boot completed).

Types of Intents

1. Explicit Intent:

- Specifies exact component (class name) to open.
- Mostly used inside the same app.
- Example: Starting a DownloadService.

Example

```
val downloadIntent = Intent(this,
DownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}
startService(downloadIntent)
```

2. Implicit Intent:

- Does not specify component name.
- Declares a general action → lets system find suitable app.
- Example: Sharing text with other apps.
- If multiple apps can handle → system shows a chooser dialog.
- If only one app matches → that app opens directly.

Mobile Application

Unit – 2 : Intents in Android

Example

```
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}

try {
    startActivity(sendIntent)
} catch (e: ActivityNotFoundException) {
    // Handle case when no app can handle intent
}
```

How Implicit Intent Works

- Activity A → Creates Intent with action description.
- Android System → Searches for apps with intent filter matching action.
- Matching Activity B → Started, and Intent passed to it via onCreate().

Building an Intent

An Intent object carries important information:

1. Component Name → Makes Intent explicit (specific class).
2. Action → Generic action (e.g., VIEW, SEND, PICK).
   Example: Intent.ACTION_VIEW.
3. Data → URI or MIME type of data.
   Example: Uri.parse("http://example.com").
4. Category → Extra info about target component.

Example:

1. CATEGORY_BROWSABLE → Can be started by a browser.
2. CATEGORY_LAUNCHER → Entry point shown in app launcher.

Mobile Application

Unit – 2 : Intents in Android

Examples of Common Intents

Start New Activity

```
val intent = Intent(this, SecondActivity::class.java)
startActivity(intent)
```

Open a Webpage

```
val intent = Intent(Intent.ACTION_VIEW)
intent.data = Uri.parse("https://www.google.com")
startActivity(intent)
```

Make a Call

```
val intent = Intent(Intent.ACTION_DIAL)
intent.data = Uri.parse("tel:123456789")
startActivity(intent)
```

Share Text

```
val intent = Intent(Intent.ACTION_SEND)
intent.type = "text/plain"
intent.putExtra(Intent.EXTRA_TEXT, "Hello World!")
startActivity(Intent.createChooser(intent, "Share via"))
```