# Architecting Mobile Applications

## 1. Introduction

Mobile applications serve various purposes and have different architectural requirements depending on their functionality, performance needs, and target platforms.

**Examples of Mobile Apps**:

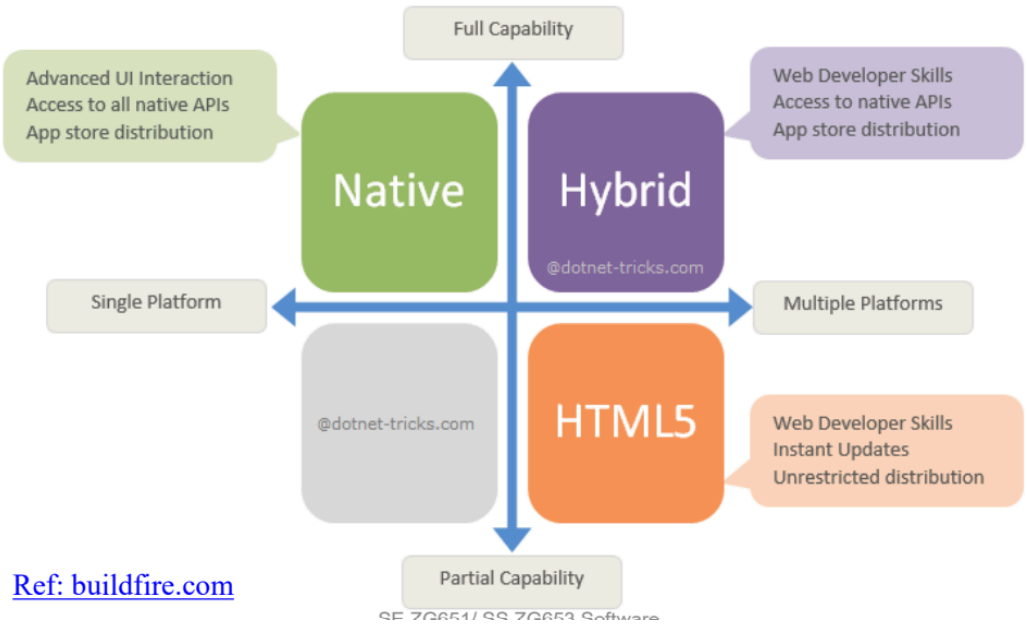
- **Uber**: Ride-hailing app
- **Swiggy**: Food delivery
- **Spotify**: Music streaming
- **Where Is My Train**: Train tracking
- **eCommerce**: Shopping apps like Amazon

---

## 2. Design Considerations for Mobile Applications

- **User Interface**: Simple, easy to navigate with large buttons and minimal features.
- **Responsive Design**: Adjusts to different screen sizes and orientations.
- **Performance**: Compact code for efficient CPU, memory, and storage usage.
- **Connectivity**: Ability to store data locally and sync later if connectivity is poor.

*Diagram Placeholder: Types of Mobile Apps*




Ref: buildfire.com

---

## 3. Types of Mobile Applications

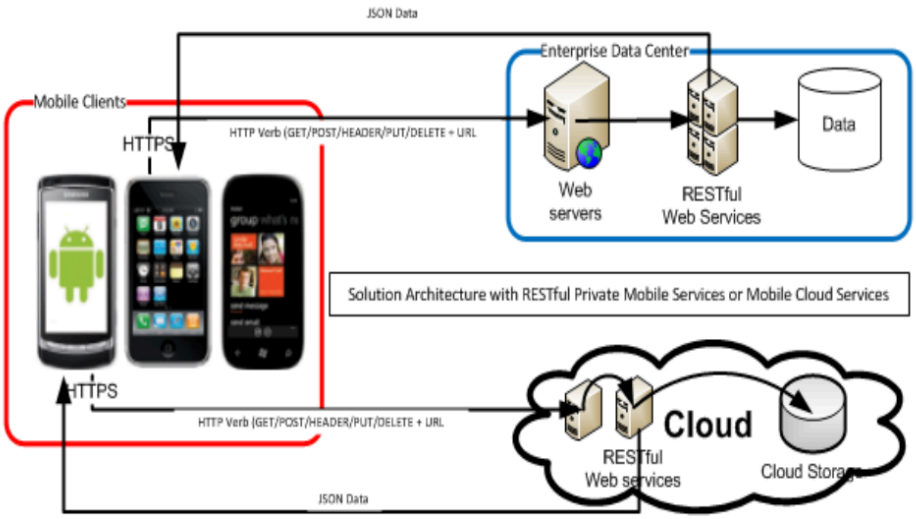
### Native Apps

- **Platform-Specific**: Developed separately for Android and iOS using Swift, Objective-C (iOS), Java, or Kotlin (Android).
- **Examples**: Google Maps, Facebook
- **Development Tools**: Android Studio, Xcode

### Web Apps

- **Browser-Based**: Accessed via a web browser and use HTML5, CSS3, JavaScript.
- **Examples**: Flipkart, BookMyShow

*Diagram Placeholder: Mobile Web Application*




### Hybrid Apps

- **Combines Native and Web Elements**: Written in HTML, CSS, JavaScript but runs in a native shell to access device features.
- **Examples**: Uber, Twitter
- **Development Tools**: Xamarin, React Native

*Diagram Placeholder: Pros & Cons of App Types*

| App type | Pro | Con |
|---|---|---|
| Native | • High performance<br>• Superior user experience<br>• Access to all features of OS | • Runs only on one platform<br>• Need to know special language<br>• Need to update versions |
| Web App | • Easy to deploy new versions<br>• Common code base | • Little scope to use device hardware<br>• Lower user experience<br>• Need to search for app |
| Hybrid | • Does not need browser<br>• Single code base<br>• Access to device hardware | • Slower |



---

## 4. User Interface (UI) Design Patterns

- **Quick Access**: Action bars for frequently used actions.
- **Third-Party Login**: Allow sign-in through social platforms like Google or Facebook.
- **Notifications**: Alerts for recent activity.
- **Discoverable Controls**: Buttons and options appear only when relevant, e.g., WhatsApp's "Forward" button.

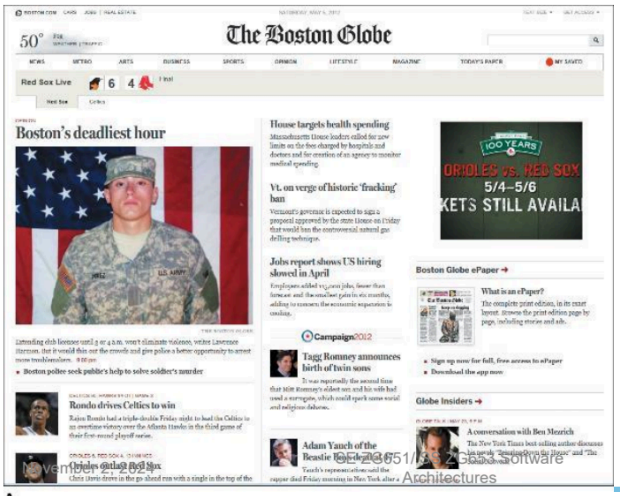
*Diagram Placeholder: UI Design Patterns*

## 5. Responsive Design

Responsive design ensures that the application adjusts to different devices (laptops, tablets, phones) with a single codebase, by using flexible grids, CSS media queries, and percentage-based images.

*Diagram Placeholder: Responsive Design Example (Desktop & Mobile)*

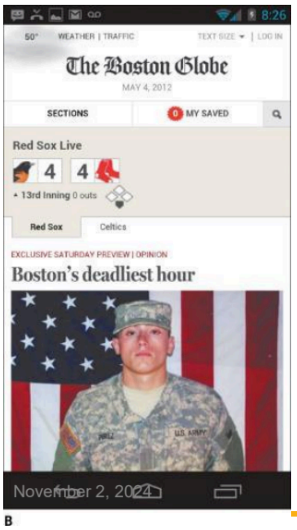
Example: Boston Globe News




A

This is how the display looks on desktop

Example: Boston Globe News




B


C

This is how the display looks on mobile

Adjusted vertically

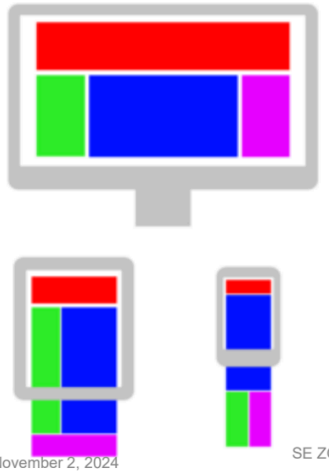
Single website for laptop & mobile & tablet

**Principle:** Adapt rendering depending on screen sizes & orientation


Ref: Wikipedia

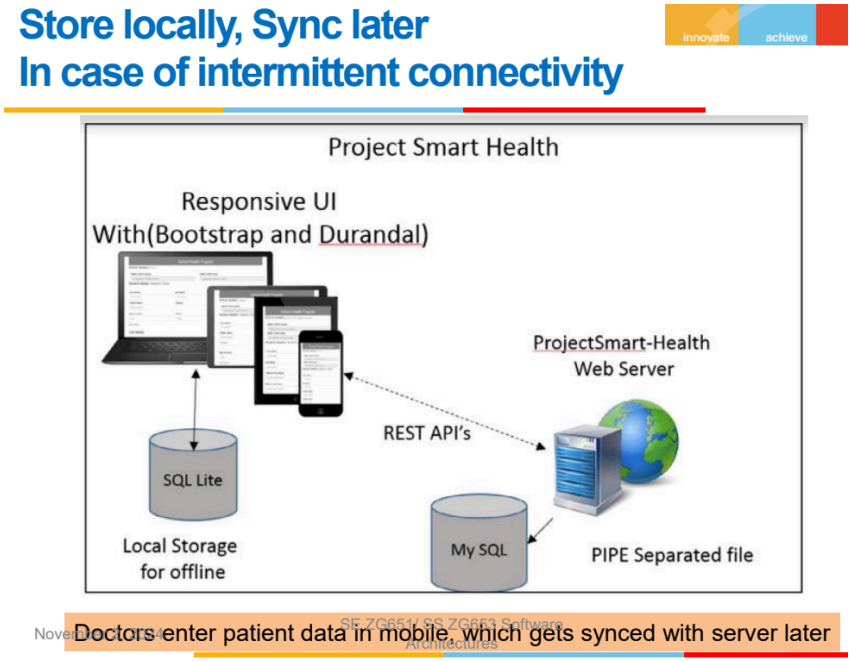


Ref: Wikipedia



**Flexi grids**

- Divide a screen into multiple columns
- Assign HTML elements to one or more columns
- Choose a different layout depending on screen size



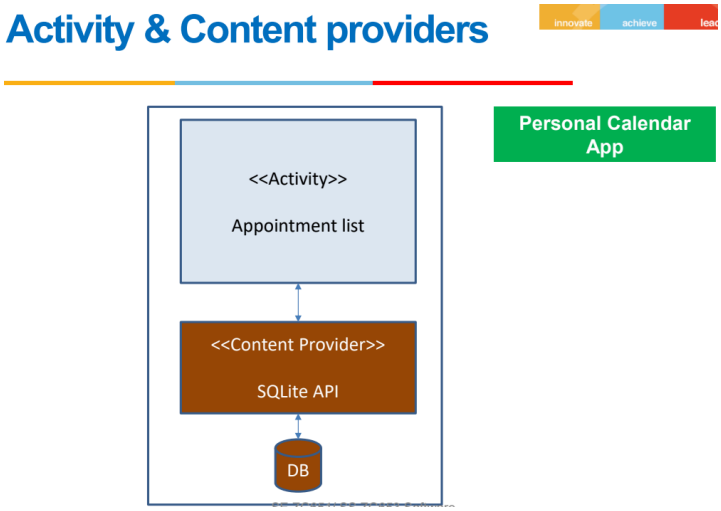

---

## 6. Android Application Architecture

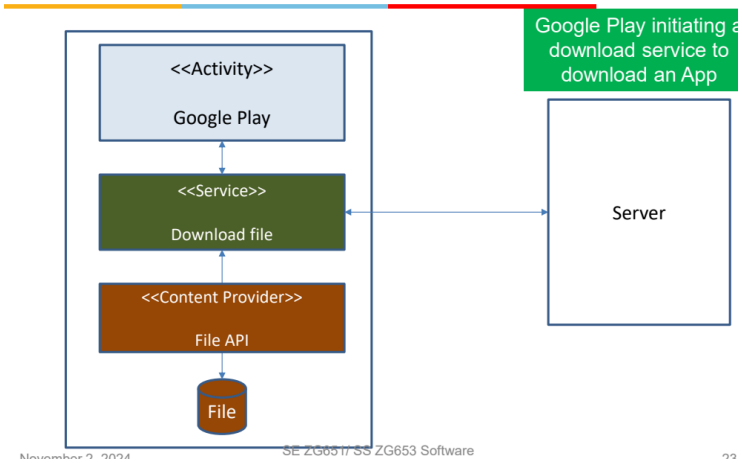
Android applications consist of four main components:

- **Activity**: User interface.
- **Service**: Background operations, e.g., music playback, file download.
- **Content Provider**: Data storage, e.g., SQLite.
- **Broadcast Receiver**: Responds to system events, e.g., SMS arrival.

*Diagram Placeholder: Android Application Structure*

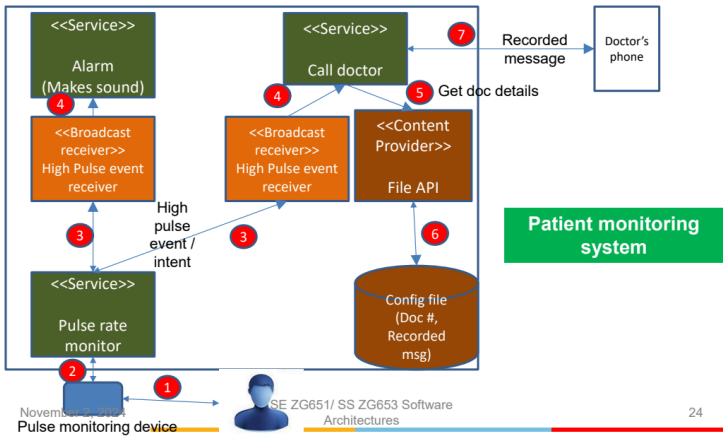


# (Background) Service



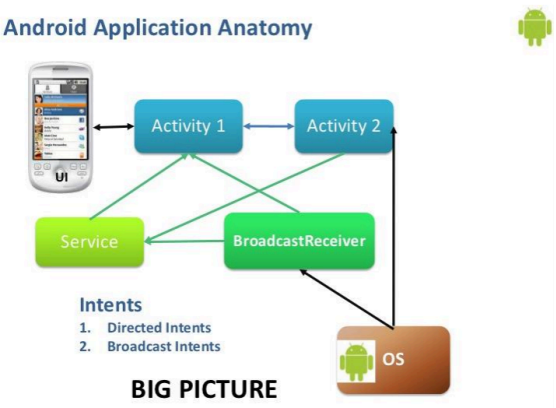



# Broadcast receiver (Event handlers)






# Android application structure



---

## 7. Example Components in a Mobile App (Uber)

- **UI**: Booking screen to select and view available cabs.
- **Broadcast Receiver**: Receives real-time cab location updates and sends location updates during the trip.
- **Database**: Stores configuration and user data.

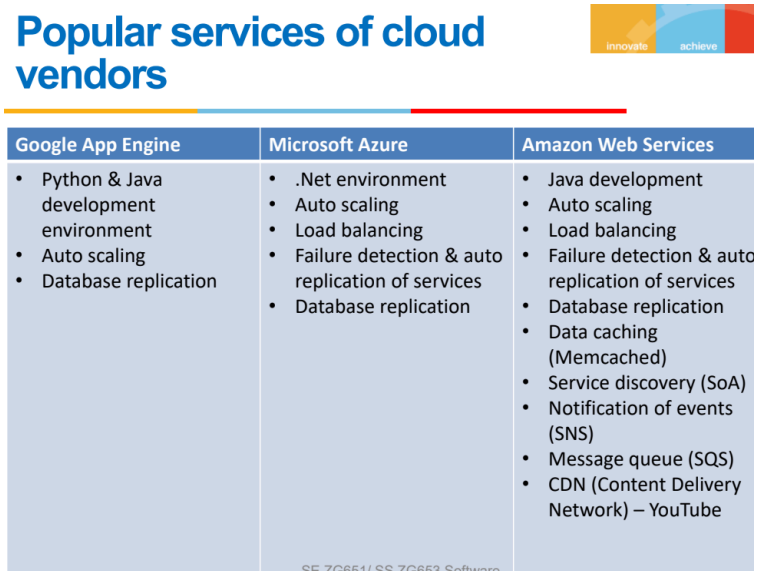
---

## 8. Cross-Platform and Cloud Integration

Cross-platform tools (e.g., Xamarin) allow developers to write a single codebase that can be deployed on multiple platforms (Android, iOS).

**Cloud Services**: Mobile apps often rely on cloud services like Google App Engine, AWS, or Microsoft Azure for scalability, load balancing, and data storage.

*Diagram Placeholder: Mobile Software Platform*




---

## 9. Mobile Application Architecture Layers

**Software Platform Layers**:

1. **Applications**: End-user apps.
2. **Framework**: Core APIs for application development.

3. **Core Libraries**: Essential libraries for operations and data handling.
4. **Kernel & Device Drivers**: Interface between hardware and software.

*Diagram Placeholder: Mobile Software & Hardware Platform*

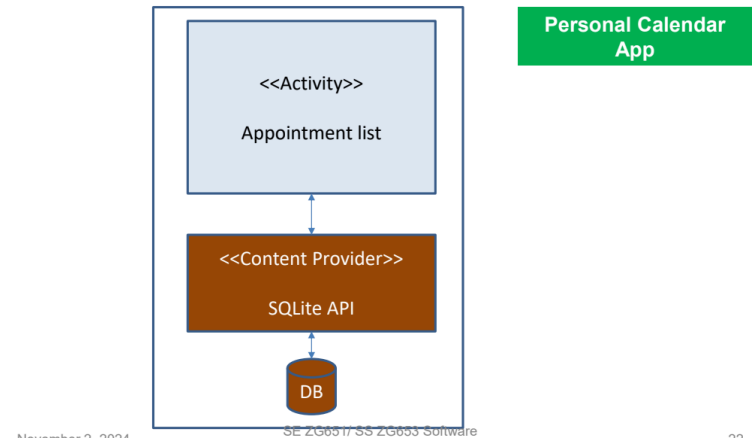
## Mobile Application Architecture

**Android application architecture**

4 types of components

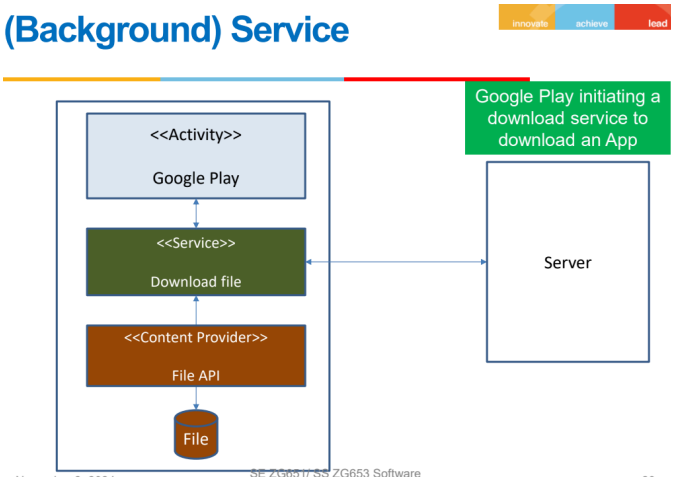
- Activity (UI)
- Service (background process) - ex. playing music, download
- Content provider (Storage) - ex. SQLite, files,
- Broadcast receiver (Acts on events received from OS and other apps) - Ex arrival of SMS

## Activity & Content providers

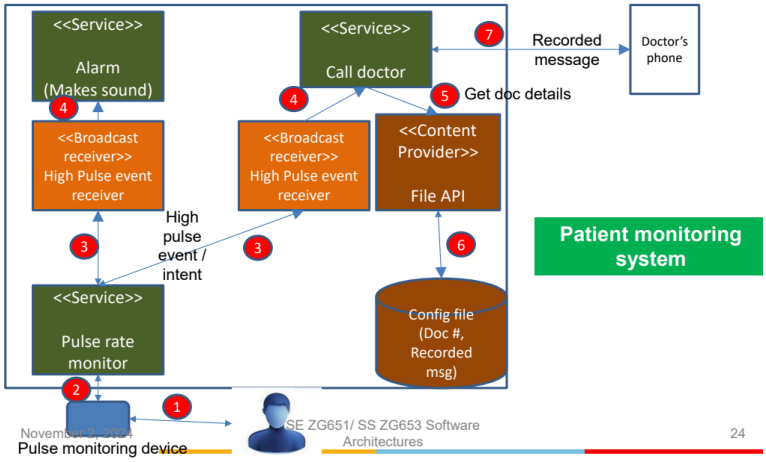





## (Background) Service

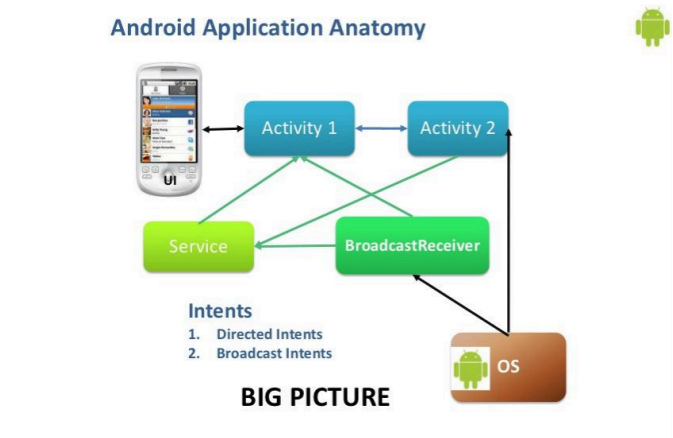


# Broadcast receiver (Event handlers)






# Android application structure




# Exercise

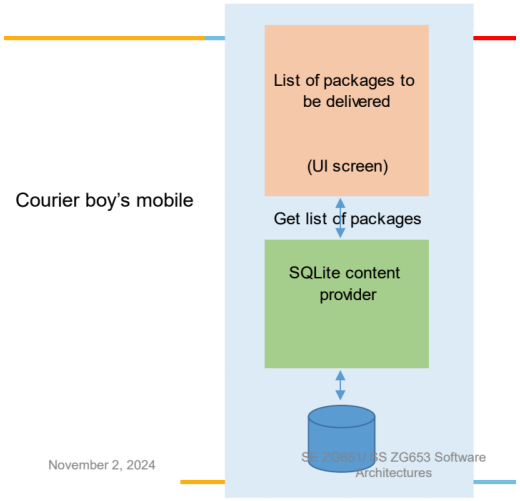
Consider a mobile app carried by the courier delivery boy.

The app should support the following functions:

a) View courier packages to be delivered
b) Mark a package as delivered.
c) Upon this event, the app should send information to central server

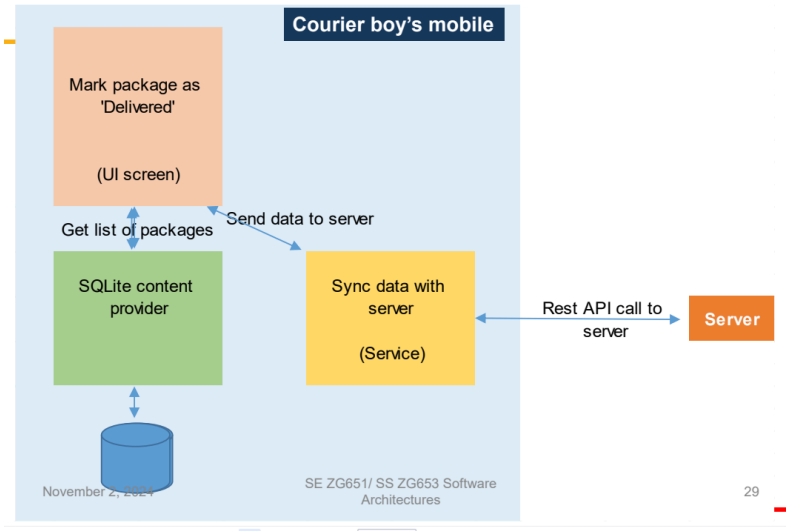
Identify the components of a mobile app & its inter-connections and draw an appropriate software architecture diagram

a) View courier packages to be delivered

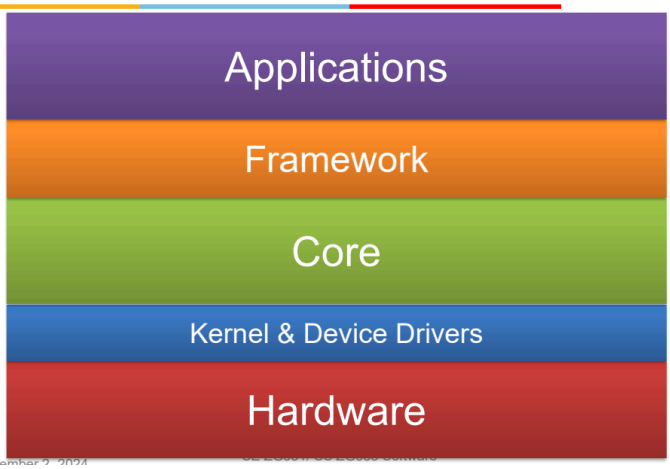





b) Mark a package as delivered. Upon this event, the app sends information to central server

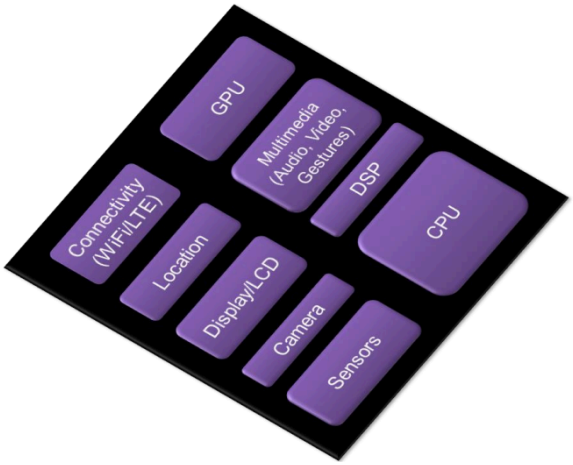


## Mobile - Software Platform

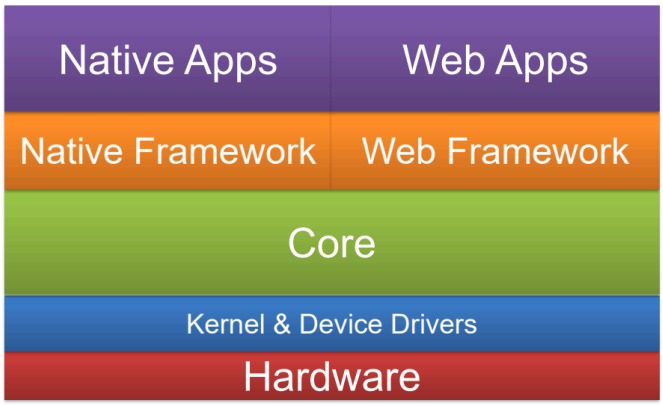





## Mobile – Hardware Platform



## Typical Software Platform

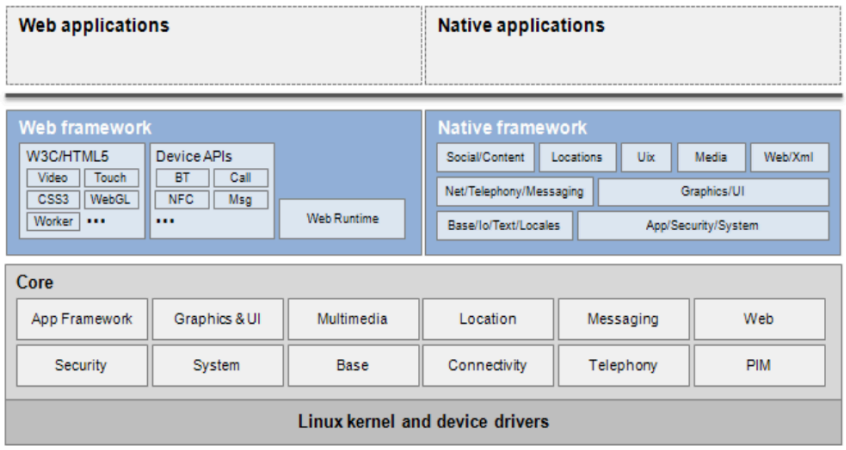



## Typical Software Platforms

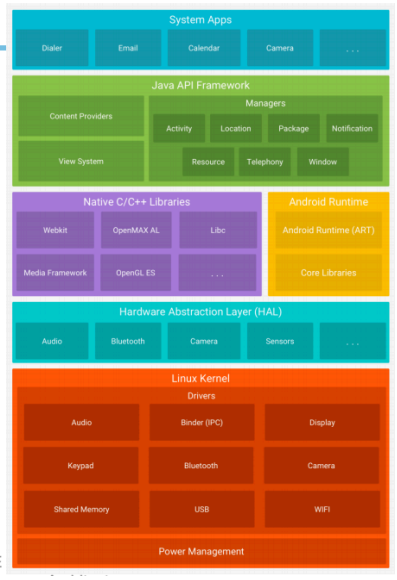
| | |
|---|---|
| iOS | • Mobile Operating System by Apple<br>• Multitasking<br>• Supports iPhone, iPad and other devices |
| Windows | • Windows Phone OS by Microsoft<br>• Multitasking<br>• Features Metro (Modern UI, Tiled UI) |
| Android | • Google's Mobile OS based on Linux Kernel<br>• Multitasking<br>• Software Stack |
| Tizen | • An open source standards based platform<br>• Based on Linux, part of the Linux Foundation<br>• OS with Multiple profiles |



## Tizen Architecture



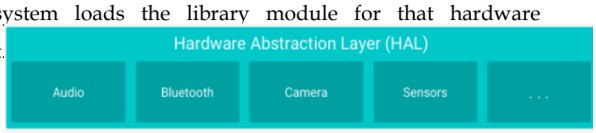
# Android Stack






# Hardware Abstraction Layer (HAL)

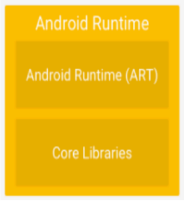
- HAL provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.
- The HAL consists of multiple library modules, each of which implements an interface for specific hardware components, such as the camera or BlueTooth module.
- When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.






# Android Runtime (ART).

- Each app runs in its own process and with its own instance of the Android Runtime (ART).
- ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed especially for Android that's optimized for minimal memory footprint.

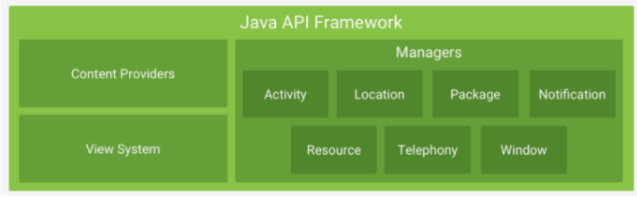



**Android Runtime**
- Include core libraries and the Dalvik VM
- Engine that powers the applications
- Forms the basis of the application framework
- Dalvik VM is a register-based VM to ensure the device can run multiple instances efficiently

## Native C/C++ Libraries


- Many core Android system components and services, such as ART and HAL, are built from native code that requires native libraries written in C and C++.
- The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps

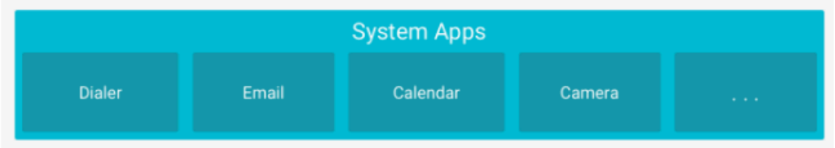
## Java API Framework



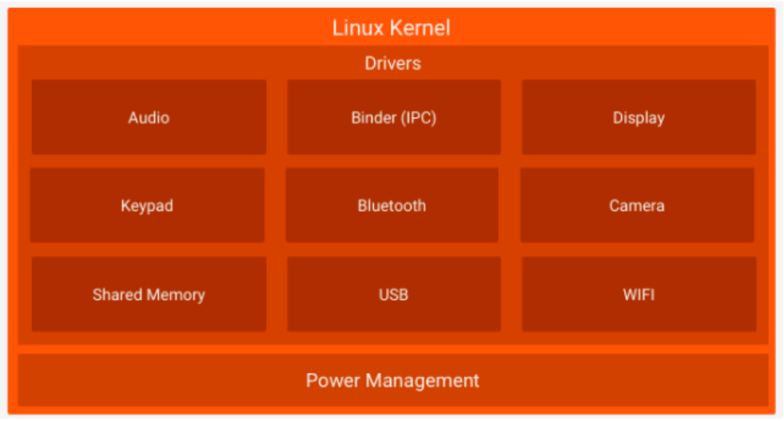

- The entire feature set of the Android OS is available to you through APIs written in the Java language.
- These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components, and services,

## System Apps

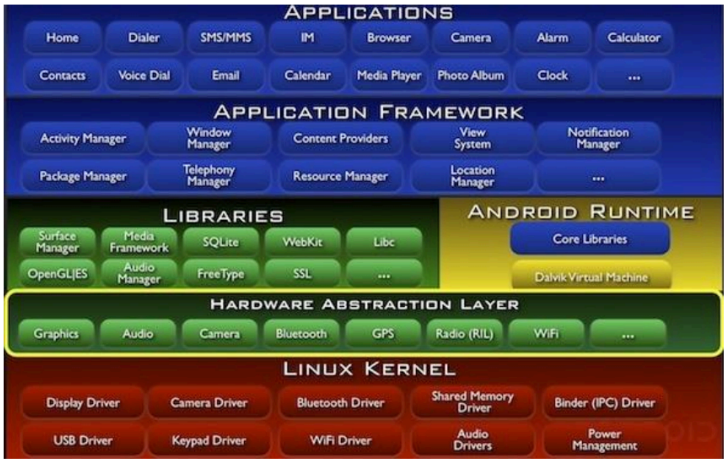
- Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more.
- Apps included with the platform have no special status among the apps the user chooses to install.
- So, a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions apply, such as the system's Settings app).



## Linux Kernel




## Android Architecture




## 10. Cloud-Based System Challenges

- **Availability**: Cloud providers ensure high uptime (e.g., 99.95%) but may not guarantee 100% availability.
- **Consistency vs. Availability**: Critical systems like e-commerce carts prioritize availability, while others, such as banking, prioritize consistency.

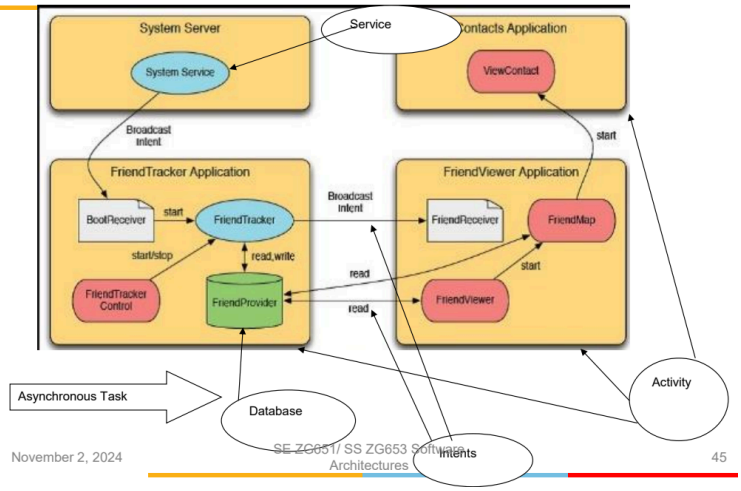
*Diagram Placeholder: Issues in Cloud-Based Systems*

# Popular services of cloud vendors

| Google App Engine | Microsoft Azure | Amazon Web Services |
|---|---|---|
| • Python & Java development environment<br>• Auto scaling<br>• Database replication | • .Net environment<br>• Auto scaling<br>• Load balancing<br>• Failure detection & auto replication of services<br>• Database replication | • Java development<br>• Auto scaling<br>• Load balancing<br>• Failure detection & auto replication of services<br>• Database replication<br>• Data caching (Memcached)<br>• Service discovery (SoA)<br>• Notification of events (SNS)<br>• Message queue (SQS)<br>• CDN (Content Delivery Network) – YouTube |



# Mobile app: Example



# Different ways to deploy applications on the cloud

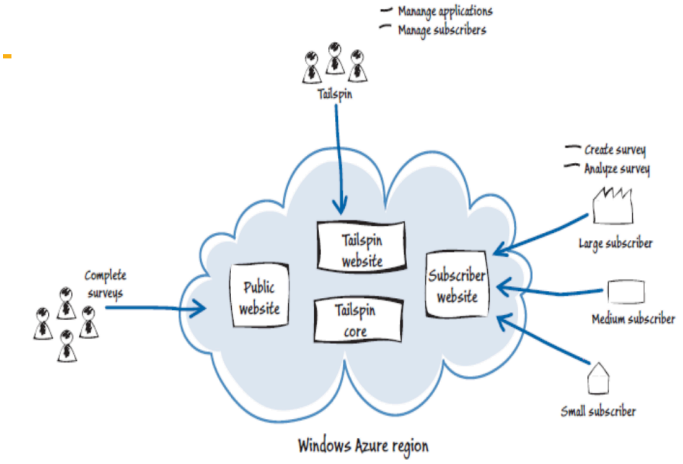
**Scenario**

A start-up company – Tailspin - is developing a **customer survey & analysis** application. This software will be deployed on the Cloud and offered to clients as a SaaS.

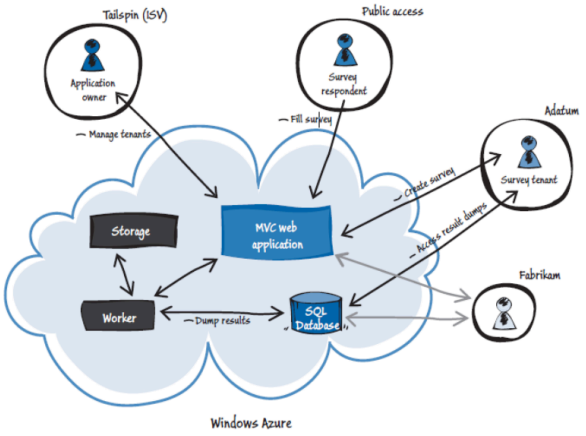
- The clients (subscriber of the application) can create and launch a survey.
- After this the survey participants will access the application and answer the survey questions.
- After the data has been gathered the application will perform analysis and present the results to the client organization

What options exist for Tailspin to deploy the application on the cloud?

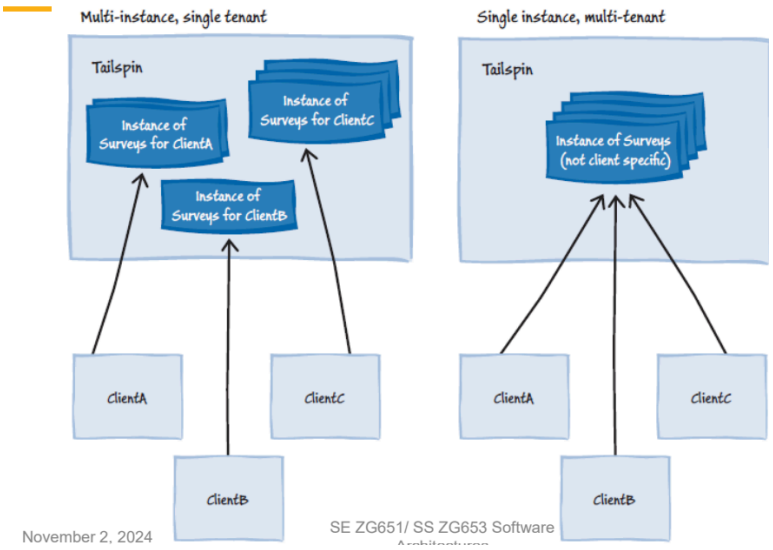
## Context diagram




# Multi-tenant application Architecture – High level



## Architecture Options – High level



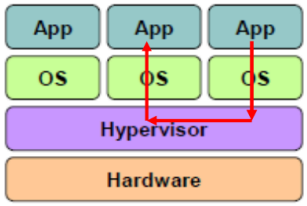



## Issues in Cloud based systems

- Security issue due to multi-tenancy
  - Poor design leading to inadvertent sharing of information
  - Virtual machine 'Escape'

| Other fields of the table | | | | | OU_ID |
|---|---|---|---|---|---|
| | | | | | Org1 |
| | | | | | Org1 |
| | | | | | Org1 |
| | | | | | Org2 |
| | | | | | Org2 |

Same table contains data of different organizations




Virtual machine 'Escape'

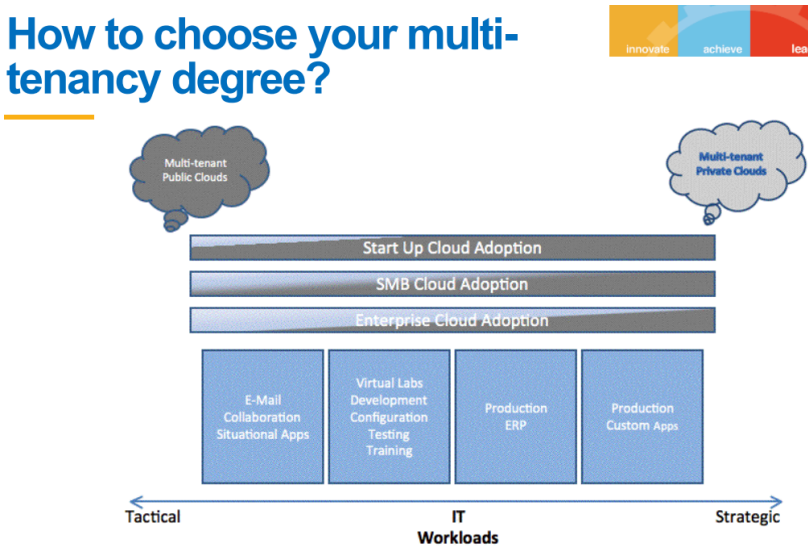
---

## 11. Multi-Tenancy and Cloud-Native Applications

**Multi-Tenancy**: Determines resource sharing in cloud applications. Higher multi-tenancy is suited for general-purpose applications like email, while lower multi-tenancy is better for data-sensitive applications.

**Cloud-Native**: Embraces modern development practices (PaaS, microservices, CI/CD, containers).

*Diagram Placeholder: Multi-Tenant Application Architecture*

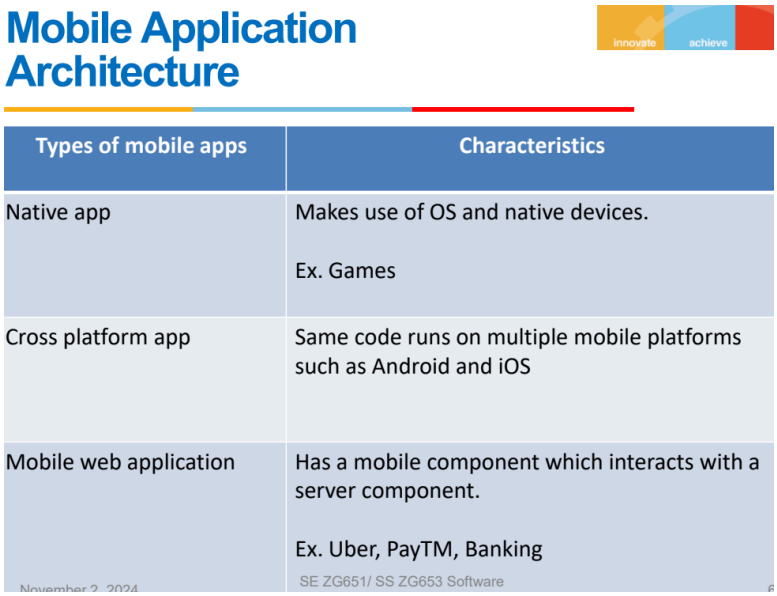



---

## 12. Android Software and Hardware Stack

- **HAL (Hardware Abstraction Layer)**: Interfaces hardware with higher-level APIs.
- **Android Runtime (ART)**: Each app runs in its own process, optimized for low-memory devices.
- **Native Libraries**: Core system components and services.
- **Java API Framework**: Building blocks for creating Android apps.

*Diagram Placeholder: Android Stack and Architecture*



## Tools and technology for mobile app development


| Development Approach | Native | Cross-Mobile Platforms | Mobile Web |
|---|---|---|---|
| Definition and Tools | Build the app using native frameworks:<br>- iPhone SDK<br>- Android SDK<br>- Windows Phone SDK | Build once, deploy on multiple platforms as native apps:<br>- RhoMobile<br>- Titanium Appcelerator<br>- PhoneGap<br>- Worklight<br>- Etc. | Build using web technologies:<br>- HTML5<br>- Sencha<br>- JQuery Mobile<br>- Etc. |
| Underlying Technology | - iPhone: Objective C<br>- Android: Java<br>- Windows Phone: .NET | - RhoMobile: Ruby on Rails<br>- Appcelerator: Javascript, HTML<br>- PhoneGap: Javascript, HTML<br>- Worklight: Javascript, HTML | - Javascript, HTML |

---

## 13. Example of Cloud Service Providers and Their Offerings

- **Amazon Web Services (AWS)**: Offers features like auto-scaling, load balancing, and data caching.
- **Google App Engine**: Provides Java and Python environments with auto-scaling.
- **Microsoft Azure**: .NET environment with failure detection, load balancing, and database replication.

---

## 14. Exercises and Review Questions

1. **Mobile App Exercise**: List components for a courier delivery app, with features like viewing deliveries, marking deliveries, and syncing with the server.
2. **Questions**:
    - What is a cross-platform app?
    - How is data consistency maintained with poor connectivity?
    - What is a Broadcast Receiver in Android?

---

## 15. Scenarios for Multi-Tenancy in Cloud Applications

- **Example**:
    - App 1: Multi-tenant web and data tiers, single-tenant application tier (high availability).

- App 2: Single-tenant web and data tiers, multi-tenant application tier (data sensitivity).

---

## 16. Conclusion

This document outlines the architecture of mobile applications, covering types of apps, UI/UX design principles, Android architecture, cloud integration, and challenges in cloud-based systems.