Sistemas de Computação Móvel e Ubíqua 2020/2021

2021/2022

Mobile Application Development

2021/2022

Outline

Mobile OS and platforms

Native and web-based applications
- Progressive web apps

Paradigms
- Client/server
- Peer-to-peer
- Offline-first, Local-first

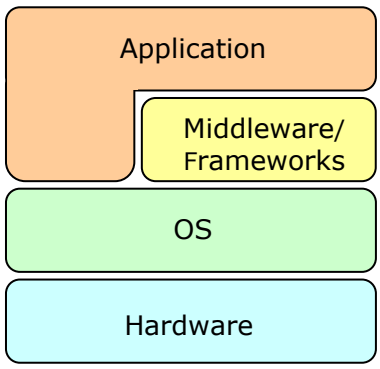
Platforms
- MBaaS

Software stack

In mobile devices, OS tend to be specific for different types of devices.

The operating system manages the resources of the devices and provides interfaces to access such devices.

Examples:
- Amazon FreeRTOS – used in IoT devices
- Embedded Linux (also used in Android)
- Embedded Darwin (used in iOS devices)
- Windows 10 IoT Platform
- …



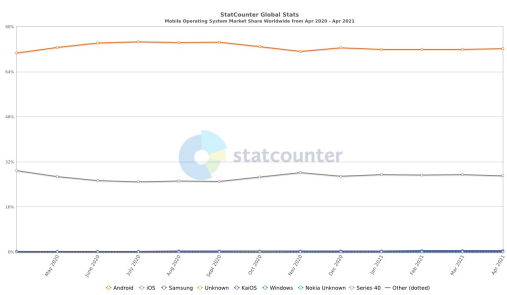
OS for smartphones/tablets

Market dominated by two OS:

- Android/Linux (~74%)
- iOS (~25%)

The operating system includes:

- A kernel managing device's resources
- Frameworks and services supporting user applications



OS kernel

Kernel includes traditional OS functionalities and more

- Process and thread management
- Memory management
- Storage
  - File management
  - SQL databases
  - HTML5 storage options
- Controls a full range of internal sensors
  - Camera, GPS, microphone, compass, accelerometer, gyroscope, etc.

OS mobile platforms

Software platforms are implemented in libraries used for application development

In small embedded devices are very small and specific (like for the sensor networks)

For smartphones and tablets (also tv boxes, watches, …) the OS platform can be an extensive library of frameworks and middleware

Application development

Embedded

Applications are specific to the device and are integrated with the operating system – used in embedded devices, sensor networks, etc.

Stand-alone

Standalone native application built on top of a platform and an operating system

Web-technology based

Applications run on the web browser or using web-based technologies.

iOS Platform

Cocoa Touch
- UI components

Media
- Audio, video related functionalities

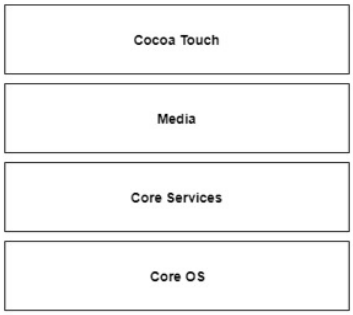
Core Services

Gives access to fundamental resources needed for apps

Networking, iCloud, Encryption, SQLite

GPS, Telephony, SMS

Core OS
- Kernel operations



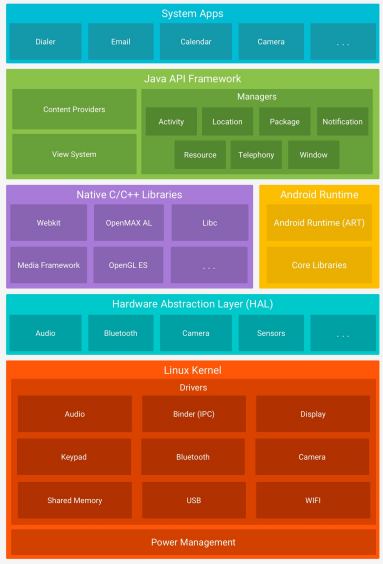
Apple iOS Architecture

Android Platform

Hardware Abstraction Layer
- Provides standard interfaces that expose device hardware capabilities to Java API framework.

Android Runtime
- Provides most of the functionalities available in the core libraries of the Java language. Each application run its runtime.

Native C/C++ Libraries
- Core components written in C/C++ for efficiency.

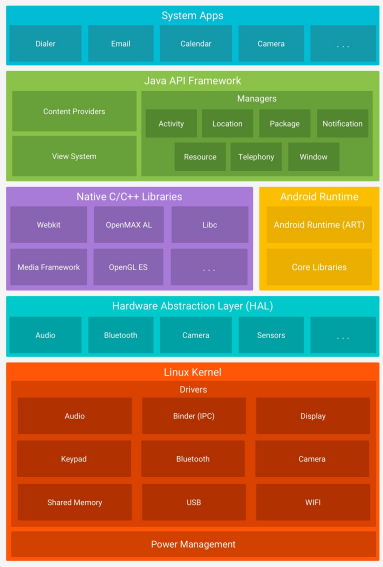


https://developer.android.com/guide/platform/

Android Platform

Java API Framework

- The entire feature-set exposed using a Java API
- **View System**, an extensible set of views used to create application user interfaces.
- **Resource Manager**, provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notification Manager**, enables all apps to display custom alerts in the status bar.
- **Activity Manager**, manages the lifecycle of apps.
- **Content Providers**, enable apps to access data from other apps, such as the Contacts app, or to share their own data.
- **Telephony Manager**, responsible for handling the settings of network connection and all information about network/telecom services on device.



https://developer.android.com/guide/platform/

Device heterogeneity

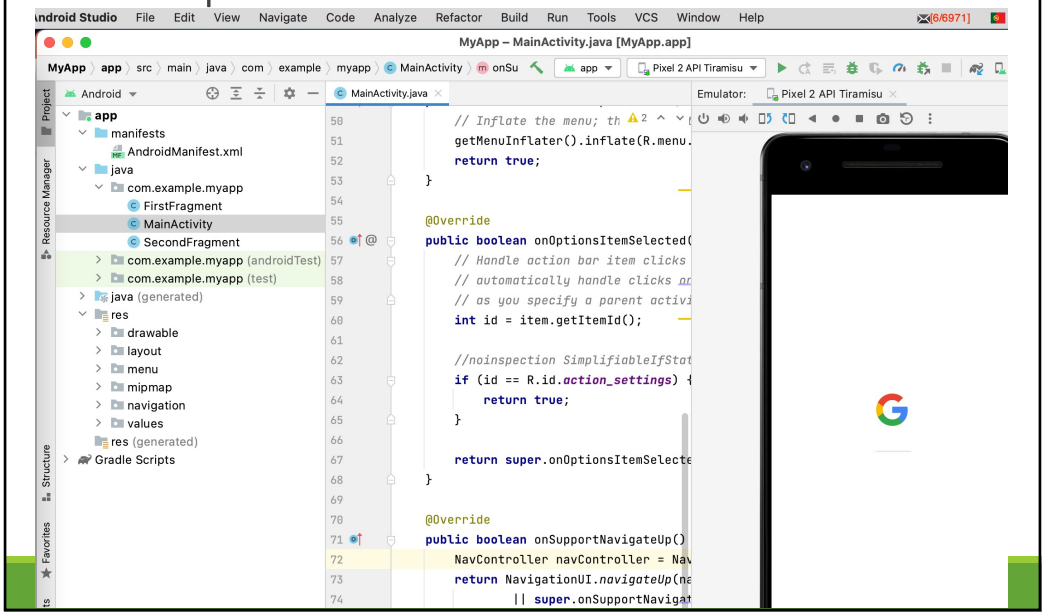
Huge device variants

- Hardware characteristics
  - Screen size and density (resolution)
  - Storage, network, memory, sensors, etc
- Languages and cultures
- User accessibility and preferences (eg. Impaired users, GUI themes)
- Security and permissions

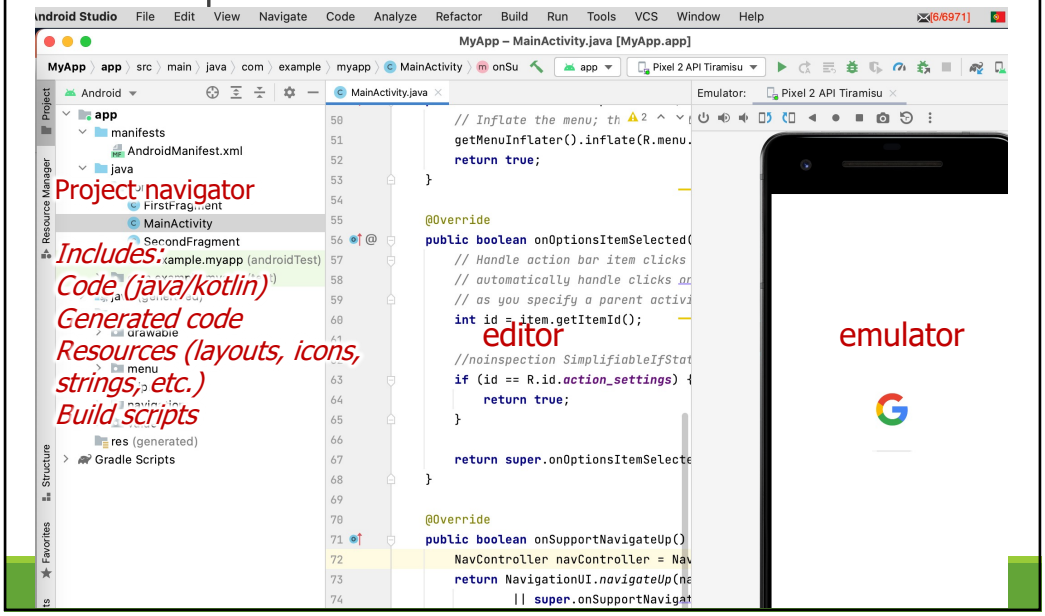
Application development must deal with heterogeneity

- Code deals with application logic
- UI design and it's elements described by resource files (eg. Icons, strings)
- Some code generated by build system

Example: Android Studio IDE



Example: Android Studio IDE



Web-based development for mobile

Traditional web technologies: HTTP, HTML, CSS, JavaScript, Ajax

Positive aspects

- Developers can use normal Web authoring tools
- May be easier to learn
- Can be develop in any platform for any platform.
- Can fix bugs in "real time"

Negative aspects

Lower quality interfaces, responsiveness, etc.
Can impose bigger overheads and resource consumption
Limited access to hardware (e.g. sensors, storage) and other services

Mobile Web applications

Pros

- Portability for a wider range of heterogeneous devices.
- Easy distribution (e.g. web-site)

Cons

- May look as a site, not as an application
- More limited GUI and foreigner look & feel
- Limited and slower access to hardware (addressed partially with HTML5 APIs).
- You may need to be connected.

Native development

Pros

- Full support of all the platform and hardware features of the device
- Can achieve best performance and better resource usage
- Applications are one click away from potential users (through application stores)
- Better documentation and IDEs for application development

Cons

- Steeper learning curve
- May need to pay to become a registered developer and use the application store.
- Your app must be approved as well as the bugs fixed
  - (an advantage for the users)

From web-based to native

Large number of solutions for improving web based applications:

(PWA) Progressive Web Applications (HTML5, CSS, JavaScript)

Standalone Web application frameworks with native support in the runtime

- Examples: Apache's Cordova, Facebook's React-native, Google's Flutter/Dart

Objectives

- Address existing limitations (improve performance and hardware access)
- Allow multiplatform/OS development

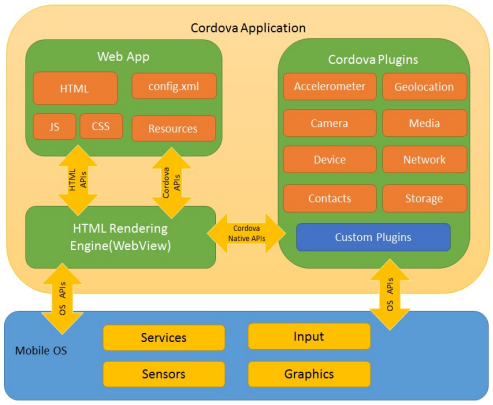
Example: Apache Cordova

Application developed using Web technologies: HTML, CSS, JavaScript.

Plugins provide access to the native components of the mobile OS.

WebView used to render user interface.

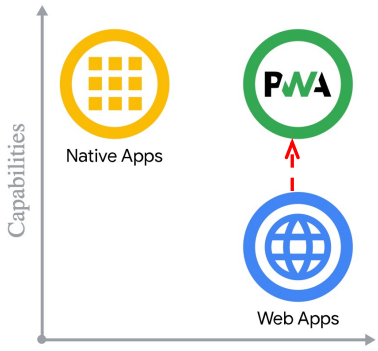
Generate code for running in Android, iOS or Web-browser



Progressive Web App (PWA)

A progressive web app (PWA) is a browser-based application that tries to look like a native application.

Builds largely on HTML 5.



https://web.dev/what-are-pwas/
https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps#core_pwa_guides
https://www.scitepress.org/Papers/2017/63537/63537.pdf
https://scholarspace.manoa.hawaii.edu/bitstream/10125/50607/1/paper0720.pdf

PWA: key principles

Progressive - progressive enhancement strategy to create cross-platform web applications usable on older browsers, but fully-functional on the latest ones.

Discoverable - contents can be found in sites through search engines.

Installable - available on the device's home screen

Linkable - share it by simply sending a URL.

Network independent - works offline or with a poor network connection.

PWA: key principles (2)

Re-engageable - able to send notifications whenever there's new content available.

Responsive - usable on any device with a screen and a browser (mobile phones, tablets, laptops, TVs, fridges, etc…)

Safe - connection between you and the app is secured against any third parties trying to get access to your sensitive data.

Browser API key features: Storage

Cache API
- System for storing previous network requests and their corresponding responses

PWAs can store data persistently (across restarts):

IndexedDB
- NOSQL database.
- Store key-value pairs. Values can be JSON objects.
- Support for indices, transactions.

LocalStorage
- Allows to store key/value pairs, but limited in size.

Browser API key features: service workers

Service workers provide support for executing background tasks, that do not necessarily need a web page or user interaction.

Service workers can intercept and handle network requests, allowing asynchronous communication with a server.

Browser API key features: notifications

PWA build on service workers to support push notifications from servers

- allows applications to subscribe and be notified when some interesting event occurs - e.g. a new message is available in a messaging application

PWA can notify the user by presenting a notification in the mobile phone.

These functionalities are supported even when the PWA is not running

Outline

Mobile OS and platforms

Native and web-based applications

- Progressive web apps

Paradigms

- Client/server
- Peer-to-peer
- Offline-first, Local-first

Platforms

- MBaaS

Architectures

While some applications are standalone and can run in a single mobile device, most applications are distributed and composed by multiple components.

Architecture: client/server

Client/server

An application is divided in two types of components:

- **Server**: typically running in a server, often located in a cloud platform.
- **Clients**: typically running in mobile devices. A client communicates with the server.

Most used architecture, due to its simplicity.

Can support most applications.

Mobile cloud computing

With the advent of cloud computing, many proposed to move all data storage and data processing outside of mobile devices.

This is often called the **thin client** approach, where the client application is mostly the user interface.

Motivations for this proposal:

- Limited resources: storage, processing, battery, etc.
- Unsecure storage at mobile devices: if a device is lost, data may be compromised or "lost".

Main problems:

- Network bandwidth and latency

Mobile cloud computing

With the advent of cloud computing, many proposed to move all data storage and data processing outside of mobile devices.

This is often called the **thin client** approach, where the client application is mostly the user interface.

Motivations for this proposal:

- Limited resources: storage, processing, battery, etc.
- Unsecure storage at mobile devices: if a device is lost, data may be compromised or "lost".

Main problems:

- Network bandwidth and latency

Smartphones have more computing recources than before.

Encryption-at-rest (data is stored encrypted on disks/flash) overcomes the problem of data compromission.

Architecture: peer-to-peer

Peer-to-peer

An application/system is composed by multiple peers that communicate directly.

Typically, more complex to implement. Challenges:

- Discovery of other peers;
- Often, peers cannot communicate directly due to network restrictions (private networks, NATs, etc.) .

Appropriate for applications where users interact directly – e.g. collaborative applications, games.

Direct communication might be faster than routing interactions between users through a server.

Some Web supporting technologies

WebRTC is a real-time communication protocol for Web applications.

Designed initially to allow browsers to establish peer-to-peer audio/video calls.

Can be used for general data communication.

A key challenge is that often clients cannot communicate directly, due to network restrictions (private addresses, NAT, firewall, etc.)

- Other protocols for dealing with those restrictions, like STUN and TURN

Offline first - design approach

Key principle: applications should be designed to work without connectivity (from cache and local storage)

- Then make use of network when available

Motivation: users experience variable connectivity, due to:

- Network coverage varies in different places
- Network coverage may be unavailable in certain places in a building
- Bandwidth varies widely (wifi, 5G to 3G)

http://offlinefirst.org/

Caching

For supporting offline-first, applications need to cache data that users will need before they need it.

- This process is highly application-specific.
- Examples: in a maps application, cache the area where the user is (and where the users is supposed to move).

All or some functionality while offline

An application should provide functionality even while it is running offline.

- Connectivity faults should not be treated as errors!

This includes supporting functionalities that change the state of the application/system.

- A user should be able to write emails while offline…

Support for conflict handling

Allowing users to change the state of the application/system while offline may lead to conflicting updates.

- A conflicting update occurs when two users concurrently change the same data item. E.g. two users reserving the same room.

Possible conflict resolution policies:

Last write wins : e.g. keep the last update to an entry in a contact.

First write wins : e.g. first writer to a calendar entry should win.

Merge : merge concurrent updates in a smart way.

- E.g. It two persons add different items to a shopping basket, keep both. If an husband and a wife concurrently change the number of item to buy regarding the same product, keep the wife's value.

[more on next lecture]

Seamless transitions from online/offline

The system should smoothly transition from online to offline and vice-versa.

Updates/communication to servers should be queued while offline and dispatched when connectivity exists.

- Asynchronous communication model or event-based model.

Applications can receive communications from servers using notification mechanisms (or publish/subscribe platforms).

Local-first for user data

Extends the idea of offline-first by advocating that:

- a system should not include servers
- or servers will be used only for backup or routing opaque information between peers

Primary data storage is in the application device!

https://martin.kleppmann.com/papers/local-first.pdf

Key ideas

Frontend/backend architecture with local-first approach for data

Key ideas:

- No Spinners: Your Work at Your Fingertips
- Your Work Is Not Trapped on One Device
- The Network Is Optional
- Seamless Collaboration with Your Colleagues
- The Long Now
- Security and Privacy by Default
- You Retain Ultimate Ownership and Control

No Spinners: Your Work at Your Fingertips

Avoid the time needed to load data from the server by holding the primary copy of the data on the local device.

All operations should be able to execute by accessing the local storage – to load and store data.

Your Work Is Not Trapped on One Device

Most users have multiple devices – your data should be accessible in every device.

Requires support for data to be synchronized across all of the devices on which a user does their work.

A large number of cross-device sync services exist, and some also store a copy of the data on a server, which provides a convenient off-site backup for the data.

The challenge with existing solution is typically that they provide poor support for handling concurrent updates.

The Network Is Optional

Applications should work without network connectivity.

The multiple devices of a user should be able to synchronize even with Internet connectivity, by using peer-to-peer communication, such as Bluetooth or Wi-fi direct.

Seamless Collaboration with Your Colleagues

Need to allow user to make contribution and support conflict resolution.

"Achieving this goal is one of the biggest challenges in realizing local-first software,"

The Long Now

"An important aspect of data ownership is that you can continue accessing the data for a long time in the future."

E.g. what do you do to all that data, music that you have on CDs?

Local-first software should be able to continue working for a long time.

Security and Privacy by Default

Local-first apps have privacy and security built in at the core.

Local devices store only the user's own data, avoiding the centralized cloud database holding everybody's data.

Local-first apps can use end-to-end encryption so that any servers that store a copy of your files only hold encrypted data that they cannot read.

You Retain Ultimate Ownership and Control

The user should be able to copy and modify data in any way, write down any thought, and no company should restrict what the user is allowed to do.

As data is in the user's, she has the freedom to process this data in arbitrary ways.

Outline

Mobile OS and platforms

Native and web-based applications
- Progressive web apps

Paradigms
- Client/server
- Peer-to-peer
- Offline-first, Local-first

Platforms
- MBaaS

Frontend/Backend

Application components are usually divided in frontend and backend. No precise definition, but:

Frontend is related to the components that the users interacts with.
- User-facing application – web, native mobile.

Backend is related to the components that manage application data, with which the user does not interact directly.
- Databases, file systems, cache server, notification servers, ML servers, etc.
- Some can be cached in the interface device (not always possible)

Mobile Backend as a Service

The goal of a Mobile Backend as a Service is to provide **web and mobile app** developers with a set of services to simplify the development of applications.

Common services include:

- Authentication;
- Storage and synchronization;
- Push notifications;
- Analytics and Machine Learning services;
- Etc.

Mobile Backend as a Service

Examples:

Parse

(Google) Firebase

(Azure) Mobile services

(AWS) Amplify

And many more.

Example: Firebase

Designed for supporting web and mobile applications.

Founded in 2011.

Initial product was backend so websites could easily host chat as part of sites.

Discovered that developers were sending non chat data (such as game state) via the tool.

Bought by Google in 2014.

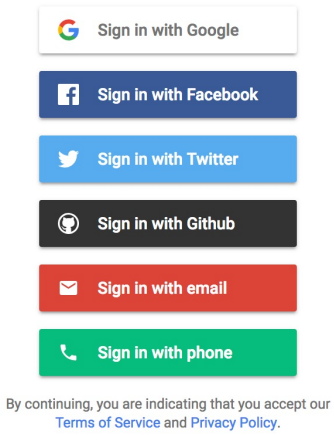
Includes several services…

Firebase authentication

Supports authentication for application users.

Email and password based authentication.

Allows authentication using third-party identity providers: Google, Apple, Facebook, Twitter, Github.



Cloud storage

Cloud Storage is designed to store and serve user-generated content, such as photos and videos.

API allows to upload and download files directly from clients. Handles poor connectivity, with operation retries and continuation.

Data stored at Google cloud storage. Allows to integrate server-side processing such as image filtering or video transcoding.

- Transcoding is the process of transforming data typically to save bandwidth: compression, image size transformation, etc.

Firebase Realtime Database

Store and sync data with our NoSQL cloud database. Data is synced across all clients in realtime, and remains available when your app goes offline.

Data model: JSON object.

Cloud-hosted database.

Clients' devices keep a copy of the database. Application access and modify the local copy. Can access the database while offline.

Data is synchronized in realtime to every connected client.

Firebase ML

Firebase included analytics integration for obtaining insight on app usage and user engagement.

Firebase Machine Learning allows to integrate ML features in applications.

- Training always performed on the cloud.
- Inference can be performed on the mobile device or on the cloud.

Designed to be used both by new and experienced developers.

Firebase ML

Includes pre-trained models for:

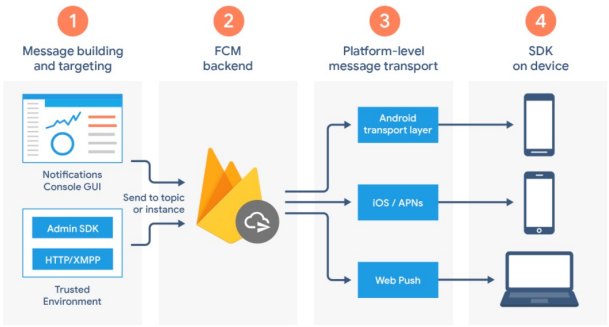
- Text recognition
- Image labelling
- Object detection and tracking
- Face detection and contour tracing
- Barcode scanning
- Language identification
- Translation
- Smart Reply

Recognized Text	
Text	`Wege der parlamentarischen Demokratie`
Bounding Box / Frame	`Rect` (Android) or `CGRect` (iOS)
Blocks	(1 block)

Firebase Cloud Messaging

Firebase Cloud Messaging a reliable connection between servers and devices, that allow to deliver and receive messages and notification to mobile devices and web apps.

Possible to implement location-based delivery (with additional libraries).



Bibliography

Previous links and…

Frank Adelstein, et. al. Fundamentals of Mobile and Pervasive Computing. McGraw-Hill. 2005. Chap 7.

Jean Dollimore, Tim Kindberg, George Coulouris. Distributed Systems: Concepts and Design (5th Edition). Addison Wesley. Chap 6.

https://www.appinf.com/docs/poco/00100-UPnPOverview.html