

Mobile Programming and Multimedia Simple (for real)



Gabriel Rovesti

Table of Contents

[2 Course Introduction 3](#_Toc161396490)

[3 Introduction to Mobile Development 6](#_Toc161396491)

[4 Frameworks for Cross-platform Development 9](#_Toc161396492)

[5 Exercise 1 – Cross-platform development frameworks 22](#_Toc161396493)

[6 Flutter Framework 23](#_Toc161396494)

[7 React Native Framework 31](#_Toc161396495)

[8 Store Deployment 36](#_Toc161396496)

[8.1 Google Play Store 36](#_Toc161396497)

[8.2 Apple Store 39](#_Toc161396498)

[9 iOS Platform 40](#_Toc161396499)

[10 Android Platform 41](#_Toc161396500)

[11 Mobile Design 42](#_Toc161396501)

[12 Wearable Devices 43](#_Toc161396502)

[13 Multimedia Data Encoding 44](#_Toc161396503)

[14 Images 45](#_Toc161396504)

[15 Audio 46](#_Toc161396505)

[16 Video 47](#_Toc161396506)

**Disclaimer**

The course is divided into two parts: Mobile Programming and Multimedia, hence the name. The contents are pretty interesting, and the course is interactive and good.

Frameworks are PhoneGap, Corona/Solar 2D, Xamarin are considered older hence not part of official program.

The course contents are presented chronologically, so you can easily understand just by opening this file *everything* about timing and content (important, I know!).

# Course Introduction

The course is based on two parts:

* *Mobile Programming*
  + this is the section that will change, given the nature of it
  + this year we will get to the detail of design of applications
* *Multimedia and Data*
  + this remains pretty much the same
  + this part is focused on understanding how to encode data
    - media can be particularly heavy on memory/battery/bandwidth usage
    - we want to save time in downloading and streaming the files
    - this of course because usage is very much mobile and not desktop anymore

Other general info:

* Slides will be gradually updated according to the need on the Moodle, which is free to access
  + There will be 4x4 (4 slides per page) and normal slides (1 per page)
* After 15 days, data will be collected in order to get account data of labs
* In case of need, an Apple PC is available in the library to be used to develop applications
* Recordings are available but only if attendance doesn’t go down

Quality of the application is measuring “how much it does what it’s supposed to do”. While in Web is important the number of clicks to get to the data, here it’s important how much data is asked to the user.

When creating an application, it’s important also to drive metaphors towards a common goal, giving the everyday use inside what is done inside of a product/application. Consider the desktop, which seems not a metaphor, but it is, actually.

When we are developing an application, consider this is done for smartphone, which do not have OS, but RTOS (Real-Time Operating System): for example, when we receive phone calls, an applications stops, has to save save state, then we get back to what we were doing before.

Immagine che contiene mammifero, grande felino, tigre, Grandi felini

Descrizione generata automaticamenteThe following is an example of compression; the difference is present but not very much:

To understand this, we will understand what we can or cannot perceive in light and image. This way, we can remove all the information we can’t see, without losing quality. It’s more important to know what the formulas do, instead of understanding for real the elaborations.

Representation and encoding revolves around:

* Immagine che contiene fiore, Viso umano, illustrazione, schizzo

  Descrizione generata automaticamenteHow human sight works
* Image properties: size, quality, transmission, visualization
* File formats
  + GIF
  + PNG
  + JPEG
  + JPEG2000
  + others

Also, it’s important to understand how *sound* is made:

* Audio properties: fidelity, ì transmission, playback
* Standard file formats:
  + WAV
  + MP3
  + others
* MIDI
* Compression (lossy – with loss of some data)
  + understanding what to perceive and what not

But also images with audio - *video*:

* Human vision with motion pictures
* Digital and analogic video
* Video properties: quality, representation, transmission
* Standard formats:
  + H261
  + H263
  + MPEG family
  + DivX, Xvid

We also talk about *data compression*, in particular:

* Reasons behind data compression
  + Storage space, transmission time
* Continuous and not-continuous media compression
* Lossless and lossy compression
* Lossless encoding
  + Entropy encoding methods
  + Semantic compression
* Lossy encoding
  + Image compression: JPEG
  + Video compression: MPEG1-2
  + Audio compression: MP3

Prof. says slides are not enough for the examination; Moodle material is suggested and also textbooks are needed here.

Fragmentation is present between platforms (Apple vs Android and all of its versions) and different values of settings according to devices and specific needs (e.g., brightness, virtual environment, singular devices with particular sensors/features, etc.).

This is the knowledge and skills targeted from the course:

* Mobile interface design
* Cross-platform development
* Emotional design
  + transform the user into a customer
* Wearable devices
* Market

The examination will require:

* A presentation of an argument with slides plus an oral examination

Or:

* A small group project with a final report plus a small oral examination
  + Develop an application
  + It should be made for all platforms
  + But it’s not mandatory, can be only for one specific system
  + Should be sent 5 days before the examination
  + Subscribed on Uniweb and poll of what examination was chosen
* In-depth analysis of an argument with a presentation and a small oral examination
  + 10-page essay presented on the end of the course
  + The argument must be defined by the end of April
  + Possible dates: 4th – 7th June 2024
  + It’s very important to choose topics and prepare/explain material seen during lectures

In any case, oral examination is made on two questions about all the class program and material. One can avoid these two oral questions if one attends in presence only the homeworks (exercises). These can be something like “solve a multimedia algorithm problem” or “understanding design flaws of something and explain it to the class”. If all are delivered (or at most you miss one – but at least half or more of the half), then this is considered.

Exercises will be evaluated anyway even if wrong and can be given ½ points, then asked in oral examination, when given feedback of course. It seems they will be on Tuesday afternoon.

# Introduction to Mobile Development

To create the right product, between web and mobile, we have to *study the user and understand his necessities and needs*. As we all know by now, smartphone market has been exploding since years and more and more users are active using a mobile device.

Some data we can give about them:

* Over 5 billion people are using smartphones, with 4/5 billions being active social media users/Internet users respectively
* There are at least 7 billion of mobile subscriptions worldwide
* China, India, and the United States are the countries with the highest number of smartphone users, but also Indonesia, Brazil, South America and Africa
* While desktop is mostly used inside USA, South America, Oceania
  + connections can be wireless easily not needing an infrastructure
* Operating systems are Android and iOS
  + who wants to create premium services or something that people will buy, the preferred choice is actually iOS
  + on Android we have a lot of fragmentation between features and various things
* Worldwide users are on smartphones and usage data is collected from developers themselves
  + this does not mean we have to forget desktop however
  + consider mobile devices are also tablets, not only smartphones
    - they weigh more, they cost more, we use them with two hands, and we sit down using them concentrating using it
    - this way the application must come with some way to handle the error situation, recovering from them
* There are differences between males and females
  + females use more mobile apps than desktop
* Smartphones beat TV for younger users (the younger, the more usage)
  + less gestures required, less fatigue, more content present
* There is a relationship daytime-device
  + low-to-middle use between morning and daytime for mobile
  + daytime to early evening for desktop
  + in evenings for tablets
* On average there are more than 2000 interactions with a smartphone on a day
  + consider the user can make a lot of errors because there are a lot of interruptions
    - good quality means good experience, even when errors happen
* Today we have smart\* (smartphones, smart watches, smart homes, etc.)

Mobile phones are not considered anymore as a simple device to make calls but incorporate a lot of different features. All of these ones are provided by apps, in whatever form.

* Messages, calls
* Internet navigation
* Sensor data collection and usage (app for training, biking, running, etc.)
* Agenda
* Entertainment (games, music, video, reading, etc.)

There are different *false myths*:

* Mobile app development is not expensive
  + A bad app is worse than no app
* Mobile app development is easy

On the contrary:

* Mobile app development requires big teams
* Mobile app development is not like winning at the lottery

The first step to determine if it is better to develop a mobile layout of your own website or a mobile application is to understand the differences between the two:

* Diversified content 🡪 content will be personalized remembering user preferences
* Native interface vs. Company brand 🡪 more opportunities to better meet user needs
* Development time 🡪 depends on the needed for the goals to achieve
* User interaction (ex: push notification)
  + Using gestures instead of point and click
  + User experience improves
* Access (icon) vs bookmark 🡪 the icon remembers you to use the app
* Target (loyalty vs. reach) 🡪 ease of access/personalized experiences/convenience

The only data we need from users comes from payment information, all the rest is needed *because the developer asks for it*. The website is the best way to get information in a quick way, apart from push notifications. Also, icons are suitable to do that the best way, because it *remembers the user what to do*.

Remember also *mobile e-commerce* is going strong and has a greater market share each time.

* Usually, transactions are simultaneous and can happen on multiple devices at a time and also multiple apps at the same time, outperforming with apps mobile browsers
  + bringing an environment together (without having to put all data again)
* Committed retailers capture more transactions on mobile apps rather than browsers
* In both platforms, iOS devices capture the majority of these retailers’ transactions
  + they will definitely spend more money

Other general statistics:

* There are millions of apps in the stores and a 25% of them are used only once
* A user usually spend 90% of the smartphone time using apps
* The 84% of the time is spent using 5 apps that change between users
  + which include, in this order, social apps, games, music, and video streaming
* It does not matter the number of downloads in the end, but the number of installations
* Study the user remember: screen time depends on different factors but also context

There is the *app vs mobile web*:

* A mobile application usually tends to encourage brand fidelity (icon on the desktop, notifications, etc.)
* A website with a mobile layout allows reaching the user in every situation, immediately

There are situations when it’s useful to create an app:

* A lot of graphics or calculations
  + Leverage device resources for better user experience
* Camera, sensors, or microphone usage
  + Audio/image processing, sensor data analysis, not feasible on web
* Gallery or contacts' access
  + Seamless user interaction
* Push notifications or background service
  + Data synchronization, location tracking, content downloading and convenience
* For games
* It is the only way to have access to the store 🡪 maximizing reach and revenue

The number of installed apps changes depending on the device (and its operating system). According to Nielsen, the best approach is to interview the users to understand if they would accept to install the new app on their device:

* Storage space
* Purpose
* Loyalty

The development of a mobile app requires several *resources*:

* Interface design
* Development
  + E.g., Which operating system?
* Maintenance

There are different *advantages in web apps*:

* They require a very low knowledge base, HTML is popular
* HTML5 now provides access to almost all smartphone features
* More straightforward “conversion” to different operating systems
* User *does not have to worry about the update* of the application
* It is not necessary to wait for application approval
  + Apple can require more than 2 weeks

Applications for mobile devices are different from desktop applications:

* Mobile operating systems are soft real-time operating systems:
  + An application can be suspended or terminated in every moment
  + The operating system manages context switch
* Only one application active
  + Not with iOS on iPad
* Limited space, it is not possible to open more windows at the same time
* Easy to install (or at least discourage less the use)
* Incredibly high number
  + To design and create an exciting app is extremely challenging
* Market fragmentation

Bibliography of this chapter: [here](https://www.business.com/articles/mobile-apps-vs-mobile-web-do-you-have-to-choose/) and [here](https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/).

# Frameworks for Cross-platform Development

Once upon a time, whenever there was an idea for a new app, the main goal was to develop it for iOS.

* On iOS there are users that spend the most
* On Android there is the highest diffusion of users
  + Before 2010, there was still choice in OS other than these two
* To have a lot of income, it’s important to develop for both platforms
* In Android, there is a lot of fragmentation between devices
  + layout needs to be flexible and suitable for many of them
  + so many different manufacturers
* iOS overall is pretty much well-updated and organized
* For different OSes different languages are needed
* It is necessary to develop different apps (all the same) for several devices
  + Creating one for each operating system *by hand* is quite expensive

There are *different variables* to consider, which are independent between devices:

* Operating system
* Programming language
* Development tools (IDE, simulators, etc.)
* API
* Sensors/equipment
* Screen size
* Computational capacity

The goal is: *develop once, adapt for all*.

* Do not follow the principle of “code forking” (e.g., if iOS do this, if Android do that)
* Cross-platform frameworks for mobile development reduce market fragmentation
  + Allows to reduce negative effects
  + “Write *once*, distribute *everywhere*”

There are different *main features*:

* Application developed on time, using only one programming language
  + or a set of languages 🡪 with one language/one environment
* The chosen framework allows the distribution of the application in several applications stores
  + so, there are several applications deployed
* The frameworks usually provide support for native API

There are so many frameworks one can choose from: jQuery Mobile, jQTouch, Sencha, Sproutcore, xui, appcelerator, PhoneGap, appMobi, QuickConnectFamily, Worklight, netbiscuits, dragonRAD, pyxismobile, kony, MoSync, bedrock, LiveCode, Unity, Unreal, Adobe…. the list will go on if you will.

We move the problem from *choosing the right platform to choose the right framework*.

* Most applications are developed with frameworks, not natively
  + because of versatility and convenience) – e.g., Uber, Pinterest, etc.
* It they are developed natively, this happens because of performances reasons
* This is incredibly time-saving and cost effective for a developer
  + it needs to create effectively just one application
  + instead of one dedicated to each and every platform

Other data:

* There are billions of dollars achievable with cross-platform tools
* Different devices spread across all countries and nations
  + The most between North America, Europe, Pacific Asia, Central/South America
* The most known frameworks for this specific development paradigm are Flutter (supported by Google), React Native (supported by Meta) and Xamarin (supported by Microsoft)
  + other ones just to quote: Ionic, Corona, Sencha, Unity, etc.)

There are different *pros and cons to cross-platform development*:

* Pros
  + Wide market reach
  + Single codebase
  + Faster and cheaper deployment
  + Reduced workload
  + Platform consistency
* Cons
  + Possibly slower performance
  + UX and UI discrepancies (create widgets suitable for the platform)

There are also *pros and cons for native development*:

* Pros
  + Usually, a native application offers a better user experience, a faster and more high-performance interaction
  + Non-native applications are limited by the expressivity of the used framework
    - e.g., available APIs
  + An Apple computer is always needed
* Cons
  + Fragmentation = higher development costs
  + Problems with test

Good question: *How to choose the best framework?* App development involves 4 steps:

1. Idea analysis
2. Interface design
3. App development
4. Store deployment

Consider in particular:

* Store deployment is necessary every time there is an update and for each platform
* Native development requires repeating steps 2-3-4 for each platform

Class discussion with Wooclap: *What are the features that influence the choice of a cross-platform mobile development?*

* Documentation and good references
* Third party support
* Development cycle time and maintenance
* Cost of framework and license prices
* Energy consumption requirements
* Good learning curve for the language
* Support for native look and feel
* Community and good support
* Compatibility between different platforms

Frameworks’ classification is still an open problem. *Raj and Tolety classification* (used in research and companies) define 4 different classes (paper is inside the fundamental material – aka need to study/read it, [here](https://ieeexplore.ieee.org/document/6420693)):

* *Web* Approach
  + *General features*
    - Immagine che contiene testo, Cellulare, gadget, schermata

      Descrizione generata automaticamenteThis is not a mobile app, instead it’s a web application accessible via URL
    - Mainly developed using HTML, CSS, JS
    - It executes a web service to obtain a native version of an app
    - It does not require an installation, easier to update without manual intervention
  + Pros
    - Same interface (but not same experience) on all devices
    - No installation necessary
    - Easy update and maintenance-free
  + Cons
    - No store publishing
    - Network connection necessary
    - Difficult test
      * Cannot access mobile device hardware and software
      * Difficult to support different screen resolutions this way
    - Strongly connected to HTML5 support of the device
      * Widgets with native look and feel for applications
      * Less control over content rendering
      * Limited to leverage the gestures offered by the platforms
    - Non-native interfaces bring to low usability
    - More difficult to monetize
* This is the concept of Progressive Web App (PWA), which are web pages that behave like native applications
  + The term was coined by Steve Jobs in 2007, since apps using new functionalities like service workers and web app manifests needed to be categorized
* In particular:
  + They are developed using web technologies, therefore HTML5, CSS3, Javascript
  + It works independently from the browser
    - using *progressive enhancement* (according to the device equipment)
    - the more features the browser provides, the more features provides the app
  + It works even offline, but with limited support
  + Can be installed without using the store (but in this case, they are a sort of link)
  + Like every web page, these apps adapt themselves to device size (responsive)
  + Secure (HTTPS) and indexed by search engines
  + Easy to update
  + Support push notifications
  + No need for stores to publish the app
    - but there is no payments management
    - and there is no control of what is published
* Examples of PWAs: Sencha Touch, Angular, React (note: not React Native), jQuery Mobile
* *Immagine che contiene testo, schermata, diagramma, design

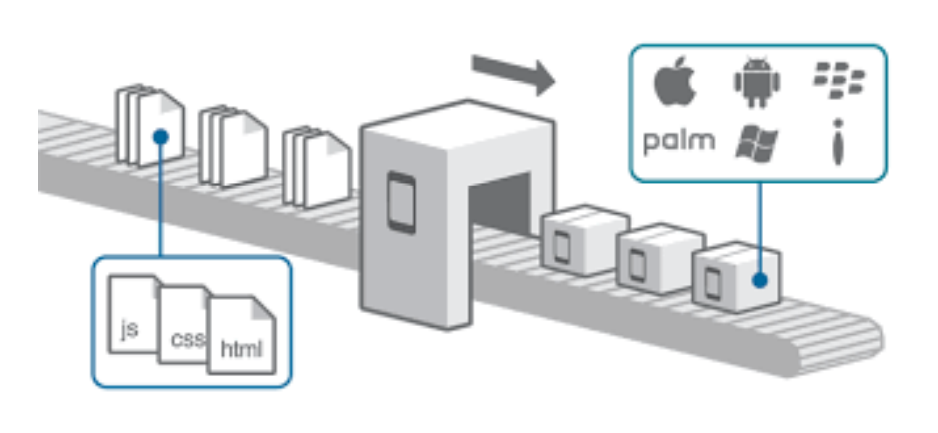
  Descrizione generata automaticamenteHybrid* Approach
  + This is developed using web technologies and gets executed inside native container on the mobile device
  + Uses the browser engine of device to render/display HTML content full screen
  + Separate in layers
    - Hybrid application
    - JavaScript Abstraction Layer
      * Allows to expose device capabilities
    - Native Library
    - Web Services
  + Pros:
    - Store publishing available
    - Reusable UI
    - Usage of device components and powered by device computing capabilities
    - Usable for both server backend and standalone applications
  + Cons:
    - Need to be installed on devices
    - Lower performances compared to native apps
    - Has cross-communication vulnerabilities because of JS
    - UI do not follow native Look and Feel hence styling is required
* Examples: PhoneGap/Cordova
  + The project started in 2008 trying to solve these problems
    - Development of mobile applications using web technologies
    - Solve the problem of low support of mobile browsers to HTML5
    - Allow access to unique features of the device
  + Actual support to HTML5 of the mobile browsers and HTML5 evolution has partially solved these problems
  + In 2011 PhoneGap code was offered to Apache to continue the development
  + Apache Cordova is the engine below PhoneGap
    - like WebKit is the engine of several browsers

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

* Apache Cordova framework is a hybrid framework
  + Applications development works with HTML, CSS and JS, well known to web devs
  + It uses plugins to access hardware components of the smartphone (camera, GPS, etc.)
  + It provides tools for testing (emulators) and deployment of the final application
  + Figure on the right shows an example of *HelloWorld* in this framework
* Other frameworks/tools allow app development using Cordova:
  + Monaca – Framework7 – NativeScript – Ionic Capacitor – Progressive Web Apps
  + Really easy to use for web developers to develop mobile applications
  + Quite easy to customize interface using CSS
* Cordova usually is not used stand-alone
  + Immagine che contiene testo, schermata, Carattere, numero

    Descrizione generata automaticamentebut as a support framework for other frameworks
* *Interpreted* Approach
  + Application code is deployed to the mobile device and gets interpreted thereafter
  + Pros:
    - Native Look and Feel
    - Store publishing
    - APIs for device components
  + Cons:
    - Immagine che contiene testo, schermata, Carattere, Marchio

      Descrizione generata automaticamenteReally difficult to reuse the UI
    - Available features depend on the framework – need code forking here
    - The interpreter can have low performances
* Examples
  + Titanium Architecture (now called Appcelerator Titanium)
  + C# runtime interpreter
* Immagine che contiene testo, schermata, Carattere, Rettangolo

  Descrizione generata automaticamente*Cross-compiled* Approach
  + The best one from the user point of view
  + Finished running the application, the compiler translates it into the native language
  + It strictly depends on the compiler and there can be errors when translating the app
  + Best for user performances
  + Pros:
    - Allows to use all the components available from native app
    - Reuse of the existing source code
      * by cross-compilation to another application run on different platforms
    - Native interface
    - Good performances
    - Store publishing
  + Cons:
    - Not reusable UI
      * specific to the platform
    - Overly complex apps can have problems during the building process
    - Mapping between source language and target language is difficult
      * so the cross-compiler supports a few platforms
      * and focuses on the common elements of those
    - Identifying/correcting cross compilation issues might be hard
* Examples:
  + Solar 2D
  + React Native 🡪 learning curve it’s really important
    - because it’s a branch from React, so it’s quite large
    - but also use common tools like HTML, JS, CSS
  + Flutter
  + Particular case: Xamarin 🡪 interpreted for Apple/cross-compiled for Android

(For your info: course started at the end of February, second week of lessons, prof. says she will release one of the homework assignments next week. I like people being organized, so I tell you straight away).

Immagine che contiene testo, Carattere, numero, ricevuta

Descrizione generata automaticamenteIn summary:

There are several methods for frameworks classification, some of them based on the development approach, others based on the result. El-Kassan et al. divide the apps into three categories (reference of this one [here](https://www.sciencedirect.com/science/article/pii/S2090447915001276)):

* *Native* app
  + developed using native languages (for the final output at least)
  + they use the tools and programming languages provided for a certain mobile platform
  + what is used in between it’s not important
  + native look and feel and full access to devices features
* *Web* app
  + website used through a mobile phone
  + no new technologies required in learning
  + same for all mobile devices
* *Hybrid* app
  + everything that falls in between
  + developed using the web technologies like the web app
    - but it is rendered inside the native app using a web view control
  + has the native app running but also the web engine (e.g., Webkit)
    - because has also other components running
    - other requirements may be precision with the user, interaction with the user, secondary factors, etc.
  + the device capabilities are exposed to the hybrid App through an abstraction layer (JavaScript APIs)
  + the app can be downloaded from the store, size is small and it’s on server
  + UI can be reused across different platforms, so lacks native look and feel

Immagine che contiene testo, diagramma, ricevuta, Parallelo

Descrizione generata automaticamenteThese are not the only ones however; consider the following figure from the paper above, detailing the other classifications:

Example:

* the game teaching fractions with pizza slices using Solar2D
  + you don’t need to know anything about the platform
  + but only know details on how the signing process work

In the compilation process, define:

* *Cross-Compiler*
  + Runs on one platform (host) but generates code for a different platform (target)
  + Used when the target architecture differs from the host architecture
  + Requires a toolchain configured for the target platform
  + Optimizes code for the target platform's hardware and operating system
  + Produces native apps and code can be reused over different platforms
    - keeping an eye on a few because mapping precisely is hard
* *Trans-Compiler*
  + Converts source code from one programming language to another
  + Often used for translating code between high-level languages or dialects
  + Doesn't necessarily involve different platforms or architectures
  + Focuses on language semantics and syntax translation
    - rather than platform compatibility
  + Mostly used to reuse legacy applications code natively

We can consider also:

* *Component based* approach
  + App development starts from different components that communicate together using interfaces, with each one having different functions
  + Each component has the same interface in every platform, but different implementations, so no need to know about them precisely
  + Usually, the output is a native application
* Pros:
  + It simplifies the development
    - assuming that there are several *off-the-shelf* components available
  + E.g., platformer games or games like Doodle Jump
* *Modeling* approach
  + With this approach the developer uses several abstract models to describe user interface and functions of the application
    - UI can be generated at execution/development time
    - Used for prototyping to evaluate apps usability in many devices/platforms
    - Helps focus on app functions rather than technical issues
  + Models are then translated into native source code
* *Cloud-based* approach
  + With this approach, all the app computations/app processing are done in the *cloud*
    - and the application receives user interaction, sends them to the cloud, and shows the result of the elaboration
  + Pros & Cons
    - Continuous network usage (high-speed network)
    - Lot of efforts in development
    - No need for specific hardware
* Examples: Stadia
  + sending logic to the server, creates the logic and sends back the streaming to the client
  + like YouTube, but not with limited commands, but the whole game itself

Again: *how to choose the right framework?* This is still an open research problem, but some steps have been taken.

Compare the different cross-platform frameworks:

* *Idea*: write one application using different frameworks, deploy on 2 target platforms, and compare results
* 4 different approaches were considered: *Web, Hybrid, Interpreted*, and *Cross Compiled*
* In this case, four frameworks were considered:
  + jQuery Mobile
  + Titanium Appcelerator
  + PhoneGap
  + MoSync
* We have to understand how to ask questions to users or to different people each time
  + Research provides “standardized” data, we would need real data and experiences, even emotions at times
  + This also consider developers feelings and habits

Immagine che contiene testo, schermata, Cartoni animati, cartone animato

Descrizione generata automaticamenteAn application like the following figure with a lot of interactions and graphics are Corona and Unity, but here was asked to the developers which other ones to use instead of those,

* This was used to help children with dyslexia
* Using the touch, the keyboard is used to select the correct letter – interaction has to be fast and in position
* Two situations
  + obstacles and letters with treasure to be found
  + fishes, background, animations changing
* When the correct path is selected, speed/difficult is increased
* In case of errors, speed/complexity decreases

The case study revolved in using 4 frameworks of different categories, then asked to evaluate the framework.

* Is this a good idea or a good result describing the judgement impartially?
* There are human factors to consider here: human psyche
* 4 times means more difficulty to make apps/algorithms each time differently
  + even for the same problem
  + more resources, more time
* The development here was to develop the app once with a framework and then be evaluated for that
* The good result would be with at least a hundred of users 🡪 very costly in testing
  + even with tens of users can be useful for smaller companies
  + big numbers remove the human factor (avoiding their personal preferences)

We analyze them (written here for the sake of completeness – you don’t need to know them in detail of course, just to give you context to think/talk about/study):

* jQuery & jQuery Mobile
  + Desktop and mobile applications were developed as a single one
  + Hardware access depends on HTML5 support of the browser (therefore not controllable by the developer)
  + Low initial knowledge (Javascript, HTML, CSS)
  + Low development complexity
  + Animation support
    - but complex animation performances rapidly decrease
* PhoneGap/Cordova
  + An application developed with Cordova is a web application plus the WebKit rendering engine
  + Allows access to several device sensors (accelerometer, compass, etc.)
  + Good performances with simple apps, but poor performances with complex apps
  + Development languages: HTML, CSS, and Javascript
* Titanium
  + Titanium allows maximizing reuse of pieces of code
  + Several APIs are available, especially for iOS and Android
  + Provides support for iOS (starting from version 5.0), Android (from 2.3.3), Window Phone, BlackBerry and Tizen (Samsung OS used inside Smart TVs)
  + Apps with native *look and feel*
  + Development language: JavaScript
* MoSync
  + Allows using different development languages: C, C++ and HTML+Javascript
  + Several APIs are available
    - but some of them are for obsolete operating systems versions
  + C++ development is a bit tricky
  + Native UI

Immagine che contiene cartone animato, testo

Descrizione generata automaticamenteThere are *different selection criteria* for applications, combining different development approaches from the ones listed before. A framework is selected *not only for convenience*, but also *considering final results* – remember this:

We will make an *energy consumption analysis*:

* Energy consumption is a crucial element for application success
  + apps that drain the battery are rapidly uninstalled by users
  + almost as tedious as having widgets/buttons cut from the UI itself
* We considered the energy consumption of apps that acquire data from different sensors:
  + Accelerometer
  + Compass
  + Microphone
  + GPS
  + Camera
* Result compared between native apps and ones developed using cross-platform frameworks
  + Cross-compiled because of platforms
* How can we avoid interferences from internal (OS events) or external (user interaction) factors?

Several authors measured energy consumption of mobile applications:

* Thompson et al. proposed a model-driven approach (SPOT, System Power Optimization Tool – link [here](https://www.dre.vanderbilt.edu/~schmidt/PDF/spot.pdf) if you want) for energy consumption estimation before application development
  + using lighter languages/predicting power consumption on API calls/power consumption on sensor usage/assessing effects on power consumption protocols
* AppScope (Yoon et al. – link [here](https://www.usenix.org/system/files/conference/atc12/atc12-final171.pdf) if you want) is an Android application that estimates energy consumption of each hardware component

As a measurement system, Monsoon Power Monitor was used (with Monsoon PowerTool used to measure) and we can give the following features over the research:

* Provided data: energy consumption, average current and consumption, estimated battery duration, etc. – measuring the difference of power consumption between devices
* Analysis
  + Goal: energy consumption comparison of different hardware components during data collection, considering different platforms and different frameworks
  + Applications considered
    - Native Android application
    - Web application
    - Hybrid application developed with PhoneGap
    - Application developed with Titanium
    - Application developed with MoSync using C++
    - Application developed with MoSync using Javascript
  + Sensors depend on the API available with each framework

The *developed applications* were various:

* Elementary applications that collect data from different sensors at a given frequency, showing data with a simple interface
* It is necessary to define *a base level (or 0 level)* of energy consumption: device in standby, airplane mode, black screen with minimum white elements
* The base level depends on the device and its battery. Considered smartphones:
  + An iPhone 4 and an iPhone 5
  + A Samsung Galaxy Nexus and a Samsung Galaxy S5
* Basically, an Hello World example was tested on each framework
* It’s quite impossible to have the same color on different screens
  + If they are not exactly the same, also the energy consumption is different
  + Colors used: black (pixel off)/white (pixel on)
    - Done to see the difference in updating

In testing, more precisely, when plotted and compared:

* base energy consumption was higher on recent devices (e.g., iPhone 5)
  + native approach has almost 0 consumption
  + web-based approach has 2-4% more in energy consumption
  + cross-compiled approaches are similar to native ones
* accelerometer energy consumption was higher on MoSync and Galaxy S5
  + JS compiler is similar to native one, C++ one requires a lots more of energy
  + this kind of data will be used to update data on the screen
* compass energy consumption was higher on iPhone devices (4/5), but almost the same for all frameworks
  + activity which requires lots of data from sensors - very high frequency
  + that’s why we want to leverage the native approach in development
* orientation sensor one between Webkit browsers was higher (Opera/Safari)
  + it uses polling, waiting for continuous updates for data when available
* GPS consumption greater when native and also with different frameworks, again, higher on Apple devices

As results:

* Cross-platform frameworks determine higher energy consumption, even if the framework generates native code
* The most expensive task is the interface update
* Data acquisition frequency strongly influence energy consumption
* Not really cross-platform
  + Frameworks have different energy consumption depending on the operating system where they are running

So, in conclusion:

* Results show that cross-platform frameworks consume more energy, hence determining lower performances and user acceptance
* Depending on the type of application, native development should be preferred:
  + Lower energy consumption
  + More APIs are available
* The results suggest that
  + Framework choice is critical
  + For a complex application, efficient frameworks are still missing
* Framework choice is crucial because it can influence user experience
  + Providing an ugly application is worse than not providing an application at all
  + Results show that, at the time of the experiment, Titanium seems to be the framework with better consumption

Some notes about *future development*:

* An efficient cross-platform framework must provide an extremely efficient user interface rendering
* Improves of efficiency of rendering engines will provide improvements even for those frameworks that work with the web approach
* The cross-compiled approach seems to be the most promising, but it is not easy to implement
  + The development of these frameworks strongly depends on the development of a compiler able to produce an efficient code for the application
  + Development should be focused on the optimization of events handling and management

Titanium seems to be the framework with better consumption for the moment, but we should always consider that providing a lousy application is worse than not providing an application at all.

All references for chapters were papers, already quoted in different parts of this chapter.

As said in the disclaimer at the beginning of this file: Corona/PhoneGap/Xamarin won’t be done. Instead, the teacher will show different examples about Corona framework:

* the first one about an endless runner with a running horse (animated GIF), with parallax scrolling of different backgrounds
* the second one about some physics simulation with soda cans thrown away
  + put images about cans/brick
  + the brick has a speed and angular velocity (called “body” inside this framework)
  + if two bodies collide, something happens
  + this allows for collision and some random mess inside a scenario
  + basically, the code define background/floor/stand and for each soda can, bodies are added to make interactions happen as physics
    - each with a bounce, friction and density
  + everything else is handled by the “Physics” library
* the third one about a guy collecting money

# Exercise 1 – Cross-platform development frameworks

*Choose a framework for cross-platform development that was not discussed during the lessons (Cordova, Solar 2D, Flutter, React Native) and classify it according to Raj and Tolety classification. Prepare a presentation of 5 minutes (max 5 slides) which briefly introduces:*

* *the chosen framework*
* *why you choose it*
* *its classification*

*The presentation must contain references to the used documentation. This exercise must be done in groups of two students and will be presented on the 19th of March 2024. The presentation must be uploaded before 15th March 2024 1.00 p.m.*

What I did

* A Google Docs with a colleague choosing the framework
* A Google Slides presentation to easily work together file base on the Math UniPD department format, because it’s simple
* 5 slides including the points above and a questions/bibliography idea

If you used Moodle at least once it’s pretty obvious, BUT:

* I asked it to prof at the end of last lesson and she said that just one to insert slides on Moodle is sufficient, she also told me to write the name and student ID of the two group members in the section "notes or comment" during the uploading phase
* Hopefully upload it in PDF, it doesn’t matter that much

Once you have uploaded, you receive a confirmation email.

# Flutter Framework

(These slides were an in-depth analysis done by students as an alternative exam modality, that’s why the format of slides is different from the others the prof. uses)

Flutter is the youngest framework, making it still a fresh solution for developing cross-platform applications – first version in 2015, released in December 2018. More precisely:

* It is an SDK for mobile devices, developed by Google, for the development of native application for iOS and Android starting from a unique *codebase* (initial name was Sky)
* It also easily allows for Windows projects/web applications to be done with this one
* It uses a *cross-compiled* approach
  + *Trans-compiled* according to El-Kassan
    - meaning we will not obtain an APK which can run on the device
    - but rather a directory which we will need to open on XCode on iOS for building the application and sending it to the store
    - So, we can run the framework everywhere
      * but in order to create a iOS application
      * we will still need an Apple computer
  + When run, OS-specific folders will be created
    - you don’t need to know the details of the program compiling
    - the only thing different is the icon, needs to be done for each OS manually
* Application written in the language Dart
  + allows for efficient execution of application
  + in particular for the web version
  + e.g., Telegram with good mobile/desktop counterpart, both interactive and good
  + React Native version was dropped because of this one
* Supported platforms: Android/iOS/IoT devices
* Some applications developed with these:
  + Google Ads/Greentea/Abbey Road Studios/Alibaba/reflectly

Its main characteristics:

* Fast development
* Expressive and flexible UI
  + a lot of effort was put to create good animations similar to the native ones
  + all done considering lots of fragmentation between Android devices
    - defining specific breakpoints and sizes
* Native performances
  + no studies on energy consumption of this one as of now

*Fast development* is done via:

* *hot reload*, which allows to build and reload the code during runtime
  + without rebuilding the application again from scratch
  + useful in particular for UI design, e.g., small elements like labels
    - moving the elements in the simulator and seeing the result immediately
  + just with small lines of code and simple changes
* it’s also *stateful*, remembering the interaction with the developer
  + redoing only the operation affected by the change and not anything else
  + also useful to test just some parts of a long, complex interaction of an application

About the *expressive and flexible UI*:

* there is a good, personalized user experience
  + thanks to the enormous amount of widgets available
    - material design (modern Android UI) and Cupertino (clean/minimal Apple UI)

Also, performances are as good as native apps:

* given each element is considered a widget even when accessing sensors
* these ones incorporate all the main characteristics of different platforms
  + e.g., scrolling, icons, fonts

There are different pros and cons to this framework:

* Pros
  + Free and open-source
  + Single codebase running virtually everywhere
  + Easy setup
  + Hot reload
  + Widgets
  + Native performances
  + Many plugins for IDE
  + Good documentation
* Cons
  + Available only for mobile
  + Low number of libraries
  + Difficult to create animations
    - since it is a general purpose framework and not thought for games
  + Need to know Dart to develop for this one

Flutter gives *guidelines* to the developers in order to – very powerful but also very risky:

* gain *control* of the whole system they are creating
  + which is a very powerful but also a risk in the wrong hands
  + maintaining predictability and consistency across all UI and interactions
* create cross-compiled applications as efficient as possible (good *performances*)
  + responsiveness, smooth animations, efficient memory usage
  + impacting positively user satisfaction and preventing frustration
* try to obtain *fidelity* from developers
  + thanks to all the quality of life improvements they focus on
  + matching UI and design specs

Another big focus is *accessibility* from this framework, bringing components doing this:

* in the mobile world there are no precise rules to realize these principles
  + this was not the first framework to do so – this is written inside of the docs
* this can be easily done defining *Semantics* and *role* inside of the code of each element
* there is native support for big fonts, screen reader support and different contrast options
  + considering mobile devices serve a lot of different purposes
    - other than calling and reading

A good option is the presence of a big *community*:

* different options, between GitHub, YT, Slack, Medium, Stack Overflow
* official website with Cookbook, Codelabs and tutorials

Let’s talk about Dart, the programming language of Flutter running on a C++ engine (similar to C#):

* It is a programming language, object-oriented, used to develop web, server, desktop and mobile applications, developed by Google (its first name was Dash)
* Supports all the known data types
* Each variable points to an object and stores a reference
* Every Dart app is a *library*
  + It is possible to use libraries for code modularity
  + *Lazy loading* for libraries (loaded only when needed)
* Once a library is imported, only some parts can be used/imported (via *show/hide*)
* Flow control is as always
* Exceptions are not managed
* Classes can inherit from other classes but only once (single-inheritance)
  + Keywords: abstract, extends, implements, @override
* (Important here) Dart code can be compiled in different ways:
  + *Just-in-Time (JIT)*
    - simply compiles the component when it’s actually needed at runtime
    - translating source code into machine code before execution
    - allows to save time during development
  + *Ahead-of-Time (AOT)* 
    - to test performances, we have to use this phase (done when deployed or run)
    - feature which makes Flutter *cross-compiled*
    - it’s only used in debug mode, never when app is built

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteLet’s have a look under the hood:

Flutter architecture is based on levels, where the lower levels implement the simplest operations and provide the operations logic, while the upper ones compose the displaying and graphical part. In particular, it is based on the following components:

* *Material and Cupertino*
  + implements widget Material (Android) and Cupertino (iOS) style
  + basically, according to OS style, they adapt
* *Widgets*
  + implements generic widgets, e.g., a button which can be pushed
* *Rendering* 
  + simplify layout management
* *Animation*
  + tween
    - an object which moves from an initial to a final position with a speed)
  + physics-based animations
* *Painting, Gestures* 
  + an area where to draw something
* *Foundation* 
  + lower layers, which allows to create easier widgets
  + allows to create a lot of widgets similar to each other (e.g., network connection)
* *Dart:ui*
  + manage communications with the Flutter engine
  + taking for example a button and rendering the shape and main functioning

A *widget* has different features (considering in Flutter, everything with an interacting interface is a widget):

* Base components of the user interface
* Each widget is an unchangeable declaration of the user interface
* A widget can define:
  + A structural element (button, menu, ...)
  + A style element (font, ...)
  + An aspect of the layout (padding, ...)
* Define as hierarchy based on composition
* Allow to manage events

Immagine che contiene diagramma, testo, schermata, Piano

Descrizione generata automaticamente

The widget building allows for a tree definition (a hierarchy basically), using build() as a method to create it. This is called at the root of this tree then activated on cascade to all other elements.

According to the following figure definition, everything is a widget and only using the *Scaffold* (top-level widget organizing app bar/body content/additional components) each one is positioned correctly.

The layout itself is a widget, organized as hierarchy.

Apart from the Scaffold, we will have the HomePage, indicating which page we are on, putting elements center/left/right to the current scene. When rendering, the build()method is not called on all elements (hot reload), since it’s not necessary to rebuild all widgets.

Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamenteThe widgets can be either stateful or stateless, considering this figure:

In particular:

* *stateful* means having a mutable state, so the element can change or be updated via an interaction or an external event
  + this might be for structural/style/layout elements
  + allows for saving time and energy
* *stateless* does not have a mutable state and remain constant overtime
  + these elements do not need to be rebuilt, because they cannot change

Immagine che contiene testo, schermata, Carattere, Rettangolo

Descrizione generata automaticamenteAn example of a stateful widget might be the following, using as elements createState() (which creates for the first time the state of an object) and setState() (which updates a widget and notifies the application when the state is changed in order to call the reload):

Flutter has a set of base widgets, the most used are:

* *Text*
* *Row*
* *Column*
* *Image*
* *RaisedButton* (it elevates when pressed upon – now replaced by ElevatedButton)
* *AppBar* (the top bar of an application)

Other features:

* to inspect all widgets available, one can use *Flutter Inspector*
* *Flutter Engine* is a runtime environment written in C++ which implements key libraries
  + it provides Dart runtime, Skia (2D graphics library), Platform channels
* *platform channels* allow communication between Dart and specific code of each platform
  + different channel types
    - *BinaryMessages* 🡪 low-level binary code to platform-specific cde
    - *MessageChannel* 🡪 bidirectional asynchronous communication
    - *MethodChannel* 🡪 calls a method and makes communication possible between specific APIs and IOS/Android OS
* *code forking*, practice of writing platform-specific code branches to accommodate differences in features, APIs, or behaviors across different target platforms
  + to import features which are not present inside other platforms, calling different APIs
  + *Immagine che contiene testo, schermata, diagramma, Rettangolo

    Descrizione generata automaticamentee.g.,* Apple which hasn’t got any vibration functionality inside iPads
* *extensions*, via usage of Package or Firebase
  + through Firebase it is possible to install packages for Flutter, in order to reduce the work needed for the developer
  + provides a vast array of packages that offer ready-made solutions for common tasks, functionalities, and integrations

To develop Flutter applications we need:

* Flutter SDK
* An editor or IDE, suggested ones are:
  + Android Studio
  + IntelliJ IDEA
  + Visual Studio Code
* For the proposed IDE there are Flutter plugins

It’s not so hard to setup this framework:

* It is possible to install Flutter on Windows, macOS o Linux
* Installation process:
  + SDK installation
  + PATH variable modification
  + command flutter doctor 🡪 Check for missing packages

With this simple example we will learn how to use the following components of the framework:

* Stateful widget
* Immagine che contiene testo, schermata, design

  Descrizione generata automaticamenteStateless widget
* Tabbed layout

The application has a tabbed layout with the following pages:

* Page 1: allows to increate a counter through button click
* Page 2: allows to decrease a counter through a but ton click

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteThe following are the classes employed:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteThis goes for the first page:

Immagine che contiene testo, schermata

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteThen, its state:

Immagine che contiene testo, schermata, software

Descrizione generata automaticamenteCompletely, the application:

Immagine che contiene testo, software, Software multimediale, Icona del computer

Descrizione generata automaticamenteOverall, the interface behaves as follows:

# React Native Framework

(Again, these are student-based slides for the examination – also, as of now, slides on Moodle are not complete considering the ones in class)

React Native is a cross-platform open-source framework that implements Facebook’s ReactJS (*react.js*) library. Some features:

* It uses JavaScript, in particular in its JSX form
  + Languages used: JavaScript and XML
  + Easy to learn both for devs/web devs and not-so-technical people
* Everything inside this framework is a component (component-based) and it’s made for creating UIs
* It can be used for developing Android and iOS mobile applications
* Like Flutter, it has a very large community and many docs/tutorials
  + supported by a very big company: Meta
  + along with the first one, the most used framework today
* Developing and debugging is very fast
  + just like Flutter which has hot reload
  + compiler understands changed components and recompiles only them
* Only pro over Flutter: easier to learn because of web technologies used
* Native look and feel
* Allows to create and use personalized components
  + they accept input data, called properties (props) and return React elements
* Possibility of integrating React Native with an already existing mobile application
* Develop once 🡪 deploy for all (without using native tools)
* It works with command-line tools

A bit of history:

* First released as an open-source project by Facebook in 2015
* Created because the use of HTML for the creation of Facebook app brought to very low performance and user experience
* They initially created a way to generate UI elements for iOS using a JS thread in background
  + then the framework was created starting from this prototype during an hackathon
  + user interaction was really slow
  + combined the pros of hybrid approach (web technologies – very low learning curve) with cross-compiled approach (good performances)

Some apps created with this: Facebook, Skype, WordPress, Discord, Tesla, Pinterest, NFL, Outlook, Office, Teams, PlayStation App, Xbox Game Pass, Puma.

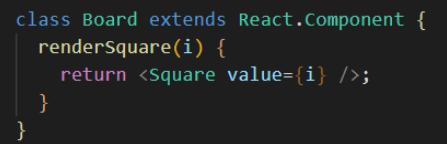
A small code example is shown: simply a JS function defined with fields and placeholders to activate logic.

* As said, it’s *component-based*:
  + Every different part of the application is a component
  + Every component is indipendent and reusable
  + They are JavaScript functions that accept input data (props) and return React elements

React uses a JavaScript representation of the DOM, called *Virtual DOM* (*VDOM*):

* Starts from DOM (Document Object Model)
  + where each element is defined inside of a tree
  + the actual dom is based upon it
* Each and every time a component is updated
* React creates a new VDOM and compares it with the old one
  + in order to render only what has been modified
  + same principle as hot reload in Flutter

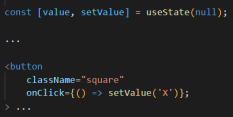
Each component can have properties (*props*) and a state:

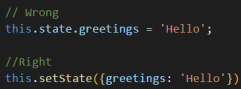
* *Props* (properties) are used to configure a component when it renders and allow to customize it
* They are ‘read only’
* They’re used to influence components in a top-down approach (Parent to Child)

There is a relationship between props and state:

* *State* is used to keep track of any component data that is expected to change over time
  + due to user action, network response, etc
* These can change overtime, props cannot
* States can be used to influence components in a bottom-up approach (Child to Parent)
  + passed from children to parents easily

More in detail over states:

* To modify the state, it should be used setState()
* React may merge multiple calls to setState() into one but these updates may be asynchronous
* Another way to manage the state is to use the ‘Hooks’ (React 16.8)



*Redux library* can be used to make the state update synchronous.

* To connect Redux to any application, we need to create a *reducer* and an *action*:
  + An *action* is an object (with a type and an optional payload) that represents the will to change the state
  + A *reducer* is a function that takes the previous state and an action as arguments and returns a new state
  + Redux also introduces two functions:
    - mapStateToProps
      * used to make state accessible to the screens that implements it
    - mapDispatchToProps
      * to access to defined actions

The following is a code example for a color-changing wheel app:

Immagine che contiene testo, schermata, Cellulare, gadget

Descrizione generata automaticamenteAnother example for a button:

Immagine che contiene testo, schermata, multimediale, Sistema operativo

Descrizione generata automaticamenteJust to show you, the following are examples of different components (not required to know):

Immagine che contiene testo, schermata, multimediale, Sistema operativo

Descrizione generata automaticamente

# Store Deployment

Once completed the development of the app, there are two critical steps to take:

* Test
* Deployment

Other notes:

* Both the stores, with different accuracy levels, test the applications
  + before adding them to their catalog
* Therefore, it is essential to *deeply test* the application before the deployment phase
* The deployment procedure asks for different screenshots of the application
  + both for the tablet and the smartphone versions
  + this phase should not be underestimated
    - as it may take some time
    - and several temporal constraints are not manageable by the developer

There are different management policies between stores:

* Apple Store has a lot more control and can take much more time to approve (up to 3 weeks)
  + they have the claim of applications having “*very* good quality”
    - God is in the details (e.g., widgets, elements, buttons, etc.)
    - app has to render perfectly on all possible resolutions
  + lots of testing and debugging are done before putting the apps on the store
  + when you want to deploy an app to a device
    - a device has to be connected to the user
    - connect devices and plug it for one of the users
      * up to 100 devices/users
  + TestFlight: private store to buy assets, test apps and collect feedback
    - it can be used by devs, companies, etc.
  + much more precision required but more manual control over testing
    - app working correctly on all possible products
* Google Play Store is much more permissive
  + the same day or the day after can be approved or rejected
    - requires manual feedback from lots of users to remove
    - a reason for rejection can be also a high number of crashes/run time errors
    - otherwise, literally everything gets published
  + much more choice when installing applications
    - *.apk* format to install them
      * not even a store required to install apps

## Google Play Store

We start looking at stores more in depth, starting from the Google Play Store.

* It is created to facilitate the search for the name of an app and then the download immediately
  + this can also happen inside a normal search engine
  + the stores have search engines ad-hoc for them with customized algorithms
    - the information present here is retrieved via reverse engineering
    - done via uploading of fake apps in order to understand how it works
* In addition to a Google account, it is necessary to have a developer account on the Google Play Developer Console (reference [here](https://developer.android.com/distribute/console/)):
  + 25$ unlimited duration, unlimited number of apps, it allows different accounts with different roles
    - inside of the Console, one can see the app status, how they are going on the store and other information
    - inside Apple store, 99 dollars/euros are paid once every year
      * Apple logic! (sorry Apple fanboys :-D)
    - overall the costs are very low, in order to stimulate developers to publish applications
      * this does not avoid the absolute chaos the store is (sadly)
      * since Play Store does not run too many checks on what’s published
  + Credit card is required to be available inside of the account
    - Normally, require this info only when needed (too much trust needed)
* For payment apps or apps with in-app purchases, it is necessary to have an account on the Google Payments Center
  + activated from the console under the section “Commercial Account”
  + and after “Configure the commercial account” with the required data
* One thing Google is pretty strict upon since last years is the change of policies
  + particularly regarding Android system updates and latest APIs compatibility
  + or even change to legal rules for applications
  + if such are not anymore compliant, they get automatically removed or unlisted

Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamenteThe *application signing* process is required in order to put an app inside the store. It is very simple (compared to Apple once again) but it’s not possible to use debug signatures. It allows all APKs to be digitally signed with Google signing key and developer key together:

To deploy the apk of an app in the Play Store it is not possible to use debug signatures:

* Immagine che contiene testo, Carattere, schermata, linea

  Descrizione generata automaticamenteTo generate a private key
  + it is possible to use the keytool command from a shell
* Some information required:
  + keystore password and key password, developer name and company name

There’s a lot of *required information* (here to be gathered comprehensively, I won’t remember either):

* APK of the application
* Name
* Description
* Screenshots
  + providing a lot of screenshots = faster approval process in stores
    - also more convincing to the users to download it
  + at least two for smartphones, min 320, max 3840, .jpg or .png 24 bits no more than 2:1
  + at least two for tablets
    - at least one screenshot for 7 inches tablet and one for 10 inches tablet
  + a good icon (png 32 bits, with alfa)
    - nice layout, good description, some interesting factors
  + 512x512
* Feature image for the presentation page
  + 1024x500, jpg or png, 24 bit
  + really important for presentation purposes and to choose it over other apps
* Other images
  + Android TV, Wear OS, promotional images, trailers, screens for wearable devices, etc.
  + not so mandatory in the end
* APK
* Categorization
  + type and category
* Content classification
  + necessary to determine the minimum age for the app
  + requires compiling a questionnaire
  + one of the few checks Google does before putting the app publicly
    - Is it suitable for children?
    - Does it contain advertisements?
    - Does it contain in-app products?
* Contact email
* URL with privacy information
* List of compatible devices
* Prices and countries where it is available

There are other services which can be added:

* App signature can be managed using the console
* It is possible to add different app localizations
* It is possible to buy translation or optimization services
* The app can be distributed as an alpha or beta version, deciding if it can be tested by
  + A closed group (email invitation)
  + An open group (all the users of the Google Play Store)

Some notes about *prices and distribution*:

* Once an application is deployed as free, it is not possible to transform it into a payment app
  + solution: ex-novo project (start from scratch)
* The price can be automatically converted into different currencies or can be manually defined
  + both fixed and even require payment in other countries
  + the Freemium version (free base version, advanced paid functionalities) is a good solution to let the user evaluate the app before buying it
* It is possible to buy different app localizations, both translations and adaptations to different cultures and useful in app promotion
* Google keeps 30% of the net amount
* Only some hours are required before publishing an app
  + it needs to undergo tests but in quick time, a response will be there
  + a wrong classification and/or categorization can lead to the app removal

## Apple Store

Now let’s talk about the Apple App Store.

# iOS Platform

# Android Platform

# Mobile Design

# Wearable Devices

# Multimedia Data Encoding

# Images

# Audio

# Video