AMD-00 (102386): Course Presentation

Applications for mobile devices & Fall 2020-2021

Jordi Mateo Fornés jordi.mateo@udl.cat

Presentation

- Dr. Jordi Mateo Fornés
- **Office**:
  - Office A.12 (Campus Igualada)
  - Office 3.08 (EPS Lleida)
- **Email**: jordi.mateo@udl.cat
- **Twitter**: https://twitter.com/MatForJordi
- **Github**: https://github/JordiMateoUdL

Course

- **Applications for mobile devices**.
- Grau en Tècniques d'Interacció Digital i de Computació
- Campus Igualada - Escola Politècnica Superior - Universitat de Lleida
- All the code developed in this course can be found in this repository: DAM Course.

Agenda

Agenda

1. Introduction
2. Mobile Platforms
3. API Frameworks
4. Current trends and Overview
5. Course: 102386

Introduction

Context

- What do you expect to learn?
- What should you learn?

Please use this jamboard to answer: Jam-DAM-vl00.

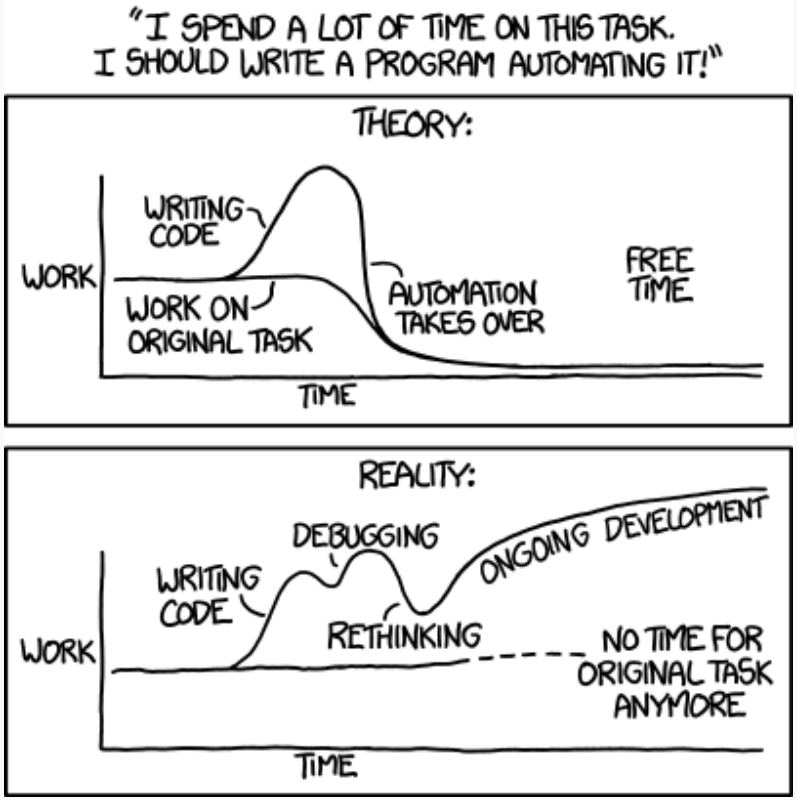
Mobile devices App development

- Programming is easy, but software engineering is hard. (Hunt and Thomas 2000)
- Developing apps is multidisciplinary (Martin 2008)
  - Write code.
  - Develop or use third-party APIs.
  - Design and scalable and maintainable architecture.
  - Creative and Usable design.

Warm-Up (1)

- Care about your craft: Why spend your life developing code unless you care about doing well?
- Don't live with broken windows. Fix bad designs, wrong decisions or poor code asap.
- DRY. Do not repeat your self.
- Make it easy to reuse
- Don't think outside the box - Find the box: We faced with an impossible problem, identify real constrains. Does it have to be done this way? Does it have to be done at all?

Warm-Up (2)



- Don't be afraid to say "I don't know" .. but follow it up with: "I'll find out"
- Work with a user to think like a user.
- Test early, test often, test automatically.
- Estimate the order of your algorithms.
- Write code that writes code.

What should you learn?

Tools

- IDE
- Sketch, JustinMind, AdoveXD
- GIT
- Issue trackers
- Slack

Workflow

- Agile methods
- Git flow
- Review code
- Testing
- Integration

Mobile Platforms

Mobile Platforms (1)

- What mobile platforms do you know?
- What is your experience?

Mobile Platforms (2)

- Native tools:
  - Android
  - iOs
- Cross-platform tools:
  - PhoneGap
  - React Native
  - Xamarin
  - Flutter
  - Kotlin Native

Native vs Cross-Platform (1)

- Native tools:
  - Allows users to learn quickly.
  - Easy to discover (Play Store or Apple Store).
  - Easy to use the device hardware.
  - High Performance and Great UX.
- Cross-Platform tools:
  - Portability.
  - Faster development.
  - Cheaper.
  - Easy to support and maintain.

Native vs Cross-Platform (2)

What is the best option? **It depends**. Everyone needs to choose the one that suits their needs in a better way.

Consider:

- One or multiple platforms
- Cost
- Technology

Android

- Advantages
  - Kotlin as a programming language.
  - Mature architecture.
  - ConstraintLayout 2.0.
  - Google is behind
- Disadvantages
  - Fragmentation
  - Android X
- Tools
  - Android Studio, IntelliJ Idea, Visual...
  - Emulators available on all platforms

iOS

- Advantages
  - Swift as a programming language.
  - Swift is open source.
  - Fast adoption of latest OS.
- Disadvantages
  - Cost
  - You need a mac to develop.
  - Strict App review.
- Tools
  - Xcode, AppCode

React Native

- Backed by Facebook.
- Used by Instagram Facebook, Airbnb, Walmart, Tesla...
- React library and JavaScript to deliver a native experience on iOS and Android
- Fast development.

Ionic

- Makes the best use of web tech HTML, CSS, javascript.
- Cordova to wrap the apps in native containers.
- Built on Angular.
- Strong community support.
- Highly interactive apps.
- Easy learning curve.

Flutter

- Backed by Google.
- Used by Alibaba, Hamilton Musical, Google Ads,. . .
- High performance.
- Dart is a modern, multi-paradigm and objected-oriented programming for building web apps, mobile apps, and desktop apps.

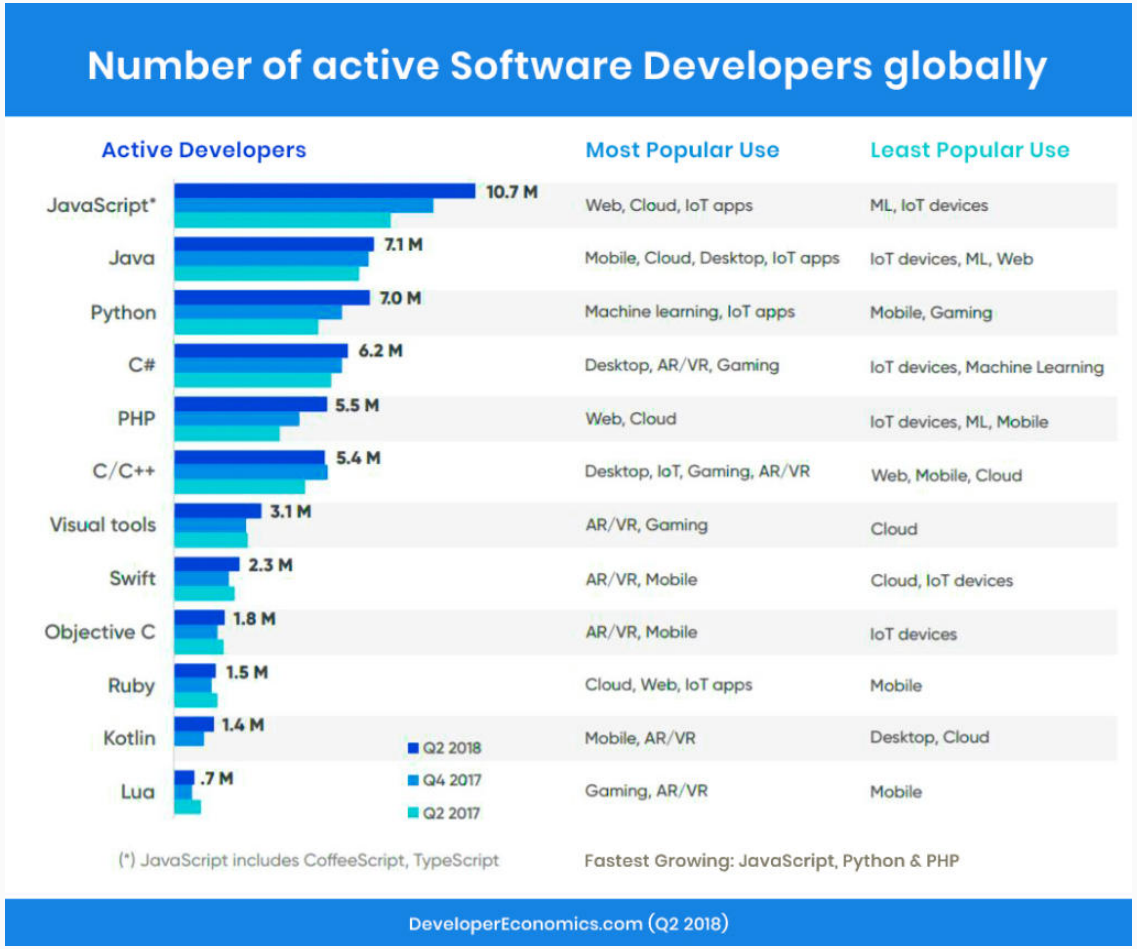
API Frameworks

Most common (API Frameworks)

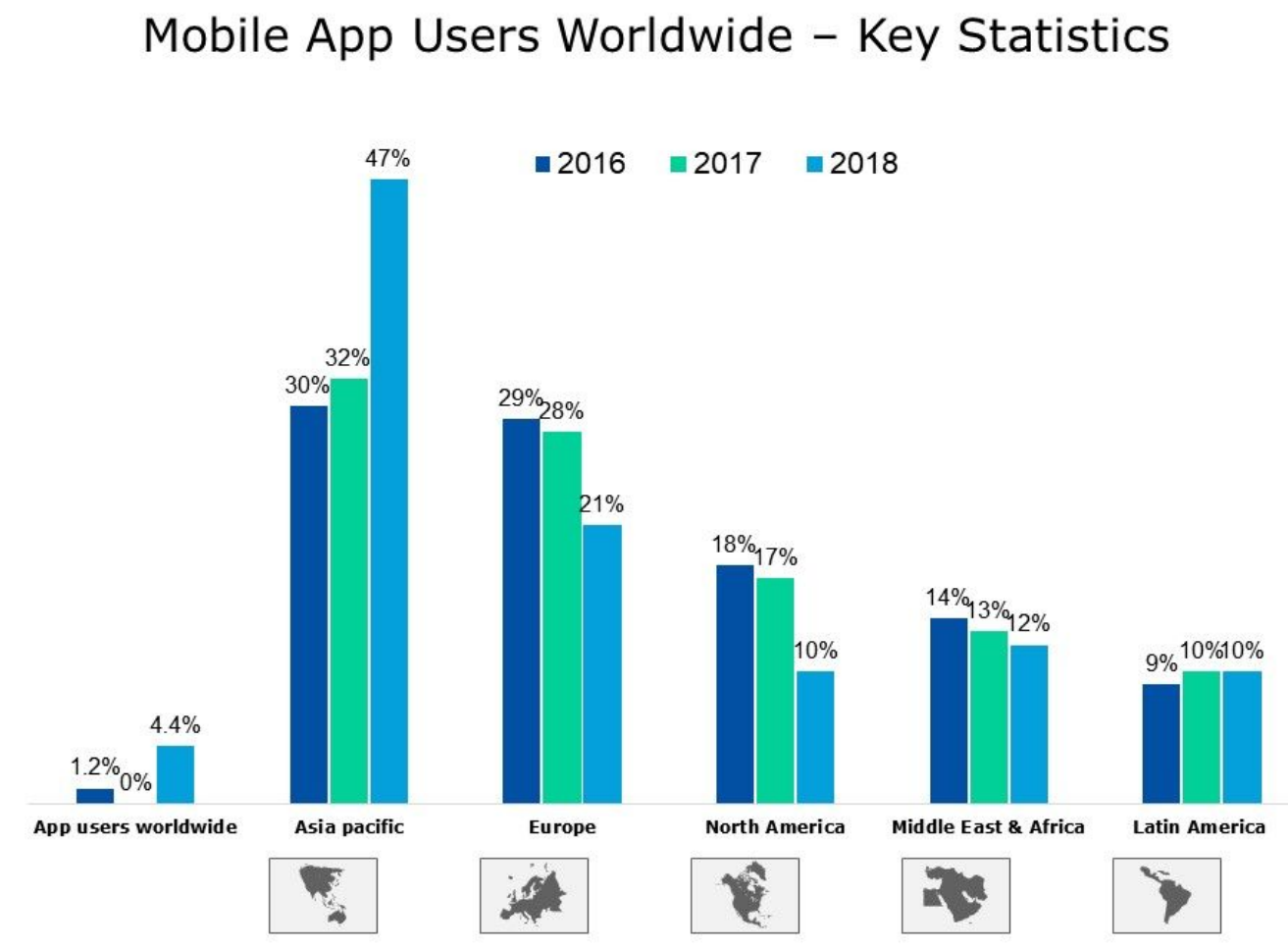
- Spring
- Django
- Flask
- Falcon
- Express (NodeJS)
- Ruby on Rails

Overview

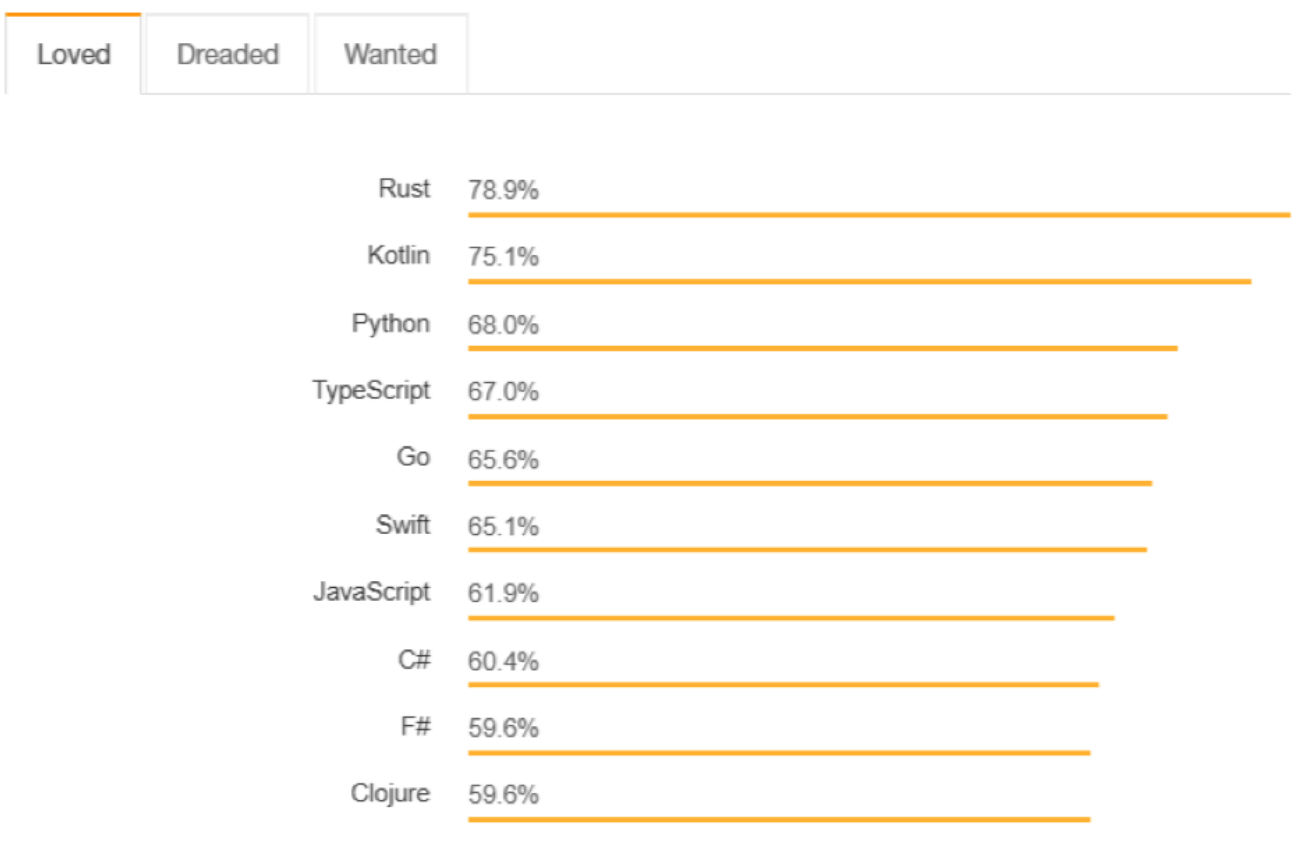
Developers


Number of active Software Developers globally
Active Developers
Most Popular Use
Least Popular Use
JavaScript*
10.7 M
Web, Cloud, IoT apps
ML, IoT devices
Java
7.1 M
Mobile, Cloud, Desktop, IoT apps
IoT devices, ML, Web
Python
7.0 M
Machine learning, IoT apps
Mobile, Gaming
C#
6.2 M
Desktop, AR/VR, Gaming
IoT devices, Machine Learning
PHP
5.5 M
Web, Cloud
IoT devices, ML, Mobile
C/C++
5.4 M
Desktop, IoT, Gaming, AR/VR
Web, Mobile, Cloud
Visual tools
3.1 M
AR/VR, Gaming
Cloud
Swift
2.3 M
AR/VR, Mobile
Cloud, IoT devices
Objective C
1.8 M
AR/VR, Mobile
IoT devices
Ruby
1.5 M
Cloud, Web, IoT apps
Mobile
Kotlin
1.4 M
Mobile, AR/VR
Desktop, Cloud
Lua
.7 M
Gaming, AR/VR
Mobile
Q2 2018
Q4 2017
Q2 2017
(*) JavaScript includes CoffeeScript, TypeScript
Fastest Growing: JavaScript, Python & PHP
DeveloperEconomics.com (Q2 2018)

Users


Mobile App Users Worldwide – Key Statistics
2016
2017
2018
1.2%
0%
4.4%
30%
32%
47%
29%
28%
21%
18%
17%
10%
14%
13%
12%
9%
10%
10%
App users worldwide
Asia pacific
Europe
North America
Middle East & Africa
Latin America

Languages

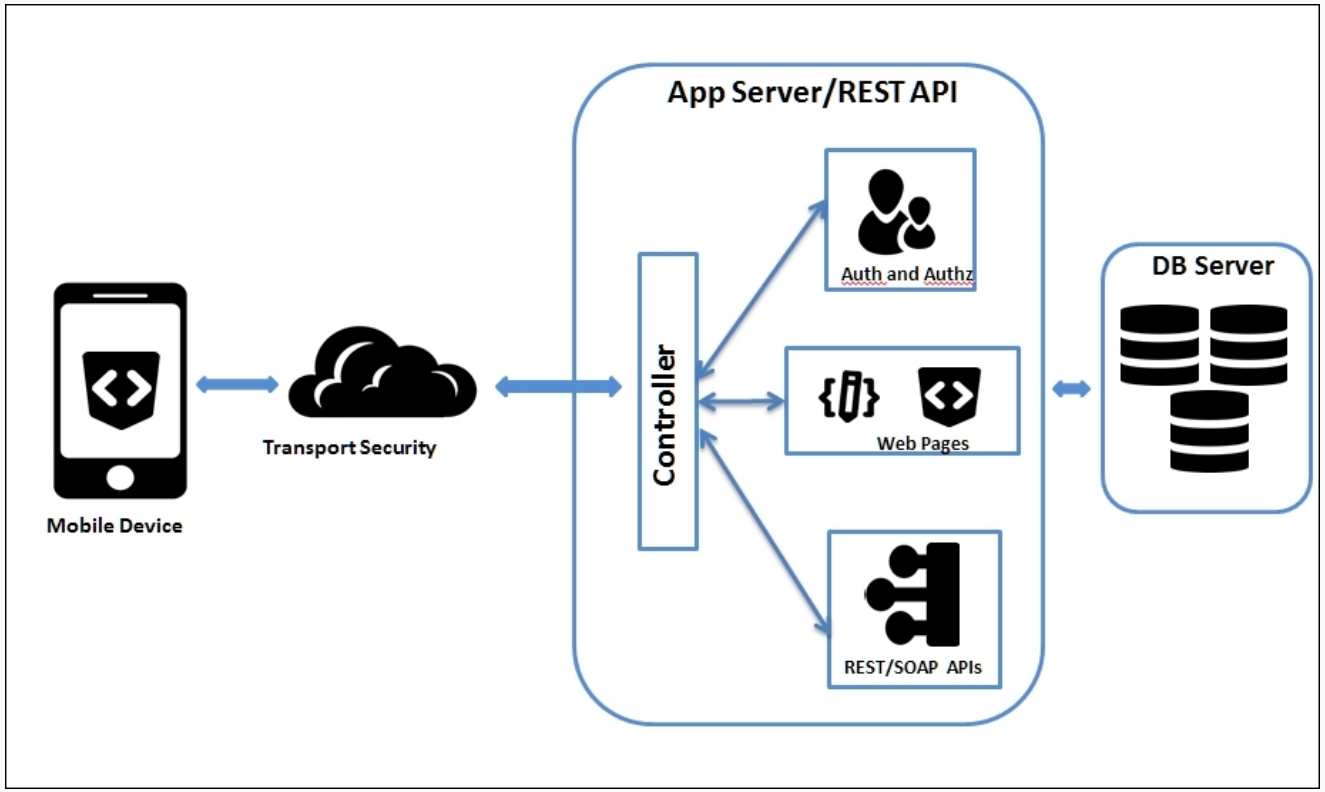

Loved
Dreaded
Wanted
Rust 78.9%
Kotlin 75.1%
Python 68.0%
TypeScript 67.0%
Go 65.6%
Swift 65.1%
JavaScript 61.9%
C# 60.4%
F# 59.6%
Clojure 59.6%

Course: 102386

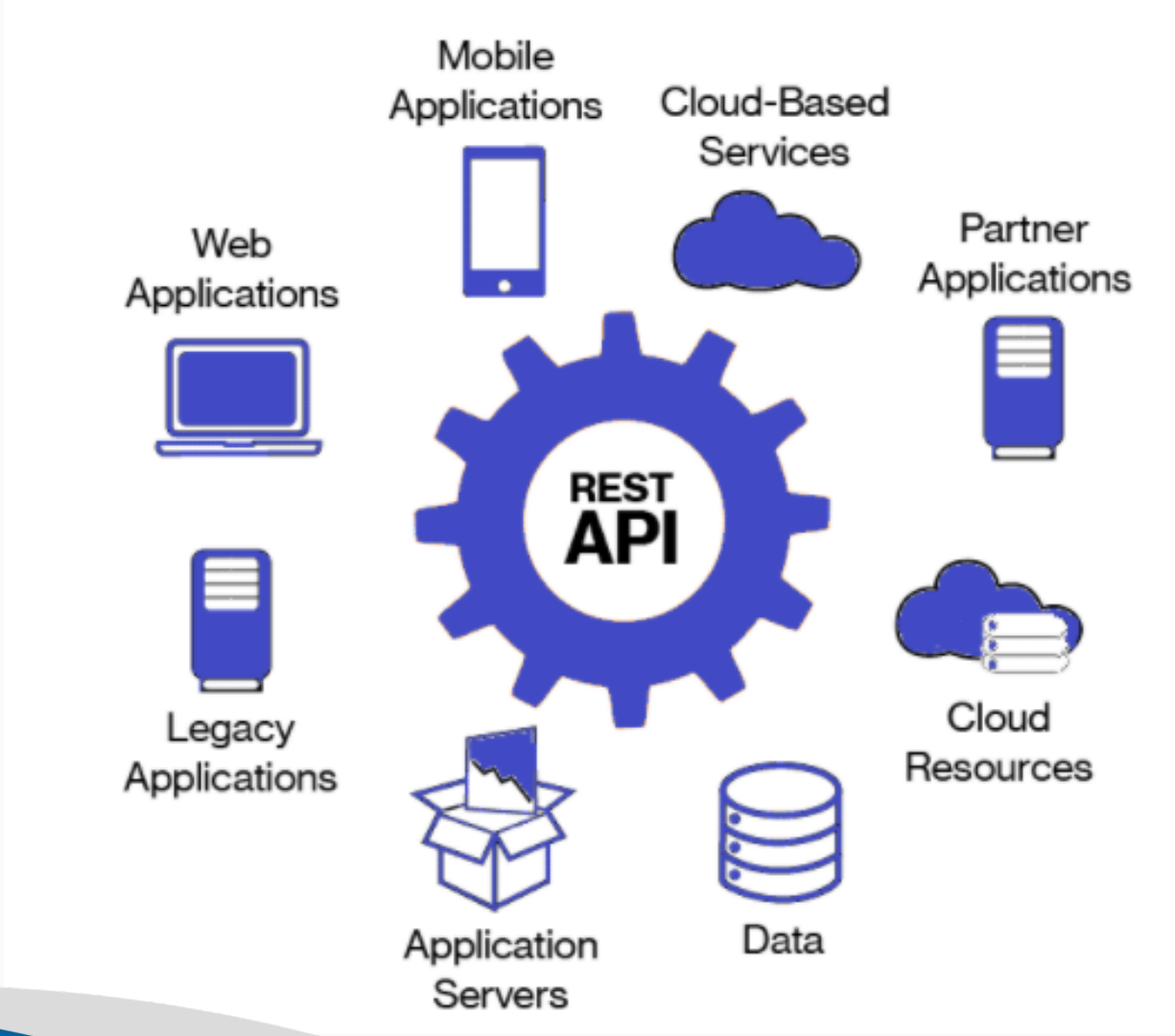
Objectives

- Understand the Android platform and the elements that make it up.
- Understand and use coding patterns.
- Develop applications for the Android operating system.
- Establish the bases for implementing additional functionalities (access to the database, access to resources and features of the mobile, etc.).
- Develop and use API as a backend.
- Get to know the step of publishing Android applications.

Architecture to learn (1)



APIs and Services



Evaluation

You must pass both exams with a minimum mark of 5 to pass the course.

- **First Exam (P1)**: *15%* Live coding exam.
- **First Exam (P2)**: *15%* Live coding exam.
- **Project**: Development of your own app. **50%**.
  - **Milestone 1 (M1)**: *12%*.
  - **Milestone 2 (M2)**: *14%*.
  - **Milestone 3 (M3)**: *24%*.
- **Common Part**: *20%*. (Projecte Integrador)

Code Evaluation

Key	Excellent (9-10)	Very Good (7-8)	Acceptable(5-6)	Unsatisfactory (>5)
Program Development	The program compiles has no logic errors and exceeds specifications.	The program compiles has no logic errors and meets specifications.	The program compiles some minimal errors and meets most specifications.	The product produces incorrect results and/or does not compile at all and fails to meet the majority of specifications.
Modularity	The program is decomposed into coherent and reusable units, and unnecessary repetition has been eliminated.	The program is decomposed into coherent units, but may still contain some unnecessary repetition.	The program is decomposed into units of appropriate size, but they lack coherence or reusability. The code contains unnecessary repetition.	The program contains big functions or is decomposed in ways that make little sense.
Clarity	The project contains appropriate documentation for all primary functions variables or non-trivial algorithms. The code can be read.	The program contains some documentation. Good formatting and style. But reading is not fluent.	The program contains some documentation, but style and format are not appropriate for good reading.	The program contains no documentation or is impossible to read due to style or formatting.
Testing	The project contains appropriate unit and integration tests for all the functionality	The project contains appropriate unit and integration tests for most of the functionality	The project contains either unit or integration tests for most of the functionality	The project does not contain or contains a few unit or integration tests
Usability	User interaction and views are fluent and natural to the users.	User interaction generally meets the specifications and is acceptable to the user.	User interaction minimally meets the specifications but does not increase the usability of the program.	User interaction is incomplete and does not meet the specifications.
Design Patterns	The code uses appropriate design patterns and principles.	The code uses patterns and some good design principles.	The code uses patterns but not always in the proper way.	The code does not use pattern either design principles.
Efficiency	The code is extremely efficient without sacrificing readability and understanding	The code is fairly efficient without sacrificing readability and understanding	The code use brute force, and waste resources and/or is unnecessarily long.	The code is huge and patched together without taking care of efficiency regarding resources
Completeness	The project shows evidence of excellent case analysis, and all possible cases are handled appropriately.	Programs show evidence of case analysis that is mostly complete but may have missed minor or unusual cases.	The program shows some evidence of case analysis but may be missing significant cases of mistakes in handling some scenarios.	The program rarely handles different cases, or flawed case analysis can be shown.

Methodology

- Students must perform the proposed **HandsOn** in the field before the sessions. The live sessions will serve to agree on doubts and expand the concepts worked on in the **HandsOn** (1h).
- Practical sessions will be focused on realizing the implementation of the project. However, a few sessions will be aimed to perform some workshops during the season of live coding (2h).
- Open Source and pragmatic methodology, all your development must be public in Github.
- Functional deliveries, with constant feedback and suggestions allow students to pivoting and make corrections.

Project using DevOps

- GitFlow:
  - Develop branch: All development until next release.
  - Master branch: Most recent release.
  - Features: Develop isolate feature and merge/rebase then in the develop branch.
  - **HandsOn 01**: Digging into Git
- Project Management (Agile and Scrum):
  - Project definition and planning
  - Project launch and execution
  - Performance and control
  - Project close
- Docker:
  - Containerizing the applications to support the development and quick releases.

What languages you should learn?

- **Mandatory**:
  - *Android Programming*:
    - JAVA
    - KOTLIN
  - *Backend Programming*:
    - Python (Falcon)
- **Optional** (depending time, availability and your motivation):
  - *Cross-Platform Programming*:
    - Flutter
  - *Game development*:
    - Unity

Java

- Advantages:
  - Easy to learn, understand, and flexible.
  - A good choice for cross-platform apps.
  - Java has an extensive open-source ecosystem.
  - More compact and light apps.
  - Fast build compared with Kotlin.
- Disadvantages:
  - Limitations that causes problems in android design.
  - You need to write more.
  - Requires a lot of memory

Kotlin

- Advantages:
  - Easy to switch from Java.
  - Smart extensions to build.
  - More concise.
  - Compatible with all Java libraries, frameworks, and JVM.
  - Compatible with Gradle or Maven.
  - Fast build compared with Kotlin.
- Disadvantages:
  - Slower compilation.
  - Less community.
  - Not as mature as JAVA.

Comparative

Attributes	Java	Kotlin
App Performance	High	Super High
Android Studio 3.0 Support	Partial	Excellent
Code Quality	Not-Optimized	Excellent
Market Presence	Excellent	Good
Adoption Cost	High	Low
App Security	Good	Excellent
Support for Complex Architecture	Excellent	Not Good

That's all folks

www — jordimateofornes.com

github — github.com/JordiMateoUdL

twitter — @MatForJordi

gdc — Distributed computation group

References:

Hunt, Andrew, and David Thomas. 2000. *The Pragmatic Programmer: From Journeyman to Master*. USA: Addison-Wesley Longman Publishing Co., Inc.

Martin, Robert C. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1st ed. USA: Prentice Hall PTR.