

Software technologies

Clara Gros
Babacar Toure
Yoann Roussel

November 2019

Contents

1	Introduction	3
2	Programming languages	3
2.1	History of programming languages	3
2.1.1	Early developments	3
2.1.2	Consolidation and growth	4
2.1.3	Compilation and interpretation	4
2.2	Uses of programming languages	5
2.2.1	Web development	6
2.2.2	Game development	6
2.2.3	Mobile development	6
3	Most popular languages overview	7
3.1	Web development	7
3.1.1	Trendings	7
3.1.2	Front-end development - client-side	8
3.1.3	Back-end development - server-side	9
3.2	Mobile development	11
3.2.1	iOS	11
3.2.2	Android	11
3.3	Java and Kotlin	12
3.4	Cross platform applications	14
4	Data structures	16
4.1	SQL vs. NoSQL	16
4.2	Most popular mobile databases	16
5	Conclusion	19

1 Introduction

Language has been our primary mean of communication and human interaction for thousands of years. For a community, the language contained the words that the people need to communicate, words themselves are abstract, but they indicate the meaning, they point to objects or actions, etc.. When you look at your computer, you'll find it's not so much different. There are many pieces of hardware and software that need to communicate with each other. Your application is reacting to the mouse and keyboard or even the mic, it can read files from your disk storage and so on. But at the end of the day, the machine understands nothing but bits, 1s, and 0s, the combination of which creates meaning.

This state of the art aims at exploring different programming languages depending on what you want to code. First we will provide you with a brief overview of the history of the implementation of these languages [ref1]. Then we will study some frequently used languages in more detail before comparing them [3]. After having an overview of the languages applied to various fields of application, we will specifically focus on the development of mobile applications and the technologies currently in use. We will also discuss the subject of choosing a database structure since they are essential for the development of a web/mobile application. We will conclude by choosing in our case the technologies that we believe are best suited to our mobile application project.

2 Programming languages

2.1 History of programming languages

2.1.1 Early developments

- First-generation programming languages: Machine languages
Very early computers, such as Colossus, were programmed without the help of a stored program, by modifying their circuitry or setting banks of physical controls. Slightly later, programs could be written in **machine language**, where the programmer writes each instruction in a numeric form the hardware can execute directly. Machine languages were later termed first-generation programming languages (**1GL**).
- Second-generation programming languages: Assembly languages
The next step was development of so-called second-generation programming languages (**2GL**) or **assembly languages**, which were still closely tied to the instruction set architecture of the specific computer. These served to make the program much more human-readable and relieved the programmer of tedious and error-prone address calculations.
- Third-generation programming language: high-level programming languages
The first high-level programming languages, or third-generation programming languages (**3GL**), were written in the 1950s. John Mauchly's Short Code, proposed in 1949, was one of the first high-level languages ever developed for an electronic computer. Unlike machine code, Short Code statements represented mathematical expressions in understandable form. However, the program had to be translated into machine code every time it ran, making the process much slower than running the equivalent machine code. At the University of Manchester, Alick Glennie developed Autocode in the early 1950s. As a programming language, it used a compiler to automatically convert the language into machine code: it is considered to be the first **compiled high-level programming language**.
- Fourth-generation programming languages (**4GL**) are computer programming languages which aim to provide a higher level of abstraction of the internal computer hardware details than 3GLs. Fifth-generation programming languages (**5GL**) are programming languages based on solving problems using constraints given to the program, rather than using an algorithm written by a programmer.

2.1.2 Consolidation and growth

The 1980s were years of relative consolidation. C++ combined object-oriented and systems programming. The United States government standardized Ada, a systems programming language derived from Pascal and intended for use by defense contractors. In Japan and elsewhere, vast sums were spent investigating so-called "fifth-generation" languages that incorporated logic programming constructs. The functional languages community moved to standardize ML and Lisp. The rapid growth of the Internet in the mid-1990s created opportunities for new languages. Perl, originally a Unix scripting tool first released in 1987, became common in dynamic websites. Java came to be used for server-side programming, and bytecode virtual machines became popular again in commercial settings with their promise of "Write once, run anywhere". These developments were not fundamentally novel, rather they were refinements of many existing languages and paradigms (although their syntax was often based on the C family of programming languages). Programming language evolution continues, in both industry and research. Current directions include security and reliability verification, new kinds of modularity (mixins, delegates, aspects), and database integration such as Microsoft's LINQ.

2.1.3 Compilation and interpretation

Lying between machine languages and high-level languages are languages called assembly languages. Assembly languages are similar to machine languages, but they are much easier to program in because they allow a programmer to substitute names for numbers. Machine languages consist of numbers only.

Lying above assembly languages are 3GLs. 3GLs are far removed from machine languages and represent the class of computer languages closest to human languages. They have made it possible to write programs in a more familiar way, close to English, and which do not depend on the processor that will be used. This has brought significant progress compared to the two previous generations. Regardless of what language you use, you eventually need to convert your program into machine language so that the computer can understand it. There are two ways to do this: Interpreting or compiling the program [8].

In interpreted languages (Python, Java...), the source code (the one you write) is interpreted by a software called an interpreter. The interpreter will use the source code and input data to calculate the output data: The interpretation of the source code is a "step-by-step" process: the interpreter will execute the code lines one by one, deciding at each step what to do next.

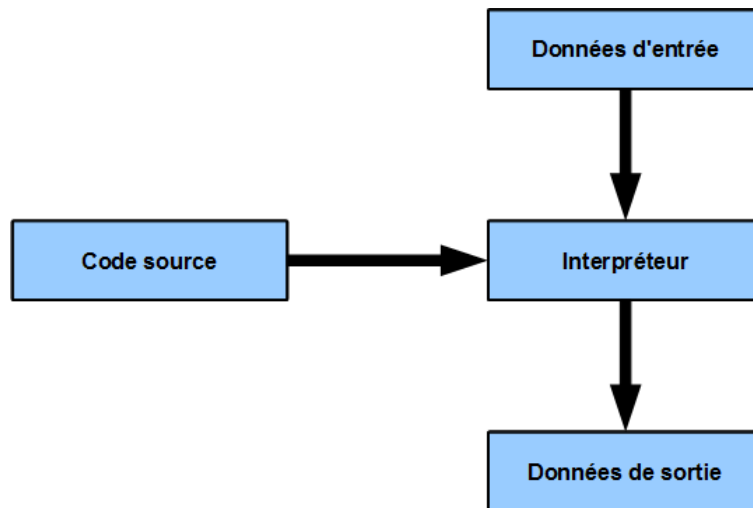


Figure 1: Interpreted languages basics

Compiled languages In these languages, the source code (the one you write) is first compiled, by a software called a compiler, into binary code that a human cannot read but which is very easy for a computer to read. It is then directly the operating system that will use the binary code and input data to calculate the output data.

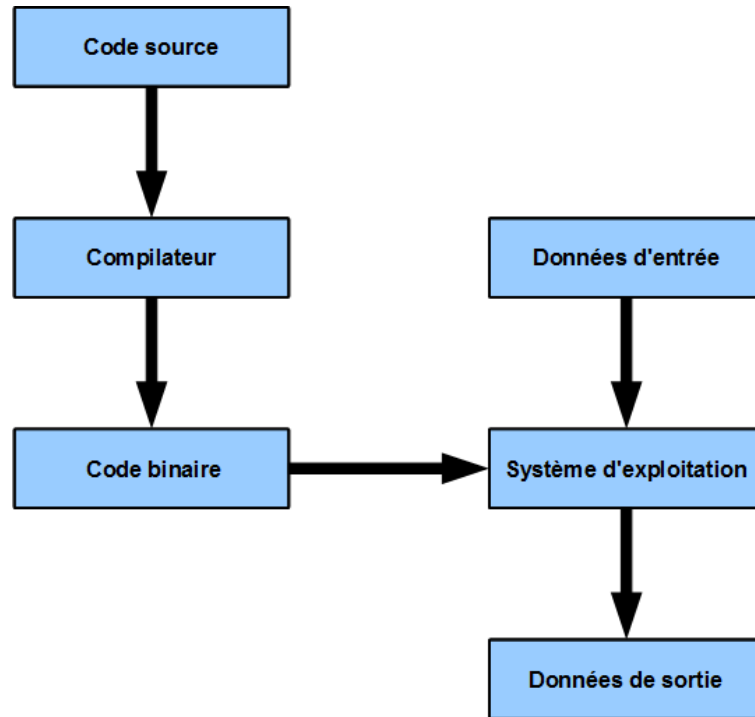


Figure 2: Compiled languages basics

The central difference between compiled and interpreted is as follows: where the compiler translates once and for all a source code into an executable independent file, the interpreter is required each time the interpreted program is started, to translate the source code into machine code as it goes along. This on-the-fly translation is the primary cause of the slowness of the so-called interpreted languages. Thus, despite improvements in interpreters and processors, compiled programs are still considered to be faster than interpreted programs, and are therefore more suitable for projects where computational speed is essential (for example, performing thousands of preventive tests during the takeoff of a space shuttle).

In fact, interpreted languages were for a long time poorly perceived by developers, until their use ended up having more advantages than disadvantages. The constant improvement of processors has overcome the problem of extreme slowness, the ease of evolution of an interpreted program has changed the vision of busy developers, and the speed of implementation of small programs has made them essential in everyday life, especially for Web applications.

2.2 Uses of programming languages

The question of which language is best is one that consumes a lot of time and energy among computer professionals. Every language has its strengths and weaknesses. For example, FORTRAN is a particularly good language for processing numerical data, but it does not lend itself very well to organizing large programs. Pascal is very good for writing well-structured and readable programs, but it is not as flexible as the C

programming language. C++ embodies powerful object-oriented features, but it is complex and difficult to learn. The choice of the language mainly depends on what you want to code.

2.2.1 Web development

If you're interested in building websites there are two intertwining parts to look into.

First, there's front-end development, which is the part of web development that creates the application that runs on your browser and adjusts the styling, the colors, the interactions. It's basically concerned with what the user of a website sees. You are reading this blog on some screen which is shown to you by front-end code. Front-end basics start with HTML and CSS with use of JavaScript. Javascript has become one of the most dominant languages in the last few years for front-end work.

The other part for creating websites is back-end development, which is related to the server, the computer that runs the website software and serves it to the world. It's mostly concerned with routing, which pages to deliver to the user when they visit a certain URL, it also communicates with the database that stores the website's information and sends this data over to the user. Back-end development is where the magic happens and there are many options to choose from when it comes to a programming language, you can stick to Javascript just like in front-end development, or go with PHP, Ruby, C#, Elixir, Python, Erlang.

2.2.2 Game development

Game development is one of the most interesting tracks there is, many developers enjoy it and there are developers who develop games just for having fun. Creating games requires what's called a game engine, which is a software that is used as the infrastructure for building the game and defines what the game has and what it can do. If you're familiar with Epic Games and Fortnite, Epic Games is, in fact, a game engine and Fortnite is built upon it. The languages used in game development are mostly C++, C# since it requires a lot of memory optimization and fast performance to create rich graphics. It's not limited to C++ and C#, however, and it kind of is about which engine you're using and which platform you're targeting, Lua and Java are also very famous candidates in this industry.

2.2.3 Mobile development

Creating mobile applications is a little tricky, as there is more than one operating system for mobiles and the different operating system would require different languages for these applications. An operating system is the piece of software on your device that is responsible for dealing with the hardware of this device, it's the layer that sits between the application you create and the hardware, whether it's a mic or a touchscreen or GPS. The most two common operating systems are Android and IOS. Android is most commonly used in Samsung while IOS is used in Apple. To create Android apps, you'd need either Java or Kotlin, and for creating IOS applications you'd need Objective-C or Swift.

3 Most popular languages overview









Top 10 Programming Languages										
										
Paradigm	Multi-paradigm: object-oriented, imperative, functional, procedural, reflective	Imperative (procedural), structured	Multi-paradigm: object-oriented (class-based), structured, imperative, generic, reflective, concurrent	Multi-paradigm: procedural, functional, object-oriented, generic	Multi-paradigm: structured, imperative, object-oriented, event-driven, task-driven, functional, generic, reflective, concurrent	Multi-paradigm: array, object-oriented, imperative, functional, procedural, reflective	Multi-paradigm: object-oriented (prototype-based), imperative, functional, event-driven	Imperative, object-oriented, procedural, reflective	Compiled, concurrent, imperative, structured	Multi-paradigm: protocol-oriented, object-oriented, functional, imperative, block-structured
Designed by	Guido van Rossum	Dennis Ritchie	James Gosling	Bjarne Stroustrup	Microsoft	Ross Ihaka and Robert Gentleman	Brendan Eich	Rasmus Lerdorf	Robert Griesemer, Rob Pike, Ken Thompson	Chris Lattner and Apple Inc
Developer	Python Software Foundation	Dennis Ritchie & Bell Labs (creators), ANSI X3J11 (ANSI C), ISO/IEC	Sun Microsystems (now owned by Oracle corporation)	Bell Labs	Microsoft	R Core Team	Netscape Communications Corporation, Mozilla Foundation, Ecma International	The PHP Development Team, Zend Technologies	Google Inc.	Apple Inc
First appeared	20 February 1991 (26 years ago)	1972 (45 years ago)	May 23 1995 (22 years ago)	1983 (34 years ago)	2000 (17 years ago)	August 1993 (24 years ago)	December 4, 1995 (21 years ago)	June 8, 1995 (22 years ago)	November 10, 2009 (7 years ago)	June 2, 2014 (3 years ago)
Typing discipline	Duck, dynamic, strong	Static, weak, manifest, nominal	Static, strong, safe, nominative, manifest	Static, nominative, partially inferred	Static, dynamic, strong, safe, nominative, partially inferred	Dynamic	Dynamic, duck	Dynamic, weak, gradual (as for PHP 7.0.0)	Strong, static, inferred, structural	Static, strong, inferred
Platform	Cross-platform	Cross-platform	Windows, Solaris, Linux, OS X	Linux, MacOS, Solaris	Common Language Infrastructure	UNIX platforms, Windows, MacOS	Cross-platform	Unix-like, Windows	Linux, macOS, FreeBSD, NetBSD, OpenBSD, Windows, Plan 9, DragonFly BSD, Solaris	Darwin, Linux, FreeBSD
Filename extensions	.py, .pyc, .pyo (prior to 3.5), .pyw, .pyz (since 3.5)	.c, .h	.java, .class, .jar	.cc, .cpp, .C, .c++, .h, .hh, .hpp, .hxx, .h++	.cs	.r, .R, .RData, .rds, .rda	.js	.php, .phtml, .php3, .php4, .php5, .php7, .phps	.go	.swift

Figure 3: <https://dzone.com/articles/top-10-programming-languages-in-2017>

3.1 Web development

3.1.1 Trendings

The most popular (i.e., the most visited) websites have in common that they are dynamic websites. Their development typically involves server-side coding, client-side coding and database technology. The programming languages applied to deliver similar dynamic web content however vary vastly between sites.

Websites	Front-end (Client-side)	Back-end (Server-side)
Google.com	JavaScript, TypeScript	C, C++, Go, Java, Python
Facebook.com	JavaScript	Hack, PHP, Python, C++, Java, Erlang, D, XHP, Haskell
WordPress.com	JavaScript	PHP
Pinterest	JavaScript	Python (Django), Erlang
Linkedin.com	JavaScript	Java, JavaScript, Scala
Wikipedia.org	JavaScript	PHP, Hack

Table 1: Programming languages used in most popular websites [10]

Below is a graph highlighting the most popular programming languages according to the number of pull requests on Github which appears to be a relevant indicator in that case.

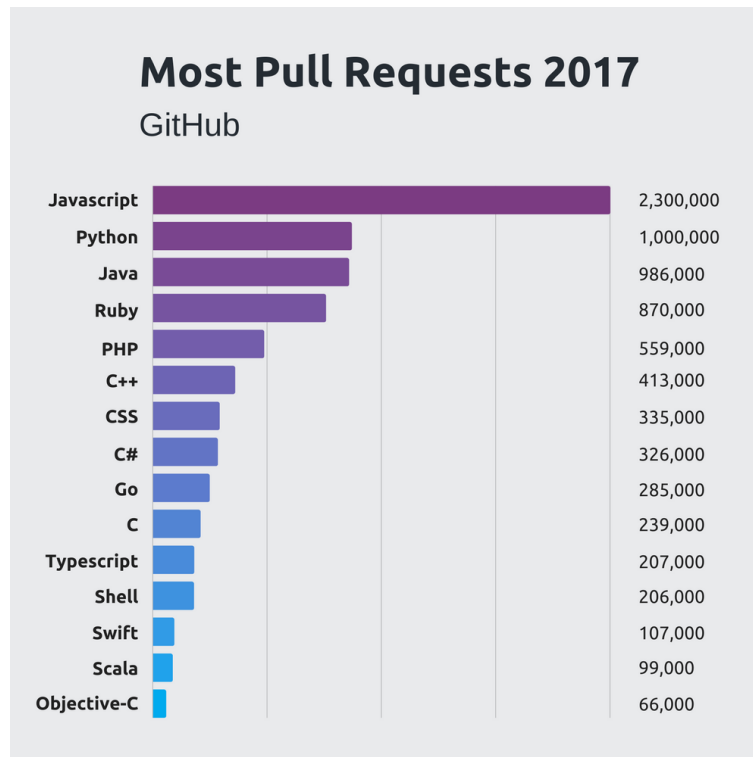


Figure 4: Most pull requests in 2017 - source: [Github.com](https://github.com)

3.1.2 Front-end development - client-side



JavaScript is a scripting programming language mainly used in interactive web pages but also for servers with the use (for example) of Node.js. It is an object-oriented language with a prototype, i.e. the language's bases and main interfaces are provided by objects that are not class instances, but are each equipped with builders to create their properties, and in particular a prototyping property that allows to create customized heir objects. In addition, the functions are first-class objects. The language supports the object, imperative and functional paradigm.

JavaScript is the language with the largest ecosystem thanks to its npm dependency manager, with about 500,000 packages in August 2017. With HTML and CSS technologies, JavaScript is sometimes considered one of the core technologies of the World Wide Web. JavaScript allows interactive web pages, and as such is an essential part of web applications. A large majority of websites use it, and most web browsers have a dedicated JavaScript engine to interpret it, regardless of any security considerations that may apply.

For front-end developers, it's increasingly challenging to make up their minds about which JavaScript application framework to choose, especially when they need to build a single-page application. *A framework is, as its name suggests in English, a "framework". The objective of a framework is generally to simplify the work of IT developers (coders if you prefer), by offering them a "ready to use" architecture that allows them not to start from scratch with each new project.*

To simplify this choice of a JS framework for client-side development, we should reduce our options to several top solutions. For client-side development, our list of JavaScript frameworks includes React, Angular

(Angular 2 or higher), Vue, Backbone, and Ember.



Figure 5: Top JS frameworks. For more information visit this [website](#) [1].

3.1.3 Back-end development - server-side

Concerning back-end development, there are many commonly used languages and each of these languages has several frameworks: the choice for a developer is therefore really complicated. This chapter aims to provide a non-exhaustive list of the most commonly used languages and some of their most popular frameworks [4].

- **Python** is an understood high-level programming language for general-purpose programming. Its a style philosophy that emphasizes code readability ,notable using significant white space. It constructs clear programming on each tiny and enormous scales. It is a object oriented scripting language ,the run time speed of this language is 71.90secs and the memory utilized is 2.80mb per sec. For turning into good at all-in-one language, you ought to begin learning Python language that has the power to expand internet apps, information analysis, user interfaces, and for additional frameworks are also accessible for these tasks. Python is employed by larger firms principally which will appraise that can evaluate huge data sets.

Advantages	Disadvantages
Extensive librairies	Speed limitations
Improved productivity	Undeveloped data base access layer
Free and open source	Design restriction

Table 2: Python overview



Django is a Python framework for the web. It is not the only one in its category, we can count other Python frameworks of the same kind like web2py, TurboGears, Tornado or Flask. However, it has the merit of being the most exhaustive, automating a good number of tasks and having a very large community. Today, it has become very popular and is used by companies around the world, such as Instagram, Pinterest, and even NASA.

Django applies his "Don't repeat yourself" philosophy in several ways. For example, an administration of all data can be automatically generated, while being very easily adaptable. Interaction with a database

is done through a set of specialized and very practical tools, so it is useless to waste time writing queries directly to the database, because Django does it automatically. In addition, the framework includes various functionalities such as a member space, or a library allowing the translation of your web application into several languages. Obviously, Django has the advantages of all frameworks in general. It is supported by an active and experienced community, which regularly publishes new versions of the framework with new features, bug fixes, etc.

- **PHP** is additionally referred to as hypertext preprocessor could be a service-side scripting language designed for internet development, however its is also used as a general-purpose programming language. It had been created by Rasmus Lerdorf in 1994. The web developers ought to study Hypertext Preprocessor, a widely known programming language. With the assistance of PHP, you can enlarge a web app terribly quickly and effortlessly. The run time employed by the compiler to execute the logic is 27.64 secs and the memory used is 2.57mb. PHP is that the actual foundation of many strong content management systems, as an instance Word Press.

Advantages	Disadvantages
User friendly interference	Desire a lot of manual work
Fast access to data base	Additional storage is required
Extremely helpful text process options	It has no formal error handling mechanisms

Table 3: PHP overview

- In addition to being a front-end language, **JavaScript** can also be used for back-end development. Back-end developers use a type of JavaScript called Node.js for back-end work. The Node.js framework allows a developer to handle data updates from the front end and build scalable network applications able to process many simultaneous user requests, amongst other things. All of this means that JavaScript is a crucial language to add to your web developer's toolkit, whether you plan to specialize in front end, back end, or full stack development.
- **Ruby** could be a understood ,dynamic ,reflective and general purpose scripting programming language. It supports multiple programming paradigms, including practical ,object-oriented and imperative. Its automatic memory management. The run time speed for executing the logic is 59.34secs and therefore the memory used by the compiler is 3.97mb/sec. This can be learned simply, and also very strong and clear-cut. If you've less time in hand and still wish to craft any project, then you can definitely use Ruby language. This programming language is applied massively for web programming and hence turned out to be the ideal selection for the beginner firms. Rails is the most popular Ruby framework.

Advantages	Disadvantages
Rails provides fantastic tooling that helps you deliver more option in less time	Runtime speed on ruby on rails is slow
The ruby community is huge in to testing and test automation	Active record is used heavily with in ruby on rails world and is hard dependency for many of the ruby gems
There's a gem for just about anything you can think of	It can be hard to find smart documentation

Table 4: Ruby overview

3.2 Mobile development

3.2.1 iOS

iOS app development requires a deep understanding of the Swift programming language, and familiarizing yourself with the many valuable features of Xcode, Apple's integrated development environment (IDE) [7].

3.2.2 Android

Android is a mobile operating system, meaning that, like Windows or Linux, it is a large program, composed of small programs, that allows you to run other software. For example, Windows allows you to run Internet Explorer, and to do so, it must make the link between the mouse and the cursor on the screen, between the keyboard and the input fields, etc. And with the explosion in smartphone sales in recent years, Android has become an important part of the daily lives of millions of people, to the point that it is the mobile operating system with the most applications in circulation. Android applications are almost mainly coded in Java [2].



When you think of Android, you immediately think of Google, and yet you have to know that this multinational company is not the initiator of the project. Moreover, it is not even the only one to contribute full-time to its development. Originally, "Android" was the name of an American SME, Android Incorporated, founded in 2003 and acquired by Google in 2005, which had the firm intention of entering the mobile products market. Android's objective was to develop a more intelligent mobile operating system, which would not only allow sending SMS and transmitting calls, but would also allow the user to interact with his environment (including his geographical location). Its main competitors at the time were Symbian and Windows Mobile. This is why, contrary to popular belief,

it is not possible to say that Android is a response from Google to Apple's iPhone, since its existence was only revealed two years later.

It was in 2007 that things got worse. At that time, manufacturers all designed a specific operating system for their phones, and there was no common basis between the mobile operating systems of two different manufacturers. This system made it difficult to easily develop applications that would fit all phones, especially between manufacturers, since the base was completely different. A developer was rather specialized in a particular system and had to settle for low-level languages like C or C++. In addition, manufacturers were ensuring that they delivered very small development libraries in order to hide their trade secrets. In January 2007, Apple unveiled the iPhone, a phone that was simply revolutionary at the time, capable of going on the Internet, playing videos, etc. The announcement is a disaster for other manufacturers, who must align themselves with this new competition. The problem is that to reach the level of iOS (iPhone OS, the operating system for iPhone), it would have taken years of research and development for each manufacturer...

This is why the Open Handset Alliance was created in November 2007, with 35 companies operating in the mobile world, including Google. The purpose of this alliance is to develop an open source system (i.e. one whose source code is accessible to all) for mobile operation and thus compete with proprietary systems, in particular iOS. This alliance has Android as its star software, but it is not its only activity. The OHA currently has 80 members.



Figure 6: Open Handset Alliance

The philosophy and benefits of Android

- **Open source**
- **Free:** Android is free, both for you and the manufacturers. However, to post your applications on the Play Store, it will cost you around \$25.
- **Easy to develop:** Many available APIs that make work much easier and faster. *An API, or "programming interface", is a set of rules to follow in order to be able to communicate with other applications. For example, one can ask Google Maps to show the map of Paris if one knows how to ask them (which methods are to be used with which parameters and what they return).*

3.3 Java and Kotlin



Kotlin was designed by programmers from JetBrains to add some modern features to Java mobile development. It has definitely gained momentum after being announced as an official programming language for Android at Google I/O in 2018. Google has also internally switched to using Kotlin instead of Java on Android. Kotlin is an open source, statically typed language based on the Java Virtual Machine (JVM), but you can also compile it to JavaScript or native to build code that can run on iOS. All it takes is installing the Kotlin Plugin and letting it configure your project.

Pros of Kotlin

- Kotlin has a lot of traction in Android development
- Kotlin is not only able to compile to almost every platform including Android, JVM, JavaScript, and native - using the Kotlin Multiplatform framework you can extract one common codebase that will target all of them at the same time

- Switching from Java to Kotlin is easy
- It supports modern programming concepts like extension functions, higher-order functions, delegates, and more out-of-the-box to help devs build clean APIs
- Kotlin provides a built-in null safety support which is a lifesaver, especially on Android, which is full of old Java-style APIs
- It's way more concise and expressive than Java, which means less room for error
- Kotlin is compatible with all Java libraries and frameworks
- Devs can benefit from a rapidly growing collection of open source projects on GitHub, many books, learning resources, and online courses
- Adopting Kotlin doesn't cost anything (except for learning and training)

Cons of Kotlin

- Rather steep learning curve when switching entire teams to Kotlin due to the language's concise syntax (both a blessing and a challenge)
- Slower compilation speed than Java
- Relatively smaller developer community compared to Java
- Kotlin, being a highly declarative language, sometimes tends to generate great amounts of boilerplate in corresponding JVM bytecode



Java is the favourite of many developers when it comes to Android app development – mainly because Android itself used to be written in Java. Developed by Sun Microsystems (now the property of Oracle), Java is an object-oriented programming language that boasts the title of the second most active language on GitHub. And no wonder – it's been around for over 20 years, and its popularity only seems to grow.

Pros of Java

- Easy to learn and understand
- Flexible – you can run it in a browser window or a virtual machine. This comes in handy when you reuse code and update software
- Android relies on Java – the Android SDK contains many standard Java libraries
- Java has a large open-source ecosystem
- Java apps are more compact and are easier to be optimized when better performance is required. Also, in comparison to Kotlin, Java apps tend to be lighter (even in full analogue) as Java is a more imperative language

Cons of Java

- Java has limitations that cause problems with Android API design
- As a verbose language, Java requires writing more code, which carries a higher risk of errors and bugs

- It's slower in comparison to many other languages and requires a lot of memory

As a conclusion, it seems for general-purpose programming, Java gains the upper hand. On the flip side, more and more developers and organizations are adopting Kotlin for rapidly developing Android applications. Both Java and Kotlin have their own advantages over the other. The debate about which of the two is the greatest has just started, and it is not likely to be over anytime soon.

3.4 Cross platform applications

There are two types of mobile applications: native applications and cross-platform applications. The vast majority of mobile applications produced were native applications

Native applications are applications specifically developed for an operating system. The most popular operating systems for mobile phones are iOS and Android. If you want to develop an application that is native and compatible with iOS and Android, you will need to develop two completely different applications: one for iOS, in Swift or Objective-C language, and a second for Android, in Kotlin or Java language.

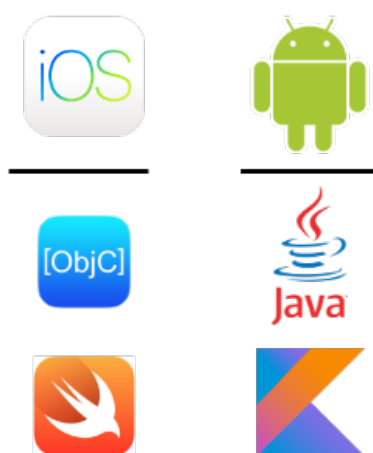


Figure 7: Native languages

It is in this context, where learning remained difficult and development times long, that cross-platforms applications have emerged [5]. Cross-platform applications, unlike native applications, are developed only once and are compatible with iOS and Android. The development of cross-platform applications requires frameworks. Among the best known are Ionic, PhoneGap, Xamarin and Titanium.



Figure 8: Cross-platform applications

We do not develop a cross-platform application with native languages, we often use simpler languages. For example, Ionic / PhoneGap / Titanium work with Javascript. Xamarin (Microsoft product) works with C#, a language not really considered simple. You will have understood it, no more need to learn a programming language per platform. We develop our application once, in a language, and the framework takes care of creating an iOS and Android compatible application for you. We go from two applications created (one iOS, one Android) to one; the time saving is enormous. However, cross-platform applications are considered less efficient and less fluid than native applications. In reality, there are often adjustments to be made for each

platform, which makes your project less and less clear and especially less and less unique.

Facebook releases the first version of React Native in 2015 and completely Open Source. Thanks to this openness to external contributors, the project is developing. In some time, what was an experimental project (coming out of a Hackaton all the same) becomes a real powerful and stable framework to deliver mobile applications on Apple and Google stores. React Native is positioned in the creation of cross-platform applications, so don't worry, you will only have to develop one application and only one language to learn, Javascript. While some cross-platform frameworks only use web components (Ionic in particular), React Native uses native mobile components. This means that, when you define a view in React Native, for example, on iOS your application displays a UIView and on Android an android.view.View, two native components. The operation is the same for all types of graphic elements: buttons, texts, lists, loading, etc. React Native converts all your elements into their native equivalent. It is thanks to this feature that your applications will be more efficient, more fluid and more similar to a mobile application.



React Native therefore works with native mobile components, which makes it different from Ionic and PhoneGap. However, this feature is not new. Titanium, a direct competitor of React Native, has also been using native mobile components for much longer.

However, React Native has distinguished itself from Titanium for several reasons:

- React Native is free of charge, from the beginning of your developments to the delivery of your mobile application on the blinds. Titanium is also free (for a single developer), but only since May 2017.
- React Native is Open Source. A large community has been built around the framework and has allowed it to grow.
- React Native is more flexible than Titanium. It is quite possible to use React Native in an existing native project, where Titanium would require you to start from scratch.

There are a multitude of cross-platform frameworks, often less known than those mentioned here. Examples include: Native Script, Framework 7, Mobile Angular UI, Onsen UI, etc. Of course, I haven't tested them all, but some of them are worth it. It's not quite up to date, but you can find information on this site comparing mobile cross-platform frameworks: <https://www.mobile-frameworks-comparison-chart.com/>.

4 Data structures

A database is an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques. The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system". Often the term "database" is also used to loosely refer to any of the DBMS, the database system or an application associated with the database.

Databases and DBMSs can be categorized according to the database models that they support (such as relational or XML), the types of computer they run on (from a server cluster to a mobile phone), the query languages used to access the database (such as SQL or XQuery), and their internal engineering, which affects performance, scalability, resilience, and security.

4.1 SQL vs. NoSQL

As any database administrator knows, the first question you have to ask yourself is whether to use a SQL or NoSQL database for your application. What's the difference between the two? [11] Also known as relational databases, SQL Databases define and manipulate data based on structured query language (SQL). These are most popularly used and useful for handling structured data that organizes elements of data and standardizes how they relate to one another and to different properties. On the contrary, NoSQL Databases, also known as non-relational databases (NRDB), allow you to store and retrieve unstructured data using a dynamic schema. NoSQL is popularly used for its flexible ability to create a unique structure, and can be document, graph, column, or even KeyValue organized as a data structure.

SQL has had a large lead over the non-relational alternatives for decades, but NoSQL is quickly closing the gap with popular databases such as MongoDB, Redis, and Cassandra. Though many organizations are choosing to migrate from legacy databases, such as Oracle, not all are moving to NoSQL way. Based on our findings, SQL still holds 60% with rising demand for systems such as PostgreSQL.

- SQL Database Use: 60.48%
- NoSQL Database Use: 39.52%

Category	Relational	Non Relational
Queries	Can handle more complicated queries (like joins, for example)	Better at simpler queries
Ease of scaling	Harder to scale (vertical scaling)	Easier to scale (horizontal scaling)
Data type	Structured data only	Unstructured data

Table 5: Comparison between SQL and NoSQL [6]

4.2 Most popular mobile databases

A mobile database is one that a mobile computing device can connect to over a wireless network. Mobile databases can further be classified as those that are physically separate from the central database server, those that reside on mobile devices, and those capable of communicating with a central database server or other mobile clients from remote sites and handling local queries without connectivity [9].

Mobile DBMSs should satisfy the following requirements:

- Small memory footprint

- Flash-optimised storage system
- Data synchronisation
- Security
- Low power consumption
- Self-management
- Strong coordination via embeddable features in applications

There are many mobile databases available in the market with many new ones also emerging, but it is the developer who has to choose the right database that satisfies all the requirements of the particular mobile app being created.

A few popular open source databases designed especially for mobile apps are discussed here:

- **SQLite** database is a small, highly reliable, embedded, fully-featured, compact and self-contained relational DBMS available as a public domain software package. As compared to other database management systems, SQLite is not a client-server database engine. It is regarded as a popular database, especially for application software like Web browsers and even for mobile app developers to store data from front-end mobile apps. SQLite offers an amazing set of tools to handle all sorts of data with ease and fewer constraints, compared to hosted and process based server relational databases.

Features:

- **Serverless:** It doesn't require a separate server process or system to operate. Its library has direct access to storage files.
 - **Zero configuration:** No server is required, which means 'No setup'. Creating a SQLite database instance is as easy as opening a simple file.
 - **Cross-platform:** It is available for various platforms like Android, BSD, iOS, MAC, Solaris, Windows and VxWorks.
 - **Self-contained:** The entire database is contained in a single library and is integrated into the host application.
 - **Transactional:** SQLite transactions are fully ACID-compliant, which allows safe access from multiple processes.
 - **Reliability:** SQLite development is under consistent development and proper testing is done before launching new versions.
 - **SQL extensions:** SQLite provides a number of enhancements to SQL, which are not normally found in other database engines. These enhancements include the EXPLAIN keyword and manifest typing. It also provides statements like REPLACE and the ON CONFLICT clause that allows for adding control over the resolution of the constraint conflict. It also supports the ATTACH and DETACH commands.
 - Official website: <https://www.sqlite.org/>
- **Realm DB** is a relational database management system which is like conventional databases, data can be queried and filtered, interconnected, and persisted but also have objects which are live and fully reactive. Realm DB is developed by Realm and specially designed to run on mobile devices. Like SQLite, Realm is also serverless and cross-platform. It can be stored both on disk as well as in memory. Realm is much faster than SQLite, Realm has a reactive architecture, means it can be directly connected to UI, if data changes it will automatically refresh and appear on screen. One application can have multiple Realms, both local and remote and different permissions can be set for different users. Realm is available for Android, iOS, JavaScript etc.

- **ORMLite** is lighter version of Object Relational Mapping which provide some simple functionality for persisting java objects to SQL databases. It is ORM wrapper over any mobile SQL related DB.

ORMLite is used to simplify complicated SQL operations by providing flexible query builder. It also provides powerful abstract Database Access Object (DAO) classes.

ORMLite is helpful in big size applications with complex queries because it handles "compiled" SQL statements for repetitive query tasks. It also has support for configuring of tables and fields without annotations and supports native calls to Android SQLite database APIs.

But ORMLite does not fulfil all the requirements, like it is bulky as compared to SQLite or Realm, slower than SQLite and Realm but faster than most of the other ORMs present in market.

All in all ORMLite is a good SQLite replacement if application is big and complex in terms of DB usage.

- **Berkeley DB** is an open source high performance embedded DB that allow us to handle data in different ways. It was developed by Sleepycat Software but acquired by Oracle in 2006. It provides API for so many languages including Android and iOS.

Berkeley DB can handle data in many ways. It can be in relational way like SQLite (by replacing SQLite with its own library), or it can be in Key/Value pair data as byte arrays and supports multiple data items for a single key. It also supports java objects as data or it can also be XML documents. Different libraries provides different types of API to handle multiple formats but all packaged Berkeley DB.

Berkeley can work as relational DB as well as NoSQL DB (Depends on which library you are using).

Good thing about Berkeley DB is that the API provided by it are compatible with SQLite. So one can use Berkeley DB without rewriting whole code again. Combination of Berkeley and SQLite is considered faster and perform better in concurrent and single writing multiple reading operations.

Berkeley is relatively faster than SQLite but because of so many different features it is bulkier than any other discussed DBs. So if Size of the application is a criteria try to use some other DB, unless you want a feature exclusively provided by Berkeley DB.

- **Couchbase Lite** is powerful NoSQL embedded JSON database. It is a highly scalable DB with enterprise-level security.

Data in Couchbase Lite is stored as JSON documents. Each documents can have one or more attachments which is stored and loaded separately from documents.

Couchbase Mobile is the solution provided by Couchbase Lite for mobile applications. It is comprised of three different components: Couchbase Lite, an embedded NoSQL database, Sync Gateway. Couchbase is an offline first DB and sync with Cloud is needed or when network is available. Couchbase Lite runs locally on the device and persists data as JSON and binary format. All crud operations performed on local DB. Developer does not need to write sync code (if needed) to sync local DB with cloud, it is handled by Sync Gateway. Couchbase Lite comes with a conflict resolution mechanism that is quite similar to the one used by Git.

Another advantage of Couchbase Lite is that it provided native APIs for Android and iOS and plugins for Xamarin and PhoneGap.

So if there is a requirement of any NoSQL DB in mobile OS, Couchbase lite is the best bet as it is very fast, reliable and moderate in size.

In the end it totally depends on the requirements and feasibility for the application to choose which DB will fit in. But every mobile DB should fill most of the above mentioned requirements if not all.

5 Conclusion

This document provides a non-exhaustive description of the technologies currently in use, with an emphasis on tools useful for developing web and mobile applications. Thus, using this state of the art and the specifications that will indicate the major criteria to be met, we will write a report explaining our choices of languages and programming software as well as the choice of the database.

References

- [1] *5 best JavaScript frameworks for frontend (2019)*. <https://rubygarage.org/blog/best-javascript-frameworks-for-front-end>.
- [2] *Android*. <https://openclassrooms.com/fr/courses/2023346-creez-des-applications-pour-android>.
- [3] *Comparison between programming languages*. https://www.webopedia.com/TERM/P/programming_language.html.
- [4] “Comparison of Programming Languages; Review”. In: *International Journal of Computer Science and Communication* (Sept. 2018), pp. 113–122.
- [5] *Cross platform applications*. <https://openclassrooms.com/fr/courses/4902061-developpez-une-application-mobile-react-native/4902068-decouvrez-le-developpement-mobile-actuel>.
- [6] *Database Management System*. <https://opensourceforu.com/2018/04/open-source-dbms/>.
- [7] *Developing iOS Apps*. <https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsS>
- [8] *Differences between compiler and interpreter*. <https://opensourceforgeeks.blogspot.com/2013/03/difference-between-compiler-interpreter.html>.
- [9] *Most popular databases for mobile apps*. <https://blog.trigent.com/five-of-the-most-popular-databases-for-mobile-apps>.
- [10] *Programming languages used in most popular websites*. https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites.
- [11] *Relational versus Non-Relational databases*. <https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/>.