Guia Ágil

TechGuide - Alura

Android

Nivel 1

- [ ] **Kotlin - Fundamentals**:

- Orientação a Objetos com Kotlin (Properties, Data class, Companion Objects, Delegation)
- Kotlin is a cross-platform, object-oriented, functional, statically typed programming language. It compiles to the Java Virtual Machine and can also be translated to the JavaScript language and compiled to native code (via LLVM). It is the official language of Google's Android system.
- Understanding its syntax
- Knowing the primitive types
- Declaring and using variables and constants
- Using conditional structures (if, else)
- Using repetition structures and loops (while, for)
- Using functions, passing parameters and arguments
- Implementing methods and reusing them
- Null Safety (Eliminate the danger of null references)
- Exceptions and Throwables
- Coding conventions
- Manipulating Collections, arrays and lists

- Functional Paradigm Features
- OOP with Kotlin (Properties, Data class, Companion Objects, Delegation)

- [ ] **Android - Fundamentals**:

- Getting to know Kotlin, Java or C++, which are the languages to develop Android apps.
- Understanding how the Android SDK packages the App code and resources into an APK (Android Package) to run on the Android OS
- Knowing the input components of an App - Activity, Service, Broadcast Receiver and Content Provider
- Understanding the Android component life cycle and how it works - The Activity life cycle
- Activating App input components with Intents
- Knowing the manifest file and the main configuration items
- Understanding what the resources of an Android project are - source code, static resources, drawables, layout, mipmap, values etc
- Creating an Android project with Android Studio and running it on a physical or virtual device
- Knowing the jetpack libraries to ensure compatibility between Android versions

- [ ] **Object-oriented Programming Concepts**:

- Object-oriented programming (OOP) is a programming paradigm based on the concept of 'objects', which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). A common feature of objects is that procedures (or methods) are attached to them and can access and modify the object's data fields. Some of the main concepts are classes and instances, inheritance, and encapsulation.
- How objects work
- Creating and using constructors
- What classes are

- Creating and using Methods
- How encapsulation works
- What inheritance is
- What polymorphism is
- How interfaces work
- What abstractions are

- [ ] **Android - View system**:

- The View class represents the basic building block of user interface components in Android.
- Knowing the View and ViewGroup reference for defining a Layout
- Knowing the main ViewGroups to build layouts - LinearLayout, RelativeLayout, FrameLayout and ConstraintLayout
- Using the visual Layout editor to customize the layout, or use the XML file for editing
- Knowing the Preview of the visual editor and use it to prepare the screens and components of the App
- Using the Views from the Android SDK - TextView, Button, EditText, CheckBox, DatePicker, etc
- Creating custom Views to meet specific demands
- Customizing Views with Android or View properties - height, width, visibility, spacing adjustments
- Implementing dynamic content layouts with adapters - AdapterView and RecyclerView
- Integrating business code with screen layout - Calling and manipulating Views with View Binding
- Reacting to View events from listeners such as clicks, long clicks, scrolling, drag and drop

- [ ] **Android - Fragments**:

- A Fragment represents the behavior or a part of the user interface in a host Activity. You can combine multiple fragments into a single Activity to create a multi-panel UI and reuse a fragment in multiple activities. You can imagine a fragment as a modular section of an Activity, which has its own life cycle, receives its own input events and can be added or removed during the execution of the Activity.
- Understanding what a Fragment is
- How to use and reuse Fragments in the same Activity
- Understanding the reasons to use Fragments in Android projects
- Implementing layouts with multiple panels
- Migrating Android projects that only use Activities to use Fragments
- Handling transactions from the fragment manager

☐ **Android - App navigation**:

- Screen navigation in apps is fundamental to the user experience of Android Apps. For this it is essential to know the principles of Android navigation.
- Learning what the Navigation library is and how you can implement it, either for the Views system or for Jetpack Compose
- Integrating navigation with app components, either in tabbed navigation, or in visual component views depending on the screen
- Knowing what an Android back stack task is
- Knowing the app links - Deep Link, Web Links, and Android App Links

☐ **Jetpack Compose - Fundamentals**:

- Jetpack Compose is a tool that brings the proposal to create native Android interfaces with less code, faster and more beautiful your apps, it does this through the declarative approach.
- Creating an Android app from scratch using Jetpack Compose
- Buildinging Layouts from Composables
- Previewing Composables
- Managinging states, events, composing and recomposing

- Configuring Compose in an existing project and applying interoperability
- Working with forms
- Understanding the difference between XML and Compose
- Maintaining states using the MVVM pattern with ViewModel and StateFlow, understanding the life cycle

- [ ] **Android - Persistence**:

- The concept of "data persistence" refers to ensuring that the information inputted into an application will be stored in a medium where it can be retrieved consistently. In other words, they are permanent records that are not lost when the session is closed.
- In Android we can persistently store App-specific or shared files, primitive type information with preferences, and structured database data.

- [ ] **Android - Gradle**:

- Gradle and the Android plug-in for Gradle(AGP) provide a flexible way to build, build, manage, and package your Android app or library.
- What is a build tool
- Learning the structure of an Android project as a multi-module Gradle project
- What is Gradle for and how to use it
- What are dependencies and how to use them

- [ ] **Android - Image loading**:

- Images come in different sizes and shapes. In most cases, they are larger than needed for the UI of an app. Since we are working with limited memory, Android uses a bitmap decoding technique, to avoid all the work of using this we have several libraries that facilitate this process.
- What is Bitmap
- Knowing the image loading libraries (Glide, Picasso, Coil, and others) and using them
- Advantages and disadvantages of the image loading libraries

Nivel 2

- [ ] **Android - Permissions**:
  - The app's permissions help support user privacy by protecting restricted data such as system statuses and the user's contact data, there are also restricted actions such as connecting to a paired device and recording audio or using a camera.
  - Understanding what permissions are
  - Learning about the different types of permissions
  - Special permissions
  - Best practices
- [ ] **Kotlin - Asynchronous**:
  - In asynchronous programming, the functions are not executed in order. We can interrupt the code to get some other information needed to continue execution. This means that the code waits for another part of the code, and while it waits, it can execute the other parts.
  - Learning the possibilities to run code asynchronously on Android
  - Getting to know the java.concurrent package and its solutions
  - Using Coroutines as a solution for asynchronous code in Kotlin
  - Understanding and use reactive frameworks like LiveData, Flow, StateFlow, RX, etc
- [ ] **Kotlin - Communication with APIs**:
  - An API is an interface that software developers use to programmatically interact with software components or resources outside of their own code. An even simpler definition is that an API is the part of a software component that is accessible to other components.
  - Performing HTTP requests to communicate with online services such as REST API is fundamental to most Android apps. As such, it is important to know the main tools and techniques needed for this type of functionality.

- Knowing the most famous libraries in the Android world for making requests - Retrofit, Ktor or Volley
- Setting up requests to execute asynchronously
- Converting objects to JSON and vice-versa

- [ ] **Android - System resources**:

- The smartphones that are under the Android platform, for the most part, have several system-specific resources, such as cameras, sensors, gps, and others.
- Learning what these features are and how to use them

- [ ] **Kotlin - Dependency Injection**:

- Dependency Injection is a design pattern in which a class requests dependencies from external sources instead of creating them.
- When writing code in Android projects, it is often the case that a feature uses library code, such as Room to save data or Retrofit to make requests to the REST API. These libraries are known as dependencies in our codes, precisely because of their necessity to perform the expected action. Using these libraries easily anywhere in the app can have its challenges, such as providing unique instances and performing all the necessary configuration for correct operation. For this, we use dependency injection tools that make our job easier.
- Learning how to use the dependency injection technique and one of the most common Android tools for this - Hilt, Dagger or Koin

- [ ] **Android - Testing**:

- Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include preventing bugs, reducing development costs and improving performance.
- Using unit tests
- Using integration testing
- Using instrumented testing

- Using mocks to facilitate test implementation with dependencies
- Implementing tests with Dependency Injection tools such as Hilt

Nivel 3

- [ ] **Android - Architecture**:
  - The Android architecture guide covers practices and recommended architecture for building robust apps with high production quality. Organizing your code base into contained and flexibly coupled parts (Modularization).
  - Learning what the MVVM (Model-View-ViewModel) architecture is
  - UI, data and domain layers
  - Data flow within the app
  - How to use the ViewModel
  - Architecture recommendations, repository pattern, offline-first
  - State management
  - Dependency injection
  - Improving app user experience with ViewModel
  - Learning what modularization is
  - Benefits of modularization
- [ ] **Android - Core App quality**:
  - When writing an Android App, there are a number of requirements to ensure an expected quality, such as compatibility of the App with the navigation expected by the Android system, or the gestures, etc.
  - The requirements are classified as - Visual experience, Functionality, Performance and stability, Privacy and security, Google Play, and Testing procedure
  - Fulfilling all or almost all requirements means that the App has a higher quality for the users
- [ ] **Android - CI/CD**:

- CI/CD is short for Continuous Integration/Continuous Delivery. It is a software development practice that aims to make code integration more efficient through automated builds and testing.
- When implementing new App functionality, we need to ensure that all deliveries will work correctly. To do this, we can use continuous integration and continuous delivery techniques, this way we speed up the evolution of the App and try to ensure the expected behaviors at the same time.
- Knowing one of the tools to do continuous delivery, such as Firebase Test Lab, Jenkins, GitHub Actions, etc

☐ **Android - App optimization**:

- When generating an App, there are some optimization details to make it fast, smaller and optimized, such as removing unnecessary code, obfuscation that reduces code names, and optimization that applies more aggressive strategies to further reduce the app.
- Learning how to enable each optimization in the Android plugin for Gradle
- Using proguard to perform a more aggressive optimization
- Activating multidex so that the app will be able to get more than 64000 methods and be able to use the optimization techniques
- Knowing and using helper tools to increase the performance of the App

☐ **Java - Fundamentals**:

- Java is a widely-used programming language for coding web applications. Java is a multi-platform, object-oriented, and network-centric language that can be used as a platform in itself. It is a fast, secure, reliable programming language for coding everything from mobile apps and enterprise software to big data applications and server-side technologies.
- Knowing the primitive types
- Declaring variables, considering the different types
- Using conditional structures ('if', 'else')
- Knowing the assignment and comparison operators
- Using repetition structures and loops ('while', 'for')

- Using functions, passing parameters and arguments
- Manipulating methods
- Manipulating arrays and lists
- Getting data from an API
- Making asynchronous 'Future' calls, etc
- Creating constructors

☐ **Android - Debugging**:

- During the development of an app, it is quite common to have unexpected bugs, as well as a lack of understanding of the reason for the bug. To speed up the analysis and investigation of problems or unexpected behavior in the app, we need to learn how to debug an Android app.
- Learning how to debug an app in Android Studio
- Learning how to enable debugging on physical devices
- Using logs to identify events
- Analyzing stack tracks
- Inspecting layout, system resources like processor, memory, and network
- Debugging the app database and pre-compiled APK files
- Analyzing the build with APK parser

☐ **Android - Security and Monitoring**:

- As your project grows, it becomes more visible, attracting both a larger audience and potential attackers. Application security, from the earliest stages of development, is crucial to protecting the integrity of the project and the privacy of users. Adopting robust security practices and keeping up to date is essential to avoid threats and guarantee a safe experience for your company and its users
- Making sure you understand your project's needs
- Monitoring is fundamental to understanding how your application operates in production.

- Understanding the user flow and identifying possible funnels helps to improve your project strategy and can be a determining factor in the application's success.
- The ability to anticipate and predict problem scenarios before they affect the end user is a crucial factor.

Habilidade Auxiliar: Infrastructure and good practices

- [ ] **Git & GitHub - Fundamentals**:

- Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- GitHub is a hosting service for software development and version control using Git.
- Creating a repository
- Cloning a repository
- Committing, pushing and pulling to and from the repository
- Reversing a commit
- Creating branches and pull requests
- Handling merge and conflicts

- [ ] **HTTP - Fundamentals**:

- HTTP stands for Hyper Text Transfer Protocol. Communication between client computers and web servers is done by sending HTTP Requests and receiving HTTP Responses.
- Understanding the difference between HTTP verbs
- Testing requests and checking the status codes in the browser
- Learning how to make a HTTP request on the command line with WGET
- Downloading an image with WGET
- Performing a POST

- [ ] **JSON**:
  - JSON stands for JavaScript Object Notation. It is a text format for storing and transporting data.
  - Creating an object
  - Transforming an object into a string
  - Transforming a string into an object
  - Manipulating an object
- [ ] **Design Patterns**:
  - In software engineering, a Design Pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is a description or template for how to solve a problem that can be used in many different situations. Design Patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.
  - Getting familiarized with and applying the main Design Patterns
- [ ] **Command Line - Fundamentals**:
  - CLI is a command line program that accepts text input to execute operating system functions.
  - Knowing the most important commands
- [ ] **Cloud - Fundamentals**:
  - Cloud, or cloud computing, is the distribution of computing services over the Internet using a pay-as-you-go pricing model. A cloud is composed of various computing resources, ranging from the computers themselves (or instances, in cloud terminology) to networks, storage, databases, and everything around them. In other words, everything that is normally needed to set up the equivalent of a server room, or even a complete data center, will be ready to use, configured, and run.
  - Knowing the difference between IaaS, PaaS and SaaS
  - Knowing the largest cloud providers

- Specializing in a specific provider of your choice

- [ ] **SOLID**:

- SOLID has five principles that are considered best practices in software development that help programmers write cleaner code by separating responsibilities, reducing docking, easing refactoring, and encouraging code reuse.

- [ ] **Clean Architecture**:

- Clean architecture is a way of developing software, such that just by looking at the source code of a program, you should be able to tell what the program does.

- [ ] **Firebase**:

- Firebase is a Backend-as-a-Service (BaaS) app development platform that provides hosted backend services such as a realtime database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files.
- Understanding how to install Firebase
- Getting acquainted with Firebase documentation
- Learning about the Firebase tools available

Habilidade Auxiliar: UX & Design

- [ ] **Material Design**:

- Material Design is Google's open source design system, where it gives you components with certain usage patterns and some customization for your apps.
- User Experience Foundations
- Customizing your components
- Adaptive layouts

- [ ] **Design Systems**:

- A design system is a collection of reusable components, guided by clear standards, that can be assembled together to build applications.
- Creating and maintaining libraries that will be consumed and used as a standard for building a project

- [ ] **Color systems**:

- Defining a color palette that makes sense for a given interface

- [ ] **How to use Fonts**:

- Choosing the most appropriate font for a given project

- [ ] **Responsive Design**:

- Responsive web design (RWD) or responsive design is an approach to web design that aims to make web pages render well on a variety of devices and window or screen sizes from minimum to maximum display size to ensure usability and satisfaction.
- Adjusting your pages to the user's screen size
- Learning about Media queries
- Knowing the concept of Mobile first

- [ ] **Android - Acessibility**:

- Web accessibility is the elimination of barriers on the web. The concept assumes that websites and applications are designed so that all people can effectively perceive, understand, navigate, and interact with pages, including people with accessibility needs, such as those with vision impairments, color blindness, hearing impairments, motor impairments, cognitive disabilities, and many other types of disabilities.
- Increasing the visibility of text
- Using large, simple controls
- Describing each UI element

- [ ] **Figma - Fundamentals**:

- Figma is a collaborative web application for interface design. The feature set of Figma focuses on user interface and user experience design, with an emphasis on real-time collaboration, utilising a variety of vector graphics editor and prototyping tools.
- Creating page layouts and components
