Guia Ágil

TechGuide - Alura

Android

Nivel 1

- [ ] **Kotlin - Fundamentos**:

- Kotlin é uma linguagem de programação multiplataforma, orientada a objetos, funcional e estaticamente tipada. Ela compila para a máquina virtual Java e também pode ser traduzida para a linguagem JavaScript e compilada para código nativo (via LLVM). É a linguagem oficial do sistema Android da Google.
- Entender a sua sintaxe
- Conhecer os tipos primitivos
- Declarar e usar variáveis e constantes
- Usar estruturas condicionais (if, else)
- Usar estruturas de repetição e laços (while, for)
- Usar funções, passando parâmetros e argumentos
- Implementando métodos e reutilizando eles
- Null Safety (Elimine o perigo de referências nulas)
- Exceptions e Throwables
- Convenções de código
- Manipular Coleções, arrays e listas

- Orientação a Objetos com Kotlin (Properties, Data class, Companion Objects, Delegation)
- Receber dados de uma API

- [ ] **Android - Fundamentos**:

- Conhecer Kotlin, Java ou C++ que são as linguagens para desenvolver Apps Android.
- Entender como o SDK do Android empacota o código e recursos do App em um APK (Android Package - Pacote Android) para rodar no SO do Android
- Saber os componentes de entrada de um App - Activity, Service, Broadcast Receiver e Content Provider
- Compreender os ciclos de vida de componentes do Android e como funcionam - O ciclo de vida de uma Activity
- Ativar componentes de entrada do App com Intents
- Conhecer o arquivo de manifesto e os principais itens de configuração
- Entender quais são os recursos de um projeto Android - código fonte, recursos estáticos, drawables, layout, mipmap, values etc
- Criar um projeto Android com o Android Studio e rodar em um dispositivo físico ou virtual
- Conhecer bibliotecas do jetpack para garantir a compatibilidade entre as versões do Android

- [ ] **Conceitos de Orientação a Objetos**:

- A Programação Orientada a Objetos é um paradigma de programação de software baseado na composição e interação entre diversas unidades chamadas de 'objetos' e as classes, que contêm uma identidade, propriedades e métodos). Ela é baseada em quatro componentes da programação: abstração digital, encapsulamento, herança e polimorfismo.
- Como funcionam objetos
- Criar e utilizar construtores
- O que são classes

- Criar e utilizar métodos
- Como funciona encapsulamento
- O que é herança
- O que é polimorfismo
- Como funcionam interfaces
- O que são abstrações

- [ ] **Android - Sistema de View**:

- A classe View representa o bloco básico de construção dos componentes de interface do usuário em Android.
- Conhecer a referência View e ViewGroup para definir um Layout
- Saber as principais ViewGroups para construir layouts - LinearLayout, RelativeLayout, FrameLayout e ConstraintLayout
- Usar o editor visual de Layout para personalizar a tela, ou então, utilizar o arquivo XML para a edição
- Conhecer o Preview do editor visual e usá-lo para preparar as telas e componentes do App
- Utilizar as Views do SDK do Android - TextView, Button, EditText, CheckBox, DatePicker etc
- Criar Views personalizadas para atender demandas específicas
- Personalizar as Views com propriedades do Android ou da própria View - Ajustes de altura, largura, visibilidade, espaçamento
- Implementar Layouts de conteúdos dinâmico com adaptadores - AdapterView e RecyclerView
- Integrar código de negócio com o layout da tela - Chamar e manipular as Views com o View Binding
- Reagir a eventos das Views a partir de listeners como cliques, cliques longos, rolagem, arrastar e soltar

- [ ] **Android - Fragments**:

- Um Fragment representa o comportamento ou uma parte da interface do usuário em uma Activity hospedeira. É possível combinar vários fragmentos em uma única Activity para criar uma IU de vários painéis e reutilizar um fragmento em diversas atividades. Você pode imaginar um fragment como uma seção modular de uma Activity, que tem o próprio ciclo de vida, recebe os próprios eventos de entrada e que pode ser adicionada ou removida durante a execução da Activity.
- Entender o que é um Fragment
- Como usar e reutilizar Fragments na mesma Activity
- Entender os motivos para utilizar Fragments em projetos Android
- Implementar layouts com múltiplos painéis
- Migrar projetos Android que utilizam apenas Activities para utilizar fragments
- Lidar com transações do gerenciador de fragments

☐ **Android - Navegação no App**:

- A navegação de telas em Apps é fundamental para a experiência de uso de Apps Android. Para isso é essencial conhecer os princípios de navegação do Android.
- Aprender o que é a biblioteca Navigation e como é possível implementá-la, seja para o sistema de Views ou para o Jetpack Compose
- Integrar a navegação com componentes do App, seja em navegação de abas, ou na visualização de componentes visuais dependendo da tela
- Saber o que é uma task de back stack do Android
- Conhecer os App links - Deep Link, Web Links e Android App Links

☐ **Jetpack Compose - Fundamentos**:

- O Jetpack Compose é uma ferramenta que trás a proposta de criar interfaces nativas Android com menos código, mais rápido e deixa seus apps mais bonitos, ele faz isso através da abordagem declarativa.
- Criar um app Android do zero utilizando o Jetpack Compose
- Construção de Layouts apartir de composables

- Pré-visualizações de Composables
- Gerenciamento de estados, eventos, composição e recomposição
- Configurar o Compose em um projeto já existente e aplicar a interoperabilidade
- Trabalhar com formulários
- Entender a diferença do XML para o Compose
- Manter estados utilizando o padrão MVVM com ViewModel e StateFlow, entendendo o ciclo de vida

☐ **Android - Persistência**:

- O conceito de "persistência de dados" refere-se a garantir que as informações inseridas na aplicação serão armazenadas em um meio em que possam ser recuperadas de forma consistente. Ou seja, são registros permanentes e que não são perdidos quando há o encerramento da sessão.
- No Android podemos armazenar de maneira persistente arquivos específicos ou para o App ou compartilhado, informações de tipo primitivo com preferências e dados estruturados com banco de dados.

☐ **Android - Gradle**:

- O Gradle e o plug-in do Android para Gradle(AGP) oferecem uma maneira flexível de compilar, criar, gerenciar e empacotar seu app ou biblioteca Android.
- O que é uma build tool
- Aprender a estrutura de um projeto Android como um projeto multi módulo do Gradle
- Para que serve o Gradle e como usá-lo
- O que são dependências e como utilizá-las

☐ **Android - Carregamento de imagens**:

- As imagens vem em tamanhos e formas diferentes. Na maioria dos casos, elas são maiores do que o necessário para a Interface de usuário (UI) de um app. Dado que estamos trabalhando com memória limitada, no Android é

usada uma técnica de decodificação de bitmaps, para evitar todo o trabalho de utilizar isso, temos diversas bibliotecas que facilitam esse processo.

- O que é Bitmap
- Conhecer as bibliotecas de carregamento de imagens (Glide, Picasso, Coil, entre oturas) e usá-las
- Vantagens e desvantagens das bibliotecas de carregamento de imagens

Nivel 2

- [ ] **Android - Permissões**:
  - As permissões do app ajudam a apoiar a privacidade do usuário protegendo dados restritos, como estados do sistema e os dados de contatos dos usuários, há também as ações restritas, como a conexão a um dispositivo pareado e a gravação de áudio ou uso de câmera.
  - Entender o que são permissões
  - Aprender sobre os diferentes tipos de permissões
  - Permissões especiais
  - Práticas recomendadas
- [ ] **Kotlin - Assíncrono**:
  - Na programação assíncrona as funções não são executadas em ordem. Com o assincronismo, podemos interromper do código para conseguirmos alguma outra informação necessária para a continuar a execução. Isso significa que o código espera por uma outra parte do código e, enquanto espera, executa as demais partes.
  - Aprender as possibilidades para rodar o código de maneira assíncrona no Android
  - Conhecer o pacote java.concurrent e as suas soluções
  - Utilizar Coroutines como uma solução de código assíncrono no Kotlin
  - Entender e usar estruturas reativas como LiveData, Flow, StateFlow, RX, etc
- [ ] **Kotlin - Comunicação com APIs**:

- Uma API é uma interface que desenvolvedores de software utilizam para programar a interação com componentes ou recursos de software fora de seu próprio código. Uma definição ainda mais simples é que uma API é a parte de um componente de software que é acessível a outros componentes.
- Realizar requisições HTTP para se comunicar com serviços online, como REST API é fundamental na maioria dos Apps Android. Sendo assim, é importante conhecer as principais ferramentas e técnicas necessárias para esse tipo de funcionalidade.
- Conhecer bibliotecas famosas no mundo Android para realizar requisições - Retrofit, Ktor ou Volley
- Configurar as requisições para executararem de maneira assíncrona
- Converter objetos para JSON e vice-versa

☐ **Android - Recursos do sistema**:

- Os smartphones que estão sob a plataforma Android, na sua grande maioria possuem diversos recursos específicos do sistema, tais como câmeras, sensores, gps, entre outros.
- Aprenda quais são e como utilizar esses recursos

☐ **Android (IA)**:

- Um Client-SDK para IA é uma ferramenta intermediária que simplifica a comunicação entre o seu aplicativo e serviços de IA remotos. Imagine como uma ponte que liga o seu aplicativo (que está local no seu dispositivo) a um servidor poderoso de IA (que geralmente fica na nuvem);
- IA On-Device, ou inteligência artificial no dispositivo, refere-se à execução de tarefas de IA diretamente no seu dispositivo pessoal, como smartphone, tablet ou smartwatch, em vez de depender de servidores remotos na nuvem;
- Aprender a traduzir textos automaticamente, reconhecer imagens, gerar respostas inteligentes e mais utilizando IA no seu app;
- Como acessar a API Gemini/GPT diretamente do app Android usando o SDK do cliente para Android;

- Como baixar e utilizar ferramentas de machine learning localmente nos apps;

- [ ] **Kotlin - Injeção de Dependências**:

- Injeção de Dependências é um padrão de projeto no qual uma classe solicita dependências de fontes externas ao invés de criá-las.
- Ao escrever códigos em projetos Android, é muito que uma funcionalidade utilize códigos de bibliotecas, como o Room para salvar dados ou o Retrofit para fazer requisições para REST API. Essas bibliteocas são conhecidas como dependências dos nossos códigos, justamente pela necessidade para realizar a ação esperada. Usar essas bibliotecas com facilidade em qualquer parte do app pode ter os seus desafios, como oferecer instâncias únicas e realizar toda a configuração necessária para o funcionamento correto. Para isso, utilizamos ferramentas de injeção de dependência que facilitam o nosso trabalho.
- Aprender a usar a técnica de injeção de dependências e alguma das ferramentas comuns no Android para tal - Hilt, Dagger ou Koin

- [ ] **Android - Testes**:

- O teste de software é o processo de avaliação e verificação de que um software realmente faz o que deveria fazer. Os benefícios dos testes incluem a prevenção de bugs, a redução dos custos de desenvolvimento e a melhoria do desempenho.
- Usar testes unitários
- Usar testes de integração
- Usar testes instrumentados
- Usar mocks para facilitar a implementação de testes com dependências
- Implementar testes com ferramentas de Injeção de Dependências como o Hilt

Nivel 3

- [ ] **Android - Arquitetura**:

- O guia de arquitetura Android aborda práticas e a arquitetura recomendada para a criação de apps robustos com alta qualidade de produção. Organizar sua base de código em partes contidas e acopladas com flexibilidade (Modularização).
- Aprender o que é e para que serve a arquitetura MVVM (Model-View-ViewModel)
- Camadas de IU, dados e domínios
- Fluxo de dados dentro do app
- Como utilizar o ViewModel
- Recomendações da arquitetura, padrão repositório, offline-first
- Gerenciamento de estado
- Injeção de dependências
- Melhorar a experiência dos usuários do app com o ViewModel
- Aprender o que é modularização
- Benefícios da modularização

☐ **Android - Critérios de qualidade do App**:

- Ao escrever um App Android, existe uma série de requisitos para garantir uma qualidade esperada, como por exemplo, a compatibilidade do App com a navegação esperada pelo sistema Android, ou então, pelos gestos, etc.
- Os requisitos são classificados como - Experiência visual, Funcionalidade, Desempenho e estabilidade, Privacidade e segurança, Google Play e Procedimento de teste
- Cumprir todos os requisitos, ou quase todos, significa que o App possui uma maior qualidade para os usuários.

☐ **Android - Entrega e integração contínuas (CI/CD)**:

- CI/CD é a abreviação de Continuous Integration/Continuous Delivery, traduzindo para o português "entrega e integração contínuas". Trata-se de uma prática de desenvolvimento de software que visa tornar a integração de código mais eficiente por meio de builds e testes automatizados.

- Ao implementar novas funcionalidades do App, precisamos garantir que todas as entregas irão funcionar corretamente. Para isso, podemos utilizar técnicas de integração e entrega contínua, dessa forma, agilizamos a evolução do App e tentamos garantir os comportamentos esperados ao mesmo tempo.
- Conhecer alguma das ferramentas para fazer a entrega contínua, como Firebase Test Lab, Jenkins, GitHub Actions, etc

☐ **Android - Otimização do app**:

- Ao gerar um App, existem alguns detalhes de otimização para torná-lo rápido, menor e otimizado, como por exemplo, a remoção de código desnecessário, a ofuscação que reduz o nome dos códigos e a otimização que aplica estratégias mais agressivas para reduzir mais ainda o app.
- Aprender a ativar cada otimização no plugin do Android para o Gradle
- Utilizar o proguard para realizar uma otimização mais agressiva
- Ativar o multidex para que o app seja capaz de obter mais de 64000 métodos e conseguir utilizar as técnicas de otimização
- Conhecer e utilizar ferramentas auxiliarem para aumentar o desempenho do App

☐ **Java - Fundamentos**:

- Java é uma linguagem de programação amplamente usada para codificar aplicações Web. Java é uma linguagem multiplataforma, orientada a objetos e centrada em rede que pode ser usada como uma plataforma em si. É uma linguagem de programação rápida, segura e confiável para codificar tudo, desde aplicações móveis e software empresarial até aplicações de big data e tecnologias do servidor.
- Conhecer os tipos primitivos
- Declarar variáveis, considerando os diferentes tipos
- Usar estruturas condicionais ('if', 'else')
- Conhecer os operadores de atribuição e comparação
- Usar estruturas de repetição e laços ('while', 'for')

- Usar funções, passando parâmetros e argumentos
- Manipular métodos
- Manipular arrays e listas
- Obter dados de uma API
- Criar construtores

- [ ] **Android - Depuração**:

- Durante o desenvolvimento de um app, é bastante comum a presença de bugs inesperados, como também, a falta de compreensão do motivo do bug. Para agilizar a análise e investigação de problemas ou comportamentos inesperados no app, precisamos aprender a depurar ou debuggar um app Android.
- Saber como executar um app em depuração no Android Studio
- Aprender a ativar a depuração em dispositivos físicos
- Utilizar Logs para identificar eventos
- Analisar stack track
- Inspecionar o Layout, recursos do sistema como processador, memória e rede
- Depurar o banco de dados do app e arquivos APKs pré-compilados
- Analisar o build com o analizador de APK

- [ ] **Android - Segurança e Monitoramento**:

- À medida que seu projeto cresce, ele se torna mais visível, atraindo tanto um público maior quanto potenciais invasores. A segurança do aplicativo, desde as fases iniciais de desenvolvimento, é crucial para proteger a integridade do projeto e a privacidade dos usuários. Adotar práticas de segurança robustas e se manter atualizado é essencial para evitar ameaças e garantir uma experiência segura para sua empresa e seu usuário
- Atente-se a entender as necessidades do seu projeto
- Monitoramento é fundamental para compreender como seu aplicativo opera em produção.

- Compreender o fluxo do usuário e identificar possíveis funis ajuda a aprimorar a estratégia do seu projeto e pode ser um fator determinante para o sucesso do aplicativo.
- A capacidade de antecipar e prever cenários de problemas antes que afetem o usuário final é um diferencial crucial.

☐ **Android - Implantação (Deployment)**:

- O deployment ou implantação é o processo de disponibilizar um aplicativo Android para os usuários. Existem várias maneiras de fazer isso, um deles é o uso da Play Store. Ela permite que os desenvolvedores distribuam seus aplicativos para um público global.
- Conhecer os conceitos básicos de deployment de aplicativos Android;
- Criar uma conta de Google Developer;
- Usar a Play Store para distribuir aplicativos Android para um público global.

Habilidade Auxiliar: Infraestrutura e boas práticas

☐ **Git e GitHub - Fundamentos**:

- Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência.
- GitHub é um serviço de hospedagem para desenvolvimento de software e controle de versão usando Git.
- Criar um repositório
- Clonar um repositório
- Fazer commit, push e pull de e para o repositório
- Reverter um commit
- Criar branches e pul requests
- Lidar com merge e conflitos

☐ **HTTP - Fundamentos**:

- HTTP significa Hyper Text Transfer Protocol. A comunicação entre computadores cliente e servidores web é feita enviando solicitações HTTP e recebendo respostas HTTP.
- Entender a diferença dos verbos HTTP
- Testar os requests e ver os status codes no navegador
- Saber fazer uma requisição HTTP na linha de comando com WGET
- Baixar uma imagem com WGET
- Fazer um post

☐ **JSON**:

- JSON significa JavaScript Object Notation (notação de objeto JavaScript). É um formato de texto para armazenar e transmitir dados.
- Criar um objeto
- Transformar um objeto em uma string
- Transformar uma string em objeto
- Manipular um objeto

☐ **Design Patterns**:

- Na engenharia de software, um "padrão de projeto" (Design Pattern em inglês) é uma solução geral e reutilizável para um problema que ocorre normalmente dentro de um determinado contexto de projeto de software.
- Conhecer e aplicar os principais Design Patterns

☐ **Linha de comando - Fundamentos**:

- CLI é um programa de linha de comando que aceita entradas de texto para executar funções do sistema operacional.
- Conhecer os principais comandos

☐ **Cloud - Fundamentos**:

- Cloud, ou computação em nuvem é a distribuição de serviços de computação pela Internet usando um modelo de preço pago conforme o uso. Uma nuvem é composta de vários recursos de computação, que

abrangem desde os próprios computadores (ou instâncias, na terminologia de nuvem) até redes, armazenamento, bancos de dados e o que estiver em torno deles. Ou seja, tudo o que normalmente é necessário para montar o equivalente a uma sala de servidores, ou mesmo um data center completo, estará pronto para ser utilizado, configurado e executado.

- Conhecer a diferença entre IaaS, PaaS e SaaS
- Conhecer os maiores provedores de cloud
- Especializar-se em algum provedor

- [ ] **SOLID**:

- O Solid possui cinco princípios considerados como boas práticas no desenvolvimento de software que ajudam os programadores a escrever os códigos mais limpos, separando as responsabilidades, diminuindo acoplamentos, facilitando na refatoração e estimulando o reaproveitamento do código.

- [ ] **Clean Architecture**:

- A Clean Architecture (Arquitetura Limpa) é uma forma de desenvolver software, de tal forma que apenas olhando para o código fonte de um programa, você deve ser capaz de dizer o que o programa faz.

- [ ] **Firebase**:

- O Firebase é uma plataforma de desenvolvimento de aplicativos Backend-as-a-Service (BaaS) que fornece serviços de backend hospedados, tais como banco de dados em tempo real, armazenamento em nuvem, autenticação, relatórios de falhas, aprendizado de máquina, configuração remota e hospedagem para seus arquivos estáticos.
- Entender como Instalar o Firebase
- Conhecer a documentação do Firebase
- Conhecer as ferramentas do Firebase disponíveis

Habilidade Auxiliar: UX & Design

- [ ] **Material Design**:
  - Material Design é um sistema de design de código aberto do Google, onde ele te passa componentes com determinados padrões de uso e certa personalização para seus apps.
  - Fundações de experiência de usuário
  - Personalizar seus componentes
  - Layouts adaptativos
- [ ] **Design System**:
  - Um Design System (sistema de design) é uma coleção de componentes reutilizáveis, guiados por padrões claros, que podem ser colocados juntos para construir aplicações.
  - Criar e manter bibliotecas que serão consumidas e usadas como padrão para a construção de um projeto
  - Design tokens
  - Estilos fundamentais
  - Construção de componentes
  - Microinterações
  - Documentação
- [ ] **Sistemas de cores**:
  - Definir uma paleta de cores que faça sentido para determinada interface
- [ ] **Como usar fontes**:
  - Escolher a fonte mais adequada para determinado projeto
- [ ] **Design Responsivo**:
  - Ajustar suas páginas para o tamanho da tela do usuário
  - Media queries

- Conhecer o conceito de Mobile first

- [ ] **Android - Acessibilidade**:

- Acessibilidade Digital é a eliminação de barreiras na Web. O conceito pressupõe que os sites e aplicativos sejam projetados de modo que todas as pessoas possam perceber, entender, navegar e interagir de maneira efetiva com as páginas, incluindo pessoas com necessidades de acessibilidades, como pessoas com problemas de visão, daltonismo, dificuldades auditivas, comprometimento motor, deficiência cognitivas e muitos outros tipos de deficiência.
- Aumentar a visibilidade do texto
- Usar controles grandes e simples
- Descrever cada elemento de IU

- [ ] **Figma - Fundamentos**:

- Figma é uma aplicação web colaborativa para design de interfaces. O conjunto de recursos do Figma se concentra na interface do usuário e no design da experiência do usuário, com ênfase na colaboração em tempo real, utilizando uma variedade de editores de gráficos vetoriais e ferramentas de prototipagem.
- Criar layouts de páginas e componentes
