DDM-II
Desenvolvimento para Dispositivos Móveis II

Prof. Anderson Vanin

2026

Apresentação das Competências, Habilidades e Bases Tecnológicas

Competências	Habilidades
1. Projetar aplicativos, selecionando linguagens de programação e ambientes de desenvolvimento.	1.1. Codificar aplicativos em tecnologia móvel.
	1.2. Utilizar ambientes de desenvolvimento mobile.
	1.3. Elaborar aplicativos com acesso a banco de dados.
	1.4. Construir leiaute de aplicativos dispositivos móveis.
	1.5. Utilizar recursos avançados do dispositivo (smartphones e tablets).

Bases Tecnológicas
Conectividade: • Consumo de APIs REST; • Comunicação TCP full-duplex (sockets); • Integração com dispositivos embarcados via Bluetooth. **Autenticação e autorização** **Recursos do dispositivo:** • Câmera; • Sensores; • Localização, orientação e mapas; • Telefonia e SMS. **Empacotamento e distribuição**

Prof. Anderson Vanin

Critérios de avaliação

- Assiduidade
- Participação em aula
- Entrega em dia de trabalhos e atividades
- Participação ativa em trabalhos desenvolvidos em grupo
- Avaliação individual
- Quando a avaliação contiver várias questões, uma nota máxima deve ser calculada (10) e ao final será atribuída uma menção com base na seguinte classificação:
  - De 0 a 5 ➔ **MENÇÃO I**
  - De 5,1 a 7 ➔ **MENÇÃO R**
  - De 7,1 a 8,5 ➔ **MENÇÃO B**
  - Acima de 8,5 ➔ **MENÇÃO MB**

Semana 1: O Cenário Mobile e Configuração

1. Introdução: O Dilema dos Sistemas Operacionais

Antes de escrevermos código, precisamos entender o problema fundamental do desenvolvimento móvel. O mercado é dominado por dois gigantes:

- **Android (Google):** Baseado em Linux, usa historicamente Java e Kotlin.
- **iOS (Apple):** Baseado em Unix (Darwin), usa historicamente Objective-C e Swift.

Antigamente, se uma empresa quisesse um app, ela precisava de **duas equipes**: uma de Android e uma de iOS. Isso dobrava o custo e o tempo. Para resolver isso, surgiram diferentes **arquiteturas de desenvolvimento**.

2. As Três Grandes Arquiteturas

A. **Desenvolvimento Nativo (Native)**

B. **Desenvolvimento Híbrido (Webview)**

C. **Desenvolvimento Nativo Multiplataforma (Cross-Platform)**

2. As Três Grandes Arquiteturas

A. Desenvolvimento Nativo (Native)

É a forma "clássica" de criar apps. Você usa as ferramentas oficiais da Apple ou do Google.

- **Linguagens**: Kotlin (Android), Swift (iOS).
- **Como funciona**: O código fala diretamente com o Sistema Operacional sem intermediários.
- **Vantagens**: Performance máxima (60/120 FPS), acesso total e imediato a recursos novos do hardware (câmera, AR, sensores).
- **Desvantagens**: Custo duplicado (dois códigos bases), manutenção difícil (bugs diferentes em cada plataforma).

2. As Três Grandes Arquiteturas

B. Desenvolvimento Híbrido (Webview)

Surgiu com a ideia: "E se colocássemos um site dentro de um aplicativo?".

- **Ferramentas**: Apache Cordova (antigo PhoneGap), Ionic, Capacitor.
- **Como funciona**: O app é, na verdade, uma janela de navegador sem a barra de endereços (chamada de **WebView**). O que você vê é HTML, CSS e JavaScript. Para acessar a câmera ou GPS, o app usa uma "ponte" (plugin) que traduz o JavaScript para código nativo.
- **Vantagens**: Curva de aprendizado baixa (quem sabe fazer site, faz app), código único.
- **Desvantagens**: Performance inferior (sensação de "site lento"), animações complexas travam, dependência de plugins para hardware.

2. As Três Grandes Arquiteturas

c. Desenvolvimento Nativo Multiplataforma (Cross-Platform)

É a evolução atual. O objetivo é ter um código único, mas com performance próxima do nativo. Aqui temos dois gigantes: **React Native** e **Flutter**.

React Native (Facebook/Meta)

- **Linguagem**: JavaScript/TypeScript.
- **Arquitetura** (The Bridge): O código de lógica roda em JS, mas ele "manda" o celular desenhar componentes nativos. É como se o JS fosse um marionetista controlando os bonecos (botões, listas) originais do Android/iOS.

2. As Três Grandes Arquiteturas

c. Desenvolvimento Nativo Multiplataforma (Cross-Platform)

Flutter (Google) - Foco do nosso curso

- **Linguagem**: Dart.Arquitetura (Canvas/Skia): O Flutter não usa os componentes visuais do Android/iOS. Ele tem sua própria "máquina de desenho" (Engine Skia ou Impeller). Ele desenha cada pixel na tela.
- **Analogia**: Imagine que o React Native pede para o Android desenhar um botão. O Flutter pega um pincel e desenha o botão ele mesmo.
- **Vantagens**: Performance excelente, visual idêntico em todos os celulares, desenvolvimento rápido (Hot Reload).

3. Tabela Comparativa

Característica	Nativo (Kotlin/Swift)	Híbrido (Ionic/Cordova)	Multiplataforma (Flutter)
Código Base	Dois (Android + iOS)	Um (Web)	Um (Dart)
Performance	Máxima	Baixa/Média	Alta (Quase Nativa)
Acesso Hardware	Direto	Via Plugins (Lento)	Via Channels (Rápido)
Renderização	Componentes do SO	HTML (WebView)	Engine Própria (Pixels)
Custo	$$$	$	$$

4. O Papel do Node.js no Ecossistema

O Node.js é um ambiente de execução JavaScript fora do navegador. No desenvolvimento mobile, ele atua como Ferramenta de Construção (Build Tool), e não necessariamente como servidor.

Por que ele é citado na ementa?

1. **Gerenciamento de Pacotes (NPM/Yarn):** No desenvolvimento híbrido (Ionic/React Native), todas as bibliotecas (câmera, mapas) são baixadas via NPM, que vem com o Node.
2. **Transpilação e Bundling:** O celular não entende código de desenvolvimento moderno (TypeScript, SCSS, Modularização). O Node.js roda ferramentas (como Webpack ou Metro Bundler) que "empacotam" e traduzem esse código para algo que o app consiga ler.
3. **Servidor Local de Desenvolvimento:** Quando você roda um app Ionic ou React Native no emulador, geralmente há um pequeno servidor Node rodando no seu PC enviando as atualizações de código em tempo real (Hot Reload).

Atenção: *No Flutter, o papel do Node.js é substituído pelas ferramentas do Dart SDK e pelo gerenciador Pub. Porém, entender Node é vital pois nossa API (semanas 29+) e ferramentas de IoT frequentemente utilizam Node.js.*

5. Roteiro Prático

1. **Instalação do Node.js**: Mostrar o comando `node -v` e `npm -v`. Será usado para instalar ferramentas auxiliares.
2. **Instalação do Flutter SDK**: Baixar o zip, configurar o **PATH** (variável de ambiente).
3. **Flutter Doctor**: Rodar o comando `flutter doctor` no terminal.Objetivo: Ensinar o aluno a ler diagnósticos. O *flutter doctor* mostra se o Android Studio está instalado, se o Java está configurado, etc.
4. O "**Hello World**": Criar o projeto (`flutter create primeiro_app`) e rodar no emulador.
5. **Análise de Pastas**: Mostrar que dentro da pasta do projeto Flutter existem as pastas `/android` e `/ios`.
   - Lição: Provar que o Flutter gera, no final das contas, um projeto nativo real para cada plataforma, diferente do Híbrido que apenas "empacota" um site.

5. Roteiro Prático

Exemplo Prático e Configuração

O objetivo aqui é garantir que a "**Habilidade 1.2**" (Utilizar ambientes de desenvolvimento mobile) seja iniciada com sucesso.

- **Instalação do SDK**: Download e configuração das variáveis de ambiente para o Flutter 2026.
- **O Comando Mágico**: Execução do `flutter doctor` no terminal para diagnosticar dependências faltantes (Android SDK, Xcode, VS Code plugins).
- **Primeiro Projeto**:
  - 1. Abrir o terminal e digitar: `flutter create meu_primeiro_app`.
  - 2. Explorar a estrutura de pastas: `lib/` (onde o código vive), `pubspec.yaml` (gerenciamento de dependências) e pastas nativas (android/, ios/).

alomundo

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: .fromSeed(seedColor: Colors.deepPurple),
      ),
      home: const MyHomePage(title: 'Aula 01 - Flutter'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});
  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor:
Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: .center,
          children: [
            const Text('Pressione o botão algumas vezes:'),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}
```