The Mobile Landscape Your First Deploy & the AI-Era Developer

Mobile Applications — Lecture 1

Today we ship something to a real phone.

War Story: The App That Shipped With AI-Written Auth

Real incident, 2024. A startup used an LLM (Large Language Model) to write their login flow.

- The AI produced clean, well-formatted Swift code
- Passed the first code review — looked great
- The JWT (JSON Web Token) was verified **client-side only**
- Any user could forge any token and access any account
- 40,000 users. Found by a security researcher, not the team

The lesson: The AI does not know your threat model. You do.

What This Course Teaches

Three things that industry actually tests in interviews:

1. **Understanding** — can you explain why the code works?
2. **Debugging** — can you find what the AI got wrong?
3. **Design** — can you choose the right stack and architecture?

We teach these across **four stacks simultaneously**.

> "One concept, four dialects" — the course's superpower.

Meet UniEats — One App, Four Stacks

The campus food-spots app we build all semester — the same screens, grown lecture by lecture, in all four stacks at once.

Stack	Language + UI	Code
Android	Kotlin + Jetpack Compose	`unieats/android`
iOS	Swift + SwiftUI	`unieats/ios`
Flutter	Dart	`unieats/flutter`
React Native	TypeScript + Expo	`unieats/rn`

> Each lecture adds one capability via a git tag (`l01-hello` → `l02-ui` → … → `l14-tests`). In class we `git diff` the previous tag against today's — and see exactly what the topic adds, in all four stacks side by side.

Repo: `github.com/dancojocar/ma`

Part 1: A Brief History

From a Coke machine to 7 billion devices in 15 years

Mobile in 15 Milestones

Year	Event
2007	iPhone ships. No SDK (Software Development Kit), no third-party apps.
2008	Android 1.0. App Store opens (500 apps).
2010	Windows Phone 7. Beautiful design, fatal app gap.
2013	Cordova/PhoneGap: "use your web skills." Performance killed it.
2014	Swift announced. Objective-C begins its long exit.
2015	React Native open-sourced by Meta.
2017	Flutter alpha. Kotlin is first-class Android language.
2019	SwiftUI, Jetpack Compose alpha, Flutter 1.0. **Declarative UI wins on every platform at once.**
2022	RN New Architecture (JSI/Fabric) stable. Flutter 3.0.
2024	Compose Multiplatform 1.0. DMA (Digital Markets Act) forces EU App Store opening.

What Died and Why — the Pattern

The graveyard

- Symbian — 47% share in 2010, zero by 2013
- BlackBerry OS — killed by the touchscreen
- Windows Phone — great design, no app ecosystem
- Cordova / PhoneGap — web tech, native feel → slow
- Xamarin — C# on mobile, Microsoft pivoted away
- AndroidThings — IoT (Internet of Things) Android, discontinued 2022

The pattern every death shares

- No community → no libraries → no jobs
- OR: performance or look-and-feel fell short of native
- OR: the platform owner changed their mind

What the survivors have

- Platform-owner backing (Kotlin, Swift)
- Own rendering engine (Flutter) or JS devs (RN)
- Genuine performance story

Why These Four Survived

Kotlin — Google backed it after the Oracle/Java lawsuit; 100% Java interop; coroutines solved async better than anything else on Android.

Swift — Apple controls the platform; genuinely safer and faster than Objective-C; ABI (Application Binary Interface) stability in 2019 enabled binary frameworks.

Flutter — Own renderer (no platform-widget translation); Dart compiles to native ARM; Google internal adoption (Pay, Ads) gave credibility; hot reload changed developer experience.

React Native — Largest developer pool on earth (JavaScript); New Architecture (JSI, JavaScript Interface) fixed the bridge bottleneck; Meta + Shopify + Coinbase run it at scale.

> **This course teaches all four** because each has a structural reason to exist that is not going away.

📊 Your Turn: The Course Survey (2 minutes, right now)

1. What OS (Operating System) does your **development machine** run?
2. What phone do you carry (Android / iPhone)?
3. Which languages/stacks do you **want** to learn this semester?
4. Which stack do you *predict* you will use at the **exam**?
5. Which **AI coding tools** do you already use?
6. Have you ever installed an app you built on a real device?

The same survey runs every year since 2018 — your answers join the trend charts coming up.

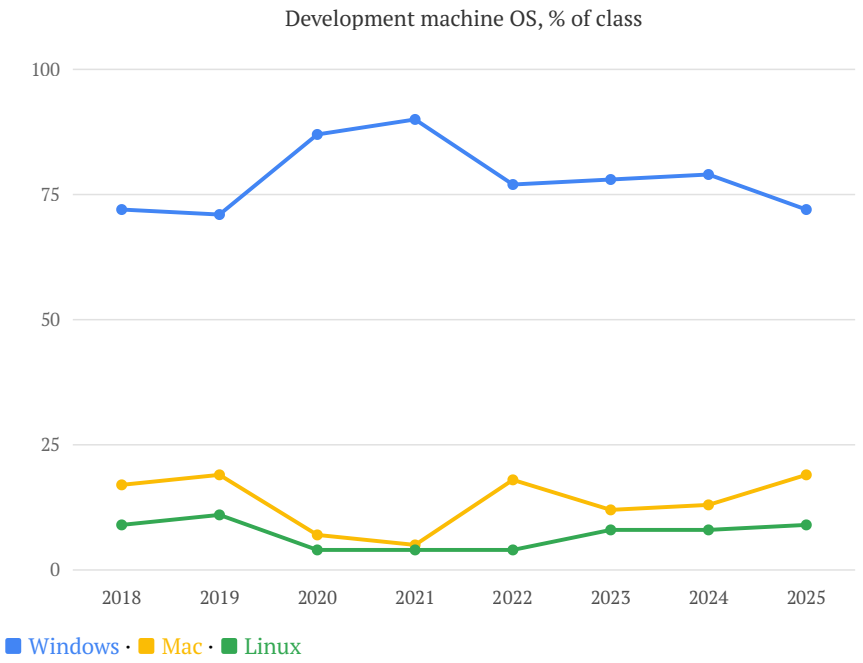


Live Results

Seven Years of You — Development Machines

"What OS is your development machine running?" (2018–2024, 61–111 responses/year)

Year	Windows	Mac	Linux
2018	72%	17%	9%
2019	71%	19%	11%
2020	87%	7%	4%
2021	90%	5%	4%
2022	77%	18%	4%
2023	78%	12%	8%
2024	79%	13%	8%
2025	**72%**	**19%**	**9%**

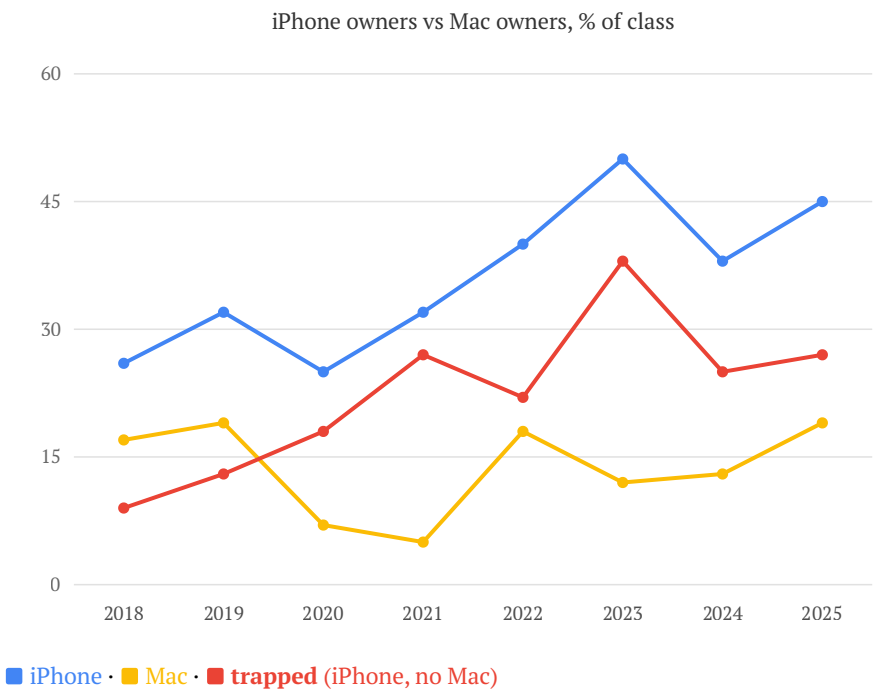


The constant: Windows dominates, Macs stay under ~20%. This is *why* iOS/Swift is read-first — the data, not a preference.

The iPhone + Windows Trap

Every year, a group of students **cannot build an app for the phone in their own pocket**:

Eight years of survey data:



2018–2024 trapped = estimated floor (iPhone − Mac). 2025: ***counted exactly — 26 of 96 students (27%),*** *and* ***21 of them own only an iPhone****: no Mac, no Android.*

Why it's a trap:

- Building iOS apps requires **Xcode**, and Xcode runs **only on macOS** — no exceptions
- iPhone ownership **roughly doubled** since 2018; Mac ownership stayed flat
- Today, **~1 in 4 of you** owns a phone you cannot write an app for

Your escape routes (all covered in this course):

- Android emulator on your laptop (Lab 1)
- Flutter / React Native + a cloud build service (EAS, Codemagic) for the iOS binary
- CI (Continuous Integration) macOS runners — free for public repos (L11)

What You Say vs. What You Do (2025)

Wanted to learn (survey, week 1)

Stack	%
Flutter/Dart	62%
Kotlin	58%
Swift	44%
React Native	~1%

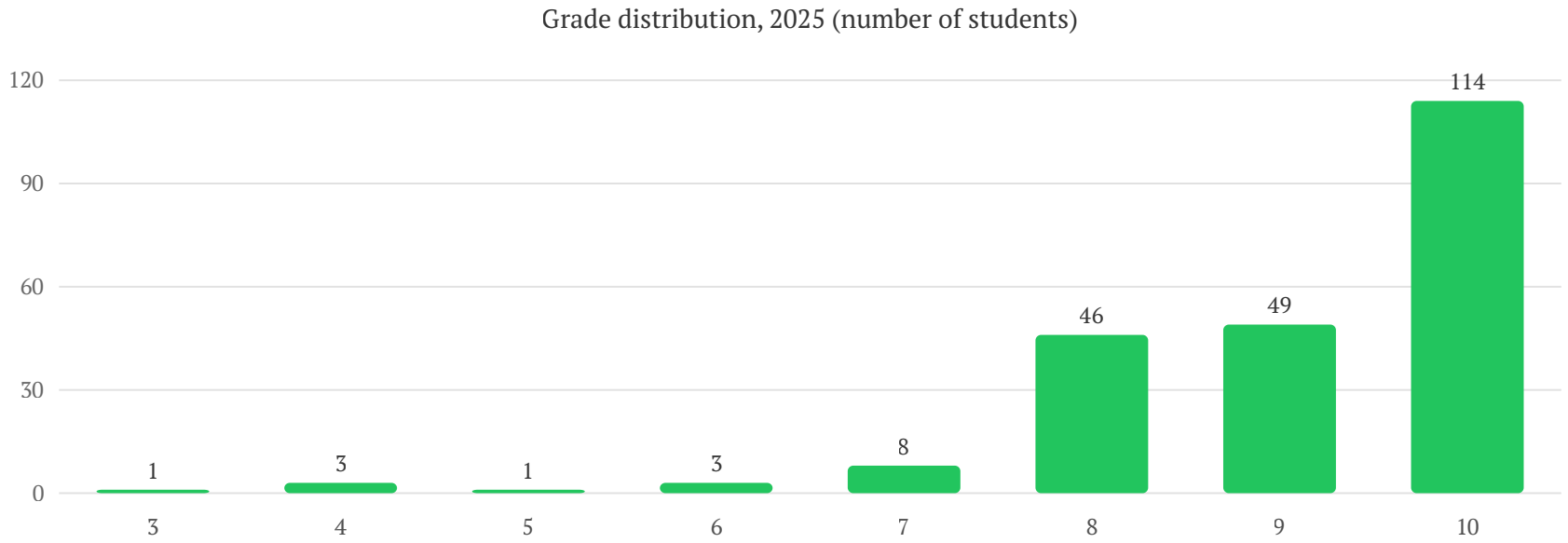
Actually used at the exam (225 students)

Stack	Used	%
Flutter	105	47%
Kotlin	58	26%
React Native	52	**23%**
iOS/Swift	10	4%

Three lessons hide in this gap ↓

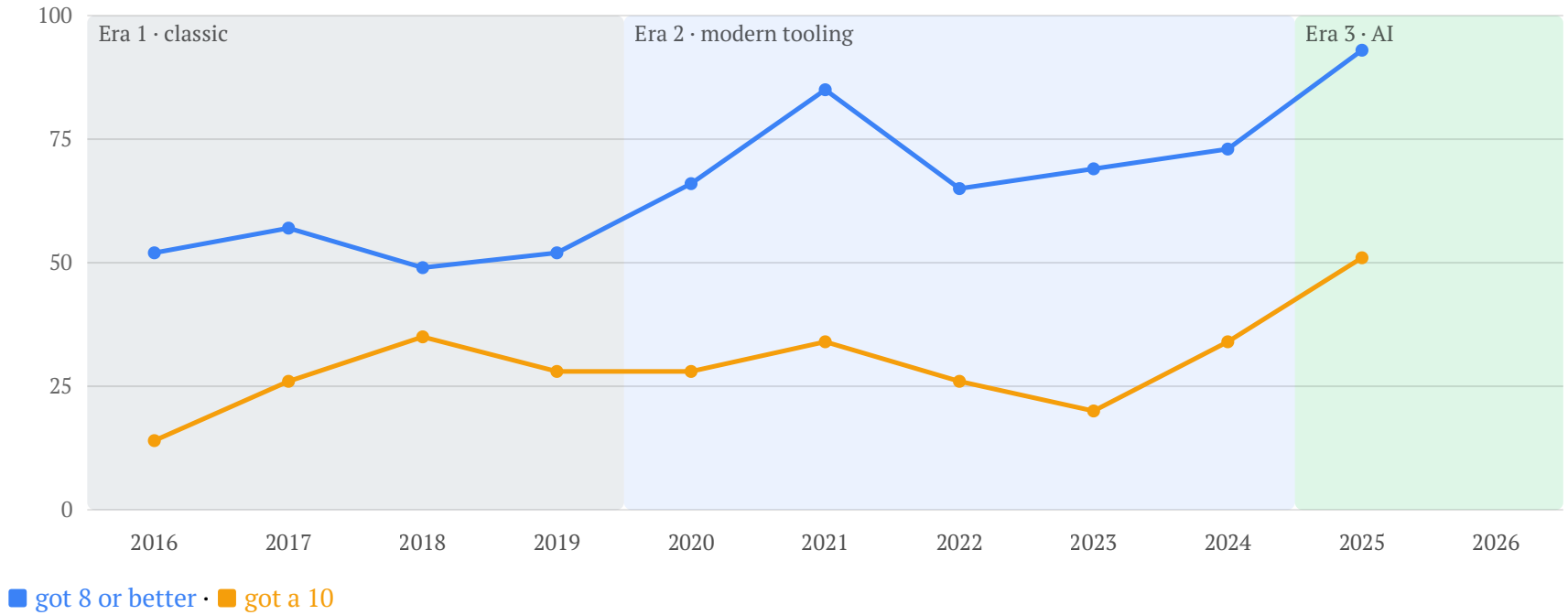
Last Year’s Exam — How It Ended

2025: 225 students, practical exam, AI (Artificial Intelligence) tools allowed



- **Half the class earned a 10; 93% earned 8 or better; nobody scored below 3**
- Students who attend the labs and practice simply do not fail this exam

Ten Years of Exams, Three Eras



Era 1 (2016–19): ~half at 8+ · **Era 2** (2020–24): two thirds, with the 2021 online-exam spike at 85% · **Era 3** (2025, AI universal): **93%** — above even the at-home year

Part 2: The Practical Core

Getting your phone ready and understanding what you are building

Enabling Developer Options on Android

Step 1 — Find your build number: Settings → About phone → Software information → Build number

Step 2 — Tap Build number 7 times You will see a countdown: "3 more steps to become a developer"

Step 3 — Enable USB debugging: Settings → Developer options → USB (Universal Serial Bus) debugging → ON

Step 4 — Connect via USB and authorize:

```
adb devices          # ADB (Android Debug Bridge): lists your device serial number
adb shell getprop ro.product.model   # confirm it is your phone
```

> If `adb devices` shows "unauthorized" — check the dialog on your phone screen and tap Allow.

Enabling Developer Mode on iPhone (iOS 16+)

Building iOS apps yourself requires a Mac. But **everyone** should know this procedure for reading and interview purposes.

Step 1 — Connect iPhone to a Mac running Xcode: Xcode will prompt to enable Developer Mode automatically on first connect.

Step 2 — On the iPhone: Settings → Privacy & Security → Developer Mode → ON → Restart

Step 3 — Pair the device in Xcode: Window → Devices and Simulators → your iPhone → Trust

Step 4 — Why a Mac is required: Apple's code signing requires a Mac-only toolchain (`codesign` , `xcrun`). There is no workaround.

> **Non-Mac owners:** You will read all iOS code this semester. Lab builds target Android. CI (Continuous Integration) runners on GitHub Actions macOS machines let your team compile-check Swift without owning a Mac — covered in L11.

Emulator Fallback — For Device-Less Students

If you do not have an Android phone available:

In Android Studio (IDE — Integrated Development Environment): Device Manager → Create Virtual Device → Phone → Pixel 8 → API 35 (Android 15) → Finish

Recommended specs for your laptop:

- Minimum 8 GB RAM (emulator takes 2–4 GB)
- Enable virtualization in BIOS (Intel VT-x / AMD-V) — check Task Manager → Performance → Virtualization: Enabled
- Keep at least 20 GB free disk space

> iPhone owners without a mac: use the emulator as your primary target.

Anatomy of an Android Project

```
MyApp/
├── settings.gradle.kts       ← project name, which modules exist
├── build.gradle.kts          ← top-level: plugin versions, repositories
├── gradle/
│   └── libs.versions.toml    ← version catalog: all dependency versions in one place
└── app/
    ├── build.gradle.kts      ← app module: SDK versions, dependencies, signing
    ├── src/
    │   ├── main/
    │   │   ├── AndroidManifest.xml   ← permissions, activities, services declared
    │   │   ├── java/com/example/     ← (or kotlin/) your source code
    │   │   └── res/                  ← drawables, strings, themes
    │   ├── test/                     ← JVM (Java Virtual Machine) unit tests
    │   └── androidTest/              ← instrumented tests (run on device/emulator)
    └── proguard-rules.pro
```

Anatomy of an iOS Project

```
MyApp.xcodeproj/
├── project.pbxproj          ← build settings, file references (rarely edit by hand)
└── xcshareddata/

MyApp/
├── MyAppApp.swift           ← @main entry point, equivalent to MainActivity
├── ContentView.swift        ← first SwiftUI (Swift User Interface) View shown
├── Assets.xcassets/         ← images, icons, colors
├── Info.plist               ← app permissions, bundle ID, supported orientations
└── Preview Content/         ← SwiftUI preview assets

MyAppTests/                  ← XCTest (Apple's test framework) unit tests
MyAppUITests/                ← XCUITest end-to-end tests
```

> `Info.plist` is iOS's equivalent of `AndroidManifest.xml`. Both declare what your app is allowed to do.

Project Trees Compared — Flutter and React Native

Flutter

```
my_app/
├── pubspec.yaml      ← dependencies + metadata
├── pubspec.lock      ← pinned dependency versions
├── lib/
│   ├── main.dart     ← entry point (runApp)
│   ├── app.dart      ← root widget, theme, routes
│   └── features/     ← feature folders
├── android/          ← generated; rarely touched
├── ios/              ← generated; rarely touched
├── test/             ← flutter_test unit tests
└── assets/           ← fonts, images (see pubspec)
```

React Native / Expo

```
my-app/
├── package.json      ← dependencies + npm scripts
├── package-lock.json ← pinned dependency versions
├── app.json          ← Expo config (icons, splash)
├── tsconfig.json     ← TypeScript compiler settings
├── app/              ← Expo Router file-based routes
│   ├── (tabs)/
│   │   └── index.tsx ← first tab screen
│   └── _layout.tsx   ← root layout, navigation shell
├── android/          ← Expo-managed native code
├── ios/              ← Expo-managed native code
└── assets/           ← images, fonts
```

Build Pipelines — Android and iOS

Android

`.kt` → `.class` → `.dex` → `.apk` / `.aab` → signed → device

```
./gradlew assembleDebug
adb install -r app/build/outputs/apk/debug/app-debug.apk
```

- D8 / R8: JVM (Java Virtual Machine) bytecode → Dalvik bytecode
- Play Store requires `.aab` (Android App Bundle) since Aug 2021; generates a device-specific `.apk`, cutting download size 40–60%

iOS (read-only context)

`.swift` → `.o` → `.app` → signed `.app` → `.xcarchive` → `.ipa` → App Store Connect

- Every real-device build must be code-signed (certificate + provisioning profile)
- `.ipa` (iOS App Store Package) ≈ `.aab` for iOS distribution
- Signing requires a Mac + Xcode (IDE — Integrated Development Environment); no workaround

🖥️ Demo — Deploy UniEats to a Real Phone

> The UniEats Android app installs and runs on a real device (or emulator) in the room. Tag `l01-hello` is one screen, **UniEats** / *Find your next campus meal.*, the same build you install on your own phone in Lab 1. The spot list arrives with `l02-ui` next lecture.

Repo: `github.com/dancojocar/ma` · **Code:** `unieats/android` @ tag `l01-hello` · no server needed

```
git clone https://github.com/dancojocar/ma.git && cd ma
git switch --detach l01-hello
./unieats/android/gradlew -p unieats/android :app:installDebug
```

Or open `unieats/android` in Android Studio and press Run ▶. App id: `com.unieats.app`.

Part 3: Languages — Positioned, Not Taught

A 90-second preview of three languages you will learn properly starting L02

Kotlin is Java, Fixed

```
// Java you already know:
String name = null;
if (name != null) System.out.println(name.length());

// Kotlin equivalent — null safety is built in:
val name: String? = null          // ? means "nullable"
println(name?.length)             // ?. means "only call if not null" — safe, no crash
println(name ?: "unknown")        // ?: means "use this if null" — the Elvis operator

// Data class replaces 30 lines of Java boilerplate:
data class Place(val id: String, val name: String, val rating: Float)
// equals(), hashCode(), toString(), copy() — all generated automatically
```

> **Kotlin in one sentence:** Java with null safety, type inference, and less ceremony. We teach it fully in L02.

Dart is JavaScript with Types

```
// JavaScript you already know:
async function fetchPlace(id) { const data = await api.get(id); return data; }

// Dart equivalent — async/await works exactly the same:
Future<Place> fetchPlace(String id) async {
  final data = await api.get(id);   // await works just like JS
  return data;                       // Future<T> is like Promise<T> in JS
}

// Types are not optional — but inference does the work:
var name = 'UniEats';          // type inferred as String
final items = <Place>[];       // final means the binding cannot be reassigned (like const in JS)
```

> **Dart in one sentence:** JavaScript with mandatory types, classes like Java, and async/await you already know. We teach it fully in L02.

Swift — You Will Read It, Not Write It

Swift (you will read this)

```swift
struct Place {
    let id: String          // let ≈ val
    var name: String        // var = var
    var rating: Float
}

// Optional — same idea, same syntax:
var place: Place? = nil
print(place?.name ?? "unknown")
```

Kotlin (you will write this)

```kotlin
data class Place(
    val id: String,         // val = immutable
    var name: String,       // var = mutable
    var rating: Float,
)

// null safety:
var place: Place? = null
println(place?.name ?: "unknown")
```

Swift in one sentence: Kotlin's sibling — same null-safety model (`?.` is identical, `??` ≈ `?:`), same value-type preference, Apple's platform. Read-first all semester: you explain and compare Swift; you never produce it (unless you own a Mac and want to).

TypeScript / React Native — JavaScript You Already Know

```tsx
// JavaScript (JS) you already know:
// function Button({ label }) { return <button>{label}</button>; }

// TypeScript (TS) / React Native (RN) equivalent — types are the only new thing:
import { useState } from 'react';
import { View, Text, Pressable } from 'react-native';

type Props = { label: string; initialCount?: number };   // TS type — like a JS object shape, enforced

export default function Counter({ label, initialCount = 0 }: Props) {
  const [count, setCount] = useState(initialCount);      // useState hook — same as React on the web
  return (
    <View>                                                {/* View ≈ <div> */}
      <Text>{label}: {count}</Text>                       {/* Text ≈ <span> */}
      <Pressable onPress={() => setCount(c => c + 1)}>
        <Text>Tap</Text>
      </Pressable>
    </View>
  );
}
```

> **TypeScript (TS) in one sentence:** JavaScript with types you already know from web development — the same hooks, the same JSX (JavaScript XML), just `View` instead of `div`. We teach it fully in L02.

Part 4: The Grading Details

How points work — read this carefully

The Lab Grade: One Project, Five Steps

Lab Grade (LG) = 5 assignments × 2 points = 10. Same app, growing every step.

#	Assignment	Due (Sunday 23:59)	Evaluated in lab
1	**Project Idea** — client-view README + mockups	18 Oct (week 3)	week 4
2	**UI only — Native** (Android or iOS), in-memory CRUD	8 Nov (week 6)	week 7
3	**UI only — Non-Native** (Flutter, RN, …), in-memory CRUD	29 Nov (week 9)	week 10
4	**Local DB** — one UI project, persisted on device	20 Dec (week 12)	week 13
5	**Server Integration** — your own REST server, offline-first	10 Jan (week 13)	week 14

- Each assignment: **2 points**, graded with a published rubric (next slide)
- Deadlines are final and shared by all groups; **25% penalty per calendar week** late
- Submit by pushing to **your assignment repository in the course GitHub organization**; nothing else is graded

How an Assignment Is Graded

1. The rubric (published on the course portal, the same for every group)

- Every assignment is split into weighted criteria, e.g. read list 20%, create 20%, update 20%, delete 20%, GitHub 20%
- Each criterion is scored **Excellent** / **Good** / **Fair** / **Poor** with written expectations, e.g. *update*: element reused with the same id, fields pre-populated, errors shown in the form
- The weighted result scales to the assignment's **2 points**

2. The demo — on a **physical device or an emulator**, never a web preview, Figma or screenshots

3. The knowledge check — the part that decides the grade this year

> At the evaluation you answer questions about **your own code**: why this class, what happens on rotation, where does the id come from, what breaks offline. If you cannot answer in a reasonable time, on your own, **the assignment is not delivered** — no matter how well the app runs.

Exam, Bonus & Final Grade

Practical exam (EG) — you build, at your machine, then explain what you built. Details in L14.

Final Grade = LG × 0.6 + EG × 0.4, only if **LG ≥ 4.5 and EG ≥ 5** — otherwise the course is not passed

To sit the exam: at least **90% lab attendance** and LG ≥ 4.5. Otherwise, the re-examination session.

Bonus points on the exam grade (up to +3):

- **+2 Early Completion** — present **all five** assignments by **6 Dec (week 10)**, two weeks before the winter break
- **+1 Advanced Feature** — a significant extra feature in the server-integrated project (camera, location, background sync, biometric lock, reactive search, on-device ML, tests…); same deadline, and only on top of the +2

> Use AI at home as much as you want. In the lab you explain the result on your own — that is the whole contract.

Part 5: Spot the AI Bug 🐞

Find the flaw before the next slide

Spot the AI Bug 🐞 — Find the Flaw

You ask an AI: *"I have a Windows laptop and an iPhone — how do I build and ship my own iOS app?"* It answers, confidently:

> 1. Install **Xcode** on your Windows laptop from the Microsoft Store.
> 2. React Native apps are **compiled to native machine code**, so they run exactly as fast as Swift.
> 3. Once it builds, just **drag the app file onto your iPhone** to install it.

What's wrong? You already have everything you need from today's lecture. Take 30 seconds.

🐞 Bug Revealed

1. **Install Xcode on your Windows laptop…❌ Xcode is macOS-only.** There is no Xcode for Windows — this is the iPhone + Windows Trap from earlier. On Windows/Linux you build iOS through Flutter or React Native plus a cloud Mac (CI), never Xcode locally.
2. **React Native is compiled to native machine code…❌ React Native runs JavaScript** on a JS engine and shows real *native* views — it is **not** compiled to machine code like Swift (or like Flutter's ahead-of-time build). Fast enough, but a different model.
3. **Just drag the app file onto your iPhone…❌ iOS apps must be signed** and installed via Xcode / TestFlight / the App Store — you can't drag-and-drop an `.ipa`. Android is open (`adb install`); iOS is not.

The lesson: the AI sounded certain about all three. Knowing the landscape is what let you catch it.

Part 6: Interview Corner 🎤

Questions you will be asked

Interview Corner 🎤

Q1: "Walk me through what happens when a user taps your app icon and the first screen appears."

Strong answer: OS process creation → Application/AppDelegate/main() init → first Activity/ViewController/Widget → layout pass → first frame → cold vs warm vs hot start on Android.

Q2: "What is the difference between a `.apk` and a `.aab`?"

Strong answer: AAB (Android App Bundle) not directly installable; Play Store generates a device-specific APK (Android Package Kit); download size drops 40–60%; required for new Play apps since Aug 2021.

Q3: "Why did Google move Android from Java to Kotlin?"

Strong answer: Oracle lawsuit over Java APIs (2010–2021, $9B at stake); 100% Java interop — no rewrite; null safety, coroutines, less boilerplate; JetBrains partnership.

Q4: "Device not in `adb devices`. What do you check?"

Strong answer: USB (Universal Serial Bus) debugging toggle; authorization dialog on phone (behind lock screen); cable type; OEM (Original Equipment Manufacturer) USB driver on Windows; `adb kill-server && adb start-server`.

Q5: "Which stack for a two-person startup shipping Android + iOS?"

Strong answer: decision framework — team's existing skills, platform-specific features needed, timeline; Flutter if no mobile background; React Native (RN) if TypeScript (TS) team; never "I always use X."

🖥️ Demo — UniEats on Four Devices

> The same two lines, **UniEats** / *Find your next campus meal.*, in **all four stacks**: Android (Kotlin), iOS (Swift), Flutter (Dart), React Native (Expo/TypeScript). That one screen is all of `l01-hello`. Goal: feel the one app across four stacks before L02 digs into the code.

Repo: `github.com/dancojocar/ma` · all four at tag `l01-hello` · run from the repo root

Stack	Run command	App id
Android / Kotlin	`./unieats/android/gradlew -p unieats/android :app:installDebug`	`com.unieats.app`
iOS / Swift (Mac)	`open unieats/ios/UniEats.xcodeproj` → Run on a simulator	`com.unieats.app`
Flutter / Dart	`cd unieats/flutter && flutter run`	`com.unieats.flutter`
React Native / Expo	`cd unieats/rn && npm ci && npx expo run:android`	`com.unieats.rn`

Different app ids, so all three Android builds sit side by side on one phone. `expo run:android` builds RN as its own app; Expo Go (`npx expo start`) also opens it up to `l08` .

The AI-Era Contract (Summary)

We encourage:

- AI (Artificial Intelligence) for boilerplate, scaffolding, syntax lookups
- AI to generate a first draft → read every line → verify the explanation
- AI on your project at home: directing it well on a real codebase is a professional skill

We grade (in the lab, at your screen):

- Your project, five steps, against a published rubric
- Your **answers about your own code** — no AI, no notes, a reasonable amount of time
- The practical exam: build, then explain

We assume:

- Code produced at home may be AI-generated, and that is fine
- Therefore the code alone earns nothing; **explaining it earns the points**
- Code you cannot explain is code you did not deliver

> "The only winning strategy is to understand the code you ship well enough to explain and modify it under pressure."

Course Logistics

- **Stack choice:** pick Android/Kotlin plus one cross-platform stack (Flutter or RN) for labs
- **All four stacks in lectures:** you are responsible for reading and comparing all four
- **iOS / Swift:** literacy tier for everyone; build tier for Mac owners
- **Lab 1:** device bootcamp (install UniEats `l01-hello` on your phone), join the Teams class, accept the **Project Idea** assignment on GitHub Classroom — no points, all load-bearing
- **Toolchain setup:** download SDKs (Software Development Kits) at home; setup guide on course platform
- **Repo:** GitHub Classroom invitation in the course platform
- **Office hours:** after each lecture + during lab open-workshop hour

> **Before Lab 1:** Install Android Studio 2026.2+ (IDE — Integrated Development Environment) and, if Flutter is your stack, Flutter 3.47+, enable Developer Options on your phone, clone `github.com/dancojocar/ma` and build `l01-hello` once at home (`./unieats/android/gradlew -p unieats/android :app:assembleDebug`). Arrive with a working build.

Recap — What We Covered Today

1. **War story** — AI-written auth, 40k users at risk: understanding > output
2. **Brief history** — 2007–2024 compressed; why four stacks survived the graveyard
3. **Android device setup** — Developer Options, USB debugging, wireless ADB (Android Debug Bridge), emulator fallback
4. **iOS device setup** — Developer Mode (iOS 16+), pairing with Xcode, why a Mac is required
5. **Project anatomy** — Android Gradle tree, iOS xcodeproj/Info.plist, Flutter pubspec, RN (React Native) package.json compared
6. **Build pipelines** — Android: source → .dex → APK (Android Package Kit) → sign → install; iOS: source → .app → .ipa (iOS App Store Package) → sign → distribute
7. **Live deploy** — UniEats `l01-hello` (one hello screen) on a real phone or emulator; you install it yourself in Lab 1
8. **Languages positioned** — Kotlin = Java fixed; Dart = JS with types; Swift = read-only
9. **Grading** — one project in five steps, 2 pts each, rubric + demo + knowledge check; code you cannot explain is not delivered
10. **Spot the AI Bug** — Xcode is macOS-only; RN runs JS, not native machine code; iOS installs require code-signing, not drag-and-drop
11. **Interview Corner** — cold start, APK vs AAB, Kotlin history, ADB debugging, stack selection

If You Only Remember 3 Things

1. **Enabling Developer Options takes 7 taps on the build number** — you will need this in every lab this semester

2. **Understanding is the graded asset, not output** — build your project with any tools you like, then explain every line of it on your own at the evaluation

3. **All four stacks converged on `UI = f(state)` in 2019** — learn that one model and you understand all four dialects; we spend the rest of the semester drilling it

Questions?

Common first-lecture questions:

- *"I only have Windows — can I do iOS?"* → Yes: literacy via SwiftUI snippets in Xcode Online / Swift Playgrounds; CI (Continuous Integration) builds via GitHub Actions macOS runners (L11); no penalty for hardware limits.
- *"Which stack should I pick?"* → Android + Flutter if you are new to mobile; Android + RN (React Native) if you know TypeScript. Decide by Lab 2.
- *"Can I use ChatGPT / Copilot for the project?"* → Yes, at home, as much as you like. At the evaluation you explain the code on your own, without AI or notes. Code you cannot explain is not delivered.
- *"What if I miss a deadline?"* → 25% off per calendar week, same for everyone. Four weeks late is zero. The deadlines are on the portal and in the calendar feed.

Resources:

- `developer.android.com/jetpack/compose` · `docs.flutter.dev` · `reactnative.dev` · `docs.expo.dev`
- DartPad (`dartpad.dev`) — Flutter in the browser · Expo Snack (`snack.expo.dev`) — RN in the browser
- Swift Playgrounds / `swiftfiddle.com` — read and run Swift without Xcode