React Native

Introduzione e Modulo 1

Framework open-source sviluppato da Meta per creare app iOS e Android con JavaScript e React.

Obiettivo: riuso del codice tra piattaforme, mantenendo un’esperienza utente fluida e reattiva.

Documento di riferimento durante la lezione (con esempi, comandi e note operative).

Setup iniziale — checklist (prima della lezione)

Obiettivo: ridurre al minimo i blocchi in live

Installare Node.js (con npm) e verificare che i comandi funzionino.

Installare VS Code e Git (o equivalenti).

Installare Expo Go sul telefono (Android/iOS).

Per mirroring Android su Mac: installare Android Platform Tools (adb) + scrcpy.

Testare una volta la sera prima: creare progetto Expo e aprirlo in Expo Go.

Setup (macOS) — installazioni consigliate via Homebrew

Opzione pratica per lezioni/lab

Spiegazione

Homebrew è un package manager per macOS: semplifica installazioni e aggiornamenti.

Se non vuoi usare Homebrew, puoi installare manualmente (Node installer, ecc.).

Per il Modulo 1 (Expo): Node + npm + Expo Go sono sufficienti per partire.

adb e scrcpy servono solo se vuoi condividere lo schermo del telefono su Zoom.

```
# 1) Homebrew (se non presente)
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homeb
rew/install/HEAD/install.sh)"

# 2) Tool base
brew install git
brew install --cask visual-studio-code

# 3) Node.js (scelta semplice)
brew install node

# 4) Mirroring Android (opzionale ma
consigliato per Zoom)
brew install android-platform-tools
brew install scrcpy
```

Setup (macOS) — verifica che tutto funzioni

Prima di entrare in aula

Spiegazione

Verifica che node/npm siano nel PATH e rispondano.

Verifica che adb veda il dispositivo (solo se userai mirroring).

Se adb devices non mostra nulla: controlla cavo, autorizzazione e USB Debugging.

Obiettivo: arrivare in aula con una checklist “verde”.

```
node -v
npm -v
git --version

# solo se usi mirroring Android
adb version
adb devices
```

Setup telefono Android (Samsung)

Passi essenziali per Expo + (opzionale) mirroring

- Installa Expo Go dal Play Store.
- Stessa rete Wi-Fi del Mac (consigliato per collegamento rapido).
- Per mirroring via cavo: abilita Developer Options e USB Debugging.
- Quando colleghi il cavo: accetta la richiesta “Allow USB debugging”.
- Se la rete in aula è instabile: prevedi hotspot come piano B (opzionale).

Mirroring Android su Mac (scrcpy)

Per Zoom: fai vedere il telefono come finestra condivisibile

Collega il telefono via cavo USB (stabilità > wireless).

Nel terminale: adb devices deve mostrare un device “authorized”.

Avvia scrcpy: si apre una finestra con lo schermo del telefono.

In Zoom: condividi solo la finestra scrcpy (non tutto lo schermo).

Mirroring Android (scrcpy) — comandi tipici

Mac + Samsung

Spiegazione

Se adb devices mostra “unauthorized”: guarda il telefono e accetta l’autorizzazione.

Se non appare alcun device: prova un altro cavo o porta USB e controlla Debug USB.

scrcpy funziona come mirror: ottimo per mostrare Expo Go agli studenti.

```
adb devices
scrcpy
```

Setup (Windows) — installazioni rapide (indicativo)

Per studenti su Windows (comandi comuni)

Spiegazione

Su Windows è comune usare winget (Windows Package Manager).

Alternativa: installer ufficiali (Node, Git, VS Code).

Per Expo in Modulo 1: bastano Node + npm + Expo Go su telefono.

adb/scrcpy servono solo se vuoi mirroring (non obbligatorio).

```
# Node + Git + VS Code (se winget
disponibile)
winget install OpenJS.NodeJS.LTS
winget install Git.Git
winget install
Microsoft.VisualStudioCode

# verifica
node -v
npm -v
git --version
```

Nota Windows: iOS

Chiarimento importante

La build e il simulatore iOS richiedono macOS con Xcode.

Su Windows si lavora serenamente con Expo + Android (device reale o emulator).

Per il corso: l’obiettivo del Modulo 1 è partire e vedere risultati, non fare build iOS native.

Obiettivo del corso

Imparare a sviluppare applicazioni mobile per iOS e Android utilizzando React Native.

Focus su componenti, navigazione, gestione dello stato e integrazione con API.

Approccio: teoria + esempi pratici + codice spiegato passo per passo.

Modulo 1 — Programma

Introduzione a React Native

Differenze tra app native, ibride e React Native

Setup dell’ambiente di sviluppo (con Expo e senza Expo)

Struttura di base di un progetto React Native

Componenti fondamentali: View, Text, Image, Button/Pressable

Metodo di lezione (operativo)

Come usare questo PDF durante la live

Il PDF è la scaletta: ogni concetto ha esempi e snippet per guidare la spiegazione.

Durante la live: alternare slide → editor → terminale → risultato sul device.

Consiglio pratico: tenere un progetto Expo “backup” già creato, pronto per evitare blocchi.

Regola: se qualcosa non funziona, si continua con il backup e si riprende il punto dopo.

Prerequisiti concettuali (in breve)

Terminale: capire cartelle, comandi e messaggi di errore.

JavaScript: funzioni, oggetti, array, async/await (livello base).

React: componenti e props (stato lo approfondiremo nel corso).

Mobile: differenze iOS/Android, dimensioni schermo, gesture, Safe Area.

Strumenti principali

Che cosa sono e cosa fanno (riassunto)

Tool di base

Node.js: runtime per eseguire tool e script JS.

npm: gestisce dipendenze e script; standard de-facto.

Terminale: punto centrale per creare/avviare progetti.

Git: versionamento, branch, condivisione.

Tool mobile

Expo: workflow gestito per sviluppo rapido.

Expo Go: app sul telefono per test in sviluppo.

Android Studio: SDK + emulator + build Android.

Xcode: simulator + build iOS (macOS).

Node.js

Cos’è e perché serve

È l’ambiente che esegue JavaScript fuori dal browser (server e tooling).

Permette di usare npm: installare librerie e lanciare comandi di progetto.

In React Native, Node esegue bundler e strumenti (Metro, Expo CLI, ecc.).

Best practice: usare una versione LTS stabile per ridurre incompatibilità.

npm

Cos’è e cosa gestisce

È il package manager più diffuso nell’ecosistema Node.

Installa dipendenze in node_modules e registra versioni nel lockfile.

Esegue scripts definiti nel package.json (npm run start, npm run ios, …).

In corso: esempi e comandi principali sono tutti basati su npm.

pnpm (confronto)

Perché viene usato in alcuni team

Obiettivo: ridurre duplicazioni e rendere installazioni più rapide e deterministic.

Usa uno store globale e link: spesso meno spazio su disco su progetti grandi.

È ottimo in monorepo e progetti multi-package; non è obbligatorio per RN.

Nel corso: lo citiamo come alternativa, ma il flusso didattico resta npm.

npm vs pnpm — differenze pratiche

Cosa cambia nella vita reale (senza cambiare gli esempi)

Quando restare su npm

Massima compatibilità e supporto.

Ambienti “standard” (aziende, tutorial, CI comuni).

Meno variabili durante una live.

Ottimo per progetti singoli.

Quando valutare pnpm

Monorepo o molti pacchetti.

Repo grandi e installazioni frequenti.

Team che vuole regole più “strict” sulle dipendenze.

Ottimizzazione spazio/velocità.

Terminale: comandi minimi utili

macOS / Windows (PowerShell)

Spiegazione

cd: cambia cartella.

ls (macOS) / dir (Windows): elenca file e cartelle.

pwd: mostra la cartella corrente.

Obiettivo: navigare nel progetto e lanciare comandi.

```
# macOS
pwd
ls
cd myApp

# Windows (PowerShell)
pwd
dir
cd myApp
```

PATH e errori di comando

Come interpretarli

Se vedi “command not found” / “not recognized”: il tool non è installato o non è nel PATH.

Dopo un’installazione: spesso serve chiudere e riaprire il terminale.

Se node -v non risponde: reinstallare Node oppure verificare l’installer.

Stessa logica per git, adb, scrcpy.

React Native

Definizione

Framework open-source di Meta per creare app iOS e Android usando JavaScript e React.

La UI è composta da componenti nativi (View, Text, Image, …).

Gran parte della logica e UI può essere condivisa tra piattaforme.

Possibile estendere con moduli nativi quando necessario (bridge / native modules).

Caratteristiche principali

Cross-platform: una codebase principale con adattamenti mirati.

Componenti nativi: interfaccia coerente con la piattaforma.

Fast Refresh: feedback rapido durante lo sviluppo.

Ecosistema ampio: librerie e community.

Integrazione nativa: accesso a funzioni avanzate quando serve.

Nativo vs Ibrido vs React Native

Come ragionare sulla scelta tecnologica

Nativo

Due codebase: iOS + Android.

Performance e API al massimo livello.

Costi e tempi più elevati.

Ideale per funzionalità native complesse.

React Native

UI nativa con logica JS/React.

Buon equilibrio velocità/qualità.

Condivisione di codice tra piattaforme.

Spesso scelta ottimale per prodotti iterativi.

Ibrido (WebView)

Cosa significa e limiti tipici

UI web dentro un contenitore mobile (WebView).

Ottimo per contenuti web; meno adatto a UX mobile avanzata.

Performance e integrazione native possono essere limitate.

Scelta valida per app semplici o wrapper di servizi web.

Come “arriva” il codice al telefono

Concetto di bundling

Durante lo sviluppo, Metro prepara il bundle JavaScript.

Il device/simulator carica il bundle e lo esegue nel runtime React Native.

Fast Refresh aggiorna UI e logica rapidamente.

In produzione, il bundle viene distribuito insieme all’app (build).

Pausa

10–15 minuti

Setup: due workflow principali

Workflow Expo (managed): setup più rapido, ottimo per iniziare e per live.

Workflow React Native CLI (bare): massimo controllo nativo, toolchain completa.

In questo Modulo 1 useremo Expo come percorso principale.

La CLI viene spiegata per capire differenze e casi d’uso.

React Native CLI

Che cos’è

Tooling ufficiale per progetti “bare” con cartelle native iOS e Android.

Gestisce comandi per run/build e integrazione con simulator/emulator.

Richiede Xcode, Android Studio/SDK, Gradle, CocoaPods (iOS) in molti casi.

È la scelta quando devi personalizzare profondamente la parte native.

Expo

Che cos’è

Piattaforma/tooling che semplifica sviluppo React Native.

Fornisce un workflow gestito: meno configurazione iniziale.

Include librerie Expo per funzionalità comuni (camera, notifiche, ecc.).

Consente test rapido con Expo Go su telefono reale.

Expo Go

Cosa è e a cosa serve

È un’app (iOS/Android) che carica il tuo progetto Expo in modalità sviluppo.

Scansionando un QR o usando un link, il telefono apre l’app in pochi secondi.

Ottimo per una lezione: riduce dipendenza da simulator/emulator.

Limite: alcune integrazioni native specifiche richiedono workflow bare o build dedicate.

Come funziona Expo (passo per passo)

1) Crei progetto con npm.

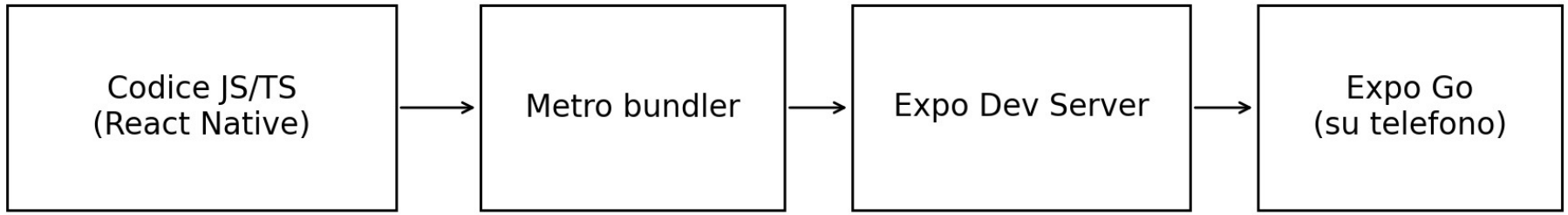
2) Avvii il server (npm start): parte Metro + Expo Dev Server.

3) Il telefono (Expo Go) si collega al Dev Server (QR/link).

4) Ogni modifica al codice viene riflessa rapidamente (Fast Refresh).

Expo — workflow (schema)

Vista di insieme

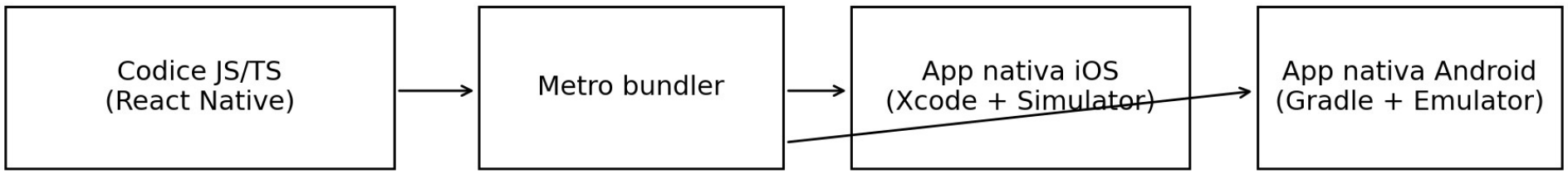


Idea: avvii il server, il telefono carica l'app in modalità sviluppo e riceve aggiornamenti (Fast Refresh).

Schema concettuale: codice → Metro → Dev Server → Expo Go sul telefono.

React Native CLI — workflow (schema)

Vista di insieme



Idea: generi e compili app native (iOS/Android). Più controllo, ma più toolchain e configurazioni.

Schema concettuale: codice → Metro → build e run dell'app nativa su iOS/Android.

Expo vs CLI — tabella riassuntiva

Quando scegliere cosa

Aspetto	Expo (managed)	React Native CLI (bare)
Setup iniziale	Molto rapido	Più complesso (SDK, IDE)
Test su device	Expo Go	Build nativa + install
Moduli nativi	Limitazioni/SDK Expo	Massima libertà
Rischio in live	Basso	Medio/alto
Quando sceglierlo	Avvio, prototipi, molti progetti	Requisiti nativi avanzati, enterprise

Tabella indicativa: differenze pratiche tra workflow.

package.json

Perché è centrale

È il manifest del progetto: nome, dipendenze, script, configurazioni.

npm lo usa per installare librerie e per eseguire comandi (scripts).

Esempio: "start": "expo start" permette npm run start.

In team, package.json + lockfile garantiscono coerenza.

package.json — esempio commentato

Spiegazione

scripts: comandi che lanci con npm run …

dependencies: librerie necessarie a runtime.

devDependencies: strumenti di sviluppo (lint, types, test).

lockfile: blocca versioni reali installate per riproducibilità.

```
{
  "name": "myApp",
  "private": true,
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios"
  },
  "dependencies": {
    "expo": "…",
    "react": "…",
    "react-native": "…"
  }
}
```

Lockfile

A cosa serve

package-lock.json (npm) registra l’albero reale delle dipendenze.

Assicura che tutti installino le stesse versioni (team e CI).

Riduce “works on my machine”.

Se cambi package.json, rigeneri lockfile con npm install.

node_modules

Cosa è e cosa non fare

Cartella con le librerie installate.

Non si committa (è generabile).

Se si corrompe: spesso si elimina e si reinstalla (pulizia).

Problema tipico: versioni diverse → lockfile aiuta a evitarlo.

CocoaPods e Watchman

Cosa sono e quando diventano importanti

CocoaPods (iOS)

Gestore dipendenze native per iOS.

Tipico con workflow CLI (progetti con cartella ios).

Integra librerie native in Xcode tramite Pods.

Con Expo managed: di solito non serve nel Modulo 1.

Watchman (macOS)

Ottimizza il file-watching su macOS.

Può rendere Metro più stabile su progetti grandi.

Non è obbligatorio ma spesso consigliato.

Con Expo: utile, ma non blocca l’avvio del progetto.

Creare un progetto Expo (npm)

Comandi consigliati per la live

Spiegazione

Questi comandi sono l'opzione più supportata e prevedibile in aula.

Se vuoi evitare download lunghi, crea il progetto prima della lezione (backup).

Dopo npm start, apri Expo Go e scansiona il QR.

```
npm create expo-app myApp
cd myApp
npm install
npm start
```

Avvio e test (Expo)

Cosa fare quando sei in classe

Spiegazione

Device reale: Expo Go + QR è la strada più solida.

Se QR non funziona: apri il link manualmente nell’app Expo Go.

Se rete è un problema: valuta modalità tunnel/hotspot (piano B).

Simulatore/emulator sono opzionali per il Modulo 1.

```
npm start

# opzioni utili (se necessario)
# expo start --tunnel
# expo start --lan
```

Zoom: cosa condividere

Setup consigliato (Mac + Android)

Condividi il PDF quando spieghi concetti e comandi.

Condividi VS Code quando scrivi/modifichi il codice.

Per mostrare il telefono: condividi la finestra scrcpy (mirroring) su Zoom.

Evita di condividere “tutto lo schermo” se non necessario: più ordine e meno distrazioni.

Pausa

10–15 minuti

Struttura di base di un progetto Expo

Cosa trovi dentro (e a cosa serve)

Spiegazione

package.json: comandi e dipendenze (cuore del progetto).

App.tsx: entry point della UI (schermata iniziale).

assets/: immagini e risorse statiche.

Consiglio: introdurre una cartella src/ per organizzare il codice.

```
myApp/
  package.json
  App.tsx (o App.js)
  assets/
  app.json (config Expo)
  src/ (consigliato)
    components/
    screens/
    styles/
```

app.json / app.config

A cosa serve in Expo

Contiene configurazioni dell’app (nome, icona, splash, id).

In Expo, molte impostazioni “di app” stanno qui anziché in file native.

È utile per personalizzare progetto senza entrare in IDE native.

In moduli successivi vedremo configurazioni più avanzate.

Componenti fondamentali

Blocchi base della UI

View: contenitore per layout.

Text: testo (obbligatorio per renderizzare stringhe).

Image: immagini locali o remote.

Pressable/Button: interazione (Pressable è più flessibile).

SafeAreaView: evita sovrapposizioni con notch e status bar.

Hello World — App.tsx minimale

Spiegazione

Punto di partenza: una schermata con testo.

SafeAreaView è consigliata per compatibilità UI.

Da qui si costruisce layout e componenti.

Durante la live: fai cambiare testo e stile per vedere Fast Refresh.

```
import { SafeAreaView, Text } from
'react-native';

export default function App() {
  return (
    <SafeAreaView style={{ flex: 1,
justifyContent: 'center', alignItems:
'center' }}>
      <Text>React Native — Hello
World</Text>
    </SafeAreaView>
  );
}
```

Stili in React Native

Concetti chiave

Non è CSS: è un subset di proprietà in JS (StyleSheet o oggetti inline).

Layout basato su Flexbox (di default flexDirection = 'column').

Unità: numeri (densità indipendente), non px espliciti.

Buona pratica: definire una scala di spaziatura e tipografia.

StyleSheet — separare stile dalla UI

Spiegazione

StyleSheet.create aiuta a organizzare e riusare stili.

È più leggibile che avere molti oggetti inline.

In team, favorisce consistenza (naming e scala).

Consiglio: creare file styles.ts per schermi e componenti.

```
import { StyleSheet } from 'react-
native';

export const styles =
StyleSheet.create({
  screen: { flex: 1, padding: 16 },
  title: { fontSize: 24, fontWeight:
'700' },
  subtitle: { marginTop: 4, fontSize:
14, opacity: 0.7 },
});
```

Image: remoto e locale

Spiegazione

Immagini remote: source={{ uri: 'https://...' }} (dipende dalla rete).

Immagini locali: require('./assets/img.png') (più stabile in aula).

Ricorda: width e height sono necessari.

In live: mostra un asset locale per evitare problemi di rete.

```
import { Image } from 'react-native';

export function Avatar() {
  return (
    <Image

source={require('./assets/avatar.png')}
      style={{ width: 80, height: 80,
borderRadius: 40 }}
    />
  );
}
```

Pressable: bottone personalizzato

Spiegazione

Pressable è più flessibile di Button (stile e stati).

Puoi creare un componente riutilizzabile (PrimaryButton).

onPress collega UI a logica (es. console.log).

In live: aggiungi effetto pressed per mostrare interazione.

```
import { Pressable, Text } from 'react-
native';

export function PrimaryButton({ label,
onPress }) {
  return (
    <Pressable
      onPress={onPress}
      style={({ pressed }) => ({
        padding: 12,
        borderRadius: 12,
        opacity: pressed ? 0.85 : 1,
      })}
    >
      <Text style={{ fontWeight:
'700' }}>{label}</Text>
    </Pressable>
  );
}
```

Problemi comuni (checklist)

Cosa controllare rapidamente

Expo Go non vede il progetto: rete diversa → allineare rete o usare tunnel.

QR non funziona: aprire link manualmente o cambiare modalità.

Cache Metro: riavviare con reset cache se necessario.

Immagini remote: rete instabile → usare assets locali.

Se scrcpy non va: continua la lezione senza mirroring (piano B).

Riepilogo del Modulo 1

React Native: app iOS/Android con JavaScript e componenti nativi.

Expo vs CLI: due workflow con compromessi diversi.

Setup: Node/npm + Expo Go (minimo), adb/scrcpy per live su Zoom (opzionale).

Struttura progetto: package.json, App.tsx, assets, src/.

Componenti base: View, Text, Image, Pressable.

Domande di ripasso

Spunta le risposte che ritieni corrette.

Le domande verificano concetti chiave del modulo.

Non sono presenti soluzioni in questo documento.

Durante la revisione si discute la motivazione delle scelte.

Domande di ripasso

Domanda 1 di 5

1. Quale affermazione descrive meglio React Native?

- ☐ Sistema di database integrato per mobile
- ☐ Framework per creare siti web statici senza JavaScript
- ☐ Libreria solo per Android basata su Java
- ☐ Framework per creare app iOS/Android con JavaScript e componenti nativi

Domande di ripasso

Domanda 2 di 5

2. Quale descrizione di Expo è corretta?

- ☐ Sostituto di React che elimina componenti e stato
- ☐ Tecnologia basata su WebView per app ibride
- ☐ Tooling che semplifica avvio e test di app React Native, anche con Expo Go
- ☐ Sistema operativo mobile alternativo

Domande di ripasso

Domanda 3 di 5

3. Quale affermazione su React Native CLI è corretta?

- ☐ Non permette integrazioni native avanzate
- ☐ Funziona solo su Windows e non supporta Android
- ☐ Crea e gestisce progetti bare con cartelle iOS/Android e richiede toolchain completa
- ☐ Sostituisce npm e non usa package.json

Domande di ripasso

Domanda 4 di 5

4. Quale descrizione di package.json è corretta?

- ☐ File binario che contiene l’app compilata
- ☐ File che descrive progetto, dipendenze e script eseguibili
- ☐ Database locale per salvare dati utente
- ☐ Configurazione esclusiva di Android Studio

Domande di ripasso

Domanda 5 di 5

5. Quali sono componenti base tipici in React Native?

- ☐ View e Text
- ☐ div e span
- ☐ table e tr
- ☐ form e input (HTML)

Cosa succede “in background”

Panoramica end-to-end (dev e produzione)

Scrivi codice TS/JS + JSX nel progetto.

In sviluppo: Babel/Metro generano il bundle e lo servono via Dev Server.

Il device carica il bundle (Expo Go o app) e lo esegue nel runtime JS.

In produzione: il bundle viene generato e impacchettato nell’app durante la build nativa.

React Native collega JS e mondo nativo (UI e moduli) tramite Bridge/JSI.

Transpile vs bundle vs build

Definizioni operative

Transpile: trasformare il codice (TypeScript/JSX → JavaScript compatibile).

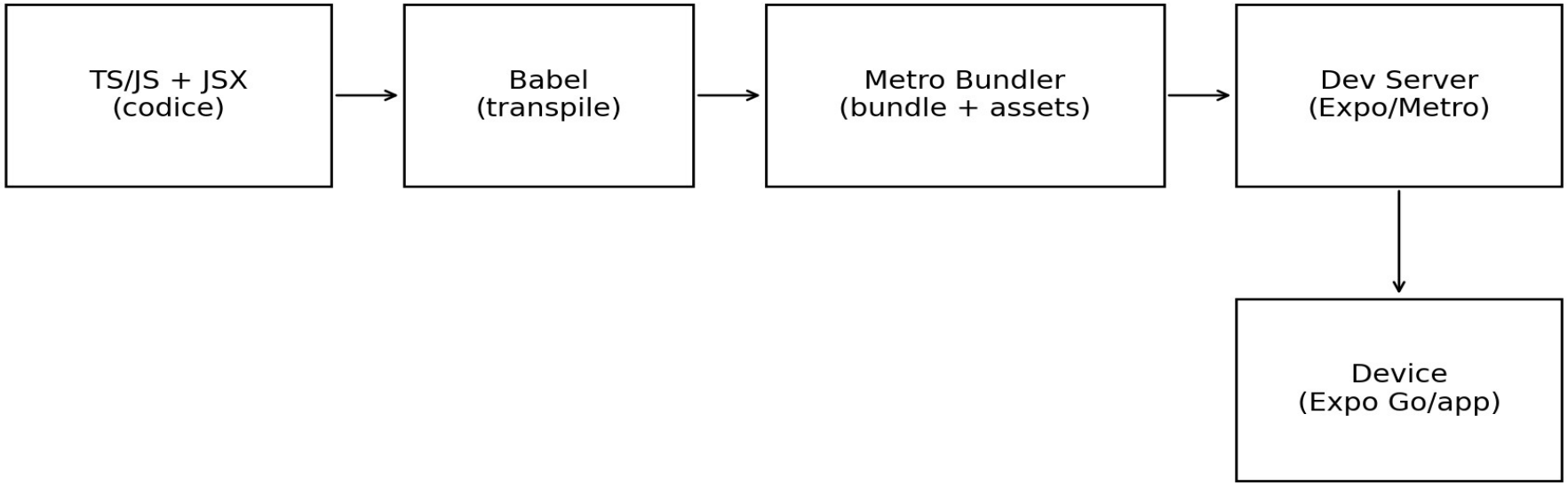
Bundle: unire moduli JS e assets in un output eseguibile dal runtime (Metro).

Build: compilare l’app nativa (Android/iOS) e includere bundle e risorse.

Runtime: l’ambiente che esegue il JavaScript (Hermes o JavaScriptCore).

Pipeline in sviluppo (Expo/Metro)

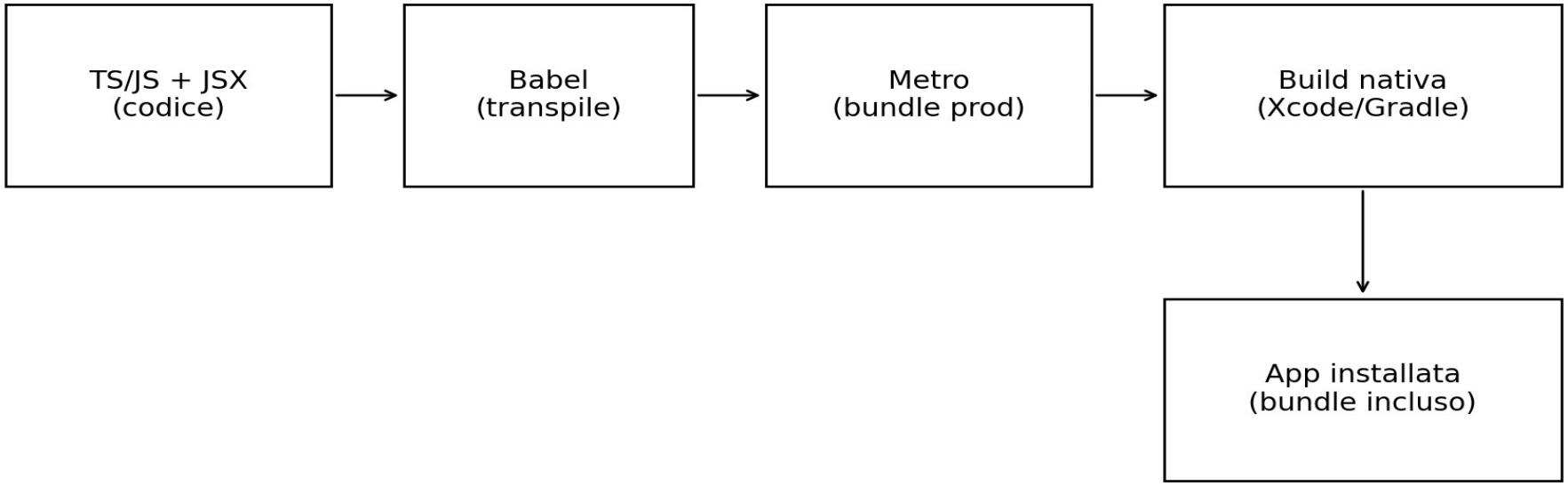
Step-by-step



Dev: transpile + bundle + Dev Server → device scarica il bundle; Fast Refresh evita rebuild completa.

Pipeline in produzione

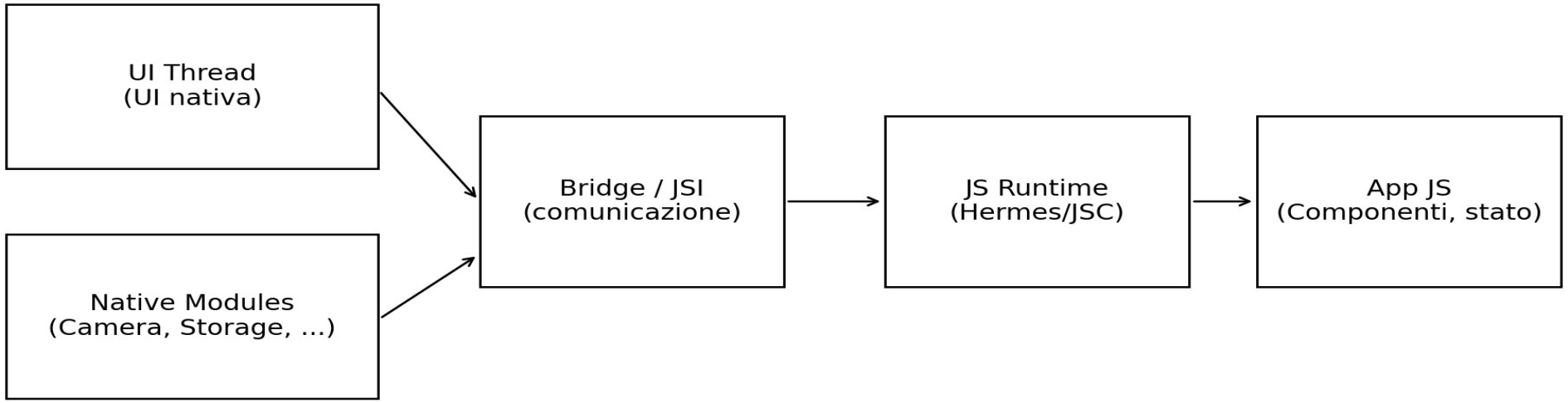
Step-by-step



Prod: bundle generato e incluso nella build nativa (Xcode/Gradle o build Expo/EAS).

Runtime JS ↔ Native

Architettura semplificata



Il codice JS gira nel runtime; UI e moduli sono nativi. Comunicazione tramite Bridge/JSI.

Fast Refresh e cache

Perché a volte “non si aggiorna”

Fast Refresh aggiorna rapidamente UI e logica senza ricompilare tutto.

Se lo stato si comporta “strano”, un reload completo spesso risolve.

Metro usa cache: in caso di errori strani, reset cache può aiutare.

In live: un progetto backup evita di perdere tempo su edge case.