Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: RNAT • **Updated:** 04 Nov 2026
https://mockattempt.com/courses/react-native-training

RNAT • CROSS-PLATFORM MOBILE ENGINEERING (IOS & ANDROID) **BEGINNER TO SENIOR REACT NATIVE & EXPO ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

React Native Training

Master production React Native & Expo from Hermes bytecode, JSI, Fabric & TurboModules to View/Text primitives, Props/Hooks, StyleSheet, Yoga Flexbox, TextInput, Pressable, FlatList/SectionList virtualization, Image caching, Native-Driver Animations, WebView security & GPS Geolocation — with 244 Topic Mock Tests (10,440 MCQs) and Spoken Voice + MockAttempt AI Interviews at ₹3,799 (Incl. 18% GST).

Duration: 15–17 Weeks • Live + Hands-On Projects (207 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Subtopics:** 336+ • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹3,799 (Net Tuition ₹3,219 + Included 18% GST ₹580) • **Split Option:** 2 × ₹1,900 Split • **Effective ROI:** ~₹18/Guided Hr (~₹90/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (JavaScript ES6+ & TypeScript): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: React Native 0.76+ (New Architecture) • Hermes Engine, JSI, Fabric & TurboModules • Expo SDK 52, Expo Router & EAS Build • Yoga C++ Layout Engine & StyleSheet • FlatList, SectionList & Pressable • Animated (useNativeDriver) & Reanimated • WebView Hardening, Alerts & Geolocation

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Explain React Native's New Architecture: how C++ JSI, synchronous Fabric Shadow Trees, TurboModules, and Hermes AOT bytecode replace the legacy async JSON bridge
2. Build responsive, notch-safe iOS & Android interfaces using View, Text, SafeAreaView, StyleSheet.create, useWindowDimensions, and Yoga Flexbox
3. Optimize 10,000-row virtualized FlatList/SectionList feeds, 60/120fps native-driver animations, hardened WebViews, and permission-safe GPS tracking
4. Clear 244 Timed Mock Exams (10,000 React Native MCQs) and 200 Spoken Voice + MockAttempt AI Mobile Engineering Interviews
5. Strict TypeScript React Native components with React Navigation native stacks, deep linking, and auth flows.
6. Crack LeetCode and HackerRank problems in JavaScript ES6+ & TypeScript with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. High-performance lists (`FlashList` / tuned `FlatList`), Reanimated UI-thread animations, and keyboard-safe forms.
8. Server-state caching & offline mutation queues via TanStack Query paired with lightweight Zustand client stores.

Eligibility & Prerequisites: Working knowledge of modern ES6+ JavaScript and basic React fundamentals (JSX, components, hooks) • Any Windows, macOS, or Linux laptop with Node.js 20+ LTS, VS Code, and Expo Go / Android Studio / Xcode Simulator • No prior React Native industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 42 LIVE TOPICS • 336+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (8 Topics / 64 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (8 Topics / 64 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (14 Topics / 112 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: React Native New Architecture (Hermes, JSI, Fabric, TurboModules) & Expo Setup

Understand how React Native renders genuine native iOS (`UIView`) and Android (`android.view.View`) components via C++ JSI, Fabric, TurboModules, and Hermes AOT bytecode, and set up Expo Development Builds.

Objectives: ✓ React Native Overview — Native OEM Views (`UIView` / `android.view.View`), Hermes, JSI & Fabric — Why React Native is NOT a WebView wrapper: React components reconcile into C++ Yoga/Fabric Shadow Nodes mounted as real platform views, powered by Hermes bytecode and zero-serialization C++ JSI. • ✓ Environment Setup — Expo SDK (`create-expo-app`), Development Builds (CNG), Metro & Fast Refresh — Bootstrap modern React Native apps as recommended by the official React Native docs using Expo (`npx create-expo-app`), Continuous Native Generation (`npx expo run:android` / `run:ios`), Metro bundler, and Fast Refresh. • ✓ Build, verify, and ship Module Capstone: New Architecture Runtime Inspector & Expo Config Validator.

1.1 React Native Overview — Native OEM Views (`UIView` / `android.view.View`), Hermes, JSI & Fabric

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Real UIView & android.view.View & C++ JavaScript Interface (JSI) • Core Syntax, CLI & Production APIs: JSI (JavaScript Interface) + Fabric Renderer + TurboModules + Hermes Engine • Deep-Dive Mechanics: Fabric synchronous C++ renderer & Lazy-loaded TurboModules • Advanced Patterns: Hermes AOT bytecode (.hbc) & JS Thread vs Main UI Thread • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Runtime Engine Inspector (Hermes + Fabric Detection) • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native Overview

Purpose: To understand and apply React Native Overview — Native OEM Views (`UIView` / `android.view.View`), Hermes, JSI & Fabric in real React Native systems — specifically: Why R...

What Topic Is Next → Next Topic 1.2: Environment Setup — Expo SDK (`create-expo-app`), Development Builds (CNG), Metro & Fast Refresh

1.2 Environment Setup — Expo SDK (`create-expo-app`), Development Builds (CNG), Metro & Fast Refresh

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Npx create-expo-app recommended flow & Expo Go vs Development Builds • Core Syntax, CLI & Production APIs: npx create-expo-app@latest + npx expo start + npx expo prebuild + metro.config.js • Deep-Dive Mechanics: Continuous Native Generation (CNG) & Metro JS bundler & resolver • Advanced Patterns: Fast Refresh hook preservation & Platform.OS & .ios.tsx / .android.tsx • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Platform-Specific Endpoint Resolver (`Platform.select`) • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for Environment Setup

Purpose: To understand and apply Environment Setup — Expo SDK (`create-expo-app`), Development Builds (CNG), Metro & Fast Refresh in real React Native systems — specifically: Boot...

What Topic Is Next → Module 1 Capstone (New Architecture Runtime Inspector & Expo Config Validator) → Next Topic 2.1: First React Native App — `<Text>` Containment Rule, Falsy `0` Crash & `SafeAreaProvider`

Module 1 Hands-On Capstone Project (100 Marks): New Architecture Runtime Inspector & Expo Config Validator — Create a React Native diagnostics helper inspecting global JSI/Fabric/Hermes runtime flags (`global.HermesInternal`, `global.nativeFabricUIManager`) and validating `app.json` bundle identifi...

MODULE 2 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Core Native Primitives — Hello World Entry Point, `<View>`, `<Text>` & Safe Areas

Build your first cross-platform screen without HTML tags (`<div>`/`<p>` do not exist!), master the strict `<Text>` string rule, notch-safe `SafeAreaProvider` / `useSafeAreaInsets`, and `StatusBar`.

Objectives: ✓ First React Native App — `<Text>` Containment Rule, Falsy `0` Crash & `SafeAreaProvider` — Understand why raw strings cannot sit directly inside `<View>`, how `{items.length && <List />}` crashes production apps when `length === 0`, and how `<Text>` inherits typography from parent `<Text>` nodes. • ✓ React Native `<View>` — Native Mapping, View Flattening (`collapsable`), `pointerEvents` & `onLayout` — Master React Native's fundamental layout box `<View>`: Fabric view flattening (`collapsable`), pass-through touch overlays (`pointerEvents="box-none"`), `onLayout` synchronous Fabric reads, and screen-reader grouping. • ✓ Build, verify, and ship Module Capstone: Notch-Aware Course Welcome Card with Zero Falsy-Zero `<View>` Crashes.

2.1 First React Native App — `<Text>` Containment Rule, Falsy `0` Crash & `SafeAreaProvider`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Text strings inside <Text> rule & {count > 0 && ...} falsy-0 fix • Core Syntax, CLI & Production APIs: <SafeAreaProvider> + <View> + <Text numberOfLines={2}> + StatusBar • Deep-Dive Mechanics: Spannable / NSAttributedString nesting & No View-to-Text style inheritance • Advanced Patterns: React-native-safe-area-context & AppRegistry.registerComponent • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Crash-Safe Conditional Badge & Nested `<Text>` Banner • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for First React Native App

Purpose: To understand and apply First React Native App — `<Text>` Containment Rule, Falsy `0` Crash & `SafeAreaProvider` in real React Native systems — specifically: Understand w...

What Topic Is Next → Next Topic 2.2: React Native `<View>` — Native Mapping, View Flattening (`collapsable`), `pointerEvents` & `onLayout`

2.2 React Native `<View>` — Native Mapping, View Flattening (`collapsable`), `pointerEvents` & `onLayout`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: View -> UIView / ViewGroup mapping & Fabric View Flattening optimization • Core Syntax, CLI & Production APIs: <View onLayout={e => e.nativeEvent.layout} pointerEvents="box-none" accessibilityRole="summa... • Deep-Dive Mechanics: PointerEvents="box-none" vs "none" & OnLayout { x, y, width, height } • Advanced Patterns: Default flexDirection: column & Accessible={true} VoiceOver grouping • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Overlay `pointerEvents` & `onLayout` Aspect Evaluator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<View>`

Purpose: To understand and apply React Native `<View>` — Native Mapping, View Flattening (`collapsable`), `pointerEvents` & `onLayout` in real React Native systems — specifically:...

What Topic Is Next → Module 2 Capstone (Notch-Aware Course Welcome Card with Zero Falsy-Zero `<View>` Crashes) → Next Topic 3.1: React Native Props — Read-Only Data Flow, ES6 Default Params, `children` & `React.memo`

Module 2 Hands-On Capstone Project (100 Marks): Notch-Aware Course Welcome Card with Zero Falsy-Zero `<View>` Crashes — Build a reusable `<CourseHeroCard>` using `<View>`, nested `<Text>` style inheritance, safe boolean rendering (`count > 0 ? ... : null`), and dynamic top/bottom safe-area inset padding.

MODULE 3 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Component Architecture — Unidirectional Props, Memoization & React Hooks State

Compose reusable mobile UI components with read-only `props`, default destructuring, `React.memo`, and immutable state updates via `useState`, functional updaters `setCount(c => c + 1)`, and `useReducer`.

Objectives: ✓ React Native Props — Read-Only Data Flow, ES6 Default Params, `children` & `React.memo` — Configure mobile components cleanly via immutable `props`, ES6 default parameter destructuring (replacing deprecated `defaultProps` in React 19), slot `children`, and `React.memo` shallow equality. • ✓ React Native State — `useState`, Functional Updaters, Immutable Arrays & `Object.is` Batching — Drive reactive UI updates with `useState` and `useReducer`, understand React 18/19 automatic state batching across async promises/timers, and eliminate stale closures and in-place mutation bugs. • ✓ Build, verify, and ship Module Capstone: Memoized Quiz Question Card & Stale-Closure-Safe Timer Hook Reducer.

3.1 React Native Props — Read-Only Data Flow, ES6 Default Params, `children` & `React.memo`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Unidirectional read-only props & ES6 default parameter destructuring • Core Syntax, CLI & Production APIs: function TopicCard({ title, free = false, onSelect }) + React.memo(TopicCard) • Deep-Dive Mechanics: Children slot composition & React.memo shallow Object.is check • Advanced Patterns: UseCallback reference stability & Avoiding prop drilling with Context • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Configurable `<CoursePriceTag>` Component with Default Props • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native Props

Purpose: To understand and apply React Native Props — Read-Only Data Flow, ES6 Default Params, `children` & `React.memo` in real React Native systems — specifically: Configure mob...

What Topic Is Next → Next Topic 3.2: React Native State — `useState`, Functional Updaters, Immutable Arrays & `Object.is` Batching

3.2 React Native State — `useState`, Functional Updaters, Immutable Arrays & `Object.is` Batching

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: UseState & Object.is bailout rule & Functional updater prev => next • Core Syntax, CLI & Production APIs: const [items, setItems] = useState([]) + setCount(prev => prev + 1) + useReducer • Deep-Dive Mechanics: Immutable spread [...prev, item] & React 18+ automatic async batching • Advanced Patterns: Derived state during render & Stale closure prevention • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Immutable Bookmark Toggle Reducer (Zero In-Place Mutation) • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native State

Purpose: To understand and apply React Native State — `useState`, Functional Updaters, Immutable Arrays & `Object.is` Batching in real React Native systems — specifically: Drive r...

What Topic Is Next → Module 3 Capstone (Memoized Quiz Question Card & Stale-Closure-Safe Timer Hook Reducer) → Next Topic 4.1: React Native StyleSheet — `StyleSheet.create`, Array Merging, `hairlineWidth` & Platform Shadows

Module 3 Hands-On Capstone Project (100 Marks): Memoized Quiz Question Card & Stale-Closure-Safe Timer Hook Reducer — Build a memoized `<QuestionOptionCard>` paired with a pure quiz state reducer (`SELECT_OPTION`, `TICK_SECOND`, `SUBMIT_QUIZ`) that guarantees immutable transitions and zero stale-closure sco...

MODULE 4 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: `StyleSheet` Architecture, Cross-Platform Shadows & Responsive Screen Dimensions

Write clean, static `StyleSheet.create()` definitions, unify iOS shadow props with Android `elevation`, and build foldable/rotation-responsive layouts using `useWindowDimensions()` and `PixelRatio`.

Objectives: ✓ React Native StyleSheet — `StyleSheet.create`, Array Merging, `hairlineWidth` & Platform Shadows — Style components with unitless density-independent numbers, camelCase CSS subsets, `StyleSheet.absoluteFillObject`, `StyleSheet.hairlineWidth` 1-physical-pixel dividers, and cross-platform shadows. • ✓ Height & Width — DP Units, Flex Proportions, Percentage Sizing & `useWindowDimensions()` — Choose between fixed density-independent dimensions, proportional `flex` weights, percentage strings (`"50%"`), `aspectRatio`, and rotation-reactive `useWindowDimensions()`. • ✓ Build, verify, and ship Module Capstone: Cross-Platform Shadow Elevation Token System & Foldable Breakpoint Hook.

4.1 React Native StyleSheet — `StyleSheet.create`, Array Merging, `hairlineWidth` & Platform Shadows

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: StyleSheet.create static allocation & Unitless density-independent dp • Core Syntax, CLI & Production APIs: StyleSheet.create({ ... }) + StyleSheet.hairlineWidth + StyleSheet.absoluteFillObject • Deep-Dive Mechanics: Array precedence [base, conditional] & StyleSheet.hairlineWidth crisp border • Advanced Patterns: IOS shadow* vs Android elevation & No shorthand border: string rule • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Cross-Platform Card Shadow & Elevation Token Generator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native StyleSheet

Purpose: To understand and apply React Native StyleSheet — `StyleSheet.create`, Array Merging, `hairlineWidth` & Platform Shadows in real React Native systems — specifically: Styl...

What Topic Is Next → Next Topic 4.2: Height & Width — DP Units, Flex Proportions, Percentage Sizing & `useWindowDimensions()`

4.2 Height & Width — DP Units, Flex Proportions, Percentage Sizing & `useWindowDimensions()` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Unitless pt / dp sizing & UseWindowDimensions() reactive hook • Core Syntax, CLI & Production APIs: const { width, height, scale, fontScale } = useWindowDimensions() + aspectRatio • Deep-Dive Mechanics: Dimensions.get static snapshot trap & Percentage width/height strings • Advanced Patterns: AspectRatio media framing & PixelRatio.getPixelSizeForLayoutSize • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Responsive Grid & Retina CDN Image Width Calculator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for Height

Purpose: To understand and apply Height & Width — DP Units, Flex Proportions, Percentage Sizing & `useWindowDimensions()` in real React Native systems — specifically: Choose betwe...

What Topic Is Next → Module 4 Capstone (Cross-Platform Shadow Elevation Token System & Foldable Breakpoint Hook) → Next Topic 5.1: Layout & Yoga Flexbox — `flexDirection: "column"` Default, `gap` & Text `flexShrink: 1`

Module 4 Hands-On Capstone Project (100 Marks): Cross-Platform Shadow Elevation Token System & Foldable Breakpoint Hook — Create a design-token helper generating unified iOS + Android elevation style objects for levels `1..5`, paired with a `useWindowDimensions`-compatible breakpoint classifier handling portrai...

MODULE 5 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Yoga C++ Flexbox Engine & Controlled Mobile `<TextInput>` Forms

Master Meta's C++ **Yoga** Flexbox defaults (`flexDirection: "column"`, `gap`, `flex: 1` shorthand) and build keyboard-aware `<TextInput>` forms with `KeyboardAvoidingView` and focus ref chaining.

Objectives: ✓ Layout & Yoga Flexbox — `flexDirection: "column"` Default, `gap` & Text `flexShrink: 1` — Master Yoga C++ layout rules: `flexDirection`, `justifyContent` (main axis), `alignItems` (cross axis), modern `gap` spacing, and why `flex: 1` inside a `<View>` needs a bounded parent height. • ✓ React Native `<TextInput>` — `onChangeText`, IME `keyboardType`, Focus Refs & `KeyboardAvoidingView` — Capture text cleanly via `value` + `onChangeText` (note: passes the string directly, NOT `e.target.value`!), configure mobile keyboards, chain focus with `useRef`, and prevent soft keyboards from covering inputs. • ✓ Build, verify, and ship Module Capstone: Keyboard-Safe OTP & Login Screen State Controller with Yoga Row Text-Wrap Guard.

5.1 Layout & Yoga Flexbox — `flexDirection: "column"` Default, `gap` & Text `flexShrink: 1` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Yoga C++ background layout engine & FlexDirection: column default • Core Syntax, CLI & Production APIs: flexDirection, justifyContent, alignItems, gap, flex: 1, flexShrink: 1 • Deep-Dive Mechanics: JustifyContent vs alignItems axes & Gap / rowGap / columnGap support • Advanced Patterns: FlexShrink: 1 row text wrapping & Position: absolute parent-relative • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Yoga Main/Cross Axis & Row Overflow Validator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for Layout

Purpose: To understand and apply Layout & Yoga Flexbox — `flexDirection: "column"` Default, `gap` & Text `flexShrink: 1` in real React Native systems — specifically: Master Yoga C...

What Topic Is Next → Next Topic 5.2: React Native `<TextInput>` — `onChangeText`, IME `keyboardType`, Focus Refs & `KeyboardAvoidingView`

5.2 React Native `<TextInput>` — `onChangeText`, IME `keyboardType`, Focus Refs & `KeyboardAvoidingView` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: OnChangeText(string) vs web onChange & AutoCapitalize="none" email rule • Core Syntax, CLI & Production APIs: <TextInput value={v} onChangeText={setV} keyboardType="email-address" autoCapitalize="none"... • Deep-Dive Mechanics: SecureTextEntry password masking & ReturnKeyType + onSubmitEditing focus • Advanced Patterns: KeyboardAvoidingView iOS vs Android & Keyboard.dismiss() tap-outside • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Controlled `<TextInput>` Email Normalizer & Keyboard Config • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<TextInput>`

Purpose: To understand and apply React Native `<TextInput>` — `onChangeText`, IME `keyboardType`, Focus Refs & `KeyboardAvoidingView` in real React Native systems — specifically:...

What Topic Is Next → Module 5 Capstone (Keyboard-Safe OTP & Login Screen State Controller with Yoga Row Text-Wrap Guard) → Next Topic 6.1: React Native `<Button>` — Native `UIBarButton` / Material Button Rendering & Customization Limits

Module 5 Hands-On Capstone Project (100 Marks): Keyboard-Safe OTP & Login Screen State Controller with Yoga Row Text-Wrap Guard — Create a controlled login form validation hook handling email normalization (`trim` + lowercase), password masking toggles, and platform-specific `KeyboardAvoidingView` behavior (`"padding"`...

MODULE 6 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Touch Interactions — Native `<Button>` Limits vs Modern `<Pressable>` & `hitSlop`

Understand why `<Button>` is only for quick prototypes and why production design systems standardize on `<Pressable>` (replacing legacy `TouchableOpacity` / `TouchableHighlight`) with `android_ripple` and `hitSlop`.

Objectives: ✓ React Native `<Button>` — Native `UIBarButton` / Material Button Rendering & Customization Limits — Use the built-in `<Button title="..." onPress={...} color="..." disabled />` for quick prototypes and alert actions while understanding why it purposefully rejects custom `style`, border radius, and icons. • ✓ Touchables & `<Pressable>` — `TouchableOpacity` vs Future-Proof `<Pressable>`, `android_ripple` & `hitSlop` — Master touch handling from `TouchableOpacity` / `TouchableHighlight` / `TouchableWithoutFeedback` to the modern core `<Pressable>` API supporting `onPressIn`, `onLongPress`, `({ pressed })` render callbacks, `android_ripple`, and `hitSlop`. • ✓ Build, verify, and ship Module Capstone: Design-System `<AcademyButton>` Built on

`<Pressable>` with Ripple, Loading & `hitSlop`.

6.1 React Native `<Button>` — Native `UIBarButton` / Material Button Rendering & Customization Limits [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Title & onPress required props & No style prop on built-in Button • Core Syntax, CLI & Production APIs: <Button title="Start Exam" color="#2563EB" disabled={isLocked} onPress={handleStart} /> • Deep-Dive Mechanics: IOS text tint vs Android bg color & Disabled boolean state • Advanced Patterns: AccessibilityLabel screen-reader & When to upgrade to Pressable • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Cross-Platform `<Button>` Color Semantics Mapper • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<Button>`

Purpose: To understand and apply React Native `<Button>` — Native `UIBarButton` / Material Button Rendering & Customization Limits in real React Native systems — specifically: Use...

What Topic Is Next → Next Topic 6.2: Touchables & `<Pressable>` — `TouchableOpacity` vs Future-Proof `<Pressable>`, `android_ripple` & `hitSlop`

6.2 Touchables & `<Pressable>` — `TouchableOpacity` vs Future-Proof `<Pressable>`, `android_ripple` & `hitSlop` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Pressable ({ pressed }) render prop & Android_ripple hardware Ink ripple • Core Syntax, CLI & Production APIs: <Pressable onPress={fn} hitSlop={10} android_ripple={{ color }} style={({ pressed }) => [..... • Deep-Dive Mechanics: HitSlop invisible tap expansion & OnPressIn -> onLongPress -> onPressOut • Advanced Patterns: TouchableOpacity legacy comparison & AccessibilityRole="button" + busy state • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: `<Pressable>` Minimum 44dp `hitSlop` Auto-Expander • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for Touchables

Purpose: To understand and apply Touchables & `<Pressable>` — `TouchableOpacity` vs Future-Proof `<Pressable>`, `android_ripple` & `hitSlop` in real React Native systems — specifi...

What Topic Is Next → Module 6 Capstone (Design-System `<AcademyButton>` Built on `<Pressable>` with Ripple, Loading & `hitSlop`) → Next Topic 7.1: React Native `<ScrollView>` — `contentContainerStyle`, `keyboardShouldPersistTaps` & Eager Mounting

Module 6 Hands-On Capstone Project (100 Marks): Design-System `<AcademyButton>` Built on `<Pressable>` with Ripple, Loading & `hitSlop` — Engineer a reusable cross-platform button state helper supporting `primary`, `outline`, and `ghost` variants, pressed-opacity scaling on iOS, native Material `android_ripple` on Android, `di...

MODULE 7 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Scroll Containers (`<ScrollView>`) vs High-Speed Virtualized `<FlatList>` Feeds

Know exactly when to use eager `<ScrollView>` (`contentContainerStyle`, `RefreshControl`) vs windowed `<FlatList>` (`keyExtractor`, `getItemLayout`, `windowSize`) to scroll 10,000+ rows at 60/120fps.

Objectives: ✓ React Native `<ScrollView>` — `contentContainerStyle`, `keyboardShouldPersistTaps` & Eager Mounting — Use `<ScrollView>` for bounded forms and article pages (< 25 items), understand `style` vs `contentContainerStyle`, pull-to-refresh (`RefreshControl`), and `keyboardShouldPersistTaps="handled"`. • ✓ React Native `<FlatList>` — `VirtualizedList` Windowing, `keyExtractor` & `getItemLayout` — Render massive datasets smoothly with `<FlatList>`: `VirtualizedList` viewport windowing (`windowSize`, `initialNumToRender`, `maxToRenderPerBatch`), `ListEmptyComponent`, `ListHeaderComponent`, and O(1) `getItemLayout`. • ✓ Build, verify, and ship Module Capstone: O(1) `getItemLayout` Virtualized Question Navigator & Pull-to-Refresh Feed.

7.1 React Native `<ScrollView>` — `contentContainerStyle`, `keyboardShouldPersistTaps` & Eager Mounting [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Eager O(N) child mounting & Style vs contentContainerStyle • Core Syntax, CLI & Production APIs: <ScrollView contentContainerStyle={{ padding: 16 }} keyboardShouldPersistTaps="handled" refr... • Deep-Dive Mechanics: FlexGrow: 1 vs flex: 1 inside scroll & KeyboardShouldPersistTaps="handled" • Advanced Patterns: RefreshControl pull-to-refresh & PagingEnabled horizontal carousels • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: `ScrollView` vs `FlatList` Threshold & Prop Validator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<ScrollView>`

Purpose: To understand and apply React Native `<ScrollView>` — `contentContainerStyle`, `keyboardShouldPersistTaps` & Eager Mounting in real React Native systems — specifically: U...

What Topic Is Next → Next Topic 7.2: React Native `<FlatList>` — `VirtualizedList` Windowing, `keyExtractor` & `getItemLayout`

7.2 React Native `<FlatList>` — `VirtualizedList` Windowing, `keyExtractor` & `getItemLayout` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: VirtualizedList viewport windowing & Stable string keyExtractor • Core Syntax, CLI & Production APIs: <FlatList data={rows} keyExtractor={i => String(i.id)} renderItem={renderRow} getItemLayout=... • Deep-Dive Mechanics: O(1) getItemLayout(data, index) & ListHeaderComponent vs nested ScrollView • Advanced Patterns: Memoized renderItem + React.memo row & OnEndReachedThreshold pagination • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Exact `<FlatList>` `getItemLayout` Calculator (Row + Separator) • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<FlatList>`

Purpose: To understand and apply React Native `<FlatList>` — `VirtualizedList` Windowing, `keyExtractor` & `getItemLayout` in real React Native systems — specifically: Render mass...

What Topic Is Next → Module 7 Capstone (O(1) `getItemLayout` Virtualized Question Navigator & Pull-to-Refresh Feed) → Next Topic 8.1: React Native `<SectionList>` — `{ data: [...] }` Schema, `renderSectionHeader` & Sticky Headers

Module 7 Hands-On Capstone Project (100 Marks): O(1) `getItemLayout` Virtualized Question Navigator & Pull-to-Refresh Feed — Implement `<FlatList>` optimization helpers including deterministic `getItemLayout` offset calculation (accounting for row height + `ItemSeparatorComponent` height) and infinite-scroll curso...

MODULE 8 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Grouped `<SectionList>` Headers & High-Performance `<Image>` Caching

Render sticky-header grouped curricula/contacts using `<SectionList>` (`sections: [{ title, data }]`), and load local/remote images cleanly without layout shifts or memory spikes.

Objectives: ✓ React Native `<SectionList>` — `{ data: [...] }` Schema, `renderSectionHeader` & Sticky Headers — Display virtualized grouped lists (alphabetical contacts, date-grouped transactions, course modules -> topics) using `<SectionList>` with `renderSectionHeader` and `stickySectionHeadersEnabled`. • ✓ React Native `<Image>` — Static `require()` vs Remote `{ uri }`, `resizeMode` & `expo-image` Caching — Render bundled `@2x`/`@3x` assets via static `require()` and remote HTTPS images via `source={{ uri }}`, master `resizeMode` (`cover`, `contain`), `ImageBackground`, and `expo-image` SDWebImage/Glide disk caching. • ✓ Build, verify, and ship Module Capstone: Grouped Curriculum `<SectionList>` Transformer & Remote `<Image>` Dimension Guard.

8.1 React Native `<SectionList>` — `{ data: [...] }` Schema, `renderSectionHeader` & Sticky Headers [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Mandatory section.data array key & RenderSectionHeader({ section }) • Core Syntax, CLI & Production APIs: <SectionList sections={[{ title, data: [...] }]} renderItem={...} renderSectionHeader={...}... • Deep-Dive Mechanics: StickySectionHeadersEnabled parity & SectionSeparatorComponent vs ItemSeparator • Advanced Patterns: VirtualizedList backing engine & ScrollToLocation({ sectionIndex, itemIndex }) • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Curriculum Topic Grouping Transformer for `<SectionList>` • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<SectionList>`

Purpose: To understand and apply React Native `<SectionList>` — `{ data: [...] }` Schema, `renderSectionHeader` & Sticky Headers in real React Native systems — specifically: Displ...

What Topic Is Next → Next Topic 8.2: React Native `<Image>` — Static `require()` vs Remote `{ uri }`, `resizeMode` & `expo-image` Caching

8.2 React Native `<Image>` — Static `require()` vs Remote `{ uri }`, `resizeMode` & `expo-image` Caching [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: 0x0 remote { uri } dimension rule & Static require() @2x/@3x bundling • Core Syntax, CLI & Production APIs: <Image source={{ uri: "https://..." }} style={{ width: 120, height: 80 }} resizeMode="cover"... • Deep-Dive Mechanics: No dynamic strings inside require() & ResizeMode: cover vs contain • Advanced Patterns: IOS ATS https:// requirement & Expo-image memory + disk caching • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Remote `<Image>` ATS & Zero-Dimension Safety Validator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<Image>`

Purpose: To understand and apply React Native `<Image>` — Static `require()` vs Remote `{ uri }`, `resizeMode` & `expo-image` Caching in real React Native systems — specifically:...

What Topic Is Next → Module 8 Capstone (Grouped Curriculum `<SectionList>` Transformer & Remote `<Image>` Dimension Guard) → Next Topic 9.1: React Native Animations — `Animated.Value`, `useNativeDriver: true`, `interpolate` & Reanimated

Module 8 Hands-On Capstone Project (100 Marks): Grouped Curriculum `<SectionList>` Transformer & Remote `<Image>` Dimension Guard — Build a flat-to-grouped curriculum transformer outputting valid `<SectionList>` `sections` arrays with module completion counts, paired with a remote `<Image>` source validator enforcing HTT...

MODULE 9 OF 16 • STAGE 3 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: 60/120fps Native-Driver Animations (`Animated`) & Hardened `<WebView>` Integration

Drive silky-smooth 60/120fps animations off the JS thread using `Animated.timing` / `Animated.spring` with `useNativeDriver: true` (and `LayoutAnimation` / Reanimated), and embed secure two-way `<WebView>` flows.

Objectives: ✓ React Native Animations — `Animated.Value`, `useNativeDriver: true`, `interpolate` & Reanimated — Animate `opacity` and `transform` (`translateX/Y`, `scale`, `rotate`) directly on the native UI thread via `useNativeDriver: true` so animations stay 60/120fps even if the JS thread is busy. • ✓ React Native `<WebView>` — `react-native-webview`, `originWhitelist` & `postMessage` Bridge Security — Embed web checkout/OAuth/HTML content using community `react-native-webview` (`WKWebView` on iOS, Chromium `WebView` on Android), enforce strict `originWhitelist` rules, and pass typed messages via `window.ReactNativeWebView.postMessage`. • ✓ Build, verify, and ship Module Capstone: Native-Driver Progress Spring Interpolator & Secure `<WebView>` Origin Guard.

9.1 React Native Animations — `Animated.Value`, `useNativeDriver: true`, `interpolate` & Reanimated [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: UseNativeDriver: true UI-thread ticks & Opacity & transform-only native rule • Core Syntax, CLI & Production APIs: useAnimatedValue(0) + Animated.spring(val, { toValue: 1, useNativeDriver: true }).start() • Deep-Dive Mechanics: UseAnimatedValue / useRef stability & Val.interpolate({ inputRange, outputRange }) • Advanced Patterns: Animated.parallel / sequence / stagger & Reanimated 3 UI-thread worklets • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Clamped Linear `Animated.interpolate` & Native-Driver Validator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native Animations

Purpose: To understand and apply React Native Animations — `Animated.Value`, `useNativeDriver: true`, `interpolate` & Reanimated in real React Native systems — specifically: Anima...

What Topic Is Next → Next Topic 9.2: React Native `<WebView>` — `react-native-webview`, `originWhitelist` & `postMessage` Bridge Security

9.2 React Native `<WebView>` — `react-native-webview`, `originWhitelist` & `postMessage` Bridge Security [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: React-native-webview WKWebView/Chromium & Strict originWhitelist hardening • Core Syntax, CLI & Production APIs: <WebView source={{ uri }} originWhitelist={["https://checkout.mockattempt.com"]} onMessage={... • Deep-Dive Mechanics: Window.ReactNativeWebView.postMessage & OnMessage origin + JSON try/catch check • Advanced Patterns: OnShouldStartLoadWithRequest deep-links & Never render WebView inside FlatList • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Hardened `<WebView>` `onMessage` Origin & Payload Verifier • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `<WebView>`

Purpose: To understand and apply React Native `<WebView>` — `react-native-webview`, `originWhitelist` & `postMessage` Bridge Security in real React Native systems — specifically:...

What Topic Is Next → Module 9 Capstone (Native-Driver Progress Spring Interpolator & Secure `<WebView>` Origin Guard) → Next Topic 10.1: React Native `Alert` — Imperative `Alert.alert()`, Android 3-Button Limit & Destructive Styles

Module 9 Hands-On Capstone Project (100 Marks): Native-Driver Progress Spring Interpolator & Secure `<WebView>` Origin Guard — Build an `Animated` interpolation mapper converting `[0, 1]` progress values into `scale` and `translateY` transforms compatible with `useNativeDriver: true`, plus a `<WebView>` navigation g...

MODULE 10 OF 16 • STAGE 3 • 11 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Native OS Dialogs (`Alert.alert`) & Permission-Safe GPS Geolocation

Trigger native `UIAlertController` / `AlertDialog` confirmation prompts cleanly across iOS & Android (`Alert.alert` 3-button Android rule!), and acquire live GPS coordinates with foreground/background permission flows and cleanup.

Objectives: ✓ React Native `Alert` — Imperative `Alert.alert()`, Android 3-Button Limit & Destructive Styles — Invoke native OS modal alerts via `Alert.alert(title, message, buttons, options)`, understand iOS `style: "destructive" | "cancel"` ordering, Android's strict 3-button `AlertDialog` cap, and why `Alert.prompt` is iOS-only. • ✓ React Native Geolocation — `expo-location` / Geolocation Permissions, Timeouts & `watchPosition` Cleanup — Request foreground/background location permissions (`NSLocationWhenInUseUsageDescription` + `ACCESS_FINE_LOCATION`), query GPS fixes with `enableHighAccuracy` and `maximumAge`, and unsubscribe watchers on unmount. • ✓ Build, verify, and ship Module Capstone: Cross-Platform Exam Exit Confirmation & Geofenced Attendance Verifier.

10.1 React Native `Alert` — Imperative `Alert.alert()`, Android 3-Button Limit & Destructive Styles [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Alert.alert(title, msg, buttons) & Android max 3 buttons rule • Core Syntax, CLI & Production APIs: Alert.alert("Submit Exam?", "Unanswered MCQs score 0.", [{ text: "Cancel", style: "cancel" }... • Deep-Dive Mechanics: Alert.prompt iOS-only limitation & Style: cancel vs destructive • Advanced Patterns: Android cancelable: true backdrop & Promise-wrapped confirm dialog helper • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Android-Safe `Alert.alert` 3-Button Contract Validator • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native `Alert`

Purpose: To understand and apply React Native `Alert` — Imperative `Alert.alert()`, Android 3-Button Limit & Destructive Styles in real React Native systems — specifically: Invoke...

What Topic Is Next → Next Topic 10.2: React Native Geolocation — `expo-location` / Geolocation Permissions, Timeouts & `watchPosition` Cleanup

10.2 React Native Geolocation — `expo-location` / Geolocation Permissions, Timeouts & `watchPosition` Cleanup [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: NSLocationWhenInUseUsageDescription & Android 12+ Coarse vs Fine location • Core Syntax, CLI & Production APIs: Location.requestForegroundPermissionsAsync() + getCurrentPositionAsync() / watchPositionAsyn... • Deep-Dive Mechanics: EnableHighAccuracy + timeout + maximumAge & WatchPosition unmount cleanup rule • Advanced Patterns: DistanceInterval hardware throttling & Linking.openSettings() denied recovery • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Geofence Radius Checker & Watcher Cleanup Tracker • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native Geolocation

Purpose: To understand and apply React Native Geolocation — `expo-location` / Geolocation Permissions, Timeouts & `watchPosition` Cleanup in real React Native systems — specifical...

What Topic Is Next → Module 10 Capstone (Cross-Platform Exam Exit Confirmation & Geofenced Attendance Verifier) → Next Topic 11.1: Bridgeless JSI Synchronous Calls & Fabric Concurrent Renderer

Module 10 Hands-On Capstone Project (100 Marks): Cross-Platform Exam Exit Confirmation & Geofenced Attendance Verifier — Build a cross-platform `Alert.alert` builder enforcing Android's 3-button ceiling (`[Neutral, Cancel, Destructive]`) and a Haversine GPS distance verifier with automatic `watchPosition` subs...

MODULE 11 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

React Native New Architecture (Fabric, JSI, TurboModules) & Navigation

Production-grade React Native Training engineering module covering React Native New Architecture (Fabric, JSI, TurboModules) & Navigation with hands-on labs and capstone build.

Objectives: ✓ Bridgeless JSI Synchronous Calls & Fabric Concurrent Renderer — Purpose: Understand why modern React Native achieves native-grade startup and gesture latency. • ✓ Typed React Navigation Stacks, Tabs, Modals & Universal Deep Links — Purpose: Route WhatsApp/SMS links directly to authenticated product or order screens. • ✓ TanStack Query + Zustand + MMKV Synchronous Persistence — Purpose: Read cached mobile state 30x faster than legacy AsyncStorage on app launch. • ✓ KeyboardAvoiding Patterns, SafeAreaInsets & Responsive Breakpoints — Purpose: Prevent notches and on-screen keyboards from covering input fields across devices.

11.1 Bridgeless JSI Synchronous Calls & Fabric Concurrent Renderer [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Fabric Concurrent Renderer • Integration with Typed React Navigation Stacks, Tabs, Modals & Universal Deep Links • Live Debugging, Telemetry & Technical Interview Trade • What Is Bridgeless JSI Synchronous Calls & Fabric Concurrent Renderer & Why Industry Teams Use It • In React Native Training, Bridgeless JSI Synchronous Calls & Fabric Concurrent Renderer exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Bridgeless JSI Synchronous Calls & Fabric Concurrent Renderer in real React Native systems — specifically: Understand why modern React Native achi...

What Topic Is Next → Next Topic 11.2: Typed React Navigation Stacks, Tabs, Modals & Universal Deep Links

11.2 Typed React Navigation Stacks, Tabs, Modals & Universal Deep Links [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Tabs • Integration with TanStack Query + Zustand + MMKV Synchronous Persistence • Live Debugging, Telemetry & Technical Interview Trade • What Is Typed React Navigation Stacks, Tabs, Modals & Universal Deep Links & Why Industry Teams Use It • In React Native Training, Typed React Navigation Stacks, Tabs, Modals & Universal Deep Links exists for one clear r... • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Typed React Navigation Stacks, Tabs, Modals & Universal Deep Links in real React Native systems — specifically: Route WhatsApp/SMS links directly...

What Topic Is Next → Next Topic 11.3: TanStack Query + Zustand + MMKV Synchronous Persistence

11.3 TanStack Query + Zustand + MMKV Synchronous Persistence [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with KeyboardAvoiding Patterns, SafeAreaInsets & Responsive Breakpoints • Live Debugging, Telemetry & Technical Interview Trade • What Is TanStack Query + Zustand + MMKV Synchronous Persistence & Why Industry Teams Use It • In React Native Training, TanStack Query + Zustand + MMKV Synchronous Persistence exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply TanStack Query + Zustand + MMKV Synchronous Persistence in real React Native systems — specifically: Read cached mobile state 30x faster than lega...

What Topic Is Next → Next Topic 11.4: KeyboardAvoiding Patterns, SafeAreaInsets & Responsive Breakpoints

11.4 KeyboardAvoiding Patterns, SafeAreaInsets & Responsive Breakpoints [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of SafeAreaInsets • Integration with Bridgeless JSI Synchronous Calls & Fabric Concurrent Renderer • Live Debugging, Telemetry & Technical Interview Trade • What Is KeyboardAvoiding Patterns, SafeAreaInsets & Responsive Breakpoints & Why Industry Teams Use It • In React Native Training, KeyboardAvoiding Patterns, SafeAreaInsets & Responsive Breakpoints exists for one clear r... • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply KeyboardAvoiding Patterns, SafeAreaInsets & Responsive Breakpoints in real React Native systems — specifically: Prevent notches and on-screen keyb...

What Topic Is Next → Module 11 Capstone (Type-Safe Deep-Linked Mobile Commerce Shell) → Next Topic 12.1: Cell Recycling with FlashList vs FlatList Memory Spikes

Module 11 Hands-On Capstone Project (100 Marks): Type-Safe Deep-Linked Mobile Commerce Shell — Design, implement, test, and document 'Type-Safe Deep-Linked Mobile Commerce Shell' applying every production concept from Module 11 (React Native New Architecture (Fabric, JSI, TurboModules...

MODULE 12 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

60fps Lists (FlashList), Reanimated UI-Thread Gestures & Offline Sync

Production-grade React Native Training engineering module covering 60fps Lists (FlashList), Reanimated UI-Thread Gestures & Offline Sync with hands-on labs and capstone build.

Objectives: ✓ Cell Recycling with FlashList vs FlatList Memory Spikes — Purpose: Eliminate blank white cells while flick-scrolling thousands of catalog cards. • ✓ Reanimated Worklets & Gesture Handler on the Native UI Thread — Purpose: Run buttery 60/120fps swipe-to-pay and bottom-sheet animations even when JS is busy. • ✓ NetInfo Connectivity Queues & SQLite/WatermelonDB Offline Storage — Purpose: Queue field updates offline and replay idempotently once 4G/Wi-Fi returns. • ✓ Expo SecureStore, Biometric Auth & SSL Pinning — Purpose: Harden mobile sessions and sensitive healthcare/fintech payloads on device.

12.1 Cell Recycling with FlashList vs FlatList Memory Spikes [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Reanimated Worklets & Gesture Handler on the Native UI Thread • Live Debugging, Telemetry & Technical Interview Trade • What Is Cell Recycling with FlashList vs FlatList Memory Spikes & Why Industry Teams Use It • In React Native Training, Cell Recycling with FlashList vs FlatList Memory Spikes exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Cell Recycling with FlashList vs FlatList Memory Spikes in real React Native systems — specifically: Eliminate blank white cells while flick-scrol...

What Topic Is Next → Next Topic 12.2: Reanimated Worklets & Gesture Handler on the Native UI Thread

12.2 Reanimated Worklets & Gesture Handler on the Native UI Thread [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Gesture Handler on the Native UI Thread • Integration with NetInfo Connectivity Queues & SQLite/WatermelonDB Offline Storage • Live Debugging, Telemetry & Technical Interview Trade • What Is Reanimated Worklets & Gesture Handler on the Native UI Thread & Why Industry Teams Use It • In React Native Training, Reanimated Worklets & Gesture Handler on the Native UI Thread exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Reanimated Worklets & Gesture Handler on the Native UI Thread in real React Native systems — specifically: Run buttery 60/120fps swipe-to-pay and...

What Topic Is Next → Next Topic 12.3: NetInfo Connectivity Queues & SQLite/WatermelonDB Offline Storage

12.3 NetInfo Connectivity Queues & SQLite/WatermelonDB Offline Storage [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of SQLite/WatermelonDB Offline Storage • Integration with Expo SecureStore, Biometric Auth & SSL Pinning • Live Debugging, Telemetry & Technical Interview Trade • What Is NetInfo Connectivity Queues & SQLite/WatermelonDB Offline Storage & Why Industry Teams Use It • In React Native Training, NetInfo Connectivity Queues & SQLite/WatermelonDB Offline Storage exists for one clear re... • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply NetInfo Connectivity Queues & SQLite/WatermelonDB Offline Storage in real React Native systems — specifically: Queue field updates offline and rep...

What Topic Is Next → Next Topic 12.4: Expo SecureStore, Biometric Auth & SSL Pinning

12.4 Expo SecureStore, Biometric Auth & SSL Pinning [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Biometric Auth • Integration with Cell Recycling with FlashList vs FlatList Memory Spikes • Live Debugging, Telemetry & Technical Interview Trade • What Is Expo SecureStore, Biometric Auth & SSL Pinning & Why Industry Teams Use It • In React Native Training, Expo SecureStore, Biometric Auth & SSL Pinning exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Expo SecureStore, Biometric Auth & SSL Pinning in real React Native systems — specifically: Harden mobile sessions and sensitive healthcare/fintec...

What Topic Is Next → Module 12 Capstone (Fluid Gesture-Driven Feed & Offline Order Sync App) → Next Topic 13.1: Graceful Runtime Permission Flows (Camera, Location, Notifications)

Module 12 Hands-On Capstone Project (100 Marks): Fluid Gesture-Driven Feed & Offline Order Sync App — Design, implement, test, and document 'Fluid Gesture-Driven Feed & Offline Order Sync App' applying every production concept from Module 12 (60fps Lists (FlashList), Reanimated UI-Thread Ges...

MODULE 13 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Native Permissions, Camera/Media Uploads, Push & Native Module Config

Production-grade React Native Training engineering module covering Native Permissions, Camera/Media Uploads, Push & Native Module Config with hands-on labs and capstone build.

Objectives: ✓ Graceful Runtime Permission Flows (Camera, Location, Notifications) — Purpose: Explain permission value upfront to maximize user opt-in and pass App Store review. • ✓ Chunked/Presigned Multipart Media Uploads & Image Compression — Purpose: Upload compressed receipts and photos reliably over flaky mobile networks. • ✓ FCM / APNs Push Notification Channels, Badges & Background Tap Routing — Purpose: Re-engage users reliably across foreground, background, and cold-killed app states. • ✓ Expo Config Plugins & Custom Native Android/iOS Manifest Tuning — Purpose: Customize native Gradle/Podfile capabilities cleanly without losing upgrade velocity.

13.1 Graceful Runtime Permission Flows (Camera, Location, Notifications) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Location • Integration with Chunked/Presigned Multipart Media Uploads & Image Compression • Live Debugging, Telemetry & Technical Interview Trade • What Is Graceful Runtime Permission Flows (Camera, Location, Notifications) & Why Industry Teams Use It • In React Native Training, Graceful Runtime Permission Flows (Camera, Location, Notifications) exists for one clear.... • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Graceful Runtime Permission Flows (Camera, Location, Notifications) in real React Native systems — specifically: Explain permission value upfront...

What Topic Is Next → Next Topic 13.2: Chunked/Presigned Multipart Media Uploads & Image Compression

13.2 Chunked/Presigned Multipart Media Uploads & Image Compression [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Image Compression • Integration with FCM / APNs Push Notification Channels, Badges & Background Tap Routing • Live Debugging, Telemetry & Technical Interview Trade • What Is Chunked/Presigned Multipart Media Uploads & Image Compression & Why Industry Teams Use It • In React Native Training, Chunked/Presigned Multipart Media Uploads & Image Compression exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Chunked/Presigned Multipart Media Uploads & Image Compression in real React Native systems — specifically: Upload compressed receipts and photos r...

What Topic Is Next → Next Topic 13.3: FCM / APNs Push Notification Channels, Badges & Background Tap Routing

13.3 FCM / APNs Push Notification Channels, Badges & Background Tap Routing [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Badges • Integration with Expo Config Plugins & Custom Native Android/iOS Manifest Tuning • Live Debugging, Telemetry & Technical Interview Trade • What Is FCM / APNs Push Notification Channels, Badges & Background Tap Routing & Why Industry Teams Use It • In React Native Training, FCM / APNs Push Notification Channels, Badges & Background Tap Routing exists for one cle... • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply FCM / APNs Push Notification Channels, Badges & Background Tap Routing in real React Native systems — specifically: Re-engage users reliably acros...

What Topic Is Next → Next Topic 13.4: Expo Config Plugins & Custom Native Android/iOS Manifest Tuning

13.4 Expo Config Plugins & Custom Native Android/iOS Manifest Tuning [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Custom Native Android/iOS Manifest Tuning • Integration with Graceful Runtime Permission Flows (Camera, Location, Notifications) • Live Debugging, Telemetry & Technical Interview Trade • What Is Expo Config Plugins & Custom Native Android/iOS Manifest Tuning & Why Industry Teams Use It • In React Native Training, Expo Config Plugins & Custom Native Android/iOS Manifest Tuning exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Expo Config Plugins & Custom Native Android/iOS Manifest Tuning in real React Native systems — specifically: Customize native Gradle/Podfile capab...

What Topic Is Next → Module 13 Capstone (Media Capture, Push Alert & Native Payment Integration) → Next Topic 14.1: Component Specs with Jest & React Native Testing Library

Module 13 Hands-On Capstone Project (100 Marks): Media Capture, Push Alert & Native Payment Integration — Design, implement, test, and document 'Media Capture, Push Alert & Native Payment Integration' applying every production concept from Module 13 (Native Permissions, Camera/Media Uploads, Pus...

MODULE 14 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

RNTL + Maestro E2E Testing, Hermes Profiling, Sentry & EAS OTA Releases

Production-grade React Native Training engineering module covering RNTL + Maestro E2E Testing, Hermes Profiling, Sentry & EAS OTA Releases with hands-on labs and capstone build.

Objectives: ✓ Component Specs with Jest & React Native Testing Library — Purpose: Test mobile screens via accessibility labels and user events cleanly. • ✓ Black-Box Mobile E2E Flows with Maestro / Detox — Purpose: Verify login, OTP, and cart checkout automatically on iOS Simulator & Android Emulator. • ✓ Hermes Bytecode Startup Profiling, Bundle Size & Sentry Crash Sourcemaps — Purpose: Slash Time-to-Interactive (TTI) and debug native + JS crashes from one console. • ✓ EAS Build Profiles, Store Submissions & Safe Over-the-Air (OTA) Updates — Purpose: Ship critical JS bugfixes to users in minutes while respecting native runtime compatibility.

14.1 Component Specs with Jest & React Native Testing Library [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of React Native Testing Library • Integration with Black • Live Debugging, Telemetry & Technical Interview Trade • What Is Component Specs with Jest & React Native Testing Library & Why Industry Teams Use It • In React Native Training, Component Specs with Jest & React Native Testing Library exists for one clear real • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Component Specs with Jest & React Native Testing Library in real React Native systems — specifically: Test mobile screens via accessibility labels...

What Topic Is Next → Next Topic 14.2: Black-Box Mobile E2E Flows with Maestro / Detox

14.2 Black-Box Mobile E2E Flows with Maestro / Detox [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Hermes Bytecode Startup Profiling, Bundle Size & Sentry Crash Sourcemaps • Live Debugging, Telemetry & Technical Interview Trade • What Is Black • In React Native Training, Black • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Black-Box Mobile E2E Flows with Maestro / Detox in real React Native systems — specifically: Verify login, OTP, and cart checkout automatically on...

What Topic Is Next → Next Topic 14.3: Hermes Bytecode Startup Profiling, Bundle Size & Sentry Crash Sourcemaps

14.3 Hermes Bytecode Startup Profiling, Bundle Size & Sentry Crash Sourcemaps [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Bundle Size • Integration with EAS Build Profiles, Store Submissions & Safe Over • Live Debugging, Telemetry & Technical Interview Trade • What Is Hermes Bytecode Startup Profiling, Bundle Size & Sentry Crash Sourcemaps & Why Industry Teams Use It • In React Native Training, Hermes Bytecode Startup Profiling, Bundle Size & Sentry Crash Sourcemaps exists for one c... • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply Hermes Bytecode Startup Profiling, Bundle Size & Sentry Crash Sourcemaps in real React Native systems — specifically: Slash Time-to-Interactive (T...

What Topic Is Next → Next Topic 14.4: EAS Build Profiles, Store Submissions & Safe Over-the-Air (OTA) Updates

14.4 EAS Build Profiles, Store Submissions & Safe Over-the-Air (OTA) Updates [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Store Submissions • Integration with Component Specs with Jest & React Native Testing Library • Live Debugging, Telemetry & Technical Interview Trade • What Is EAS Build Profiles, Store Submissions & Safe Over • In React Native Training, EAS Build Profiles, Store Submissions & Safe Over • Even if you come from a non • Runnable Code Implementation (TSX)

Purpose: To understand and apply EAS Build Profiles, Store Submissions & Safe Over-the-Air (OTA) Updates in real React Native systems — specifically: Ship critical JS bugfixes to...

What Topic Is Next → Module 14 Capstone (Production Hermes Bundle with Maestro E2E & Instant OTA) → Next Topic 15.1: React Native New Architecture: JSI, Fabric Synchronous Layout & TurboModules

Module 14 Hands-On Capstone Project (100 Marks): Production Hermes Bundle with Maestro E2E & Instant OTA — Design, implement, test, and document 'Production Hermes Bundle with Maestro E2E & Instant OTA' applying every production concept from Module 14 (RNTL + Maestro E2E Testing, Hermes Profiling...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

React Native New Architecture (JSI/Fabric), Reanimated & Offline Sync

Self-explanatory list-and-bullet production mastery module covering React Native New Architecture (JSI/Fabric), Reanimated & Offline Sync in React Native Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ React Native New Architecture: JSI, Fabric Synchronous Layout & TurboModules — Purpose: Understand how direct C++ JSI bindings replace legacy JSON bridge bottlenecks for native speed. • ✓ High-Speed Lists (FlashList), UI-Thread Gestures (Reanimated) & Keyboard Handling — Purpose: Build silky-smooth mobile feeds, bottom sheets, and chat screens on budget Android & iOS devices. • ✓ Offline Queueing, MMKV/SecureStore Token Safety & Universal Deep Linking — Purpose: Persist mutations across offline periods and protect sensitive credentials in Keychain/Keystore. • ✓ Build, verify, and document '60fps Gesture-Driven Offline-First React Native App with MMKV & FlashList' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 React Native New Architecture: JSI, Fabric Synchronous Layout & TurboModules [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for React Native New Architecture • Performance Optimization, Complexity & Latency Impact of React Native New Architecture • Live Debugging, Stack • Runnable Code Implementation (TSX) • Hands • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for React Native New Architecture • Production Hardening & Security: Expo SecureStore / Keychain biometrics, deep-link validation & SSL pinning

Purpose: To understand and apply React Native New Architecture: JSI, Fabric Synchronous Layout & TurboModules in real React Native systems — specifically: Understand how direct C+...

What Topic Is Next → Next Topic 15.2: High-Speed Lists (FlashList), UI-Thread Gestures (Reanimated) & Keyboard Handling

15.2 High-Speed Lists (FlashList), UI-Thread Gestures (Reanimated) & Keyboard Handling [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for High • Performance Optimization, Complexity & Latency Impact of High • Live Debugging, Stack • Runnable Code Implementation (TSX) • Hands • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for High-Speed Lists (FlashList) • Production Hardening & Security: Expo SecureStore / Keychain biometrics, deep-link validation & SSL pinning

Purpose: To understand and apply High-Speed Lists (FlashList), UI-Thread Gestures (Reanimated) & Keyboard Handling in real React Native systems — specifically: Build silky-smooth...

What Topic Is Next → Next Topic 15.3: Offline Queueing, MMKV/SecureStore Token Safety & Universal Deep Linking

15.3 Offline Queueing, MMKV/SecureStore Token Safety & Universal Deep Linking [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Offline Queueing, MMKV/SecureStore Token Safety & Universal Deep Linking • Performance Optimization, Complexity & Latency Impact of Offline Queueing, MMKV/SecureStore Token Safety & Universa... • Live Debugging, Stack • Runnable Code Implementation (TSX) • Hands • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for Offline Queueing • Production Hardening & Security: Expo SecureStore / Keychain biometrics, deep-link validation & SSL pinning

Purpose: To understand and apply Offline Queueing, MMKV/SecureStore Token Safety & Universal Deep Linking in real React Native systems — specifically: Persist mutations across off...

What Topic Is Next → Module 15 Capstone (60fps Gesture-Driven Offline-First React Native App with MMKV & FlashList) → Next Topic 16.1: Hermes Bytecode Startup Tuning, Image Caching & Eliminating JS-Thread Frame Drops

Module 15 Hands-On Capstone Project (100 Marks): 60fps Gesture-Driven Offline-First React Native App with MMKV & FlashList — Architect, implement, test, and document '60fps Gesture-Driven Offline-First React Native App with MMKV & FlashList' applying every bullet-wise engineering concept from Module 15 (React Nati...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

React Native Performance Profiling, E2E Testing & EAS App Store Delivery

Self-explanatory list-and-bullet production mastery module covering React Native Performance Profiling, E2E Testing & EAS App Store Delivery in React Native Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Hermes Bytecode Startup Tuning, Image Caching & Eliminating JS-Thread Frame Drops — Purpose: Cut cold TTI startup times and keep navigation transitions responsive under heavy state updates. • ✓ Component Testing with React Native Testing Library & Automated E2E Mobile Flows — Purpose: Verify login, checkout, and offline recovery journeys automatically before store submission. • ✓ App Store / Play Store Signing, OTA Update Rules & Mobile System Design Interviews — Purpose: Navigate native permissions, push notifications, and crash triage in React Native interviews. • ✓ Build, verify, and document 'Hermes Bundle Audit, Jest + Maestro/Detox Test Flow & Signed Release Pipeline' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Hermes Bytecode Startup Tuning, Image Caching & Eliminating JS-Thread Frame Drops [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Hermes Bytecode Startup Tuning, Image Caching & Eliminating JS • Performance Optimization, Complexity & Latency Impact of Hermes Bytecode Startup Tuning, Image Caching & Eliminatin... • Live Debugging, Stack • Runnable Code Implementation (TSX) • Hands • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for Hermes Bytecode Startup Tuning • Production Hardening & Security: Expo SecureStore / Keychain biometrics, deep-link validation & SSL pinning

Purpose: To understand and apply Hermes Bytecode Startup Tuning, Image Caching & Eliminating JS-Thread Frame Drops in real React Native systems — specifically: Cut cold TTI startu...

What Topic Is Next → Next Topic 16.2: Component Testing with React Native Testing Library & Automated E2E Mobile Flows

16.2 Component Testing with React Native Testing Library & Automated E2E Mobile Flows [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Component Testing with React Native Testing Library & Automated E2E Mobile... • Performance Optimization, Complexity & Latency Impact of Component Testing with React Native Testing Library & Auto... • Live Debugging, Stack • Runnable Code Implementation (TSX) • Hands • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for Component Testing with React Native Testing Library • Production Hardening & Security: Expo SecureStore / Keychain biometrics, deep-link validation & SSL pinning

Purpose: To understand and apply Component Testing with React Native Testing Library & Automated E2E Mobile Flows in real React Native systems — specifically: Verify login, checko...

What Topic Is Next → Next Topic 16.3: App Store / Play Store Signing, OTA Update Rules & Mobile System Design Interviews

16.3 App Store / Play Store Signing, OTA Update Rules & Mobile System Design Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for App Store / Play Store Signing, OTA Update Rules & Mobile System Design In... • Performance Optimization, Complexity & Latency Impact of App Store / Play Store Signing, OTA Update Rules & Mobile.... • Live Debugging, Stack • Runnable Code Implementation (TSX) • Hands • Idiomatic RNAT Implementation: TypeScript typed navigation, FlashList virtualization, Reanimated UI-thread &... • React Native New Architecture (JSI, Fabric Renderer, TurboModules) & Hermes Execution & Memory Lifecycle for App Store / Play Store Signing • Production Hardening & Security: Expo SecureStore / Keychain biometrics, deep-link validation & SSL pinning

Purpose: To understand and apply App Store / Play Store Signing, OTA Update Rules & Mobile System Design Interviews in real React Native systems — specifically: Navigate native pe...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: RouteMate Real-Time Rider Delivery, Live Map & Order Wallet Mobile App

Module 16 Hands-On Capstone Project (100 Marks): Hermes Bundle Audit, Jest + Maestro/Detox Test Flow & Signed Release Pipeline — Architect, implement, test, and document 'Hermes Bundle Audit, Jest + Maestro/Detox Test Flow & Signed Release Pipeline' applying every bullet-wise engineering concept from Module 16 (React...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: RouteMate Real-Time Rider Delivery, Live Map & Order Wallet Mobile App • **Confirmed Role:** React Native & TypeScript Mobile Engineering Intern

Domain & Production Infra: Last-Mile E-Commerce & Hyperlocal Mobility Client (React Native + Expo/CLI + TypeScript + Zustand/TanStack Query + Reanimated + OTA Releases)

- Developed cross-platform iOS & Android screens in React Native + TypeScript with React Navigation stacks/tabs.
- Optimized FlatList/FlashList rendering, gesture interactions, and offline request queues with TanStack Query.
- Integrated native device permissions, secure token storage, push alerts, and biometric session locks.
- Shipped production mobile builds with crash telemetry and automated component test verification.

Target Job Roles: React Native Mobile Developer, Cross-Platform iOS/Android Engineer, React & React Native Product Engineer, Mobile UI/UX Engineer

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹5.0 LPA – ₹9.5 LPA **Mid-Level React Native Engineer (1–3 Yrs):** ₹10.0 LPA – ₹20.5 LPA **Unified Web + Mobile Staff React Architect (3–6+ Yrs):** ₹21.0 LPA – ₹40.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 98765 43210 • Web: https://mockattempt.com/courses/react-native-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.