React Native

End-to-End Mobile Development

Mahima Kaushik

2-Hour Comprehensive Session

December 18, 2025

Session Agenda

What is React Native?

Definition:

- Framework for building mobile apps
- Uses JavaScript and React
- Created by Facebook (Meta) in 2015
- Write once, run on iOS & Android
- Real native apps, not hybrid

Key Benefits:

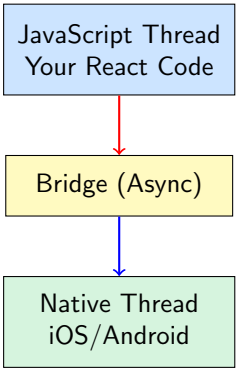
- Cross-platform development
- Reuse web development skills
- Hot reload for fast iteration
- Large community support
- Native performance

Analogy

React Native is a translator - you speak JavaScript, it speaks Swift/Kotlin to the phone

How React Native Works

The Bridge Architecture



Key Concepts

- **Asynchronous:** Bridge is non-blocking
- **Batching:** Updates are batched for performance

How It Works - Step by Step

1. **You write JSX:**

```
<View style={styles.container}>
  <Text>Hello!</Text>
</View>
```

2. **React Native translates:**
   - View → iOS: UIView / Android: ViewGroup
   - Text → iOS: UILabel / Android: TextView
3. **Bridge sends instructions** to native side
4. **Native code renders** actual components
5. **User interaction** → Event → Bridge → JavaScript

Real-World Success Stories

Apps Built with RN:

- Instagram (99% shared code)
- Discord (60 FPS scrolling)
- Shopify (Complex features)
- Uber Eats (Real-time tracking)
- Facebook
- SoundCloud

When to Use RN:

- Cross-platform apps
- Rapid prototyping
- JavaScript team
- Frequent updates

When NOT to use:

- Heavy 3D graphics/games
- Maximum performance critical

Prerequisites & Installation

Required Software:

- Node.js (v16+)
- npm/npx
- Code editor (VS Code)
- Android Studio (for emulator)

Verify Installation:

```
node --version
npm --version
npx --version
```

Development Approaches

- **Expo** (Recommended): Managed workflow, quick setup
- **React Native CLI**: Full control, more complex

npm vs npx

npm (Package Manager):

```
npm install package-name
npm install -g package-name
npm run start
```

npx (Package Executor):

```
npx create-expo-app MyApp
npx react-native init App
```

Why npx?

- No global installation needed
- Always uses latest version
- Saves disk space

Component Mapping: Web vs Native

React (Web)	React Native
div	View
span, p	Text
img	Image
input	TextInput
button	Button / TouchableOpacity
ul, li	FlatList

Key Difference

No HTML in React Native! Everything is a React component.

React vs React Native

Aspect	React (Web)	React Native
Platform	Browsers	iOS & Android
Elements	HTML tags	Native components
Styling	CSS files	StyleSheet objects
Layout	CSS Box Model	Flexbox (default)
Navigation	React Router	React Navigation

Similarities:

- Same React principles (components, props, state, hooks)
- Same lifecycle methods
- JSX syntax

Styling in React Native

No CSS Files!

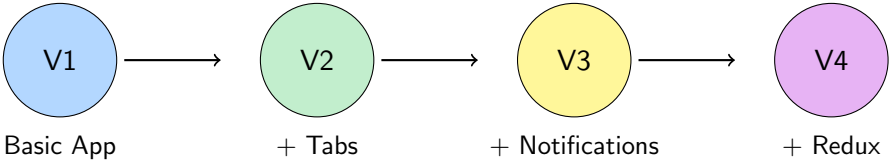
- JavaScript objects
- StyleSheet.create()
- Flexbox by default
- No pixels

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center'
  }
});
```

Flexbox Properties

flexDirection, justifyContent, alignItems, flex, padding, margin

The Journey - One App, Four Versions



Learning Path:

1. Start simple → Build foundation
2. Add navigation → Multiple screens
3. Add complexity → Real features
4. Add state management → Scale up

VERSION 1: Basic App

What We'll Build:

- Welcome screen with counter
- Buttons to increment/decrement
- Professional styling

Concepts Covered:

- Components: View, Text, TouchableOpacity
- State: useState hook
- Events: onPress handlers
- Styling: StyleSheet.create, Flexbox

```
npx create-expo-app MyApp
cd MyApp
npx expo start --android
```

VERSION 1: Key Code

```
import { useState } from 'react';
import { View, Text, TouchableOpacity } from 'react-native';

export default function App() {
  const [count, setCount] = useState(0);

  return (
    <View style={styles.container}>
      <Text>{count}</Text>
      <TouchableOpacity onPress={() => setCount(count + 1)}>
        <Text>+</Text>
      </TouchableOpacity>
    </View>
  );
}
```

VERSION 1: Key Learnings

Components

- View is like div - a container
- Text for all text (required!)
- TouchableOpacity for touchable elements

State

- useState() creates state variable
- State updates trigger re-render
- Always use setter function

Styling

- StyleSheet.create() for optimization
- Camel case properties
- No units on numbers

VERSION 2: Adding Tab Navigation

Evolution:

- Keep counter from V1
- Add multiple screens
- Add bottom tab navigation
- Add icons for tabs

Install Dependencies:

```
npm install @react-navigation/native @react-navigation/bottom-tabs
npx expo install react-native-screens react-native-safe-area-context
```

Concepts Added:

- React Navigation setup
- NavigationContainer
- createBottomTabNavigator
- Screen components

VERSION 2: Key Learnings

Navigation

- NavigationContainer wraps entire app
- Each screen is a separate component
- Tab navigator manages screen switching
- Icons change based on focused state

Screen Organization

- Separate function for each screen
- Each screen has its own state
- Shared styles across screens

VERSION 3: Adding Notifications

Evolution:

- Add Notifications tab
- List of notifications with FlatList
- Mark as read/unread
- Delete notifications
- Show unread count

New Components:

- FlatList for efficient lists
- Alert API for confirmations
- Complex state (array of objects)

Concepts Added:

- Lists and keys
- Array methods: map, filter
- Immutable state updates

VERSION 3: Key Learnings

FlatList

- Use for long lists (not .map())
- renderItem function returns component
- keyExtractor for unique keys (required!)
- Virtualized - only renders visible items

Immutable State Updates

- Don't mutate arrays directly
- Use spread operator: [...array, newItem]
- Use map() and filter() for updates

VERSION 4: Props & Redux

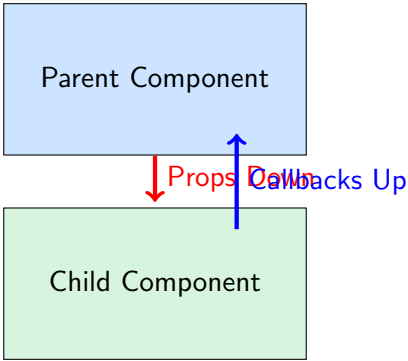
Final Evolution:

- Extract components with props
- Share state with Redux
- Connect all screens to store
- Global notification count

Concepts Added:

- Props: passing data to components
- Redux store: centralized state
- useSelector: read from store
- useDispatch: send actions to store

Props: Data Flow



Remember

- Props flow DOWN (parent → child)
- Events flow UP (child → parent via callbacks)
- Props are READ-ONLY in child

Redux: When to Use

Use Redux when:

- State shared across many components
- Complex state logic
- Large applications

Don't use if:

- Simple apps
- State in 1-2 components
- Can use Context API

Common Mistakes to Avoid

1. Mutating State Directly

Wrong: items.push(newItem); setItems(items);
Correct: setItems([...items, newItem]);

2. Missing Keys in Lists

Wrong: items.map(item =¿ ¡Item item=item /¿)
Correct: items.map(item =¿ ¡Item key=item.id item=item /¿)

3. Inline Styling

Wrong: style= flex: 1 (creates new object every render)
Correct: style=styles.container

Performance Tips

1. **Use FlatList** for long lists
2. **Memoize** expensive components with React.memo
3. **Optimize images** - resize before loading
4. **Use StyleSheet.create()** - not inline styles
5. **Profile** with React DevTools

Swift vs JavaScript: Overview

Feature	JavaScript	Swift
Type System	Dynamic	Static, strongly typed
Compilation	Interpreted (JIT)	Compiled (AOT)
Platform	Cross-platform	Apple only
Performance	Good	Excellent
Learning Curve	Easier	Steeper

Key Difference:

- JavaScript: Flexible, dynamic, forgiving
- Swift: Strict, safe, explicit

Why Learn Swift?

Swift is for:

- Native iOS development
- Maximum performance
- Full iOS platform access
- Official Apple language

When you need Swift:

- iOS-specific features (Face ID, Apple Pay)
- Deep system integration
- Writing native modules for React Native
- Performance-critical code

Good News

If you know JavaScript, Swift isn't that different!

What We Covered Today

1. Introduction to React Native
2. How it works (Bridge architecture)
3. Core Concepts (Components, Props, State)
4. Built Progressive App (V1 $\rightarrow$ V2 $\rightarrow$ V3 $\rightarrow$ V4)
5. Best Practices
6. Swift Comparison

You've learned:

- Built 4 app versions
- Learned navigation
- Mastered state management
- Understood props and Redux

Next Steps

Continue Learning

- Build a todo app from scratch
- Add features to progressive app
- Explore TypeScript
- Deploy to app stores

Resources

- reactnative.dev
- docs.expo.dev
- reactnavigation.org

Join Community

- React Native Discord
- Stack Overflow

Project Ideas

Beginner:

- Todo List, Calculator, Weather app

Intermediate:

- Chat app, E-commerce, Social feed

Advanced:

- Real-time collaboration, Fitness tracker, Music player

Thank You!

Questions?

Contact:
Email: mahima.kaushik@example.com

Keep Building!