ON MOBILE
WITH REACT NATIVE

Fabrizio Rizzi
Senior Front-end Developer

SORINT lab

PERCHÈ REACT NATIVE?

Come sviluppare app mobile

- Applicazioni Native
  - iOS (Xcode —> Objective-C, Swift, …)
  - Android (Android Studio —> Java, Kotlin, …)
- Flutter (Dart)
- Xamarin (C#)
- NativeScript (Javascript)
- React Native (Javascript)
- Ionic (Javascript)
- Cordova (Javascript)
- …

PERCHÈ REACT NATIVE?

Punti di forza del framework

- Consente di sviluppare applicazioni mobile Android e iOS con la stessa codebase
- Basato su Javascript (React), consente di creare applicazioni utilizzando le tecnologie già note agli sviluppatori front-end
- Utilizza componenti Javascript costruiti su componenti iOS e Android dando un feeling completamente nativo alla vostra applicazione
- Hot / Live reloading per visualizzare le modifiche in tempo reale senza dover buildare l'applicazione
- Ampia community, utilizzato da molte grandi aziende, esistono molti plugin e librerie utili

REACT FUNDAMENTALS

Introduzione alla libreria React

Come suggerisce il nome del framework, React Native è basato sulla famosa libreria utilizzata per lo sviluppo di applicazioni web React (https://it.reactjs.org/ - https://react.dev/)

Principali caratteristiche:

- Componenti
- JSX
- Props

REACT FUNDAMENTALS

Componenti

“I Componenti ti permettono di suddividere la UI (User Interface, o interfaccia utente) in parti indipendenti, riutilizzabili e di pensare ad ognuna di esse in modo isolato.

Concettualmente, i componenti sono come funzioni JavaScript: accettano in input dati arbitrari (sotto il nome di “props”) e ritornano elementi React che descrivono cosa dovrebbe apparire sullo schermo.”

https://react.dev/reference/react/Component

REACT FUNDAMENTALS

JSX

JSX è una sintassi JavaScript che ti consente di scrivere un markup simile all'HTML ma all'interno del tuo file Javascript

- estensione .jsx (.tsx per Typescript)
- deve essere contenuto in un singolo tag HTML
- per utilizzare Javascript all'interno di JSX si utilizzano le parentesi {}
- class —> className

https://react.dev/learn/writing-markup-with-jsx

REACT FUNDAMENTALS

Props

Sono dati ricevuti in input da un componente, consentono di mettere in comunicazione un componente “padre” con un componente “figlio”

Le Props sono in Sola Lettura (Tutti i componenti React devono comportarsi come funzioni pure rispetto alle proprie props)

https://react.dev/learn/passing-props-to-a-component

REACT FUNDAMENTALS

Eventi

“La gestione degli eventi negli elementi React è molto simile alla gestione degli eventi negli elementi DOM. Vi sono alcune differenze sintattiche:

- *Gli eventi React vengono dichiarati utilizzando camelCase, anziché in minuscolo*
- *In JSX, il gestore di eventi (event handler) viene passato come funzione, piuttosto che stringa”*

https://react.dev/learn/responding-to-events

REACT FUNDAMENTALS

Stato

Per stato si intendono i dati che dobbiamo gestire nel tempo all'interno della nostra applicazione.
Possiamo pensare allo stato di un componente come la sua personale memoria ed è utile quando vogliamo gestire i cambiamenti dei dati all'interno dello stato dovuti a eventi che accadono nel tempo o in seguito all'interazione degli utenti

Sebbene lo stato sembri un concetto semplice, ci sono due problemi con la sua gestione quando si utilizza solo normale JavaScript:

- Non è ovvio cosa sia lo stato o dove viva
- La lettura e l'aggiornamento dello stato sono un processo innaturale e spesso ripetitivo quando si utilizzano API browser native come document

https://react.dev/learn/updating-objects-in-state

REACT FUNDAMENTALS

Hooks

“Gli Hooks sono stati aggiunti in React 16.8. Ti permettono di utilizzare state ed altre funzioni di React senza dover scrivere una classe”

Possono essere utilizzati solo all’interno di functional components di React e i principali che vedremo sono:

- useState
- useEffect

https://react.dev/reference/react

REACT FUNDAMENTALS

useState

```
const [state, setState] = useState(initialState);
```

https://react.dev/reference/react/useState

REACT FUNDAMENTALS

useEffect

```
useEffect(() => {
 …code…
  return () => {
    …cleanup function…
  };
}, [dependencies]);
```

https://react.dev/reference/react/useEffect

REACT FUNDAMENTALS

Bonus tips

Conditional rendering

I componenti spesso devono visualizzare cose diverse in base a determinate condizioni. In React, è possibile renderizzare JSX in maniera condizionale utilizzando sintassi Javascript come *if, &&* e l'operatore ternario.

Iterate over elements

array.map(element => {element})

READY TO GO!!!

Homeworks: https://react.dev/learn/tutorial-tic-tac-toe

Siamo quindi pronti per iniziare la nostra avventura con React Native!

REACT NATIVE COMPONENTS

Core Components

Nello sviluppo Android, è possibile creare views in Kotlin or Java. Con iOS potete utilizzarle Swift or Objective-C. Con React Native, potete realizzare queste views con JavaScript utilizzando i componenti React. A runtime, React Native crea le corrispondenti views per Android e iOS di questi componenti.
React Native vi fornisce una varietà di componenti nativi pronti da usare per costruire le vostre applicazioni. Chiamiamo questi componenti Core Components.
Per approfondire come React Native trasforma il vostro codice in codice nativo
https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8

Qui trovate la lista completa dei Core Components
https://reactnative.dev/docs/components-and-apis

REACT NATIVE STYLE

Style

Con React Native, è possibile applicare lo stile alla vostra applicazione utilizzando JavaScript. Tutti i core components accettano una prop chiamata style. I nomi ed i valori solitamente corrispondono ai CSS utilizzati nelle applicazioni web, tranne per il fatto che i nomi sono scritti utilizzando il camel case. Ex backgroundColor invece che background-color.

Quando i componenti crescono in complessità, spesso è utile utilizzare StyleSheet.create come nell'esempio riportato alla seguente pagina:

https://reactnative.dev/docs/style

REACT NATIVE COMPONENTS

<View>

Il componente più importante per costruire le vostre interfacce, View è un contenitore che supporta la gestione del layout con flexbox, style, qualche controllo touch e controlli per l'accessibilità.
View può essere nestato all'interno di altri View e può avere da 0 a molti figli dello stesso tipo .

https://reactnative.dev/docs/view

REACT NATIVE COMPONENTS

<Text>

Componente per visualizzare il testo all’interno della vostra applicazione.
Supporta il nesting, style, e la gestione dei controlli touch.

https://reactnative.dev/docs/text

REACT NATIVE COMPONENTS

<TextInput>

Componente per l'inserimento del testo equivalente al tag input html. Le Props forniscono la possibilità di configurare diverse funzionalità come auto-correction, auto-capitalization, placeholder text, e keyboard types, per mostrare diversi tipi di tastiera.

https://reactnative.dev/docs/textinput

REACT NATIVE COMPONENTS

<Button>

Semplice bottone che supporta poche customizzazioni.
Se non ha un aspetto adatto alla vostra app, potete creare i vostri bottoni tramite l'utilizzo del componente Pressable.

https://reactnative.dev/docs/button

REACT NATIVE COMPONENTS

<Pressable>

Pressable è un Core Component wrapper che può rilevare vari momenti dell'interazione press su ciascuno dei suoi "figli".

https://reactnative.dev/docs/pressable

REACT NATIVE COMPONENTS

<ScrollView>

Componente che permette lo scroll della pagina quando il contenuto fuoriesce dalla pagina.

https://reactnative.dev/docs/scrollview

REACT NATIVE COMPONENTS

<FlatList>

Una performante interfaccia per visualizzare liste di elementi. Supporta molte funzionalità utili quali il pull to refresh, la modalità orizzontale, lo scroll loading e tante altre…

https://reactnative.dev/docs/flatlist

REACT NAVIGATION

Navigazione delle schermate

Difficilmente un’applicazione è composta da una singola schermata. A questo scopo introduciamo la libreria React Navigation che ci consentirà di creare una navigazione tra le diverse schermate della nostra app

https://reactnative.dev/docs/navigation

https://reactnavigation.org/

REACT NAVIGATION

Muoversi tra le schermate

Ogni componente all'interno di uno stack navigator può accedere alle props "navigation" e "route".

Per navigare verso una nuova schermata:
navigation.navigate('Details')

Per tornare alla schermata precedente:
navigation.goBack()

https://reactnavigation.org/docs/navigating

REACT NAVIGATION

Passare parametri tra le schermate

Per passare dei dati tra le schermate è necessario:

- passare un oggetto con i dati come secondo argomento del metodo navigate: *navigation.navigate('RouteName', { /* params go here */ })*

- Leggere i dati dalla schermata di contenuti nell'oggetto route.params

https://reactnavigation.org/docs/params

HACKERSGEN
GENERATION: HACKERS

REACT NAVIGATION

Configurare la header bar

Tramite delle apposite opzioni da passare allo Stack.Screen è possibile personalizzare la header bar cambiando il titolo della schermata, modificandone lo stile oppure aggiungendo dei pulsanti.
Nel caso in cui si desideri modificare la header bar per tutte le schermate di una navigazione è possibile utilizzare la proprietà screenOptions dello Stack.Navigator

https://reactnavigation.org/docs/headers
https://reactnavigation.org/docs/header-buttons

SORINT.lab

REACT NAVIGATION

Tab Navigation

Probabilmente lo stile più utilizzato per la navigazione delle schermate di un'applicazione mobile è la navigazione a tab.

Per creare una navigazione a tab dobbiamo installare una libreria aggiuntiva:
https://reactnavigation.org/docs/tab-based-navigation

Inoltre, per migliorare lo stile della nostra tab navigation, aggiungiamo delle icone:
https://github.com/oblador/react-native-vector-icons

THANKS!

@FabrizioRizzi