Modulo 3 — Gestione della navigazione

React Navigation • Stack • Tabs • Drawer • Parametri • App multi-pagina

Obiettivi del modulo

Cosa saprai costruire

- Aggiungere navigazione completa ad un’app React Native.
- Creare Stack Navigator per passare tra schermate.
- Creare Tab Navigator per sezioni principali.
- Creare Drawer Navigator per un menu laterale.
- Passare e tipizzare parametri tra schermate.
- Comporre navigator annidati in un’app multi-pagina.

Perché serve la navigazione

Il problema che risolve

- In un’app reale hai più schermate (Home, Dettagli, Profilo…).
- Serve un modo standard per passare da una schermata all’altra.
- Serve anche: history (indietro), animazioni, header, deep link.
- React Navigation è una soluzione completa e modulare.

React Navigation

Panoramica

- Libreria modulare: stack, tab, drawer, ecc.
- Gestisce stato di navigazione e history.
- Supporta parametri, header, gesture e deep linking.
- Navigator annidati = pattern reale per app complesse.

Concetti base

Parole chiave

- NavigationContainer: radice che contiene lo stato di navigazione.
- Navigator: contenitore di schermate (Stack/Tab/Drawer…).
- Screen: schermata registrata nel navigator.
- navigation: oggetto per navigare (navigate, goBack…).
- route: info della schermata (name, params).

Installazione

Expo (managed)

Passi

- Installa i pacchetti principali.
- Aggiungi dipendenze richieste (screens, safe-area, gesture).
- Scegli navigator specifici (stack/tabs/drawer).
- Avvolgi l’app con NavigationContainer.

Comandi (npm)

```
npm install
@react-navigation/native

npx expo install react-native-screens
react-native-safe-area-context

npx expo install
react-native-gesture-handler react-
native-reanimated
```

Installazione

React Native CLI (bare)

Nota pratica

- Stessi pacchetti JS, ma setup nativo più frequente.
- Su iOS spesso serve CocoaPods (pod install).
- Su Android può servire sync/rebuild.
- Expo riduce molta complessità di setup.

Comandi (npm)

```
npm install
@react-navigation/native
npm install react-native-screens
react-native-safe-area-context
npm install react-native-gesture-
handler
react-native-reanimated

# iOS (se necessario)
cd ios && pod install && cd ..
```

Avvio minimo

NavigationContainer + Stack

Idea

- Crea uno Stack navigator.
- Registra schermate come Stack.Screen.
- Metti tutto dentro NavigationContainer.
- Ora puoi navigare con navigation.navigate().

App.tsx

```
import { NavigationContainer } from
"@react-navigation/native";
import { createNativeStackNavigator } from
"@react-navigation/native-stack";

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home"
        component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Expo Router vs React Navigation

Quando scegliere cosa

Confronto operativo

Expo Router: file-based, setup veloce, ottimo per app Expo e progetti moderni.

React Navigation “manuale”: controllo esplicito dei navigator, utile se vuoi una struttura totalmente custom.

Entrambi supportano Stack/Tabs/Drawer; differisce soprattutto il modo di “definire” le rotte.

Se lavori con Expo e vuoi velocità + convenzioni: Expo Router è spesso la scelta più semplice.

Stack Navigation

Come funziona

- Pila di schermate: l’ultima in cima è visibile.
- Avanti = push; indietro = pop.
- Include header e animazioni di transizione.
- È il pattern più usato per flussi ‘dettaglio’.

Drawer Navigation

Idea visiva (menu laterale)

Drawer
(menu laterale)
- Home
- Profilo
- Impostazioni
- About

Area contenuti
Schermata selezionata dal menu

Quando usarla: molte sezioni e menu laterale.

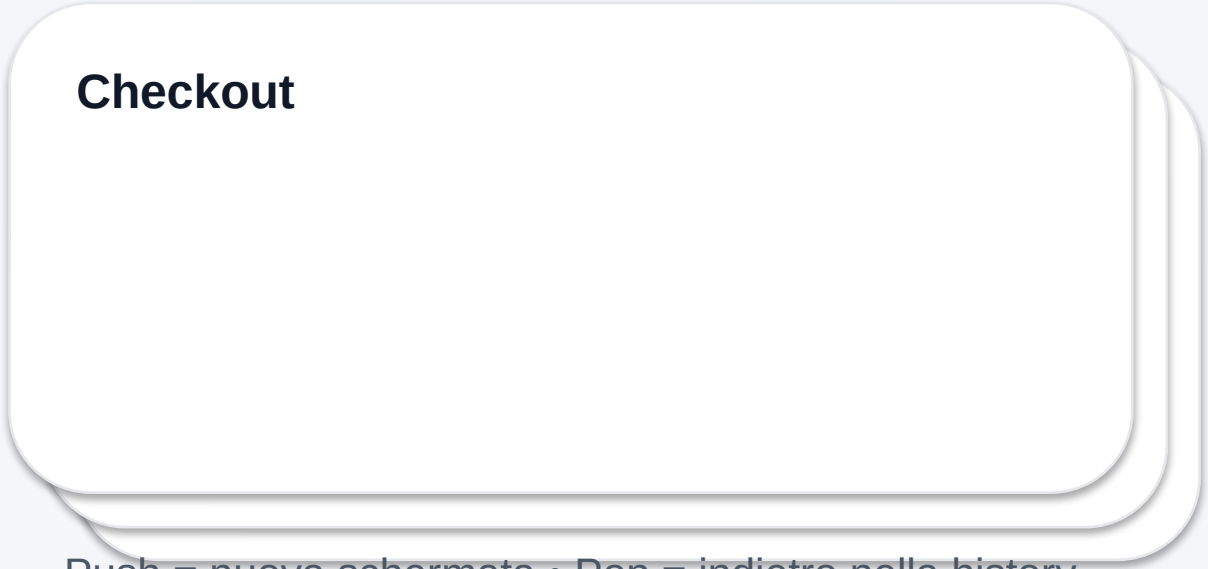
Tab Navigation

Idea visiva (sezioni principali)

Quando usarla: 2–5 sezioni principali sempre disponibili.

Stack Navigation

Idea visiva (history a pila)



Push = nuova schermata • Pop = indietro nella history

Quando usarla: flusso lista → dettaglio → step successivo.

Stack (setup)

createNativeStackNavigator

Punti chiave

- native-stack usa componenti nativi (performance).
- Configuri opzioni per screen (header, presentation).
- Ogni screen ha name + component.

RootStack.tsx

```
import { createNativeStackNavigator }
from
"@react-navigation/native-stack";

const Stack =
createNativeStackNavigator();

export function RootStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home"
      component={HomeScreen} />
      <Stack.Screen name="Details"
      component={DetailsScreen} />
    </Stack.Navigator>
  );
}
```

Navigare nello stack

navigate / push / goBack

Comandi principali

- navigate(name): va a una schermata (o la porta in cima).
- push(name): nuova istanza nello stack.
- replace(name): sostituisce la schermata corrente.
- goBack(): torna indietro.

Esempi

```
navigation.navigate("Details",
{ id: "p_123"
 });
navigation.push("Details", { id:
"p_999" });
navigation.replace("Home");
navigation.goBack();
```

Stack (opzioni)

Header e comportamento

Cosa puoi configurare

- title: titolo in header
- headerShown: mostra/nasconde header
- presentation: modal vs card
- gestureEnabled: gesture indietro

Options

```
<Stack.Screen
  name="Details"
  component={DetailsScreen}
  options={{
    title: "Dettagli",
    headerShown: true,
    presentation: "card",
    gestureEnabled: true,
  }}
/>
```

Azioni avanzate

Reset e flussi

- reset: riscrive la history (es. dopo login).
- popToTop: torna alla prima schermata dello stack.
- setParams: aggiorna params della schermata corrente.
- Sono utili per flussi autenticazione e wizard.

Reset

Esempio concettuale

Quando usarlo

- Dopo login vuoi impedire di tornare alla schermata login.
- Con reset imposti una nuova history con una nuova route iniziale.

reset()

```
navigation.reset({
  index: 0,
  routes: [{ name: "Home" }],
});
```

Parametri tra schermate

Concetto

- Apri ‘Dettagli’ passando un id.
- I parametri viaggiano in route.params.
- Con TypeScript conviene tipizzare (ParamList).
- Meno bug e più autocomplete.

Passare parametri

navigate(name, params)

Esempio

- Passi params nel navigate.
- La screen di arrivo legge route.params.
- Usa dati serializzabili (string, number, oggetti semplici).

Params

```
// Home
navigation.navigate("Details",
{ id: "p_123"
 });

// Details
function DetailsScreen({ route }) {
  const { id } = route.params;
  return <Text>Dettaglio:
{id}</Text>;
}
```

Tipizzare params

ParamList con TypeScript

Idea

- Mappa: screenName → tipo params.
- Usa NativeStackScreenProps per tipizzare.
- route.params non è mai any.

Typing

```
export type RootStackParamList = {
  Home: undefined;
  Details: { id: string };
};

type Props =
NativeStackScreenProps<RootStackPar
amList,
"Details">;

export function
DetailsScreen({ route }:
Props) {
  return
<Text>{route.params.id}</Text>;
}
```

Params opzionali

Caso frequente

- Alcune schermate supportano params opzionali.
- Tipizza con?: e gestisci fallback (default).
- Questo rende le screen più riutilizzabili.

Params opzionali

Esempio

Pattern

- Se id non c’è, mostra un placeholder o fai redirect.
- Evita crash per route.params undefined.

Optional params

```
type ParamList = {
  Details: { id?: string };
};

const id = route.params?.id ??
"default_id";
```

Tab Navigation

Quando usarla

- Sezioni principali sempre disponibili (Home, Cerca, Profilo).
- Tipico: bottom tabs (barra in basso).
- Ogni tab può contenere uno stack (pattern comune).
- Ottimo per app multi-sezione.

Tab (setup)

createBottomTabNavigator

Punti chiave

- Ogni Tab.Screen è una sezione.
- Configura label/icon.
- Spesso ogni tab contiene uno Stack.

MainTabs.tsx

```
import { createBottomTabNavigator }
from
"@react-navigation/bottom-tabs";

const Tab = createBottomTabNavigator();

export function MainTabs() {
  return (
    <Tab.Navigator>
      <Tab.Screen name="HomeTab"
      component={HomeStack} />
      <Tab.Screen name="SettingsTab"
      component={SettingsStack} />
    </Tab.Navigator>
  );
}
```

Tab (opzioni)

header, lazy, badge

Configurazioni tipiche

- headerShown: spesso false nel tab (header gestito dallo stack).
- lazy: carica tab quando serve (default).
- tabBarBadge: mostra un contatore (notifiche).

Options

```
<Tab.Screen
  name="Inbox"
  component={InboxStack}
  options={{ tabBarBadge: 3,
headerShown:
  false }}
/>
```

Navigator annidati

Pattern comune

Pattern frequente: Drawer contiene Tabs; ogni Tab contiene uno Stack. Menu laterale + sezioni + dettaglio.

Drawer Navigation

Quando usarla

- Menu laterale con molte sezioni.
- Utile quando le tabs diventano troppe.
- Può contenere screen o navigator annidati.
- Si apre con gesture o pulsante.

Drawer (setup)

createDrawerNavigator

Punti chiave

- Drawer.Screen = voce del menu.
- Ogni voce può aprire screen o navigator.
- Configura opzioni globali (edgeWidth, drawerType…).

AppDrawer.tsx

```
import { createDrawerNavigator } from
"@react-navigation/drawer";

const Drawer = createDrawerNavigator();

export function AppDrawer() {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Main"
      component={MainTabs} />
      <Drawer.Screen name="About"
      component={AboutScreen} />
    </Drawer.Navigator>
  );
}
```

Aprire il drawer

Header button

Idea

- Aggiungi pulsante in header (hamburger).
- Usa openDrawer() o toggleDrawer().
- useLayoutEffect per impostare header dinamico.

toggleDrawer()

```
import { useLayoutEffect } from
"react";

useLayoutEffect(() => {
  navigation.setOptions({
    headerLeft: () => (
      <Pressable onPress={() =>
      navigation.toggleDrawer()}>
        <Text>≡</Text>
      </Pressable>
    ),
  });
}, [navigation]);
```

Back button (Android)

Cosa sapere

- Su Android il tasto indietro è gestito dal sistema.
- React Navigation integra la history, ma in casi speciali puoi intercettare BackHandler.
- Esempi: modali, conferma uscita, wizard non completato.

BackHandler

Esempio (concetto)

Quando usarlo

- Solo quando serve davvero: non bloccare il back senza motivo.
- Chiedi conferma e poi decidi se lasciare andare indietro.

BackHandler

```
import { BackHandler } from "react-
native";
import { useFocusEffect } from
"@react-navigation/native";

useFocusEffect(() => {
  const sub =
BackHandler.addEventListener("
  hardwareBackPress", () => {
    // ritorna true per bloccare,
false per
    default
    return false;
  });
  return () => sub.remove();
});
```

Navigare dentro navigator annidati

Caso reale

- Aprire Details dentro uno stack che sta dentro un tab.
- Puoi specificare screen + params annidati.
- Utile per link, notifiche e azioni globali.
- Tipizzare aiuta a non sbagliare i nomi.

navigate annidato

screen + params

Esempio

- Vai al tab HomeTab.
- Dentro HomeTab apri Details.
- Passi i params a Details.

Nested

```
navigation.navigate("HomeTab", {
  screen: "Details",
  params: { id: "p_123" },
});
```

Hook utili per navigation

Cosa usare e quando

- useNavigation: ottenere navigation in componenti non-screen.
- useRoute: leggere route e params.
- useFocusEffect: eseguire codice quando una screen torna in focus.
- useIsFocused: sapere se una screen è attiva.

useFocusEffect

Eseguire codice al focus

Perché è utile

- In una tab app, una screen può essere montata ma non visibile.
- useEffect non basta: useFocusEffect parte quando entri/torni.
- Utile per refresh dati o reset stati temporanei.

Focus

```
import { useCallback } from
"react";
import { useFocusEffect } from
"@react-navigation/native";

useFocusEffect(
  useCallback(() => {
    // refresh dati
    return () => {
      // cleanup (opzionale)
    };
  }, [])
);
```

Deep Linking

Aprire una schermata da un link

- Deep link = URL che apre una schermata specifica.
- Utile per notifiche, email, link condivisi.
- Serve mapping URL → screen + params (linking config).
- React Navigation gestisce conversione URL ⇄ route.

Linking config

Schema base

Idea

- Definisci prefissi (es. myapp://).
- Mappa path a screen.
- React Navigation crea route.params dai segmenti.

Linking

```
const linking = {
  prefixes: ["myapp://"],
  config: {
    screens: {
      Home: "home",
      Details: "details/:id",
    },
  },
};

<NavigationContainer
linking={linking}>...</
NavigationContainer>
```

Debug e problemi tipici

Checklist

- “Couldn't find a navigation object” → manca container o sei fuori dal tree.
- Schermata non trovata → name non coincide.
- Params undefined → navigate senza params o typing errato.
- Header non cambia → setOptions nel posto giusto (useLayoutEffect).

Pausa

15 minuti

Riprendiamo tra 15 minuti

Progetto: app multi-pagina

Cosa costruiremo

- App con Drawer + Tabs + Stack.
- Tab Home con lista e Details (params).
- Tab Settings con schermate base.
- Drawer per sezioni extra (About).

Architettura proposta

Drawer → Tabs → Stack

Obiettivo: combinare Stack/Tabs/Drawer senza confusione e con tipizzazione dei parametri.

Step 1

Definire le screen

Suggerimento

- Crea componenti: Home, Details, Settings, About.
- UI minima: Text + Pressable.
- Aggiungi navigazione dopo aver registrato le screen.

Screens

```
export function HomeScreen() {
  return (
    <View>
      <Text>Home</Text>
    </View>
  );
}

export function DetailsScreen() {
  return <Text>Details</Text>;
}
```

Step 2

HomeStack: Home → Details

Punti chiave

- HomeStack contiene Home e Details.
- Details riceve params (id).
- Home naviga a Details con un id.

HomeStack

```
export type HomeStackParamList = {
  Home: undefined;
  Details: { id: string };
};

const Stack = createNativeStackNavigator<Hom
eStackParamList>();

export function HomeStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home"
      component={HomeScreen} />
      <Stack.Screen name="Details"
      component={DetailsScreen} />
    </Stack.Navigator>
  );
}
```

Step 3

Navigare con params

Esempio

- In Home: apri Details passando id.
- In Details: leggi route.params.id.
- Mostra UI diversa in base all’id.

Params

```
// Home
navigation.navigate("Details",
{ id: "p_123"
 });

// Details
const { id } = route.params;
return <Text>Dettaglio:
{id}</Text>;
```

Step 4

MainTabs: HomeStack + SettingsStack

Perché così

- Ogni tab ha il suo stack (dettagli indipendenti).
- Header: gestito nel livello stack.
- Tabs restano stabili mentre navighi in profondità.

Tabs

```
const Tab =
createBottomTabNavigator();

export function MainTabs() {
  return (
    <Tab.Navigator screenOptions={{
    headerShown: false }}>
      <Tab.Screen name="HomeTab"
      component={HomeStack} />
      <Tab.Screen
name="SettingsTab"
      component={SettingsStack} />
    </Tab.Navigator>
  );
}
```

Step 5

AppDrawer: MainTabs + About

Idea

- Drawer aggiunge un menu laterale per sezioni extra.
- MainTabs resta il cuore dell’app.
- About è una schermata semplice dal menu.

Drawer

```
const Drawer =
createDrawerNavigator();

export function AppDrawer() {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Main"
      component={MainTabs}
options={{ title:
       "App" }} />
      <Drawer.Screen name="About"
      component={AboutScreen} />
    </Drawer.Navigator>
  );
}
```

Step 6

Root App

Radice

- NavigationContainer contiene AppDrawer.
- Qui puoi aggiungere linking config.
- Qui puoi impostare tema e opzioni globali.

App.tsx

```
export default function App() {
  return (
    <NavigationContainer>
      <AppDrawer />
    </NavigationContainer>
  );
}
```

Extra (opzionale)

Cosa aggiungere se avanza tempo

- Schermata Modal (presentation: 'modal').
- Deep link verso Details.
- Reset stack dopo login fittizio.
- Aggiungere badge alle tab e contatori.

Consigli per mantenere ordine

Quando crescono le schermate

- Cartelle: navigators/, screens/, components/, types/.
- Tieni ParamList in types/navigation.ts.
- Evita name duplicati e cerca coerenza (HomeTab vs Home).
- Un navigator per responsabilità (non uno gigante).

Riepilogo

Cosa porti a casa

- Stack: flusso a schermate con history.
- Tabs: sezioni principali sempre disponibili.
- Drawer: menu laterale per molte sezioni.
- Params: dati tra schermate + tipizzazione.
- Annidamento: pattern reale per app multi-pagina.

Domande di ripasso

Domanda 1 di 5

1. Qual è lo scopo di NavigationContainer?

- ☐ Mostrare automaticamente una tab bar
- ☐ Contenere lo stato di navigazione e la radice del tree
- ☐ Sostituire StyleSheet
- ☐ Gestire solo le immagini dell’app

Domande di ripasso

Domanda 2 di 5

2. In uno Stack Navigator, cosa fa replace()?

- ☐ Apre il drawer
- ☐ Sostituisce la schermata corrente con un’altra
- ☐ Aggiunge sempre una nuova schermata in cima
- ☐ Chiude l’app

Domande di ripasso

Domanda 3 di 5

3. Dove trovi i parametri passati a una schermata?

- ☐ In navigation.state.params sempre
- ☐ In route.params
- ☐ In StyleSheet.params
- ☐ In props.children

Domande di ripasso

Domanda 4 di 5

4. Quando preferisci Tabs rispetto a Drawer?

- ☐ Quando hai 2–5 sezioni principali sempre disponibili
- ☐ Quando vuoi solo una schermata
- ☐ Quando non vuoi gestire history
- ☐ Quando vuoi evitare NavigationContainer

Domande di ripasso

Domanda 5 di 5

5. Perché tipizzare i ParamList in TypeScript?

- ☐ Per rendere l’app più lenta
- ☐ Per evitare errori su name e params e avere autocomplete
- ☐ Per sostituire il bundler Metro
- ☐ Per non usare più navigation.navigate