Mobile Applications

Lab 3

React Navigation: Stack, Tabs, parameters, headers, typed navigation (TS)

Mateusz Pawełkiewicz

Instruction version: October 2026

Expo SDK 58 · React Native 0.88 · React 19 · TypeScript 6

Example code checked on Expo SDK 57 and 58

Goal of the lab

Set up React Navigation in an Expo project (TypeScript): add stack navigation (Stack) and bottom tabs (Tabs), type the routes and parameters correctly, pass data between screens, use `useNavigation` / `useRoute`, configure headers and the theme, and protect screens from users who are not signed in.

Before the lab

- ☐ The project from the previous labs (the **blank-typescript** template, the `App.tsx` file), `npx expo start` works.
- ☐ The dependencies from Step 0 are installed.
- ☐ The files from Steps 1–4 are typed in / copied; the app starts and you can go: Sign in → Home → Profile / Settings.
- ☐ You have read the **task at the end of the instruction** and prepared the data file `src/data/courses.ts` (the template is in the task).

Step 0: Installing the dependencies

We install all packages with one command in the project folder:

Terminal

```
npx expo install @react-navigation/native @react-navigation/native-stack @react-navigation/bottom-tabs react-native-screens react-native-safe-area-context @expo/vector-icons expo-font
```

Package	Role
`@react-navigation/native`	The core: `NavigationContainer`, themes, the `useNavigation`, `useRoute`, `useIsFocused` hooks.
`@react-navigation/native-stack`	Stack navigation using the system's native screens (animations and gestures like in native apps).
`@react-navigation/bottom-tabs`	The bottom tab bar.
`react-native-screens`, `react-native-safe-area-context`	Native dependencies required by React Navigation.
`@expo/vector-icons`	Icon sets (Ionicons, MaterialIcons…). It is **not installed automatically** together with `expo` – you have to add it yourself.
`expo-font`	A required *peer dependency* of `@expo/vector-icons` – it loads the icon fonts. Without it `npx expo-doctor` reports an error and the app may crash outside Expo Go.

Library versions

- This instruction covers **React Navigation 7** (the current stable version). Version 8 is in testing (alpha). Check in `package.json` that all `@react-navigation/*` packages have version `^7.x` – if version 8 gets installed after its release, you can force version 7: `npx expo install @react-navigation/native@^7 ...`.
- **Do not mix React Navigation with Expo Router.** If the project was created from the *default* template (it has an `app/` folder), `npx expo-doctor` reports a conflict. Create the project from the `blank-typescript` template (Lab 1).
- The `@expo/vector-icons` library is being replaced by the `@react-native-vector-icons/*` packages, but it still works and is available in Expo Go – we use it in the labs.

If the computer and the phone are not on the same network, start the bundler in tunnel mode: `npx expo start --tunnel`.

Step 1: Folder structure and route types

Organize the code in the `src/` folder:

Project structure

```
App.tsx
src/
  theme.ts                <- shared colors
  navigation/
    types.ts              <- route and parameter types
    RootNavigator.tsx     <- Stack + Tabs + theme
  screens/
    LoginScreen.tsx
    HomeScreen.tsx
    ProfileScreen.tsx
    SettingsScreen.tsx
  components/
```

The most important file is `types.ts`. For every navigator we define a **parameter list** (*ParamList*): the key is the screen name, the value is the type of its parameters (`undefined` when the screen takes none).

File: src/navigation/types.ts

```
// src/navigation/types.ts
import type { CompositeScreenProps, NavigatorScreenParams } from "@react-navigation/native";
import type { NativeStackScreenProps } from "@react-navigation/native-stack";
import type { BottomTabScreenProps } from "@react-navigation/bottom-tabs";

// Tabs (bottom navigation)
export type AppTabsParamList = {
  Home: undefined; // a screen without parameters
  Settings: { section?: SettingsSection } | undefined; // an optional parameter
};

export type SettingsSection = "general" | "notifications" | "privacy";

// The main stack
export type RootStackParamList = {
  Login: undefined;
  // A nested navigator: allows navigate("AppTabs", { screen: "Settings" })
  AppTabs: NavigatorScreenParams<AppTabsParamList> | undefined;
  Profile: { userId: string }; // a required parameter
};
```

```
// Ready-made props types for the screens
export type RootStackScreenProps<T extends keyof RootStackParamList> =
  NativeStackScreenProps<RootStackParamList, T>;

// A tab screen “sees” both the tab navigation and the parent stack
export type AppTabsScreenProps<T extends keyof AppTabsParamList> = CompositeScreenProps<
  BottomTabScreenProps<AppTabsParamList, T>,
  RootStackScreenProps<keyof RootStackParamList>
>;

// Default typing for useNavigation() in the whole app
declare global {
  namespace ReactNavigation {
    // The empty interface is intentional here (it extends a library type)
    // eslint-disable-next-line @typescript-eslint/no-empty-object-type
    interface RootParamList extends RootStackParamList {}
  }
}
```

- `NavigatorScreenParams<AppTabsParamList>` – tells TypeScript that the `AppTabs` screen is a **nested navigator**; thanks to that you can safely call `navigate("AppTabs", { screen: "Settings", params: {...} })` without workarounds such as `as never`.
- `CompositeScreenProps` – a tab screen can navigate both to other tabs and to screens of the parent stack (e.g. `Profile`).
- `declare global { namespace ReactNavigation ... }` – sets the default type of `useNavigation()` in the whole app.

File: src/theme.ts

```
// src/theme.ts – a shared color palette for all screens
export const COLORS = {
  bg: "#0f1420",
  card: "#121a2a",
  border: "#1f2b40",
  text: "#e8eaed",
  subtext: "#a8b0bd",
  primary: "#4c9aff",
  danger: "#ff6b6b",
  overlay: "rgba(0,0,0,0.6)",
};
```

Step 2: Stack + Tabs – configuration and theme

File: src/navigation/RootNavigator.tsx

```
// src/navigation/RootNavigator.tsx
import { DarkTheme, NavigationContainer, type Theme } from "@react-navigation/native";
import {
  createNativeStackNavigator,
  type NativeStackNavigationOptions,
} from "@react-navigation/native-stack";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import { Ionicons } from "@expo/vector-icons";
import type { AppTabsParamList, RootStackParamList } from "./types";
import { COLORS } from "../theme";

import LoginScreen from "../screens/LoginScreen";
import HomeScreen from "../screens/HomeScreen";
import SettingsScreen from "../screens/SettingsScreen";
import ProfileScreen from "../screens/ProfileScreen";
```

```
const Stack = createNativeStackNavigator<RootStackParamList>();
const Tabs = createBottomTabNavigator<AppTabsParamList>();

// A custom theme – we extend the built-in DarkTheme (it also contains fonts)
const AppDarkTheme: Theme = {
  ...DarkTheme,
  colors: {
    ...DarkTheme.colors,
    background: COLORS.bg,
    card: COLORS.card,
    text: COLORS.text,
    border: COLORS.border,
    primary: COLORS.primary,
  },
};

type IconName = keyof typeof Ionicons.glyphMap;

// The [active, inactive] icon for every tab
const TAB_ICONS: Record<keyof AppTabsParamList, [IconName, IconName]> = {
  Home: ["home", "home-outline"],
  Settings: ["settings", "settings-outline"],
};

function AppTabsNavigator() {
  return (
    <Tabs.Navigator
      screenOptions={({ route }) => ({
        headerStyle: { backgroundColor: COLORS.card },
        headerTitleStyle: { color: COLORS.text },
        tabBarStyle: { backgroundColor: COLORS.card, borderTopColor: COLORS.border },
        tabBarActiveTintColor: COLORS.primary,
        tabBarInactiveTintColor: COLORS.subtext,
        tabBarIcon: ({ focused, color, size }) => {
          const [active, inactive] = TAB_ICONS[route.name];
          return <Ionicons name={focused ? active : inactive} size={size} color={color} />;
        },
      })}
    >
      <Tabs.Screen name="Home" component={HomeScreen} options={{ title: "Home" }} />
      <Tabs.Screen
        name="Settings"
        component={SettingsScreen}
        options={{ title: "Settings" }}
      />
    </Tabs.Navigator>
  );
}

const stackScreenOptions: NativeStackNavigationOptions = {
  headerStyle: { backgroundColor: COLORS.card },
  headerTitleStyle: { color: COLORS.text },
  headerTintColor: COLORS.text, // the color of the back arrow and buttons
  contentStyle: { backgroundColor: COLORS.bg }, // no white flashes during transitions
};

export default function RootNavigator() {
  return (
    <NavigationContainer theme={AppDarkTheme}>
      <Stack.Navigator screenOptions={stackScreenOptions} initialRouteName="Login">
        <Stack.Screen
          name="Login"
          component={LoginScreen}
          options={{ headerShown: false }}
        />
        <Stack.Screen
```

```
          name="AppTabs"
          component={AppTabsNavigator}
          options={{ headerShown: false }} // the tabs show the header (no duplication)
        />
        <Stack.Screen
          name="Profile"
          component={ProfileScreen}
          // A dynamic title – it depends on the route parameter (fully typed)
          options={({ route }) => ({ title: `Profile: ${route.params.userId}` })}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

- There is **exactly one** `NavigationContainer` in the whole app – we nest navigators, but not the container.
- The `AppTabs` screen is a whole tab navigator inserted as a stack screen. It has `headerShown: false` because the tabs draw the header themselves – otherwise **two headers** appear.
- The `TAB_ICONS` map is typed with the keys of `AppTabsParamList` – adding a tab without an icon ends with a TypeScript error.
- `contentStyle` in the stack and a custom `Theme` remove white background "flashes" during transition animations in the dark theme.

Step 3: Integration with App.tsx

File: App.tsx

```
// App.tsx
import { StatusBar } from "expo-status-bar";
import { SafeAreaProvider } from "react-native-safe-area-context";
import RootNavigator from "./src/navigation/RootNavigator";

export default function App() {
  return (
    <SafeAreaProvider>
      <StatusBar style="light" />
      <RootNavigator />
    </SafeAreaProvider>
  );
}
```

`SafeAreaProvider` must wrap `NavigationContainer` – headers and the tab bar take the safe edges into account themselves.

Step 4: Screens – typed props, useNavigation, useRoute

The sign-in screen (Login → AppTabs)

The screen has no header, so it handles all the safe edges itself. `navigation.replace()` replaces the sign-in screen with the tabs – the back button/gesture will not return to sign-in. `KeyboardAvoidingView` + `ScrollView` keep the fields from being covered by the keyboard.

File: src/screens/LoginScreen.tsx

```
// src/screens/LoginScreen.tsx
import { useState } from "react";
import {
  KeyboardAvoidingView,
  Platform,
```

```
  Pressable,
  ScrollView,
  StyleSheet,
  Switch,
  Text,
  TextInput,
  View,
} from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";
import { Ionicons } from "@expo/vector-icons";
import type { RootStackScreenProps } from "../navigation/types";
import { COLORS } from "../theme";

type Props = RootStackScreenProps<"Login">;

export default function LoginScreen({ navigation }: Props) {
  const [email, setEmail] = useState("");
  const [pwd, setPwd] = useState("");
  const [showPwd, setShowPwd] = useState(false);
  const [remember, setRemember] = useState(true);

  const canLogin = email.includes("@") && pwd.length >= 3;

  const onLogin = () => {
    if (!canLogin) return;
    // replace: the login screen cannot be reached again with the back button
    navigation.replace("AppTabs", { screen: "Home" });
  };

  return (
    // A screen without a header -> we handle all the safe edges ourselves
    <SafeAreaView style={styles.safe} edges={["top", "right", "bottom", "left"]}>
      <KeyboardAvoidingView
        style={{ flex: 1 }}
        behavior={Platform.OS === "ios" ? "padding" : undefined}
      >
        <ScrollView
          contentContainerStyle={styles.container}
          keyboardShouldPersistTaps="handled"
        >
          <View style={styles.header}>
            <View style={styles.logoWrap}>
              <Ionicons name="sparkles-outline" size={28} color={COLORS.primary} />
            </View>
            <Text style={styles.title}>Sign in</Text>
            <Text style={styles.subtitle}>Welcome back</Text>
          </View>

          <View style={styles.card}>
            <Text style={styles.label}>E-mail</Text>
            <View style={styles.inputWrap}>
              <Ionicons name="mail-outline" size={18} color={COLORS.subtext} />
              <TextInput
                style={styles.input}
                value={email}
                onChangeText={setEmail}
                placeholder="you@example.com"
                placeholderTextColor="#7c8799"
                keyboardType="email-address"
                autoCapitalize="none"
                autoCorrect={false}
                returnKeyType="next"
              />
            </View>
```

```
            <Text style={[styles.label, { marginTop: 14 }]}>Password</Text>
            <View style={styles.inputWrap}>
              <Ionicons name="lock-closed-outline" size={18} color={COLORS.subtext} />
              <TextInput
                style={styles.input}
                value={pwd}
                onChangeText={setPwd}
                placeholder="••••••••"
                placeholderTextColor="#7c8799"
                secureTextEntry={!showPwd}
                autoCapitalize="none"
                onSubmitEditing={onLogin}
              />
              <Pressable onPress={() => setShowPwd((v) => !v)} hitSlop={8}>
                <Ionicons
                  name={showPwd ? "eye-off-outline" : "eye-outline"}
                  size={18}
                  color={COLORS.subtext}
                />
              </Pressable>
            </View>

            <View style={styles.row}>
              <View style={styles.rowLeft}>
                <Switch
                  value={remember}
                  onValueChange={setRemember}
                  trackColor={{ false: "#3a455a", true: "#2e6db3" }}
                  thumbColor={remember ? COLORS.primary : "#dfe3ea"}
                />
                <Text style={styles.remember}>Remember me</Text>
              </View>
              <Pressable>
                <Text style={styles.forgot}>Forgot your password?</Text>
              </Pressable>
            </View>

            <Pressable
              onPress={onLogin}
              disabled={!canLogin}
              android_ripple={{ color: "rgba(0,0,0,0.12)" }}
              style={[styles.btn, !canLogin && { opacity: 0.5 }]}
            >
              <Text style={styles.btnText}>Enter the app</Text>
            </Pressable>

            <Pressable
              onPress={() => navigation.replace("AppTabs")}
              android_ripple={{ color: "rgba(255,255,255,0.08)" }}
              style={styles.btnGhost}
            >
              <Ionicons name="person-outline" size={16} color={COLORS.text} />
              <Text style={styles.btnGhostText}>Continue as a guest</Text>
            </Pressable>
          </View>
        </ScrollView>
      </KeyboardAvoidingView>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  safe: { flex: 1, backgroundColor: COLORS.bg },
  container: { flexGrow: 1, justifyContent: "center", padding: 24, gap: 16 },
  header: { alignItems: "center", gap: 6, marginBottom: 6 },
  logoWrap: {
```

```
    width: 56,
    height: 56,
    borderRadius: 16,
    backgroundColor: "rgba(76,154,255,0.12)",
    borderWidth: 1,
    borderColor: "rgba(76,154,255,0.25)",
    alignItems: "center",
    justifyContent: "center",
  },
  title: { color: COLORS.text, fontSize: 22, fontWeight: "800" },
  subtitle: { color: COLORS.subtext, fontSize: 14 },
  card: {
    backgroundColor: COLORS.card,
    borderRadius: 16,
    borderWidth: 1,
    borderColor: COLORS.border,
    padding: 16,
    gap: 10,
  },
  label: {
    color: COLORS.subtext,
    fontSize: 12,
    textTransform: "uppercase",
    letterSpacing: 0.4,
  },
  inputWrap: {
    flexDirection: "row",
    alignItems: "center",
    gap: 10,
    backgroundColor: "rgba(255,255,255,0.03)",
    borderWidth: 1,
    borderColor: "rgba(255,255,255,0.08)",
    borderRadius: 12,
    paddingHorizontal: 12,
    height: 48,
  },
  input: { flex: 1, color: COLORS.text, fontSize: 16 },
  row: { flexDirection: "row", alignItems: "center", justifyContent: "space-between" },
  rowLeft: { flexDirection: "row", alignItems: "center", gap: 8 },
  remember: { color: COLORS.subtext, fontSize: 13 },
  forgot: { color: COLORS.primary, fontSize: 13, fontWeight: "700" },
  btn: {
    backgroundColor: COLORS.primary,
    borderRadius: 12,
    alignItems: "center",
    justifyContent: "center",
    height: 48,
    marginTop: 8,
  },
  btnText: { color: "#0b0e13", fontWeight: "800", fontSize: 15 },
  btnGhost: {
    height: 46,
    borderRadius: 12,
    borderWidth: 1,
    borderColor: "rgba(255,255,255,0.18)",
    alignItems: "center",
    justifyContent: "center",
    flexDirection: "row",
    gap: 8,
  },
  btnGhostText: { color: COLORS.text, fontWeight: "700" },
});
```

The home screen – passing parameters

A tab screen receives `navigation` and `route` as props (type `AppTabsScreenProps<"Home">`). The example shows three ways of navigating and signing out with `reset`.

File: src/screens/HomeScreen.tsx

```
// src/screens/HomeScreen.tsx
import { Alert, FlatList, Platform, Pressable, StyleSheet, Text, View } from "react-native";
import { Ionicons } from "@expo/vector-icons";
import type { NativeStackNavigationProp } from "@react-navigation/native-stack";
import type { AppTabsScreenProps, RootStackParamList } from "../navigation/types";
import { COLORS } from "../theme";

type Props = AppTabsScreenProps<"Home">;

type Tile = {
  key: string;
  title: string;
  icon: keyof typeof Ionicons.glyphMap;
  color: string;
  onPress: () => void;
};

const GAP = 16;

export default function HomeScreen({ navigation }: Props) {
  const logout = () => {
    // reset works on the navigator that contains the screen (here: the tabs).
    // The Login screen belongs to the parent stack, so we reset the parent.
    const rootNavigation =
      navigation.getParent<NativeStackNavigationProp<RootStackParamList>>();
    rootNavigation?.reset({ index: 0, routes: [{ name: "Login" }] });
  };

  const tiles: Tile[] = [
    {
      key: "profile",
      title: "Profile",
      icon: "person",
      color: "#4C9AFF",
      // A screen from the parent stack + a required parameter
      onPress: () => navigation.navigate("Profile", { userId: "john_smith" }),
    },
    {
      key: "settings",
      title: "Notifications",
      icon: "notifications",
      color: "#7C4DFF",
      // Another tab + an optional parameter
      onPress: () => navigation.navigate("Settings", { section: "notifications" }),
    },
    {
      key: "privacy",
      title: "Privacy",
      icon: "shield-checkmark",
      color: "#00C853",
      // The full path through the nested navigator (works from anywhere)
      onPress: () =>
        navigation.navigate("AppTabs", {
          screen: "Settings",
          params: { section: "privacy" },
        }),
    },
    {
      key: "logout",
```

```
      title: "Sign out",
      icon: "log-out",
      color: "#FF7043",
      onPress: () =>
        Alert.alert("Sign out", "Are you sure you want to sign out?", [
          { text: "Cancel", style: "cancel" },
          { text: "Sign out", style: "destructive", onPress: logout },
        ]),
    },
  ];

  return (
    <View style={styles.container}>
      <FlatList
        data={tiles}
        keyExtractor={(item) => item.key}
        numColumns={2}
        columnWrapperStyle={{ gap: GAP }}
        contentContainerStyle={{ gap: GAP, paddingBottom: GAP }}
        renderItem={({ item }) => (
          <Pressable
            onPress={item.onPress}
            android_ripple={{ color: "rgba(0,0,0,0.12)" }}
            style={({ pressed }) => [
              styles.tile,
              { backgroundColor: item.color },
              pressed && Platform.OS === "ios" && { opacity: 0.8 },
            ]}
          >
            <Ionicons name={item.icon} size={28} color="#FFFFFF" />
            <Text style={styles.tileText}>{item.title}</Text>
          </Pressable>
        )}
      />
    </View>
  );
}

const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: COLORS.bg, padding: 24 },
  tile: {
    flex: 1,
    height: 110,
    borderRadius: 16,
    justifyContent: "center",
    alignItems: "center",
    borderWidth: 1,
    borderColor: "rgba(255,255,255,0.12)",
    overflow: "hidden", // clips the ripple effect to the rounded corners
  },
  tileText: { marginTop: 8, color: "#FFFFFF", fontWeight: "700" },
});
```

reset() and nested navigators

The `navigation.reset()`, `goBack()`, `setOptions()` methods work on the navigator **that contains the screen**. The `Home` screen is in the tabs and `Login` is in the parent stack – that is why we sign out on the parent: `navigation.getParent()`. Calling `navigation.reset({ routes: [{ name: "Login" }] })` directly in a tab is a **TypeScript error**, because `reset` is typed for the tabs, which have no `Login` screen. At runtime an action the tabs cannot handle is passed to the parent and works – but only "by accident". An explicit `getParent()` is readable and correctly typed. The `navigate()` method, by design, also looks for the screen in parent navigators.

The profile screen – reading a parameter with useRoute

File: src/screens/ProfileScreen.tsx

```
// src/screens/ProfileScreen.tsx
import { useMemo, useState } from "react";
import { Pressable, ScrollView, StyleSheet, Switch, Text, View } from "react-native";
import { useRoute, type RouteProp } from "@react-navigation/native";
import { SafeAreaView } from "react-native-safe-area-context";
import { Ionicons } from "@expo/vector-icons";
import type { RootStackParamList } from "../navigation/types";
import { COLORS } from "../theme";

type ProfileRoute = RouteProp<RootStackParamList, "Profile">;
type IconName = keyof typeof Ionicons.glyphMap;

export default function ProfileScreen() {
  // An alternative to the `route` prop: the useRoute hook with the route type
  const route = useRoute<ProfileRoute>();
  const { userId } = route.params; // TypeScript knows that userId: string

  // "john_smith" -> "John Smith"
  const displayName = useMemo(() => {
    const parts = userId
      .replace(/[^a-zA-Z0-9]+/g, " ")
      .trim()
      .split(" ")
      .filter(Boolean);
    if (parts.length === 0) return "User";
    return parts.slice(0, 2).map(capitalize).join(" ");
  }, [userId]);

  const initials = displayName
    .split(" ")
    .map((s) => s[0]?.toUpperCase() ?? "")
    .join("");

  const [isPrivate, setIsPrivate] = useState(false);
  const [twoFA, setTwoFA] = useState(true);

  return (
    // The stack header handles the top – we only take care of the sides and the bottom
    <SafeAreaView style={styles.safe} edges={["left", "right", "bottom"]}>
      <ScrollView contentContainerStyle={styles.content}>
        <View style={styles.headerCard}>
          <View style={styles.avatar}>
            <Text style={styles.avatarText}>{initials}</Text>
          </View>
          <Text style={styles.name}>{displayName}</Text>
          <Text style={styles.handle}>@{userId}</Text>
          <View style={styles.stats}>
            <Stat value="128" label="Followers" />
            <Stat value="87" label="Following" />
            <Stat value="24" label="Posts" />
          </View>
        </View>

        <View style={styles.card}>
          <Text style={styles.cardTitle}>Details</Text>
          <Item icon="mail-outline" label="E-mail" value="user@example.com" />
          <Divider />
          <Item icon="call-outline" label="Phone" value="+44 7700 900000" />
          <Divider />
          <Item icon="calendar-outline" label="Joined" value="12 Mar 2024" />
        </View>
```

```
        <View style={styles.card}>
          <Text style={styles.cardTitle}>Security</Text>
          <ToggleItem
            icon="shield-outline"
            label="Two-factor authentication"
            value={twoFA}
            onValueChange={setTwoFA}
          />
          <Divider />
          <ToggleItem
            icon="lock-closed-outline"
            label="Private account"
            value={isPrivate}
            onValueChange={setIsPrivate}
          />
        </View>

        <Pressable
          style={styles.rowPress}
          android_ripple={{ color: "rgba(255,255,255,0.06)" }}
        >
          <IconBadge name="share-social-outline" />
          <Text style={[styles.rowLabel, { flex: 1 }]}>Share profile</Text>
          <Ionicons name="chevron-forward" size={18} color={COLORS.subtext} />
        </Pressable>
      </ScrollView>
    </SafeAreaView>
  );
}

function Stat({ value, label }: { value: string; label: string }) {
  return (
    <View style={styles.stat}>
      <Text style={styles.statValue}>{value}</Text>
      <Text style={styles.statLabel}>{label}</Text>
    </View>
  );
}

function IconBadge({ name }: { name: IconName }) {
  return (
    <View style={styles.iconBadge}>
      <Ionicons name={name} size={18} color={COLORS.primary} />
    </View>
  );
}

function Item({ icon, label, value }: { icon: IconName; label: string; value: string }) {
  return (
    <View style={styles.row}>
      <View style={styles.rowLeft}>
        <IconBadge name={icon} />
        <Text style={styles.rowLabel}>{label}</Text>
      </View>
      <Text style={styles.rowValue}>{value}</Text>
    </View>
  );
}

type ToggleItemProps = {
  icon: IconName;
  label: string;
  value: boolean;
  onValueChange: (v: boolean) => void;
};
```

```
function ToggleItem({ icon, label, value, onValueChange }: ToggleItemProps) {
  return (
    <View style={styles.row}>
      <View style={styles.rowLeft}>
        <IconBadge name={icon} />
        <Text style={styles.rowLabel}>{label}</Text>
      </View>
      <Switch
        value={value}
        onValueChange={onValueChange}
        trackColor={{ false: "#3a455a", true: "#2e6db3" }}
        thumbColor={value ? COLORS.primary : "#dfe3ea"}
      />
    </View>
  );
}

function Divider() {
  return <View style={styles.divider} />;
}

function capitalize(s: string) {
  return s.charAt(0).toUpperCase() + s.slice(1).toLowerCase();
}

const styles = StyleSheet.create({
  safe: { flex: 1, backgroundColor: COLORS.bg },
  content: { padding: 16, gap: 16 },
  headerCard: {
    backgroundColor: COLORS.card,
    borderRadius: 16,
    borderWidth: 1,
    borderColor: COLORS.border,
    padding: 16,
    alignItems: "center",
    gap: 10,
  },
  avatar: {
    width: 84,
    height: 84,
    borderRadius: 42,
    backgroundColor: "rgba(76,154,255,0.15)",
    borderWidth: 2,
    borderColor: "rgba(76,154,255,0.35)",
    alignItems: "center",
    justifyContent: "center",
  },
  avatarText: { color: COLORS.primary, fontSize: 28, fontWeight: "800" },
  name: { color: COLORS.text, fontSize: 20, fontWeight: "800" },
  handle: { color: COLORS.subtext, fontSize: 14 },
  stats: { flexDirection: "row", gap: 18, marginTop: 6 },
  stat: { alignItems: "center", paddingHorizontal: 10 },
  statValue: { color: COLORS.text, fontSize: 16, fontWeight: "800" },
  statLabel: { color: COLORS.subtext, fontSize: 12 },
  card: {
    backgroundColor: COLORS.card,
    borderRadius: 16,
    borderWidth: 1,
    borderColor: COLORS.border,
    paddingVertical: 8,
  },
  cardTitle: {
    color: COLORS.subtext,
    fontSize: 12,
    textTransform: "uppercase",
```

```
    letterSpacing: 0.4,
    paddingHorizontal: 12,
    marginVertical: 6,
  },
  row: {
    minHeight: 54,
    paddingHorizontal: 12,
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "space-between",
  },
  rowPress: {
    minHeight: 54,
    paddingHorizontal: 12,
    flexDirection: "row",
    alignItems: "center",
    gap: 10,
    backgroundColor: COLORS.card,
    borderRadius: 16,
    borderWidth: 1,
    borderColor: COLORS.border,
  },
  rowLeft: { flexDirection: "row", alignItems: "center", gap: 10 },
  iconBadge: {
    width: 28,
    height: 28,
    borderRadius: 8,
    backgroundColor: "rgba(76,154,255,0.12)",
    alignItems: "center",
    justifyContent: "center",
  },
  rowLabel: { color: COLORS.text, fontSize: 16, fontWeight: "600" },
  rowValue: { color: COLORS.subtext, fontSize: 14 },
  divider: { height: 1, backgroundColor: "rgba(255,255,255,0.06)", marginHorizontal: 12 },
});
```

The settings screen – an optional parameter and a modal

File: src/screens/SettingsScreen.tsx

```
// src/screens/SettingsScreen.tsx
import { useState, type ReactNode } from "react";
import { Modal, Pressable, ScrollView, StyleSheet, Switch, Text, View } from "react-native";
import { Ionicons } from "@expo/vector-icons";
import type { AppTabsScreenProps, SettingsSection } from "../navigation/types";
import { COLORS } from "../theme";

type Props = AppTabsScreenProps<"Settings">;
type IconName = keyof typeof Ionicons.glyphMap;

const SECTION_TITLES: Record<SettingsSection, string> = {
  general: "General",
  notifications: "Notifications",
  privacy: "Privacy",
};

export default function SettingsScreen({ route, navigation }: Props) {
  // An optional parameter -> a default value when it is missing
  const section: SettingsSection = route.params?.section ?? "general";

  const [pushEnabled, setPushEnabled] = useState(true);
  const [emailEnabled, setEmailEnabled] = useState(false);
  const [analyticsEnabled, setAnalyticsEnabled] = useState(true);
  const [language, setLanguage] = useState<"en" | "pl">("en");
  const [showLangModal, setShowLangModal] = useState(false);
```

```
  return (
    // The header handles the top, the tab bar the bottom: no SafeAreaView needed
    <View style={styles.screen}>
      <ScrollView contentContainerStyle={styles.scrollContent}>
        <Text style={styles.hint}>Open section: {SECTION_TITLES[section]}</Text>

        <GroupCard title="Account">
          <Row
            icon="person-circle-outline"
            label="User profile"
            value="john_smith"
            onPress={() => navigation.navigate("Profile", { userId: "john_smith" })}
          />
        </GroupCard>

        <GroupCard title={SECTION_TITLES.general} highlighted={section === "general"}>
          <Row
            icon="language-outline"
            label="App language"
            value={language === "en" ? "English" : "Polski"}
            onPress={() => setShowLangModal(true)}
          />
        </GroupCard>

        <GroupCard
          title={SECTION_TITLES.notifications}
          highlighted={section === "notifications"}
        >
          <ToggleRow
            icon="notifications-outline"
            label="Push notifications"
            value={pushEnabled}
            onValueChange={setPushEnabled}
          />
          <ToggleRow
            icon="mail-unread-outline"
            label="Email notifications"
            value={emailEnabled}
            onValueChange={setEmailEnabled}
          />
        </GroupCard>

        <GroupCard title={SECTION_TITLES.privacy} highlighted={section === "privacy"}>
          <ToggleRow
            icon="analytics-outline"
            label="Usage analytics"
            value={analyticsEnabled}
            onValueChange={setAnalyticsEnabled}
          />
        </GroupCard>
      </ScrollView>

      <PickerModal
        visible={showLangModal}
        title="Choose a language"
        options={[
          { key: "en", label: "English" },
          { key: "pl", label: "Polski" },
        ]}
        selectedKey={language}
        onSelect={(k) => setLanguage(k === "pl" ? "pl" : "en")}
        onClose={() => setShowLangModal(false)}
      />
    </View>
  );
}
```

```
function GroupCard({
  title,
  highlighted = false,
  children,
}: {
  title: string;
  highlighted?: boolean;
  children: ReactNode;
}) {
  return (
    <View style={[styles.card, highlighted && { borderColor: COLORS.primary }]}>
      <Text style={[styles.cardTitle, highlighted && { color: COLORS.primary }]}>
        {title}
      </Text>
      {children}
    </View>
  );
}

function Row({
  icon,
  label,
  value,
  onPress,
}: {
  icon: IconName;
  label: string;
  value?: string;
  onPress?: () => void;
}) {
  return (
    <Pressable
      onPress={onPress}
      android_ripple={{ color: "rgba(255,255,255,0.08)" }}
      style={styles.row}
    >
      <View style={styles.rowLeft}>
        <Ionicons name={icon} size={20} color={COLORS.primary} />
        <Text style={styles.rowLabel}>{label}</Text>
      </View>
      <View style={styles.rowLeft}>
        {!!value && <Text style={styles.rowValue}>{value}</Text>}
        {onPress && <Ionicons name="chevron-forward" size={18} color={COLORS.subtext} />}
      </View>
    </Pressable>
  );
}

function ToggleRow({
  icon,
  label,
  value,
  onValueChange,
}: {
  icon: IconName;
  label: string;
  value: boolean;
  onValueChange: (v: boolean) => void;
}) {
  return (
    <View style={styles.row}>
      <View style={styles.rowLeft}>
        <Ionicons name={icon} size={20} color={COLORS.primary} />
        <Text style={styles.rowLabel}>{label}</Text>
      </View>
      <Switch value={value} onValueChange={onValueChange} />
```

```
    </View>
  );
}

type PickerModalProps = {
  visible: boolean;
  title: string;
  options: { key: string; label: string }[];
  selectedKey: string;
  onSelect: (key: string) => void;
  onClose: () => void;
};

function PickerModal({
  visible,
  title,
  options,
  selectedKey,
  onSelect,
  onClose,
}: PickerModalProps) {
  return (
    // onRequestClose – handles the system back button on Android
    <Modal transparent visible={visible} animationType="fade" onRequestClose={onClose}>
      <View style={styles.modalOverlay}>
        {/* Tapping the background outside the card closes the dialog */}
        <Pressable style={StyleSheet.absoluteFill} onPress={onClose} />
        <View style={styles.modalCard}>
          <Text style={styles.modalTitle}>{title}</Text>
          {options.map((opt) => {
            const selected = opt.key === selectedKey;
            return (
              <Pressable
                key={opt.key}
                style={styles.modalOption}
                onPress={() => {
                  onSelect(opt.key);
                  onClose();
                }}
              >
                <Text
                  style={[styles.modalOptionText, selected && { color: COLORS.primary }]}
                >
                  {opt.label}
                </Text>
                {selected && <Ionicons name="checkmark" size={18} color={COLORS.primary} />}
              </Pressable>
            );
          })}
        </View>
      </View>
    </Modal>
  );
}

const styles = StyleSheet.create({
  screen: { flex: 1, backgroundColor: COLORS.bg },
  scrollContent: { padding: 16, paddingBottom: 32, gap: 16 },
  hint: { color: COLORS.subtext },
  card: {
    backgroundColor: COLORS.card,
    borderRadius: 16,
    borderWidth: 1,
    borderColor: COLORS.border,
    padding: 12,
  },
  cardTitle: {
```

```
    color: COLORS.subtext,
    fontSize: 12,
    letterSpacing: 0.4,
    textTransform: "uppercase",
    marginBottom: 8,
  },
  row: {
    minHeight: 50,
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "space-between",
  },
  rowLeft: { flexDirection: "row", alignItems: "center", gap: 10 },
  rowLabel: { color: COLORS.text, fontSize: 16, fontWeight: "600" },
  rowValue: { color: COLORS.subtext, fontSize: 14 },
  modalOverlay: {
    flex: 1,
    backgroundColor: COLORS.overlay,
    alignItems: "center",
    justifyContent: "center",
    padding: 24,
  },
  modalCard: {
    width: "100%",
    backgroundColor: COLORS.card,
    borderRadius: 16,
    borderWidth: 1,
    borderColor: COLORS.border,
    padding: 16,
    gap: 8,
  },
  modalTitle: { color: COLORS.text, fontSize: 16, fontWeight: "800", marginBottom: 8 },
  modalOption: {
    minHeight: 44,
    flexDirection: "row",
    alignItems: "center",
    justifyContent: "space-between",
  },
  modalOptionText: { color: COLORS.text, fontSize: 15 },
});
```

Tab parameters when you come back

Tabs are not destroyed when you switch between them – the `Settings` screen stays alive all the time. When you call `navigate("Settings", { section: "privacy" })`, only `route.params` changes and the screen's state (`useState`) is kept. If something should happen every time the screen is entered, use the `useFocusEffect` or `useIsFocused` hook.

Step 5: Headers – titles, hiding, buttons

Goal	How
Hide the header	`options={{ headerShown: false }}`
A fixed title	`options={{ title: "My title" }}`
A title that depends on a parameter	`options={({ route }) => ({ title:` Profile: ${route.params.userId} `})}` (see `RootNavigator.tsx`)
A common look for all screens	`screenOptions` on `Stack.Navigator` / `Tabs.Navigator`
The color of the back arrow and buttons	`headerTintColor`

Goal	How
A badge on a tab	`options={{ tabBarBadge: 3 }}`
Changing the header from inside a screen	`navigation.setOptions({...})` in `useLayoutEffect` – example below

File: Example: a header button set from the screen

```
// A header button set from the screen (navigation.setOptions)
import { useLayoutEffect, useState } from "react";
import { Pressable, Text } from "react-native";
import { Ionicons } from "@expo/vector-icons";
import type { RootStackScreenProps } from "../navigation/types";

export default function ProfileWithStar({
  navigation,
  route,
}: RootStackScreenProps<"Profile">) {
  const [starred, setStarred] = useState(false);

  // useLayoutEffect: the header changes before the screen is drawn
  useLayoutEffect(() => {
    navigation.setOptions({
      headerRight: () => (
        <Pressable onPress={() => setStarred((s) => !s)} hitSlop={10}>
          <Ionicons name={starred ? "star" : "star-outline"} size={22} color="#ffd54f" />
        </Pressable>
      ),
    });
  }, [navigation, starred]);

  return <Text>{route.params.userId}</Text>;
}
```

Step 6: History operations and route protection

Method	What it does
`navigate("X", params)`	Go to screen X. If X is currently displayed, only its parameters are updated.
`popTo("X")`	Go back to an earlier screen X on the stack (removing the screens above it).
`push("X", params)`	Always add a new copy of the screen to the stack (e.g. details → more details).
`goBack()`	Go back one screen.
`popToTop()`	Go back to the first screen of the stack.
`replace("X")`	Replace the current screen (e.g. after signing in).
`reset({ index, routes })`	Replace the whole history (e.g. when signing out).
`setParams({...})`	Change the parameters of the current screen.

Route protection (guard) – the recommended pattern

Instead of manually moving between sign-in and the app, React Navigation recommends **rendering screens conditionally** based on the sign-in state. Screens that are not available in a given state

simply do not exist in the navigator – you cannot open them or go “back” to them. We keep the sign-in state in a context (Context API is covered in detail in Lab 5):

File: src/auth/AuthContext.tsx

```
// src/auth/AuthContext.tsx – a minimal “signed-in state” available in the whole app
import { createContext, useContext, useMemo, useState, type ReactNode } from "react";

type AuthContextValue = {
  isSignedIn: boolean;
  signIn: () => void;
  signOut: () => void;
};

const AuthContext = createContext<AuthContextValue | null>(null);

export function AuthProvider({ children }: { children: ReactNode }) {
  const [isSignedIn, setIsSignedIn] = useState(false);

  const value = useMemo(
    () => ({
      isSignedIn,
      signIn: () => setIsSignedIn(true),
      signOut: () => setIsSignedIn(false),
    }),
    [isSignedIn],
  );

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
}

export function useAuth() {
  const ctx = useContext(AuthContext);
  if (!ctx) throw new Error("useAuth must be used inside <AuthProvider>");
  return ctx;
}
```

File: src/navigation/RootNavigator.tsx (part – the guard variant)

```
// Part of RootNavigator.tsx – the variant with route protection (guard)
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import { useAuth } from "../auth/AuthContext";
import type { RootStackParamList } from "./types";
import LoginScreen from "../screens/LoginScreen";
import ProfileScreen from "../screens/ProfileScreen";

const Stack = createNativeStackNavigator<RootStackParamList>();

// AppTabsNavigator – unchanged, as in Step 2

export function GuardedStack() {
  const { isSignedIn } = useAuth();

  return (
    <Stack.Navigator>
      {isSignedIn ? (
        // Signed in: the Login screen does NOT exist in the navigator at all
        <Stack.Group>
          <Stack.Screen
            name="AppTabs"
            component={AppTabsNavigator}
            options={{ headerShown: false }}
          />
          <Stack.Screen name="Profile" component={ProfileScreen} />
```

```
        </Stack.Group>
      ) : (
        // Signed out: only Login is available
        <Stack.Screen
          name="Login"
          component={LoginScreen}
          options={{ headerShown: false }}
        />
      )}
    </Stack.Navigator>
  );
}
```

- In `App.tsx` we additionally wrap the app in `<AuthProvider>`.
- In `LoginScreen`, instead of `navigation.replace("AppTabs")`, we call `signIn()` from `useAuth()`; when signing out – `signOut()` instead of `reset`. The navigator switches the screens **by itself** when the state changes.
- In this pattern do not call `navigate("Login")` manually after signing out – the `Login` screen is not in the tree until the state changes (error *was not handled by any navigator*).

Common problems and solutions

Symptom	Cause	Solution
Couldn't find a navigation object. Is your component inside NavigationContainer?	`useNavigation()` used outside `NavigationContainer` (e.g. in `App.tsx`)	Use navigation hooks only in screens and their children.
The action 'NAVIGATE' with payload {"name":"X"} was not handled by any navigator	A typo in the screen name, the screen does not exist in the current tree (guard), or the screen is in another, nested navigator	Check the name in `types.ts`; for nested screens use `navigate("AppTabs", { screen: "X" })`.
A TS error for `navigation.reset({ routes: [{ name: "Login" }] })` in a tab	`reset()` is typed for the tabs, while the `Login` screen is in the stack	`navigation.getParent()?.reset(...)` – see Step 4.
`npx expo-doctor`: *Missing peer dependency: expo-font*	`expo-font`, required by `@expo/vector-icons`, is missing	`npx expo install expo-font`.
Unable to resolve module @expo/vector-icons	The package is not installed together with `expo`	`npx expo install @expo/vector-icons`, then `npx expo start -c`.
Icons show up as squares / question marks	A non-existent icon name, or the fonts have not loaded yet	Check the name on icons.expo.fyi; use the type `keyof typeof Ionicons.glyphMap`.
Two headers on a tab screen	The stack header and the tab header at the same time	`headerShown: false` on the `AppTabs` screen in the stack.
Double spacing under the header	`SafeAreaView` with the `top` edge on a screen with a header	`edges={["left", "right", "bottom"]}` or a plain `View`.
`route.params` is `undefined` → *Cannot read property 'userId' of undefined*	Navigating to the screen without the required parameter (e.g. via `as never`)	Fix the types – TypeScript will force you to pass the parameter; for optional ones use `route.params?.x ?? default`.
A TS error in `navigate` that needs `as never`	`NavigatorScreenParams` is missing or the screen props types are wrong	Use the types from `types.ts` (Step 1).

Symptom	Cause	Solution
Invariant Violation: requireNativeComponent: "RNSScreen" / RNSScreenStack	`react-native-screens` in an incompatible version (installed with npm)	`npx expo install --fix`, `npx expo start -c`.
Looks like you have nested a 'NavigationContainer' inside another	A second `NavigationContainer` inside a screen	Keep only one – in `RootNavigator`.
The keyboard covers a text field	The keyboard is not handled	`KeyboardAvoidingView` (iOS: `behavior="padding"`) + `ScrollView` with `keyboardShouldPersistTaps="handled"`.

Your task: the “Campus” app

> **Organization**
> You do the task during the lab by extending the project from this instruction (you can change, delete or reuse the screens from Steps 1–4). You present the result to the instructor on a device and explain the navigation code and the route types.

Build a university course catalog app with the following navigation structure:

Navigation map

```
Stack (RootStackParamList)
├── Login                          (no header)
├── AppTabs  ── Tabs (AppTabsParamList)
│               ├── Courses       "Courses"  - the list of courses
│               ├── Settings      "Settings" - optional param. section
│               └── Account       "Account"  - user details, sign-out
├── CourseDetails  { courseId: string }   - title = course name
└── Teacher        { teacherId: string }  - the teacher's details
```

Keep the course and teacher data in the `src/data/courses.ts` file. Template (extend it to **at least 6 courses** and 3 teachers):

File: src/data/courses.ts

```
// src/data/courses.ts – starter data for the task (extend it to at least 6 courses)
export type Teacher = { id: string; name: string; email: string; room: string };

export type Course = {
  id: string;
  name: string;
  ects: number;
  semester: number;
  type: "lecture" | "lab" | "project";
  teacherId: string;
  description: string;
};

export const TEACHERS: Teacher[] = [
  { id: "t1", name: "Dr Anna Novak", email: "anna.novak@university.edu", room: "B-204" },
  {
    id: "t2",
    name: "Peter Wright, MSc",
    email: "p.wright@university.edu",
```

```
    room: "C-12",
  },
];

export const COURSES: Course[] = [
  {
    id: "c1",
    name: "Mobile Applications",
    ects: 5,
    semester: 5,
    type: "lab",
    teacherId: "t2",
    description: "React Native, Expo, navigation, working with APIs and native features.",
  },
  {
    id: "c2",
    name: "Databases",
    ects: 6,
    semester: 3,
    type: "lecture",
    teacherId: "t1",
    description: "The relational model, SQL, normalization, transactions.",
  },
];

export const findCourse = (id: string) => COURSES.find((c) => c.id === id);
export const findTeacher = (id: string) => TEACHERS.find((t) => t.id === id);
```

Required features

1. **Types:** the `src/navigation/types.ts` file with `RootStackParamList` and `AppTabsParamList` matching the map above (with `NavigatorScreenParams` for `AppTabs`) and the screen props types. There is no `any` or `as never` in the code; `npx tsc --noEmit` reports no errors.
2. **Login:** email and password fields; the "Sign in" button becomes active only when the email contains `@` and the password has at least 6 characters. After signing in – `navigation.replace("AppTabs", { screen: "Courses" })`; the back gesture/button does not return to sign-in.
3. **Tabs:** 3 tabs with Ionicons icons (a different icon for the active and inactive state), English titles, colors matching the shared theme (`theme.ts` + a custom `Theme` in `NavigationContainer`). The "Courses" tab has a `tabBarBadge` with the number of courses.
4. **Course list (Courses):** a `FlatList` with the courses (name, ECTS, semester, class type as a colored label). Tapping a course → `CourseDetails` with the `courseId` parameter.
5. **Course details (CourseDetails):** read `courseId` with `useRoute` **or** the typed `route` prop; find the course in the data; the header title is **dynamic** = the course name (via `options` in the navigator or `setOptions`). When the course does not exist – the message "Course not found" and a back button (`goBack()`).
6. **Going further:** on the details screen, a button with the teacher's name → the `Teacher` screen with the `teacherId` parameter (data: full name, email, room, a list of their courses – tapping a course on this list opens another `CourseDetails` with `push`).
7. **Navigating to a nested tab:** on the details screen, a "Notification settings" button → `navigate("AppTabs", { screen: "Settings", params: { section: "notifications" } })`. The Settings screen highlights (e.g. with a border) the section passed in, and without a parameter – the "General" section.
8. **Account and sign-out:** the "Account" tab shows the email of the signed-in user (pass it e.g. as a parameter when signing in or through a simple context) and a "Sign out" button with a

confirmation (`Alert.alert`). Signing out clears the history (`reset` on the parent navigator) and returns to `Login`.

9. **Safe Area:** no screen has double spacing under the header or content hidden under the tab bar or the gesture bar.

Extensions (optional)

- Route protection with `AuthContext` and conditional rendering of screens (Step 6) instead of `replace`/`reset`.
- A course search on the list (a `TextInput` filtering by name) and a semester filter.
- A "star" button in the details header (`setOptions`) that adds the course to favorites, and a fourth "Favorites" tab (state shared through a context).

Acceptance criteria

- ☐ Navigation works according to the navigation map; the back button/gesture behaves logically (it does not return to sign-in after signing in or after signing out).
- ☐ Parameters reach the screens and are correctly typed; no `any` / `as never`; `npx tsc --noEmit` without errors.
- ☐ The details title is dynamic; the case of a non-existent course is handled.
- ☐ A consistent theme (headers, tabs, screen backgrounds), no double headers or spacing.
- ☐ The student can explain: the difference between `navigate` / `push` / `replace` / `reset`, what `NavigatorScreenParams` is for, why `reset` on sign-out is called on the parent.