Développement d'application mobile — React Native

Guide complet du développement mobile avec React Native : écosystèmes iOS/Android, création de projet Expo Router, Firebase Authentication (email, Google, Apple), connexion à une API REST Express/Node.js, gestion des données locales, tests et déploiement sur les stores.

50 min de lecture Niveau Intermédiaire

Document généré le 25/06/2026 à 21h31 · nouvy.fr/wiki/developpement-application-mobile-react-native

Sommaire

44 section(s) · 50 min de lecture

Vue d'ensemble

Ce guide couvre le développement d'applications mobiles **cross-platform** avec **React Native (Expo)**, en s'appuyant sur **Firebase Authentication** (email, Google, Apple) et une **API REST Express/Node.js**.

Stack technique

Couche	Technologie	Rôle
Mobile	React Native + Expo	Interface iOS & Android
Routing	Expo Router	Navigation basée sur les fichiers
Langage	TypeScript	Typage statique
Authentification	Firebase Auth	Email, Google, Apple Sign-In
Backend	Express / Node.js	API REST
Requêtes HTTP	Axios	Communication avec l'API
État global	Context API + hook useAuth	Partage de l'état auth

---

Introduction — Écosystèmes mobiles

Les types de développement mobile

Type	Langages	Exemples	Avantages	Inconvénients
Natif	Swift / Kotlin	Apps système Apple/Google	Performances max, accès complet OS	2 codebases à maintenir
Cross-platform	JS/Dart	React Native, Flutter	1 seul code, proche du natif	Quelques limitations natives
Web mobile (PWA)	HTML/CSS/JS	Sites web adaptatifs	Pas de store requis	Accès limité aux APIs natives

React Native compile vers des composants natifs réels (pas une WebView), offrant des performances proches du natif avec un seul codebase.

Cycle de vie d'une application mobile

```
Lancement → Initialisation (Firebase, config) → Rendu interface
    │
    ├──► Actif        (l'utilisateur interagit)
    │         ↓
    │    Inactif      (appel entrant, notification)
    │         ↓
    │    Arrière-plan (app minimisée)
    │         ↓
    └─── Reprise      (retour au premier plan)
    │
    └──► Fermeture    (nettoyage listeners)
```

Copier

```
import { AppState } from 'react-native';
import { useEffect } from 'react';

useEffect(() => {
  const subscription = AppState.addEventListener('change', (nextState) => {
    if (nextState === 'active')     console.log('App au premier plan');
    if (nextState === 'background') console.log('App en arrière-plan');
  });
  return () => subscription.remove();
}, []);
```

Copier

Mise en place de l'environnement

Prérequis

Outil	Version	Rôle
Node.js	18 LTS minimum	Runtime JavaScript
npm	Inclus avec Node	Gestionnaire de paquets
VS Code	Dernière version	Éditeur recommandé
Expo Go	App mobile	Tester sur téléphone physique

Créer le projet

```
mkdir MonAppli
cd MonAppli
npx create-expo-app .
```

Copier

> *`create-expo-app .` génère un projet **Expo Router** avec **TypeScript** par défaut depuis le SDK 50+. C'est le standard actuel — le dossier `app/` remplace React Navigation et `App.jsx`.*

Structure générée et structure cible

Après `create-expo-app .`, le projet contient déjà :

```
MonAppli/
├── app/                ← Routing basé sur les fichiers (Expo Router)
├── assets/images/      ← Images et icônes
├── components/         ← Composants pré-générés (ThemedView, ThemedText…)
│   └── ui/
├── constants/          ← Couleurs, config
├── hooks/              ← Hooks pré-générés
├── scripts/
├── app.json
├── expo-env.d.ts       ← Types TypeScript pour Expo
└── package.json
```

Copier

On va **ajouter** les dossiers suivants sans toucher à ce qui existe :

```
MonAppli/
├── app/
│   ├── _layout.tsx          ← Layout racine (AuthProvider + redirection auth)
│   ├── (auth)/              ← Groupe d'écrans non connectés
│   │   ├── _layout.tsx
│   │   ├── login.tsx
│   │   └── register.tsx
│   └── (tabs)/              ← Groupe d'écrans connectés (onglets)
│       ├── _layout.tsx
│       ├── index.tsx        ← Accueil
│       └── profile.tsx
├── services/
│   ├── firebase.ts          ← Config Firebase
│   └── api.ts               ← Instance Axios
├── context/
│   └── AuthContext.tsx      ← État global auth
├── hooks/
│   ├── useAuth.ts           ← Raccourci AuthContext
│   └── useApi.ts            ← Appels API simplifiés
└── .env                     ← Variables d'environnement
```

Copier

Installer les dépendances

```
# Firebase
npx expo install firebase

# Google Sign-In
npx expo install expo-auth-session expo-web-browser expo-crypto

# Apple Sign-In
npx expo install expo-apple-authentication

# Requêtes HTTP
npm install axios
```

Copier

> *Expo Router est déjà inclus dans le projet généré, pas besoin de l'installer.*

Lancer le projet

```
npx expo start
```

Copier

Scanner le QR code avec **Expo Go** sur votre téléphone.

Expo Router — principe du routing par fichiers

Avec Expo Router, **le nom du fichier = l'URL = la route**. Pas besoin de déclarer les routes manuellement.

Fichier	Route	Description
`app/index.tsx`	`/`	Écran racine
`app/(auth)/login.tsx`	`/login`	Écran de connexion
`app/(tabs)/index.tsx`	`/` (onglets)	Accueil
`app/_layout.tsx`	—	Layout racine (wrapping global)

Les dossiers entre parenthèses `(auth)`, `(tabs)` sont des **groupes** : ils organisent les fichiers sans apparaître dans l'URL.

La navigation se fait avec le hook `useRouter` ou le composant `<Link>` :

```
import { useRouter } from 'expo-router';

const router = useRouter();
router.replace('/(auth)/login');   // Remplace (pas de retour arrière)
router.push('/(tabs)');            // Empile (retour arrière possible)
```

Copier

Interface utilisateur (UI/UX)

Composants de base React Native

Composant	Équivalent Web	Rôle
`<View>`	`<div>`	Conteneur de mise en page
`<Text>`	`<p>` / `<span>`	Afficher du texte
`<TextInput>`	`<input>`	Champ de saisie
`<Pressable>`	`<button>`	Bouton (recommandé sur TouchableOpacity)
`<FlatList>`	`<ul>` virtualisée	Liste longue performante
`<ScrollView>`	`overflow: scroll`	Zone scrollable
`<Image>`	`<img>`	Image
`<SafeAreaView>`	—	Respecte les encoches iPhone

Mise en page avec Flexbox

React Native utilise Flexbox avec `flexDirection: 'column'` par défaut.

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

export default function Carte({ titre, description }: { titre: string; description: string
}) {
  return (
    <View style={styles.carte}>
      <Text style={styles.titre}>{titre}</Text>
      <Text style={styles.description}>{description}</Text>
    </View>
  );
}

const styles = StyleSheet.create({
  carte: {
    backgroundColor: '#fff',
    borderRadius: 12,
    padding: 16,
    marginBottom: 12,
    elevation: 3,
  },
  titre:       { fontSize: 18, fontWeight: 'bold', marginBottom: 8 },
  description: { fontSize: 14, color: '#666' },
});
```

Copier

Design adaptatif

`useWindowDimensions()` est préféré à `Dimensions.get()` car il est **réactif** (rotation, multi-fenêtre).

```
import { useWindowDimensions, View } from 'react-native';

export default function MonComposant() {
  const { width, height } = useWindowDimensions();
  return (
    <View style={{ width: width * 0.9, minHeight: height * 0.5 }} />
  );
}
```

Copier

Mise en place — ordre de création des fichiers

> *Chaque fichier n'importe que des fichiers déjà définis dans les étapes précédentes.*

---

1 — Variables d'environnement

Fichier : `.env` (ne jamais commiter — ajouter dans `.gitignore`)

```
EXPO_PUBLIC_FIREBASE_API_KEY=
EXPO_PUBLIC_FIREBASE_AUTH_DOMAIN=
EXPO_PUBLIC_FIREBASE_PROJECT_ID=
EXPO_PUBLIC_FIREBASE_STORAGE_BUCKET=
EXPO_PUBLIC_FIREBASE_MESSAGING_SENDER_ID=
EXPO_PUBLIC_FIREBASE_APP_ID=
EXPO_PUBLIC_FIREBASE_GOOGLE_WEB_CLIENT_ID=
EXPO_PUBLIC_FIREBASE_GOOGLE_IOS_CLIENT_ID=
EXPO_PUBLIC_FIREBASE_GOOGLE_ANDROID_CLIENT_ID=
EXPO_PUBLIC_API_URL=http://192.168.1.10:3000/api
```

Copier

> *Avec Expo SDK 49+, les variables `EXPO_PUBLIC_*` sont accessibles via `process.env.EXPO_PUBLIC_*` directement dans le code TypeScript.*

---

2 — Configuration Firebase

Fichier : `services/firebase.ts`

```
import AsyncStorage from '@react-native-async-storage/async-storage';
import { getApps, initializeApp } from 'firebase/app';
import { getAuth, initializeAuth } from 'firebase/auth';

// getReactNativePersistence est disponible au runtime RN (Metro résout via la condition
"react-native")
// mais absent des types TypeScript par défaut — on passe par require pour éviter l'erreur
TS
const { getReactNativePersistence } = require('@firebase/auth') as {
  getReactNativePersistence: (storage: typeof AsyncStorage) => any;
};

const firebaseConfig = {
  apiKey:            process.env.EXPO_PUBLIC_FIREBASE_API_KEY,
  authDomain:        process.env.EXPO_PUBLIC_FIREBASE_AUTH_DOMAIN,
  projectId:         process.env.EXPO_PUBLIC_FIREBASE_PROJECT_ID,
  storageBucket:     process.env.EXPO_PUBLIC_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.EXPO_PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
  appId:             process.env.EXPO_PUBLIC_FIREBASE_APP_ID,
};

const app = getApps().length === 0 ? initializeApp(firebaseConfig) : getApps()[0];

// initializeAuth avec AsyncStorage : la session persiste entre les redémarrages de l'app
// Si l'app est déjà initialisée (hot reload), on récupère l'instance existante avec getAuth
export const auth = getApps().length === 1
  ? initializeAuth(app, {
      persistence: getReactNativePersistence(AsyncStorage),
    })
  : getAuth(app);
```

Copier

Activer les providers dans Firebase Console → Authentication → Sign-in method :

- **Email/Mot de passe** → Activer
- **Google** → Activer, renseigner nom du projet et email de support
- **Apple** → Activer (nécessite un compte Apple Developer)

Configurer Google OAuth pour Expo Go (développement) :

En Expo Go, `expo-auth-session` utilise `https://auth.expo.io/@username/slug` comme redirect URI. Il faut l'enregistrer dans Google Cloud Console :

1. Récupérer le username Expo : `npx expo whoami`
2. Google Cloud Console → APIs & Services → Identifiants → OAuth 2.0 Web Client
3. **URI de redirection autorisés** → Ajouter : `https://auth.expo.io/@TON_USERNAME/bpia2-react-tasks`

Fichiers de configuration Google (nécessaires pour les builds natifs) :

Télécharger `GoogleService-Info.plist` (iOS) et `google-services.json` (Android) depuis Firebase Console → Paramètres du projet → tes apps iOS/Android, les placer à la racine du projet, puis déclarer dans `app.json` :

```
{
  "expo": {
    "scheme": "bpia2reacttasks",
    "ios": {
      "bundleIdentifier": "com.bpia2.task",
      "googleServicesFile": "./GoogleService-Info.plist"
    },
    "android": {
      "package": "com.bpia2.task",
      "googleServicesFile": "./google-services.json"
    }
  }
}
```

Copier

> *Ne pas commiter ces fichiers — ajouter `GoogleService-Info.plist` et `google-services.json` dans `.gitignore`.*

Les IDs OAuth à mettre dans `.env` se trouvent dans ces fichiers :

- `EXPO_PUBLIC_FIREBASE_GOOGLE_WEB_CLIENT_ID` → `client_type: 3` dans `google-services.json`
- `EXPO_PUBLIC_FIREBASE_GOOGLE_IOS_CLIENT_ID` → `client_type: 2` avec `ios_info` dans `google-services.json` (ou champ `CLIENT_ID` dans `GoogleService-Info.plist`)
- `EXPO_PUBLIC_FIREBASE_GOOGLE_ANDROID_CLIENT_ID` → `client_type: 3` dans `google-services.json`

---

3 — Contexte d'authentification

Fichier : `context/AuthContext.tsx`

```
import React, { createContext, useState, useEffect, ReactNode } from 'react';
import {
  User,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  signOut,
  onAuthStateChanged,
  GoogleAuthProvider,
  OAuthProvider,
  signInWithCredential,
} from 'firebase/auth';
import * as WebBrowser from 'expo-web-browser';
import * as Google from 'expo-auth-session/providers/google';
import * as AppleAuth from 'expo-apple-authentication';
import { makeRedirectUri } from 'expo-auth-session';
import { auth } from '@/services/firebase';

WebBrowser.maybeCompleteAuthSession();

interface AuthContextType {
  utilisateur: User | null;
  chargement: boolean;
  inscrire: (email: string, motDePasse: string) => Promise<any>;
  connecter: (email: string, motDePasse: string) => Promise<any>;
  deconnecter: () => Promise<void>;
  connecterGoogle: () => Promise<any>;
```

```
  connecterApple: () => Promise<any>;
}

export const AuthContext = createContext<AuthContextType | null>(null);

export function AuthProvider({ children }: { children: ReactNode }) {
  const [utilisateur, setUtilisateur] = useState<User | null>(null);
  const [chargement, setChargement] = useState(true);

  const [, googleResponse, googlePromptAsync] = Google.useAuthRequest({
    webClientId: process.env.EXPO_PUBLIC_FIREBASE_GOOGLE_WEB_CLIENT_ID,
    iosClientId: process.env.EXPO_PUBLIC_FIREBASE_GOOGLE_IOS_CLIENT_ID,
    androidClientId: process.env.EXPO_PUBLIC_FIREBASE_GOOGLE_ANDROID_CLIENT_ID,
    redirectUri: makeRedirectUri({ scheme: 'bpia2reacttasks' }),
  });

  useEffect(() => {
    const desabonner = onAuthStateChanged(auth, (user) => {
      setUtilisateur(user);
      setChargement(false);
    });
    return desabonner;
  }, []);

  useEffect(() => {
    if (googleResponse?.type === 'success') {
      const { id_token } = googleResponse.params;
      const credential = GoogleAuthProvider.credential(id_token);
      signInWithCredential(auth, credential);
    }
  }, [googleResponse]);

  const inscrire = (email: string, motDePasse: string) =>
    createUserWithEmailAndPassword(auth, email, motDePasse);

  const connecter = (email: string, motDePasse: string) =>
    signInWithEmailAndPassword(auth, email, motDePasse);

  const deconnecter = () => signOut(auth);

  const connecterGoogle = () => googlePromptAsync();

  const connecterApple = async () => {
    const credential = await AppleAuth.signInAsync({
      requestedScopes: [
        AppleAuth.AppleAuthenticationScope.FULL_NAME,
        AppleAuth.AppleAuthenticationScope.EMAIL,
      ],
    });
    const provider = new OAuthProvider('apple.com');
    const authCredential = provider.credential({
      idToken: credential.identityToken!,
      rawNonce: (credential as any).nonce ?? undefined,
    });
    return signInWithCredential(auth, authCredential);
  };

  return (
    <AuthContext.Provider
      value={{
        utilisateur,
        chargement,
        inscrire,
        connecter,
        deconnecter,
```

```
        connecterGoogle,
        connecterApple,
      }}>
      {!chargement && children}
    </AuthContext.Provider>
  );
}
```

Copier

---

4 — Hook useAuth

Fichier : `hooks/useAuth.ts`

```
import { useContext } from 'react';
import { AuthContext } from '@/context/AuthContext';

export function useAuth() {
  const context = useContext(AuthContext);
  if (!context) throw new Error('useAuth doit être utilisé dans un AuthProvider');
  return context;
}
```

Copier

---

5 — Instance Axios

Fichier : `services/api.ts`

```
import axios from 'axios';
import { auth } from './firebase';

const api = axios.create({
  baseURL: process.env.EXPO_PUBLIC_API_URL,
  timeout: 10000,
  headers: { 'Content-Type': 'application/json' },
});

// Injecte automatiquement le token Firebase dans chaque requête
api.interceptors.request.use(async (config) => {
  const user = auth.currentUser;
  if (user) {
    const token = await user.getIdToken();
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Déconnecte si l'API rejette le token
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) auth.signOut();
    return Promise.reject(error);
  }
);

export default api;
```

Copier

---

6 — Hook useApi

Fichier : `hooks/useApi.ts`

```
import { useState, useCallback } from 'react';
import api from '@/services/api';

export function useApi() {
  const [loading, setLoading] = useState(false);
  const [erreur,  setErreur]  = useState<string | null>(null);

  const get = useCallback(async <T>(endpoint: string): Promise<T> => {
    setLoading(true); setErreur(null);
    try {
      const { data } = await api.get<T>(endpoint);
      return data;
    } catch (err: any) {
      const msg = err.response?.data?.message || 'Erreur réseau';
      setErreur(msg); throw err;
    } finally { setLoading(false); }
  }, []);

  const post = useCallback(async <T>(endpoint: string, payload: unknown): Promise<T> => {
    setLoading(true); setErreur(null);
    try {
      const { data } = await api.post<T>(endpoint, payload);
      return data;
    } catch (err: any) {
      const msg = err.response?.data?.message || 'Erreur réseau';
      setErreur(msg); throw err;
    } finally { setLoading(false); }
  }, []);

  return { get, post, loading, erreur };
}
```

Copier

---

7 — Layout racine avec redirection auth

Fichier : `app/_layout.tsx`

C'est le point d'entrée de l'application. Il enveloppe tout dans l'`AuthProvider` et redirige vers login ou les onglets selon l'état de connexion.

```
import { DarkTheme, DefaultTheme, ThemeProvider } from '@react-navigation/native';
import { Stack, useRouter, useSegments } from 'expo-router';
import { StatusBar } from 'expo-status-bar';
import { useEffect } from 'react';
import 'react-native-reanimated';

import { useColorScheme } from '@/hooks/use-color-scheme';
import { AuthProvider } from '@/context/AuthContext';
import { useAuth } from '@/hooks/useAuth';

function RootLayoutNav() {
  const colorScheme = useColorScheme();
  const { utilisateur, chargement } = useAuth();
  const segments = useSegments();
  const router = useRouter();

  useEffect(() => {
    if (chargement) return;

    const inAuthGroup = (segments[0] as string) === '(auth)';

    if (!utilisateur && !inAuthGroup) {
      router.replace('/(auth)/login' as any);
    } else if (utilisateur && inAuthGroup) {
      router.replace('/(tabs)' as any);
    }
  }, [utilisateur, chargement, segments]);

  return (
    <ThemeProvider value={colorScheme === 'dark' ? DarkTheme : DefaultTheme}>
      <Stack>
        <Stack.Screen name="(auth)" options={{ headerShown: false }} />
        <Stack.Screen name="(tabs)" options={{ headerShown: false }} />
        <Stack.Screen name="modal" options={{ presentation: 'modal', title: 'Modal' }} />
      </Stack>
      <StatusBar style="auto" />
    </ThemeProvider>
  );
}

export default function RootLayout() {
  return (
    <AuthProvider>
      <RootLayoutNav />
    </AuthProvider>
  );
}
```

Copier

---

8 — Layout du groupe auth

Fichier : `app/(auth)/_layout.tsx`

```
import { Stack } from 'expo-router';

export default function AuthLayout() {
  return (
    <Stack screenOptions={{ headerShown: false }}>
      <Stack.Screen name="login" />
      <Stack.Screen name="register" />
    </Stack>
  );
}
```

Copier

---

9 — Écran de connexion

Fichier : `app/(auth)/login.tsx`

```
import { useState } from 'react';
import {
  View,
  Text,
  TextInput,
  TouchableOpacity,
  StyleSheet,
  ActivityIndicator,
  Alert,
  Platform,
} from 'react-native';
import { Link } from 'expo-router';
import * as AppleAuthentication from 'expo-apple-authentication';
import { useAuth } from '@/hooks/useAuth';

export default function LoginScreen() {
  const { connecter, connecterGoogle, connecterApple, chargement } = useAuth();
  const [email, setEmail] = useState('');
  const [motDePasse, setMotDePasse] = useState('');
  const [loading, setLoading] = useState(false);

  const handleLogin = async () => {
    if (!email || !motDePasse) {
      Alert.alert('Erreur', 'Veuillez remplir tous les champs.');
      return;
    }
    setLoading(true);
    try {
      await connecter(email, motDePasse);
    } catch (err: any) {
      const code = err?.code;
      if (code === 'auth/invalid-credential' || code === 'auth/user-not-found' || code ===
'auth/wrong-password') {
        Alert.alert('Erreur', 'Email ou mot de passe incorrect.');
      } else {
        Alert.alert('Erreur', 'Impossible de se connecter. Réessayez.');
      }
    } finally {
      setLoading(false);
    }
  };

  return (
```

```
    <View style={styles.container}>
      <Text style={styles.title}>Connexion</Text>

      <TextInput
        style={styles.input}
        placeholder="Email"
        placeholderTextColor="#999"
        value={email}
        onChangeText={setEmail}
        keyboardType="email-address"
        autoCapitalize="none"
        autoCorrect={false}
      />

      <TextInput
        style={styles.input}
        placeholder="Mot de passe"
        placeholderTextColor="#999"
        value={motDePasse}
        onChangeText={setMotDePasse}
        secureTextEntry
      />

      <TouchableOpacity
        style={[styles.button, styles.buttonPrimary]}
        onPress={handleLogin}
        disabled={loading}>
        {loading ? (
          <ActivityIndicator color="#fff" />
        ) : (
          <Text style={styles.buttonTextWhite}>Se connecter</Text>
        )}
      </TouchableOpacity>

      <TouchableOpacity
        style={[styles.button, styles.buttonGoogle]}
        onPress={connecterGoogle}
        disabled={chargement}>
        <Text style={styles.buttonTextDark}>Continuer avec Google</Text>
      </TouchableOpacity>

      {Platform.OS === 'ios' && (
        <AppleAuthentication.AppleAuthenticationButton
          buttonType={AppleAuthentication.AppleAuthenticationButtonType.SIGN_IN}
          buttonStyle={AppleAuthentication.AppleAuthenticationButtonStyle.BLACK}
          cornerRadius={8}
          style={styles.appleButton}
          onPress={connecterApple}
        />
      )}

      <Link href={'/(auth)/register' as any} style={styles.link}>
        <Text style={styles.linkText}>Pas encore de compte ? S'inscrire</Text>
      </Link>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    padding: 24,
    backgroundColor: '#fff',
  },
```

```
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    marginBottom: 32,
    textAlign: 'center',
    color: '#1a1a1a',
  },
  input: {
    borderWidth: 1,
    borderColor: '#ddd',
    borderRadius: 8,
    padding: 14,
    marginBottom: 12,
    fontSize: 16,
    color: '#1a1a1a',
    backgroundColor: '#fafafa',
  },
  button: {
    borderRadius: 8,
    padding: 14,
    alignItems: 'center',
    marginBottom: 12,
  },
  buttonPrimary: {
    backgroundColor: '#2563eb',
  },
  buttonGoogle: {
    backgroundColor: '#f1f3f4',
    borderWidth: 1,
    borderColor: '#ddd',
  },
  buttonTextWhite: {
    color: '#fff',
    fontWeight: '600',
    fontSize: 16,
  },
  buttonTextDark: {
    color: '#1a1a1a',
    fontWeight: '600',
    fontSize: 16,
  },
  appleButton: {
    width: '100%',
    height: 48,
    marginBottom: 12,
  },
  link: {
    marginTop: 8,
    alignSelf: 'center',
  },
  linkText: {
    color: '#2563eb',
    fontSize: 15,
  },
});
```

Copier

---

10 — Écran d'inscription

Fichier : `app/(auth)/register.tsx`

```
import { useState } from 'react';
import {
  View,
  Text,
  TextInput,
  TouchableOpacity,
  StyleSheet,
  ActivityIndicator,
  Alert,
} from 'react-native';
import { Link } from 'expo-router';
import { useAuth } from '@/hooks/useAuth';

export default function RegisterScreen() {
  const { inscrire } = useAuth();
  const [email, setEmail] = useState('');
  const [motDePasse, setMotDePasse] = useState('');
  const [confirmation, setConfirmation] = useState('');
  const [loading, setLoading] = useState(false);

  const handleRegister = async () => {
    if (!email || !motDePasse || !confirmation) {
      Alert.alert('Erreur', 'Veuillez remplir tous les champs.');
      return;
    }
    if (motDePasse !== confirmation) {
      Alert.alert('Erreur', 'Les mots de passe ne correspondent pas.');
      return;
    }
    if (motDePasse.length < 6) {
      Alert.alert('Erreur', 'Le mot de passe doit contenir au moins 6 caractères.');
      return;
    }
    setLoading(true);
    try {
      await inscrire(email, motDePasse);
    } catch (err: any) {
      const code = err?.code;
      if (code === 'auth/email-already-in-use') {
        Alert.alert('Erreur', 'Cet email est déjà utilisé.');
      } else if (code === 'auth/invalid-email') {
        Alert.alert('Erreur', 'Adresse email invalide.');
      } else {
        Alert.alert('Erreur', 'Impossible de créer le compte. Réessayez.');
      }
    } finally {
      setLoading(false);
    }
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Créer un compte</Text>

      <TextInput
        style={styles.input}
        placeholder="Email"
        placeholderTextColor="#999"
        value={email}
        onChangeText={setEmail}
        keyboardType="email-address"
        autoCapitalize="none"
        autoCorrect={false}
      />
      <TextInput
```

```
        style={styles.input}
        placeholder="Mot de passe"
        placeholderTextColor="#999"
        value={motDePasse}
        onChangeText={setMotDePasse}
        secureTextEntry
      />
      <TextInput
        style={styles.input}
        placeholder="Confirmer le mot de passe"
        placeholderTextColor="#999"
        value={confirmation}
        onChangeText={setConfirmation}
        secureTextEntry
      />

      <TouchableOpacity
        style={[styles.button, styles.buttonPrimary]}
        onPress={handleRegister}
        disabled={loading}>
        {loading ? (
          <ActivityIndicator color="#fff" />
        ) : (
          <Text style={styles.buttonTextWhite}>S'inscrire</Text>
        )}
      </TouchableOpacity>

      <Link href={'/(auth)/login' as any} style={styles.link}>
        <Text style={styles.linkText}>Déjà un compte ? Se connecter</Text>
      </Link>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    padding: 24,
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    marginBottom: 32,
    textAlign: 'center',
    color: '#1a1a1a',
  },
  input: {
    borderWidth: 1,
    borderColor: '#ddd',
    borderRadius: 8,
    padding: 14,
    marginBottom: 12,
    fontSize: 16,
    color: '#1a1a1a',
    backgroundColor: '#fafafa',
  },
  button: {
    borderRadius: 8,
    padding: 14,
    alignItems: 'center',
    marginBottom: 12,
  },
  buttonPrimary: {
```

```
    backgroundColor: '#2563eb',
  },
  buttonTextWhite: {
    color: '#fff',
    fontWeight: '600',
    fontSize: 16,
  },
  link: {
    marginTop: 8,
    alignSelf: 'center',
  },
  linkText: {
    color: '#2563eb',
    fontSize: 15,
  },
});
```

Copier

---

11 — Layout des onglets

Fichier : `app/(tabs)/_layout.tsx`

```
import { Tabs } from 'expo-router';
import React from 'react';

import { HapticTab } from '@/components/haptic-tab';
import { IconSymbol } from '@/components/ui/icon-symbol';
import { Colors } from '@/constants/theme';
import { useColorScheme } from '@/hooks/use-color-scheme';

export default function TabLayout() {
  const colorScheme = useColorScheme();

  return (
    <Tabs
      screenOptions={{
        tabBarActiveTintColor: Colors[colorScheme ?? 'light'].tint,
        headerShown: false,
        tabBarButton: HapticTab,
      }}>
      <Tabs.Screen
        name="index"
        options={{
          title: 'Accueil',
          tabBarIcon: ({ color }) => <IconSymbol size={28} name="house.fill" color={color}
/>,
        }}
      />
      <Tabs.Screen
        name="explore"
        options={{
          title: 'Explorer',
          tabBarIcon: ({ color }) => <IconSymbol size={28} name="paperplane.fill"
color={color} />,
        }}
      />
      <Tabs.Screen
        name="profile"
        options={{
          title: 'Profil',
          tabBarIcon: ({ color }) => <IconSymbol size={28} name="person.fill" color={color}
/>,
        }}
      />
    </Tabs>
  );
}
```

Copier

---

12 — Écran d'accueil

Fichier : `app/(tabs)/index.tsx`

```
import { useEffect, useState } from 'react';
import { View, Text, StyleSheet, ActivityIndicator } from 'react-native';
import { useAuth } from '@/hooks/useAuth';
import { useApi } from '@/hooks/useApi';

export default function HomeScreen() {
  const { utilisateur } = useAuth();
  const { get, loading, erreur } = useApi();
  const [data, setData] = useState<any>(null);

  useEffect(() => {
    // Exemple : récupérer des données depuis l'API
    // get('/tasks').then(setData).catch(() => {});
  }, []);

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Accueil</Text>
      {utilisateur && (
        <Text style={styles.subtitle}>
          Connecté en tant que {utilisateur.email}
        </Text>
      )}
      {loading && <ActivityIndicator style={{ marginTop: 16 }} />}
      {erreur && <Text style={styles.error}>{erreur}</Text>}
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    paddingTop: 64,
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    marginBottom: 8,
    color: '#1a1a1a',
  },
  subtitle: {
    fontSize: 15,
    color: '#555',
    marginBottom: 16,
  },
  error: {
    color: '#dc2626',
    marginTop: 8,
  },
});
```

Copier

---

13 — Écran profil

Fichier : `app/(tabs)/profile.tsx`

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

```
import { useAuth } from '@/hooks/useAuth';

export default function ProfileScreen() {
  const { utilisateur, deconnecter } = useAuth();

  const handleDeconnexion = async () => {
    Alert.alert('Déconnexion', 'Voulez-vous vous déconnecter ?', [
      { text: 'Annuler', style: 'cancel' },
      {
        text: 'Déconnecter',
        style: 'destructive',
        onPress: async () => {
          try {
            await deconnecter();
          } catch {
            Alert.alert('Erreur', 'Impossible de se déconnecter.');
          }
        },
      },
    ]);
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Profil</Text>

      <View style={styles.card}>
        <Text style={styles.label}>Email</Text>
        <Text style={styles.value}>{utilisateur?.email ?? '-'}</Text>
      </View>

      <View style={styles.card}>
        <Text style={styles.label}>UID Firebase</Text>
        <Text style={styles.value} numberOfLines={1}>{utilisateur?.uid ?? '-'}</Text>
      </View>

      <TouchableOpacity style={styles.button} onPress={handleDeconnexion}>
        <Text style={styles.buttonText}>Se déconnecter</Text>
      </TouchableOpacity>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    paddingTop: 64,
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    marginBottom: 24,
    color: '#1a1a1a',
  },
  card: {
    backgroundColor: '#f8f8f8',
    borderRadius: 8,
    padding: 16,
    marginBottom: 12,
  },
  label: {
    fontSize: 12,
    color: '#888',
```

```
    marginBottom: 4,
    textTransform: 'uppercase',
    letterSpacing: 0.5,
  },
  value: {
    fontSize: 15,
    color: '#1a1a1a',
    fontWeight: '500',
  },
  button: {
    marginTop: 24,
    backgroundColor: '#dc2626',
    borderRadius: 8,
    padding: 14,
    alignItems: 'center',
  },
  buttonText: {
    color: '#fff',
    fontWeight: '600',
    fontSize: 16,
  },
});
```

Copier

Connexion à l'API Express/Node.js

Principe — token Firebase comme passeport

```
App mobile              Firebase              API Express
    │                       │                     │
    │── connecter() ────────►│                     │
    │◄── idToken ────────────│                     │
    │                       │                     │
    │── GET /api/articles ──────────────────────►│
    │   Authorization: Bearer <idToken>           │
    │                                             │── verifyIdToken()
    │                                             │◄ uid, email OK
    │◄──────────────── données JSON ──────────────│
```

Copier

Côté serveur Express

Fichier : `middleware/authFirebase.js` (projet Express — pas dans l'app)

```
const admin = require('firebase-admin');

module.exports = async (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader?.startsWith('Bearer '))
    return res.status(401).json({ message: 'Token manquant' });

  try {
    const token = authHeader.split(' ')[1];
    req.user    = await admin.auth().verifyIdToken(token);
    next();
  } catch {
    return res.status(401).json({ message: 'Token invalide ou expiré' });
  }
};
```

Copier

Fichier : `routes/articles.js` (projet Express)

```
const router       = require('express').Router();
const authFirebase = require('../middleware/authFirebase');

router.get('/',    authFirebase, async (req, res) => {
  const articles = await Article.findAll();
  res.json(articles);
});

router.post('/',   authFirebase, async (req, res) => {
  const article = await Article.create({ ...req.body, auteurUid: req.user.uid });
  res.status(201).json(article);
});

module.exports = router;
```

Copier

Données locales

AsyncStorage — préférences et cache simple

```
import AsyncStorage from '@react-native-async-storage/async-storage';

await AsyncStorage.setItem('theme', JSON.stringify({ mode: 'dark' }));
const raw   = await AsyncStorage.getItem('theme');
const theme = raw ? JSON.parse(raw) : null;
await AsyncStorage.removeItem('theme');
```

Copier

Expo SQLite — données structurées

> *`openDatabase()` + `db.transaction()` sont **dépréciés depuis Expo SDK 50**. Utiliser uniquement la nouvelle API async.*

```
import * as SQLite from 'expo-sqlite';

const db = await SQLite.openDatabaseAsync('appli.db');

await db.execAsync(`
  CREATE TABLE IF NOT EXISTS articles (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    titre TEXT NOT NULL,
    contenu TEXT
  );
`);

await db.runAsync('INSERT INTO articles (titre) VALUES (?)', ['Mon titre']);
const articles = await db.getAllAsync<{ id: number; titre: string }>('SELECT * FROM
articles');
const one      = await db.getFirstAsync<{ id: number; titre: string }>('SELECT * FROM
articles WHERE id = ?', [1]);
```

Copier

Tests

Tests unitaires avec Jest

Fichier : `__tests__/utils.test.ts`

```
import { validerEmail } from '@/utils';

describe('validerEmail', () => {
  test('accepte un email valide',  () => expect(validerEmail('a@b.com')).toBe(true));
  test('refuse un email sans @',   () => expect(validerEmail('abcom')).toBe(false));
});
```

Copier

Tests de composants avec React Native Testing Library

```
npm install --save-dev @testing-library/react-native
```

Copier

Fichier : `__tests__/login.test.tsx`

```
import React                                  from 'react';
import { render, fireEvent, waitFor }         from '@testing-library/react-native';
import LoginScreen                            from '@/app/(auth)/login';
import { AuthContext }                        from '@/context/AuthContext';

const mockConnecter = jest.fn();

const wrapper = ({ children }: { children: React.ReactNode }) => (
  <AuthContext.Provider value={{
    utilisateur: null, chargement: false,
    connecter: mockConnecter,
    inscrire: jest.fn(), deconnecter: jest.fn(),
    connecterGoogle: jest.fn(), connecterApple: jest.fn(),
  }}>
    {children}
  </AuthContext.Provider>
);

test('n'appelle pas connecter si les champs sont vides', async () => {
  const { getByText } = render(<LoginScreen />, { wrapper });
  fireEvent.press(getByText('Se connecter'));
  await waitFor(() => expect(mockConnecter).not.toHaveBeenCalled());
});
```

Copier

Déploiement

Build avec EAS

```
npm install -g eas-cli
eas login
eas build:configure

eas build --platform android --profile production   # → .aab pour le Play Store
eas build --platform ios     --profile production   # → .ipa pour l'App Store
```

Copier

Google Play Store

1. Compte **Google Play Developer** (frais uniques)
2. Uploader le `.aab` → configurer les permissions → révision (1 à 3 jours)

Apple App Store

1. Compte **Apple Developer Program** (abonnement annuel)
2. Activer **Sign in with Apple** dans App Store Connect → Capabilities
3. Uploader le `.ipa` → révision Apple (24 à 48 heures)

Récapitulatif — ordre de création des fichiers

Ordre	Fichier	Dépend de
1	`.env`	—
2	`services/firebase.ts`	`.env`
3	`context/AuthContext.tsx`	`firebase.ts`
4	`hooks/useAuth.ts`	`AuthContext.tsx`
5	`services/api.ts`	`firebase.ts`
6	`hooks/useApi.ts`	`api.ts`
7	`app/_layout.tsx`	`AuthContext.tsx` + `useAuth.ts`
8	`app/(auth)/_layout.tsx`	—
9	`app/(auth)/login.tsx`	`useAuth.ts`
10	`app/(auth)/register.tsx`	`useAuth.ts`
11	`app/(tabs)/_layout.tsx`	—
12	`app/(tabs)/index.tsx`	`useAuth.ts` + `useApi.ts`
13	`app/(tabs)/profile.tsx`	`useAuth.ts`