



# React Native et Expo

Une formation complète sur le développement d'applications mobiles avec  
React Native et Expo.

Appuyez sur espace pour la page suivante →

# SOMMAIRE

Voici le sommaire de cette formation sur React Native et Expo:

 Introduction à React Native et Expo

 Configuration de l'environnement de développement

 Bases de React Native

 Composants natifs et styling

 Gestion de l'état et navigation

 Intégration d'API et gestion des données

 Utilisation de la caméra et des médias

 Animations et gestes

 Publication de l'application

 Bonnes pratiques et optimisation

 Code source du projet

# Introduction à React Native

- **Qu'est-ce que React Native ?**

- Framework pour développer des applications mobiles natives sur ios et android
- Utilise React et JavaScript pour créer des interfaces utilisateur

- **Avantages de React Native**

- Développement multiplateforme (iOS et Android)
- Performance proche du natif (avec quelques ajustements nécessaires pour certains modules via des bridges etc)
- Large communauté et écosystème riche
- Basé sur React, courbe d'apprentissage très facile au début.

# Introduction à Expo

- **Qu'est-ce qu'Expo ?**
  - Ensemble d'outils et de services pour React Native
  - Simplifie le développement et le déploiement énormément !
- **Pourquoi utiliser Expo ?**
  - Configuration rapide et facile
  - Accès aux API natives sans configuration complexe
  - Mise à jour Over-The-Air (OTA)

# Expo Go

- **Présentation d'Expo Go**

- Application mobile gratuite disponible sur iOS et Android
- Permet de tester les applications en développement
- Pas besoin de compiler ou déployer pour tester

- **Fonctionnement**

- Scanner un QR code depuis l'application Expo Go
- Visualisation instantanée des modifications en temps réel
- Accès aux APIs natives d'Expo directement

- **Avantages**

- Développement et tests rapides
- Partage facile avec l'équipe et les clients
- Pas besoin d'Apple Developer Account pour tester sur iOS

# Limitations d'Expo Go

- **Restrictions**

- Ne peut pas utiliser de modules natifs personnalisés
- Certaines fonctionnalités natives avancées non disponibles
- Non utilisable en production

- **Quand passer au "bare workflow"**

- Besoin de modules natifs personnalisés
- Nécessité d'optimisations natives
- Publication sur les stores

Pensez à installer dès maintenant expo go sur votre iphone ou sur votre android.

# Disclaimer concernant React Native

- **React Native n'est pas toujours nécessaire**
  - Pour des applications simples, des solutions web peuvent suffire
  - PWA (Progressive Web Apps) comme alternative viable
  - Solutions natives pures parfois plus adaptées selon les besoins

# Disclaimer concernant React Native (suite)

- **React Native n'est pas toujours suffisant**
  - Certaines fonctionnalités nécessitent du code natif (Kotlin/Swift)
  - Besoin de compétences natives pour des fonctionnalités complexes
  - Exemples :
    - Modules natifs personnalisés (serveur torrent dans l'app etc)
    - Optimisations de performance spécifiques
    - Intégrations matérielles avancées (il faut comprendre android studio/xcode , les manipuler)



# Disclaimer concernant React Native (suite)

- **Compétences complémentaires requises**
  - Connaissance basique d'Android (Kotlin/Java) peut être nécessaire
  - Notions d'iOS (Swift/Objective-C) parfois indispensables
  - Compréhension de l'architecture native mobile

# Disclaimer concernant Expo

- **Limitations de personnalisation**

- Accès limité aux modules natifs personnalisés
- Impossibilité d'utiliser certaines bibliothèques natives non supportées
- Configuration native restreinte

# Disclaimer concernant Expo (suite)

- **Taille des applications**

- Applications plus volumineuses car inclusion de nombreuses dépendances
- Impact sur le temps de téléchargement initial

- **Dépendance à Expo**

- Mises à jour forcées de l'écosystème Expo
- Difficultés potentielles pour "éjecter" le projet plus tard
- Dépendance aux serveurs Expo pour certaines fonctionnalités (eas build et je vais vous en parler)

# Disclaimer concernant Expo (suite)

- **Performance**

- Légère baisse de performance par rapport à React Native pur
- Temps de compilation plus longs
- Consommation mémoire plus importante

# Donc comment faire ?

Donc dans ce cas , il faudra ejecter expo avec dans votre terminal

```
expo eject
```

# Metro : Le bundler de React Native

- **Qu'est-ce que Metro ?**
  - Bundler JavaScript par défaut pour React Native
  - Compile et empaquette le code source
  - Gère les assets (images, polices, etc.)

# Metro : Fonctionnalités

- **Fonctionnalités principales**
  - Hot Reloading (HMR)
  - Source maps pour le debugging
  - Optimisation du bundle
  - Gestion des dépendances

# Metro : Architecture

- **Architecture**

- Serveur de développement intégré
- Cache intelligent pour des builds rapides
- Support des transformations personnalisées
- Compatible avec les plugins Babel



# Metro : Avantages

- **Avantages**
  - Rapide et optimisé pour mobile
  - Configuration simple
  - Intégration native avec React Native
  - Support du Fast Refresh

# A quoi ressemble Metro ?

- **Interface de Metro**

- Console avec informations de build
- Affichage des erreurs et warnings
- QR code pour connexion des appareils
- Menu avec options de développement

- **Fonctionnalités visibles**

- Logs en temps réel
- État du bundle et du HMR
- Liste des appareils connectés
- Statistiques de performance

# Des concurrents à Metro ?

- **Webpack**

- Plus généraliste
- Écosystème plus large
- Configuration plus complexe
- Moins optimisé pour mobile

- **Vite**

- Plus récent et moderne
- Très rapide en développement
- Support expérimental de React Native
- Communauté grandissante

# Mais aussi

- **esbuild**
  - Ultra rapide
  - Écrit en Go
  - Support limité pour React Native
  - Utilisé comme dépendance par d'autres bundlers

# Installation d'Expo CLI

```
# Installation globale d'Expo CLI  
npm install -g expo-cli
```

Ça c'est la façon globale.

Cependant, nous devrions plutôt utiliser npx qui permet de télécharger la dernière dépendance, de l'exécuter et de ne pas la stocker sur notre pc , ce qui permet d'avoir toujours la commande à jour sans prise de tête

de toute façon si vous utilisez cette commande, vous aurez un jolie deprecated et le expo init ne marchera pas, donc si vous lisez un tuto en ligne pensez y

Donc lisons la doc officiel et lançons plutot la commande :

```
npx create-expo-app@latest leNomDeVotreApp
```

[Revenir au sommaire](#)

# Création du projet

(l'ancienne façon de faire)

```
# Création d'un nouveau projet  
expo init TinderLikeApp
```

[Revenir au sommaire](#)

# Navigation et lancement

```
# Navigation dans le dossier  
cd TinderLikeApp  
  
# Lancement de l'application  
expo start
```



# Test de l'application

Vous pouvez tester rapidement le développement moderne :

- Installer expo sur android/ios (play store ou app store)
- Scanner le QRcode dans votre terminal (Metro)
- Ça va automatiquement ouvrir expo sur votre mobile et pré compiler

# Hot Module Reload

Vous pouvez voir votre application en temps réel, et à chaque changement le HMR (Hot module reload) se lance, vous êtes donc toujours à jour !

# Structure du composant Hello World

```
// Structure du composant
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Hello World</Text>
    </View>
  );
}
```

# Styles de l'application

```
// Styles du composant
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  text: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
});
```

# Félicitations !

Vous avez créé votre première application React Native avec Expo.

[Revenir au sommaire](#)

# Configuration de l'environnement (2024/2025)

## Installation de Node.js et des outils de base

```
# Installation de Node.js via nvm
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
nvm install node

# Installation des outils globaux
npm install -g typescript
```

# Création d'un projet Expo moderne

```
# Création du projet avec le template TypeScript  
npx create-expo-app@latest -t expo-template-typescript  
  
# OU avec le template Tabs (recommandé pour une app type Tinder)  
npx create-expo-app@latest -t tabs
```

# Installation des dépendances essentielles

```
npx expo install nativewind@3.0.0
npm install tailwindcss@3.3.2 --save-dev

# Animations et gestes
npx expo install react-native-reanimated
npx expo install react-native-gesture-handler

# UI et effets
npx expo install expo-linear-gradient
npm install react-native-deck-swiper
```



# Configuration de NativeWind

```
// tailwind.config.js
module.exports = {
  content: [
    './app/**/*..{js,jsx,ts,tsx}',
    './components/**/*..{js,jsx,ts,tsx}'
  ],
  presets: [require("nativewind/preset")],
  theme: {
    extend: {
      colors: {
        primary: {
          500: '#FF6B6B',
          600: '#FF5252',
        }
      }
    }
  }
}
```

# Configuration de Babel pour Reanimated

```
// babel.config.js
module.exports = function(api) {
  api.cache(true);
  return {
    presets: ['babel-preset-expo'],
    plugins: [
      'nativewind/babel',
      'react-native-reanimated/plugin',
    ],
  };
};
```

# Configuration de Metro

```
// metro.config.js
const { getDefaultConfig } = require('expo/metro-config');
const { withNativeWind } = require('nativewind/metro');

const config = getDefaultConfig(__dirname);

module.exports = withNativeWind(config, {
  input: './global.css',
});
```

# Création des fichiers de style

```
/* global.css */  
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

# Configuration TypeScript

```
// tsconfig.json
{
  "extends": "expo/tsconfig.base",
  "compilerOptions": {
    "strict": true,
    "paths": {
      "@/*": ["./*"]
    }
  },
  "include": [
    "**/*.ts",
    "**/*.tsx",
    ".expo/types/**/*.ts",
    "expo-env.d.ts"
  ]
}
```

# Structure du projet recommandée (2024/2025)

```
MonProjet/  
├── app/                # Routing avec Expo Router  
├── components/         # Composants réutilisables  
├── hooks/              # Custom hooks  
├── services/           # Services (API, etc.)  
├── types/              # Types TypeScript  
├── utils/              # Utilitaires  
├── assets/             # Images, fonts  
├── global.css          # Styles Tailwind  
├── tailwind.config.js  # Config Tailwind  
├── babel.config.js     # Config Babel  
├── metro.config.js     # Config Metro  
└── tsconfig.json       # Config TypeScript
```

Cette configuration moderne permet de développer des applications React Native performantes avec un excellent DX (Developer Experience).

# Installation d'Expo CLI

```
# Installation globale d'Expo CLI  
npm install -g expo-cli  
# deprecated !
```

utilisez plutôt :

```
npx expo `laCommande`
```

pour créer une app :

```
npx create-expo-app@latest
```

# Configuration de l'éditeur

Je vous conseille sur vs code ou cursor ou autre d'avoir , déjà installer les plugins pour prettier (et ses nouveaux concurrents), eslint etc



# Configuration de l'éditeur (suite)

```
// settings.json - Partie 2
{
  "prettier.semi": true,
  "prettier.singleQuote": true,
  "prettier.printWidth": 80,
  "prettier.tabWidth": 2
}
```

# Installation des dépendances

## Dépendances de base

Elles sont déjà installer avec expo , seulement si vous faite un projet sans expo.

```
npm install @react-navigation/native @react-navigation/stack
```

# Résultat

Votre environnement de développement est maintenant configuré pour React Native et Expo.

[Revenir au sommaire](#)

# Configuration de l'environnement de développement (suite)

- **Éditeur de code**

- Recommandation : Visual Studio Code avec les extensions React Native
- Autres options : WebStorm, Atom, Sublime Text

- **Simulateurs et émulateurs**

- iOS : Xcode (Mac uniquement)
- Android : Android Studio avec AVD Manager

- **Application Expo Go**

- Téléchargez sur votre appareil mobile depuis l'App Store ou Google Play
- Permet de tester l'application sur un appareil physique

# Exercice : Configuration de votre projet TinderLikeApp

1. Ouvrez votre projet TinderLikeApp dans Visual Studio Code

# Exercice : Configuration de votre projet TinderLikeApp (suite)

4. Initialisez un dépôt Git :

```
git init
git add .
git commit -m "Initial commit"
```

5. Lancez l'application avec `expo start` et testez-la sur :

- Un simulateur iOS (Mac uniquement)
- Un émulateur Android
- Votre appareil mobile avec Expo Go

# Structure du projet Expo moderne (2024/2025)

Voici la structure actuelle d'un projet créé avec `create-expo-app` :

[Revenir au sommaire](#)

```
MonProjet/
├── app/                                # Dossier principal avec le nouveau système de routing
│   ├── _layout.tsx                   # Layout principal de l'application
│   ├── index.tsx                     # Page d'accueil
│   ├── (tabs)/                       # Groupe de routes pour les tabs
│   │   ├── home.tsx                 # Tab Home
│   │   ├── profile.tsx              # Tab Profile
│   │   └── _layout.tsx              # Layout spécifique aux tabs
│   └── (auth)/                       # Groupe de routes pour l'authentification
│       ├── login.tsx                # Page de login
│       └── register.tsx             # Page d'inscription
├── assets/                           # Images, fonts et autres ressources
├── components/                       # Composants réutilisables
├── constants/                       # Constants (colors, dimensions, etc.)
└── hooks/                           # Custom hooks
```



# Points clés de la nouvelle structure

- **Routing basé sur les fichiers :**
  - Plus besoin de React Navigation explicite
  - Les fichiers dans `app/` définissent automatiquement les routes
  - Système similaire à Next.js

# Organisation des routes

- **Groupes de routes :**
  - `(tabs)` : Routes avec navigation par tabs
  - `(auth)` : Routes pour l'authentification
  - Les parenthèses indiquent des groupes logiques
- **Fichiers spéciaux :**
  - `_layout.tsx` : Définit le layout d'un groupe de routes
  - `index.tsx` : Page par défaut d'un dossier

# Avantages de la nouvelle structure

- Organisation plus intuitive
- Routing automatique
- Support TypeScript par défaut
- Meilleure séparation des préoccupations
- Plus facile à maintenir et à faire évoluer

Cette nouvelle structure est inspirée des frameworks web modernes comme Next.js et offre une meilleure expérience de développement.

# Bases de React Native

- **Composants React Native**

- View : conteneur générique (équivalent à une `div` en HTML)
- Text : conteneur pour le texte (équivalent à `p` , `h1` , `span` en HTML)
- Image : affichage d'images (équivalent à `img` en HTML)
- TextInput : champ de saisie (équivalent à `input` en HTML)
- TouchableOpacity : zone cliquable (équivalent à `button` en HTML)

# Bases de React Native (suite)

- **Composants React Native (suite)**

- ScrollView : conteneur avec défilement (équivalent à une `<div>` avec `overflow-y: scroll` en CSS)
- FlatList : liste optimisée pour de grandes quantités de données (équivalent à une boucle `.map()` sur un tableau en React Web, mais avec virtualisation)

- **JSX et styles**

- Utilisation de JSX ou TSX pour décrire l'interface utilisateur
- Styles inspirés de CSS, mais avec des différences clés, alors ça je vous le résume à l'oral
- `StyleSheet.create` pour définir des styles ou d'autres packages à importer soi-même

# Bases de React Native (suite)

## Props et State

### Props – Structure

```
// Composant parent  
<UserProfile name="John" age={25} />
```

# Props - Utilisation

```
// Composant enfant UserProfile
const UserProfile = (props) => {
  return (
    <Text>
      Nom:
      {props.name}
      , Age:
      {props.age}
    </Text>
  )
}
```

# State - Structure

```
// Gestion de l'état avec useState
const Counter = () => {
  const [count, setCount] = useState(0)

  return (
    <TouchableOpacity
      onPress={() => setCount(count + 1)}
      {/* quand j'appuie, l'état d'avant de la variable count se met à jour donc 0 + 1 , au prochain ap
    >
    <Text>
      Compteur:
      {count}
      {/* count se met à jour automatiquement dans la vue */}
    </Text>
  </TouchableOpacity>
```



Si vous n'avez aucune expérience avec ce concept :

Ce sont des variables en fait, juste React relis plusieurs fois la page à chaque changement, donc ça voudrais dire que si il relis il reverrais :

let count = 0

donc il ferai toujours  $0 + 1$ , puis au prochain tour  $0 + 1$  ce qui n'as aucun sens quand on veut incrémenter un bouton

# Exercice : Création d'un profil utilisateur

## Structure du composant - Partie 1

```
// UserProfile.js - Structure
import React from 'react';
import { View, Text, Image, StyleSheet } from 'react-native';

const UserProfile = ({ name, bio, imageUrl }) => {
  return (
    <View style={styles.container}>
      <Image source={{ uri: imageUrl }} style={styles.image} />
      <Text style={styles.name}>{name}</Text>
      <Text style={styles.bio}>{bio}</Text>
    </View>
  );
};
```

## Structure du composant – Partie 2

```
// UserProfile.js – Styles
const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
    padding: 20,
  },
  image: {
    width: 150,
    height: 150,
    borderRadius: 75,
    marginBottom: 20,
  },
  name: {
    fontSize: 24,
    fontWeight: 'bold',
  },
});
```

# Utilisation du composant

```
// App.js
import React from 'react';
import { View, StyleSheet } from 'react-native';
import UserProfile from './components/UserProfile';

export default function App() {
  return (
    <View style={styles.container}>
      <UserProfile
        name="John Doe"
        bio="Passionné de voyages et de photographie."
        imageUrl="https://randomuser.me/api/portraits/men/1.jpg"
      />
    </View>
  );
}
```

# Styles de l'application

```
// App.js - Styles
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
});
```

# Résultat de l'exercice

Cet exercice vous permet de pratiquer la création de composants, l'utilisation de props, et le styling en React Native.

[Revenir au sommaire](#)

# Composants Natifs et Styling (2024/2025)

## Styling avec NativeWind

```
// components/Card.tsx
export function Card({ children }) {
  return (
    <View className="bg-white rounded-2xl shadow-lg p-4 m-2">
      {children}
    </View>
  );
}
```

# Composants de base stylisés

```
// components/StyledComponents.tsx
export function Button({ onPress, children }) {
  return (
    <TouchableOpacity
      onPress={onPress}
      className="bg-primary-500 px-4 py-2 rounded-lg active:bg-primary-600"
    >
      <Text className="text-white font-medium text-center">
        {children}
      </Text>
    </TouchableOpacity>
  );
}

export function Input({ placeholder, ...props }) {
```



# Composants Safe Area et Platform

```
// app/_layout.tsx
import { Stack } from 'expo-router';
import { SafeAreaProvider } from 'react-native-safe-area-context';

export default function Layout() {
  return (
    <SafeAreaProvider>
      <Stack
        screenOptions={{
          headerStyle: {
            backgroundColor: '#fff',
          },
          headerShadowVisible: false,
          headerBlurEffect: 'regular',
        }}
      />
    </SafeAreaProvider>
  );
}
```

# Gestion des images avec Expo Image

```
// components/ProfileImage.tsx
import { Image } from 'expo-image';

export function ProfileImage({ uri }) {
  return (
    <Image
      source={uri}
      className="w-32 h-32 rounded-full"
      transition={1000}
      contentFit="cover"
      placeholder={blurhash}
      cachePolicy="memory-disk"
    />
  );
}
```

# Animations avec Reanimated

```
// components/AnimatedCard.tsx
import Animated, {
  useAnimatedStyle,
  withSpring
} from 'react-native-reanimated';

export function AnimatedCard({ isVisible }) {
  const animatedStyles = useAnimatedStyle(() => {
    return {
      transform: [
        {
          scale: withSpring(isVisible ? 1 : 0.8),
        },
      ],
      opacity: withSpring(isVisible ? 1 : 0),
    };
  });
```

# Gestes avec Reanimated

```
// components/SwipeableCard.tsx
import { Gesture, GestureDetector } from 'react-native-gesture-handler';

export function SwipeableCard() {
  const gesture = Gesture.Pan()
    .onUpdate((event) => {
      // Logique de mise à jour
    })
    .onEnd((event) => {
      // Logique de fin de geste
    });

  return (
    <GestureDetector gesture={gesture}>
      <Animated.View className="w-full aspect-[3/4] bg-white rounded-2xl">
```

# Composants spécifiques à la plateforme

```
// components/PlatformSpecific.tsx
import { Platform } from 'react-native';

export function StatusBarHeight() {
  return (
    <View
      className={Platform.select({
        ios: 'h-11',
        android: 'h-7',
        default: 'h-0'
      })}
    />
  );
}
```

# Exercice : Création d'une carte de profil

Créez une carte de profil pour l'application de rencontre avec :

1. Styling avec NativeWind
2. Animations fluides avec Reanimated
3. Gestion optimisée des images
4. Gestes de swipe

# Solution de l'exercice

```
// components/ProfileCard.tsx
import { Image } from 'expo-image';
import Animated, {
  useAnimatedStyle,
  withSpring
} from 'react-native-reanimated';
import { Gesture, GestureDetector } from 'react-native-gesture-handler';

export function ProfileCard({ profile, onSwipe }) {
  const gesture = Gesture.Pan()
    .onUpdate((event) => {
      // Logique de swipe
    })
    .onEnd((event) => {
      if (Math.abs(event.velocityX) > 500) {
```

Cette version modernisée utilise les dernières pratiques et composants disponibles dans l'écosystème React Native/Expo en 2024/2025.

# Gestion de l'état et Navigation (2024/2025)

## État Local avec Hooks

```
// Exemple basique de useState
const [count, setCount] = useState(0);

// État avec objet
const [user, setUser] = useState<User>({
  name: '',
  age: 0
});
```



# Gestion d'état complexe

```
// useReducer pour état complexe
const [state, dispatch] = useReducer(reducer, {
  matches: [],
  likes: 0,
  messages: []
});

function reducer(state, action) {
  switch(action.type) {
    case 'ADD_MATCH':
      return { ...state, matches: [...state.matches, action.payload] };
    case 'INCREMENT_LIKES':
      return { ...state, likes: state.likes + 1 };
    default:
      return state;
  }
}
```

# Navigation Moderne avec Expo Router

## Structure de base (2024/2025)

```
app/  
├── _layout.tsx  
├── index.tsx  
├── (tabs)/  
│   ├── _layout.tsx  
│   ├── home.tsx  
│   ├── matches.tsx  
│   └── profile.tsx  
└── (auth)/  
    ├── login.tsx  
    └── register.tsx
```

# Layout Principal

```
// app/_layout.tsx
import { Stack } from 'expo-router';

export default function RootLayout() {
  return (
    <Stack>
      <Stack.Screen
        name="(tabs)"
        options={{ headerShown: false }}
      />
      <Stack.Screen
        name="(auth)"
        options={{ headerShown: false }}
      />
    </Stack>
  );
}
```

# Navigation par Tabs

```
// app/(tabs)/_layout.tsx
import { Tabs } from 'expo-router';
import { Home, Heart, User } from 'lucide-react-native';

export default function TabsLayout() {
  return (
    <Tabs screenOptions={{
      tabBarActiveTintColor: '#FF6B6B',
    }}>
      <Tabs.Screen
        name="home"
        options={{
          title: 'Découvrir',
          tabBarIcon: ({ color }) => (
            <Home size={24} color={color} />

```

# Navigation vers un profil

```
// app/(tabs)/home.tsx
import { Link } from 'expo-router';

export default function Home() {
  return (
    <View className="flex-1 p-4">
      <Link href="/profile/123" asChild>
        <TouchableOpacity className="bg-primary p-4 rounded-lg">
          <Text className="text-white">Voir Profil</Text>
        </TouchableOpacity>
      </Link>
    </View>
  );
}
```

# Pages dynamiques

```
// app/profile/[id].tsx
import { useLocalSearchParams } from 'expo-router';

export default function Profile() {
  const { id } = useLocalSearchParams();

  return (
    <View className="flex-1 p-4">
      <Text className="text-xl">Profil {id}</Text>
    </View>
  );
}
```

# Exercice : Application de rencontre

Créez une application de rencontre avec :

1. Navigation par tabs (Découvrir, Matches, Profil)
2. Système de likes avec useReducer
3. Pages de profil dynamiques
4. Authentification protégée

# Structure de l'exercice

```
// app/(tabs)/home.tsx
export default function Home() {
  const [profiles, dispatch] = useReducer(profilesReducer, []);

  const handleLike = (profile) => {
    dispatch({ type: 'LIKE_PROFILE', payload: profile });
  };

  return (
    <View className="flex-1">
      <Swiper
        cards={profiles}
        onSwipedRight={({cardIndex}) => {
          handleLike(profiles[cardIndex]);
        }}
      >
```



# Authentification sécurisée

```
// app/_layout.tsx
import { useAuth } from '../hooks/useAuth';

export default function RootLayout() {
  const { isAuthenticated } = useAuth();

  return (
    <Stack>
      {isAuthenticated ? (
        <Stack.Screen
          name="(tabs)"
          options={{ headerShown: false }}
        />
      ) : (
        <Stack.Screen
```

Cette version modernisée reflète les pratiques actuelles de 2024/2025 avec Expo Router, tout en gardant les concepts fondamentaux de la gestion d'état.

# Implémentation du Swiper

## Composant HomeScreen complet

```
// app/(tabs)/home.tsx
import React from "react";
import { View, SafeAreaView } from "react-native";
import Swiper from "react-native-deck-swiper";
import ProfileCard from "../components/ProfileCard";
import { User, users } from "../data/users";
import { X, Heart } from "lucide-react-native";
import { Button } from "~/components/ui/button";

export default function HomeScreen() {
  const swiperRef = React.useRef<Swiper<User>>>(null);

  const handleLike = (cardIndex: number) => {
    console.log(`Liked ${users[cardIndex].name}`);
  };
}
```

# Explications des fonctionnalités clés

## 1. Configuration du Swiper

```
// Props importantes du Swiper
<Swiper
  stackSize={10}           // Nombre de cartes visibles dans la pile
  stackScale={10}          // Échelle de réduction pour les cartes empilées
  stackSeparation={14}    // Espacement vertical entre les cartes
  animateOverlayLabelsOpacity // Animation des labels OUI/NON
  animateCardOpacity       // Fondu des cartes
  disableTopSwipe           // Désactive le swipe vers le haut
  disableBottomSwipe       // Désactive le swipe vers le bas
/>
```

## 2. Gestion des labels de swipe

```
// Configuration des labels OUI/NON
overlayLabels={
  left: {
    title: "NON",
    style: {
      label: {
        backgroundColor: "red",
        // ... styles du label
      },
      wrapper: {
        // ... styles du wrapper
      },
    },
  },
  right: {
```

### 3. Boutons physiques avec refs

```
// Utilisation de la ref pour contrôler le Swiper
const swiperRef = React.useRef<Swiper<User>>(null);

// Déclenchement manuel des swipes
<Button
  onPress={() => {
    if (swiperRef.current) {
      swiperRef.current.swipeLeft();
    }
  }}
>
  <X />
</Button>
```

## 4. Gestion des événements de swipe

```
// Handlers pour les swipes
const handleLike = (cardIndex: number) => {
  console.log(`Liked ${users[cardIndex].name}`);
  // Ici vous pouvez ajouter votre logique de match
  // Par exemple : appel API, mise à jour du state, etc.
};

const handleDislike = (cardIndex: number) => {
  console.log(`Disliked ${users[cardIndex].name}`);
  // Logique de rejet
};
```

Cette implémentation moderne combine :

- Styling avec NativeWind
- Gestion d'état avec TypeScript
- Animations fluides
- Interface utilisateur intuitive
- Gestion des gestes tactiles

Le tout dans un composant réutilisable et maintenable.

[Revenir au sommaire](#)

# Intégration API pour l'application de rencontre

## Types et configuration

```
// types/profile.ts
export interface Profile {
  id: string;
  name: {
    first: string;
    last: string;
  };
  picture: {
    large: string;
    medium: string;
  };
  dob: {
    age: number;
  };
  location: {
```

# Hook personnalisé pour les profils

```
// hooks/useProfiles.ts
import { useQuery } from '@tanstack/react-query';

export function useProfiles(count: number = 10) {
  return useQuery({
    queryKey: ['profiles'],
    queryFn: async (): Promise<Profile[]> => {
      const response = await fetch(
        `https://randomuser.me/api/?results=${count}&inc=name,picture,dob,location&nat=fr`
      );
      if (!response.ok) {
        throw new Error('Erreur lors du chargement des profils');
      }
    }
  });
}
```



# Utilisation dans le Swiper

```
// app/(tabs)/home.tsx
import { useProfiles } from '../../hooks/useProfiles';

export default function HomeScreen() {
  const swiperRef = React.useRef<Swiper<Profile>>>(null);
  const { data: profiles, isLoading, error } = useProfiles(20);

  if (isLoading) {
    return <LoadingView />;
  }

  if (error) {
    return <ErrorView error={error} />;
  }
}
```

# Gestion du cache et mise à jour

```
// hooks/useUpdateProfile.ts
import { useMutation, useQueryClient } from '@tanstack/react-query';

export function useUpdateProfile() {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async (profileId: string) => {
      // Simuler un délai réseau
      await new Promise(resolve => setTimeout(resolve, 300));
      return profileId;
    },
    onSuccess: (profileId) => {
      // Mettre à jour le cache en retirant le profil
      queryClient.setQueryData(['profiles'], (oldData: Profile[]) =>
```

# Composants d'UI optimisés

```
// components/LoadingView.tsx
export function LoadingView() {
  return (
    <View className="flex-1 items-center justify-center">
      <ActivityIndicator size="large" color="#FF6B6B" />
      <Text className="mt-4 text-gray-600">
        Recherche de profils...
      </Text>
    </View>
  );
}
```

```
// components/ErrorView.tsx
export function ErrorView({ error, onRetry }: { error: Error; onRetry: () => void }) {
  return (
```

Cette implémentation simplifiée :

- Utilise uniquement React Query avec fetch
- Gère le cache et les mises à jour optimistes
- Recharge automatiquement quand il reste peu de profils
- Inclut une gestion d'erreur et de chargement élégante
- Reste performante grâce au staleTime et au cache

Le tout sans dépendance supplémentaire autre que React Query.

# Utilisation de la caméra et des médias

## Configuration des permissions

```
// Configuration des permissions caméra
const { status } = await Camera.requestPermissionsAsync();
if (status === 'granted') {
  // On peut utiliser la caméra
}
```

# Capture de photos

```
// Composant de capture photo
<Camera ref={cameraRef}>
  <Button title="Photo" onPress={async () => {
    const photo = await cameraRef.current.takePictureAsync();
    console.log(photo.uri);
  }} />
</Camera>
```

# Sélection d'images

```
// Sélection depuis la galerie
const result = await ImagePicker.launchImageLibraryAsync({
  mediaTypes: 'Images',
  allowsEditing: true,
});
if (!result.cancelled) {
  console.log(result.uri);
}
```

# Téléchargement d'images

```
// Configuration du FormData
const formData = new FormData();
formData.append('photo', {
  uri: imageUrl,
  type: 'image/jpeg',
  name: 'photo.jpg',
});

// Envoi au serveur
await fetch('https://monapi.com/upload', {
  method: 'POST',
  body: formData
});
```



# Utilisation de la caméra et des médias (suite)

- Manipulation d'images avec Expo ImageManipulator

- Redimensionnement et rotation d'images

```
// Comme canvas.getContext('2d') en web
const manipResult = await ImageManipulator.manipulateAsync(
  imageUrl,
  [
    { resize: { width: 300 } },
    { rotate: 90 }
  ],
  { compress: 0.8 }
);
```

- Application de filtres simples

```
// Comme les filtres CSS mais en natif
const filteredImage = await ImageManipulator.manipulateAsync(
  imageUrl,
  [{ flip: { horizontal: true } }]
);
```

- **Stockage des médias**

- Sauvegarde locale avec Expo FileSystem

```
// Comme localStorage mais pour fichiers
const fileName = `${FileSystem.documentDirectory}photo.jpg`;
await FileSystem.copyAsync({
  from: photoUri,
  to: fileName
});
```

- Téléchargement vers un serveur distant

```
// Comme un upload de fichier classique
const uploadPhoto = async (uri) => {
  const response = await fetch('https://monapi.com/photos', {
    method: 'POST',
    body: JSON.stringify({ photo: uri })
  });
  return response.json();
};
```

# Exercice : Ajout de photo de profil

Améliorons notre application TinderLikeApp en permettant aux utilisateurs d'ajouter une photo de profil en utilisant la caméra ou en sélectionnant une image de la galerie.

1. Installez les dépendances nécessaires :

```
expo install expo-camera expo-image-picker expo-permissions
```

2. Créez un nouveau composant `ProfileImagePicker.js`

## Exercice : Ajout de photo de profil (suite)

Voici le début du code pour `ProfileImagePicker.js` :

```
import React, { useState, useEffect } from 'react';
import { View, Image, Button, StyleSheet } from 'react-native';
import * as ImagePicker from 'expo-image-picker';
import { Camera } from 'expo-camera';

const ProfileImagePicker = ({ onImageSelected }) => {
  const [hasPermission, setHasPermission] = useState(null);
  const [image, setImage] = useState(null);

  useEffect(() => {
    (async () => {
      const { status } = await Camera.requestPermissionsAsync();
      setHasPermission(status === 'granted');
    })();
  }, []);
```

# Exercice : Ajout de photo de profil (suite)

Suite du code pour `ProfileImagePicker.js` :

```
const takePhoto = async () => {
  const result = await ImagePicker.launchCameraAsync({
    allowsEditing: true,
    aspect: [1, 1],
    quality: 1,
  });

  if (!result.cancelled) {
    setImage(result.uri);
    onImageSelected(result.uri);
  }
};

const pickImage = async () => {
  const result = await ImagePicker.launchImageLibraryAsync({
```

# Exercice : Ajout de photo de profil (suite)

Fin du code pour ProfileImagePicker.js :

```
if (hasPermission === null) {
  return <View />;
}
if (hasPermission === false) {
  return <Text>Pas d'accès à la caméra</Text>;
}

return (
  <View style={styles.container}>
    {image && <Image source={{ uri: image }} style={styles.image} />}
    <View style={styles.buttonContainer}>
      <Button title="Prendre une photo" onPress={takePhoto} />
      <Button title="Choisir une image" onPress={pickImage} />
    </View>
  </View>
)
```

# Style

```
const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
  },
  image: {
    width: 200,
    height: 200,
    borderRadius: 100,
    marginBottom: 20,
  },
  buttonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    width: '100%',
  },
});
```

## Exercice : Ajout de photo de profil (suite)

Intégrez ce composant dans `UserProfile.js` :

```
import React, { useState } from 'react';
import { View, Text, StyleSheet, SafeAreaView } from 'react-native';
import ProfileImagePicker from './ProfileImagePicker';

const UserProfile = ({ name, bio }) => {
  const [profileImage, setProfileImage] = useState(null);

  const handleImageSelected = (imageUri) => {
    setProfileImage(imageUri);
  };

  return (
    <SafeAreaView style={styles.container}>
      <ProfileImagePicker onImageSelected={handleImageSelected} />
      <Text style={styles.name}>{name}</Text>
    </SafeAreaView>
  );
};
```



# Exercice : Ajout de photo de profil (suite)

Styles pour `UserProfile.js` :

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    padding: 20,
  },
  name: {
    fontSize: 24,
    fontWeight: 'bold',
    marginTop: 20,
  },
  bio: {
    fontSize: 16,
    textAlign: 'center',
    marginTop: 10,
  },
});
```

Cet exercice vous permet de pratiquer l'utilisation de la caméra et de la galerie d'images dans une application React Native.

## Manipulation d'images - Redimensionnement

```
// Redimensionnement et rotation
const manipResult = await ImageManipulator.manipulateAsync(
  imageUri,
  [
    { resize: { width: 300 } },
    { rotate: 90 }
  ],
  { compress: 0.8 }
);
```

# Manipulation d'images - Filtres

```
// Application de filtres
const filteredImage = await ImageManipulator.manipulateAsync(
  imageUri,
  [{ flip: { horizontal: true } } ]
);
```

# Stockage local

```
// Sauvegarde dans le système de fichiers
const fileName = `${FileSystem.documentDirectory}photo.jpg`;
await FileSystem.copyAsync({
  from: photoUri,
  to: fileName
});
```

# Upload vers serveur

```
// Fonction d'upload
const uploadPhoto = async (uri) => {
  const response = await fetch('https://monapi.com/photos', {
    method: 'POST',
    body: JSON.stringify({ photo: uri })
  });
  return response.json();
};
```

# Configuration de l'exercice

Améliorons notre application TinderLikeApp en permettant aux utilisateurs d'ajouter une photo de profil en utilisant la caméra ou en sélectionnant une image de la galerie.

## Installation des dépendances

```
expo install expo-camera expo-image-picker expo-permissions
```

# Structure du composant - Partie 1

```
// ProfileImagePicker.js - Imports et état
import React, { useState, useEffect } from 'react';
import { View, Image, Button, StyleSheet } from 'react-native';
import * as ImagePicker from 'expo-image-picker';
import { Camera } from 'expo-camera';

const ProfileImagePicker = ({ onImageSelected }) => {
  const [hasPermission, setHasPermission] = useState(null);
  const [image, setImage] = useState(null);

  useEffect(() => {
    (async () => {
      const { status } = await Camera.requestPermissionsAsync();
      setHasPermission(status === 'granted');
    })();
  });
}
```

## Structure du composant - Partie 2

```
// ProfileImagePicker.js - Méthodes de capture
const takePhoto = async () => {
  const result = await ImagePicker.launchCameraAsync({
    allowsEditing: true,
    aspect: [1, 1],
    quality: 1,
  });

  if (!result.cancelled) {
    setImage(result.uri);
    onImageSelected(result.uri);
  }
};
```



## Structure du composant - Partie 3

```
// ProfileImagePicker.js - Méthode de sélection
const pickImage = async () => {
  const result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    aspect: [1, 1],
    quality: 1,
  });

  if (!result.cancelled) {
    setImage(result.uri);
    onImageSelected(result.uri);
  }
};
```

## Structure du composant - Partie 4

```
// ProfileImagePicker.js - Rendu conditionnel
if (hasPermission === null) {
  return <View />;
}
if (hasPermission === false) {
  return <Text>Pas d'accès à la caméra</Text>;
}
```

## Structure du composant – Partie 5

```
// ProfileImagePicker.js - Rendu principal
return (
  <View style={styles.container}>
    {image && <Image source={{ uri: image }} style={styles.image} />}
    <View style={styles.buttonContainer}>
      <Button title="Prendre une photo" onPress={takePhoto} />
      <Button title="Choisir une image" onPress={pickImage} />
    </View>
  </View>
);
```

# Styles du composant

```
// ProfileImagePicker.js - Styles
const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
  },
  image: {
    width: 200,
    height: 200,
    borderRadius: 100,
    marginBottom: 20,
  },
  buttonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    width: '100%',
  },
});
```

# Intégration du composant

## UserProfile - Structure

```
// UserProfile.js - Structure
import React, { useState } from 'react';
import { View, Text, StyleSheet, SafeAreaView } from 'react-native';
import ProfileImagePicker from './ProfileImagePicker';

const UserProfile = ({ name, bio }) => {
  const [profileImage, setProfileImage] = useState(null);

  const handleImageSelected = (imageUri) => {
    setProfileImage(imageUri);
  };
};
```

# UserProfile - Rendu

```
// UserProfile.js - Rendu
return (
  <SafeAreaView style={styles.container}>
    <ProfileImagePicker onImageSelected={handleImageSelected} />
    <Text style={styles.name}>{name}</Text>
    <Text style={styles.bio}>{bio}</Text>
  </SafeAreaView>
);
```

# UserProfile - Styles

```
// UserProfile.js - Styles
const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    padding: 20,
  },
  name: {
    fontSize: 24,
    fontWeight: 'bold',
    marginTop: 20,
  },
  bio: {
    fontSize: 16,
    textAlign: 'center',
  },
});
```

# Résultat de l'exercice

Cet exercice vous permet de pratiquer l'utilisation de la caméra et de la galerie d'images dans une application React Native.

[Revenir au sommaire](#)



# Animations de base

## Animated Value - Configuration

```
// Création de la valeur animée  
const fadeAnim = useRef(new Animated.Value(0)).current;
```

# Animated Value - Animation

```
// Configuration et démarrage de l'animation
Animated.timing(fadeAnim, {
  toValue: 1,
  duration: 1000,
  useNativeDriver: true,
}).start();
```

# Animations parallèles - Configuration

```
// Configuration des animations multiples  
const fadeAnim = useRef(new Animated.Value(0)).current;  
const scaleAnim = useRef(new Animated.Value(1)).current;
```

# Animations parallèles - Exécution

```
// Exécution des animations en parallèle
Animated.parallel([
  Animated.timing(fadeAnim, {
    toValue: 1,
    duration: 1000,
  }),
  Animated.spring(scaleAnim, {
    toValue: 1.2,
    friction: 2,
  }),
]).start();
```

# Animations séquentielles

```
// Animations l'une après l'autre
Animated.sequence([
  Animated.timing(fadeAnim, {
    toValue: 1,
    duration: 500,
  }),
  Animated.timing(slideAnim, {
    toValue: 100,
    duration: 500,
  }),
]).start();
```

# Interpolation de valeurs

```
// Transformation d'une valeur en une autre
const rotation = animValue.interpolate({
  inputRange: [0, 1],
  outputRange: ['0deg', '360deg'],
});

return (
  <Animated.View
    style={{
      transform: [{ rotate: rotation }]
    }}
  />
);
```

# Gestion des gestes

## Configuration PanResponder

```
const panResponder = PanResponder.create({
  onStartShouldSetPanResponder: () => true,
  onPanResponderMove: (evt, gestureState) => {
    // Gestion du déplacement
    console.log(gestureState.dx, gestureState.dy);
  },
  onPanResponderRelease: () => {
    // Gestion du relâchement
  },
});
```

# Gestion du tap (appui simple)

```
const tapGesture = {  
  onStartShouldSetPanResponder: () => true,  
  onPanResponderRelease: (e, gestureState) => {  
    if (Math.abs(gestureState.dx) < 5 &&  
        Math.abs(gestureState.dy) < 5) {  
      // C'est un tap  
      handleTap();  
    }  
  },  
};
```



# Gestion du swipe

```
const isSwipe = (gestureState) => {  
  return Math.abs(gestureState.dx) > 50;  
};  
  
const handleSwipe = (gestureState) => {  
  if (gestureState.dx > 50) {  
    // Swipe vers la droite  
    handleRightSwipe();  
  } else if (gestureState.dx < -50) {  
    // Swipe vers la gauche  
    handleLeftSwipe();  
  }  
};
```

# Gestion du pinch (zoom)

```
const calculatePinchDistance = (evt) => {  
  const touches = evt.nativeEvent.touches;  
  if (touches.length !== 2) return 0;  
  
  const [touch1, touch2] = touches;  
  return Math.sqrt(  
    Math.pow(touch2.pageX - touch1.pageX, 2) +  
    Math.pow(touch2.pageY - touch1.pageY, 2)  
  );  
};
```

# Animations avancées

## LayoutAnimation

```
const toggleLayout = () => {  
  LayoutAnimation.configureNext(  
    LayoutAnimation.Presets.spring  
  );  
  setExpanded(!expanded);  
};
```

# Animations de liste

```
<FlatList
  data={items}
  renderItem={({ item, index }) => (
    <Animated.View
      style={{
        opacity: fadeAnim,
        transform: [{
          translateY: slideAnim.interpolate({
            inputRange: [0, 1],
            outputRange: [50 * index, 0]
          })
        }]
      }}
    >
      <ListItem item={item} />
    </Animated.View>
  )
  />
```

# Bibliothèques tierces

## React Native Reanimated

```
import Animated, {
  withSpring,
  useAnimatedStyle,
} from 'react-native-reanimated';

const animatedStyle = useAnimatedStyle(() => {
  return {
    transform: [{ scale: withSpring(1.2) }],
  };
});
```

# React Native Gesture Handler

```
import { PanGestureHandler } from 'react-native-gesture-handler';

const onGestureEvent = useAnimatedGestureHandler({
  onStart: (_, ctx) => {
    ctx.startX = translateX.value;
  },
  onActive: (event, ctx) => {
    translateX.value = ctx.startX + event.translationX;
  },
});
```

# Exercice : Carte swipeable

## Configuration initiale

```
// Installation
import { PanGestureHandler } from 'react-native-gesture-handler';

const SCREEN_WIDTH = Dimensions.get('window').width;
const SWIPE_THRESHOLD = 0.25 * SCREEN_WIDTH;
```

# Logique de base du swipe

```
const SwipeableCard = ({ profile, onSwipeLeft, onSwipeRight }) => {  
  const position = useRef(new Animated.ValueXY()).current;  
  
  const panResponder = PanResponder.create({  
    onStartShouldSetPanResponder: () => true,  
    onPanResponderMove: (_, gesture) => {  
      position.setValue({  
        x: gesture.dx,  
        y: gesture.dy  
      });  
    },  
  });  
});
```



# Gestion des swipes

```
const forceSwipe = (direction) => {  
  const x = direction === 'right' ?  
    SCREEN_WIDTH : -SCREEN_WIDTH;  
  
  Animated.timing(position, {  
    toValue: { x, y: 0 },  
    duration: 250,  
    useNativeDriver: false,  
  }).start(() => onSwipeComplete(direction));  
};  
  
const onSwipeComplete = (direction) => {  
  direction === 'right' ? onSwipeRight() : onSwipeLeft();  
  position.setValue({ x: 0, y: 0 });  
};
```

# Styles et animations – Partie 1

```
// Configuration du style de la carte
const getCardStyle = () => {
  const rotate = position.x.interpolate({
    inputRange: [-SCREEN_WIDTH * 1.5, 0, SCREEN_WIDTH * 1.5],
    outputRange: ['-120deg', '0deg', '120deg'],
  });

  return {
    ...position.getLayout(),
    transform: [{ rotate }],
  };
};
```

## Styles et animations – Partie 2

```
// Rendu du composant
return (
  <Animated.View
    style={[styles.card, getCardStyle()]}
    {...panResponder.panHandlers}
  >
    <Image
      source={{ uri: profile.imageUrl }}
      style={styles.image}
    />
    <View style={styles.textContainer}>
      <Text style={styles.name}>{profile.name}</Text>
      <Text style={styles.bio}>{profile.bio}</Text>
    </View>
  </Animated.View>
)
```

# Résultat de l'exercice

Cet exercice vous permet de créer une interface de swipe interactive et fluide, similaire à celle de Tinder, en utilisant les animations et gestes de React Native.

[Revenir au sommaire](#)

# Préparation pour la production

## Configuration des icônes et splash screen

```
// app.json - Partie 1
{
  "expo": {
    "name": "TinderLikeApp",
    "slug": "tinder-like-app",
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./assets/icon.png",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    }
  }
}
```

# Configuration des plateformes

```
// app.json - Partie 2
{
  "expo": {
    "ios": {
      "supportsTablet": true,
      "bundleIdentifier": "com.yourcompany.tinderlikeapp"
    },
    "android": {
      "adaptiveIcon": {
        "foregroundImage": "./assets/adaptive-icon.png",
        "backgroundColor": "#FFFFFF"
      },
      "package": "com.yourcompany.tinderlikeapp"
    }
  }
}
```

# Publication iOS

## Compte développeur Apple

```
# Création du certificat
xcodebuild archive -scheme TinderLikeApp -configuration Release

# Génération de l'archive
xcodebuild -exportArchive -archivePath TinderLikeApp.xcarchive \
    -exportPath ./build -exportOptionsPlist ExportOptions.plist
```

# Soumission App Store

```
# Validation du build
xcrun altool --validate-app -f build/TinderLikeApp.ipa \
    -t ios -u user@email.com -p pass

# Upload sur App Store Connect
xcrun altool --upload-app -f build/TinderLikeApp.ipa \
    -t ios -u user@email.com -p pass
```

Nous n'allons pas nous attarder sur cette façon de faire car cela demande un compte payant développeur pour chacun d'entre vous.



# Publication Android

## Génération du bundle

```
# Génération de la keystore
keytool -genkey -v -keystore tinder-like-app.keystore \
        -alias tinder-like-app -keyalg RSA -validity 10000

# Build du bundle
./gradlew bundleRelease
```

Pareil dans ce cas de figure

# Soumission Play Store

```
# Génération des APKs
bundletool build-apks --bundle=./app/build/outputs/bundle/release/app.aab \
                      --output=./app/build/outputs/apks/release/app.apks \
                      --ks=tinder-like-app.keystore \
                      --ks-key-alias=tinder-like-app
```

# Expo EAS

## Configuration initiale

```
# Installation d'EAS CLI
npm install -g eas-cli

# Login et configuration
eas login
eas build:configure
```

# Configuration EAS

```
// eas.json
{
  "build": {
    "preview": {
      "android": {
        "buildType": "apk"
      }
    },
    "production": {
      "android": {
        "buildType": "app-bundle"
      },
      "ios": {
        "distribution": "store"
      }
    }
  }
}
```

# Commandes de build

```
# Build iOS
eas build --platform ios

# Build Android
eas build --platform android

# Pour tester

# Build pour simulateur iOS
eas build --platform ios --profile development --simulator

# Une fois le build terminé, téléchargez et installez sur le simulateur
eas build:run -p ios

# Ou directement avec le lien de téléchargement fourni par EAS
```

# Bonnes pratiques de publication

## Checklist avant soumission

1. Tests approfondis sur différents appareils
2. Vérification des performances
3. Validation des assets (icônes, splash screen)
4. Préparation des captures d'écran
5. Rédaction de la description
6. Configuration de la confidentialité

N'oubliez pas de tester minutieusement votre application avant la soumission, et assurez-vous de respecter les directives de chaque store pour maximiser vos chances d'approbation.

# Exercice : Préparation publication

## Étapes initiales

### 1. Configuration des icônes et splash screen

- Remplacer les icônes dans `assets`
- Modifier `app.json`

### 2. Optimisation des performances

- Images optimisées
- Pagination des profils

### 3. Configuration EAS

- Installation : `npm install -g eas-cli`
- Initialisation : `eas init`

# Configuration app.json

```
{
  "expo": {
    "name": "TinderLikeApp",
    "slug": "tinder-like-app",
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./assets/icon.png",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    },
    "updates": {
      "fallbackToCacheTimeout": 0
    },
  },
}
```



# Configuration eas.json

```
{
  "build": {
    "preview": {
      "android": {
        "buildType": "apk"
      }
    },
    "preview2": {
      "android": {
        "gradleCommand": ":app:assembleRelease"
      }
    },
    "preview3": {
      "developmentClient": true
    },
  },
}
```

# Commandes de build

Pour créer un build de production :

```
# Build iOS
eas build --platform ios

# Build Android
eas build --platform android
```

Ces commandes généreront des builds que vous pourrez soumettre aux stores.

N'oubliez pas de tester minutieusement votre application avant la soumission, et assurez-vous de respecter les directives de chaque store pour maximiser vos chances d'approbation.

# Clean Code - Nommage

```
// Mauvais
const x = users.filter(u => u.a > 5);

// Bon
const usersActifs = utilisateurs.filter(user => user.age > 5);
```

# Clean Code – Fonctions

```
// Mauvais
function gererUtilisateur(user) {
  // 50 lignes qui font plein de choses
}

// Bon
function validerUtilisateur(user) {
  return user.age >= 18;
}

function sauvegarderUtilisateur(user) {
  // Sauvegarde uniquement
}
```

# Principes SOLID - Single Responsibility

```
// Single Responsibility
class UtilisateurService {
    creerUtilisateur() {}
    supprimerUtilisateur() {}
}

class EmailService {
    envoyerEmail() {}
}
```

# Exercice de Refactoring

## Code Initial

```
// Avant refactoring
class User {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }

  save() {
    // Logique de sauvegarde
  }

  sendEmail(subject, body) {
    // Logique d'envoi d'email
  }
}
```

# Code Refactorisé - Partie 1

```
// Classes séparées
class User {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }
}

class UserRepository {
  save(user) {
    // Logique de sauvegarde
  }
}
```

## Code Refactorisé – Partie 2

```
// Services séparés
class EmailService {
    sendEmail(to, subject, body) {
        // Logique d'envoi d'email
    }
}

class UserFactory {
    static createUser(name, email) {
        return new User(name, email);
    }
}
```



# Design Patterns (suite)

```
// Pattern Module (comme en web)
const monModule = (function() {
  // Variables privées
  let compteur = 0;

  // Méthodes publiques
  return {
    increment() {
      compteur++;
      return compteur;
    }
  };
})();
```

# Clean Code

- Nommage significatif

```
// Mauvais
const x = users.filter(u => u.a > 5);

// Bon
const usersActifs = utilisateurs.filter(user => user.age > 5);
```

# Clean Code (suite)

- Fonctions courtes et focalisées

```
// Mauvais
function gererUtilisateur(user) {
  // 50 lignes qui font plein de choses
}

// Bon
function validerUtilisateur(user) {
  return user.age >= 18;
}

function sauvegarderUtilisateur(user) {
  // Sauvegarde uniquement
}
```

# Principes SOLID

- Adaptés au contexte JavaScript/TypeScript

```
// Single Responsibility
class UtilisateurService {
  // Une seule responsabilité : gestion utilisateur
  creerUtilisateur() {}
  supprimerUtilisateur() {}
}

class EmailService {
  // Une seule responsabilité : envoi d'emails
  envoyerEmail() {}
}
```

# Parlons des principes SOLID en JavaScript

- **Single Responsibility Principle (SRP)**
  - Une fonction ou classe ne doit avoir qu'une seule raison de changer
- **Open/Closed Principle (OCP)**
  - Les entités logicielles doivent être ouvertes à l'extension, mais fermées à la modification

# Principes SOLID (suite)

- **Liskov Substitution Principle (LSP)**

- Les objets d'une superclasse doivent pouvoir être remplacés par des objets de ses sous-classes sans altérer le fonctionnement du programme

- **Interface Segregation Principle (ISP)**

- Préférer plusieurs interfaces spécifiques plutôt qu'une interface générale

- **Dependency Inversion Principle (DIP)**

- Dépendre des abstractions, pas des implémentations concrètes

## Exercice : Refactoring d'un code existant

1. Prenez un morceau de code JavaScript existant (peut être fourni ou de votre propre projet).
2. Identifiez les violations des principes SOLID et des bonnes pratiques.
3. Refactorisez le code pour le rendre plus propre et maintenable.
4. Appliquez un ou deux design patterns appropriés.

# Correction de l'exercice

Voici un exemple de refactoring appliquant le principe de responsabilité unique (SRP) et le pattern Factory :

```
// Avant
class User {
  constructor(name, email) {
    this.name = name
    this.email = email
  }

  save() {
    // Logique pour sauvegarder l'utilisateur dans la base de données
  }

  sendEmail(subject, body) {
    // Logique pour envoyer un email
  }
}
```



## Correction de l'exercice (suite)

```
// Après
class User {
    constructor(name, email) {
        this.name = name
        this.email = email
    }
}

class UserRepository {
    save(user) {
        // Logique pour sauvegarder l'utilisateur dans la base de données
    }
}

class EmailService {
```

# Bonnes pratiques React Native

## Architecture des composants – Partie 1

```
// Mauvaise pratique
const MauvaisComposant = () => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  useEffect(() => {
    setLoading(true);
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => setError(error))
      .finally(() => setLoading(false));
  }, []);
};
```

## Architecture des composants – Partie 2

```
// Bonne pratique – Hook personnalisé
const useData = () => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      setLoading(true);
      try {
        const response = await fetch('https://api.example.com/data');
        const result = await response.json();
        setData(result);
      } catch (err) {
        setError(err);
      }
    };
    fetchData();
  }, []);
}
```

## Architecture des composants – Partie 3

```
// Utilisation du hook personnalisé
const BonComposant = () => {
  const { data, loading, error } = useData();

  if (loading) return <LoadingSpinner />;
  if (error) return <ErrorMessage error={error} />;

  return (
    <View>
      {data.map(item => (
        <ItemComponent key={item.id} item={item} />
      ))}
    </View>
  );
};
```

# Optimisation des performances – Partie 1

```
// Utilisation de useMemo pour les calculs coûteux
const MemoizedComponent = () => {
  const expensiveValue = useMemo(() => {
    return someExpensiveCalculation(props);
  }, [props]);

  return <Text>{expensiveValue}</Text>;
};
```

Bientôt il n'y aura plus besoin de l'utiliser grâce à React Forget compiler donc ne vous prenez pas trop la tête pour l'instant avec ça, c'est du bonus à savoir au cas où (dans du legacy)

## Optimisation des performances – Partie 2

```
// Utilisation de useCallback pour les fonctions
const OptimizedComponent = () => {
  const handlePress = useCallback(() => {
    // Logique de gestion du clic
  }, []);

  return <TouchableOpacity onPress={handlePress} />;
};
```

# Performance et État

- **Performance**

- Utilisation de `React.memo` pour éviter les re-rendus inutiles
- Optimisation des listes avec `FlatList` et `VirtualizedList`
- Lazy loading des composants et des images

- **Gestion de l'état**

- Utilisation appropriée des hooks (`useState`, `useEffect`, `useCallback`, `useMemo`)
- Mise en place d'un état global avec Context API ou Redux

# Debugging et Tests

- **Debugging**

- Utilisation de React Native Debugger
- Mise en place de logs appropriés

- **Tests**

- Mise en place de tests unitaires avec Jest
- Tests d'intégration avec React Native Testing Library



# Exercice : Optimisation de l'application TinderLikeApp

Optimisons notre application TinderLikeApp en appliquant certaines des meilleures pratiques.

## 1. Optimisez le rendu de la liste des profils :

```
import React, { memo } from 'react';
import { FlatList } from 'react-native';

const ProfileItem = memo(({ profile, onPress }) => {
  // Composant d'item de profil optimisé
});

const ProfileList = ({ profiles, onProfilePress }) => {
  const renderItem = ({ item }) => (
    <ProfileItem profile={item} onPress={() => onProfilePress(item)} />
  );

  return (
    <FlatList
      data={profiles}
    />
  );
};
```

# Exercice : Optimisation (suite)

2. Implémentez le lazy loading des images :

```
import React, { useState } from 'react';
import { Image, View } from 'react-native';

const LazyImage = ({ source, style }) => {
  const [loaded, setLoaded] = useState(false);

  return (
    <View>
      {!loaded && <View style={[style, { backgroundColor: '#ccc' }]} />}
      <Image
        source={source}
        style={[style, { display: loaded ? 'flex' : 'none' }]}
        onLoad={() => setLoaded(true)}
      />
    </View>
  )
}
```

## Exercice : Optimisation (suite)

3. Mettez en place un système de logging :

```
// utils/logger.js
const logger = {
  info: (message) => {
    if (__DEV__) {
      console.log(`[INFO] ${message}`);
    }
  },
  error: (message, error) => {
    if (__DEV__) {
      console.error(`[ERROR] ${message}`, error);
    }
    // Ici, vous pourriez également envoyer les erreurs à un service de suivi des erreurs
  }
};
```

# Exercice : Optimisation (fin)

4. Ajoutez un test unitaire simple :

```
// __tests__/ProfileItem.test.js
import React from 'react';
import { render, fireEvent } from '@testing-library/react-native';
import ProfileItem from '../components/ProfileItem';

describe('ProfileItem', () => {
  it('renders correctly', () => {
    const profile = { id: '1', name: 'John Doe', bio: 'Test bio' };
    const { getByText } = render(<ProfileItem profile={profile} />);

    expect(getByText('John Doe')).toBeTruthy();
    expect(getByText('Test bio')).toBeTruthy();
  });

  it('calls onPress when pressed', () => {
```

Ces optimisations et bonnes pratiques amélioreront les performances et la maintenabilité de votre application TinderLikeApp. N'oubliez pas de les appliquer tout au long du développement de votre application.

Félicitations ! Vous avez maintenant terminé cette formation sur React Native et Expo. Vous avez appris à créer une application mobile complète, de la configuration initiale à l'optimisation et au déploiement. [Revenir au sommaire](#)

# Code source du projet d'exercice :

*Ci dessous le lien du projet :*

[Lien du projet - Github](#)

