TP 4 : Fonctionnalités Avancées avec Expo et TypeScript

Objectif

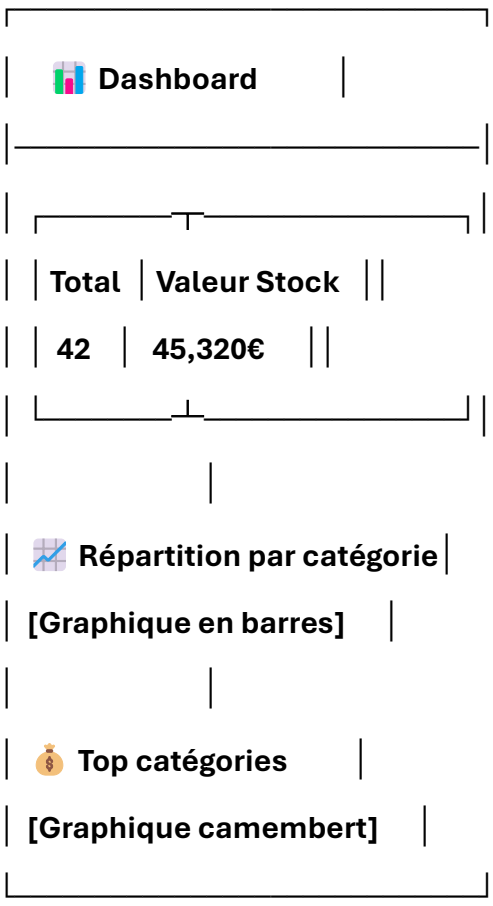
Enrichir l'application avec des fonctionnalités avancées : statistiques, graphiques, filtres, upload d'images et mode offline.

Ce que nous allons ajouter

- **Dashboard avec statistiques et graphiques**
- **Système de catégories complet**
- **Filtres et tri avancés**
- **Upload d'images avec Expo**
- **Mode offline avec cache**
- **Animations et transitions**

Nouvelles fonctionnalités visuelles

Dashboard



Étape 1 : Installation des dépendances

```
cd GestionProduitsMobile

# Graphiques
npx expo install react-native-svg react-native-chart-kit

# Images
npx expo install expo-image-picker expo-media-library

# Cache et stockage
npx expo install @react-native-async-storage/async-storage

# Modal et animations
npx expo install react-native-modal react-native-reanimated

# Sélecteur et slider
npm install @react-native-picker/picker @react-native-community/slider
```

Étape 2 : Types pour les statistiques

2.1 Types mis à jour (src/types/index.ts)

```
// Ajouter aux types existants...

export interface Statistics {
 general: {
  total_produits: number;
  stock_total: number;
  valeur_totale: number;
  prix_moyen: number;
 };
 parCategorie: CategoryStats[];
```

```
}

export interface CategoryStats {
  categorie: string;
  couleur: string;
  nombre_produits: number;
  stock_categorie: number;
  valeur_categorie: number;
}

export interface Filters {
  categorie_id: string;
  prix_min: number;
  prix_max: number;
  stock_min: number;
  sort_by: 'nom' | 'prix' | 'stock' | 'created_at';
  sort_order: 'ASC' | 'DESC';
}

// Navigation mise à jour
export type RootStackParamList = {
  ProductList: undefined;
  ProductDetail: { productId: number };
  ProductForm: { product?: Product };
  Dashboard: undefined;
};
```

Étape 3 : Service des statistiques

3.1 Service statistiques (src/services/statisticsService.ts)

```
import apiClient from './apiClient';
import { Statistics, ApiResponse } from '../types';

class StatisticsService {
  async getStatistics(): Promise<Statistics> {
    const response = await apiClient.get<ApiResponse<Statistics>>('/statistiques');
    if (!response.data.success || !response.data.data) {
      throw new Error(response.data.message || 'Erreur lors de la récupération des statistiques');
    }
    return response.data.data;
  }
}

export default new StatisticsService();
```

3.2 Hook pour les statistiques (src/hooks/useStatistics.ts)

```
import { useState, useEffect, useCallback } from 'react';
import statisticsService from '../services/statisticsService';
import { Statistics, LoadingState, ApiError } from '../types';

interface UseStatisticsReturn extends LoadingState {
  statistics: Statistics | null;
  fetchStatistics: () => Promise<void>;
  refreshStatistics: () => Promise<void>;
}

export const useStatistics = (): UseStatisticsReturn => {
  const [statistics, setStatistics] = useState<Statistics | null>(null);
```

```
const [loadingState, setLoadingState] = useState<LoadingState>({
 isLoading: true,
 isRefreshing: false,
 error: null
});

const fetchStatistics = useCallback(async () => {
 try {
  setLoadingState(prev => ({ ...prev, error: null }));
  const data = await statisticsService.getStatistics();
  setStatistics(data);
 } catch (error) {
  const apiError = error as ApiError;
  setLoadingState(prev => ({
   ...prev,
   error: apiError.customMessage || apiError.message
  }));
 } finally {
  setLoadingState(prev => ({
   ...prev,
   isLoading: false,
   isRefreshing: false
  }));
 }
}, []);

const refreshStatistics = useCallback(async () => {
 setLoadingState(prev => ({ ...prev, isRefreshing: true }));
```

```
  await fetchStatistics();
 }, [fetchStatistics]);

 useEffect(() => {
  fetchStatistics();
 }, [fetchStatistics]);

 return {
  statistics,
  ...loadingState,
  fetchStatistics,
  refreshStatistics
 };
};
```

📊 **Étape 4 : Dashboard avec graphiques**

4.1 Écran Dashboard (src/screens/DashboardScreen.tsx)

```
import React from 'react';
import {
 View,
 ScrollView,
 Text,
 StyleSheet,
 Dimensions,
 RefreshControl,
 TouchableOpacity,
 ActivityIndicator,
 SafeAreaView
} from 'react-native';
```

```
import {
  BarChart,
  PieChart,
  LineChart,
  ProgressChart
} from 'react-native-chart-kit';
import { StackNavigationProp } from '@react-navigation/stack';
import { MaterialIcons } from '@expo/vector-icons';
import { useStatistics } from '../hooks/useStatistics';
import { RootStackParamList } from '../types';
import { colors, fonts, spacing, shadows } from '../styles/theme';

const screenWidth = Dimensions.get('window').width;

type DashboardScreenNavigationProp = StackNavigationProp<
  RootStackParamList,
  'Dashboard'
>;

interface Props {
  navigation: DashboardScreenNavigationProp;
}

const DashboardScreen: React.FC<Props> = ({ navigation }) => {
  const { statistics, isLoading, isRefreshing, error, refreshStatistics } =
useStatistics();

  if (isLoading && !isRefreshing) {
```

```
  return (
    <View style={styles.loadingContainer}>
      <ActivityIndicator size="large" color={colors.primary} />
      <Text style={styles.loadingText}>Chargement des statistiques...</Text>
    </View>
  );
}

if (error || !statistics) {
  return (
    <View style={styles.errorContainer}>
      <MaterialIcons name="error-outline" size={64} color={colors.danger} />
      <Text style={styles.errorText}>
        {error || 'Impossible de charger les statistiques'}
      </Text>
      <TouchableOpacity
        style={[styles.retryButton, shadows.small]}
        onPress={refreshStatistics}
      >
        <Text style={styles.retryText}>Réessayer</Text>
      </TouchableOpacity>
    </View>
  );
}

// Préparer les données pour les graphiques
const pieData = statistics.parCategorie
  .filter(cat => cat.nombre_produits > 0)
```

```
  .map(cat => ({
    name: cat.categorie,
    population: cat.nombre_produits,
    color: cat.couleur,
    legendFontColor: colors.dark,
    legendFontSize: 12
  }));

const barData = {
  labels: statistics.parCategorie.map(cat => cat.categorie.substring(0, 4)),
  datasets: [{
    data: statistics.parCategorie.map(cat => cat.stock_categorie)
  }]
};

const chartConfig = {
  backgroundColor: colors.white,
  backgroundGradientFrom: colors.white,
  backgroundGradientTo: colors.white,
  decimalPlaces: 0,
  color: (opacity = 1) => `rgba(52, 152, 219, ${opacity})`,
  labelColor: (opacity = 1) => `rgba(44, 62, 80, ${opacity})`,
  style: {
    borderRadius: 16
  },
  propsForDots: {
    r: '6',
    strokeWidth: '2',
```

```
   stroke: colors.primary
  }
};

const progressData = {
 labels: ["Stock"],
 data: [statistics.general.stock_total / 1000] // Normaliser sur 1000
};

return (
 <SafeAreaView style={styles.container}>
  <ScrollView
   showsVerticalScrollIndicator={false}
   refreshControl={
    <RefreshControl
     refreshing={isRefreshing}
     onRefresh={refreshStatistics}
     colors={[colors.primary]}
     tintColor={colors.primary}
    />
   }
  >
   {/* Header */}
   <View style={styles.header}>
    <Text style={styles.title}>Dashboard</Text>
    <TouchableOpacity
     style={styles.listButton}
     onPress={() => navigation.navigate('ProductList')}
```

```
 >
  <MaterialIcons name="list" size={24} color={colors.primary} />
 </TouchableOpacity>
</View>

{/* Cartes de statistiques */}
<View style={styles.statsGrid}>
 <View style={[styles.statCard, { backgroundColor: colors.primary + '20' }]}>
  <MaterialIcons name="inventory" size={32} color={colors.primary} />
  <Text style={styles.statValue}>{statistics.general.total_produits}</Text>
  <Text style={styles.statLabel}>Produits</Text>
 </View>

 <View style={[styles.statCard, { backgroundColor: colors.secondary + '20' }]}>
  <MaterialIcons name="inventory-2" size={32} color={colors.secondary} />
  <Text style={styles.statValue}>{statistics.general.stock_total}</Text>
  <Text style={styles.statLabel}>Stock Total</Text>
 </View>

 <View style={[styles.statCard, { backgroundColor: colors.warning + '20' }]}>
  <MaterialIcons name="euro" size={32} color={colors.warning} />
  <Text style={styles.statValue}>
   {Math.round(statistics.general.valeur_totale).toLocaleString('fr-FR')}€
  </Text>
  <Text style={styles.statLabel}>Valeur Stock</Text>
 </View>

 <View style={[styles.statCard, { backgroundColor: colors.danger + '20' }]}>
```

```
    <MaterialIcons name="trending-up" size={32} color={colors.danger} />
    <Text style={styles.statValue}>
     {Math.round(statistics.general.prix_moyen)}€
    </Text>
    <Text style={styles.statLabel}>Prix Moyen</Text>
   </View>
  </View>

  {/* Graphique en camembert */}
  {pieData.length > 0 && (
   <View style={[styles.chartContainer, shadows.small]}>
    <Text style={styles.chartTitle}>Répartition par catégorie</Text>
    <PieChart
     data={pieData}
     width={screenWidth - 40}
     height={220}
     chartConfig={chartConfig}
     accessor="population"
     backgroundColor="transparent"
     paddingLeft="15"
     absolute
    />
   </View>
  )}

  {/* Graphique en barres */}
  <View style={[styles.chartContainer, shadows.small]}>
   <Text style={styles.chartTitle}>Stock par catégorie</Text>
```

```
 <BarChart
  data={barData}
  width={screenWidth - 40}
  height={220}
  yAxisLabel=""
  chartConfig={chartConfig}
  verticalLabelRotation={0}
  showValuesOnTopOfBars
  fromZero
  style={styles.chart}
 />
</View>

{/* Progress Chart */}
<View style={[styles.chartContainer, shadows.small]}>
 <Text style={styles.chartTitle}>Indicateur de stock</Text>
 <ProgressChart
  data={progressData}
  width={screenWidth - 40}
  height={220}
  strokeWidth={16}
  radius={32}
  chartConfig={chartConfig}
  hideLegend={false}
  style={styles.chart}
 />
</View>
```

```
      {/* Liste détaillée */}
      <View style={[styles.detailsContainer, shadows.small]}>
       <Text style={styles.chartTitle}>Détails par catégorie</Text>
       {statistics.parCategorie.map((cat, index) => (
        <View key={index} style={styles.categoryItem}>
         <View style={[styles.categoryDot, { backgroundColor: cat.couleur }]} />
         <View style={styles.categoryInfo}>
          <Text style={styles.categoryName}>{cat.categorie}</Text>
          <Text style={styles.categoryStats}>
           {cat.nombre_produits} produits • {cat.stock_categorie} unités
          </Text>
         </View>
         <Text style={styles.categoryValue}>
          {Math.round(cat.valeur_categorie).toLocaleString('fr-FR')}€
         </Text>
        </View>
       ))}
      </View>
     </ScrollView>
    </SafeAreaView>
  );
};

const styles = StyleSheet.create({
  container: {
   flex: 1,
   backgroundColor: colors.background,
  },
```

```
loadingContainer: {
 flex: 1,
 justifyContent: 'center',
 alignItems: 'center',
 backgroundColor: colors.background,
},
loadingText: {
 marginTop: spacing.md,
 fontSize: fonts.medium,
 color: colors.gray,
},
errorContainer: {
 flex: 1,
 justifyContent: 'center',
 alignItems: 'center',
 backgroundColor: colors.background,
 padding: spacing.xl,
},
errorText: {
 fontSize: fonts.medium,
 color: colors.gray,
 textAlign: 'center',
 marginVertical: spacing.md,
},
retryButton: {
 backgroundColor: colors.primary,
 paddingHorizontal: spacing.xl,
 paddingVertical: spacing.md,
```

```
  borderRadius: 8,
},
retryText: {
  color: colors.white,
  fontSize: fonts.medium,
  fontWeight: '600',
},
header: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  padding: spacing.lg,
},
title: {
  fontSize: fonts.xxlarge,
  fontWeight: 'bold',
  color: colors.dark,
},
listButton: {
  padding: spacing.sm,
},
statsGrid: {
  flexDirection: 'row',
  flexWrap: 'wrap',
  paddingHorizontal: spacing.md,
},
statCard: {
  width: '47%',
```

```
  margin: '1.5%',
  padding: spacing.lg,
  borderRadius: 12,
  alignItems: 'center',
 },
 statValue: {
  fontSize: fonts.xlarge,
  fontWeight: 'bold',
  color: colors.dark,
  marginTop: spacing.sm,
 },
 statLabel: {
  fontSize: fonts.small,
  color: colors.gray,
  marginTop: spacing.xs,
 },
 chartContainer: {
  backgroundColor: colors.white,
  margin: spacing.lg,
  padding: spacing.lg,
  borderRadius: 12,
 },
 chartTitle: {
  fontSize: fonts.large,
  fontWeight: '600',
  color: colors.dark,
  marginBottom: spacing.md,
 },
```

```
chart: {
 borderRadius: 16,
},
detailsContainer: {
 backgroundColor: colors.white,
 margin: spacing.lg,
 padding: spacing.lg,
 borderRadius: 12,
 marginBottom: spacing.xl,
},
categoryItem: {
 flexDirection: 'row',
 alignItems: 'center',
 paddingVertical: spacing.sm,
 borderBottomWidth: 1,
 borderBottomColor: colors.lightGray,
},
categoryDot: {
 width: 12,
 height: 12,
 borderRadius: 6,
 marginRight: spacing.sm,
},
categoryInfo: {
 flex: 1,
},
categoryName: {
 fontSize: fonts.medium,
```

```
  fontWeight: '600',
  color: colors.dark,
 },
 categoryStats: {
  fontSize: fonts.small,
  color: colors.gray,
  marginTop: 2,
 },
 categoryValue: {
  fontSize: fonts.medium,
  fontWeight: 'bold',
  color: colors.primary,
 },
});

export default DashboardScreen;
```

Étape 5 : Filtres avancés

5.1 Composant de filtres (src/components/FilterModal.tsx)

```
import React, { useState } from 'react';
import {
 View,
 Text,
 Modal,
 StyleSheet,
 TouchableOpacity,
 TextInput,
 ScrollView,
 Platform
```

```
} from 'react-native';
import { Picker } from '@react-native-picker/picker';
import Slider from '@react-native-community/slider';
import { MaterialIcons } from '@expo/vector-icons';
import { Category, Filters } from '../types';
import { colors, fonts, spacing, shadows } from '../styles/theme';

interface FilterModalProps {
  visible: boolean;
  onClose: () => void;
  onApply: (filters: Filters) => void;
  categories: Category[];
  currentFilters: Filters;
}

const FilterModal: React.FC<FilterModalProps> = ({
  visible,
  onClose,
  onApply,
  categories,
  currentFilters
}) => {
  const [filters, setFilters] = useState<Filters>(currentFilters);

  const handleApply = () => {
    onApply(filters);
    onClose();
  };
```

```
const handleReset = () => {
 const defaultFilters: Filters = {
  categorie_id: '',
  prix_min: 0,
  prix_max: 9999,
  stock_min: 0,
  sort_by: 'nom',
  sort_order: 'ASC'
 };
 setFilters(defaultFilters);
};

return (
 <Modal
  visible={visible}
  animationType="slide"
  transparent={true}
  onRequestClose={onClose}
 >
  <View style={styles.overlay}>
   <View style={styles.container}>
    <View style={styles.header}>
     <Text style={styles.title}>Filtres avancés</Text>
     <TouchableOpacity onPress={onClose}>
      <MaterialIcons name="close" size={24} color={colors.dark} />
     </TouchableOpacity>
    </View>
```

```
<ScrollView showsVerticalScrollIndicator={false}>
 {/* Catégorie */}
 <View style={styles.section}>
  <Text style={styles.label}>Catégorie</Text>
  <View style={styles.pickerContainer}>
   <Picker
    selectedValue={filters.categorie_id}
    onValueChange={(value) => setFilters({...filters, categorie_id: value})}
    style={styles.picker}
   >
    <Picker.Item label="Toutes les catégories" value="" />
    {categories.map(cat => (
     <Picker.Item
      key={cat.id}
      label={cat.nom}
      value={cat.id.toString()}
     />
    ))}
   </Picker>
  </View>
 </View>

 {/* Prix */}
 <View style={styles.section}>
  <Text style={styles.label}>
   Prix ({filters.prix_min}€ - {filters.prix_max}€)
  </Text>
```

```
 <View style={styles.rangeContainer}>
  <TextInput
   style={styles.rangeInput}
   value={filters.prix_min.toString()}
   onChangeText={(text) => setFilters({
    ...filters,
    prix_min: parseInt(text) || 0
   })}
   keyboardType="numeric"
   placeholder="Min"
  />
  <Text style={styles.rangeSeparator}>-</Text>
  <TextInput
   style={styles.rangeInput}
   value={filters.prix_max.toString()}
   onChangeText={(text) => setFilters({
    ...filters,
    prix_max: parseInt(text) || 9999
   })}
   keyboardType="numeric"
   placeholder="Max"
  />
 </View>
</View>

{/* Stock minimum */}
<View style={styles.section}>
 <Text style={styles.label}>Stock minimum: {filters.stock_min}</Text>
```

```
  <Slider
    style={styles.slider}
    minimumValue={0}
    maximumValue={100}
    value={filters.stock_min}
    onValueChange={(value) => setFilters({
      ...filters,
      stock_min: Math.round(value)
    })}
    minimumTrackTintColor={colors.primary}
    maximumTrackTintColor={colors.lightGray}
  />
</View>

{/* Tri */}
<View style={styles.section}>
  <Text style={styles.label}>Trier par</Text>
  <View style={styles.sortOptions}>
    {[
      { label: 'Nom', value: 'nom' as const },
      { label: 'Prix', value: 'prix' as const },
      { label: 'Stock', value: 'stock' as const },
      { label: 'Date', value: 'created_at' as const }
    ].map(option => (
      <TouchableOpacity
        key={option.value}
        style={[
          styles.sortButton,
```

```
        filters.sort_by === option.value && styles.sortButtonActive
      ]}
      onPress={() => setFilters({...filters, sort_by: option.value})}
    >
      <Text style={[
        styles.sortButtonText,
        filters.sort_by === option.value && styles.sortButtonTextActive
      ]}>
        {option.label}
      </Text>
    </TouchableOpacity>
  ))}
 </View>
</View>

{/* Ordre */}
<View style={styles.section}>
 <Text style={styles.label}>Ordre</Text>
 <View style={styles.orderContainer}>
  <TouchableOpacity
   style={[
    styles.orderButton,
    filters.sort_order === 'ASC' && styles.orderButtonActive
   ]}
   onPress={() => setFilters({...filters, sort_order: 'ASC'})}
  >
   <MaterialIcons name="arrow-upward" size={20} color={colors.dark} />
   <Text style={styles.orderText}>Croissant</Text>
```

```
    </TouchableOpacity>

    <TouchableOpacity
     style={[
      styles.orderButton,
      filters.sort_order === 'DESC' && styles.orderButtonActive
     ]}
     onPress={() => setFilters({...filters, sort_order: 'DESC'})}
    >
     <MaterialIcons name="arrow-downward" size={20} color={colors.dark} />
     <Text style={styles.orderText}>Décroissant</Text>
    </TouchableOpacity>
   </View>
  </View>
 </ScrollView>

 {/* Actions */}
 <View style={styles.actions}>
  <TouchableOpacity
   style={[styles.actionButton, styles.resetButton]}
   onPress={handleReset}
  >
   <Text style={styles.resetButtonText}>Réinitialiser</Text>
  </TouchableOpacity>
  <TouchableOpacity
   style={[styles.actionButton, styles.applyButton]}
   onPress={handleApply}
  >
```

```
          <Text style={styles.applyButtonText}>Appliquer</Text>
        </TouchableOpacity>
      </View>
    </View>
  </View>
 </Modal>
);
};

const styles = StyleSheet.create({
 overlay: {
  flex: 1,
  backgroundColor: 'rgba(0,0,0,0.5)',
  justifyContent: 'flex-end',
 },
 container: {
  backgroundColor: colors.white,
  borderTopLeftRadius: 20,
  borderTopRightRadius: 20,
  maxHeight: '80%',
 },
 header: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  padding: spacing.lg,
  borderBottomWidth: 1,
  borderBottomColor: colors.lightGray,
```

```
},
title: {
 fontSize: fonts.large,
 fontWeight: 'bold',
 color: colors.dark,
},
section: {
 padding: spacing.lg,
 borderBottomWidth: 1,
 borderBottomColor: colors.lightGray,
},
label: {
 fontSize: fonts.medium,
 fontWeight: '600',
 color: colors.dark,
 marginBottom: spacing.sm,
},
pickerContainer: {
 backgroundColor: colors.background,
 borderRadius: 8,
 overflow: 'hidden',
},
picker: {
 height: Platform.OS === 'ios' ? 200 : 50,
},
rangeContainer: {
 flexDirection: 'row',
 alignItems: 'center',
```

```
},
rangeInput: {
 flex: 1,
 backgroundColor: colors.background,
 padding: spacing.sm,
 borderRadius: 8,
 textAlign: 'center',
},
rangeSeparator: {
 marginHorizontal: spacing.sm,
 color: colors.gray,
},
slider: {
 width: '100%',
 height: 40,
},
sortOptions: {
 flexDirection: 'row',
 flexWrap: 'wrap',
},
sortButton: {
 paddingHorizontal: spacing.md,
 paddingVertical: spacing.sm,
 borderRadius: 20,
 backgroundColor: colors.background,
 marginRight: spacing.sm,
 marginBottom: spacing.sm,
},
```

```
sortButtonActive: {
 backgroundColor: colors.primary,
},
sortButtonText: {
 color: colors.dark,
 fontSize: fonts.small,
},
sortButtonTextActive: {
 color: colors.white,
},
orderContainer: {
 flexDirection: 'row',
 justifyContent: 'space-around',
},
orderButton: {
 flexDirection: 'row',
 alignItems: 'center',
 padding: spacing.md,
 borderRadius: 8,
 backgroundColor: colors.background,
},
orderButtonActive: {
 backgroundColor: colors.primary,
},
orderText: {
 marginLeft: spacing.sm,
 color: colors.dark,
},
```

```
  actions: {
    flexDirection: 'row',
    padding: spacing.lg,
    borderTopWidth: 1,
    borderTopColor: colors.lightGray,
  },
  actionButton: {
    flex: 1,
    padding: spacing.md,
    borderRadius: 8,
    alignItems: 'center',
    marginHorizontal: spacing.xs,
  },
  resetButton: {
    backgroundColor: colors.lightGray,
  },
  resetButtonText: {
    color: colors.dark,
    fontWeight: '600',
  },
  applyButton: {
    backgroundColor: colors.primary,
  },
  applyButtonText: {
    color: colors.white,
    fontWeight: '600',
  },
});
```

```
export default FilterModal;
```

📸 **Étape 6 : Upload d'images avec Expo**

6.1 Service d'images (src/services/imageService.ts)

```
import * as ImagePicker from 'expo-image-picker';
import { Alert, Platform } from 'react-native';

interface ImageResult {
  uri: string;
  base64?: string;
  width: number;
  height: number;
  type?: string;
}

class ImageService {
  // Demander les permissions
  async requestPermissions(): Promise<boolean> {
    if (Platform.OS !== 'web') {
      const { status: cameraStatus } = await
ImagePicker.requestCameraPermissionsAsync();
      const { status: mediaStatus } = await
ImagePicker.requestMediaLibraryPermissionsAsync();

      if (cameraStatus !== 'granted' || mediaStatus !== 'granted') {
        Alert.alert(
          'Permissions requises',
          'L\'application a besoin d\'accéder à votre caméra et galerie photo.'
        );
```

```
    return false;
   }
  }
  return true;
}

// Options communes pour les images
private getImageOptions(): ImagePicker.ImagePickerOptions {
  return {
   mediaTypes: ImagePicker.MediaTypeOptions.Images,
   allowsEditing: true,
   aspect: [1, 1],
   quality: 0.8,
   base64: true,
  };
}

// Prendre une photo
async takePhoto(): Promise<ImageResult | null> {
  const hasPermission = await this.requestPermissions();
  if (!hasPermission) return null;

  const result = await ImagePicker.launchCameraAsync(this.getImageOptions());

  if (!result.canceled && result.assets[0]) {
   return result.assets[0] as ImageResult;
  }
  return null;
```

```
  }

  // Sélectionner depuis la galerie
  async pickImage(): Promise<ImageResult | null> {
    const hasPermission = await this.requestPermissions();
    if (!hasPermission) return null;

    const result = await
ImagePicker.launchImageLibraryAsync(this.getImageOptions());

    if (!result.canceled && result.assets[0]) {
      return result.assets[0] as ImageResult;
    }
    return null;
  }

  // Afficher le menu de sélection
  async selectImage(): Promise<ImageResult | null> {
    return new Promise((resolve) => {
      Alert.alert(
        'Sélectionner une image',
        'Choisissez la source de l\'image',
        [
          { text: 'Annuler', style: 'cancel', onPress: () => resolve(null) },
          {
            text: 'Prendre une photo',
            onPress: async () => {
              const result = await this.takePhoto();
```

```
          resolve(result);
        }
      },
      {
        text: 'Choisir depuis la galerie',
        onPress: async () => {
          const result = await this.pickImage();
          resolve(result);
        }
      }
    ]
  );
 });
}

 // Convertir l'image pour l'upload
 prepareImageForUpload(image: ImageResult): FormData {
  const formData = new FormData();

  formData.append('image', {
   uri: image.uri,
   type: image.type || 'image/jpeg',
   name: `photo_${Date.now()}.jpg`
  } as any);

  return formData;
 }
}
```

```
export default new ImageService();
```

💾 Étape 7 : Cache et persistance

7.1 Service de cache (src/services/cacheService.ts)

```
import AsyncStorage from '@react-native-async-storage/async-storage';
import { Product, Category, Statistics } from '../types';

interface CacheData<T> {
 data: T;
 timestamp: number;
}

class CacheService {
 private readonly CACHE_DURATION = 60 * 60 * 1000; // 1 heure
 private readonly KEYS = {
  PRODUCTS: '@cache_products',
  CATEGORIES: '@cache_categories',
  STATISTICS: '@cache_statistics',
  FILTERS: '@cache_filters'
 };

 // Sauvegarder en cache
 private async set<T>(key: string, data: T): Promise<void> {
  try {
   const cacheData: CacheData<T> = {
    data,
    timestamp: Date.now()
   };
```

```
    await AsyncStorage.setItem(key, JSON.stringify(cacheData));
  } catch (error) {
    console.error('Erreur cache set:', error);
  }
}

// Récupérer du cache
private async get<T>(key: string): Promise<T | null> {
  try {
    const cached = await AsyncStorage.getItem(key);
    if (!cached) return null;

    const cacheData: CacheData<T> = JSON.parse(cached);

    // Vérifier l'expiration
    if (Date.now() - cacheData.timestamp > this.CACHE_DURATION) {
      await AsyncStorage.removeItem(key);
      return null;
    }

    return cacheData.data;
  } catch (error) {
    console.error('Erreur cache get:', error);
    return null;
  }
}

// Méthodes publiques
```

```
async saveProducts(products: Product[]): Promise<void> {
 await this.set(this.KEYS.PRODUCTS, products);
}

async getProducts(): Promise<Product[] | null> {
 return this.get<Product[]>(this.KEYS.PRODUCTS);
}

async saveCategories(categories: Category[]): Promise<void> {
 await this.set(this.KEYS.CATEGORIES, categories);
}

async getCategories(): Promise<Category[] | null> {
 return this.get<Category[]>(this.KEYS.CATEGORIES);
}

async saveStatistics(statistics: Statistics): Promise<void> {
 await this.set(this.KEYS.STATISTICS, statistics);
}

async getStatistics(): Promise<Statistics | null> {
 return this.get<Statistics>(this.KEYS.STATISTICS);
}

async clearAll(): Promise<void> {
 try {
  await AsyncStorage.multiRemove(Object.values(this.KEYS));
 } catch (error) {
```

```
    console.error('Erreur clear cache:', error);
  }
 }

 async getCacheSize(): Promise<string> {
  try {
    const keys = await AsyncStorage.getAllKeys();
    const cacheKeys = keys.filter(key => key.startsWith('@cache_'));

    let totalSize = 0;
    for (const key of cacheKeys) {
      const value = await AsyncStorage.getItem(key);
      if (value) {
        totalSize += value.length;
      }
    }

    // Convertir en KB ou MB
    if (totalSize < 1024) {
      return `${totalSize} B`;
    } else if (totalSize < 1024 * 1024) {
      return `${(totalSize / 1024).toFixed(2)} KB`;
    } else {
      return `${(totalSize / (1024 * 1024)).toFixed(2)} MB`;
    }
  } catch (error) {
    console.error('Erreur calcul taille cache:', error);
    return '0 B';
```

```
  }
 }
}

export default new CacheService();
```

7.2 Hook avec cache (src/hooks/useProductsWithCache.ts)

```
import { useState, useEffect, useCallback } from 'react';
import { Alert } from 'react-native';
import productService from '../services/productService';
import cacheService from '../services/cacheService';
import { Product, ProductInput, Category, LoadingState, ApiError } from '../types';

export const useProductsWithCache = () => {
 const [products, setProducts] = useState<Product[]>([]);
 const [categories, setCategories] = useState<Category[]>([]);
 const [loadingState, setLoadingState] = useState<LoadingState>({
  isLoading: true,
  isRefreshing: false,
  error: null
 });

 // Charger depuis le cache d'abord
 const loadFromCache = useCallback(async () => {
  const cachedProducts = await cacheService.getProducts();
  const cachedCategories = await cacheService.getCategories();

  if (cachedProducts && cachedCategories) {
   setProducts(cachedProducts);
```

```
    setCategories(cachedCategories);
    setLoadingState(prev => ({ ...prev, isLoading: false }));
    return true;
   }
   return false;
  }, []);

  // Charger depuis l'API
  const fetchFromAPI = useCallback(async () => {
   try {
    const [productsData, categoriesData] = await Promise.all([
     productService.getAllProducts(),
     productService.getCategories()
    ]);

    setProducts(productsData);
    setCategories(categoriesData);

    // Sauvegarder en cache
    await Promise.all([
     cacheService.saveProducts(productsData),
     cacheService.saveCategories(categoriesData)
    ]);

    setLoadingState(prev => ({ ...prev, error: null }));
   } catch (error) {
    const apiError = error as ApiError;
    setLoadingState(prev => ({
```

```
      ...prev,
      error: apiError.customMessage || apiError.message
    }));

    // Si erreur et pas de cache, afficher l'alerte
    if (products.length === 0) {
      Alert.alert(
        'Mode hors ligne',
        'Impossible de charger les données. Vérifiez votre connexion.',
        [{ text: 'OK' }]
      );
    }
  } finally {
    setLoadingState(prev => ({
      ...prev,
      isLoading: false,
      isRefreshing: false
    }));
  }
}, [products.length]);

// Charger les produits
const fetchProducts = useCallback(async () => {
  const hasCache = await loadFromCache();
  if (!hasCache) {
    setLoadingState(prev => ({ ...prev, isLoading: true }));
  }
  await fetchFromAPI();
```

```
  }, [loadFromCache, fetchFromAPI]);

  // Rafraîchir (forcer l'API)
  const refreshProducts = useCallback(async () => {
    setLoadingState(prev => ({ ...prev, isRefreshing: true }));
    await fetchFromAPI();
  }, [fetchFromAPI]);

  // Charger au montage
  useEffect(() => {
    fetchProducts();
  }, []);

  return {
    products,
    categories,
    ...loadingState,
    fetchProducts,
    refreshProducts
  };
};
```

Étape 8 : Navigation mise à jour

8.1 Navigation avec Dashboard (src/navigation/AppNavigator.tsx)

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { MaterialIcons } from '@expo/vector-icons';
```

```
import ProductListScreen from '../screens/ProductListScreen';
import ProductDetailScreen from '../screens/ProductDetailScreen';
import ProductFormScreen from '../screens/ProductFormScreen';
import DashboardScreen from '../screens/DashboardScreen';
import { RootStackParamList } from '../types';
import { colors } from '../styles/theme';

const Stack = createStackNavigator<RootStackParamList>();
const Tab = createBottomTabNavigator();

// Navigateur principal avec tabs
const MainTabNavigator = () => {
 return (
  <Tab.Navigator
   screenOptions={({ route }) => ({
    tabBarIcon: ({ focused, color, size }) => {
     let iconName: keyof typeof MaterialIcons.glyphMap = 'home';

     if (route.name === 'Dashboard') {
      iconName = 'dashboard';
     } else if (route.name === 'Products') {
      iconName = 'inventory';
     }

     return <MaterialIcons name={iconName} size={size} color={color} />;
    },
    tabBarActiveTintColor: colors.primary,
    tabBarInactiveTintColor: colors.gray,
```

```
      headerShown: false
    })}
  >
    <Tab.Screen
      name="Dashboard"
      component={DashboardScreen}
      options={{ title: 'Tableau de bord' }}
    />
    <Tab.Screen
      name="Products"
      component={ProductListScreen}
      options={{ title: 'Produits' }}
    />
  </Tab.Navigator>
 );
};

// Navigateur de pile principal
const AppNavigator: React.FC = () => {
 return (
  <NavigationContainer>
   <Stack.Navigator
    screenOptions={{
     headerStyle: {
      backgroundColor: colors.white,
      elevation: 0,
      shadowOpacity: 0,
     },
```

```
    headerTintColor: colors.dark,
    headerTitleStyle: {
      fontWeight: '600',
    },
  }}
>
  <Stack.Screen
    name="MainTabs"
    component={MainTabNavigator}
    options={{ headerShown: false }}
  />
  <Stack.Screen
    name="ProductDetail"
    component={ProductDetailScreen}
    options={{
      title: 'Détail du produit',
      headerBackTitle: 'Retour'
    }}
  />
  <Stack.Screen
    name="ProductForm"
    component={ProductFormScreen}
    options={({ route }) => ({
      title: route.params?.product ? 'Modifier le produit' : 'Nouveau produit',
      headerBackTitle: 'Annuler'
    })}
  />
</Stack.Navigator>
```

```
    </NavigationContainer>
  );
};

export default AppNavigator;
```

Pour lancer l'application complète

```
# Terminal 1 - Backend avec toutes les routes
cd backend-crud-produits
npm run dev

# Terminal 2 - Frontend Expo
cd GestionProduitsMobile
npx expo start
```