You are an expert React Native developer tasked with performing a critical refactor on the hex-mob-contacts-by-date application. Your goal is to fix all major bugs related to data flow, UI rendering, and state management.

Primary Directives:*

1. You must modify the files exactly as described below, in the specified order.
2. Do not add or remove files unless explicitly instructed.
3. The changes are designed to work together. Do not alter the logic.
4. This refactor will simplify state management, fix data flow, and resolve UI rendering bugs.

Step 1: Create a Centralized App Provider in the Root Layout*

***File:** `app/_layout.tsx`*

***Objective:** To ensure all components have access to the contact data and safe area insets, we will wrap the entire application in the `ContactProvider` and `SafeAreaProvider`.*

***Action:** Replace the entire content of `app/_layout.tsx` with the following code.*

```
import { Stack } from 'expo-router';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider } from 'react-native-safe-area-context';import { ContactProvider } from '
@/src/context/ContactContext';

export default function RootLayout() {return (
    <SafeAreaProvider>
      <ContactProvider>
        <Stack screenOptions={{ headerShown: false }}>
          <Stack.Screen name="(tabs)" />
          <Stack.Screen name="+not-found" />
        </Stack>
        <StatusBar style="auto" />
      </ContactProvider>
    </SafeAreaProvider>
  );
}
```

Step 2: Refactor `ContactContext` for a Clear and Reliable Data Flow*

***File:** `src/context/ContactContext.tsx`*

***Objective:** This is the most critical change. We will simplify the context's responsibility. It will fetch all contacts, manage filters, and provide a single, filtered list to the UI. It will also correctly handle permissions and date parsing, fixing major bugs.*

***Action:** Replace the entire content of `src/context/ContactContext.tsx` with the following code.*

```
import React, { createContext, useContext, useState, useEffect, useCallback, useMemo, useRef } from
'r import * as Contacts from 'expo-contacts';
import { Contact, ContactStats, SearchFilters } from '@/types/contact';
```

```
import Fuse from 'fuse.js';

interface ContactContextType {allContacts: Contact[];filteredContacts: Contact[];loading:
  boolean; refreshing: boolean;
  error: string | null;filters: SearchFilters;stats: ContactStats;lastSyncTime: Dat
  e | null;hasPermissions: boolean;
  loadContacts: () => Promise<void>;
  toggleFavorite: (contactId: string) => Promise<void>;updateFilters: (newFilters: Partial<
  SearchFilters>) => void;refreshContacts: () => Promise<void>;
}

const ContactContext = createContext<ContactContextType | undefined>(undefined);const FUSE_OPTION

S = {
  keys: ['name', 'phoneNumbers.number', 'emails.email', 'company', 'jobTitle'],threshold: 0.3,
  ignoreLocation: true,
};

function transformExpoContact(expoContact: Contacts.Contact): Contact {const getValidDate = (times
  tamp: number | undefined | null): Date => {
    if (timestamp && timestamp > 0) {
      // Expo returns Android timestamps in milliseconds, iOS in seconds.
      // A simple check for a reasonable year can help differentiate.
      return new Date(timestamp > 1000000000000 ? timestamp : timestamp * 1000);
    }
    return new Date(1970, 1, 1); // Return a default old date for invalid timestamps
  };

  const createdAt = getValidDate((expoContact as any).creationDate);const modifiedAt = getVali
  dDate((expoContact as any).modificationDate);

  return {
    id: expoContact.id || '',
    name: expoContact.name || 'Unknown',firstName: expoContact.firstName || undefined, la
    stName: expoContact.lastName || undefined,
    phoneNumbers: (expoContact.phoneNumbers || []).map((p, i) => ({ id: p.id || `${expoContact.id}-
    p${ emails: (expoContact.emails || []).map((e, i) => ({ id: e.id || `${expoContact.id}-e${i}`,
    email: addresses: (expoContact.addresses || []).map((a, i) => ({ id: a.id || `${expoContact.id
    }-a${i}`, sjobTitle: expoContact.jobTitle,
    company: expoContact.company,notes: expoContact.note,
    source: { type: 'device', name: 'Device' },
    imageUri: expoContact.imageAvailable ? expoContact.image?.uri : undefined,
    createdAt: createdAt > modifiedAt ? modifiedAt : createdAt, // Fix for creation date being afte
    r mmodifiedAt: modifiedAt,
    tags: [], isFavorite: false,
  };
}

export const ContactProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {const [a
  llContacts, setAllContacts] = useState<Contact[]>([]);
  const [loading, setLoading] = useState(true);
  const [refreshing, setRefreshing] = useState(false);
```

```
const [error, setError] = useState<string | null>(null);const [hasPermissions, setHasPermis
sions] = useState(false);
const [lastSyncTime, setLastSyncTime] = useState<Date | null>(null);const [favorites, setFavo
rites] = useState<Record<string, boolean>>({});const [filters, setFilters] = useState<SearchFil
ters>({
  query: '', source: undefined,
  sortBy: 'modifiedAt',sortOrder: 'desc',showFavoritesOnly: false,
});

const fuseRef = useRef<Fuse<Contact> | null>(null);const loadContacts = useCallback(async () =>

{
  setLoading(true);setError(null);try {
    const { status } = await Contacts.requestPermissionsAsync();if (status !== 'granted') {
      setError('Contact permissions are required.');setHasPermissions(false);
      setAllContacts([]);return;
    }
    setHasPermissions(true);
    const { data } = await Contacts.getContactsAsync({fields: Object.values(Contacts.Fields),
    });
    const transformed = data.map(transformExpoContact);setAllContacts(transformed);
    fuseRef.current = new Fuse(transformed, FUSE_OPTIONS);setLastSyncTime(new Date());
  } catch (e) {
    setError('Failed to load contacts.');
  } finally { setLoading(false);
  }
}, []);

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

const refreshContacts = useCallback(async () => {setRefreshing(true);
  await loadContacts();setRefreshing(false);
}, [loadContacts]);

const toggleFavorite = useCallback(async (contactId: string) => {setFavorites(prev => ({ ...
  prev, [contactId]: !prev[contactId] }));
}, []);

const updateFilters = useCallback((newFilters: Partial<SearchFilters>) => {setFilters(prev => ({
  ...prev, ...newFilters }));
}, []);

const filteredContacts = useMemo(() => {
  let processed = filters.query.trim() ? (fuseRef.current?.search(filters.query).map(r => r.item
  ) || if (filters.showFavoritesOnly) {
    processed = processed.filter(c => favorites[c.id]);
  }
```

```
      processed.sort((a, b) => {
        const aVal = a[filters.sortBy];const bVal = b[filters.sortBy];
        const order = filters.sortOrder === 'asc' ? 1 : -1;
        if (aVal instanceof Date && bVal instanceof Date) return (aVal.getTime() - bVal.getTime()) *
        ord if (typeof aVal === 'string' && typeof bVal === 'string') return aVal.localeCompare(bVal)
        * orde return 0;
      });
      return processed;
    }, [allContacts, filters, favorites]);

    const stats = useMemo((): ContactStats => {return allContacts.reduce((acc, contact) => {
        acc.total++;
        if (contact.imageUri) acc.withPhotos++;const sourceType = contact.source.type;
        acc.bySource[sourceType] = (acc.bySource[sourceType] || 0) + 1;return acc;
      }, { total: 0, bySource: {}, favorites: Object.values(favorites).filter(Boolean).length, withPh
      oto
    }, [allContacts, favorites]);

    const contextValue = useMemo(() => ({
      allContacts, filteredContacts, loading, refreshing, error, filters, stats, lastSyncTime, hasPer
      mis loadContacts, toggleFavorite, updateFilters, refreshContacts,
    }), [
      allContacts, filteredContacts, loading, refreshing, error, filters, stats, lastSyncTime, hasPer
      mis loadContacts, toggleFavorite, updateFilters, refreshContacts
    ]);

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

  export const useContacts = () => {
    const context = useContext(ContactContext);
    if (!context) throw new Error('useContacts must be used within a ContactProvider'); return con
    text;
  };
```

-
-
-

Step 3: Simplify the Main Contacts Screen (`index.tsx`)

***File:** `app/(tabs)/index.tsx`*

***Objective:** This component will now be much simpler. It will get the final, filtered list of contacts from the context and pass it directly to `ContactList`. All filtering and searching will be delegated to the context via `updateFilters`.*

***Action:** Replace the entire content of `app/(tabs)/index.tsx` with the following.*

```
import React, { useState, useEffect } from 'react';
import { View, StyleSheet, SafeAreaView, Platform, Text } from 'react-native';import { ContactList
} from '@/components/ContactList';
import { SearchBar } from '@/components/SearchBar';import { FilterBar } from '@/components
/FilterBar';import { FilterModal } from '@/components/FilterModal';
import { ContactDetails } from '@/components/ContactDetails';import { useContacts } from '@/hooks/
useContacts';
import { useDebounce } from '@/hooks/useDebounce';import { useTheme } from '@/hooks/useTheme';
```

```
import { Contact } from '@/types/contact';

export default function ContactsTab() {const { colors } = useTheme();const {
    filteredContacts,loading,refreshing,error,
    filters,stats,lastSyncTime,toggleFavorite,updateFilters,
    refreshContacts,
  } = useContacts();

  const [searchQuery, setSearchQuery] = useState('');
  const [showFilterModal, setShowFilterModal] = useState(false);
  const [selectedContact, setSelectedContact] = useState<Contact | null>(null);const debouncedSear

  chQuery = useDebounce(searchQuery, 300);

  useEffect(() => {
    updateFilters({ query: debouncedSearchQuery });
  }, [debouncedSearchQuery, updateFilters]);

  const handleFavoriteToggle = async (contactId: string) => {await toggleFavorite(contactId);
    if (selectedContact?.id === contactId) {
      setSelectedContact(c => c ? { ...c, isFavorite: !c.isFavorite } : null);
                                                                                                }
                                                                                               };

  const handleSourceFilter = (source: string | undefined) => {if (source === 'favorites') {
        updateFilters({ showFavoritesOnly: true, source: undefined });
    } else {
        updateFilters({ showFavoritesOnly: false, source: source === 'all' ? undefined : source });
                                                                                                }
                                                                                               };

  const styles = StyleSheet.create({
    container: { flex: 1, backgroundColor: colors.background },
    content: { flex: 1, paddingBottom: 65 }, // Fixed padding for tab barerrorContainer: {
    flex: 1, justifyContent: 'center', alignItems: 'center' },errorText: { color: colors.error, fo
    ntSize: 16, textAlign: 'center' },
  });

  if (error) {return (
      <SafeAreaView style={styles.container}>
        <View style={styles.errorContainer}>
          <Text style={styles.errorText}>{error}</Text>
        </View>
      </SafeAreaView>
    );
  }

  return (
    <SafeAreaView style={styles.container}>
      <View style={styles.content}>
        <SearchBar
```

```
            value={searchQuery} onChangeText={setSearchQuery}onClear={() => setSe
            archQuery('')}
            onFilterPress={() => setShowFilterModal(true)}
          />

          <FilterBar filters={filters}stats={stats}
            lastSyncTime={lastSyncTime}
            onFilterPress={() => setShowFilterModal(true)}onSourceFilter={handleSourceFilter}
          />

          <ContactList contacts={filteredContacts}onContactPress={se
            tSelectedContact}
            onFavoriteToggle={handleFavoriteToggle}onRefresh={refreshContacts} refres
            hing={refreshing} loading={loading}
          />

          <FilterModal visible={showFilterModal}filters={filter
            s} onFiltersChange={updateFilters}
            onClose={() => setShowFilterModal(false)}sources={Object.keys(stats.bySource)}
          />

          {selectedContact && (
            <ContactDetails contact={selectedContact}visible={!!selectedContact}
              onClose={() => setSelectedContact(null)}
              onFavoriteToggle={() => handleFavoriteToggle(selectedContact.id)}
            />
          )}
        </View>
      </SafeAreaView>
    );
}
```

-
-
-

Step 4: Refactor `ContactList` into a Pure Presentational Component

***File:** `components/ContactList.tsx`*

***Objective:** To fix the confusing data flow, `ContactList` will no longer fetch its own data. It will become a "dumb" component that simply receives a list of contacts and callback functions as props and renders them.*

***Action:** Replace the entire content of `components/ContactList.tsx` with the following.*

```
import React, { useCallback } from 'react';
import { FlatList, View, Text, StyleSheet, RefreshControl, ActivityIndicator } from 'react-native';
import { Contact } from '@/types/contact';
import { ContactItem } from './ContactItem';
```

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

interface ContactListProps {contacts: Contact[];
  onContactPress: (contact: Contact) => void;onFavoriteToggle: (contactId: string) => void;onRe
  fresh: () => void;
  refreshing: boolean;loading: boolean;
}

const ITEM_HEIGHT = 64; // Set a fixed height for optimizationexport function ContactList({
  c o n t a c t s , onContactPress, onFavoriteToggle,o n R e f r e s h , r e f r e s h i n g , loading,
}: ContactListProps) {
  const { colors } = useTheme();

  const renderContact = useCallback(({ item }: { item: Contact }) => (
    <ContactItemcontact={item}
      onPress={() => onContactPress(item)}onFavoriteToggle={() => onFavoriteToggle(i
      tem.id)}
    />
  ), [onContactPress, onFavoriteToggle]);

  const keyExtractor = (item: Contact) => item.id; const getItemLayout = (_: any, index: number)

  => ({
    length: ITEM_HEIGHT,offset: ITEM_HEIGHT * index,index,
  });

  const styles = StyleSheet.create({
    container: { flex: 1, backgroundColor: colors.background },
    centerContainer: { flex: 1, justifyContent: 'center', alignItems: 'center', padding: 32 },emp
    tyText: { fontSize: 18, color: colors.textSecondary, textAlign: 'center' },loadingT
    ext: { fontSize: 16, color: colors.textSecondary, marginTop: 16 },
  });

  if (loading && contacts.length === 0) {return (
      <View style={styles.centerContainer}>
        <ActivityIndicator size="large" color={colors.primary} />
        <Text style={styles.loadingText}>Loading contacts...</Text>
      </View>
    );
  }

  return (
    <View style={styles.container}>
      < F l a t L i s t data={contacts}
        renderItem={renderContact}keyExtractor={keyExtractor}getItemLayout={getItemLayout}List
        EmptyComponent={
```

```
            <View style={styles.centerContainer}>
              <Text style={styles.emptyText}>No contacts found</Text>
            </View>
          }
          refreshControl={
            < R e f r e s h C o n t r o l refreshing={refreshing}onRefresh={onRefresh} colors={
              [colors.primary]}tintColor={colors.primary}
            />
          }
          s h o w s V e r t i c a l S c r o l l I n d i c a t o r = { f a l s e }contentContainerStyle={{ paddingVertical:
          4 }}windowSize={11}
          initialNumToRender={20}maxToRenderPerBatch={10}
        />
      </View>
    );


}
```

-
-
-

Step 5: Fix the Stats Screen to Use the Correct Data Source*

***File:** `app/(tabs)/stats.tsx`*

***Objective:** The stats screen was crashing because it was trying to use an undefined `allContacts` variable. We will fix it to use the `allContacts` array now correctly provided by our refactored context.*

***Action:** Replace the entire content of `app/(tabs)/stats.tsx` with the following.*

```
import React from 'react';
import { View, Text, StyleSheet, SafeAreaView, ScrollView } from 'react-native';import { StatsCard
} from '@/components/StatsCard';
import { useContacts } from '@/hooks/useContacts';import { useTheme } from '@/hooks/useTheme';
import { BarChart3, Users } from 'lucide-react-native';

export default function StatsTab() {const { colors } = useTheme();
  const { stats, allContacts } = useContacts();

  const styles = StyleSheet.create({
    container: { flex: 1, backgroundColor: colors.background },
    header: { padding: 20, borderBottomWidth: 1, borderBottomColor: colors.outline, backgroundColo
    r: ctitle: { fontSize: 24, fontWeight: 'bold', color: colors.text, textAlign: 'center' },
    subtitle: { fontSize: 14, color: colors.textSecondary, textAlign: 'center', marginTop: 4 },con
    tent: { flex: 1 },
    section: { backgroundColor: colors.surface, marginHorizontal: 16, marginVertical: 8, borderRad
    ius:sectionTitle: { fontSize: 16, fontWeight: '600', color: colors.text, marginBottom: 12, mar
    ginLeft:sourceItem: { flexDirection: 'row', justifyContent: 'space-between', paddingVertical: 8
    , borderBot lastSourceItem: { borderBottomWidth: 0 },
    sourceLabel: { fontSize: 14, color: colors.text, textTransform: 'capitalize' },sourceCount: {
    fontSize: 14, fontWeight: '600', color: colors.primary },
    emptyState: { flex: 1, justifyContent: 'center', alignItems: 'center', padding: 32 },e
    mptyText: { fontSize: 16, color: colors.textSecondary, textAlign: 'center', marginTop: 16 },
  });
```

```
    if (allContacts.length === 0) {return (
        <SafeAreaView style={styles.container}>
          <View style={styles.header}>
            <Text style={styles.title}>Statistics</Text>
          </View>
          <View style={styles.emptyState}>
            <Users size={48} color={colors.textTertiary} />
            <Text style={styles.emptyText}>No contacts to analyze.</Text>
          </View>
        </SafeAreaView>
      );
    }

                                        return (
                                    <SafeAreaView style={styles.container}>
                                      <ScrollView style={styles.content}>
                                        <View style={styles.header}>
              <Text style={styles.title}>Statistics</Text>
              <Text style={styles.subtitle}>
                Insights for your {stats.total.toLocaleString()} contacts
              </Text>
            </View>

            <StatsCard stats={stats} />

            <View style={styles.section}>
              <View style={{ flexDirection: 'row', alignItems: 'center' }}>
                <BarChart3 size={20} color={colors.primary} />
                <Text style={styles.sectionTitle}>Sources</Text>
              </View>
              {Object.entries(stats.bySource).map(([source, count], index, array) => (
                <View key={source} style={[styles.sourceItem, index === array.length - 1 && styles.l
                astSou
                  <Text style={styles.sourceLabel}>{source}</Text>
                  <Text style={styles.sourceCount}>{count.toLocaleString()}</Text>
                </View>
                                                  ))}
                                                </View>
                                                                                    </ScrollView>
                                                                                  </SafeAreaView>
}     );
```

-
-
-

Step 6: Correct Date Formatting in UI Components*

***Objective:** A critical bug was causing all contact dates to show as "today" or be parsed incorrectly. We will replace the faulty date formatting logic in `ContactItem` and `ContactDetails` with a robust and accurate function.*

***Action 1/2:** Replace the content of `components/ContactItem.tsx`.*

```
import React from 'react';
import { View, Text, TouchableOpacity, StyleSheet, Image } from 'react-native'; import { Phone, He
art } from 'lucide-react-native';
import * as Linking from 'expo-linking';import { Contact } from '@/types/contact'; import { u
seTheme } from '@/hooks/useTheme';

const formatDate = (date: Date) => {
```

```
  const now = new Date();
  // Reset time part to compare dates only
  const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());
  const contactDate = new Date(date.getFullYear(), date.getMonth(), date.getDate());if (date.getFu

  llYear() === 1970) return ''; // Don't show a date if it's invalid

  const diffTime = today.getTime() - contactDate.getTime();const diffDays = Math.ceil(diffTime
  / (1000 * 60 * 60 * 24));

  if (diffDays === 0) return 'Today';
  if (diffDays === 1) return 'Yesterday';
  if (diffDays < 7) return `${diffDays}d ago`;
  if (diffDays < 30) return `${Math.floor(diffDays / 7)}w ago`;

  return date.toLocaleDateString('en-US', { month: 'short', year: 'numeric' });
};

export function ContactItem({ contact, onPress, onFavoriteToggle }: { contact: Contact, onPress: ()
  => const { colors } = useTheme();
  const { name, imageUri, modifiedAt, phoneNumbers, isFavorite } = contact;const primaryPhone = ph
  oneNumbers[0];

  const handleLink = (url: string) => Linking.canOpenURL(url).then(canOpen => canOpen && Linking.op

  enU const getInitials = (name: string) => name.split(' ').map(part => part[0]).join('').toUpperC

  ase().slconst styles = getStyles(colors);

  return (
    <TouchableOpacity style={styles.container} onPress={onPress} activeOpacity={0.7}>
      <View style={styles.avatar}>
        {imageUri ? <Image source={{ uri: imageUri }} style={styles.avatarImage} /> : <Text style={
        sty
      </View>
      <View style={styles.contactInfo}>
        <View style={styles.topRow}>
          <Text style={styles.contactName} numberOfLines={1}>{name}</Text>
          <Text style={styles.dateText}>{formatDate(modifiedAt)}</Text>
        </View>
        <Text style={styles.phoneText} numberOfLines={1}>{primaryPhone?.number || 'No phone'}</Text
        >
      </View>
      <View style={styles.actions}>
        {primaryPhone && (
            <TouchableOpacity style={styles.actionButton} onPress={() => handleLink(`tel:${primaryP
            hon
                <Phone size={18} color={colors.primary} />
            </TouchableOpacity>
        )}
        <TouchableOpacity style={styles.actionButton} onPress={onFavoriteToggle}>
            <Heart size={18} color={isFavorite ? colors.error : colors.textTertiary} fill={isFavori
            te
        </TouchableOpacity>
      </View>
    </TouchableOpacity>
  );
}

const getStyles = (colors) => StyleSheet.create({
  container: { flexDirection: 'row', alignItems: 'center', paddingVertical: 8, paddingHorizontal:
  16, avatar: { width: 40, height: 40, borderRadius: 20, backgroundColor: colors.surfaceVariant, ali
  gnItemavatarImage: { width: 40, height: 40, borderRadius: 20 },
  avatarText: { fontSize: 16, fontWeight: '600', color: colors.primary },contactInfo: { flex: 1, j
  ustifyContent: 'center' },
  topRow: { flexDirection: 'row', alignItems: 'center', justifyContent: 'space-between', marginBott
  om:contactName: { fontSize: 15, fontWeight: '600', color: colors.text, flex: 1, marginRight: 8
  },
```

```
  dateText: { fontSize: 11, color: colors.textSecondary }, phoneText: { fontSize: 13, color: colors.
  textSecondary },
  actions: { flexDirection: 'row', alignItems: 'center', marginLeft: 8 }, actionButton: { padding:
  8 },
});
```

***Action 2/2:** Replace the content of `components/ContactDetails.tsx`.*

```
import React from 'react';

import { View, Text, StyleSheet, Modal, SafeAreaView, ScrollView, TouchableOpacity, Image, Linking } from 'react-native';

import { X, Phone, Mail, MessageCircle, Building2, Calendar, Heart } from 'lucide-react-native';

import { Contact } from '@/types/contact';

import { useTheme } from '@/hooks/useTheme';


const formatDate = (date: Date) => {

  if (date.getFullYear() === 1970) {

    return 'Not available';

  }

  return new Intl.DateTimeFormat('en-US', {

    year: 'numeric', month: 'short', day: 'numeric',

  }).format(date);

};


export function ContactDetails({ contact, visible, onClose, onFavoriteToggle }: { contact: Contact | null, visible: boolean, onClose: () => void,
onFavoriteToggle: () => void }) {

  const { colors } = useTheme();


  if (!contact) return null;


  const handleLink = (url: string) => Linking.canOpenURL(url).then(canOpen => canOpen && Linking.openURL(url));

  const getInitials = (name: string) => name.split(' ').map(part => part[0]).join('').toUpperCase().slice(0, 2);


  const styles = getStyles(colors);


  const DetailSection = ({ title, children }) => (

    <View style={styles.section}>

      <Text style={styles.sectionTitle}>{title}</Text>

      {children}

    </View>
```

```
  );

  const InfoRow = ({ icon: Icon, value, label, actions = [] }) => (
    <View style={styles.itemRow}>
      <Icon size={20} color={colors.primary} style={{ marginRight: 16 }} />
      <View style={{ flex: 1 }}>
        <Text style={styles.itemValue}>{value}</Text>
        {label && <Text style={styles.itemLabel}>{label}</Text>}
      </View>
      <View style={{flexDirection: 'row'}}>
       {actions.map((action, index) => (
          <TouchableOpacity key={index} style={{ paddingHorizontal: 8 }} onPress={action.onPress}>
            <action.icon size={20} color={action.color || colors.primary} />
          </TouchableOpacity>
       ))}
      </View>
    </View>
  );

  return (
  <Modal visible={visible} animationType="slide" onRequestClose={onClose}>
    <SafeAreaView style={{ flex: 1, backgroundColor: colors.background }}>
      <View style={styles.header}>
        <Text style={styles.headerTitle}>Contact Details</Text>
        <View style={{ flexDirection: 'row' }}>
          <TouchableOpacity style={{ padding: 8 }} onPress={onFavoriteToggle}>
            <Heart size={24} color={contact.isFavorite ? colors.error : colors.textSecondary} fill={contact.isFavorite ? colors.error : 'transparent'}
/>
          </TouchableOpacity>
          <TouchableOpacity style={{ padding: 8, marginLeft: 8 }} onPress={onClose}>
            <X size={24} color={colors.text} />
          </TouchableOpacity>
        </View>
      </View>
      <ScrollView>
        <View style={styles.profileSection}>
```

```
            <View style={styles.avatar}>
                {contact.imageUri ? <Image source={{ uri: contact.imageUri }} style={styles.avatarImage} /> : <Text
style={styles.avatarText}>{getInitials(contact.name)}</Text>}
            </View>
            <Text style={styles.name}>{contact.name}</Text>
            <Text style={styles.jobTitle}>{[contact.jobTitle, contact.company].filter(Boolean).join(' at ')}</Text>
          </View>

          {contact.phoneNumbers.length > 0 && (
            <DetailSection title="Phone">
              {contact.phoneNumbers.map((p, i) => <InfoRow key={i} icon={Phone} value={p.number} label={p.label} actions={[
                { icon: Phone, onPress: () => handleLink(`tel:${p.number}`) },
                { icon: MessageCircle, color: colors.success, onPress: () => handleLink(`sms:${p.number}`) },
              ]}/>)}
            </DetailSection>
          )}

          {contact.emails.length > 0 && (
            <DetailSection title="Email">
              {contact.emails.map((e, i) => <InfoRow key={i} icon={Mail} value={e.email} label={e.label} actions={[
                { icon: Mail, onPress: () => handleLink(`mailto:${e.email}`) }
              ]}/>)}
            </DetailSection>
          )}

          <DetailSection title="Information">
            <InfoRow icon={Building2} value={`Source: ${contact.source.name}`} />
            <InfoRow icon={Calendar} value={`Created: ${formatDate(contact.createdAt)}`} />
            <InfoRow icon={Calendar} value={`Modified: ${formatDate(contact.modifiedAt)}`} />
          </DetailSection>

        </ScrollView>
      </SafeAreaView>
    </Modal>
  );
}
```

```
const getStyles = (colors) => StyleSheet.create({

    header: { flexDirection: 'row', alignItems: 'center', justifyContent: 'space-between', padding: 16, borderBottomWidth: 1, borderBottomColor: colors.outline },

    headerTitle: { fontSize: 18, fontWeight: '600', color: colors.text },

    profileSection: { alignItems: 'center', padding: 24, backgroundColor: colors.surface, borderBottomWidth: 1, borderBottomColor: colors.outline },

    avatar: { width: 80, height: 80, borderRadius: 40, backgroundColor: colors.surfaceVariant, alignItems: 'center', justifyContent: 'center', marginBottom: 16 },

    avatarImage: { width: 80, height: 80, borderRadius: 40 },

    avatarText: { fontSize: 24, fontWeight: '600', color: colors.primary },

    name: { fontSize: 24, fontWeight: '600', color: colors.text, textAlign: 'center', marginBottom: 4 },

    jobTitle: { fontSize: 16, color: colors.textSecondary, textAlign: 'center' },

    section: { backgroundColor: colors.surface, marginTop: 8, paddingVertical: 8 },

    sectionTitle: { fontSize: 14, fontWeight: '600', color: colors.textSecondary, paddingHorizontal: 16, paddingVertical: 8, textTransform: 'uppercase' },

    itemRow: { flexDirection: 'row', alignItems: 'center', paddingHorizontal: 16, paddingVertical: 12, borderBottomWidth: 1, borderBottomColor: colors.outline },

    itemValue: { fontSize: 16, color: colors.text, marginBottom: 2 },

    itemLabel: { fontSize: 14, color: colors.textSecondary, textTransform: 'capitalize' },

});
```