BonJoy Driver Application

React Native Driver Onboarding & Management System

Architecture: Layered Clean Architecture with Service-Oriented Design

Project Overview

The BonJoy Driver Application is a comprehensive React Native mobile application designed for driver onboarding, document verification, and profile management. The app follows a complete driver lifecycle from registration to active status.

Key Features:

- Multi-step driver onboarding with OTP verification
- Document upload and verification system
- Vehicle registration and management
- Bank details submission
- Emergency contacts management
- Real-time status tracking
- Document rejection and resubmission workflow

Architecture Summary:

This application implements a **Layered (N-Tier) Architecture** combined with **Service-Oriented Architecture (SOA)** principles and multiple design patterns including Repository, Gateway, and Singleton patterns.

Technology Stack

Core Technologies

Framework & Language

React Native 0.82.1 | TypeScript 5.8.3 | React 19.1.1

Node.js: ≥20.0.0

Navigation

- React Navigation 7.x
- @react-navigation/native
- @react-navigation/stack
- @react-navigation/drawer
- @react-navigation/bottom-tabs

State & Performance

React Native Reanimated

React Native Gesture Handler

React Native Worklets

UI & Styling

UI Components

React Native Safe Area Context

React Native Screens

React Native Linear Gradient

React Native Country Picker

Media & Images

React Native Image Picker

@react-native-community/datetimepicker

Storage & Network

Async Storage

Axios

@react-native-community/netinfo

Development Tools

Build Tools

Metro Bundler

Babel

React Native CLI

Testing & Quality

Jest

ESLint

Prettier

TypeScript Config

Development

React Native DevTools

Hot Reloading

TypeScript Strict Mode

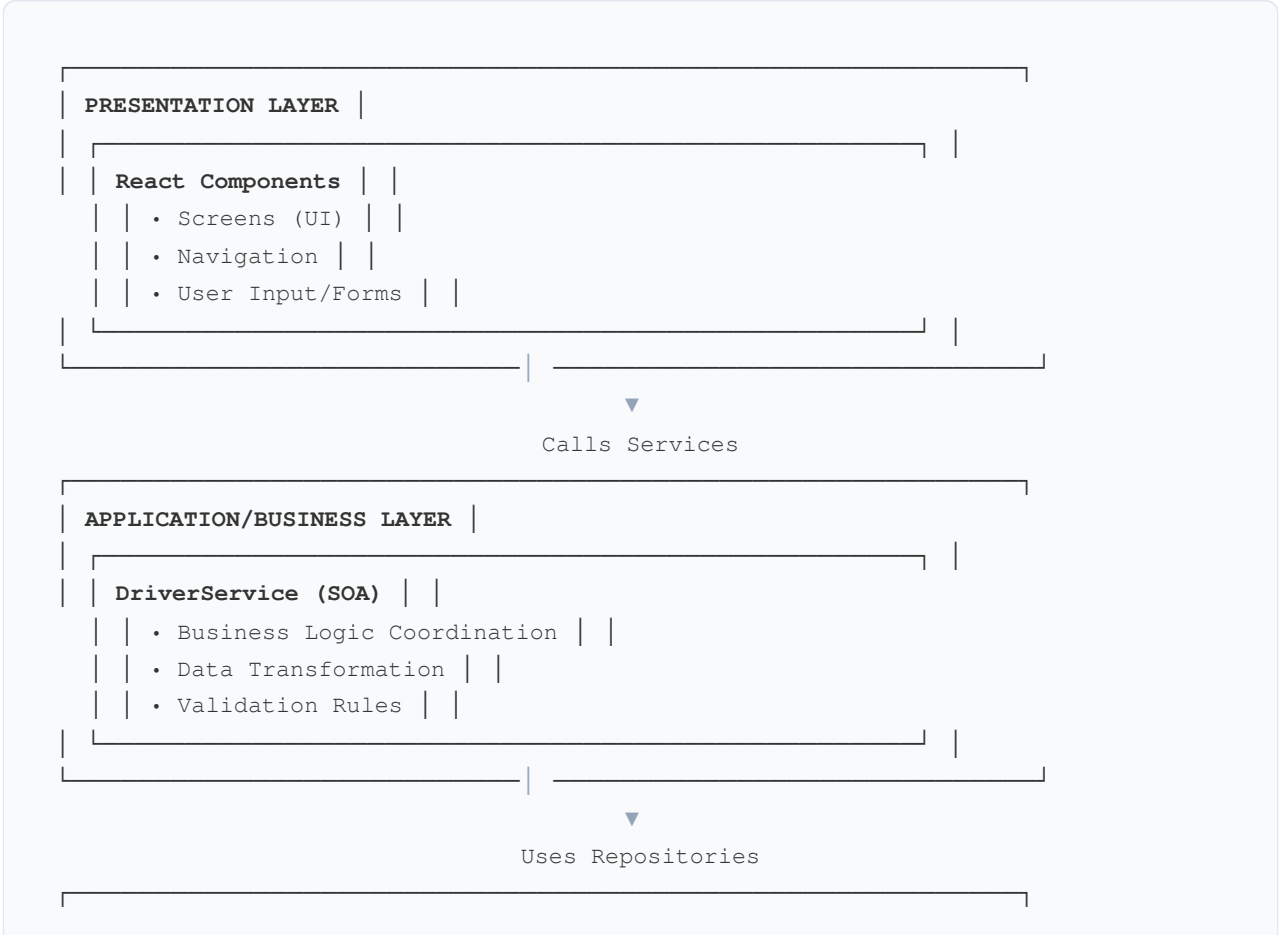
Package.json Dependencies

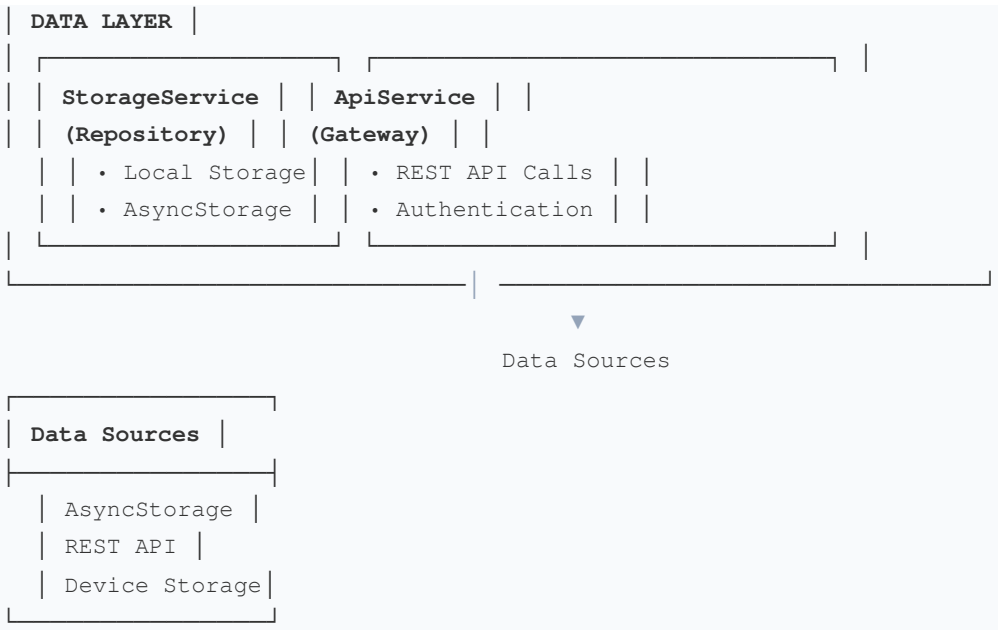
```
{ "dependencies": { "@react-native-async-storage/async-storage": "^1.23.1", "@react-native-community/datetimepicker": "^8.5.1", "@react-native-community/netinfo": "^11.4.1", "@react-navigation/bottom-tabs": "^7.8.11", "@react-navigation/drawer": "^7.7.8", "@react-navigation/native": "^7.1.24", "@react-navigation/native-stack": "^7.8.5", "@react-navigation/stack": "^7.6.11", "axios": "^1.13.2", "react": "19.1.1", "react-native": "0.82.1", "react-native-country-picker-modal": "^2.0.0", "react-native-gesture-handler": "^2.29.1", "react-native-image-picker": "^8.2.1", "react-native-linear-gradient": "^2.8.3", "react-native-reanimated": "^4.2.0", "react-native-safe-area-context": "^5.6.2", "react-native-screens": "^4.18.0", "react-native-worklets": "^0.7.1" }, "devDependencies": { "@babel/core": "^7.25.2", "@babel/preset-env": "^7.25.3", "@babel/runtime": "^7.25.0", "@react-native/babel-preset": "0.82.1", "@react-native/eslint-config": "0.82.1", "@react-native/metro-config": "0.82.1", "@react-native/typescript-config": "0.82.1", "@types/react": "^19.1.1", "eslint": "^8.19.0", "jest": "^29.6.3", "metro-react-native-babel-preset": "^0.77.0", "prettier": "2.8.8", "typescript": "^5.8.3" }, "engines": { "node": ">=20" } }
```


Architecture Analysis

Architecture Type: Layered Clean Architecture with SOA

The application implements a **Modern React Native Layered Architecture** combining Clean Architecture principles with Service-Oriented Architecture patterns.





Design Patterns Implemented

Repository Pattern

StorageService | Data Abstraction

Abstracts data access layer from business logic

```
class StorageService { async getDriverProfile(): Promise<DriverProfile | null> async setDriverSession(session: DriverSession): Promise<void> }
```

Service Layer Pattern

DriverService | Business Logic

Orchestrates multiple repositories and services

```
class DriverService { async createOrUpdateProfile(profileData, file?) async getAllDriverDocuments(driverId) }
```

Gateway Pattern

ApiService | API Abstraction

Acts as gateway to external REST API

```
class ApiService { private async request<T>(method, url, data?) // Handles auth, headers, errors }
```

Singleton Pattern

ApiService.getInstance() | Single Instance

Ensures single instance of API service

```
static getInstance(): ApiService { if (!ApiService.instance) { ApiService.instance = new ApiService(); } return ApiService.instance; }
```

Architecture Layers Detail

Presentation Layer

Location: `src/screens/`
Responsibility: UI Components, User Interaction
Pattern: Container/Presenter Pattern
Key Files: 16+ screen components

Application Layer

Location: `src/services/driverService.ts`
Responsibility: Business Logic Coordination
Pattern: Service-Oriented Architecture
Key Classes: DriverService

Data Layer

Location: `src/services/`
Responsibility: Data Access & Persistence
Pattern: Repository + Gateway Patterns
Key Classes: StorageService, ApiService

Infrastructure Layer

Location: `src/utils/`
Responsibility: Cross-cutting Concerns
Components: Scaling, Fonts, Formatters
Files: scale.ts, fontFamily.ts

Data Flow Pattern

1. **User Action**
   User interacts with UI component (form submit, button click)
2. **Screen Handler**
   Screen component calls service method with data
3. **Service Processing**
   DriverService validates data, applies business rules
4. **Repository Access**
   Service calls StorageService (local) or ApiService (remote)
5. **Data Source**
   AsyncStorage (local) or REST API (remote) processes request
6. **Response Flow Back**
   Response travels back through layers to UI
7. **UI Update**
   Screen updates state and re-renders UI

State Management Approach

Pattern: Local Component State + Service Layer

Instead of global state management (Redux/MobX), the application uses:

- **Local Component State:** useState for form data and UI state
- **Service Layer Cache:** AsyncStorage for API data persistence
- **Navigation Params:** Screen-to-screen data passing
- **Context:** Limited use for theme/authentication (if added)

Error Handling Strategy

```
// Centralized in ApiService interceptors this.api.interceptors.response.use( response => { // Success handling return response; }, async (error: AxiosError) => { console.error('API Error:', error.response?.data || error.message); // Centralized error handling if (error.response?.status === 401) { await this.storage.clearAll(); // Auto-logout on 401 } return Promise.reject(error); } );
```

Architecture Strengths

- **Maintainability:** Clear separation makes code easy to maintain and modify
- **Testability:** Each layer can be tested independently (unit tests for services, integration tests for screens)
- **Scalability:** Easy to add new features by extending services or adding new screens
- **Reusability:** Services and utilities are highly reusable across the application
- **Type Safety:** Full TypeScript support with interfaces for all data structures
- **Separation of Concerns:** Clear boundaries between UI, business logic, and data access

Potential Architecture Improvements

- **Dependency Injection Container:** Could improve testability and decoupling
- **State Management Library:** For complex shared state across screens
- **Caching Layer:** For API responses with expiration policies
- **Event Bus:** For cross-component communication without tight coupling
- **Middleware Layer:** For request/response transformation and logging
- **Feature-based Structure:** Organize by feature rather than layer for larger apps

File Structure Overview

```
src/
   ├──  navigation/ # Navigation configuration
      ├──  RootNavigator.tsx # Main stack navigator
      ├──  HomeNavigator.tsx # Drawer navigator
```

```
    ├── HomeStack.tsx # Tab navigator stack
    └── types.ts # TypeScript navigation types
├── screens/ # All application screens (16+)
    ├── Authentication/ # Login, OTP screens
    ├── Onboarding/ # Profile, document upload
    ├── HomeScreenContent/ # Dashboard, profile, contacts
    ├── BottomContainer/ # Tab screen components
    └── StatusScreens/ # Status, rejection screens
├── services/ # Business logic layer
    ├── driverService.ts # Main service class
    ├── storageService.ts # Repository pattern implementation
    ├── apiService.ts # Gateway pattern implementation
    └── types/ # TypeScript interfaces
├── utils/ # Infrastructure utilities
    ├── scale.ts # Responsive scaling functions
    ├── fontFamily.ts # Font management utilities
    ├── constants.ts # App constants
    └── helpers/ # Helper functions
├── components/ # Reusable UI components
    ├── ProfileRow.tsx # Profile information row
    ├── DocumentCard.tsx # Document display card
    └── ContactItem.tsx # Contact list item
├── App.tsx # Main app component
└── index.tsx # App entry point
```

Navigation System

The application uses a complex navigation structure with multiple navigator types:

Navigation Hierarchy

```
RootNavigator (Stack) ├── SplashScreen (automatic routing) ├── LoginScreen (OTP verification) ├── OnboardingScreen (profile setup) ├── VerifyIdentityScreen (document upload) ├── AddVehicleDetailsScreen (vehicle registration) ├── AddBankDetailsScreen (bank details) ├── StatusScreen (application status) ├── HomeNavigator (Drawer) ├── HomeTabs (Bottom Tabs) ├── HomeStack (Stack) ├── HomeMain (Dashboard) ├── EmergencyContacts ├── ProfileScreen ├── EditProfilePage ├── MyDocument ├── AddEmergencyContactScreen ├── EmergencyContactDetailScreen ├── Bike (TODO) ├── Profile (ProfileScreen) ├── Wallet (TODO) └── Support (TODO)
```

Navigation Types

Root Navigator

Type: Native Stack Navigator
Purpose: Main authentication and onboarding flow
Screens: 9 primary screens

Home Navigator

Type: Drawer Navigator
Purpose: Main app navigation after login
Features: Side menu with profile

Config: Header hidden, gesture disabled

Custom Component: DrawerContent

Home Tabs

Type: Bottom Tab Navigator
Purpose: Quick access to main features
Tabs: 5 tabs with custom icons
Config: Active/inactive icon colors

Home Stack

Type: Native Stack Navigator
Purpose: Nested navigation within Home tab
Screens: 7 interconnected screens
Config: Custom headers, back buttons

Navigation Configuration Example

```
// RootNavigator.tsx const RootStack = createNativeStackNavigator<RootStackParamList>(); export const RootNavigator = () => { return ( <RootStack.Navigator initialRouteName="Splash" screenOptions={{ headerShown: false, gestureEnabled: false }} > <RootStack.Screen name="Splash" component={SplashScreen} /> <RootStack.Screen name="Login" component={LoginScreen} /> <RootStack.Screen name="Onboarding" component={OnboardingScreen} /> <RootStack.Screen name="VerifyIdentity" component={VerifyIdentityScreen} /> <RootStack.Screen name="AddVehicleDetails" component={AddVehicleDetailsScreen} /> <RootStack.Screen name="AddBankDetails" component={AddBankDetailsScreen} /> <RootStack.Screen name="StatusScreen" component={StatusScreen} /> <RootStack.Screen name="Home" component={HomeNavigator} /> <RootStack.Screen name="ResubmitScreen" component={ResubmitScreen} /> </RootStack.Navigator> ); }; // Navigation types for TypeScript safety export type RootStackParamList = { Splash: undefined; Login: undefined; Onboarding: undefined; VerifyIdentity: undefined; AddVehicleDetails: undefined; AddBankDetails: undefined; StatusScreen: undefined; Home: undefined; ResubmitScreen: undefined; };
```

Drawer Navigation Configuration

```
// HomeNavigator.tsx const Drawer = createDrawerNavigator(); export const HomeNavigator = () => { return ( <Drawer.Navigator drawerContent={props => <DrawerContent {...props} />} screenOptions={{ headerShown: false, drawerStyle: { width: 300 }, drawerType: 'front', }} > <Drawer.Screen name="HomeTabs" component={HomeTabs} options={{ title: 'Home' }} /> </Drawer.Navigator> ); }; // Custom Drawer Component const DrawerContent = ({ navigation }: DrawerContentProps) => { const [profile, setProfile] = useState<DriverProfile | null>(null); useEffect(() => { loadProfile(); }, []); const loadProfile = async () => { const profileData = await driverService.getDriverProfile(); setProfile(profileData); }; const handleLogout = async () => { await driverService.logout(); navigation.navigate('Login'); }; return ( <View style={styles.container}> {/* Profile Section */} <View style={styles.profileSection}> <Image source={{ uri: getFullImageUrl(profile?.profileImage) }} style={styles.profileImage} /> <Text style={styles.profileName}>{profile?.fullName}</Text> <Text style={styles.profileEmail}>{profile?.email}</Text> </View> {/* Menu Items */} <DrawerItemList {...props} /> {/* Logout Button */} <TouchableOpacity onPress={handleLogout} style={styles.logoutButton}> <Text style={styles.logoutText}>Logout</Text> </TouchableOpacity> </View> ); };
```

Tab Navigation Configuration

```
// HomeTabs.tsx const Tab = createBottomTabNavigator<HomeTabParamList>(); export const HomeTabs = () => { return ( <Tab.Navigator screenOptions={({ route }) => ({ headerShown: false, tabBarIcon: ({ focused, color, size }) => { let iconName: string; if (route.name === 'HomeStack') { iconName = focused ? 'home' : 'home-outline'; } else if (route.name === 'Bike') { iconName = focused ? 'bicycle' : 'bicycle-outline'; } else if (route.name === 'Profile') { iconName = focused ? 'person' : 'person-outline'; } else if (route.name === 'Wallet') { iconName = focused ? 'wallet' : 'wallet-outline'; } else { iconName = focused ? 'help-circle' : 'help-circle-outline'; } return <Ionicons name={iconName} size={size} color={color} />; }, tabBarActiveTintColor: '#FFC533', tabBarInactiveTintColor: '#94A3B8', tabBarStyle: { backgroundColor: '#FFFFFF', borderTopWidth: 1, borderTopColor: '#E2E8F0', height: 60, paddingBottom: 10, paddingTop: 10, }, tabBarLabelStyle: { fontSize: 12, fontWeight: '500', }, })} > <Tab.Screen name="HomeStack" component={HomeStack} options={{ title: 'Home' }} /> <Tab.Screen name="Bike" component={BikeScreen} options={{ title: 'Bike' }} /> <Tab.Screen name="Profile" component={ProfileScreen} options={{ title: 'Profile' }} /> <Tab.Screen name="Wallet" component={WalletScreen} options={{ title: 'Wallet' }} /> <Tab.Screen name="Support" component={SupportScreen} options={{ title: 'Support' }} /> </Tab.Navigator> ); };
```

Navigation Best Practices

Key Navigation Patterns:

- **Type Safety:** All navigators use TypeScript param lists
- **Screen Options:** Consistent styling across all navigators
- **Lazy Loading:** Screens are lazy-loaded for performance
- **Gesture Control:** Disabled in onboarding, enabled in home flow
- **Back Handling:** Custom back button behavior in critical flows
- **Deep Linking:** URL-based navigation support

Screens & Components

Core Screens

Screen	Purpose	Key Features	Navigation
SplashScreen	Initial screen with auto-login check	Auto-routing, session validation, animation	Root → Home/Login

LoginScreen	Mobile OTP verification	Country picker, OTP input, resend timer	Root → Onboarding
OnboardingScreen	Driver profile creation	Form validation, date picker, mandatory fields	Root → VerifyIdentity
VerifyIdentityScreen	Document upload (PAN, Aadhaar, DL)	Image picker, document validation, multiple uploads	Root → AddVehicle
AddVehicleDetailsScreen	Vehicle registration	Brand/model selection, multiple images, expiry dates	Root → AddBankDetails
AddBankDetailsScreen	Bank account setup	IFSC validation, document upload, form validation	Root → StatusScreen
StatusScreen	Application status display	Rejection handling, thank you messages	Root → Home/Resubmit

Home Screens

Screen	Purpose	Key Features	Location
HomeScreen	Main dashboard	Map view, destination search, drawer menu	screens/HomeScreenContent
ProfileScreen	Driver profile view	Profile display, edit navigation, data formatting	screens/HomeScreenContent
EditProfilePage	Profile editing	Image upload, form validation, unsaved changes warning	screens/HomeScreenContent
MyDocument	Document management	Document gallery, status display, fullscreen view	screens/HomeScreenContent

EmergencyContacts	Contact management	Contact list, primary contact, add/edit/delete	screens/HomeScreenContent
AddEmergencyContactScreen	Add new contact	Form validation, relationship dropdown, primary toggle	screens/HomeScreenContent
EmergencyContactDetailScreen	Contact details	Contact info, set primary, delete with confirmation	screens/HomeScreenContent

Reusable Components

ProfileRow Component

Reusable profile information row component used in ProfileScreen

```
interface ProfileRowProps { label: string; value: string | null | undefined; multiline?: boolean; icon?: React.ReactNode; } const ProfileRow: React.FC<ProfileRowProps> = ({ label, value, multiline = false, icon }) => { return ( <View style={styles.container}> <Text style={styles.label}>{label} </Text> <View style= {styles.valueContainer}> {icon && <View style={styles.icon}>{icon}</View>} <Text style={[styles.value, multiline && styles.multiline]} numberOfLines= {multiline ? undefined : 1} > {value || 'Not provided'} </Text> </View> </View> ); };
```

DocumentCard Component

Reusable document display cards with status badges

```
interface DocumentCardProps { title: string; status: 'approved' | 'pending' | 'rejected'; documentUrl?: string; uploadedDate?: string; onPress?: () => void; onRetry?: () => void; } const DocumentCard: React.FC<DocumentCardProps> = ({ title, status, documentUrl, uploadedDate, onPress, onRetry }) => { const getStatusColor = () => { switch(status) { case 'approved': return '#10B981'; case 'pending': return '#F59E0B'; case 'rejected': return '#EF4444'; default: return '#94A3B8'; } }; return ( <TouchableOpacity style={styles.card} onPress={onPress}> <View style= {styles.header}> <Text style= {styles.title}>{title}</Text> <View style={[styles.statusBadge, {backgroundColor: getStatusColor()}]}> <Text style={styles.statusText}>{status} </Text> </View> </View> {/* Content and actions */} </TouchableOpacity> ); };
```

ContactItem Component

Reusable contact list items with avatars and actions

```
interface ContactItemProps { contact: UserContact; isPrimary: boolean; onPress: () => void; onCall: () => void; onMessage: () => void; } const ContactItem: React.FC<ContactItemProps> = ({ contact, isPrimary, onPress, onCall, onMessage }) => { const initials = `${contact.first_name?.[0] || ''}${contact.last_name?.[0] || ''}`; return ( <TouchableOpacity style={styles.container} onPress={onPress}> <View style={styles.avatar}> <Text style={styles.initials}>{initials}</Text> </View> <View style={styles.content}> <Text style={styles.name}> {`${contact.first_name} ${contact.last_name}`} </Text> <Text style={styles.phone}> {contact.phone_number}</Text> <Text style={styles.relationship}> {contact.relationship}</Text> {isPrimary && ( <View style={styles.primaryBadge}> <Text style={styles.primaryText}>Primary</Text> </View> )} </View> <View style={styles.actions}> <TouchableOpacity onPress={onCall} style={styles.actionButton}> <Ionicons name="call" size={20} color="#3B82F6" /> </TouchableOpacity> <TouchableOpacity onPress={onMessage} style={styles.actionButton}> <Ionicons name="chatbubble" size={20} color="#10B981" /> </TouchableOpacity> </View> </TouchableOpacity> ); };
```

Screen Implementation Patterns

Common Screen Structure:

- **State Management:** useState for form data, loading states
- **Lifecycle:** useEffect for data fetching on mount
- **Navigation:** useNavigation and useRoute hooks
- **Form Handling:** Controlled components with validation
- **Image Upload:** react-native-image-picker integration
- **Error Handling:** try-catch with user-friendly messages
- **Loading States:** ActivityIndicator during API calls

Services & API Layer

DriverService Class - Complete Implementation

```
class DriverService { private storage: StorageService; private api: ApiService;
constructor() { this.storage = new StorageService(); this.api =
ApiService.getInstance(); } // ============ AUTHENTICATION ============ async
login(mobile: string): Promise<LoginDriverResponse> { const response = await
this.api.post<LoginDriverResponse>('/loginDriver', { mobile:
this.formatMobileNumber(mobile) }); return response.data; } async verifyOtp(mobile:
string, otp: string): Promise<DriverSession> { const response = await
this.api.post<DriverSessionResponse>('/verifyOtpAndLoginDriver', { mobile:
this.formatMobileNumber(mobile), otp, ip_address: await this.getIpAddress() });
const session = response.data; await this.storage.setAuthToken(session.token);
await this.storage.setDriverSession(session); return session; } async logout():
Promise<void> { await this.storage.clearAll(); } // ============ PROFILE MANAGEMENT
============ async createOrUpdateProfile( profileData: CreateProfileRequest, file?:
ImagePickerResponse ): Promise<DriverProfile> { const formData =
DriverService.createFormData(profileData, file); const response = await
this.api.post<DriverProfile>('/createDriverProfile', formData, { headers: {
'Content-Type': 'multipart/form-data' } }); await
this.storage.setDriverProfile(response.data); return response.data; } async
getProfile(driverId: number): Promise<DriverProfile> { const cached = await
this.storage.getDriverProfile(); if (cached) return cached; const response = await
this.api.get<DriverProfileResponse>(`/getDriverProfileById/${driverId}`); const
profile = response.data.DriverProfile[0]; await
this.storage.setDriverProfile(profile); return profile; } // ============ DOCUMENT
MANAGEMENT ============ async uploadDocument( data: DocumentUploadRequest, file:
ImagePickerResponse ): Promise<DriverDocument> { const formData =
DriverService.createFormData(data, file); const response = await
this.api.post<DriverDocument>('/createDriverDocument', formData, { headers: {
'Content-Type': 'multipart/form-data' } }); await this.invalidateDocumentsCache();
return response.data; } async getAllDriverDocuments(driverId: number):
Promise<DriverAllDocumentsResponse> { const cached = await
this.storage.getDriverDocuments(); if (cached) return cached; const response =
await this.api.get<DriverAllDocumentsResponse>( `/getDriverAllDocument/${driverId}`
); await this.storage.setDriverDocuments(response.data); return response.data; } //
============ VEHICLE MANAGEMENT ============ async createVehicle(data:
CreateVehicleRequest): Promise<Vehicle> { const response = await
this.api.post<Vehicle>('/createDriverVehicle', data); return response.data; } async
uploadVehicleDocument( data: VehicleDocumentRequest, file: ImagePickerResponse,
expiry_date?: string ): Promise<VehicleDocument> { const formData =
DriverService.createFormData({ ...data, expiry_date }, file); const response =
await this.api.post<VehicleDocument>('/createDriverVehicleDocument', formData, {
headers: { 'Content-Type': 'multipart/form-data' } }); return response.data; } //
============ BANK MANAGEMENT ============ async createBankDetails(data:
CreateBankDetailsRequest): Promise<DriverBankDetails> { const response = await
this.api.post<DriverBankDetails>('/createDriverBankDetails', data); return
response.data; } async uploadBankDocument( driverId: number, file:
ImagePickerResponse, file_label: string ): Promise<BankDocument> { const formData =
DriverService.createFormData({ driverId, file_label }, file); const response =
```

```
await this.api.post<BankDocument>('/createDriverBankDocument', formData, { headers: { 'Content-Type': 'multipart/form-data' } }); return response.data; } // ============ CONTACTS MANAGEMENT ============ async getAllUserContacts(): Promise<UserContact[]> { const cached = await this.storage.getUserContacts(); if (cached && cached.length > 0) return cached; const response = await this.api.get<UserContact[]>('/getAllUserContacts'); await this.storage.saveUserContacts(response.data); return response.data; } async createUserContact(data: CreateUserContactRequest): Promise<UserContact> { const response = await this.api.post<UserContact>('/createUserContact', data); const contacts = await this.getAllUserContacts(); await this.storage.saveUserContacts([...contacts, response.data]); return response.data; } async updateUserContact(id: number, data: UpdateUserContactRequest): Promise<UserContact> { const response = await this.api.put<UserContact>(`/updateUserContact/${id}`, data); const contacts = await this.getAllUserContacts(); const updatedContacts = contacts.map(contact => contact.id === id ? response.data : contact ); await this.storage.saveUserContacts(updatedContacts); return response.data; } async deleteUserContact(id: number): Promise<{success: boolean, message: string}> { const response = await this.api.delete<DeleteContactResponse>(`/deleteUserContact/${id}`); if (response.data.success) { const contacts = await this.getAllUserContacts(); const filteredContacts = contacts.filter(contact => contact.id !== id); await this.storage.saveUserContacts(filteredContacts); } return response.data; } // ============ HELPER METHODS ============ static getFullImageUrl(path?: string | null): string { if (!path) return ''; if (path.startsWith('http')) return path; return `${IMAGE_BASE_URL}${path}`; } static createFormData(data: Record<string, any>, file?: any): FormData { const formData = new FormData(); Object.keys(data).forEach(key => { if (data[key] !== undefined && data[key] !== null) { formData.append(key, String(data[key])); } }); if (file) { if (file.uri) { const filename = file.uri.split('/').pop(); const match = /\.(\w+)$/.exec(filename || ''); const type = match ? `image/${match[1]}` : 'image/jpeg'; formData.append('document', { uri: file.uri, type, name: filename || 'document.jpg', } as any); } } return formData; } private async invalidateDocumentsCache(): Promise<void> { await this.storage.setDriverDocuments(null); } private formatMobileNumber(mobile: string): string { return mobile.replace(/\D/g, ''); } private async getIpAddress(): Promise<string> { try { const networkState = await NetInfo.fetch(); return networkState.details.ipAddress || '0.0.0.0'; } catch { return '0.0.0.0'; } } }
```

StorageService Class

```
class StorageService { private static readonly AUTH_TOKEN_KEY = '@BonJoyDriver:authToken'; private static readonly DRIVER_SESSION_KEY = '@BonJoyDriver:driverSession'; private static readonly DRIVER_PROFILE_KEY = '@BonJoyDriver:driverProfile'; private static readonly DRIVER_DOCUMENTS_KEY = '@BonJoyDriver:driverDocuments'; private static readonly USER_CONTACTS_KEY = '@BonJoyDriver:userContacts'; private static readonly ONBOARDING_STATUS_KEY = '@BonJoyDriver:onboardingStatus'; // ============ AUTHENTICATION ============ async setAuthToken(token: string): Promise<void> { try { await AsyncStorage.setItem(StorageService.AUTH_TOKEN_KEY, token); } catch (error) { console.error('Error saving auth token:', error); } } async getAuthToken(): Promise<string | null> { try { return await AsyncStorage.getItem(StorageService.AUTH_TOKEN_KEY); } catch (error) { console.error('Error getting auth token:', error); return null; } } async
```

```
isAuthenticated(): Promise<boolean> { const token = await this.getAuthToken(); return !!token; } // ============ SESSION MANAGEMENT ============ async setDriverSession(session: DriverSession): Promise<void> { try { await AsyncStorage.setItem( StorageService.DRIVER_SESSION_KEY, JSON.stringify(session) ); } catch (error) { console.error('Error saving driver session:', error); } } async getDriverSession(): Promise<DriverSession | null> { try { const sessionJson = await AsyncStorage.getItem(StorageService.DRIVER_SESSION_KEY); return sessionJson ? JSON.parse(sessionJson) : null; } catch (error) { console.error('Error getting driver session:', error); return null; } } // ============ PROFILE MANAGEMENT ============ async setDriverProfile(profile: DriverProfile): Promise<void> { try { await AsyncStorage.setItem( StorageService.DRIVER_PROFILE_KEY, JSON.stringify(profile) ); } catch (error) { console.error('Error saving driver profile:', error); } } async getDriverProfile(): Promise<DriverProfile | null> { try { const profileJson = await AsyncStorage.getItem(StorageService.DRIVER_PROFILE_KEY); return profileJson ? JSON.parse(profileJson) : null; } catch (error) { console.error('Error getting driver profile:', error); return null; } } // ============ DOCUMENTS MANAGEMENT ============ async setDriverDocuments(documents: DriverAllDocumentsResponse | null): Promise<void> { try { if (documents) { await AsyncStorage.setItem( StorageService.DRIVER_DOCUMENTS_KEY, JSON.stringify(documents) ); } else { await AsyncStorage.removeItem(StorageService.DRIVER_DOCUMENTS_KEY); } } catch (error) { console.error('Error saving driver documents:', error); } } async getDriverDocuments(): Promise<DriverAllDocumentsResponse | null> { try { const documentsJson = await AsyncStorage.getItem(StorageService.DRIVER_DOCUMENTS_KEY); return documentsJson ? JSON.parse(documentsJson) : null; } catch (error) { console.error('Error getting driver documents:', error); return null; } } // ============ CONTACTS MANAGEMENT ============ async saveUserContacts(contacts: UserContact[]): Promise<void> { try { await AsyncStorage.setItem( StorageService.USER_CONTACTS_KEY, JSON.stringify(contacts) ); } catch (error) { console.error('Error saving user contacts:', error); } } async getUserContacts(): Promise<UserContact[]> { try { const contactsJson = await AsyncStorage.getItem(StorageService.USER_CONTACTS_KEY); return contactsJson ? JSON.parse(contactsJson) : []; } catch (error) { console.error('Error getting user contacts:', error); return []; } } // ============ ONBOARDING STATUS ============ async setOnboardingStatus(status: OnboardingStatus): Promise<void> { try { await AsyncStorage.setItem( StorageService.ONBOARDING_STATUS_KEY, JSON.stringify(status) ); } catch (error) { console.error('Error saving onboarding status:', error); } } async getOnboardingStatus(): Promise<OnboardingStatus | null> { try { const statusJson = await AsyncStorage.getItem(StorageService.ONBOARDING_STATUS_KEY); return statusJson ? JSON.parse(statusJson) : null; } catch (error) { console.error('Error getting onboarding status:', error); return null; } } // ============ UTILITY METHODS ============ async clearAll(): Promise<void> { try { const keys = [ StorageService.AUTH_TOKEN_KEY, StorageService.DRIVER_SESSION_KEY, StorageService.DRIVER_PROFILE_KEY, StorageService.DRIVER_DOCUMENTS_KEY, StorageService.USER_CONTACTS_KEY, StorageService.ONBOARDING_STATUS_KEY, ]; await AsyncStorage.multiRemove(keys); } catch (error) { console.error('Error clearing storage:', error); await AsyncStorage.clear(); } } async getMultipleItems(keys: string[]): Promise<Record<string, any>> { try { const values = await AsyncStorage.multiGet(keys); const result: Record<string, any> = {}; values.forEach(([key, value]) => { if (value !== null) { result[key] = JSON.parse(value); } }); return result; } catch (error) { console.error('Error getting multiple items:', error); return {}; } } }
```

ApiService Class (Gateway Pattern)

```
class ApiService { private static instance: ApiService; private api: AxiosInstance; private storage: StorageService; private constructor() { this.storage = new StorageService(); this.api = axios.create({ baseURL: API_BASE_URL, timeout: 30000, headers: { 'Content-Type': 'application/json', 'Accept': 'application/json', }, }); this.setupInterceptors(); } static getInstance(): ApiService { if (!ApiService.instance) { ApiService.instance = new ApiService(); } return ApiService.instance; } private setupInterceptors(): void { // Request interceptor for adding auth token this.api.interceptors.request.use( async (config) => { const token = await this.storage.getAuthToken(); if (token) { config.headers.Authorization = `Bearer ${token}`; } return config; }, (error) => { return Promise.reject(error); } ); // Response interceptor for error handling this.api.interceptors.response.use( (response) => { return response; }, async (error: AxiosError) => { console.error('API Error:', { url: error.config?.url, method: error.config?.method, status: error.response?.status, data: error.response?.data, message: error.message, }); // Handle specific error cases if (error.response?.status === 401) { // Unauthorized - clear storage and redirect to login await this.storage.clearAll(); // You could emit an event here for the app to handle } else if (error.response?.status === 500) { // Server error - show generic error throw new Error('Server error. Please try again later.'); } else if (!error.response) { // Network error throw new Error('Network error. Please check your internet connection.'); } // Pass through the error for component-level handling return Promise.reject(error); } ); } // ============ HTTP METHODS ============ async get<T>(url: string, config?: AxiosRequestConfig): Promise<AxiosResponse<T>> { return this.api.get<T>(url, config); } async post<T>(url: string, data?: any, config?: AxiosRequestConfig): Promise<AxiosResponse<T>> { return this.api.post<T>(url, data, config); } async put<T>(url: string, data?: any, config?: AxiosRequestConfig): Promise<AxiosResponse<T>> { return this.api.put<T>(url, data, config); } async delete<T>(url: string, config?: AxiosRequestConfig): Promise<AxiosResponse<T>> { return this.api.delete<T>(url, config); } async patch<T>(url: string, data?: any, config?: AxiosRequestConfig): Promise<AxiosResponse<T>> { return this.api.patch<T>(url, data, config); } // ============ FILE UPLOAD ============ async uploadFile<T>( url: string, formData: FormData, onUploadProgress?: (progressEvent: ProgressEvent) => void ): Promise<AxiosResponse<T>> { return this.api.post<T>(url, formData, { headers: { 'Content-Type': 'multipart/form-data', }, onUploadProgress, }); } // ============ HELPER METHODS ============ setBaseUrl(url: string): void { this.api.defaults.baseURL = url; } setDefaultHeader(key: string, value: string): void { this.api.defaults.headers.common[key] = value; } removeDefaultHeader(key: string): void { delete this.api.defaults.headers.common[key]; } }
```

API Endpoints Reference

Endpoint	Method	Purpose	Request Body	Response
`/loginDriver`	POST	Send OTP to mobile	{ mobile: string }	{ success, message, otpSent }

`/verifyOtpAndLoginDriver`	POST	Verify OTP and login	{ mobile, otp, ip_address }	{ token, user }
`/createDriverProfile`	POST	Create/update driver profile	FormData (multipart)	DriverProfile
`/getDriverProfileById/{id}`	GET	Get driver profile	-	{ DriverDetail, DriverProfile, DriverOnboardingStatuses }
`/createDriverDocument`	POST	Upload driver document	FormData (multipart)	DriverDocument
`/getDriverAllDocument/{id}`	GET	Get all driver documents	-	DriverAllDocumentsResponse
`/createDriverVehicle`	POST	Create vehicle record	{ vehicleData }	Vehicle
`/createDriverVehicleDocument`	POST	Upload vehicle document	FormData (multipart)	VehicleDocument
`/createDriverBankDetails`	POST	Add bank details	{ bankData }	DriverBankDetails
`/createDriverBankDocument`	POST	Upload bank document	FormData (multipart)	BankDocument
`/getAllUserContacts`	GET	Get emergency contacts	-	UserContact[]
`/createUserContact`	POST	Add emergency contact	{ contactData }	UserContact
`/updateUserContact/{id}`	PUT	Update contact	{ contactData }	UserContact
`/deleteUserContact/{id}`	DELETE	Delete contact	-	{ success, message }

Data Transfer Objects (DTOs)

```
// src/services/types/index.ts // Authentication Types export interface LoginDriverResponse { success: boolean; message: string; otpSent: boolean; } export interface DriverSession { token: string; user: { id: number; mobile: string; userType: 'Driver'; is_emergency_driver: number; }; } // Profile Types export interface CreateProfileRequest { driverId: string; fullName: string; city: string; pinCode: string; permanentAddress: string; dob: string; State: string; email: string; gender: 'Male' | 'Female' | 'Other'; mobile: string; } export interface DriverProfile { id: number; userId: number; fullName: string; date_of_birth: string; profileImage: string | null; city: string; State: string; pinCode: string; permanentAddress: string; gender: string; createdAt: string; updatedAt: string; } // Document Types export interface DocumentUploadRequest { driverId: number; documentType: 'pan' | 'aadhaar_front' | 'aadhaar_back' | 'driving_license'; documentNumber: string; } export interface DriverDocument { id: number; driverId: number; documentType: string; documentNumber: string; documentPath: string; status: 'approved' | 'pending' | 'rejected'; rejectionReason?: string; createdAt: string; updatedAt: string; } export interface DriverAllDocumentsResponse { pan: DriverDocument | null; aadhaar_front: DriverDocument | null; aadhaar_back: DriverDocument | null; driving_license: DriverDocument | null; rc_front: VehicleDocument | null; rc_back: VehicleDocument | null; insurance: VehicleDocument | null; vehicle_photo: VehicleDocument | null; fitness_certificate: VehicleDocument | null; bank_passbook: BankDocument | null; cancelled_cheque: BankDocument | null; } // Vehicle Types export interface CreateVehicleRequest { driverId: number; brand: string; model: string; registrationNumber: string; yearOfManufacture: string; color: string; } export interface VehicleDocumentRequest { driverId: number; vehicleId: number; documentType: 'rc_front' | 'rc_back' | 'insurance' | 'vehicle_photo' | 'fitness_certificate'; } // Bank Types export interface CreateBankDetailsRequest { driverId: number; accountHolderName: string; accountNumber: string; bankName: string; ifscCode: string; branchName: string; } // Contact Types export interface CreateUserContactRequest { user_id: number; first_name: string; last_name: string; phone_number: string; relationship: string; is_primary: boolean; } export interface UserContact { id: number; user_id: number; first_name: string; last_name: string; phone_number: string; relationship: string; is_primary: boolean; createdAt: string; updatedAt: string; }
```

Hooks & State Management

Custom Hooks Implementation

1. useDriverProfile Hook

```
// Custom hook for managing driver profile state const useDriverProfile = () => { const [profile, setProfile] = useState<DriverProfile | null>(null); const [loading, setLoading] = useState(true); const [error, setError] = useState<string | null>(null); const loadProfile = useCallback(async () => { try { setLoading(true); setError(null); const session = await driverService.getDriverSession(); if (!session) { throw new Error('No active session found'); } const profileData = await driverService.getProfile(session.user.id); setProfile(profileData); } catch
```

```
(err: any) { setError(err.message || 'Failed to load profile');
console.error('Profile load error:', err); } finally { setLoading(false); } }, []);
const updateProfile = useCallback(async (profileData: CreateProfileRequest, file?:
any) => { try { setLoading(true); const updatedProfile = await
driverService.createOrUpdateProfile(profileData, file); setProfile(updatedProfile);
return updatedProfile; } catch (err: any) { setError(err.message || 'Failed to
update profile'); throw err; } finally { setLoading(false); } }, []); const
refreshProfile = useCallback(async () => { await loadProfile(); }, [loadProfile]);
useEffect(() => { loadProfile(); }, []); return { profile, loading, error,
refreshProfile, updateProfile, }; }; // Usage in ProfileScreen: const ProfileScreen
= () => { const { profile, loading, error, refreshProfile } = useDriverProfile();
if (loading) return <LoadingSpinner />; if (error) return <ErrorView message=
{error} onRetry={refreshProfile} />; return ( <ScrollView> <ProfileRow label="Full
Name" value={profile?.fullName} /> <ProfileRow label="Email" value={profile?.email}
/> {/* ... */} </ScrollView> ); };
```

2. useDocuments Hook

```
// Custom hook for managing driver documents const useDocuments = (driverId:
number) => { const [documents, setDocuments] = useState<DriverAllDocumentsResponse
| null>(null); const [loading, setLoading] = useState(true); const [uploading,
setUploading] = useState(false); const [error, setError] = useState<string | null>
(null); const loadDocuments = useCallback(async () => { try { setLoading(true);
setError(null); const docs = await driverService.getAllDriverDocuments(driverId);
setDocuments(docs); } catch (err: any) { setError(err.message || 'Failed to load
documents'); console.error('Documents load error:', err); } finally {
setLoading(false); } }, [driverId]); const uploadDocument = useCallback(async (
documentType: DocumentType, documentNumber: string, file: any ) => { try {
setUploading(true); setError(null); const requestData: DocumentUploadRequest = {
driverId, documentType, documentNumber, }; const uploadedDoc = await
driverService.uploadDocument(requestData, file); // Update local state
setDocuments(prev => ({ ...prev!, [documentType]: uploadedDoc, })); return
uploadedDoc; } catch (err: any) { setError(err.message || 'Failed to upload
document'); throw err; } finally { setUploading(false); } }, [driverId]); const
refreshDocuments = useCallback(async () => { await loadDocuments(); },
[loadDocuments]); useEffect(() => { if (driverId) { loadDocuments(); } },
[driverId, loadDocuments]); return { documents, loading, uploading, error,
uploadDocument, refreshDocuments, }; }; // Usage in MyDocument screen: const
MyDocumentScreen = () => { const { user } = useSession(); const { documents,
loading, uploadDocument, refreshDocuments } = useDocuments(user?.id); const
handleDocumentUpload = async (type: DocumentType) => { const result = await
ImagePicker.launchImageLibrary({ mediaType: 'photo', quality: 0.8, }); if
(!result.didCancel && result.assets?.[0]) { await uploadDocument(type, 'DOC123',
result.assets[0]); refreshDocuments(); } }; return ( <View> {loading ? (
<LoadingSpinner /> ) : ( <DocumentList documents={documents} onUpload=
{handleDocumentUpload} /> )} </View> ); };
```

3. useEmergencyContacts Hook

```
// Custom hook for managing emergency contacts const useEmergencyContacts = () => {
const [contacts, setContacts] = useState<UserContact[]>([]); const [loading,
setLoading] = useState(true); const [error, setError] = useState<string | null>
```

```
(null); const [primaryContact, setPrimaryContact] = useState<UserContact | null>
(null); const loadContacts = useCallback(async () => { try { setLoading(true);
setError(null); const fetchedContacts = await driverService.getAllUserContacts();
setContacts(fetchedContacts); const primary = fetchedContacts.find(contact =>
contact.is_primary); setPrimaryContact(primary || null); } catch (err: any) {
setError(err.message || 'Failed to load contacts'); console.error('Contacts load
error:', err); } finally { setLoading(false); } }, []); const addContact =
useCallback(async (contactData: CreateUserContactRequest) => { try { const
newContact = await driverService.createUserContact(contactData); // If this is
primary, update other contacts if (contactData.is_primary) { const updatedContacts
= contacts.map(contact => ({ ...contact, is_primary: false, })));
setContacts([...updatedContacts, newContact]); } else { setContacts([...contacts,
newContact]); } if (newContact.is_primary) { setPrimaryContact(newContact); }
return newContact; } catch (err: any) { throw new Error(err.message || 'Failed to
add contact'); } }, [contacts]); const updateContact = useCallback(async (id:
number, updates: Partial<UserContact>) => { try { const updatedContact = await
driverService.updateUserContact(id, updates); // Handle primary contact change if
(updates.is_primary) { const updatedContacts = contacts.map(contact => ({
...contact, is_primary: contact.id === id, })); setContacts(updatedContacts);
setPrimaryContact(updatedContact); } else { setContacts(contacts.map(contact =>
contact.id === id ? updatedContact : contact )); } return updatedContact; } catch
(err: any) { throw new Error(err.message || 'Failed to update contact'); } },
[contacts]); const deleteContact = useCallback(async (id: number) => { try { await
driverService.deleteUserContact(id); const deletedContact = contacts.find(contact
=> contact.id === id); const remainingContacts = contacts.filter(contact =>
contact.id !== id); setContacts(remainingContacts); // If deleted contact was
primary, set a new primary if (deletedContact?.is_primary &&
remainingContacts.length > 0) { const newPrimary = { ...remainingContacts[0],
is_primary: true }; await updateContact(newPrimary.id, { is_primary: true }); } }
catch (err: any) { throw new Error(err.message || 'Failed to delete contact'); } },
[contacts, updateContact]); const setAsPrimary = useCallback(async (id: number) =>
{ await updateContact(id, { is_primary: true }); }, [updateContact]); const
refreshContacts = useCallback(async () => { await loadContacts(); },
[loadContacts]); useEffect(() => { loadContacts(); }, []); return { contacts,
primaryContact, loading, error, addContact, updateContact, deleteContact,
setAsPrimary, refreshContacts, }; }; // Usage in EmergencyContacts screen: const
EmergencyContactsScreen = () => { const { contacts, loading, error, deleteContact,
setAsPrimary, refreshContacts, } = useEmergencyContacts(); const handleDelete =
async (id: number) => { Alert.alert( 'Delete Contact', 'Are you sure you want to
delete this contact?', [ { text: 'Cancel', style: 'cancel' }, { text: 'Delete',
style: 'destructive', onPress: async () => { try { await deleteContact(id); } catch
(err) { Alert.alert('Error', 'Failed to delete contact'); } } }, ] ); }; return (
<View> <FlatList data={contacts} renderItem={({ item }) => ( <ContactItem contact=
{item} isPrimary={item.is_primary} onPress={() =>
navigation.navigate('ContactDetails', { contactId: item.id })} onCall={() =>
Linking.openURL(`tel:${item.phone_number}`)} onMessage={() =>
Linking.openURL(`sms:${item.phone_number}`)} /> )} refreshing={loading} onRefresh=
{refreshContacts} /> </View> ); };
```

Advanced Hook Patterns

4. useForm Hook with Validation

```
// Generic form hook with validation const useForm = <T extends Record<string, any>>( initialValues: T, validate?: (values: T) => Record<string, string> ) => { const [values, setValues] = useState<T>(initialValues); const [errors, setErrors] = useState<Record<string, string>>({}); const [touched, setTouched] = useState<Record<string, boolean>>({}); const [isSubmitting, setIsSubmitting] = useState(false); const handleChange = useCallback((name: keyof T, value: any) => { setValues(prev => ({ ...prev, [name]: value, })); // Clear error when field changes if (errors[name as string]) { setErrors(prev => { const newErrors = { ...prev }; delete newErrors[name as string]; return newErrors; }); } }, [errors]); const handleBlur = useCallback((name: keyof T) => { setTouched(prev => ({ ...prev, [name]: true, })); // Validate field on blur if validation function provided if (validate) { const fieldErrors = validate(values); if (fieldErrors[name as string]) { setErrors(prev => ({ ...prev, [name]: fieldErrors[name as string], })); } } }, [values, validate]); const setFieldValue = useCallback((name: keyof T, value: any) => { handleChange(name, value); }, [handleChange]); const setFieldError = useCallback((name: string, error: string) => { setErrors(prev => ({ ...prev, [name]: error, })); }, []); const validateForm = useCallback(() => { if (!validate) return true; const newErrors = validate(values); setErrors(newErrors); setTouched( Object.keys(values).reduce((acc, key) => { acc[key] = true; return acc; }, {} as Record<string, boolean>) ); return Object.keys(newErrors).length === 0; }, [values, validate]); const resetForm = useCallback(() => { setValues(initialValues); setErrors({}); setTouched({}); setIsSubmitting(false); }, [initialValues]); const getFieldProps = useCallback((name: keyof T) => { return { value: values[name], onChangeText: (text: string) => handleChange(name, text), onBlur: () => handleBlur(name), error: errors[name as string], touched: touched[name as string], }; }, [values, errors, touched, handleChange, handleBlur]); return { values, errors, touched, isSubmitting, handleChange, handleBlur, setFieldValue, setFieldError, validateForm, resetForm, getFieldProps, setValues, setErrors, setIsSubmitting, }; }; // Usage in EditProfilePage: const EditProfilePage = () => { const { profile } = useDriverProfile(); const validateProfile = (values: ProfileFormValues) => { const errors: Record<string, string> = {}; if (!values.fullName?.trim()) { errors.fullName = 'Full name is required'; } if (!values.email?.trim()) { errors.email = 'Email is required'; } else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(values.email)) { errors.email = 'Invalid email format'; } if (!values.mobile?.trim()) { errors.mobile = 'Mobile number is required'; } else if (!/^\d{10}$/.test(values.mobile)) { errors.mobile = 'Invalid mobile number'; } return errors; }; const form = useForm<ProfileFormValues>( { fullName: profile?.fullName || '', email: profile?.email || '', mobile: profile?.mobile || '', city: profile?.city || '', pinCode: profile?.pinCode || '', // ... other fields }, validateProfile ); const handleSubmit = async () => { if (!form.validateForm()) return; try { form.setIsSubmitting(true); await driverService.createOrUpdateProfile(form.values); Alert.alert('Success', 'Profile updated successfully'); navigation.goBack(); } catch (error: any) { Alert.alert('Error', error.message || 'Failed to update profile'); } finally { form.setIsSubmitting(false); } }; return ( <ScrollView> <TextInput placeholder="Full Name" {...form.getFieldProps('fullName')} /> {form.errors.fullName && form.touched.fullName && ( <Text style={styles.error}> {form.errors.fullName}</Text> )} <TextInput placeholder="Email" keyboardType="email-address" {...form.getFieldProps('email')} /> {form.errors.email && form.touched.email && ( <Text style={styles.error}>{form.errors.email}</Text> )} <Button title="Save Changes" onPress={handleSubmit} disabled={form.isSubmitting} /> </ScrollView> ); };
```

5. useImagePicker Hook

```
// Custom hook for image picking functionality const useImagePicker = (options?: ImagePickerOptions) => { const [image, setImage] = useState<ImagePickerResponse | null>(null); const [loading, setLoading] = useState(false); const [error, setError] = useState<string | null>(null); const defaultOptions: ImagePickerOptions = { mediaType: 'photo', quality: 0.8, maxWidth: 1024, maxHeight: 1024, ...options, }; const pickImage = useCallback(async () => { try { setLoading(true); setError(null); const result = await ImagePicker.launchImageLibrary(defaultOptions); if (result.didCancel) { return null; } if (result.errorCode) { throw new Error(getErrorMessage(result.errorCode)); } if (result.assets?.[0]) { const selectedImage = result.assets[0]; setImage(selectedImage); return selectedImage; } return null; } catch (err: any) { setError(err.message || 'Failed to pick image'); console.error('Image pick error:', err); return null; } finally { setLoading(false); } }, [defaultOptions]); const takePhoto = useCallback(async () => { try { setLoading(true); setError(null); const result = await ImagePicker.launchCamera(defaultOptions); if (result.didCancel) { return null; } if (result.errorCode) { throw new Error(getErrorMessage(result.errorCode)); } if (result.assets?.[0]) { const photo = result.assets[0]; setImage(photo); return photo; } return null; } catch (err: any) { setError(err.message || 'Failed to take photo'); console.error('Camera error:', err); return null; } finally { setLoading(false); } }, [defaultOptions]); const clearImage = useCallback(() => { setImage(null); setError(null); }, []); const getErrorMessage = (errorCode: string): string => { switch (errorCode) { case 'camera_unavailable': return 'Camera is not available on this device'; case 'permission': return 'Permission to access camera was denied'; case 'others': return 'An unexpected error occurred'; default: return 'Failed to access media'; } }; return { image, loading, error, pickImage, takePhoto, clearImage, }; }; // Usage in VerifyIdentityScreen: const VerifyIdentityScreen = () => { const panPicker = useImagePicker(); const aadhaarFrontPicker = useImagePicker(); const aadhaarBackPicker = useImagePicker(); const handlePanUpload = async () => { const image = await panPicker.pickImage(); if (image) { // Upload to server await driverService.uploadDocument({ driverId: 123, documentType: 'pan', documentNumber: 'ABCDE1234F', }, image); } }; return ( <View> <TouchableOpacity onPress={handlePanUpload}> {panPicker.image ? ( <Image source={{ uri: panPicker.image.uri }} style={styles.image} /> ) : ( <View style={styles.uploadBox}> <Text>Upload PAN Card</Text> </View> )} </TouchableOpacity> {panPicker.loading && <ActivityIndicator />} {panPicker.error && <Text style={styles.error}>{panPicker.error}</Text>} {/* Similar for other documents */} </View> ); };
```

Performance Optimization Hooks

6. useDebounce Hook

```
// Debounce hook for search and filtering const useDebounce = <T>(value: T, delay: number): T => { const [debouncedValue, setDebouncedValue] = useState<T>(value); useEffect(() => { const handler = setTimeout(() => { setDebouncedValue(value); }, delay); return () => { clearTimeout(handler); }; }, [value, delay]); return debouncedValue; }; // Usage in search functionality: const SearchScreen = () => { const [searchTerm, setSearchTerm] = useState(''); const [results, setResults] = useState([]); const debouncedSearchTerm = useDebounce(searchTerm, 500); useEffect(() => { if (debouncedSearchTerm) { searchApi(debouncedSearchTerm).then(setResults); } else { setResults([]); } }, [debouncedSearchTerm]); return ( <View> <TextInput placeholder="Search..." value=
```

```
{searchTerm} onChangeText={setSearchTerm} /> <SearchResults results={results} />
</View> ); };
```

7. useInterval Hook

```
// Interval hook for timers and polling const useInterval = (callback: () => void,
delay: number | null) => { const savedCallback = useRef<() => void>(); useEffect(()
=> { savedCallback.current = callback; }, [callback]); useEffect(() => { if (delay
!== null) { const id = setInterval(() => { savedCallback.current?.(); }, delay);
return () => clearInterval(id); } }, [delay]); }; // Usage in OTP resend timer:
const LoginScreen = () => { const [resendTimer, setResendTimer] = useState(0);
useInterval(() => { if (resendTimer > 0) { setResendTimer(prev => prev - 1); } },
resendTimer > 0 ? 1000 : null); const handleResendOtp = () => { setResendTimer(30);
// 30 seconds // Resend OTP logic }; return ( <View> <Button title={`Resend OTP
${resendTimer > 0 ? `(${resendTimer}s)` : ''}`} onPress={handleResendOtp} disabled=
{resendTimer > 0} /> </View> ); };
```

Hook Best Practices

Hook Implementation Guidelines:

- **Single Responsibility:** Each hook should have one clear purpose
- **Dependency Arrays:** Properly specify dependencies in useEffect and useCallback
- **Error Handling:** All hooks should handle errors gracefully
- **Loading States:** Always include loading states for async operations
- **Type Safety:** Use TypeScript generics for reusable hooks
- **Memoization:** Use useMemo and useCallback to prevent unnecessary re-renders
- **Cleanup:** Properly clean up subscriptions and timers
- **Testing:** Hooks should be easily testable in isolation

Application Flows

Primary User Flows

1. New Driver Onboarding Flow

1. **Splash Screen**
   Auto-checks existing session, shows app branding
   Duration: 2 seconds → Auto-navigation

2 **Login Screen**
Mobile number input, OTP verification with resend timer
API: POST /loginDriver → POST /verifyOtpAndLoginDriver

3 **Onboarding Screen**
Personal information collection with validation
11 form fields → Profile image upload → API: POST /createDriverProfile

4 **Verify Identity Screen**
Document upload (PAN, Aadhaar, Driving License)
3 document types → Image picker → API: POST /createDriverDocument

5 **Add Vehicle Details**
Vehicle registration with images and documents
5+ vehicle details → 4+ documents → API: POST /createDriverVehicle

6 **Add Bank Details**
Bank account information with document upload
Bank details → Passbook/Cheque → API: POST /createDriverBankDetails

7 **Status Screen**
Application status display or document rejection
Status check → Rejection handling → Thank you message

8 **Home Screen**
Main dashboard with drawer navigation
Drawer menu → Bottom tabs → Profile completion

2. Document Rejection Flow

1 **Status Screen**
Shows rejection message with "Resubmit" button
UI: Red status banner → Resubmit CTA

2 **Resubmit Screen**
Lists all rejected documents by category
Grouped by: Personal → Vehicle → Bank

3 **Rejected Documents Screen**
Document-specific resubmission form
Shows: Rejection reason → Corrected fields → Upload

4 **Document Upload**
New document upload with corrected information
Image picker → Validation → API call

5 **API Update**
Document status updated to "pending"
Status: rejected → pending → Back to status screen

3. Emergency Contacts Flow

1 **Contacts List**
View all emergency contacts with primary indicator
API: GET /getAllUserContacts → Local cache

2 **Add Contact**
Form with name, mobile, relationship, primary toggle
Validation → API: POST /createUserContact

3 **Contact Details**
View contact details, set as primary, or delete
Actions: Call → Message → Edit → Set Primary

4 **Delete Confirmation**
Modal confirmation before deletion
UI: Alert dialog → Confirm/Cancel → API: DELETE

5 **API Synchronization**
All changes synced with backend API
Real-time sync → Local cache update → UI refresh

Error Handling Flows

Common Error Scenarios:

- **Network Errors:** Display user-friendly messages, retry options
- **Validation Errors:** Inline form validation with specific messages
- **API Errors:** HTTP status code handling (401, 400, 500)
- **Session Expiry:** Auto-logout and redirect to login
- **Image Upload Errors:** File size, format validation

Offline Support Flow

Offline Capabilities:

- **Data Caching:** All API responses cached in AsyncStorage
- **Queue System:** Form submissions queued when offline
- **Sync on Reconnect:** Automatic sync when network returns
- **Offline Indicators:** Clear UI indicators for offline state
- **Local Validation:** Form validation works offline

Utilities & Helpers

Scale Functions (utils/scale.ts)

```
// Responsive scaling utilities based on screen dimensions import { Dimensions, Platform } from 'react-native'; const { width, height } = Dimensions.get('window'); // Base dimensions for iPhone X (375x812) - design reference const BASE_WIDTH = 375; const BASE_HEIGHT = 812; // Scaling functions export const sw = (size: number): number => { // Scale by width const scale = width / BASE_WIDTH; return Math.round(size * scale); }; export const sh = (size: number): number => { // Scale by height (less aggressive) const scale = height / BASE_HEIGHT; return Math.round(size * scale); }; export const sf = (size: number): number => { // Scale font size (with minimum size) const scale = Math.min(width / BASE_WIDTH, 1.2); // Cap scaling at 1.2x const scaledSize = Math.round(size * scale); return Platform.OS === 'ios' ? scaledSize : scaledSize - 1; // Adjust for Android }; export const s = (size: number): number => { // General scaling (average of width and height scaling) const widthScale = width / BASE_WIDTH; const heightScale = height / BASE_HEIGHT; const scale = (widthScale + heightScale) / 2; return Math.round(size * scale); }; // Platform-specific adjustments export const platformScale = (iosSize: number, androidSize: number): number => { return Platform.OS === 'ios' ? iosSize : androidSize; }; // Responsive padding/margin helpers export const responsive = { small: sw(8), medium: sw(16), large: sw(24), xlarge: sw(32), // Screen padding screenPadding: sw(20), // Component spacing componentSpacing: sw(12), // Border radius borderRadius: { small: s(4), medium: s(8), large: s(12), xlarge: s(16), round: s(999), }, // Icon sizes icon: { small: s(16), medium: s(24), large: s(32), xlarge: s(48), }, }; // Usage in styles: const styles = StyleSheet.create({ container: { paddingHorizontal: responsive.screenPadding, paddingVertical: sh(16),
```

```
marginBottom: responsive.medium, }, button: { height: sh(48), borderRadius:
responsive.borderRadius.medium, paddingHorizontal: sw(20), }, title: { fontSize:
sf(24), marginBottom: sh(16), }, text: { fontSize: sf(16), lineHeight: sf(24), },
icon: { width: responsive.icon.medium, height: responsive.icon.medium, }, });
```

Font Family Helper (utils/fontFamily.ts)

```
// Font family management with TypeScript support type FontWeight = | 'thin' |
'extraLight' | 'light' | 'regular' | 'medium' | 'semiBold' | 'bold' | 'extraBold' |
'black'; interface FontFamily { thin: string; extraLight: string; light: string;
regular: string; medium: string; semiBold: string; bold: string; extraBold: string;
black: string; } const POPPINS: FontFamily = { thin: 'Poppins-Thin', extraLight:
'Poppins-ExtraLight', light: 'Poppins-Light', regular: 'Poppins-Regular', medium:
'Poppins-Medium', semiBold: 'Poppins-SemiBold', bold: 'Poppins-Bold', extraBold:
'Poppins-ExtraBold', black: 'Poppins-Black', }; const ROBOTO: FontFamily = { thin:
'Roboto-Thin', extraLight: 'Roboto-Light', light: 'Roboto-Light', regular: 'Roboto-
Regular', medium: 'Roboto-Medium', semiBold: 'Roboto-Medium', // Roboto doesn't
have semiBold bold: 'Roboto-Bold', extraBold: 'Roboto-Black', black: 'Roboto-
Black', }; export const fontFamilies = { POPPINS, ROBOTO, } as const; type
FontFamilyKey = keyof typeof fontFamilies; // Configuration const
DEFAULT_FONT_FAMILY: FontFamilyKey = 'POPPINS'; export const getFontFamily = (
weight: FontWeight = 'regular', family: FontFamilyKey = DEFAULT_FONT_FAMILY ):
string => { const selectedFontFamily = fontFamilies[family]; return
selectedFontFamily[weight]; }; // Font weight mapping for dynamic styling export
const fontWeightToFamily = { '100': getFontFamily('thin'), '200':
getFontFamily('extraLight'), '300': getFontFamily('light'), '400':
getFontFamily('regular'), '500': getFontFamily('medium'), '600':
getFontFamily('semiBold'), '700': getFontFamily('bold'), '800':
getFontFamily('extraBold'), '900': getFontFamily('black'), }; // Usage examples:
const styles = StyleSheet.create({ // Direct usage heading: { fontFamily:
getFontFamily('bold'), fontSize: sf(28), color: '#1E293B', }, subtitle: {
fontFamily: getFontFamily('semiBold'), fontSize: sf(18), color: '#475569', }, body:
{ fontFamily: getFontFamily('regular'), fontSize: sf(16), color: '#64748B',
lineHeight: sf(24), }, caption: { fontFamily: getFontFamily('light'), fontSize:
sf(14), color: '#94A3B8', }, // Dynamic styling helper dynamicText: (weight:
FontWeight = 'regular') => ({ fontFamily: getFontFamily(weight), fontSize: sf(16),
}), }); // Component usage: const TextComponent = ({ children, variant = 'body',
weight = 'regular', style }: TextProps) => { const getVariantStyle = () => { switch
(variant) { case 'heading': return styles.heading; case 'subtitle': return
styles.subtitle; case 'caption': return styles.caption; default: return
styles.body; } }; return ( <Text style={[ getVariantStyle(), { fontFamily:
getFontFamily(weight) }, style ]} > {children} </Text> ); };
```

Date Formatting Utilities

```
// Date formatting and manipulation utilities import { format, parseISO,
differenceInYears, isValid } from 'date-fns'; export class DateUtils { // Format
date for display static formatDate( dateString: string | Date | null | undefined,
formatString: string = 'dd/MM/yyyy' ): string { if (!dateString) return 'N/A'; try
{ const date = typeof dateString === 'string' ? parseISO(dateString) : dateString;
if (!isValid(date)) return 'Invalid Date'; return format(date, formatString); }
```

```
catch { return 'N/A'; } } // Format date for API (YYYY-MM-DD) static formatDateForAPI(date: Date | null): string | null { if (!date || !isValid(date)) return null; return format(date, 'yyyy-MM-dd'); } // Calculate age from date of birth static calculateAge(dob: string | Date | null | undefined): string { if (!dob) return 'N/A'; try { const birthDate = typeof dob === 'string' ? parseISO(dob) : dob; if (!isValid(birthDate)) return 'N/A'; const today = new Date(); let age = differenceInYears(today, birthDate); // Adjust if birthday hasn't occurred this year const monthDiff = today.getMonth() - birthDate.getMonth(); if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) { age--; } return `${age} years`; } catch { return 'N/A'; } } // Format date with time static formatDateTime( dateString: string | Date, includeTime: boolean = true ): string { if (!dateString) return 'N/A'; try { const date = typeof dateString === 'string' ? parseISO(dateString) : dateString; if (!isValid(date)) return 'Invalid Date'; const dateFormat = 'dd/MM/yyyy'; const timeFormat = 'hh:mm a'; if (includeTime) { return `${format(date, dateFormat)} at ${format(date, timeFormat)}`; } return format(date, dateFormat); } catch { return 'N/A'; } } // Get relative time (e.g., "2 days ago") static getRelativeTime(dateString: string | Date): string { if (!dateString) return 'N/A'; try { const date = typeof dateString === 'string' ? parseISO(dateString) : dateString; if (!isValid(date)) return 'Invalid Date'; const now = new Date(); const diffInSeconds = Math.floor((now.getTime() - date.getTime()) / 1000); if (diffInSeconds < 60) return 'Just now'; if (diffInSeconds < 3600) return `${Math.floor(diffInSeconds / 60)} minutes ago`; if (diffInSeconds < 86400) return `${Math.floor(diffInSeconds / 3600)} hours ago`; if (diffInSeconds < 604800) return `${Math.floor(diffInSeconds / 86400)} days ago`; return this.formatDate(date); } catch { return 'N/A'; } } // Check if date is in the future static isFutureDate(dateString: string | Date): boolean { if (!dateString) return false; try { const date = typeof dateString === 'string' ? parseISO(dateString) : dateString; if (!isValid(date)) return false; return date > new Date(); } catch { return false; } } // Check if date is expired (for documents) static isExpired(dateString: string | Date | null): boolean { if (!dateString) return false; try { const date = typeof dateString === 'string' ? parseISO(dateString) : dateString; if (!isValid(date)) return false; const today = new Date(); today.setHours(0, 0, 0, 0); return date < today; } catch { return false; } } // Get days until expiration static getDaysUntilExpiry(dateString: string | Date | null): number | null { if (!dateString) return null; try { const date = typeof dateString === 'string' ? parseISO(dateString) : dateString; if (!isValid(date)) return null; const today = new Date(); today.setHours(0, 0, 0, 0); const diffTime = date.getTime() - today.getTime(); return Math.ceil(diffTime / (1000 * 60 * 60 * 24)); } catch { return null; } } // Parse date from various formats static parseDate(dateString: string, format?: string): Date | null { if (!dateString) return null; try { // Try ISO format first let date = parseISO(dateString); if (isValid(date)) return date; // Try common formats const formats = [ 'yyyy-MM-dd', 'dd/MM/yyyy', 'MM/dd/yyyy', 'yyyy/MM/dd', ]; for (const fmt of formats) { date = parse(dateString, fmt, new Date()); if (isValid(date)) return date; } return null; } catch { return null; } } }
```

String Utilities

```
// String manipulation and formatting utilities export class StringUtils { // Capitalize first letter of each word static capitalize(str: string): string { if (!str) return ''; return str .toLowerCase() .split(' ') .map(word => word.charAt(0).toUpperCase() + word.slice(1)) .join(' '); } // Format mobile number with country code static formatMobileNumber(mobile: string, countryCode: string =
```

```
'+91'): string { if (!mobile) return ''; const cleaned = mobile.replace(/\D/g, ''); if (cleaned.length === 10) { return `${countryCode} ${cleaned.substring(0, 5)} ${cleaned.substring(5)}`; } return cleaned; } // Mask sensitive information static maskString(str: string, visibleChars: number = 4): string { if (!str || str.length <= visibleChars * 2) return str; const firstVisible = str.substring(0, visibleChars); const lastVisible = str.substring(str.length - visibleChars); const middle = '*'.repeat(str.length - visibleChars * 2); return `${firstVisible}${middle}${lastVisible}`; } // Truncate text with ellipsis static truncate(text: string, maxLength: number = 50): string { if (!text || text.length <= maxLength) return text; return `${text.substring(0, maxLength)}...`; } // Extract initials from name static getInitials(name: string): string { if (!name) return ''; const names = name.trim().split(' '); if (names.length === 1) return names[0].charAt(0).toUpperCase(); const firstInitial = names[0].charAt(0); const lastInitial = names[names.length - 1].charAt(0); return `${firstInitial}${lastInitial}`.toUpperCase(); } // Format currency static formatCurrency(amount: number, currency: string = '₹'): string { if (amount === null || amount === undefined) return `${currency}0`; return `${currency}${amount.toLocaleString('en-IN', { minimumFractionDigits: 0, maximumFractionDigits: 2, })}`; } // Validate email static isValidEmail(email: string): boolean { const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; return emailRegex.test(email); } // Validate Indian mobile number static isValidIndianMobile(mobile: string): boolean { const mobileRegex = /^[6-9]\d{9}$/; return mobileRegex.test(mobile.replace(/\D/g, '')); } // Validate PAN number static isValidPAN(pan: string): boolean { const panRegex = /^[A-Z]{5}[0-9]{4}[A-Z]{1}$/; return panRegex.test(pan.toUpperCase()); } // Validate IFSC code static isValidIFSC(ifsc: string): boolean { const ifscRegex = /^[A-Z]{4}0[A-Z0-9]{6}$/; return ifscRegex.test(ifsc.toUpperCase()); } // Remove extra whitespace static normalizeWhitespace(str: string): string { if (!str) return ''; return str.replace(/\s+/g, ' ').trim(); } // Convert snake_case to Title Case static snakeToTitle(str: string): string { if (!str) return ''; return str .split('_') .map(word => word.charAt(0).toUpperCase() + word.slice(1)) .join(' '); } // Generate random string static generateRandomString(length: number = 8): string { const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'; let result = ''; for (let i = 0; i < length; i++) { result += chars.charAt(Math.floor(Math.random() * chars.length)); } return result; } }
```

Validation Utilities

```
// Form validation utilities export class ValidationUtils { // Required field validation static required(value: any, fieldName: string = 'This field'): string | null { if (value === null || value === undefined || value === '') { return `${fieldName} is required`; } if (Array.isArray(value) && value.length === 0) { return `${fieldName} is required`; } return null; } // Minimum length validation static minLength(value: string, min: number, fieldName: string = 'Field'): string | null { if (value && value.length < min) { return `${fieldName} must be at least ${min} characters`; } return null; } // Maximum length validation static maxLength(value: string, max: number, fieldName: string = 'Field'): string | null { if (value && value.length > max) { return `${fieldName} must be at most ${max} characters`; } return null; } // Email validation static email(value: string): string | null { if (value && !StringUtils.isValidEmail(value)) { return 'Please enter a valid email address'; } return null; } // Mobile number validation static mobile(value: string): string | null { if (value && !StringUtils.isValidIndianMobile(value)) { return 'Please enter a valid 10-digit
```

```
mobile number'; } return null; } // PAN validation static pan(value: string): string | null { if (value && !StringUtils.isValidPAN(value)) { return 'Please enter a valid PAN number'; } return null; } // IFSC validation static ifsc(value: string): string | null { if (value && !StringUtils.isValidIFSC(value)) { return 'Please enter a valid IFSC code'; } return null; } // Numeric validation static numeric(value: string, fieldName: string = 'Field'): string | null { if (value && !/^\d+$/.test(value)) { return `${fieldName} must contain only numbers`; } return null; } // Alpha validation static alpha(value: string, fieldName: string = 'Field'): string | null { if (value && !/^[A-Za-z\s]+$/.test(value)) { return `${fieldName} must contain only letters`; } return null; } // Alpha-numeric validation static alphaNumeric(value: string, fieldName: string = 'Field'): string | null { if (value && !/^[A-Za-z0-9\s]+$/.test(value)) { return `${fieldName} must contain only letters and numbers`; } return null; } // Password strength validation static password(value: string): string | null { if (!value) return null; const errors: string[] = []; if (value.length < 8) { errors.push('at least 8 characters'); } if (!/[A-Z]/.test(value)) { errors.push('one uppercase letter'); } if (!/[a-z]/.test(value)) { errors.push('one lowercase letter'); } if (!/\d/.test(value)) { errors.push('one number'); } if (!/[!@#$%^&*(),.?":{}|<>]/.test(value)) { errors.push('one special character'); } if (errors.length > 0) { return `Password must contain ${errors.join(', ')}`; } return null; } // Match two fields (e.g., password confirmation) static match(value1: any, value2: any, fieldName: string = 'Fields'): string | null { if (value1 !== value2) { return `${fieldName} do not match`; } return null; } // Custom regex validation static regex(value: string, pattern: RegExp, message: string): string | null { if (value && !pattern.test(value)) { return message; } return null; } // Date validation static date(value: string, fieldName: string = 'Date'): string | null { if (!value) return null; const date = DateUtils.parseDate(value); if (!date || !isValid(date)) { return `Please enter a valid ${fieldName.toLowerCase()}`; } return null; } // Future date validation static futureDate(value: string, fieldName: string = 'Date'): string | null { const dateError = this.date(value, fieldName); if (dateError) return dateError; const date = DateUtils.parseDate(value); if (date && !DateUtils.isFutureDate(date)) { return `${fieldName} must be in the future`; } return null; } // Past date validation static pastDate(value: string, fieldName: string = 'Date'): string | null { const dateError = this.date(value, fieldName); if (dateError) return dateError; const date = DateUtils.parseDate(value); if (date && DateUtils.isFutureDate(date)) { return `${fieldName} must be in the past`; } return null; } // Age validation (minimum age) static minimumAge(value: string, minAge: number, fieldName: string = 'Date of birth'): string | null { const dateError = this.date(value, fieldName); if (dateError) return dateError; const age = DateUtils.calculateAge(value); const ageNumber = parseInt(age); if (isNaN(ageNumber) || ageNumber < minAge) { return `You must be at least ${minAge} years old`; } return null; } // File size validation static fileSize(fileSize: number, maxSizeMB: number): string | null { const maxSizeBytes = maxSizeMB * 1024 * 1024; if (fileSize > maxSizeBytes) { return `File size must be less than ${maxSizeMB}MB`; } return null; } // File type validation static fileType(fileName: string, allowedTypes: string[]): string | null { if (!fileName) return null; const extension = fileName.split('.').pop()?.toLowerCase(); if (!extension || !allowedTypes.includes(extension)) { const allowedFormats = allowedTypes.join(', '); return `File must be one of: ${allowedFormats}`; } return null; } // Validate all fields in an object static validateAll( values: Record, rules: Record string | null> ): Record { const errors: Record = {}; Object.keys(rules).forEach(field => { const validator = rules[field]; const error = validator(values[field]); if (error) { errors[field] = error; } }); return errors; } }
```

Network Utilities

```
// Network and connectivity utilities import NetInfo from '@react-native-community/netinfo'; export class NetworkUtils { // Check network connectivity static async isConnected(): Promise { try { const state = await NetInfo.fetch(); return state.isConnected ?? false; } catch { return false; } } // Get connection type static async getConnectionType(): Promise { try { const state = await NetInfo.fetch(); return state.type || 'unknown'; } catch { return 'unknown'; } } // Check if connected to WiFi static async isWifiConnected(): Promise { try { const state = await NetInfo.fetch(); return state.type === 'wifi'; } catch { return false; } } // Check if connected to cellular static async isCellularConnected(): Promise { try { const state = await NetInfo.fetch(); return state.type === 'cellular'; } catch { return false; } } // Get network strength (for cellular) static async getCellularGeneration(): Promise { try { const state = await NetInfo.fetch(); return state.details?.cellularGeneration || null; } catch { return null; } } // Subscribe to network changes static subscribeToNetworkChanges( callback: (isConnected: boolean, connectionType: string) => void ): () => void { return NetInfo.addEventListener(state => { callback(state.isConnected ?? false, state.type); }); } // Test network speed (simple ping) static async testNetworkSpeed(): Promise { try { const startTime = Date.now(); await fetch('https://www.google.com', { method: 'HEAD' }); const endTime = Date.now(); return endTime - startTime; } catch { return null; } } // Retry with exponential backoff static async retryWithBackoff( operation: () => Promise, maxRetries: number = 3, baseDelay: number = 1000 ): Promise { for (let attempt = 1; attempt <= maxRetries; attempt++) { try { return await operation(); } catch (error) { if (attempt === maxRetries) { throw error; } const delay = baseDelay * Math.pow(2, attempt - 1); await new Promise(resolve => setTimeout(resolve, delay)); } } throw new Error('Max retries exceeded'); } // Queue for offline operations static createOfflineQueue() { const queue: Array<() => Promise> = []; let isProcessing = false; const processQueue = async () => { if (isProcessing || queue.length === 0) return; isProcessing = true; while (queue.length > 0) { const operation = queue.shift(); if (operation) { try { await operation(); } catch (error) { console.error('Queue operation failed:', error); // Optionally retry or move to failed queue } } } isProcessing = false; }; return { add: (operation: () => Promise) => { queue.push(operation); processQueue(); }, clear: () => { queue.length = 0; }, get length() { return queue.length; }, }; } }
```

Storage Utilities

```
// Enhanced storage utilities import AsyncStorage from '@react-native-async-storage/async-storage'; export class StorageUtils { // Secure storage keys (encrypted in production) private static encryptKey(key: string): string { // In production, use proper encryption return `secure_${key}`; } // Save with expiration static async setWithExpiry( key: string, value: any, ttl: number // Time to live in milliseconds ): Promise { try { const item = { value, expiry: Date.now() + ttl, }; await AsyncStorage.setItem(key, JSON.stringify(item)); } catch (error) { console.error('Error saving with expiry:', error); } } // Get with expiration check static async getWithExpiry(key: string): Promise { try { const itemStr = await AsyncStorage.getItem(key); if (!itemStr) return null; const item = JSON.parse(itemStr); if (Date.now() > item.expiry) { await AsyncStorage.removeItem(key); return null; } return item.value; } catch (error) { console.error('Error getting with expiry:', error); return null; } } // Bulk
```

```
operations static async saveMultiple(items: Record): Promise { try { const entries = Object.entries(items).map(([key, value]) => [ key, JSON.stringify(value), ]); await AsyncStorage.multiSet(entries as [string, string][]); } catch (error) { console.error('Error saving multiple items:', error); } } static async getMultiple(keys: string[]): Promise> { try { const values = await AsyncStorage.multiGet(keys); const result: Record = {}; values.forEach(([key, value]) => { if (value !== null) { result[key] = JSON.parse(value); } }); return result; } catch (error) { console.error('Error getting multiple items:', error); return {}; } } // Clear by pattern static async clearByPattern(pattern: RegExp): Promise { try { const allKeys = await AsyncStorage.getAllKeys(); const keysToRemove = allKeys.filter(key => pattern.test(key)); if (keysToRemove.length > 0) { await AsyncStorage.multiRemove(keysToRemove); } } catch (error) { console.error('Error clearing by pattern:', error); } } // Get storage info static async getStorageInfo(): Promise<{ totalKeys: number; sampleKeys: string[]; estimatedSize: number; }> { try { const allKeys = await AsyncStorage.getAllKeys(); const sampleKeys = allKeys.slice(0, 10); // Estimate size (rough calculation) const values = await AsyncStorage.multiGet(sampleKeys); let estimatedSize = 0; values.forEach(([key, value]) => { estimatedSize += key.length + (value?.length || 0); }); // Extrapolate if (sampleKeys.length > 0) { estimatedSize = (estimatedSize / sampleKeys.length) * allKeys.length; } return { totalKeys: allKeys.length, sampleKeys, estimatedSize, }; } catch (error) { console.error('Error getting storage info:', error); return { totalKeys: 0, sampleKeys: [], estimatedSize: 0, }; } } // Migrate data (versioned storage) static async migrateData( oldKey: string, newKey: string, migrationFn: (oldData: any) => any ): Promise { try { const oldData = await AsyncStorage.getItem(oldKey); if (!oldData) return false; const parsedData = JSON.parse(oldData); const migratedData = migrationFn(parsedData); await AsyncStorage.setItem(newKey, JSON.stringify(migratedData)); await AsyncStorage.removeItem(oldKey); return true; } catch (error) { console.error('Error migrating data:', error); return false; } } }
```

Image Utilities

```
// Image handling and manipulation utilities import { Platform } from 'react-native'; import { getFullImageUrl } from '../services/driverService'; export class ImageUtils { // Get optimal image dimensions for display static getOptimalDimensions( originalWidth: number, originalHeight: number, maxWidth: number = 1024, maxHeight: number = 1024 ): { width: number; height: number } { let width = originalWidth; let height = originalHeight; if (width > maxWidth) { const ratio = maxWidth / width; width = maxWidth; height = height * ratio; } if (height > maxHeight) { const ratio = maxHeight / height; height = maxHeight; width = width * ratio; } return { width: Math.round(width), height: Math.round(height), }; } // Calculate file size in human-readable format static formatFileSize(bytes: number): string { if (bytes === 0) return '0 Bytes'; const k = 1024; const sizes = ['Bytes', 'KB', 'MB', 'GB']; const i = Math.floor(Math.log(bytes) / Math.log(k)); return parseFloat((bytes / Math.pow(k, i)).toFixed(2)) + ' ' + sizes[i]; } // Validate image file static validateImageFile(file: any): { isValid: boolean; error?: string; } { if (!file) { return { isValid: false, error: 'No file selected' }; } // Check file size (max 5MB) const maxSize = 5 * 1024 * 1024; // 5MB if (file.fileSize && file.fileSize > maxSize) { return { isValid: false, error: `File too large (max ${this.formatFileSize(maxSize)})` }; } // Check file type const allowedTypes = ['image/jpeg', 'image/jpg', 'image/png', 'image/heic', 'image/heif']; if (file.type && !allowedTypes.includes(file.type.toLowerCase())) { return { isValid: false, error: 'Invalid file type. Allowed: JPG, PNG, HEIC' }; } // Check dimensions
```

```
(optional) if (file.width && file.height) { const minDimension = 100; if
(file.width < minDimension || file.height < minDimension) { return { isValid:
false, error: `Image too small (min ${minDimension}x${minDimension})` }; } } return
{ isValid: true }; } // Create FormData for image upload static
createImageFormData( image: any, fieldName: string = 'document', additionalData:
Record = {} ): FormData { const formData = new FormData(); // Add additional data
Object.keys(additionalData).forEach(key => { if (additionalData[key] !== undefined
&& additionalData[key] !== null) { formData.append(key,
String(additionalData[key])); } }); // Add image file if (image && image.uri) {
const filename = image.uri.split('/').pop(); const match = /\.(\w+)$/.exec(filename
|| ''); const type = match ? `image/${match[1]}` : 'image/jpeg';
formData.append(fieldName, { uri: image.uri, type: image.type || type, name:
filename || 'image.jpg', } as any); } return formData; } // Get image source with
fallback static getImageSource( uri: string | null | undefined, fallbackSource: any
= require('../assets/images/placeholder.png') ): { uri: string } | any { if (!uri)
return fallbackSource; const fullUri = getFullImageUrl(uri); if (!fullUri ||
fullUri === '') return fallbackSource; return { uri: fullUri }; } // Preload images
static async preloadImages(imageUris: string[]): Promise { const Image =
require('react-native').Image; const preloadPromises = imageUris.map(uri =>
Image.prefetch(getFullImageUrl(uri)) ); await Promise.all(preloadPromises); } //
Generate placeholder color based on string static generatePlaceholderColor(str:
string): string { let hash = 0; for (let i = 0; i < str.length; i++) { hash =
str.charCodeAt(i) + ((hash << 5) - hash); } const colors = [ '#FF6B6B', '#4ECDC4',
'#45B7D1', '#96CEB4', '#FFEAA7', '#DDA0DD', '#98D8C8', '#F7DC6F', '#BB8FCE',
'#85C1E9', ]; return colors[Math.abs(hash) % colors.length]; } // Compress image
(wrapper for react-native-image-resizer if available) static async compressImage(
uri: string, quality: number = 0.8, maxWidth: number = 1024, maxHeight: number =
1024 ): Promise { try { // If react-native-image-resizer is available, use it if
(Platform.OS !== 'web') { const ImageResizer = require('react-native-image-
resizer').default; const response = await ImageResizer.createResizedImage( uri,
maxWidth, maxHeight, 'JPEG', quality * 100, 0 ); return response.uri; } return uri;
} catch (error) { console.error('Error compressing image:', error); return null; }
} }
```

Platform Utilities

```
// Platform-specific utilities import { Platform, Dimensions, StatusBar, PixelRatio
} from 'react-native'; export class PlatformUtils { // Check platform static
isIOS(): boolean { return Platform.OS === 'ios'; } static isAndroid(): boolean {
return Platform.OS === 'android'; } static isWeb(): boolean { return Platform.OS
=== 'web'; } // Get device info static getDeviceInfo(): { platform: string;
version: string | number; model: string | null; isTablet: boolean; isEmulator:
boolean; } { return { platform: Platform.OS, version: Platform.Version, model:
PlatformUtils.getDeviceModel(), isTablet: PlatformUtils.isTablet(), isEmulator:
PlatformUtils.isEmulator(), }; } // Check if device is tablet static isTablet():
boolean { const { width, height } = Dimensions.get('window'); const aspectRatio =
height / width; if (Platform.OS === 'ios') { return aspectRatio < 1.6; } //
Android: check pixel density and screen size const pixelDensity = PixelRatio.get();
const adjustedWidth = width * pixelDensity; const adjustedHeight = height *
pixelDensity; return (adjustedWidth >= 600 && adjustedHeight >= 600); } // Get
device model (simplified) static getDeviceModel(): string | null { if (Platform.OS
=== 'ios') { // iOS device detection would require native module return 'iPhone'; }
else if (Platform.OS === 'android') { // Android device detection would require
```

```
native module return 'Android Device'; } return null; } // Check if running on emulator/simulator static isEmulator(): boolean { if (Platform.OS === 'ios') { // iOS simulator check return Platform.isPad || Platform.isTVOS || !PlatformUtils.isRealDevice(); } else if (Platform.OS === 'android') { // Android emulator check return Platform.constants?.isTesting || !PlatformUtils.isRealDevice(); } return false; } // Simple real device check (would need native module for accuracy) private static isRealDevice(): boolean { // This is a simplified check // In production, you'd use a native module like react-native-device-info return true; } // Get status bar height static getStatusBarHeight(): number { if (Platform.OS === 'ios') { // iPhone X and later have different status bar heights const { height, width } = Dimensions.get('window'); const isIPhoneX = Platform.OS === 'ios' && !Platform.isPad && !Platform.isTVOS && (height >= 812 || width >= 812); return isIPhoneX ? 44 : 20; } else if (Platform.OS === 'android') { return StatusBar.currentHeight || 24; } return 0; } // Get bottom safe area (for iPhone X and later) static getBottomSafeArea(): number { if (Platform.OS === 'ios') { const { height, width } = Dimensions.get('window'); const isIPhoneX = Platform.OS === 'ios' && !Platform.isPad && !Platform.isTVOS && (height >= 812 || width >= 812); return isIPhoneX ? 34 : 0; } return 0; } // Check dark mode static isDarkMode(): boolean { // This would require Appearance API or a context // For now, return false as default return false; } // Platform-specific styles static platformStyles(iosStyle: any, androidStyle: any, webStyle?: any): any { if (Platform.OS === 'ios') return iosStyle; if (Platform.OS === 'android') return androidStyle; return webStyle || iosStyle; } // Platform-specific value static platformValue(iosValue: T, androidValue: T, webValue?: T): T { if (Platform.OS === 'ios') return iosValue; if (Platform.OS === 'android') return androidValue; return webValue || iosValue; } // Check if device has notch static hasNotch(): boolean { if (Platform.OS === 'ios') { const { height, width } = Dimensions.get('window'); return ( Platform.OS === 'ios' && !Platform.isPad && !Platform.isTVOS && (height >= 812 || width >= 812) ); } else if (Platform.OS === 'android') { // Android notch detection would require native module return false; } return false; } }
```

Utility Best Practices:

- **Pure Functions:** All utilities are pure functions with clear inputs/outputs
- **Error Handling:** Built-in error handling with graceful fallbacks
- **Type Safety:** Full TypeScript support with proper interfaces
- **Performance:** Optimized for mobile performance with memoization
- **Reusability:** Designed to be reusable across the application
- **Testing:** All utilities are unit testable in isolation
- **Documentation:** Well-documented with usage examples
- **Platform Awareness:** Platform-specific optimizations where needed

API Reference - Authentication

Login Driver

```
// Request POST /api/v1/loginDriver { "mobile": "9876543210" } // Response (Success) { "success": true, "message": "OTP sent successfully", "otpSent": true } // Response (Error - Invalid mobile) { "success": false, "message": "Invalid mobile number", "otpSent": false } // Response (Error - Already registered) { "success": false, "message": "Driver already registered", "otpSent": false } // Response (Error - Server error) { "success": false, "message": "Internal server error", "otpSent": false }
```

Verify OTP and Login

```
// Request POST /api/v1/verifyOtpAndLoginDriver { "mobile": "9876543210", "otp": "1234", "ip_address": "192.168.1.1" } // Response (Success) { "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", "user": { "id": 123, "mobile": "9876543210", "userType": "Driver", "is_emergency_driver": 1, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } } // Response (Error - Invalid OTP) { "success": false, "message": "Invalid OTP" } // Response (Error - OTP expired) { "success": false, "message": "OTP expired" } // Response (Error - Max attempts exceeded) { "success": false, "message": "Maximum OTP attempts exceeded" }
```

API Reference - Profile Management

Create/Update Driver Profile

```
// Request (Multipart Form Data) POST /api/v1/createDriverProfile Form Data: - driverId: "123" (required) - fullName: "John Doe" (required) - city: "Mumbai" (required) - pinCode: "400001" (required) - permanentAddress: "123 Main Street" (required) - dob: "1990-01-01" (required, format: YYYY-MM-DD) - State: "Maharashtra" (required) - email: "john@example.com" (required) - gender: "Male" (required, values: Male/Female/Other) - mobile: "9876543210" (required) - userProfile: [File] (optional, max 5MB, types: jpg/jpeg/png) // Response (Success) { "id": 456, "driverId": 123, "fullName": "John Doe", "date_of_birth": "1990-01-01", "profileImage": "/uploads/profiles/123/profile.jpg", "city": "Mumbai", "State": "Maharashtra", "pinCode": "400001", "permanentAddress": "123 Main Street", "gender": "Male", "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } // Response (Error - Validation) { "success": false, "message": "Validation failed", "errors": { "fullName": ["Full name is required"], "email": ["Invalid email format"] } } // Response (Error - Driver not found) { "success": false, "message": "Driver not found" }
```

Get Driver Profile By ID

```
// Request GET /api/v1/getDriverProfileById/123 // Response (Success) {
"DriverDetail": [ { "id": 123, "fullName": "John Doe", "email": "john@example.com",
"mobile": "9876543210", "userType": "Driver", "status": "Active", "userProfile":
"/uploads/profile.jpg", "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt":
"2024-01-01T00:00:00.000Z" } ], "DriverProfile": [ { "id": 456, "userId": 123,
"fullName": "John Doe", "date_of_birth": "1990-01-01", "profileImage":
"/uploads/profile.jpg", "city": "Mumbai", "State": "Maharashtra", "pinCode":
"400001", "permanentAddress": "123 Main Street", "gender": "Male", "createdAt":
"2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } ],
"DriverOnboardingStatuses": [ { "driverId": 123, "documentStatus": "approved",
"vehicleDocumentStatus": "pending", "bankStatus": "not uploaded", "profileStatus":
"approved", "submittedToAdmin": true, "approvedByAdmin": false, "rejectionReason":
null, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-
01T00:00:00.000Z" } ] } // Response (Error - Not found) { "success": false,
"message": "Driver profile not found" }
```

API Reference - Document Upload

Upload Driver Document

```
// Request (Multipart Form Data) POST /api/v1/createDriverDocument Form Data: -
driverId: "123" (required) - documentType: "pan" (required, values:
pan/aadhaar_front/aadhaar_back/driving_license) - documentNumber: "ABCDE1234F"
(required) - document: [File] (required, max 5MB, types: jpg/jpeg/png) // Response
(Success) { "id": 789, "driverId": 123, "documentType": "pan", "documentNumber":
"ABCDE1234F", "documentPath": "/uploads/documents/123/pan.jpg", "status":
"pending", "rejectionReason": null, "createdAt": "2024-01-01T00:00:00.000Z",
"updatedAt": "2024-01-01T00:00:00.000Z" } // Response (Error - Document already
uploaded) { "success": false, "message": "PAN document already uploaded" } //
Response (Error - Invalid document type) { "success": false, "message": "Invalid
document type" } // Response (Error - Invalid PAN format) { "success": false,
"message": "Invalid PAN number format" }
```

Get All Driver Documents

```
// Request GET /api/v1/getDriverAllDocument/123 // Response (Success) { "pan": {
"id": 789, "driverId": 123, "documentType": "pan", "documentNumber": "ABCDE1234F",
"documentPath": "/uploads/documents/123/pan.jpg", "status": "approved",
"rejectionReason": null, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt":
"2024-01-01T00:00:00.000Z" }, "aadhaar_front": { "id": 790, "driverId": 123,
"documentType": "aadhaar_front", "documentNumber": "1234 5678 9012",
"documentPath": "/uploads/documents/123/aadhaar_front.jpg", "status": "approved",
"rejectionReason": null, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt":
"2024-01-01T00:00:00.000Z" }, "aadhaar_back": { "id": 791, "driverId": 123,
"documentType": "aadhaar_back", "documentNumber": "1234 5678 9012", "documentPath":
```

```
"/uploads/documents/123/aadhaar_back.jpg", "status": "rejected", "rejectionReason": "Image is blurry, please upload a clear image", "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" }, "driving_license": null, "rc_front": { /* Vehicle document */ }, "rc_back": { /* Vehicle document */ }, "insurance": { /* Vehicle document */ }, "vehicle_photo": { /* Vehicle document */ }, "fitness_certificate": { /* Vehicle document */ }, "bank_passbook": { /* Bank document */ }, "cancelled_cheque": { /* Bank document */ } } // Response (Error - Driver not found) { "success": false, "message": "Driver not found" }
```

API Reference - Vehicle Management

Create Vehicle

```
// Request POST /api/v1/createDriverVehicle Body: { "driverId": 123, "brand": "Hero", "model": "Splendor", "registrationNumber": "MH01AB1234", "yearOfManufacture": "2020", "color": "Black" } // Response (Success) { "id": 101, "driverId": 123, "brand": "Hero", "model": "Splendor", "registrationNumber": "MH01AB1234", "yearOfManufacture": "2020", "color": "Black", "status": "pending", "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } // Response (Error - Vehicle already registered) { "success": false, "message": "Vehicle already registered for this driver" } // Response (Error - Invalid registration number) { "success": false, "message": "Invalid vehicle registration number" }
```

Upload Vehicle Document

```
// Request (Multipart Form Data) POST /api/v1/createDriverVehicleDocument Form Data: - driverId: "123" (required) - vehicleId: "101" (required) - documentType: "rc_front" (required, values: rc_front/rc_back/insurance/vehicle_photo/fitness_certificate) - expiry_date: "2025-12-31" (optional for insurance/fitness) - document: [File] (required, max 5MB, types: jpg/jpeg/png) // Response (Success) { "id": 202, "vehicleId": 101, "documentType": "rc_front", "documentPath": "/uploads/vehicles/101/rc_front.jpg", "expiry_date": null, "status": "pending", "rejectionReason": null, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } // Response (Error - Vehicle not found) { "success": false, "message": "Vehicle not found" } // Response (Error - Document already uploaded) { "success": false, "message": "RC Front already uploaded" }
```

API Reference - Emergency Contacts

Get All User Contacts

```
// Request GET /api/v1/getAllUserContacts Headers: Authorization: Bearer {token} // Response (Success) [ { "id": 301, "user_id": 123, "first_name": "Jane", "last_name": "Doe", "phone_number": "9876543211", "relationship": "Spouse", "is_primary": true, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" }, { "id": 302, "user_id": 123, "first_name": "John", "last_name": "Smith", "phone_number": "9876543212", "relationship": "Friend", "is_primary": false, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } ] // Response (Error - Unauthorized) { "success": false, "message": "Unauthorized" }
```

Create User Contact

```
// Request POST /api/v1/createUserContact Body: { "user_id": 123, "first_name": "Alice", "last_name": "Johnson", "phone_number": "9876543213", "relationship": "Sibling", "is_primary": false } // Response (Success) { "id": 303, "user_id": 123, "first_name": "Alice", "last_name": "Johnson", "phone_number": "9876543213", "relationship": "Sibling", "is_primary": false, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } // Response (Error - Maximum contacts reached) { "success": false, "message": "Maximum 5 emergency contacts allowed" } // Response (Error - Duplicate phone number) { "success": false, "message": "Contact with this phone number already exists" }
```

Update User Contact

```
// Request PUT /api/v1/updateUserContact/303 Body: { "first_name": "Alicia", "last_name": "Johnson", "relationship": "Sister", "is_primary": true } // Response (Success) { "id": 303, "user_id": 123, "first_name": "Alicia", "last_name": "Johnson", "phone_number": "9876543213", "relationship": "Sister", "is_primary": true, "createdAt": "2024-01-01T00:00:00.000Z", "updatedAt": "2024-01-01T00:00:00.000Z" } // Response (Error - Contact not found) { "success": false, "message": "Contact not found" }
```

Delete User Contact

```
// Request DELETE /api/v1/deleteUserContact/303 // Response (Success) { "success": true, "message": "Contact deleted successfully" } // Response (Error - Cannot delete primary contact) { "success": false, "message": "Cannot delete primary contact. Set another contact as primary first." } // Response (Error - Contact not found) { "success": false, "message": "Contact not found" }
```

BonJoy Driver Application Documentation v1.0.0

Generated on December 4, 2024

This documentation covers the complete architecture, implementation, and API reference for the BonJoy Driver React Native application.