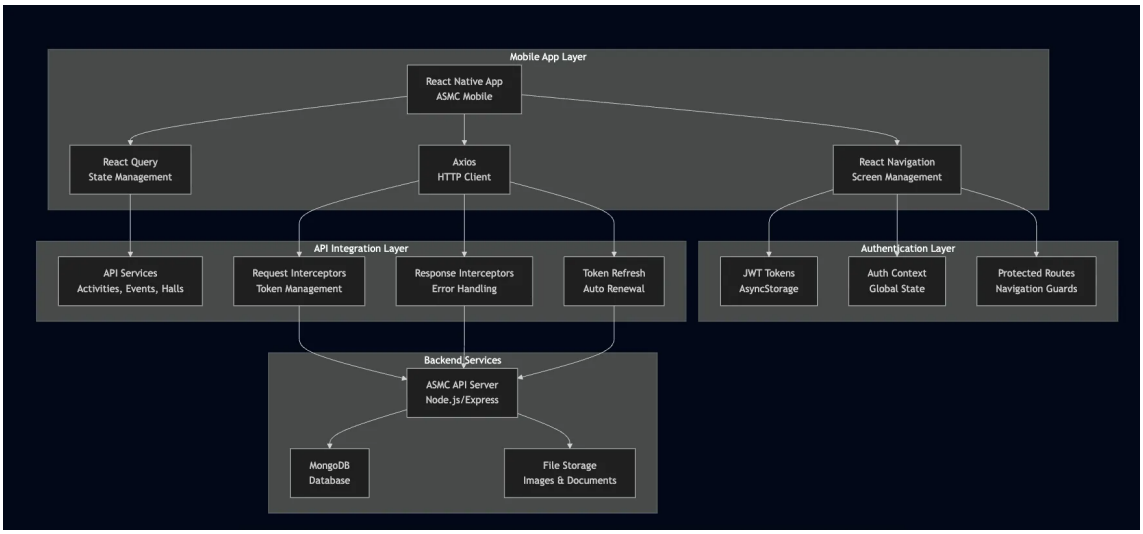
ASMC Mobile App - Architecture Overview

Comprehensive architectural documentation for the ASMC Mobile React Native application, covering system design, component architecture, state management, and integration patterns.

Table of Contents

System Architecture

High-Level Architecture



Technology Stack

Layer	Technology	Purpose
Frontend Framework	React Native 0.79.5	Cross-platform mobile app
State Management	React Query (TanStack)	Server state and caching
Navigation	React Navigation 7.x	Screen navigation
HTTP Client	Axios 1.10.0	API communication
Form Management	Formik + Yup	Form handling and validation

Storage	AsyncStorage	Local data persistence
UI Components	Custom Components	Native mobile components
Icons	React Native Vector Icons	Icon library
Calendar	React Native Calendars	Calendar functionality
Image Handling	React Native Image Picker	Image selection and upload
WebView	React Native WebView	External web content

Architecture Patterns

1. Container-Component Pattern

```
// Container (Business Logic)
const HomeContainer = () => {
    const { data: activities, isLoading } = useActivities();
    const { data: events } = useEvents();
    const { user } = useAuth();

    const handleActivityPress = (activityId) => {
        navigation.navigate('ActivityDetail', { id: activityId });
    };

    return (
        <HomeComponent
            activities={activities}
            events={events}
            user={user}
            loading={isLoading}
            onActivityPress={handleActivityPress}
        />
    );
};

// Component (Presentation)
const HomeComponent = ({ activities, events, user, loading, onActivityPress }) => {
    if (loading) return <LoadingSpinner />;

    return (
        <ScrollView style={styles.container}>
            <WelcomeSection user={user} />
            <ActivitiesSection activities={activities} onPress={onActivityPress} />
            <EventsSection events={events} />
        </ScrollView>
    );
};
```

2. Context Provider Pattern

```
// Context Provider
export const AuthProvider = ({ children }) => {
```

```
    const [state, dispatch] = useReducer(authReducer, initialState);

    const value = {
        ...state,
        login: (credentials) => loginAction(dispatch, credentials),
        logout: () => logoutAction(dispatch),
    };

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

// Context Consumer Hook
export const useAuth = () => {
    const context = useContext(AuthContext);
    if (!context) {
        throw new Error('useAuth must be used within AuthProvider');
    }
    return context;
};
```

3. Custom Hooks Pattern

```
// Custom Hook for API Integration
export const useActivities = () => {
    return useQuery({
        queryKey: ['activities'],
        queryFn: () => activitiesAPI.getActivities(),
        staleTime: 5 * 60 * 1000, // 5 minutes
        cacheTime: 10 * 60 * 1000, // 10 minutes
    });
};

// Usage in Component
const ActivitiesScreen = () => {
    const { data: activities, isLoading, error } = useActivities();
    const enrollMutation = useEnrollActivity();

    const handleEnroll = (activityId) => {
        enrollMutation.mutate(activityId);
    };

    return (
        <View>
            {isLoading ? (
                <LoadingSpinner />
            ) : (
                <ActivitiesList activities={activities} onEnroll={handleEnroll} />
            )}
        </View>
    );
};
```

Component Architecture

Component Hierarchy

```
App
├── QueryClientProvider
├── AuthProvider
├── SafeAreaProvider
└── RoutesContainer
    ├── NavigationContainer
    ├── PublicNavigator (Login, ForgotPassword)
    └── PrivateNavigator
        ├── TabNavigator
        │   ├── HomeTab
        │   ├── EventsTab
        │   ├── HallsTab
        │   ├── ActivitiesTab
        │   └── ProfileTab
        └── StackNavigator
            ├── ActivityDetail
            ├── EventDetail
            ├── BookingForm
            ├── EditProfile
            └── Other Screens
```

Component Structure

1. Screen Components

Location: `/src/containers/`

```
// Screen Container Structure
const ScreenContainer = () => {
    // Hooks and State
    const { data, isLoading, error } = useScreenData();
    const [localState, setLocalState] = useState(initialState);

    // Event Handlers
    const handleAction = useCallback(
        (params) => {
            // Action logic
        },
        [dependencies],
    );

    // Render Logic
    if (isLoading) return <LoadingScreen />;
    if (error) return <ErrorScreen error={error} />;

    return <ScreenComponent data={data} onAction={handleAction} />;
};
```

```
export default ScreenContainer;
```

2. Reusable Components

Location: `/src/components/common/`

```
// Common Component Structure
const CommonComponent = ({ prop1, prop2, onAction, style, ...props }) => {
    // Component logic
    const handlePress = () => {
        onAction?.(prop1, prop2);
    };

    return (
        <TouchableOpacity
            style={[styles.container, style]}
            onPress={handlePress}
            {...props}
        >
            {/* Component content */}
        </TouchableOpacity>
    );
};

// PropTypes (if using TypeScript, use interfaces)
CommonComponent.propTypes = {
    prop1: PropTypes.string.isRequired,
    prop2: PropTypes.number,
    onAction: PropTypes.func,
    style: PropTypes.object,
};

export default CommonComponent;
```

Component Communication

1. Props Down, Events Up

```
// Parent Component
const ParentComponent = () => {
    const [selectedItem, setSelectedItem] = useState(null);

    const handleItemSelect = (item) => {
        setSelectedItem(item);
    };

    return <ChildComponent selectedItem={selectedItem} onItemSelect={handleItemSelect}
/>;
};

// Child Component
const ChildComponent = ({ selectedItem, onItemSelect }) => {
```

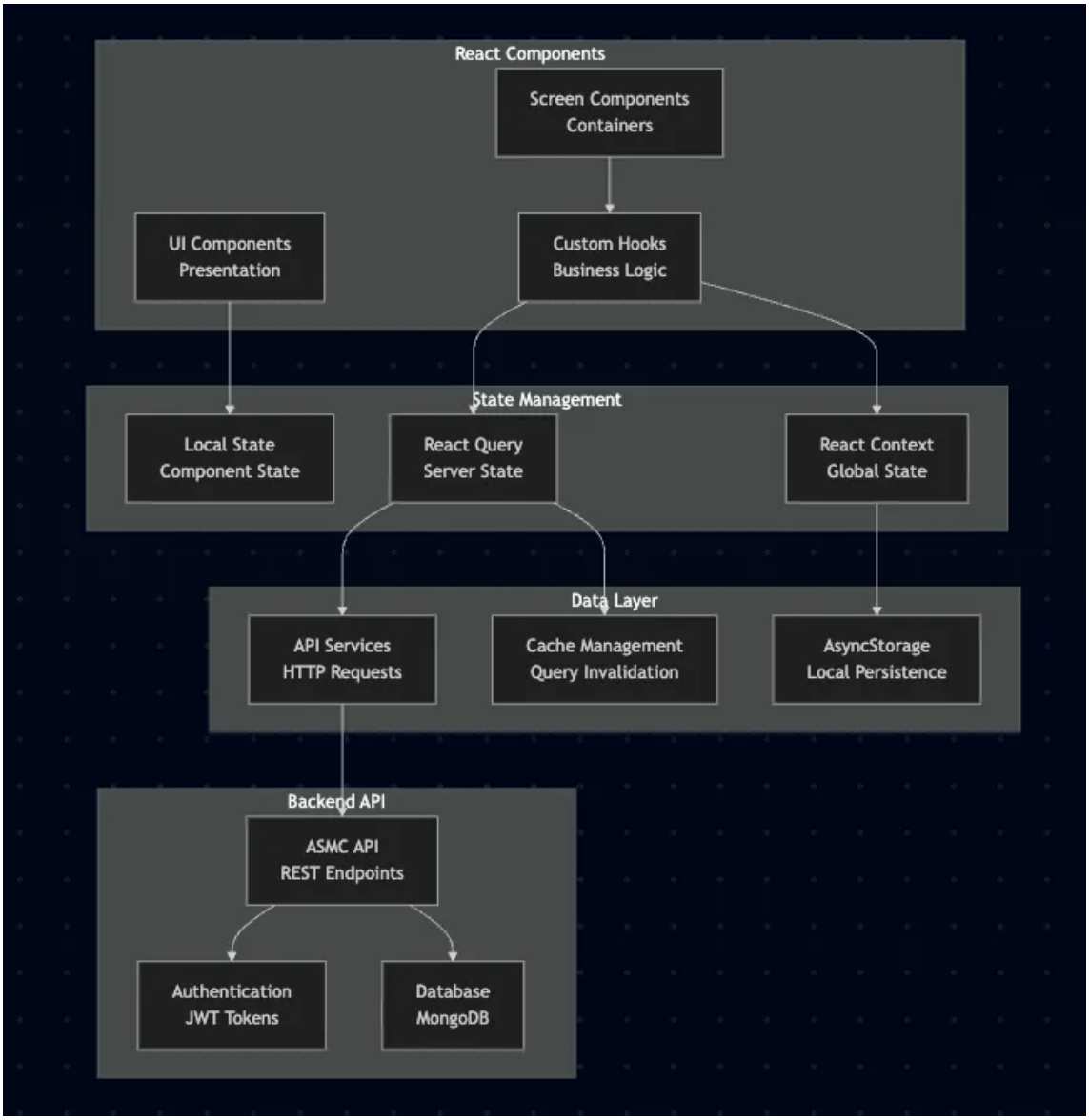
```
    const handlePress = (item) => {
        onItemSelect(item);
    };

    return (
        <TouchableOpacity onPress={() => handlePress(item)}>
            <Text>{item.name}</Text>
        </TouchableOpacity>
    );
};
```

2. Context Communication

```
// Global State Access
const ComponentUsingContext = () => {
    const { user, login, logout } = useAuth();
    const { data: activities } = useActivities();

    return (
        <View>
            <Text>Welcome, {user?.name}</Text>
            <Button title="Logout" onPress={logout} />
        </View>
    );
};
```

State Management Architecture

State Management Layers



1. Server State (React Query)

```
// Query Client Configuration
export const queryClient = new QueryClient({
    defaultOptions: {
        queries: {
            staleTime: 5 * 60 * 1000, // 5 minutes
            cacheTime: 10 * 60 * 1000, // 10 minutes
            retry: 2,
            refetchOnWindowFocus: false,
            refetchOnReconnect: true,
        },
        mutations: {
            retry: 1,
            onError: (error) => {
```

```
                handleAPIError(error);
            },
        },
    },
});

// Query Definitions
export const useActivities = (filters = {}) => {
    return useQuery({
        queryKey: ['activities', filters],
        queryFn: () => activitiesAPI.getActivities(filters),
        select: (data) => data.activities,
    });
};

export const useActivity = (id) => {
    return useQuery({
        queryKey: ['activity', id],
        queryFn: () => activitiesAPI.getActivity(id),
        enabled: !!id,
    });
};

// Mutation Definitions
export const useEnrollActivity = () => {
    const queryClient = useQueryClient();

    return useMutation({
        mutationFn: (activityId) => activitiesAPI.enrollActivity(activityId),
        onSuccess: (data, activityId) => {
            // Invalidate related queries
            queryClient.invalidateQueries(['activities']);
            queryClient.invalidateQueries(['enrolled-activities']);

            // Update cache optimistically
            queryClient.setQueryData(['activity', activityId], (oldData) => ({
                ...oldData,
                isEnrolled: true,
            }));

            showSuccessToast('Successfully enrolled in activity');
        },
        onError: (error) => {
            handleAPIError(error);
        },
    });
};
```

2. Global State (React Context)

```
// Auth Context
const AuthContext = createContext();
```

```
const authReducer = (state, action) => {
    switch (action.type) {
        case 'LOGIN_SUCCESS':
            return {
                ...state,
                user: action.payload.user,
                token: action.payload.token,
                isAuth: true,
                isLoading: false,
            };
        case 'LOGOUT':
            return {
                ...state,
                user: null,
                token: null,
                isAuth: false,
                isLoading: false,
            };
        case 'UPDATE_PROFILE':
            return {
                ...state,
                user: { ...state.user, ...action.payload },
            };
        default:
            return state;
    }
};

export const AuthProvider = ({ children }) => {
    const [state, dispatch] = useReducer(authReducer, initialState);

    const login = async (credentials) => {
        try {
            dispatch({ type: 'SET_LOADING', payload: true });

            const response = await authAPI.login(credentials);
            const { token, user } = response.data;

            // Store tokens
            await AsyncStorage.setItem('authToken', token);
            await AsyncStorage.setItem('userData', JSON.stringify(user));

            dispatch({
                type: 'LOGIN_SUCCESS',
                payload: { token, user },
            });

            return { success: true };
        } catch (error) {
            dispatch({ type: 'SET_LOADING', payload: false });
            return { success: false, error: error.response?.data?.message };
```

```
        }
    };

    const logout = async () => {
        try {
            await authAPI.logout();
        } catch (error) {
            console.error('Logout API error:', error);
        } finally {
            // Clear local storage
            await AsyncStorage.removeItem('authToken');
            await AsyncStorage.removeItem('userData');

            dispatch({ type: 'LOGOUT' });
        }
    };

    const updateProfile = (profileData) => {
        dispatch({
            type: 'UPDATE_PROFILE',
            payload: profileData,
        });
    };

    const value = {
        ...state,
        login,
        logout,
        updateProfile,
    };

    return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};
```

3. Local State (React Hooks)

```
// Local Component State
const ActivityScreen = () => {
    // Local state for UI
    const [selectedFilter, setSelectedFilter] = useState('all');
    const [searchQuery, setSearchQuery] = useState('');
    const [showFilters, setShowFilters] = useState(false);

    // Server state
    const { data: activities, isLoading } = useActivities({
        filter: selectedFilter,
        search: searchQuery,
    });

    // Computed state
    const filteredActivities = useMemo(() => {
        return (
```

```
            activities?.filter((activity) =>
                activity.name.toLowerCase().includes(searchQuery.toLowerCase()),
            ) || []
        );
    }, [activities, searchQuery]);

    // Event handlers
    const handleFilterChange = (filter) => {
        setSelectedFilter(filter);
        setShowFilters(false);
    };

    const handleSearch = (query) => {
        setSearchQuery(query);
    };

    return (
        <View>
            <SearchBar value={searchQuery} onChangeText={handleSearch} />
            <FilterButton onPress={() => setShowFilters(!showFilters)} />

            {showFilters && (
                <FilterModal
                    selectedFilter={selectedFilter}
                    onFilterChange={handleFilterChange}
                />
            )}

            <ActivitiesList activities={filteredActivities} />
        </View>
    );
};
```

API Integration Architecture

API Client Configuration

```
// Axios Configuration
const axiosInstance = axios.create({
    baseURL: baseUrl,
    timeout: 30000,
    headers: {
        'Content-Type': 'application/json',
    },
});

// Request Interceptor
axiosInstance.interceptors.request.use(
    async (config) => {
        // Add authentication token
        const token = await AsyncStorage.getItem('authToken');
        if (token) {
```

```
            config.headers.Authorization = `Bearer ${token}`;
        }

        // Add request timestamp for debugging
        config.metadata = { startTime: new Date() };

        return config;
    },
    (error) => Promise.reject(error),
);

// Response Interceptor
axiosInstance.interceptors.response.use(
    (response) => {
        // Log request duration
        const duration = new Date() - response.config.metadata.startTime;
        console.log(
            `API Request: ${response.config.method?.toUpperCase()} ${
                response.config.url
            } - ${duration}ms`,
        );

        return response;
    },
    async (error) => {
        const originalRequest = error.config;

        // Handle 401 errors (token expiration)
        if (error.response?.status === 401 && !originalRequest._retry) {
            originalRequest._retry = true;

            try {
                const refreshToken = await AsyncStorage.getItem('refreshToken');
                if (refreshToken) {
                    const response = await axiosInstance.post('/auth/refresh', {
                        refreshToken,
                    });

                    const { token, refreshToken: newRefreshToken } = response.data;

                    await AsyncStorage.setItem('authToken', token);
                    await AsyncStorage.setItem('refreshToken', newRefreshToken);

                    // Retry original request
                    originalRequest.headers.Authorization = `Bearer ${token}`;
                    return axiosInstance(originalRequest);
                }
            } catch (refreshError) {
                // Refresh failed, redirect to login
                await AsyncStorage.removeItem('authToken');
                await AsyncStorage.removeItem('refreshToken');
                // Navigate to login screen
```

```
            }
        }

        return Promise.reject(error);
    },
);
```

API Service Layer

```
// API Service Functions
export const authAPI = {
    login: (credentials) => axiosInstance.post('/auth/login', credentials),
    logout: () => axiosInstance.post('/auth/logout'),
    refreshToken: (refreshToken) => axiosInstance.post('/auth/refresh', { refreshToken
}),
    getProfile: () => axiosInstance.get('/auth/profile'),
    updateProfile: (data) => axiosInstance.put('/auth/profile', data),
    changePassword: (data) => axiosInstance.put('/auth/change-password', data),
};

export const activitiesAPI = {
    getActivities: (params) => axiosInstance.get('/activities', { params }),
    getActivity: (id) => axiosInstance.get(`/activities/${id}`),
    enrollActivity: (id) => axiosInstance.post(`/activities/${id}/enroll`),
    unenrollActivity: (id) => axiosInstance.delete(`/activities/${id}/enroll`),
    getEnrolledActivities: () => axiosInstance.get('/activities/enrolled'),
};

export const eventsAPI = {
    getEvents: (params) => axiosInstance.get('/events', { params }),
    getEvent: (id) => axiosInstance.get(`/events/${id}`),
    registerEvent: (id) => axiosInstance.post(`/events/${id}/register`),
    unregisterEvent: (id) => axiosInstance.delete(`/events/${id}/register`),
    getRegisteredEvents: () => axiosInstance.get('/events/registered'),
};

export const hallsAPI = {
    getHalls: (params) => axiosInstance.get('/halls', { params }),
    getHall: (id) => axiosInstance.get(`/halls/${id}`),
    getHallAvailability: (id, date) =>
        axiosInstance.get(`/halls/${id}/availability`, {
            params: { date },
        }),
    bookHall: (data) => axiosInstance.post('/halls/book', data),
    cancelBooking: (id) => axiosInstance.delete(`/halls/bookings/${id}`),
    getBookings: () => axiosInstance.get('/halls/bookings'),
};

export const paymentsAPI = {
    getPaymentHistory: () => axiosInstance.get('/payments/history'),
    processPayment: (data) => axiosInstance.post('/payments/process', data),
```

```
    getPaymentMethods: () => axiosInstance.get('/payments/methods'),
};
```

Error Handling Strategy

```
// Error Handling Utilities
export const handleAPIError = (error) => {
    const message = error.response?.data?.message || 'An error occurred';
    const statusCode = error.response?.status;

    // Log error for debugging
    console.error('API Error:', {
        message,
        statusCode,
        url: error.config?.url,
        method: error.config?.method,
    });

    // Show appropriate error message
    switch (statusCode) {
        case 400:
            Toast.show({
                type: 'error',
                text1: 'Invalid Request',
                text2: message,
            });
            break;
        case 401:
            Toast.show({
                type: 'error',
                text1: 'Authentication Required',
                text2: 'Please login again',
            });
            break;
        case 403:
            Toast.show({
                type: 'error',
                text1: 'Access Denied',
                text2: 'You do not have permission to perform this action',
            });
            break;
        case 404:
            Toast.show({
                type: 'error',
                text1: 'Not Found',
                text2: 'The requested resource was not found',
            });
            break;
        case 500:
            Toast.show({
                type: 'error',
                text1: 'Server Error',
```

```
                text2: 'Please try again later',
            });
            break;
        default:
            Toast.show({
                type: 'error',
                text1: 'Error',
                text2: message,
            });
    }
};

// Success Handler
export const handleAPISuccess = (message) => {
    Toast.show({
        type: 'success',
        text1: 'Success',
        text2: message,
    });
};
```

Navigation Architecture

Navigation Structure

```
// Navigation Configuration
const NavigationContainer = () => {
    const { isAuth, isLoading } = useAuth();

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

    return (
        <NavigationContainer>
            {isAuth ? <PrivateNavigator /> : <PublicNavigator />}
        </NavigationContainer>
    );
};

// Public Navigation (Unauthenticated)
const PublicNavigator = () => (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
        <Stack.Screen name="Login" component={LoginContainer} />
        <Stack.Screen name="ForgotPassword" component={ForgotPasswordContainer} />
    </Stack.Navigator>
);

// Private Navigation (Authenticated)
const PrivateNavigator = () => (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
        <Stack.Screen name="TabNavigator" component={TabNavigator} />
```

```
        {/* Modal Screens */}
        <Stack.Screen
            name="ActivityDetail"
            component={ActivityContainer}
            options={{
                presentation: 'modal',
                headerShown: true,
                header: () => <Header showBack={true} title="Activity Details" />,
            }}
        />

        <Stack.Screen
            name="EventDetail"
            component={EventDetailContainer}
            options={{
                presentation: 'modal',
                headerShown: true,
                header: () => <Header showBack={true} title="Event Details" />,
            }}
        />

        {/* Full Screen Modals */}
        <Stack.Screen
            name="BookingForm"
            component={BookingForm}
            options={{
                presentation: 'fullScreenModal',
                headerShown: true,
                header: () => <Header showBack={true} title="Book Hall" />,
            }}
        />
    </Stack.Navigator>
);

// Tab Navigation
const TabNavigator = () => {
    const navConfig = useNavigationConfig();

    return (
        <Tab.Navigator
            initialRouteName="Home"
            screenOptions={({ route }) => ({
                tabBarIcon: ({ focused, color, size }) => {
                    return (
                        <TabIcon
                            route={route}
                            focused={focused}
                            color={color}
                            size={size}
                        />
                    );
```

```
                },
                tabBarActiveTintColor: '#1976d2',
                tabBarInactiveTintColor: 'gray',
                tabBarStyle: getTabBarStyle(navConfig),
                headerShown: true,
                header: () => <Header showLogout={true} />,
            })}
        >
            <Tab.Screen name="Home" component={HomeContainer} />
            <Tab.Screen name="Events" component={EventsContainer} />
            <Tab.Screen name="Halls" component={HallsContainer} />
            <Tab.Screen name="Activities" component={ActivitiesContainer} />
            <Tab.Screen name="Profile" component={ProfileContainer} />
        </Tab.Navigator>
    );
};
```

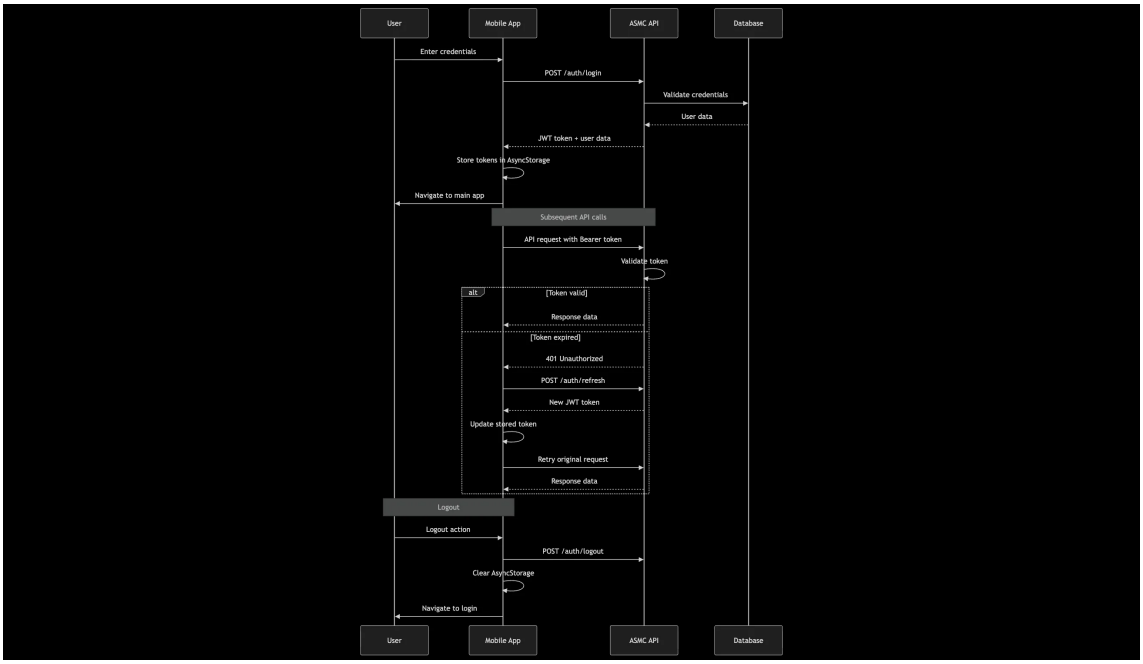
Navigation Utilities

```
// Navigation Helper Functions
export const useNavigationConfig = () => {
    const [keyboardHeight, setKeyboardHeight] = useState(0);
    const [bottomPadding, setBottomPadding] = useState(0);

    useEffect(() => {
        const keyboardDidShowListener = Keyboard.addListener('keyboardDidShow', (e) =>
{
            setKeyboardHeight(e.endCoordinates.height);
        });

        const keyboardDidHideListener = Keyboard.addListener('keyboardDidHide', () =>
{
            setKeyboardHeight(0);
        });

        return () => {
            keyboardDidShowListener.remove();
            keyboardDidHideListener.remove();
        };
    }, []);

    return {
        keyboardHeight,
        bottomPadding,
        needsBottomPadding: bottomPadding > 0,
    };
};

export const getTabBarStyle = (navConfig) => ({
    height: 60,
    paddingBottom: navConfig.needsBottomPadding ? navConfig.bottomPadding : 8,
    backgroundColor: '#ffffff',
```

```
    borderTopWidth: 1,
    borderTopColor: '#e0e0e0',
});

export const getSafeAreaStyle = (navConfig) => ({
    flex: 1,
    backgroundColor: '#fafafa',
    paddingBottom: navConfig.keyboardHeight > 0 ? navConfig.keyboardHeight : 0,
});
```

Authentication Architecture

Authentication Flow



JWT Token Management

```
// Token Storage Utilities
export const TokenManager = {
    // Store tokens
    async setTokens(token, refreshToken) {
        try {
            await AsyncStorage.setItem('authToken', token);
            await AsyncStorage.setItem('refreshToken', refreshToken);
            await AsyncStorage.setItem('tokenTimestamp', Date.now().toString());
        } catch (error) {
            console.error('Error storing tokens:', error);
        }
    },

    // Get access token
```

```
    async getAccessToken() {
        try {
            const token = await AsyncStorage.getItem('authToken');
            const timestamp = await AsyncStorage.getItem('tokenTimestamp');

            if (!token || !timestamp) return null;

            // Check if token is expired (assuming 1 hour expiry)
            const tokenAge = Date.now() - parseInt(timestamp);
            const isExpired = tokenAge > 3600000; // 1 hour in milliseconds

            if (isExpired) {
                await this.clearTokens();
                return null;
            }

            return token;
        } catch (error) {
            console.error('Error getting access token:', error);
            return null;
        }
    },

    // Get refresh token
    async getRefreshToken() {
        try {
            return await AsyncStorage.getItem('refreshToken');
        } catch (error) {
            console.error('Error getting refresh token:', error);
            return null;
        }
    },

    // Clear all tokens
    async clearTokens() {
        try {
            await AsyncStorage.multiRemove([
                'authToken',
                'refreshToken',
                'tokenTimestamp',
                'userData',
            ]);
        } catch (error) {
            console.error('Error clearing tokens:', error);
        }
    },

    // Check if user is authenticated
    async isAuthenticated() {
        const token = await this.getAccessToken();
        return !!token;
```

```
    },
};
```

Authentication Guards

```
// Route Protection
const ProtectedRoute = ({ children, requireAuth = true }) => {
    const { isAuth, isLoading } = useAuth();

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

    if (requireAuth && !isAuth) {
        return <NavigateToLogin />;
    }

    if (!requireAuth && isAuth) {
        return <NavigateToHome />;
    }

    return children;
};

// Permission-based Access
const RequirePermission = ({ permission, children, fallback }) => {
    const { user } = useAuth();

    const hasPermission = useMemo(() => {
        if (!user || !user.permissions) return false;
        return user.permissions.includes(permission);
    }, [user, permission]);

    if (!hasPermission) {
        return fallback || <AccessDeniedScreen />;
    }

    return children;
};

// Usage Examples
const ActivitiesScreen = () => (
    <ProtectedRoute>
        <RequirePermission permission="activities:read">
            <ActivitiesList />
        </RequirePermission>
    </ProtectedRoute>
);

const AdminScreen = () => (
    <ProtectedRoute>
        <RequirePermission permission="admin:access" fallback={<UnauthorizedScreen
```

```
/>}>
            <AdminPanel />
        </RequirePermission>
    </ProtectedRoute>
);
```

UI Architecture

Design System

```
// Design System Configuration
export const designSystem = {
    colors: {
        primary: {
            50: '#e3f2fd',
            100: '#bbdefb',
            200: '#90caf9',
            300: '#64b5f6',
            400: '#42a5f5',
            500: '#2196f3', // Main primary
            600: '#1e88e5',
            700: '#1976d2',
            800: '#1565c0',
            900: '#0d47a1',
        },
        secondary: {
            50: '#fce4ec',
            100: '#f8bbd9',
            200: '#f48fb1',
            300: '#f06292',
            400: '#ec407a',
            500: '#e91e63', // Main secondary
            600: '#d81b60',
            700: '#c2185b',
            800: '#ad1457',
            900: '#880e4f',
        },
        neutral: {
            50: '#fafafa',
            100: '#f5f5f5',
            200: '#eeeeee',
            300: '#e0e0e0',
            400: '#bdbdbd',
            500: '#9e9e9e',
            600: '#757575',
            700: '#616161',
            800: '#424242',
            900: '#212121',
        },
        semantic: {
            success: '#4caf50',
            warning: '#ff9800',
```

```
            error: '#f44336',
            info: '#2196f3',
        },
    },
    typography: {
        fontFamily: {
            regular: 'PlusJakartaSans-Regular',
            medium: 'PlusJakartaSans-Medium',
            semiBold: 'PlusJakartaSans-SemiBold',
            bold: 'PlusJakartaSans-Bold',
        },
        fontSize: {
            xs: 12,
            sm: 14,
            base: 16,
            lg: 18,
            xl: 20,
            '2xl': 24,
            '3xl': 32,
            '4xl': 40,
        },
        lineHeight: {
            tight: 1.2,
            normal: 1.5,
            relaxed: 1.75,
        },
    },
    spacing: {
        xs: 4,
        sm: 8,
        md: 16,
        lg: 24,
        xl: 32,
        '2xl': 48,
        '3xl': 64,
    },
    borderRadius: {
        sm: 4,
        md: 8,
        lg: 12,
        xl: 16,
        full: 9999,
    },
    shadows: {
        sm: {
            shadowColor: '#000',
            shadowOffset: { width: 0, height: 1 },
            shadowOpacity: 0.05,
            shadowRadius: 2,
            elevation: 1,
        },
        md: {
```

```
            shadowColor: '#000',
            shadowOffset: { width: 0, height: 2 },
            shadowOpacity: 0.1,
            shadowRadius: 4,
            elevation: 2,
        },
        lg: {
            shadowColor: '#000',
            shadowOffset: { width: 0, height: 4 },
            shadowOpacity: 0.15,
            shadowRadius: 8,
            elevation: 4,
        },
    },
};
```

Component Library

```
// Base Button Component
const Button = ({
    title,
    variant = 'primary',
    size = 'md',
    onPress,
    disabled = false,
    loading = false,
    style,
    ...props
}) => {
    const buttonStyle = [
        styles.button,
        styles[`button_${variant}`],
        styles[`button_${size}`],
        disabled && styles.button_disabled,
        style,
    ];

    const textStyle = [
        styles.buttonText,
        styles[`buttonText_${variant}`],
        styles[`buttonText_${size}`],
        disabled && styles.buttonText_disabled,
    ];

    return (
        <TouchableOpacity
            style={buttonStyle}
            onPress={onPress}
            disabled={disabled || loading}
            {...props}
        >
            {loading ? (
```

```
                <ActivityIndicator
                    color={variant === 'primary' ? '#ffffff' : '#1976d2'}
                    size="small"
                />
            ) : (
                <Text style={textStyle}>{title}</Text>
            )}
        </TouchableOpacity>
    );
};

// Card Component
const Card = ({ children, style, onPress, ...props }) => {
    const CardComponent = onPress ? TouchableOpacity : View;

    return (
        <CardComponent style={[styles.card, style]} onPress={onPress} {...props}>
            {children}
        </CardComponent>
    );
};

// Input Component
const Input = ({
    label,
    value,
    onChangeText,
    placeholder,
    error,
    secureTextEntry = false,
    keyboardType = 'default',
    style,
    ...props
}) => {
    return (
        <View style={styles.inputContainer}>
            {label && <Text style={styles.inputLabel}>{label}</Text>}
            <TextInput
                style={[styles.input, error && styles.inputError, style]}
                value={value}
                onChangeText={onChangeText}
                placeholder={placeholder}
                secureTextEntry={secureTextEntry}
                keyboardType={keyboardType}
                placeholderTextColor="#9e9e9e"
                {...props}
            />
            {error && <Text style={styles.inputErrorText}>{error}</Text>}
        </View>
    );
};
```

Performance Architecture

Performance Optimization Strategies

1. Image Optimization

```
// Image Component with Optimization
const OptimizedImage = ({ source, style, resizeMode = 'cover', ...props }) => {
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(false);

    const handleLoadStart = () => setLoading(true);
    const handleLoadEnd = () => setLoading(false);
    const handleError = () => {
        setError(true);
        setLoading(false);
    };

    return (
        <View style={[style, styles.imageContainer]}>
            <Image
                source={source}
                style={[style, styles.image]}
                resizeMode={resizeMode}
                onLoadStart={handleLoadStart}
                onLoadEnd={handleLoadEnd}
                onError={handleError}
                {...props}
            />

            {loading && (
                <View style={styles.imageLoadingContainer}>
                    <ActivityIndicator size="small" color="#1976d2" />
                </View>
            )}

            {error && (
                <View style={styles.imageErrorContainer}>
                    <Text style={styles.imageErrorText}>Failed to load image</Text>
                </View>
            )}
        </View>
    );
};

// Lazy Loading List
const LazyLoadingList = ({ data, renderItem, ...props }) => {
    const [visibleItems, setVisibleItems] = useState(10);

    const handleLoadMore = () => {
        setVisibleItems((prev) => Math.min(prev + 10, data.length));
    };
```

```
    return (
        <FlatList
            data={data.slice(0, visibleItems)}
            renderItem={renderItem}
            onEndReached={handleLoadMore}
            onEndReachedThreshold={0.5}
            keyExtractor={(item) => item.id.toString()}
            {...props}
        />
    );
};
```

2. Memory Management

```
// Memory-efficient Component
const MemoryEfficientComponent = React.memo(({ data, onAction }) => {
    // Use useCallback for event handlers
    const handleAction = useCallback(
        (item) => {
            onAction(item);
        },
        [onAction],
    );

    // Use useMemo for expensive computations
    const processedData = useMemo(() => {
        return data.map((item) => ({
            ...item,
            processed: expensiveComputation(item),
        }));
    }, [data]);

    return (
        <View>
            {processedData.map((item) => (
                <ItemComponent key={item.id} item={item} onAction={handleAction} />
            ))}
        </View>
    );
});

// Cleanup Hook
const useCleanup = () => {
    useEffect(() => {
        return () => {
            // Cleanup subscriptions, timers, etc.
            console.log('Component unmounting, cleaning up...');
        };
    }, []);
};
```

3. Network Optimization

```
// Request Debouncing
const useDebounce = (value, delay) => {
    const [debouncedValue, setDebouncedValue] = useState(value);

    useEffect(() => {
        const handler = setTimeout(() => {
            setDebouncedValue(value);
        }, delay);

        return () => {
            clearTimeout(handler);
        };
    }, [value, delay]);

    return debouncedValue;
};

// Search with Debouncing
const SearchComponent = () => {
    const [searchQuery, setSearchQuery] = useState('');
    const debouncedQuery = useDebounce(searchQuery, 500);

    const { data: searchResults, isLoading } = useSearch(debouncedQuery);

    return (
        <View>
            <TextInput
                value={searchQuery}
                onChangeText={setSearchQuery}
                placeholder="Search..."
            />
            {isLoading && <LoadingSpinner />}
            <SearchResults results={searchResults} />
        </View>
    );
};
```

Security Architecture

Security Measures

1. Data Encryption

```
// Secure Storage
import EncryptedStorage from 'react-native-encrypted-storage';

export const SecureStorage = {
    async setItem(key, value) {
        try {
            await EncryptedStorage.setItem(key, JSON.stringify(value));
```

```
        } catch (error) {
            console.error('Error storing secure data:', error);
        }
    },

    async getItem(key) {
        try {
            const value = await EncryptedStorage.getItem(key);
            return value ? JSON.parse(value) : null;
        } catch (error) {
            console.error('Error retrieving secure data:', error);
            return null;
        }
    },

    async removeItem(key) {
        try {
            await EncryptedStorage.removeItem(key);
        } catch (error) {
            console.error('Error removing secure data:', error);
        }
    },

    async clear() {
        try {
            await EncryptedStorage.clear();
        } catch (error) {
            console.error('Error clearing secure storage:', error);
        }
    },
};
```

2. Input Validation

```
// Input Sanitization
export const sanitizeInput = (input) => {
    if (typeof input !== 'string') return input;

    // Remove potentially dangerous characters
    return input
        .replace(/[<>]/g, '') // Remove HTML tags
        .replace(/javascript:/gi, '') // Remove javascript: protocol
        .replace(/on\w+=/gi, '') // Remove event handlers
        .trim();
};

// Form Validation
export const validateForm = (data, schema) => {
    const errors = {};

    Object.keys(schema).forEach((field) => {
        const rules = schema[field];
```

```
        const value = data[field];

        if (rules.required && (!value || value.trim() === '')) {
            errors[field] = `${field} is required`;
        } else if (rules.minLength && value.length < rules.minLength) {
            errors[field] = `${field} must be at least ${rules.minLength} characters`;
        } else if (rules.pattern && !rules.pattern.test(value)) {
            errors[field] = `${field} format is invalid`;
        }
    });

    return errors;
};
```

3. API Security

```
// Request Signing
export const signRequest = async (requestData) => {
    const timestamp = Date.now().toString();
    const nonce = Math.random().toString(36).substring(7);

    const signature = await generateSignature({
        ...requestData,
        timestamp,
        nonce,
    });

    return {
        ...requestData,
        timestamp,
        nonce,
        signature,
    };
};

// Certificate Pinning (Android)
export const setupCertificatePinning = () => {
    if (Platform.OS === 'android') {
        // Configure certificate pinning for Android
        // This would be configured in the network security config
    }
};
```

Deployment Architecture

Build Configuration

Android Build

```
// android/app/build.gradle
android {
    compileSdkVersion 34
```

```
    buildToolsVersion "34.0.0"

    defaultConfig {
        applicationId "com.asmc.mobile"
        minSdkVersion 21
        targetSdkVersion 34
        versionCode 1
        versionName "1.0.0"

        // Enable multidex for large apps
        multiDexEnabled true
    }

    buildTypes {
        debug {
            debuggable true
            minifyEnabled false
            shrinkResources false
            buildConfigField "String", "API_BASE_URL", '"http://localhost:7055/api"'
        }

        release {
            debuggable false
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
            buildConfigField "String", "API_BASE_URL", '"https://api.asmc.com/api"'

            // Signing configuration
            signingConfig signingConfigs.release
        }
    }

    // Bundle configuration
    bundle {
        language {
            enableSplit = true
        }
        density {
            enableSplit = true
        }
        abi {
            enableSplit = true
        }
    }
}
```

CI/CD Pipeline

```
# .github/workflows/mobile-build.yml
name: Mobile App Build
```

```yaml
on:
    push:
        branches: [main, develop]
    pull_request:
        branches: [main]

jobs:
    test:
        runs-on: ubuntu-latest

        steps:
            - uses: actions/checkout@v3

            - name: Setup Node.js
              uses: actions/setup-node@v3
              with:
                  node-version: '18'
                  cache: 'npm'

            - name: Setup Java
              uses: actions/setup-java@v3
              with:
                  java-version: '11'
                  distribution: 'temurin'

            - name: Install dependencies
              run: npm ci

            - name: Run tests
              run: npm test

            - name: Run linting
              run: npm run lint

    build-android:
        needs: test
        runs-on: ubuntu-latest

        steps:
            - uses: actions/checkout@v3

            - name: Setup Node.js
              uses: actions/setup-node@v3
              with:
                  node-version: '18'
                  cache: 'npm'

            - name: Setup Java
              uses: actions/setup-java@v3
              with:
                  java-version: '11'
```

```
                distribution: 'temurin'

            - name: Setup Android SDK
              uses: android-actions/setup-android@v2

            - name: Install dependencies
              run: npm ci

            - name: Build Android APK
              run: |
                  cd android
                  ./gradlew assembleRelease

            - name: Upload APK
              uses: actions/upload-artifact@v3
              with:
                  name: android-apk
                  path: android/app/build/outputs/apk/release/app-release.apk
```

Summary

The ASMC Mobile App architecture provides:

- **Scalable Architecture**: Container-component pattern with clear separation of concerns
- **Efficient State Management**: React Query for server state, Context for global state
- **Robust API Integration**: Axios with interceptors, error handling, and token management
- **Secure Authentication**: JWT tokens with refresh mechanism and secure storage
- **Performance Optimization**: Image optimization, memory management, and network optimization
- **Production-Ready**: Comprehensive build configuration and CI/CD pipeline

This architecture ensures maintainable, performant, and secure mobile application development for the ASMC system.

Last Updated: January 2025
Maintainer: ASMC Development Team