ASMC Admin Panel - API Integration Documentation

This document provides comprehensive documentation for API integration in the ASMC Admin Panel, including RTK Query setup, authentication, error handling, and data flow patterns.

Table of Contents

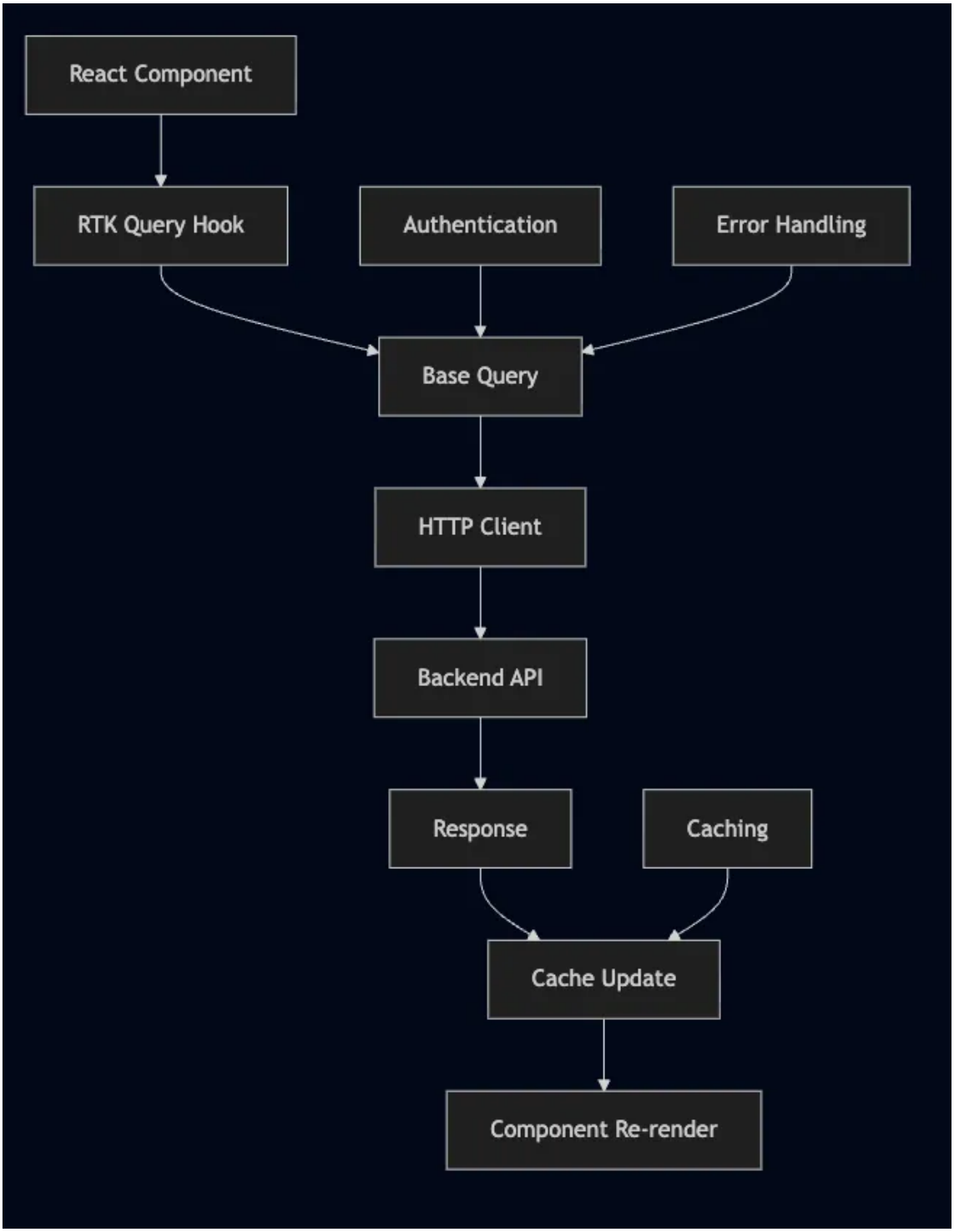
API Integration Overview

Architecture

The admin panel uses RTK Query for API integration, providing:

- **Automatic Caching**: Intelligent caching of API responses
- **Background Refetching**: Automatic data synchronization
- **Optimistic Updates**: Immediate UI updates with rollback
- **Request Deduplication**: Prevents duplicate requests
- **Error Handling**: Centralized error management
- **Type Safety**: Full TypeScript support

Integration Flow



RTK Query Configuration

Base API Configuration

Location: `/src/store/api/baseApi.js`

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

const baseQuery = fetchBaseQuery({
    baseUrl: process.env.REACT_APP_API_BASE_URL || '/api',
    prepareHeaders: (headers, { getState }) => {
        // Get token from Redux state
        const token = getState().auth?.token;
        if (token) {
            headers.set('authorization', `Bearer ${token}`);
        }

        // Set content type
        headers.set('content-type', 'application/json');

        // Add CSRF token if available
        const csrfToken = document.querySelector('meta[name="csrf-token"]');
        if (csrfToken) {
            headers.set('X-CSRF-Token', csrfToken.getAttribute('content'));
        }

        return headers;
    },
});

// Enhanced base query with error handling
const baseQueryWithReauth = async (args, api, extraOptions) => {
    let result = await baseQuery(args, api, extraOptions);

    // Handle 401 errors (unauthorized)
    if (result.error && result.error.status === 401) {
        // Try to refresh token
        const refreshResult = await baseQuery(
            {
                url: '/auth/refresh',
                method: 'POST',
                body: { refreshToken: getState().auth?.refreshToken },
            },
            api,
            extraOptions,
        );

        if (refreshResult.data) {
            // Update tokens in state
            api.dispatch(setAuthTokens(refreshResult.data));

            // Retry original request
            result = await baseQuery(args, api, extraOptions);
        } else {
            // Refresh failed, logout user
            api.dispatch(logout());
        }
    }
```

```
    return result;
};

export const baseApi = createApi({
    reducerPath: 'baseApi',
    baseQuery: baseQueryWithReauth,
    tagTypes: [
        'Members',
        'Bookings',
        'Events',
        'Facilities',
        'Staff',
        'Categories',
        'Locations',
        'Plans',
        'Activities',
        'Halls',
        'Reports',
        'Documentation',
    ],
    endpoints: () => ({}),
});
```

API Service Configuration

Location: `/src/helpers/axios.js`

```
import axios from 'axios';
import { store } from '../store';

// Create axios instance
const apiClient = axios.create({
    baseURL: process.env.REACT_APP_API_BASE_URL || '/api',
    timeout: parseInt(process.env.REACT_APP_API_TIMEOUT) || 30000,
    headers: {
        'Content-Type': 'application/json',
    },
});

// Request interceptor
apiClient.interceptors.request.use(
    (config) => {
        const state = store.getState();
        const token = state.auth?.token;

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

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

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

// Response interceptor
apiClient.interceptors.response.use(
    (response) => {
        // Calculate 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
        if (error.response?.status === 401 && !originalRequest._retry) {
            originalRequest._retry = true;

            try {
                const state = store.getState();
                const refreshToken = state.auth?.refreshToken;

                if (refreshToken) {
                    const response = await axios.post('/api/auth/refresh', {
                        refreshToken,
                    });

                    const { token, refreshToken: newRefreshToken } = response.data;
                    store.dispatch(
                        setAuthTokens({ token, refreshToken: newRefreshToken }),
                    );

                    // Retry original request
                    originalRequest.headers.Authorization = `Bearer ${token}`;
                    return apiClient(originalRequest);
                }
            } catch (refreshError) {
                // Refresh failed, logout user
                store.dispatch(logout());
                window.location.href = '/login';
            }
        }
```

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

export default apiClient;
```

Authentication Integration

Authentication API

Location: `/src/store/auth/authApi.js`

```
import { baseApi } from '../api/baseApi';

export const authApi = baseApi.injectEndpoints({
    endpoints: (builder) => ({
        login: builder.mutation({
            query: (credentials) => ({
                url: '/auth/login',
                method: 'POST',
                body: credentials,
            }),
            invalidatesTags: ['User'],
        }),

        logout: builder.mutation({
            query: () => ({
                url: '/auth/logout',
                method: 'POST',
            }),
            invalidatesTags: ['User'],
        }),

        refreshToken: builder.mutation({
            query: (refreshToken) => ({
                url: '/auth/refresh',
                method: 'POST',
                body: { refreshToken },
            }),
        }),

        getProfile: builder.query({
            query: () => '/auth/profile',
            providesTags: ['User'],
        }),

        updateProfile: builder.mutation({
            query: (profileData) => ({
                url: '/auth/profile',
                method: 'PUT',
                body: profileData,
            }),
```

```
            invalidatesTags: ['User'],
        }),

        changePassword: builder.mutation({
            query: (passwordData) => ({
                url: '/auth/change-password',
                method: 'POST',
                body: passwordData,
            }),
        }),
    }),
});

export const {
    useLoginMutation,
    useLogoutMutation,
    useRefreshTokenMutation,
    useGetProfileQuery,
    useUpdateProfileMutation,
    useChangePasswordMutation,
} = authApi;
```

Authentication Hook

Location: `/src/hooks/useAuth.js`

```
import { useDispatch, useSelector } from 'react-redux';
import { useNavigate } from 'react-router-dom';
import {
    useLoginMutation,
    useLogoutMutation,
    useGetProfileQuery,
} from '../store/auth/authApi';
import { setAuthUser, clearAuth } from '../store/auth/authSlice';

export const useAuth = () => {
    const dispatch = useDispatch();
    const navigate = useNavigate();

    const { user, token, isAuthenticated } = useSelector((state) => state.auth);

    const [loginMutation, { isLoading: isLoggingIn }] = useLoginMutation();
    const [logoutMutation, { isLoading: isLoggingOut }] = useLogoutMutation();
    const { data: profile, isLoading: isLoadingProfile } =
useGetProfileQuery(undefined, {
        skip: !isAuthenticated,
    });

    const login = async (credentials) => {
        try {
            const response = await loginMutation(credentials).unwrap();
            const { token, refreshToken, user } = response.data;
```

```
            // Store tokens
            localStorage.setItem('authToken', token);
            localStorage.setItem('refreshToken', refreshToken);

            // Update Redux state
            dispatch(setAuthUser({ user, token, refreshToken }));

            return { success: true, user };
        } catch (error) {
            return { success: false, error: error.data?.message || 'Login failed' };
        }
    };

    const logout = async () => {
        try {
            await logoutMutation().unwrap();
        } catch (error) {
            console.error('Logout error:', error);
        } finally {
            // Clear local storage
            localStorage.removeItem('authToken');
            localStorage.removeItem('refreshToken');

            // Clear Redux state
            dispatch(clearAuth());

            // Navigate to login
            navigate('/login');
        }
    };

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

    const hasRole = (role) => {
        if (!user || !user.roles) return false;
        return user.roles.includes(role);
    };

    return {
        user,
        token,
        isAuthenticated,
        isLoggingIn,
        isLoggingOut,
        isLoadingProfile,
        login,
        logout,
        hasPermission,
        hasRole,
```

```
    };
};
```

API Endpoints

Members API

Location: `/src/store/members/membersApi.js`

```
import { baseApi } from '../api/baseApi';

export const membersApi = baseApi.injectEndpoints({
    endpoints: (builder) => ({
        getMembers: builder.query({
            query: (params = {}) => ({
                url: '/members',
                params: {
                    page: params.page || 1,
                    limit: params.limit || 10,
                    search: params.search,
                    status: params.status,
                    category: params.category,
                    sortBy: params.sortBy,
                    sortOrder: params.sortOrder,
                },
            }),
            providesTags: (result) =>
                result
                    ? [
                          ...result.data.map(({ id }) => ({ type: 'Members', id })),
                          { type: 'Members', id: 'LIST' },
                      ]
                    : [{ type: 'Members', id: 'LIST' }],
        }),

        getMember: builder.query({
            query: (id) => `/members/${id}`,
            providesTags: (result, error, id) => [{ type: 'Members', id }],
        }),

        createMember: builder.mutation({
            query: (memberData) => ({
                url: '/members',
                method: 'POST',
                body: memberData,
            }),
            invalidatesTags: [{ type: 'Members', id: 'LIST' }],
        }),

        updateMember: builder.mutation({
            query: ({ id, ...memberData }) => ({
                url: `/members/${id}`,
```

```
                method: 'PUT',
                body: memberData,
            }),
            invalidatesTags: (result, error, { id }) => [
                { type: 'Members', id },
                { type: 'Members', id: 'LIST' },
            ],
        }),

        deleteMember: builder.mutation({
            query: (id) => ({
                url: `/members/${id}`,
                method: 'DELETE',
            }),
            invalidatesTags: (result, error, id) => [
                { type: 'Members', id },
                { type: 'Members', id: 'LIST' },
            ],
        }),

        getMemberHistory: builder.query({
            query: (id) => `/members/${id}/history`,
            providesTags: (result, error, id) => [{ type: 'Members', id, history: true
}],
        }),

        exportMembers: builder.query({
            query: (params) => ({
                url: '/members/export',
                params,
                responseHandler: (response) => response.blob(),
            }),
        }),

        importMembers: builder.mutation({
            query: (formData) => ({
                url: '/members/import',
                method: 'POST',
                body: formData,
            }),
            invalidatesTags: [{ type: 'Members', id: 'LIST' }],
        }),
    }),
});

export const {
    useGetMembersQuery,
    useGetMemberQuery,
    useCreateMemberMutation,
    useUpdateMemberMutation,
    useDeleteMemberMutation,
    useGetMemberHistoryQuery,
```

```
    useExportMembersQuery,
    useImportMembersMutation,
} = membersApi;
```

Bookings API

Location: `/src/store/booking/bookingApi.js`

```
import { baseApi } from '../api/baseApi';

export const bookingApi = baseApi.injectEndpoints({
    endpoints: (builder) => ({
        getBookings: builder.query({
            query: (params = {}) => ({
                url: '/booking',
                params: {
                    page: params.page || 1,
                    limit: params.limit || 10,
                    status: params.status,
                    facility: params.facility,
                    dateFrom: params.dateFrom,
                    dateTo: params.dateTo,
                    member: params.member,
                },
            }),
            providesTags: (result) =>
                result
                    ? [
                          ...result.data.map(({ id }) => ({ type: 'Bookings', id })),
                          { type: 'Bookings', id: 'LIST' },
                      ]
                    : [{ type: 'Bookings', id: 'LIST' }],
        }),

        getBooking: builder.query({
            query: (id) => `/booking/${id}`,
            providesTags: (result, error, id) => [{ type: 'Bookings', id }],
        }),

        createBooking: builder.mutation({
            query: (bookingData) => ({
                url: '/booking',
                method: 'POST',
                body: bookingData,
            }),
            invalidatesTags: [
                { type: 'Bookings', id: 'LIST' },
                { type: 'Facilities', id: 'LIST' },
            ],
        }),

        updateBooking: builder.mutation({
            query: ({ id, ...bookingData }) => ({
```

```
                url: `/booking/${id}`,
                method: 'PUT',
                body: bookingData,
            }),
            invalidatesTags: (result, error, { id }) => [
                { type: 'Bookings', id },
                { type: 'Bookings', id: 'LIST' },
                { type: 'Facilities', id: 'LIST' },
            ],
        }),

        cancelBooking: builder.mutation({
            query: (id) => ({
                url: `/booking/${id}/cancel`,
                method: 'POST',
            }),
            invalidatesTags: (result, error, id) => [
                { type: 'Bookings', id },
                { type: 'Bookings', id: 'LIST' },
            ],
        }),

        approveBooking: builder.mutation({
            query: (id) => ({
                url: `/booking/${id}/approve`,
                method: 'POST',
            }),
            invalidatesTags: (result, error, id) => [
                { type: 'Bookings', id },
                { type: 'Bookings', id: 'LIST' },
            ],
        }),

        getBookingCalendar: builder.query({
            query: (params) => ({
                url: '/booking/calendar',
                params: {
                    facility: params.facility,
                    date: params.date,
                    view: params.view, // 'day', 'week', 'month'
                },
            }),
            providesTags: ['Bookings'],
        }),
    }),
});

export const {
    useGetBookingsQuery,
    useGetBookingQuery,
    useCreateBookingMutation,
    useUpdateBookingMutation,
```

```
    useCancelBookingMutation,
    useApproveBookingMutation,
    useGetBookingCalendarQuery,
} = bookingApi;
```

Data Flow Patterns

Component Integration Pattern

```
import React, { useEffect, useState } from 'react';
import {
    useGetMembersQuery,
    useCreateMemberMutation,
    useUpdateMemberMutation,
    useDeleteMemberMutation,
} from '../store/members/membersApi';

const MembersManager = () => {
    const [filters, setFilters] = useState({
        search: '',
        status: 'all',
        page: 1,
        limit: 10,
    });

    // RTK Query hooks
    const {
        data: membersData,
        isLoading,
        isError,
        error,
        refetch,
    } = useGetMembersQuery(filters);

    const [createMember, { isLoading: isCreating }] = useCreateMemberMutation();
    const [updateMember, { isLoading: isUpdating }] = useUpdateMemberMutation();
    const [deleteMember, { isLoading: isDeleting }] = useDeleteMemberMutation();

    // Derived data
    const members = membersData?.data || [];
    const pagination = membersData?.pagination || {};

    // Event handlers
    const handleCreateMember = async (memberData) => {
        try {
            await createMember(memberData).unwrap();
            // Success notification will be handled by global error middleware
        } catch (error) {
            console.error('Failed to create member:', error);
        }
    };
```

```
    const handleUpdateMember = async ({ id, ...memberData }) => {
        try {
            await updateMember({ id, ...memberData }).unwrap();
        } catch (error) {
            console.error('Failed to update member:', error);
        }
    };

    const handleDeleteMember = async (id) => {
        try {
            await deleteMember(id).unwrap();
        } catch (error) {
            console.error('Failed to delete member:', error);
        }
    };

    const handleFilterChange = (newFilters) => {
        setFilters((prev) => ({ ...prev, ...newFilters }));
    };

    if (isError) {
        return <div>Error: {error?.data?.message || 'Failed to load members'}</div>;
    }

    return (
        <div>
            <MembersFilters filters={filters} onFilterChange={handleFilterChange} />

            <MembersTable
                members={members}
                loading={isLoading}
                onEdit={handleUpdateMember}
                onDelete={handleDeleteMember}
                pagination={pagination}
            />

            <CreateMemberModal onSubmit={handleCreateMember} loading={isCreating} />
        </div>
    );
};
```

Optimistic Updates

```
import { membersApi } from '../store/members/membersApi';

// Optimistic update example
const updateMemberOptimistic = membersApi.useUpdateMemberMutation({
    // Optimistic update
    onQueryStarted: async ({ id, ...patch }, { dispatch, queryFulfilled }) => {
        // Optimistic update
        const patchResult = dispatch(
            membersApi.util.updateQueryData('getMembers', undefined, (draft) => {
```

```
                const member = draft.data.find((m) => m.id === id);
                if (member) {
                    Object.assign(member, patch);
                }
            }),
        );

        try {
            await queryFulfilled;
        } catch {
            // Rollback on error
            patchResult.undo();
        }
    },
});
```

Error Handling

Global Error Middleware

Location: `/src/store/middleware/errorMiddleware.js`

```
import { createListenerMiddleware } from '@reduxjs/toolkit';
import { setNotification } from '../common/commonSlice';

export const errorMiddleware = createListenerMiddleware();

// Listen for rejected async actions
errorMiddleware.startListening({
    predicate: (action) => action.type.endsWith('/rejected'),
    effect: async (action, listenerApi) => {
        const { dispatch } = listenerApi;

        // Extract error information
        const error = action.payload;
        const errorMessage =
            error?.data?.message || error?.message || 'An error occurred';
        const statusCode = error?.status;

        // Handle different error types
        switch (statusCode) {
            case 400:
                dispatch(
                    setNotification({
                        message: errorMessage,
                        severity: 'warning',
                    }),
                );
                break;

            case 401:
                dispatch(
```

```
                setNotification({
                    message: 'Session expired. Please login again.',
                    severity: 'error',
                }),
            );
            // Logout will be handled by auth interceptor
            break;

        case 403:
            dispatch(
                setNotification({
                    message: 'Access denied. You do not have permission.',
                    severity: 'error',
                }),
            );
            break;

        case 404:
            dispatch(
                setNotification({
                    message: 'Resource not found.',
                    severity: 'warning',
                }),
            );
            break;

        case 422:
            // Validation errors
            if (error.data?.errors) {
                dispatch(
                    setNotification({
                        message: 'Please check your input and try again.',
                        severity: 'warning',
                    }),
                );
            } else {
                dispatch(
                    setNotification({
                        message: errorMessage,
                        severity: 'warning',
                    }),
                );
            }
            break;

        case 500:
            dispatch(
                setNotification({
                    message: 'Server error. Please try again later.',
                    severity: 'error',
                }),
            );
```

```
                break;

            default:
                dispatch(
                    setNotification({
                        message: errorMessage,
                        severity: 'error',
                    }),
                );
        }

        // Log error for debugging
        console.error('API Error:', {
            action: action.type,
            error: error,
            timestamp: new Date().toISOString(),
        });
    },
});
```

Component-Level Error Handling

```
import React from 'react';
import { Alert, Button, Box } from '@mui/material';
import { useGetMembersQuery } from '../store/members/membersApi';

const MembersList = () => {
    const { data: members, isLoading, isError, error, refetch } =
useGetMembersQuery();

    if (isError) {
        return (
            <Box>
                <Alert
                    severity="error"
                    action={
                        <Button color="inherit" size="small" onClick={() =>
refetch()}>
                            Retry
                        </Button>
                    }
                >
                    Failed to load members: {error?.data?.message || error?.message}
                </Alert>
            </Box>
        );
    }

    if (isLoading) {
        return <div>Loading members...</div>;
    }
```

```
    return (
        <div>
            {members?.data?.map((member) => (
                <div key={member.id}>{member.name}</div>
            ))}
        </div>
    );
};
```

Caching Strategies

Cache Configuration

```
export const membersApi = createApi({
    // ... other config
    endpoints: (builder) => ({
        getMembers: builder.query({
            query: (params) => ({
                url: '/members',
                params,
            }),
            // Cache for 5 minutes
            keepUnusedDataFor: 300,
            // Refetch on window focus
            refetchOnFocus: true,
            // Refetch on reconnect
            refetchOnReconnect: true,
            providesTags: ['Members'],
        }),
    }),
});
```

Manual Cache Management

```
import { useDispatch } from 'react-redux';
import { membersApi } from '../store/members/membersApi';

const useCacheManagement = () => {
    const dispatch = useDispatch();

    const invalidateMembersCache = () => {
        dispatch(membersApi.util.invalidateTags(['Members']));
    };

    const updateMemberInCache = (memberId, updates) => {
        dispatch(
            membersApi.util.updateQueryData('getMembers', undefined, (draft) => {
                const member = draft.data.find((m) => m.id === memberId);
                if (member) {
                    Object.assign(member, updates);
                }
```

```
            }),
        );
    };

    const removeMemberFromCache = (memberId) => {
        dispatch(
            membersApi.util.updateQueryData('getMembers', undefined, (draft) => {
                draft.data = draft.data.filter((m) => m.id !== memberId);
            }),
        );
    };

    return {
        invalidateMembersCache,
        updateMemberInCache,
        removeMemberFromCache,
    };
};
```

Performance Optimization

Request Deduplication

RTK Query automatically deduplicates identical requests:

```
// Multiple components can call this simultaneously
// Only one request will be made
const { data: members } = useGetMembersQuery({ page: 1, limit: 10 });
const { data: members2 } = useGetMembersQuery({ page: 1, limit: 10 }); // Same request
```

Conditional Queries

```
const MembersComponent = ({ shouldFetch, memberId }) => {
    // Only fetch when conditions are met
    const { data: members } = useGetMembersQuery(undefined, {
        skip: !shouldFetch,
    });

    const { data: member } = useGetMemberQuery(memberId, {
        skip: !memberId,
    });

    return <div>{/* Component JSX */}</div>;
};
```

Polling and Real-time Updates

```
// Poll every 30 seconds
const { data: members } = useGetMembersQuery(undefined, {
    pollingInterval: 30000,
});
```

```
// Poll with conditional logic
const { data: members } = useGetMembersQuery(undefined, {
    pollingInterval: (data, query) => {
        // Stop polling if no data or error
        return data?.length === 0 ? 30000 : 0;
    },
});
```

Background Refetching

```
const { data: members } = useGetMembersQuery(undefined, {
    // Refetch when window regains focus
    refetchOnFocus: true,
    // Refetch when network reconnects
    refetchOnReconnect: true,
    // Refetch when component mounts
    refetchOnMountOrArgChange: true,
});
```

Summary

The ASMC Admin Panel's API integration provides:

- **Seamless Integration**: RTK Query for efficient API management
- **Authentication**: JWT-based auth with automatic token refresh
- **Error Handling**: Comprehensive error management and user feedback
- **Caching**: Intelligent caching with background updates
- **Performance**: Request deduplication and optimistic updates
- **Type Safety**: Full TypeScript support for API responses

This architecture ensures reliable, performant, and maintainable API integration for the admin panel.

Version: 1.0.0
Last Updated: January 2025
Maintainer: ASMC Development Team