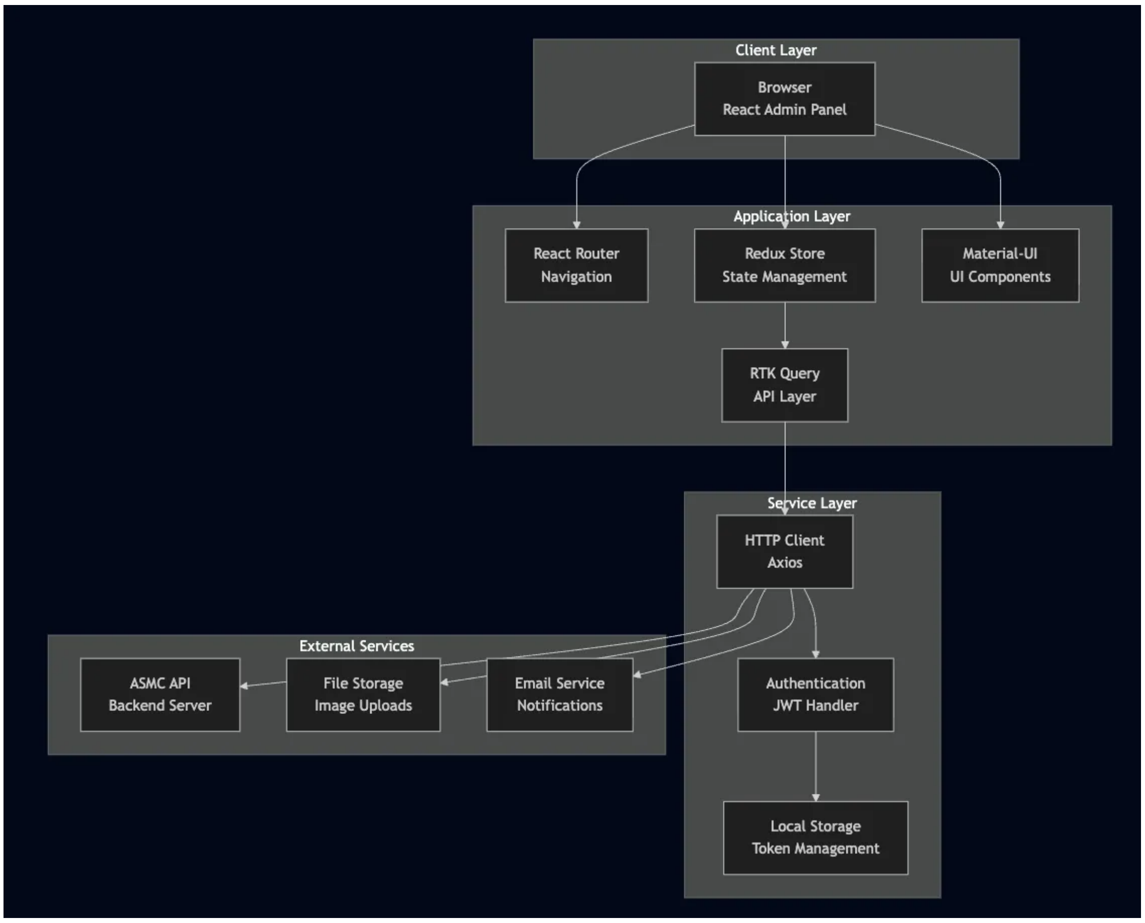
ASMC Admin Panel - Architecture Overview

This document provides a comprehensive overview of the ASMC Admin Panel architecture, design patterns, and technical implementation details.

Table of Contents

System Architecture

High-Level Architecture



Client Layer
Browser React Admin Panel
Application Layer
React Router Navigation
Redux Store State Management
Material-UI UI Components
RTK Query API Layer
Service Layer
HTTP Client Axios
Authentication JWT Handler
Local Storage Token Management
External Services
ASMC API Backend Server
File Storage Image Uploads
Email Service Notifications

Technology Stack

Layer	Technology	Purpose
Frontend Framework	React.js 18.2.0	Component-based UI
State Management	Redux Toolkit + RTK Query	Global state and API calls
UI Framework	Material-UI 5.13.7	Design system and components
Routing	React Router DOM 6.11.2	Client-side routing
HTTP Client	Axios 1.4.0	API communication
Form Management	Formik + Yup	Form handling and validation
Charts	Chart.js + React-Chartjs-2	Data visualization
Rich Text	CKEditor 5	Content editing
Build Tool	Create React App	Development and build

Component Architecture

Component Hierarchy

```
App
├── Router
│   ├── PublicRoutes
│   │   └── LoginPage
│   └── AdminRoutes
│       └── LayoutContainer
│           ├── Header
│           ├── Sidebar
│           │   └── MenuList
│           └── MainContent
│               ├── DashboardPage
│               ├── MembersManager
│               ├── BookingManager
│               ├── EventsManager
│               ├── FacilityManager
│               ├── StaffManager
│               ├── PaymentManager
│               └── DocumentationManager
```

Component Patterns

1. Container-Component Pattern

```
// Container (Business Logic)
const MembersContainer = connect(mapStateToProps, mapDispatchToProps)
(MembersComponent);

// Component (Presentation)
const MembersComponent = ({ members, loading, onEdit, onDelete }) => {
    return (
        <Box>
            <MembersTable
                data={members}
                onEdit={onEdit}
                onDelete={onDelete}
                loading={loading}
            />
        </Box>
    );
};
```

2. Higher-Order Components (HOCs)

```
// withNavigate HOC
const withNavigate = (WrappedComponent) => {
    return (props) => {
        const navigate = useNavigate();
        return <WrappedComponent {...props} navigate={navigate} />;
    };
};

// withPermission HOC
const withPermission = (permission) => (WrappedComponent) => {
    return (props) => {
        const hasPermission = usePermission(permission);
        return hasPermission ? <WrappedComponent {...props} /> : <AccessDenied />;
    };
};
```

3. Custom Hooks Pattern

```
// Custom hook for member management
const useMembers = () => {
    const dispatch = useDispatch();
    const members = useSelector(selectMembers);
    const loading = useSelector(selectMembersLoading);

    const fetchMembers = useCallback(
        (params) => {
            dispatch(fetchMembersAsync(params));
        },
        [dispatch],
    );

    const createMember = useCallback(
        (memberData) => {
            dispatch(createMemberAsync(memberData));
        },
        [dispatch],
    );

    return {
        members,
        loading,
        fetchMembers,
        createMember,
    };
};
```

State Management Architecture

Redux Store Structure

```
// Store configuration
export const store = configureStore({
    reducer: {
        // Feature-based slices
        common: commonSlice, // Global app state
        members: membersSlice, // Member management
        booking: bookingSlice, // Booking management
        events: eventsSlice, // Event management
        facility: facilitySlice, // Facility management
        staff: staffSlice, // Staff management
        documentation: documentationSlice, // Documentation

        // RTK Query APIs
        membersApi: membersApis, // Members API
        commonApi: commonApis, // Common API
        bookingApi: bookingApis, // Booking API
        eventsApi: eventsApis, // Events API
        facilityApi: facilityApis, // Facility API
        staffApi: staffApis, // Staff API
        documentationApi: documentationApi, // Documentation API
    },
    middleware: (getDefaultMiddleware) =>
        getDefaultMiddleware({
            serializableCheck: {
                ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
            },
        }).concat([
            membersApis.middleware,
            commonApis.middleware,
            bookingApis.middleware,
            eventsApis.middleware,
            facilityApis.middleware,
            staffApis.middleware,
            documentationApi.middleware,
        ]),
});
```

State Slice Architecture

```
// Example: Members slice
const membersSlice = createSlice({
    name: 'members',
    initialState: {
        list: [],
        selectedMember: null,
        filters: {
            status: 'all',
            search: '',
            page: 1,
            limit: 10,
        },
        loading: false,
        error: null,
    },
    reducers: {
        setMembers: (state, action) => {
            state.list = action.payload;
        },
        setSelectedMember: (state, action) => {
            state.selectedMember = action.payload;
        },
        updateFilters: (state, action) => {
            state.filters = { ...state.filters, ...action.payload };
        },
        setLoading: (state, action) => {
            state.loading = action.payload;
        },
        setError: (state, action) => {
            state.error = action.payload;
        },
    },
    extraReducers: (builder) => {
        builder
            .addCase(fetchMembersAsync.pending, (state) => {
                state.loading = true;
                state.error = null;
            })
            .addCase(fetchMembersAsync.fulfilled, (state, action) => {
                state.loading = false;
                state.list = action.payload.data;
            })
            .addCase(fetchMembersAsync.rejected, (state, action) => {
                state.loading = false;
                state.error = action.error.message;
            });
    },
});
```

API Integration Architecture

RTK Query Setup

```
// Base API configuration
const baseApi = createApi({
    reducerPath: 'baseApi',
    baseQuery: fetchBaseQuery({
        baseUrl: process.env.REACT_APP_API_BASE_URL,
        prepareHeaders: (headers, { getState }) => {
            const token = selectAuthToken(getState());
            if (token) {
                headers.set('authorization', `Bearer ${token}`);
            }
            headers.set('content-type', 'application/json');
            return headers;
        },
    }),
    tagTypes: ['Members', 'Bookings', 'Events', 'Facilities', 'Staff'],
    endpoints: () => ({}),
});
```

Feature-Specific APIs

```
// Members API
export const membersApi = baseApi.injectEndpoints({
    endpoints: (builder) => ({
        getMembers: builder.query({
            query: (params) => ({
                url: '/members',
                params: {
                    page: params.page,
                    limit: params.limit,
                    search: params.search,
                    status: params.status,
                },
            }),
            providesTags: ['Members'],
        }),
        getMember: builder.query({
            query: (id) => `/members/${id}`,
            providesTags: (result, error, id) => [{ type: 'Members', id }],
        }),
        createMember: builder.mutation({
            query: (memberData) => ({
                url: '/members',
                method: 'POST',
                body: memberData,
            }),
            invalidatesTags: ['Members'],
        }),
        updateMember: builder.mutation({
            query: ({ id, ...memberData }) => ({
                url: `/members/${id}`,
                method: 'PUT',
                body: memberData,
            }),
            invalidatesTags: (result, error, { id }) => [
                { type: 'Members', id },
                'Members',
            ],
        }),
        deleteMember: builder.mutation({
            query: (id) => ({
                url: `/members/${id}`,
                method: 'DELETE',
            }),
            invalidatesTags: ['Members'],
        }),
    }),
});
```

API Error Handling

```
// Global error handling
const apiErrorHandler = (error, { dispatch, getState }) => {
    if (error.status === 401) {
        // Token expired - redirect to login
        dispatch(logout());
        window.location.href = '/login';
    } else if (error.status === 403) {
        // Access denied
        dispatch(
            setNotification({
                type: 'error',
                message:
                    'Access denied. You do not have permission to perform this
action.',
            }),
        );
    } else if (error.status >= 500) {
        // Server error
        dispatch(
            setNotification({
                type: 'error',
                message: 'Server error. Please try again later.',
            }),
        );
    }
};
```

Authentication Architecture

JWT Token Management

```
    }
}
```

```javascript
// Authentication service
class AuthService {
    static getToken() {
        return localStorage.getItem('authToken');
    }

    static setToken(token) {
        localStorage.setItem('authToken', token);
    }

    static removeToken() {
        localStorage.removeItem('authToken');
        localStorage.removeItem('refreshToken');
    }

    static getRefreshToken() {
        return localStorage.getItem('refreshToken');
    }

    static setRefreshToken(token) {
        localStorage.setItem('refreshToken', token);
    }

    static isTokenExpired(token) {
        try {
            const decoded = jwt.decode(token);
            return decoded.exp < Date.now() / 1000;
        } catch {
            return true;
        }
    }

    static async refreshToken() {
        const refreshToken = this.getRefreshToken();
        if (!refreshToken) return null;

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

            const { token, refreshToken: newRefreshToken } = response.data;
            this.setToken(token);
            this.setRefreshToken(newRefreshToken);

            return token;
        } catch (error) {
            this.removeToken();
            return null;
        }
    }
}
```

Permission-Based Access Control

```
// Permission system
const usePermission = (permission) => {
    const userPermissions = useSelector(selectUserPermissions);
    return userPermissions.includes(permission);
};

// Permission wrapper component
const RequirePermission = ({ permission, children, fallback }) => {
    const hasPermission = usePermission(permission);

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

    return children;
};

// Usage
<RequirePermission permission="members:write">
    <CreateMemberButton />
</RequirePermission>;
```

Routing Architecture

Route Configuration

```
// Admin routes
const AdminRoutes = () => {
    return (
        <Routes>
            <Route path="/" element={<LayoutContainer />}>
                <Route index element={<DashboardPage />} />

                {/* Member Management */}
                <Route path="members" element={<MembersManager />} />
                <Route path="members/create" element={<CreateMemberPage />} />
                <Route path="members/:id/edit" element={<EditMemberPage />} />

                {/* Booking Management */}
                <Route path="bookings" element={<BookingManager />} />
                <Route path="bookings/create" element={<CreateBookingPage />} />

                {/* Event Management */}
                <Route path="events" element={<EventsManager />} />
                <Route path="events/create" element={<CreateEventPage />} />

                {/* Facility Management */}
                <Route path="facilities" element={<FacilityManager />} />
                {/* Staff Management */}
                <Route path="staff" element={<StaffManager />} />

                {/* Documentation */}
                <Route path="documentation" element={<DocumentationManager />} />

                {/* Catch-all route */}
                <Route path="*" element={<NotFoundPage />} />
            </Route>
        </Routes>
    );
};
```

Protected Routes

```
// Route protection
const ProtectedRoute = ({ children, requiredPermission }) => {
    const isAuthenticated = useSelector(selectIsAuthenticated);
    const hasPermission = usePermission(requiredPermission);

    if (!isAuthenticated) {
        return <Navigate to="/login" replace />;
    }

    if (requiredPermission && !hasPermission) {
        return <AccessDenied />;
    }

    return children;
};
```

UI Architecture

Material-UI Theme System

```
// Theme configuration
const theme = createTheme({
    palette: {
        primary: {
            main: '#1976d2',
            light: '#42a5f5',
            dark: '#1565c0',
        },
        secondary: {
            main: '#dc004e',
            light: '#ff5983',
            dark: '#9a0036',
        },
        background: {
            default: '#f5f5f5',
            paper: '#ffffff',
        },
    },
    typography: {
        fontFamily: '"Roboto", "Helvetica", "Arial", sans-serif',
        h1: {
            fontSize: '2.5rem',
            fontWeight: 500,
        },
        h2: {
            fontSize: '2rem',
            fontWeight: 500,
        },
    },
    components: {
        MuiButton: {
            styleOverrides: {
                root: {
                    textTransform: 'none',
                    borderRadius: 8,
                },
            },
        },
        MuiCard: {
            styleOverrides: {
                root: {
                    borderRadius: 12,
                    boxShadow: '0 2px 8px rgba(0,0,0,0.1)',
                },
            },
        },
    },
});
```

Component Library Structure

```
components/
├── Common/                # Reusable components
│   ├── Input.jsx         # Form inputs
│   ├── Button.jsx        # Buttons
│   ├── Table.jsx         # Data tables
│   ├── Modal.jsx         # Modal dialogs
│   ├── Select.jsx        # Select dropdowns
│   └── DatePicker.jsx    # Date pickers
├── admin/                # Admin-specific components
│   ├── members-manager/  # Member management
│   ├── booking-manager/  # Booking management
│   ├── events-manager/   # Event management
│   └── ...
└── layout/               # Layout components
    ├── Header.jsx        # App header
    ├── Sidebar.jsx       # Navigation sidebar
    └── Footer.jsx        # App footer
```

Performance Architecture

Code Splitting

```
// Lazy loading for routes
const DashboardPage = lazy(() => import('../pages/admin/DashboardPage'));
const MembersManager = lazy(() => import('../pages/admin/MembersManager'));
const BookingManager = lazy(() => import('../pages/admin/BookingManager'));

// Suspense wrapper
const AppRoutes = () => (
    <Suspense fallback={<LoadingSpinner />}>
        <Routes>
            <Route path="/" element={<DashboardPage />} />
            <Route path="/members" element={<MembersManager />} />
            <Route path="/bookings" element={<BookingManager />} />
        </Routes>
    </Suspense>
);
```

Memoization Strategy

```
// Component memoization
const MembersTable = memo(({ members, onEdit, onDelete }) => {
    return (
        <Table>
            {members.map((member) => (
                <MemberRow
                    key={member.id}
                    member={member}
                    onEdit={onEdit}
                    onDelete={onDelete}
                />
            ))}
        </Table>
    );
});

// Callback memoization
const MembersContainer = () => {
    const dispatch = useDispatch();

    const handleEdit = useCallback(
        (memberId) => {
            dispatch(editMember(memberId));
        },
        [dispatch],
    );

    const handleDelete = useCallback(
        (memberId) => {
            dispatch(deleteMember(memberId));
        },
        [dispatch],
    );

    return <MembersTable onEdit={handleEdit} onDelete={handleDelete} />;
};
```

Bundle Optimization

```
// Webpack bundle analyzer configuration
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;

module.exports = {
    plugins: [
        new BundleAnalyzerPlugin({
            analyzerMode: 'static',
            openAnalyzer: false,
        }),
    ],
};
```

Security Architecture

Input Validation

```
// Form validation schema
const memberValidationSchema = yup.object({
    name: yup
        .string()
        .required('Name is required')
        .min(2, 'Name must be at least 2 characters')
        .max(50, 'Name must be less than 50 characters'),
    email: yup.string().email('Invalid email format').required('Email is required'),
    phone: yup
        .string()
        .matches(/^[0-9]{10}$/, 'Phone must be 10 digits')
        .required('Phone is required'),
});
```

XSS Protection

```
// Sanitize user input
const sanitizeInput = (input) => {
    return DOMPurify.sanitize(input, {
        ALLOWED_TAGS: [],
        ALLOWED_ATTR: [],
    });
};

// Safe HTML rendering
```

```
const SafeHTML = ({ content }) => {
    const sanitizedContent = useMemo(() => sanitizeInput(content), [content]);

    return (
        <div
            dangerouslySetInnerHTML={{
                __html: sanitizedContent,
            }}
        />
    );
};
```

CSRF Protection

```
// CSRF token handling
const getCSRFToken = () => {
    const token = document.querySelector('meta[name="csrf-token"]');
    return token ? token.getAttribute('content') : null;
};

// Axios interceptor for CSRF
axios.interceptors.request.use((config) => {
    const csrfToken = getCSRFToken();
    if (csrfToken) {
        config.headers['X-CSRF-Token'] = csrfToken;
    }
    return config;
});
```

Deployment Architecture

Build Process

```
# Development build
npm run start

# Production build
npm run build

# Build analysis
npm run analyze
```

Static File Serving

```
# Nginx configuration
server {
    listen 80;
    server_name admin.asmc.com;
    root /var/www/html/build;
    index index.html;
    # Handle client-side routing
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Cache static assets
    location /static/ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # API proxy
    location /api {
        proxy_pass http://localhost:7055;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Docker Configuration

```
# Multi-stage build
FROM node:18-alpine as build

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Summary

The ASMC Admin Panel follows modern React.js best practices with a well-structured architecture that includes:

- **Component-based architecture** with clear separation of concerns
- **Redux-based state management** with RTK Query for API integration
- **Material-UI design system** for consistent UI/UX
- **Permission-based access control** for security
- **Performance optimizations** including code splitting and memoization
- **Comprehensive error handling** and validation
- **Production-ready deployment** configuration

This architecture ensures scalability, maintainability, and excellent user experience for the administrative staff managing the ASMC system.

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