**InterEd Admin Portal State Management Structure**

**Overview**

This document outlines a comprehensive state management architecture for the InterEd Admin Portal based on Redux Toolkit. The architecture is designed to handle the complex state requirements of the application while ensuring maintainability, performance, and scalability. This state management structure can be implemented at approximately 85-90% completion before the Vibathon.

**State Management Architecture**

The InterEd Admin Portal will use a combination of:

* **Redux Toolkit** as the primary state management solution
* **React Query** for server state (API data fetching, caching, and synchronization)
* **Context API** for UI-specific state that doesn't need global persistence
* **Local component state** for form inputs and UI interactions

This layered approach provides optimal performance while maintaining a clear separation of concerns.

**Redux Store Structure**

**Store Configuration**

// src/store/index.js

import { configureStore } from '@reduxjs/toolkit';

import { setupListeners } from '@reduxjs/toolkit/query';

import storage from 'redux-persist/lib/storage';

import { persistReducer, persistStore } from 'redux-persist';

import { combineReducers } from 'redux';

// Import reducers

import authReducer from './slices/authSlice';

import uiReducer from './slices/uiSlice';

import studentReducer from './slices/studentSlice';

import agentReducer from './slices/agentSlice';

import universityReducer from './slices/universitySlice';

import programReducer from './slices/programSlice';

import applicationReducer from './slices/applicationSlice';

import documentReducer from './slices/documentSlice';

import commissionReducer from './slices/commissionSlice';

import analyticsReducer from './slices/analyticsSlice';

import notificationReducer from './slices/notificationSlice';

// Import API

import { api } from './api/apiSlice';

// Configure persist

const persistConfig = {

key: 'root',

storage,

whitelist: ['auth'], // Only persist authentication data

};

const rootReducer = combineReducers({

auth: authReducer,

ui: uiReducer,

students: studentReducer,

agents: agentReducer,

universities: universityReducer,

programs: programReducer,

applications: applicationReducer,

documents: documentReducer,

commissions: commissionReducer,

analytics: analyticsReducer,

notifications: notificationReducer,

[api.reducerPath]: api.reducer,

});

const persistedReducer = persistReducer(persistConfig, rootReducer);

export const store = configureStore({

reducer: persistedReducer,

middleware: (getDefaultMiddleware) =>

getDefaultMiddleware({

serializableCheck: {

ignoredActions: ['persist/PERSIST', 'persist/REHYDRATE'],

ignoredPaths: ['some.path.to.ignore'],

},

}).concat(api.middleware),

});

export const persistor = persistStore(store);

// Setup listeners for RTK Query

setupListeners(store.dispatch);

**Redux Slices**

The application state is divided into domain-specific slices:

**1. Authentication Slice**

// src/store/slices/authSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

user: null,

token: null,

refreshToken: null,

expiresAt: null,

isAuthenticated: false,

isLoading: false,

error: null,

};

const authSlice = createSlice({

name: 'auth',

initialState,

reducers: {

loginStart: (state) => {

state.isLoading = true;

state.error = null;

},

loginSuccess: (state, action) => {

state.isLoading = false;

state.isAuthenticated = true;

state.user = action.payload.user;

state.token = action.payload.token;

state.refreshToken = action.payload.refreshToken;

state.expiresAt = action.payload.expiresAt;

},

loginFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

logout: (state) => {

state.user = null;

state.token = null;

state.refreshToken = null;

state.expiresAt = null;

state.isAuthenticated = false;

},

refreshTokenSuccess: (state, action) => {

state.token = action.payload.token;

state.refreshToken = action.payload.refreshToken;

state.expiresAt = action.payload.expiresAt;

},

updateUserProfile: (state, action) => {

state.user = { ...state.user, ...action.payload };

},

},

});

export const {

loginStart,

loginSuccess,

loginFailure,

logout,

refreshTokenSuccess,

updateUserProfile,

} = authSlice.actions;

// Selectors

export const selectCurrentUser = (state) => state.auth.user;

export const selectIsAuthenticated = (state) => state.auth.isAuthenticated;

export const selectAuthToken = (state) => state.auth.token;

export const selectAuthError = (state) => state.auth.error;

export const selectIsAdmin = (state) => state.auth.user?.role === 'admin';

export const selectUserRole = (state) => state.auth.user?.role;

export const selectUserPermissions = (state) => state.auth.user?.permissions || [];

export default authSlice.reducer;

**2. UI Slice**

// src/store/slices/uiSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

sidebarOpen: true,

currentTheme: 'light',

toasts: [],

activeModal: null,

modalData: null,

breadcrumbs: [],

isMobileView: false,

};

const uiSlice = createSlice({

name: 'ui',

initialState,

reducers: {

toggleSidebar: (state) => {

state.sidebarOpen = !state.sidebarOpen;

},

setSidebarOpen: (state, action) => {

state.sidebarOpen = action.payload;

},

setTheme: (state, action) => {

state.currentTheme = action.payload;

},

addToast: (state, action) => {

state.toasts.push({

id: Date.now(),

...action.payload,

});

},

removeToast: (state, action) => {

state.toasts = state.toasts.filter((toast) => toast.id !== action.payload);

},

openModal: (state, action) => {

state.activeModal = action.payload.modal;

state.modalData = action.payload.data || null;

},

closeModal: (state) => {

state.activeModal = null;

state.modalData = null;

},

setBreadcrumbs: (state, action) => {

state.breadcrumbs = action.payload;

},

setMobileView: (state, action) => {

state.isMobileView = action.payload;

},

},

});

export const {

toggleSidebar,

setSidebarOpen,

setTheme,

addToast,

removeToast,

openModal,

closeModal,

setBreadcrumbs,

setMobileView,

} = uiSlice.actions;

// Selectors

export const selectSidebarOpen = (state) => state.ui.sidebarOpen;

export const selectCurrentTheme = (state) => state.ui.currentTheme;

export const selectToasts = (state) => state.ui.toasts;

export const selectActiveModal = (state) => state.ui.activeModal;

export const selectModalData = (state) => state.ui.modalData;

export const selectBreadcrumbs = (state) => state.ui.breadcrumbs;

export const selectIsMobileView = (state) => state.ui.isMobileView;

export default uiSlice.reducer;

**3. Student Slice**

// src/store/slices/studentSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

students: [],

selectedStudent: null,

isLoading: false,

error: null,

filters: {

status: null,

search: '',

agent: null,

sortBy: 'createdAt',

sortOrder: 'desc',

},

pagination: {

page: 1,

limit: 10,

totalPages: 0,

totalItems: 0,

},

};

const studentSlice = createSlice({

name: 'students',

initialState,

reducers: {

fetchStudentsStart: (state) => {

state.isLoading = true;

state.error = null;

},

fetchStudentsSuccess: (state, action) => {

state.isLoading = false;

state.students = action.payload.students;

state.pagination = action.payload.pagination;

},

fetchStudentsFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

setSelectedStudent: (state, action) => {

state.selectedStudent = action.payload;

},

clearSelectedStudent: (state) => {

state.selectedStudent = null;

},

updateFilters: (state, action) => {

state.filters = { ...state.filters, ...action.payload };

state.pagination.page = 1; // Reset to first page when filters change

},

setPage: (state, action) => {

state.pagination.page = action.payload;

},

setLimit: (state, action) => {

state.pagination.limit = action.payload;

state.pagination.page = 1; // Reset to first page when limit changes

},

addStudent: (state, action) => {

state.students.unshift(action.payload);

},

updateStudent: (state, action) => {

const index = state.students.findIndex(

(student) => student.id === action.payload.id

);

if (index !== -1) {

state.students[index] = {

...state.students[index],

...action.payload,

};

}

if (state.selectedStudent?.id === action.payload.id) {

state.selectedStudent = {

...state.selectedStudent,

...action.payload,

};

}

},

removeStudent: (state, action) => {

state.students = state.students.filter(

(student) => student.id !== action.payload

);

if (state.selectedStudent?.id === action.payload) {

state.selectedStudent = null;

}

},

},

});

export const {

fetchStudentsStart,

fetchStudentsSuccess,

fetchStudentsFailure,

setSelectedStudent,

clearSelectedStudent,

updateFilters,

setPage,

setLimit,

addStudent,

updateStudent,

removeStudent,

} = studentSlice.actions;

// Selectors

export const selectStudents = (state) => state.students.students;

export const selectSelectedStudent = (state) => state.students.selectedStudent;

export const selectStudentsLoading = (state) => state.students.isLoading;

export const selectStudentsError = (state) => state.students.error;

export const selectStudentFilters = (state) => state.students.filters;

export const selectStudentPagination = (state) => state.students.pagination;

export default studentSlice.reducer;

**4. Agent Slice**

Structure similar to the Student slice, but for agent-specific state.

**5. University Slice**

Structure similar to the Student slice, but for university-specific state.

**6. Program Slice**

Structure similar to the Student slice, but for program-specific state.

**7. Application Slice**

// src/store/slices/applicationSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

applications: [],

selectedApplication: null,

isLoading: false,

error: null,

filters: {

status: null,

student: null,

agent: null,

university: null,

program: null,

submittedDateStart: null,

submittedDateEnd: null,

search: '',

sortBy: 'submittedAt',

sortOrder: 'desc',

},

pagination: {

page: 1,

limit: 10,

totalPages: 0,

totalItems: 0,

},

currentStep: 1,

formData: {},

activeTab: 'details',

};

const applicationSlice = createSlice({

name: 'applications',

initialState,

reducers: {

fetchApplicationsStart: (state) => {

state.isLoading = true;

state.error = null;

},

fetchApplicationsSuccess: (state, action) => {

state.isLoading = false;

state.applications = action.payload.applications;

state.pagination = action.payload.pagination;

},

fetchApplicationsFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

setSelectedApplication: (state, action) => {

state.selectedApplication = action.payload;

},

clearSelectedApplication: (state) => {

state.selectedApplication = null;

},

updateFilters: (state, action) => {

state.filters = { ...state.filters, ...action.payload };

state.pagination.page = 1;

},

setPage: (state, action) => {

state.pagination.page = action.payload;

},

setLimit: (state, action) => {

state.pagination.limit = action.payload;

state.pagination.page = 1;

},

addApplication: (state, action) => {

state.applications.unshift(action.payload);

},

updateApplication: (state, action) => {

const index = state.applications.findIndex(

(application) => application.id === action.payload.id

);

if (index !== -1) {

state.applications[index] = {

...state.applications[index],

...action.payload,

};

}

if (state.selectedApplication?.id === action.payload.id) {

state.selectedApplication = {

...state.selectedApplication,

...action.payload,

};

}

},

removeApplication: (state, action) => {

state.applications = state.applications.filter(

(application) => application.id !== action.payload

);

if (state.selectedApplication?.id === action.payload) {

state.selectedApplication = null;

}

},

setCurrentStep: (state, action) => {

state.currentStep = action.payload;

},

updateFormData: (state, action) => {

state.formData = { ...state.formData, ...action.payload };

},

resetFormData: (state) => {

state.formData = {};

state.currentStep = 1;

},

setActiveTab: (state, action) => {

state.activeTab = action.payload;

},

addNote: (state, action) => {

if (state.selectedApplication) {

const notes = state.selectedApplication.notes || [];

state.selectedApplication.notes = [action.payload, ...notes];

}

},

},

});

export const {

fetchApplicationsStart,

fetchApplicationsSuccess,

fetchApplicationsFailure,

setSelectedApplication,

clearSelectedApplication,

updateFilters,

setPage,

setLimit,

addApplication,

updateApplication,

removeApplication,

setCurrentStep,

updateFormData,

resetFormData,

setActiveTab,

addNote,

} = applicationSlice.actions;

// Selectors

export const selectApplications = (state) => state.applications.applications;

export const selectSelectedApplication = (state) => state.applications.selectedApplication;

export const selectApplicationsLoading = (state) => state.applications.isLoading;

export const selectApplicationsError = (state) => state.applications.error;

export const selectApplicationFilters = (state) => state.applications.filters;

export const selectApplicationPagination = (state) => state.applications.pagination;

export const selectCurrentStep = (state) => state.applications.currentStep;

export const selectFormData = (state) => state.applications.formData;

export const selectActiveTab = (state) => state.applications.activeTab;

export default applicationSlice.reducer;

**8. Document Slice**

Structure similar to previous slices, but for document-specific state.

**9. Commission Slice**

Structure similar to previous slices, but for commission-specific state.

**10. Analytics Slice**

// src/store/slices/analyticsSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

dashboardStats: {

isLoading: false,

error: null,

data: null,

},

studentAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

},

},

applicationAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

groupBy: 'status',

},

},

agentAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

limit: 10,

},

},

revenueAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

},

},

};

const analyticsSlice = createSlice({

name: 'analytics',

initialState,

reducers: {

fetchDashboardStatsStart: (state) => {

state.dashboardStats.isLoading = true;

state.dashboardStats.error = null;

},

fetchDashboardStatsSuccess: (state, action) => {

state.dashboardStats.isLoading = false;

state.dashboardStats.data = action.payload;

},

fetchDashboardStatsFailure: (state, action) => {

state.dashboardStats.isLoading = false;

state.dashboardStats.error = action.payload;

},

fetchStudentAnalyticsStart: (state) => {

state.studentAnalytics.isLoading = true;

state.studentAnalytics.error = null;

},

fetchStudentAnalyticsSuccess: (state, action) => {

state.studentAnalytics.isLoading = false;

state.studentAnalytics.data = action.payload;

},

fetchStudentAnalyticsFailure: (state, action) => {

state.studentAnalytics.isLoading = false;

state.studentAnalytics.error = action.payload;

},

updateStudentAnalyticsFilters: (state, action) => {

state.studentAnalytics.filters = {

...state.studentAnalytics.filters,

...action.payload,

};

},

fetchApplicationAnalyticsStart: (state) => {

state.applicationAnalytics.isLoading = true;

state.applicationAnalytics.error = null;

},

fetchApplicationAnalyticsSuccess: (state, action) => {

state.applicationAnalytics.isLoading = false;

state.applicationAnalytics.data = action.payload;

},

fetchApplicationAnalyticsFailure: (state, action) => {

state.applicationAnalytics.isLoading = false;

state.applicationAnalytics.error = action.payload;

},

updateApplicationAnalyticsFilters: (state, action) => {

state.applicationAnalytics.filters = {

...state.applicationAnalytics.filters,

...action.payload,

};

},

fetchAgentAnalyticsStart: (state) => {

state.agentAnalytics.isLoading = true;

state.agentAnalytics.error = null;

},

fetchAgentAnalyticsSuccess: (state, action) => {

state.agentAnalytics.isLoading = false;

state.agentAnalytics.data = action.payload;

},

fetchAgentAnalyticsFailure: (state, action) => {

state.agentAnalytics.isLoading = false;

state.agentAnalytics.error = action.payload;

},

updateAgentAnalyticsFilters: (state, action) => {

state.agentAnalytics.filters = {

...state.agentAnalytics.filters,

...action.payload,

};

},

fetchRevenueAnalyticsStart: (state) => {

state.revenueAnalytics.isLoading = true;

state.revenueAnalytics.error = null;

},

fetchRevenueAnalyticsSuccess: (state, action) => {

state.revenueAnalytics.isLoading = false;

state.revenueAnalytics.data = action.payload;

},

fetchRevenueAnalyticsFailure: (state, action) => {

state.revenueAnalytics.isLoading = false;

state.revenueAnalytics.error = action.payload;

},

updateRevenueAnalyticsFilters: (state, action) => {

state.revenueAnalytics.filters = {

...state.revenueAnalytics.filters,

...action.payload,

};

},

},

});

export const {

fetchDashboardStatsStart,

fetchDashboardStatsSuccess,

fetchDashboardStatsFailure,

fetchStudentAnalyticsStart,

fetchStudentAnalyticsSuccess,

fetchStudentAnalyticsFailure,

updateStudentAnalyticsFilters,

fetchApplicationAnalyticsStart,

fetchApplicationAnalyticsSuccess,

fetchApplicationAnalyticsFailure,

updateApplicationAnalyticsFilters,

fetchAgentAnalyticsStart,

fetchAgentAnalyticsSuccess,

fetchAgentAnalyticsFailure,

updateAgentAnalyticsFilters,

fetchRevenueAnalyticsStart,

fetchRevenueAnalyticsSuccess,

fetchRevenueAnalyticsFailure,

updateRevenueAnalyticsFilters,

} = analyticsSlice.actions;

// Selectors

export const selectDashboardStats = (state) => state.analytics.dashboardStats;

export const selectStudentAnalytics = (state) => state.analytics.studentAnalytics;

export const selectApplicationAnalytics = (state) => state.analytics.applicationAnalytics;

export const selectAgentAnalytics = (state) => state.analytics.agentAnalytics;

export const selectRevenueAnalytics = (state) => state.analytics.revenueAnalytics;

export default analyticsSlice.reducer;

**11. Notification Slice**

// src/store/slices/notificationSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

notifications: [],

unreadCount: 0,

isLoading: false,

error: null,

filters: {

read: null,

type: null,

sortBy: 'createdAt',

sortOrder: 'desc',

},

pagination: {

page: 1,

limit: 10,

totalPages: 0,

totalItems: 0,

},

};

const notificationSlice = createSlice({

name: 'notifications',

initialState,

reducers: {

fetchNotificationsStart: (state) => {

state.isLoading = true;

state.error = null;

},

fetchNotificationsSuccess: (state, action) => {

state.isLoading = false;

state.notifications = action.payload.notifications;

state.unreadCount = action.payload.unreadCount;

state.pagination = action.payload.pagination;

},

fetchNotificationsFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

markAsRead: (state, action) => {

const notification = state.notifications.find(

(notification) => notification.id === action.payload

);

if (notification && !notification.read) {

notification.read = true;

notification.readAt = new Date().toISOString();

state.unreadCount = Math.max(0, state.unreadCount - 1);

}

},

markAllAsRead: (state) => {

state.notifications.forEach((notification) => {

if (!notification.read) {

notification.read = true;

notification.readAt = new Date().toISOString();

}

});

state.unreadCount = 0;

},

addNotification: (state, action) => {

state.notifications.unshift(action.payload);

if (!action.payload.read) {

state.unreadCount += 1;

}

},

updateFilters: (state, action) => {

state.filters = { ...state.filters, ...action.payload };

state.pagination.page = 1;

},

setPage: (state, action) => {

state.pagination.page = action.payload;

},

setLimit: (state, action) => {

state.pagination.limit = action.payload;

state.pagination.page = 1;

},

},

});

export const {

fetchNotificationsStart,

fetchNotificationsSuccess,

fetchNotificationsFailure,

markAsRead,

markAllAsRead,

addNotification,

updateFilters,

setPage,

setLimit,

} = notificationSlice.actions;

// Selectors

export const selectNotifications = (state) => state.notifications.notifications;

export const selectUnreadCount = (state) => state.notifications.unreadCount;

export const selectNotificationsLoading = (state) => state.notifications.isLoading;

export const selectNotificationsError = (state) => state.notifications.error;

export const selectNotificationFilters = (state) => state.notifications.filters;

export const selectNotificationPagination = (state) => state.notifications.pagination;

export default notificationSlice.reducer;

**API Integration with RTK Query**

RTK Query is used for data fetching, caching, and synchronization with the server. This approach separates server state from client state and provides optimized API calls with automatic caching.

// src/store/api/apiSlice.js

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

// Base query with auth header

const baseQuery = fetchBaseQuery({

baseUrl: process.env.REACT\_APP\_API\_URL || 'https://api.intered.com/v1',

prepareHeaders: (headers, { getState }) => {

const token = getState().auth.token;

if (token) {

headers.set('authorization', `Bearer ${token}`);

}

return headers;

},

});

// Handle token expiration and refresh

const baseQueryWithReauth = async (args, api, extraOptions) => {

let result = await baseQuery(args, api, extraOptions);

if (result.error && result.error.status === 401) {

// Try to get a new token

const refreshToken = api.getState().auth.refreshToken;

if (refreshToken) {

const refreshResult = await baseQuery(

{

url: '/auth/refresh',

method: 'POST',

body: { refreshToken },

},

api,

extraOptions

);

if (refreshResult.data) {

// Store the new token

api.dispatch(refreshTokenSuccess(refreshResult.data));

// Retry the original request

result = await baseQuery(args, api, extraOptions);

} else {

// Refresh failed, log out

api.dispatch(logout());

}

} else {

// No refresh token, log out

api.dispatch(logout());

}

}

return result;

};

// Define the API service

export const api = createApi({

reducerPath: 'api',

baseQuery: baseQueryWithReauth,

tagTypes: [

'Students',

'Student',

'Agents',

'Agent',

'Universities',

'University',

'Programs',

'Program',

'Applications',

'Application',

'Documents',

'Document',

'Commissions',

'Commission',

'Analytics',

'Notifications',

],

endpoints: (builder) => ({

// Auth endpoints

login: builder.mutation({

query: (credentials) => ({

url: '/auth/login',

method: 'POST',

body: credentials,

}),

}),

refreshToken: builder.mutation({

query: (refreshToken) => ({

url: '/auth/refresh',

method: 'POST',

body: { refreshToken },

}),

}),

// Students endpoints

getStudents: builder.query({

query: (params) => ({

url: '/students',

params,

}),

providesTags: (result) =>

result

? [

...result.students.map(({ id }) => ({ type: 'Students', id })),

{ type: 'Students', id: 'LIST' },

]

: [{ type: 'Students', id: 'LIST' }],

}),

getStudentById: builder.query({

query: (id) => `/students/${id}`,

providesTags: (result, error, id) => [{ type: 'Student', id }],

}),

createStudent: builder.mutation({

query: (student) => ({

url: '/students',

method: 'POST',

body: student,

}),

invalidatesTags: [{ type: 'Students', id: 'LIST' }],

}),

updateStudent: builder.mutation({

query: ({ id, ...patch }) => ({

url: `/students/${id}`,

method: 'PUT',

body: patch,

}),

invalidatesTags: (result, error, { id }) => [

{ type: 'Students', id: 'LIST' },

{ type: 'Student', id },

],

}),

deleteStudent: builder.mutation({

query: (id) => ({

url: `/students/${id}`,

method: 'DELETE',

}),

invalidatesTags: [{ type: 'Students', id: 'LIST' }],

}), deleteStudent: builder.mutation({ query: (id) => ({ url: /students/${id}, method: 'DELETE', }), invalidatesTags: [{ type: 'Students', id: 'LIST' }], }),

// Similar endpoints for other entities (Agents, Universities, Programs, etc.)

// ...

// Applications endpoints

getApplications: builder.query({

query: (params) => ({

url: '/applications',

params,

}),

providesTags: (result) =>

result

? [

...result.applications.map(({ id }) => ({ type: 'Applications', id })),

{ type: 'Applications', id: 'LIST' },

]

: [{ type: 'Applications', id: 'LIST' }],

}),

getApplicationById: builder.query({

query: (id) => `/applications/${id}`,

providesTags: (result, error, id) => [{ type: 'Application', id }],

}),

createApplication: builder.mutation({

query: (application) => ({

url: '/applications',

method: 'POST',

body: application,

}),

invalidatesTags: [{ type: 'Applications', id: 'LIST' }],

}),

updateApplication: builder.mutation({

query: ({ id, ...patch }) => ({

url: `/applications/${id}`,

method: 'PUT',

body: patch,

}),

invalidatesTags: (result, error, { id }) => [

{ type: 'Applications', id: 'LIST' },

{ type: 'Application', id },

],

}),

updateApplicationStatus: builder.mutation({

query: ({ id, status, notes }) => ({

url: `/applications/${id}/status`,

method: 'PUT',

body: { status, notes },

}),

invalidatesTags: (result, error, { id }) => [

{ type: 'Applications', id: 'LIST' },

{ type: 'Application', id },

],

}),

addApplicationNote: builder.mutation({

query: ({ id, content, isPrivate }) => ({

url: `/applications/${id}/notes`,

method: 'POST',

body: { content, isPrivate },

}),

invalidatesTags: (result, error, { id }) => [{ type: 'Application', id }],

}),

// Analytics endpoints

getDashboardStats: builder.query({

query: () => '/dashboard/stats',

providesTags: ['Analytics'],

}),

getStudentAnalytics: builder.query({

query: (params) => ({

url: '/analytics/students',

params,

}),

providesTags: ['Analytics'],

}),

getApplicationAnalytics: builder.query({

query: (params) => ({

url: '/analytics/applications',

params,

}),

providesTags: ['Analytics'],

}),

getAgentAnalytics: builder.query({

query: (params) => ({

url: '/analytics/agents',

params,

}),

providesTags: ['Analytics'],

}),

getRevenueAnalytics: builder.query({

query: (params) => ({

url: '/analytics/revenue',

params,

}),

providesTags: ['Analytics'],

}),

// Notifications endpoints

getNotifications: builder.query({

query: (params) => ({

url: '/notifications',

params,

}),

providesTags: ['Notifications'],

}),

markNotificationAsRead: builder.mutation({

query: (id) => ({

url: `/notifications/${id}/read`,

method: 'PUT',

}),

invalidatesTags: ['Notifications'],

}),

markAllNotificationsAsRead: builder.mutation({

query: () => ({

url: '/notifications/read-all',

method: 'PUT',

}),

invalidatesTags: ['Notifications'],

}),

}), });

// Export hooks for usage in components export const { useLoginMutation, useRefreshTokenMutation,

useGetStudentsQuery, useGetStudentByIdQuery, useCreateStudentMutation, useUpdateStudentMutation, useDeleteStudentMutation,

// Add hooks for other entity types

useGetApplicationsQuery, useGetApplicationByIdQuery, useCreateApplicationMutation, useUpdateApplicationMutation, useUpdateApplicationStatusMutation, useAddApplicationNoteMutation,

useGetDashboardStatsQuery, useGetStudentAnalyticsQuery, useGetApplicationAnalyticsQuery, useGetAgentAnalyticsQuery, useGetRevenueAnalyticsQuery,

useGetNotificationsQuery, useMarkNotificationAsReadMutation, useMarkAllNotificationsAsReadMutation, } = api;

## Thunks for Complex Operations

For operations that involve multiple actions or side effects, we use Redux Thunks:

```javascript

// src/store/thunks/authThunks.js

import { createAsyncThunk } from '@reduxjs/toolkit';

import { loginStart, loginSuccess, loginFailure, logout } from '../slices/authSlice';

import AuthService from '../../services/authService';

// Login thunk

export const loginUser = createAsyncThunk(

'auth/loginUser',

async (credentials, { dispatch }) => {

try {

dispatch(loginStart());

const response = await AuthService.login(credentials);

dispatch(loginSuccess(response.data));

return response.data;

} catch (error) {

dispatch(loginFailure(error.response?.data?.message || error.message));

throw error;

}

}

);

// Logout thunk

export const logoutUser = createAsyncThunk(

'auth/logoutUser',

async (\_, { dispatch }) => {

try {

await AuthService.logout();

dispatch(logout());

} catch (error) {

console.error('Logout error:', error);

// Still logout on client side even if server logout fails

dispatch(logout());

}

}

);

// src/store/thunks/applicationThunks.js

import { createAsyncThunk } from '@reduxjs/toolkit';

import {

fetchApplicationsStart,

fetchApplicationsSuccess,

fetchApplicationsFailure,

setSelectedApplication,

addApplication,

updateApplication,

resetFormData,

} from '../slices/applicationSlice';

import { addToast } from '../slices/uiSlice';

import ApplicationService from '../../services/applicationService';

// Fetch applications thunk

export const fetchApplications = createAsyncThunk(

'applications/fetchApplications',

async (params, { dispatch }) => {

try {

dispatch(fetchApplicationsStart());

const response = await ApplicationService.getApplications(params);

dispatch(fetchApplicationsSuccess(response.data));

return response.data;

} catch (error) {

dispatch(fetchApplicationsFailure(error.response?.data?.message || error.message));

throw error;

}

}

);

// Create application thunk

export const createApplication = createAsyncThunk(

'applications/createApplication',

async (applicationData, { dispatch }) => {

try {

const response = await ApplicationService.createApplication(applicationData);

dispatch(addApplication(response.data.application));

dispatch(resetFormData());

dispatch(

addToast({

type: 'success',

title: 'Success',

message: 'Application created successfully',

})

);

return response.data.application;

} catch (error) {

dispatch(

addToast({

type: 'error',

title: 'Error',

message: error.response?.data?.message || 'Failed to create application',

})

);

throw error;

}

}

);

// Update application status thunk

export const updateApplicationStatus = createAsyncThunk(

'applications/updateStatus',

async ({ id, status, notes }, { dispatch }) => {

try {

const response = await ApplicationService.updateApplicationStatus(id, status, notes);

dispatch(updateApplication(response.data.application));

dispatch(

addToast({

type: 'success',

title: 'Success',

message: `Application status updated to ${status}`,

})

);

return response.data.application;

} catch (error) {

dispatch(

addToast({

type: 'error',

title: 'Error',

message: error.response?.data?.message || 'Failed to update application status',

})

);

throw error;

}

}

);

**Context API for UI State**

For UI-specific state that doesn't need to be persisted in Redux, we use React Context:

// src/contexts/SidebarContext.js

import React, { createContext, useContext, useState } from 'react';

const SidebarContext = createContext();

export const SidebarProvider = ({ children }) => {

const [expanded, setExpanded] = useState(true);

const [activeItem, setActiveItem] = useState(null);

const [openGroups, setOpenGroups] = useState([]);

const toggleSidebar = () => {

setExpanded((prev) => !prev);

};

const toggleGroup = (groupId) => {

setOpenGroups((prev) =>

prev.includes(groupId)

? prev.filter((id) => id !== groupId)

: [...prev, groupId]

);

};

const isGroupOpen = (groupId) => {

return openGroups.includes(groupId);

};

const setActive = (itemId) => {

setActiveItem(itemId);

};

return (

<SidebarContext.Provider

value={{

expanded,

toggleSidebar,

activeItem,

setActive,

toggleGroup,

isGroupOpen,

}}

>

{children}

</SidebarContext.Provider>

);

};

export const useSidebar = () => {

const context = useContext(SidebarContext);

if (!context) {

throw new Error('useSidebar must be used within a SidebarProvider');

}

return context;

};

// src/contexts/FormContext.js

import React, { createContext, useContext, useReducer } from 'react';

const FormContext = createContext();

const initialState = {

values: {},

touched: {},

errors: {},

isSubmitting: false,

};

function formReducer(state, action) {

switch (action.type) {

case 'SET\_FIELD\_VALUE':

return {

...state,

values: {

...state.values,

[action.field]: action.value,

},

};

case 'SET\_FIELD\_TOUCHED':

return {

...state,

touched: {

...state.touched,

[action.field]: true,

},

};

case 'SET\_FIELD\_ERROR':

return {

...state,

errors: {

...state.errors,

[action.field]: action.error,

},

};

case 'SET\_VALUES':

return {

...state,

values: action.values,

};

case 'SET\_ERRORS':

return {

...state,

errors: action.errors,

};

case 'RESET\_FORM':

return {

...initialState,

values: action.values || {},

};

case 'SET\_SUBMITTING':

return {

...state,

isSubmitting: action.isSubmitting,

};

default:

return state;

}

}

export const FormProvider = ({ children, initialValues = {} }) => {

const [state, dispatch] = useReducer(formReducer, {

...initialState,

values: initialValues,

});

const setFieldValue = (field, value) => {

dispatch({ type: 'SET\_FIELD\_VALUE', field, value });

};

const setFieldTouched = (field) => {

dispatch({ type: 'SET\_FIELD\_TOUCHED', field });

};

const setFieldError = (field, error) => {

dispatch({ type: 'SET\_FIELD\_ERROR', field, error });

};

const setValues = (values) => {

dispatch({ type: 'SET\_VALUES', values });

};

const setErrors = (errors) => {

dispatch({ type: 'SET\_ERRORS', errors });

};

const resetForm = (values) => {

dispatch({ type: 'RESET\_FORM', values });

};

const setSubmitting = (isSubmitting) => {

dispatch({ type: 'SET\_SUBMITTING', isSubmitting });

};

return (

<FormContext.Provider

value={{

...state,

setFieldValue,

setFieldTouched,

setFieldError,

setValues,

setErrors,

resetForm,

setSubmitting,

}}

>

{children}

</FormContext.Provider>

);

};

export const useForm = () => {

const context = useContext(FormContext);

if (!context) {

throw new Error('useForm must be used within a FormProvider');

}

return context;

};

**Custom Hooks for State Management**

Custom hooks provide an abstraction layer over Redux and Context API:

// src/hooks/useAuth.js

import { useSelector, useDispatch } from 'react-redux';

import { selectIsAuthenticated, selectCurrentUser, selectUserRole, selectUserPermissions } from '../store/slices/authSlice';

import { loginUser, logoutUser } from '../store/thunks/authThunks';

export const useAuth = () => {

const dispatch = useDispatch();

const isAuthenticated = useSelector(selectIsAuthenticated);

const user = useSelector(selectCurrentUser);

const role = useSelector(selectUserRole);

const permissions = useSelector(selectUserPermissions);

const login = (credentials) => {

return dispatch(loginUser(credentials));

};

const logout = () => {

return dispatch(logoutUser());

};

const hasPermission = (permission) => {

return permissions.includes(permission);

};

const isAdmin = role === 'admin';

const isStaff = role === 'staff' || role === 'admin';

const isAgent = role === 'agent';

const isStudent = role === 'student';

return {

isAuthenticated,

user,

role,

permissions,

login,

logout,

hasPermission,

isAdmin,

isStaff,

isAgent,

isStudent,

};

};

// src/hooks/useStudents.js

import { useSelector, useDispatch } from 'react-redux';

import {

selectStudents,

selectSelectedStudent,

selectStudentsLoading,

selectStudentsError,

selectStudentFilters,

selectStudentPagination,

updateFilters,

setPage,

setLimit,

setSelectedStudent,

clearSelectedStudent,

} from '../store/slices/studentSlice';

import { useGetStudentsQuery, useGetStudentByIdQuery } from '../store/api/apiSlice';

export const useStudents = () => {

const dispatch = useDispatch();

const students = useSelector(selectStudents);

const selectedStudent = useSelector(selectSelectedStudent);

const isLoading = useSelector(selectStudentsLoading);

const error = useSelector(selectStudentsError);

const filters = useSelector(selectStudentFilters);

const pagination = useSelector(selectStudentPagination);

// RTK Query hooks

const { refetch: refetchStudents } = useGetStudentsQuery(

{

page: pagination.page,

limit: pagination.limit,

...filters,

},

{ skip: !filters }

);

const setFilters = (newFilters) => {

dispatch(updateFilters(newFilters));

};

const changePage = (newPage) => {

dispatch(setPage(newPage));

};

const changeLimit = (newLimit) => {

dispatch(setLimit(newLimit));

};

const selectStudent = (student) => {

dispatch(setSelectedStudent(student));

};

const clearStudent = () => {

dispatch(clearSelectedStudent());

};

return {

students,

selectedStudent,

isLoading,

error,

filters,

pagination,

setFilters,

changePage,

changeLimit,

selectStudent,

clearStudent,

refetchStudents,

};

};

Similar custom hooks can be created for other entity types (agents, applications, etc.)

**Component Integration Examples**

Here's how to integrate this state management structure into React components:

**Login Form Component**

// src/components/auth/LoginForm.js

import React from 'react';

import { useForm } from 'react-hook-form';

import { yupResolver } from '@hookform/resolvers/yup';

import \* as yup from 'yup';

import { useAuth } from '../../hooks/useAuth';

import { Button } from '../ui/Button';

import { Input } from '../ui/Input';

import { Alert } from '../ui/Alert';

const schema = yup.object().shape({

email: yup.string().email('Invalid email address').required('Email is required'),

password: yup.string().required('Password is required'),

});

const LoginForm = () => {

const { login, isAuthenticated } = useAuth();

const [error, setError] = React.useState(null);

const [isLoading, setIsLoading] = React.useState(false);

const {

register,

handleSubmit,

formState: { errors },

} = useForm({

resolver: yupResolver(schema),

});

const onSubmit = async (data) => {

try {

setIsLoading(true);

setError(null);

await login(data);

} catch (err) {

setError(err.message || 'Failed to log in. Please check your credentials.');

} finally {

setIsLoading(false);

}

};

return (

<form onSubmit={handleSubmit(onSubmit)} className="space-y-6">

{error && <Alert variant="error">{error}</Alert>}

<div>

<label htmlFor="email" className="block text-sm font-medium text-gray-700">

Email

</label>

<Input

id="email"

type="email"

className="mt-1"

error={errors.email?.message}

{...register('email')}

/>

</div>

<div>

<label htmlFor="password" className="block text-sm font-medium text-gray-700">

Password

</label>

<Input

id="password"

type="password"

className="mt-1"

error={errors.password?.message}

{...register('password')}

/>

</div>

<Button type="submit" className="w-full" disabled={isLoading}>

{isLoading ? 'Logging in...' : 'Log in'}

</Button>

</form>

);

};

export default LoginForm;

**Student List Component**

// src/components/students/StudentList.js

import React from 'react';

import { useStudents } from '../../hooks/useStudents';

import { useNavigate } from 'react-router-dom';

import { DataTable } from '../ui/DataTable';

import { SearchInput } from '../ui/SearchInput';

import { Button } from '../ui/Button';

import { Select } from '../ui/Select';

import { Badge } from '../ui/Badge';

import { Plus } from 'lucide-react';

const StudentList = () => {

const navigate = useNavigate();

const {

students,

isLoading,

error,

filters,

pagination,

setFilters,

changePage,

changeLimit,

selectStudent,

} = useStudents();

const handleSearch = (value) => {

setFilters({ search: value });

};

const handleStatusChange = (e) => {

setFilters({ status: e.target.value });

};

const handleRowClick = (student) => {

selectStudent(student);

navigate(`/students/${student.id}`);

};

const columns = [

{

Header: 'Name',

accessor: (row) => `${row.firstName} ${row.lastName}`,

Cell: ({ row }) => (

<div className="flex items-center">

<div className="flex-shrink-0 h-10 w-10">

<img

className="h-10 w-10 rounded-full"

src={row.original.avatar || `https://ui-avatars.com/api/?name=${row.original.firstName}+${row.original.lastName}`}

alt=""

/>

</div>

<div className="ml-4">

<div className="text-sm font-medium text-gray-900">

{row.original.firstName} {row.original.lastName}

</div>

<div className="text-sm text-gray-500">{row.original.email}</div>

</div>

</div>

),

},

{

Header: 'Phone',

accessor: 'phone',

},

{

Header: 'Agent',

accessor: 'agentName',

Cell: ({ value }) => (value ? value : 'Direct'),

},

{

Header: 'Status',

accessor: 'status',

Cell: ({ value }) => {

const statusVariants = {

active: 'success',

inactive: 'secondary',

graduated: 'info',

withdrawn: 'warning',

};

return <Badge variant={statusVariants[value] || 'default'}>{value}</Badge>;

},

},

{

Header: 'Applications',

accessor: 'applicationCount',

},

{

Header: 'Created At',

accessor: 'createdAt',

Cell: ({ value }) => new Date(value).toLocaleDateString(),

},

];

return (

<div className="space-y-4">

<div className="flex justify-between">

<h2 className="text-2xl font-bold">Students</h2>

<Button onClick={() => navigate('/students/new')} startIcon={<Plus size={16} />}>

Add Student

</Button>

</div>

<div className="flex flex-col md:flex-row gap-4 justify-between">

<SearchInput

placeholder="Search students..."

value={filters.search}

onChange={(e) => handleSearch(e.target.value)}

className="md:w-96"

/>

<div className="flex gap-4">

<Select

value={filters.status || ''}

onChange={handleStatusChange}

className="w-48"

>

<option value="">All Statuses</option>

<option value="active">Active</option>

<option value="inactive">Inactive</option>

<option value="graduated">Graduated</option>

<option value="withdrawn">Withdrawn</option>

</Select>

</div>

</div>

<DataTable

columns={columns}

data={students}

isLoading={isLoading}

onRowClick={handleRowClick}

pagination={pagination}

onPageChange={changePage}

onLimitChange={changeLimit}

error={error}

/>

</div>

);

};

export default StudentList;

**Application State Flow Diagram**

┌─────────────────────────────────────────────────────────────────┐

│ Component │

│ │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────────┐ │

│ │ Local State │ │ React Query │ │ Custom Hooks │ │

│ └─────────────┘ └─────────────┘ └─────────────────┘ │

│ │ │ │ │

└──────────┼──────────────────┼─────────────────────┼──────────────┘

│ │ │

▼ ▼ ▼

┌────────────────┐ ┌─────────────────┐ ┌─────────────────────┐

│ UI Interactions │ │ API Interactions │ │ Business Logic │

└────────┬────────┘ └────────┬────────┘ └──────────┬──────────┘

│ │ │

▼ ▼ ▼

┌─────────────────────────────────────────────────────────────────┐

│ Actions / Thunks │

└─────────────────────────────────┬───────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────────┐

│ Redux Store │

│ │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────────┐ │

│ │ Auth Slice │ │ Entity Slices│ │ UI Slice │ │

│ └─────────────┘ └─────────────┘ └─────────────────┘ │

│ │

└─────────────────────────────────┬───────────────────────────────┘

│

▼

┌─────────────────────────────────────────────────────────────────┐

│ React Components │

│ (Re-render when state changes) │

└─────────────────────────────────────────────────────────────────┘

**Best Practices for State Management**

**1. Organization Strategy**

* **Domain-Driven Design**: Organize state by business domain (students, agents, applications)
* **Feature-Based Structure**: Group related functionality together (slice, thunks, hooks)
* **Progressive Enhancement**: Start with simple state, enhance as needed

**2. Performance Optimization**

* **Selective Rerendering**: Use selectors to minimize component updates
* **Memoization**: Memoize selectors and components to reduce calculations
* **Normalized State**: Store entities in a normalized format with IDs as references
* **Optimistic Updates**: Update the UI immediately, then sync with the server

**3. Code Maintainability**

* **Clear Actions**: Use descriptive action names and payload structures
* **Type Safety**: Use TypeScript for type checking
* **Documentation**: Document slice purposes, action parameters, and expected behavior
* **Testability**: Write unit tests for reducers, selectors, and thunks

**Specific Recommendations for the Vibathon**

1. **Pre-configure slices first**: Set up all Redux slices before the Vibathon
2. **Create API slice templates**: Prepare RTK Query endpoints for each entity
3. **Implement authentication flow**: Complete this critical path early
4. **Build custom hooks layer**: This abstraction will speed up component development
5. **Prepare mock data**: Use MSW or similar to mock API responses during development

**Conclusion**

This state management structure provides a comprehensive foundation for the InterEd Admin Portal, offering:

* **Centralized state management** with Redux Toolkit
* **Efficient data fetching** with RTK Query
* **Component-level abstraction** with custom hooks
* **UI state separation** with Context API
* **Scalable architecture** that can evolve with application needs

By implementing this structure before the Vibathon, your team will be able to focus on feature development and UI implementation rather than architectural decisions during the 48-hour sprint. The separation of concerns between client state and server state allows for efficient management of complex data requirements while maintaining performance and maintainability.

**InterEd Admin Portal State Management Structure**

**Overview**

This document outlines a comprehensive state management architecture for the InterEd Admin Portal based on Redux Toolkit. The architecture is designed to handle the complex state requirements of the application while ensuring maintainability, performance, and scalability. This state management structure can be implemented at approximately 85-90% completion before the Vibathon.

**State Management Architecture**

The InterEd Admin Portal will use a combination of:

* **Redux Toolkit** as the primary state management solution
* **React Query** for server state (API data fetching, caching, and synchronization)
* **Context API** for UI-specific state that doesn't need global persistence
* **Local component state** for form inputs and UI interactions

This layered approach provides optimal performance while maintaining a clear separation of concerns.

**Redux Store Structure**

**Store Configuration**

// src/store/index.js

import { configureStore } from '@reduxjs/toolkit';

import { setupListeners } from '@reduxjs/toolkit/query';

import storage from 'redux-persist/lib/storage';

import { persistReducer, persistStore } from 'redux-persist';

import { combineReducers } from 'redux';

// Import reducers

import authReducer from './slices/authSlice';

import uiReducer from './slices/uiSlice';

import studentReducer from './slices/studentSlice';

import agentReducer from './slices/agentSlice';

import universityReducer from './slices/universitySlice';

import programReducer from './slices/programSlice';

import applicationReducer from './slices/applicationSlice';

import documentReducer from './slices/documentSlice';

import commissionReducer from './slices/commissionSlice';

import analyticsReducer from './slices/analyticsSlice';

import notificationReducer from './slices/notificationSlice';

// Import API

import { api } from './api/apiSlice';

// Configure persist

const persistConfig = {

key: 'root',

storage,

whitelist: ['auth'], // Only persist authentication data

};

const rootReducer = combineReducers({

auth: authReducer,

ui: uiReducer,

students: studentReducer,

agents: agentReducer,

universities: universityReducer,

programs: programReducer,

applications: applicationReducer,

documents: documentReducer,

commissions: commissionReducer,

analytics: analyticsReducer,

notifications: notificationReducer,

[api.reducerPath]: api.reducer,

});

const persistedReducer = persistReducer(persistConfig, rootReducer);

export const store = configureStore({

reducer: persistedReducer,

middleware: (getDefaultMiddleware) =>

getDefaultMiddleware({

serializableCheck: {

ignoredActions: ['persist/PERSIST', 'persist/REHYDRATE'],

ignoredPaths: ['some.path.to.ignore'],

},

}).concat(api.middleware),

});

export const persistor = persistStore(store);

// Setup listeners for RTK Query

setupListeners(store.dispatch);

**Redux Slices**

The application state is divided into domain-specific slices:

**1. Authentication Slice**

// src/store/slices/authSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

user: null,

token: null,

refreshToken: null,

expiresAt: null,

isAuthenticated: false,

isLoading: false,

error: null,

};

const authSlice = createSlice({

name: 'auth',

initialState,

reducers: {

loginStart: (state) => {

state.isLoading = true;

state.error = null;

},

loginSuccess: (state, action) => {

state.isLoading = false;

state.isAuthenticated = true;

state.user = action.payload.user;

state.token = action.payload.token;

state.refreshToken = action.payload.refreshToken;

state.expiresAt = action.payload.expiresAt;

},

loginFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

logout: (state) => {

state.user = null;

state.token = null;

state.refreshToken = null;

state.expiresAt = null;

state.isAuthenticated = false;

},

refreshTokenSuccess: (state, action) => {

state.token = action.payload.token;

state.refreshToken = action.payload.refreshToken;

state.expiresAt = action.payload.expiresAt;

},

updateUserProfile: (state, action) => {

state.user = { ...state.user, ...action.payload };

},

},

});

export const {

loginStart,

loginSuccess,

loginFailure,

logout,

refreshTokenSuccess,

updateUserProfile,

} = authSlice.actions;

// Selectors

export const selectCurrentUser = (state) => state.auth.user;

export const selectIsAuthenticated = (state) => state.auth.isAuthenticated;

export const selectAuthToken = (state) => state.auth.token;

export const selectAuthError = (state) => state.auth.error;

export const selectIsAdmin = (state) => state.auth.user?.role === 'admin';

export const selectUserRole = (state) => state.auth.user?.role;

export const selectUserPermissions = (state) => state.auth.user?.permissions || [];

export default authSlice.reducer;

**2. UI Slice**

// src/store/slices/uiSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

sidebarOpen: true,

currentTheme: 'light',

toasts: [],

activeModal: null,

modalData: null,

breadcrumbs: [],

isMobileView: false,

};

const uiSlice = createSlice({

name: 'ui',

initialState,

reducers: {

toggleSidebar: (state) => {

state.sidebarOpen = !state.sidebarOpen;

},

setSidebarOpen: (state, action) => {

state.sidebarOpen = action.payload;

},

setTheme: (state, action) => {

state.currentTheme = action.payload;

},

addToast: (state, action) => {

state.toasts.push({

id: Date.now(),

...action.payload,

});

},

removeToast: (state, action) => {

state.toasts = state.toasts.filter((toast) => toast.id !== action.payload);

},

openModal: (state, action) => {

state.activeModal = action.payload.modal;

state.modalData = action.payload.data || null;

},

closeModal: (state) => {

state.activeModal = null;

state.modalData = null;

},

setBreadcrumbs: (state, action) => {

state.breadcrumbs = action.payload;

},

setMobileView: (state, action) => {

state.isMobileView = action.payload;

},

},

});

export const {

toggleSidebar,

setSidebarOpen,

setTheme,

addToast,

removeToast,

openModal,

closeModal,

setBreadcrumbs,

setMobileView,

} = uiSlice.actions;

// Selectors

export const selectSidebarOpen = (state) => state.ui.sidebarOpen;

export const selectCurrentTheme = (state) => state.ui.currentTheme;

export const selectToasts = (state) => state.ui.toasts;

export const selectActiveModal = (state) => state.ui.activeModal;

export const selectModalData = (state) => state.ui.modalData;

export const selectBreadcrumbs = (state) => state.ui.breadcrumbs;

export const selectIsMobileView = (state) => state.ui.isMobileView;

export default uiSlice.reducer;

**3. Student Slice**

// src/store/slices/studentSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

students: [],

selectedStudent: null,

isLoading: false,

error: null,

filters: {

status: null,

search: '',

agent: null,

sortBy: 'createdAt',

sortOrder: 'desc',

},

pagination: {

page: 1,

limit: 10,

totalPages: 0,

totalItems: 0,

},

};

const studentSlice = createSlice({

name: 'students',

initialState,

reducers: {

fetchStudentsStart: (state) => {

state.isLoading = true;

state.error = null;

},

fetchStudentsSuccess: (state, action) => {

state.isLoading = false;

state.students = action.payload.students;

state.pagination = action.payload.pagination;

},

fetchStudentsFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

setSelectedStudent: (state, action) => {

state.selectedStudent = action.payload;

},

clearSelectedStudent: (state) => {

state.selectedStudent = null;

},

updateFilters: (state, action) => {

state.filters = { ...state.filters, ...action.payload };

state.pagination.page = 1; // Reset to first page when filters change

},

setPage: (state, action) => {

state.pagination.page = action.payload;

},

setLimit: (state, action) => {

state.pagination.limit = action.payload;

state.pagination.page = 1; // Reset to first page when limit changes

},

addStudent: (state, action) => {

state.students.unshift(action.payload);

},

updateStudent: (state, action) => {

const index = state.students.findIndex(

(student) => student.id === action.payload.id

);

if (index !== -1) {

state.students[index] = {

...state.students[index],

...action.payload,

};

}

if (state.selectedStudent?.id === action.payload.id) {

state.selectedStudent = {

...state.selectedStudent,

...action.payload,

};

}

},

removeStudent: (state, action) => {

state.students = state.students.filter(

(student) => student.id !== action.payload

);

if (state.selectedStudent?.id === action.payload) {

state.selectedStudent = null;

}

},

},

});

export const {

fetchStudentsStart,

fetchStudentsSuccess,

fetchStudentsFailure,

setSelectedStudent,

clearSelectedStudent,

updateFilters,

setPage,

setLimit,

addStudent,

updateStudent,

removeStudent,

} = studentSlice.actions;

// Selectors

export const selectStudents = (state) => state.students.students;

export const selectSelectedStudent = (state) => state.students.selectedStudent;

export const selectStudentsLoading = (state) => state.students.isLoading;

export const selectStudentsError = (state) => state.students.error;

export const selectStudentFilters = (state) => state.students.filters;

export const selectStudentPagination = (state) => state.students.pagination;

export default studentSlice.reducer;

**4. Agent Slice**

Structure similar to the Student slice, but for agent-specific state.

**5. University Slice**

Structure similar to the Student slice, but for university-specific state.

**6. Program Slice**

Structure similar to the Student slice, but for program-specific state.

**7. Application Slice**

// src/store/slices/applicationSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

applications: [],

selectedApplication: null,

isLoading: false,

error: null,

filters: {

status: null,

student: null,

agent: null,

university: null,

program: null,

submittedDateStart: null,

submittedDateEnd: null,

search: '',

sortBy: 'submittedAt',

sortOrder: 'desc',

},

pagination: {

page: 1,

limit: 10,

totalPages: 0,

totalItems: 0,

},

currentStep: 1,

formData: {},

activeTab: 'details',

};

const applicationSlice = createSlice({

name: 'applications',

initialState,

reducers: {

fetchApplicationsStart: (state) => {

state.isLoading = true;

state.error = null;

},

fetchApplicationsSuccess: (state, action) => {

state.isLoading = false;

state.applications = action.payload.applications;

state.pagination = action.payload.pagination;

},

fetchApplicationsFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

setSelectedApplication: (state, action) => {

state.selectedApplication = action.payload;

},

clearSelectedApplication: (state) => {

state.selectedApplication = null;

},

updateFilters: (state, action) => {

state.filters = { ...state.filters, ...action.payload };

state.pagination.page = 1;

},

setPage: (state, action) => {

state.pagination.page = action.payload;

},

setLimit: (state, action) => {

state.pagination.limit = action.payload;

state.pagination.page = 1;

},

addApplication: (state, action) => {

state.applications.unshift(action.payload);

},

updateApplication: (state, action) => {

const index = state.applications.findIndex(

(application) => application.id === action.payload.id

);

if (index !== -1) {

state.applications[index] = {

...state.applications[index],

...action.payload,

};

}

if (state.selectedApplication?.id === action.payload.id) {

state.selectedApplication = {

...state.selectedApplication,

...action.payload,

};

}

},

removeApplication: (state, action) => {

state.applications = state.applications.filter(

(application) => application.id !== action.payload

);

if (state.selectedApplication?.id === action.payload) {

state.selectedApplication = null;

}

},

setCurrentStep: (state, action) => {

state.currentStep = action.payload;

},

updateFormData: (state, action) => {

state.formData = { ...state.formData, ...action.payload };

},

resetFormData: (state) => {

state.formData = {};

state.currentStep = 1;

},

setActiveTab: (state, action) => {

state.activeTab = action.payload;

},

addNote: (state, action) => {

if (state.selectedApplication) {

const notes = state.selectedApplication.notes || [];

state.selectedApplication.notes = [action.payload, ...notes];

}

},

},

});

export const {

fetchApplicationsStart,

fetchApplicationsSuccess,

fetchApplicationsFailure,

setSelectedApplication,

clearSelectedApplication,

updateFilters,

setPage,

setLimit,

addApplication,

updateApplication,

removeApplication,

setCurrentStep,

updateFormData,

resetFormData,

setActiveTab,

addNote,

} = applicationSlice.actions;

// Selectors

export const selectApplications = (state) => state.applications.applications;

export const selectSelectedApplication = (state) => state.applications.selectedApplication;

export const selectApplicationsLoading = (state) => state.applications.isLoading;

export const selectApplicationsError = (state) => state.applications.error;

export const selectApplicationFilters = (state) => state.applications.filters;

export const selectApplicationPagination = (state) => state.applications.pagination;

export const selectCurrentStep = (state) => state.applications.currentStep;

export const selectFormData = (state) => state.applications.formData;

export const selectActiveTab = (state) => state.applications.activeTab;

export default applicationSlice.reducer;

**8. Document Slice**

Structure similar to previous slices, but for document-specific state.

**9. Commission Slice**

Structure similar to previous slices, but for commission-specific state.

**10. Analytics Slice**

// src/store/slices/analyticsSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

dashboardStats: {

isLoading: false,

error: null,

data: null,

},

studentAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

},

},

applicationAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

groupBy: 'status',

},

},

agentAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

limit: 10,

},

},

revenueAnalytics: {

isLoading: false,

error: null,

data: null,

filters: {

period: 'month',

startDate: null,

endDate: null,

},

},

};

const analyticsSlice = createSlice({

name: 'analytics',

initialState,

reducers: {

fetchDashboardStatsStart: (state) => {

state.dashboardStats.isLoading = true;

state.dashboardStats.error = null;

},

fetchDashboardStatsSuccess: (state, action) => {

state.dashboardStats.isLoading = false;

state.dashboardStats.data = action.payload;

},

fetchDashboardStatsFailure: (state, action) => {

state.dashboardStats.isLoading = false;

state.dashboardStats.error = action.payload;

},

fetchStudentAnalyticsStart: (state) => {

state.studentAnalytics.isLoading = true;

state.studentAnalytics.error = null;

},

fetchStudentAnalyticsSuccess: (state, action) => {

state.studentAnalytics.isLoading = false;

state.studentAnalytics.data = action.payload;

},

fetchStudentAnalyticsFailure: (state, action) => {

state.studentAnalytics.isLoading = false;

state.studentAnalytics.error = action.payload;

},

updateStudentAnalyticsFilters: (state, action) => {

state.studentAnalytics.filters = {

...state.studentAnalytics.filters,

...action.payload,

};

},

fetchApplicationAnalyticsStart: (state) => {

state.applicationAnalytics.isLoading = true;

state.applicationAnalytics.error = null;

},

fetchApplicationAnalyticsSuccess: (state, action) => {

state.applicationAnalytics.isLoading = false;

state.applicationAnalytics.data = action.payload;

},

fetchApplicationAnalyticsFailure: (state, action) => {

state.applicationAnalytics.isLoading = false;

state.applicationAnalytics.error = action.payload;

},

updateApplicationAnalyticsFilters: (state, action) => {

state.applicationAnalytics.filters = {

...state.applicationAnalytics.filters,

...action.payload,

};

},

fetchAgentAnalyticsStart: (state) => {

state.agentAnalytics.isLoading = true;

state.agentAnalytics.error = null;

},

fetchAgentAnalyticsSuccess: (state, action) => {

state.agentAnalytics.isLoading = false;

state.agentAnalytics.data = action.payload;

},

fetchAgentAnalyticsFailure: (state, action) => {

state.agentAnalytics.isLoading = false;

state.agentAnalytics.error = action.payload;

},

updateAgentAnalyticsFilters: (state, action) => {

state.agentAnalytics.filters = {

...state.agentAnalytics.filters,

...action.payload,

};

},

fetchRevenueAnalyticsStart: (state) => {

state.revenueAnalytics.isLoading = true;

state.revenueAnalytics.error = null;

},

fetchRevenueAnalyticsSuccess: (state, action) => {

state.revenueAnalytics.isLoading = false;

state.revenueAnalytics.data = action.payload;

},

fetchRevenueAnalyticsFailure: (state, action) => {

state.revenueAnalytics.isLoading = false;

state.revenueAnalytics.error = action.payload;

},

updateRevenueAnalyticsFilters: (state, action) => {

state.revenueAnalytics.filters = {

...state.revenueAnalytics.filters,

...action.payload,

};

},

},

});

export const {

fetchDashboardStatsStart,

fetchDashboardStatsSuccess,

fetchDashboardStatsFailure,

fetchStudentAnalyticsStart,

fetchStudentAnalyticsSuccess,

fetchStudentAnalyticsFailure,

updateStudentAnalyticsFilters,

fetchApplicationAnalyticsStart,

fetchApplicationAnalyticsSuccess,

fetchApplicationAnalyticsFailure,

updateApplicationAnalyticsFilters,

fetchAgentAnalyticsStart,

fetchAgentAnalyticsSuccess,

fetchAgentAnalyticsFailure,

updateAgentAnalyticsFilters,

fetchRevenueAnalyticsStart,

fetchRevenueAnalyticsSuccess,

fetchRevenueAnalyticsFailure,

updateRevenueAnalyticsFilters,

} = analyticsSlice.actions;

// Selectors

export const selectDashboardStats = (state) => state.analytics.dashboardStats;

export const selectStudentAnalytics = (state) => state.analytics.studentAnalytics;

export const selectApplicationAnalytics = (state) => state.analytics.applicationAnalytics;

export const selectAgentAnalytics = (state) => state.analytics.agentAnalytics;

export const selectRevenueAnalytics = (state) => state.analytics.revenueAnalytics;

export default analyticsSlice.reducer;

**11. Notification Slice**

// src/store/slices/notificationSlice.js

import { createSlice } from '@reduxjs/toolkit';

const initialState = {

notifications: [],

unreadCount: 0,

isLoading: false,

error: null,

filters: {

read: null,

type: null,

sortBy: 'createdAt',

sortOrder: 'desc',

},

pagination: {

page: 1,

limit: 10,

totalPages: 0,

totalItems: 0,

},

};

const notificationSlice = createSlice({

name: 'notifications',

initialState,

reducers: {

fetchNotificationsStart: (state) => {

state.isLoading = true;

state.error = null;

},

fetchNotificationsSuccess: (state, action) => {

state.isLoading = false;

state.notifications = action.payload.notifications;

state.unreadCount = action.payload.unreadCount;

state.pagination = action.payload.pagination;

},

fetchNotificationsFailure: (state, action) => {

state.isLoading = false;

state.error = action.payload;

},

markAsRead: (state, action) => {

const notification = state.notifications.find(

(notification) => notification.id === action.payload

);

if (notification && !notification.read) {

notification.read = true;

notification.readAt = new Date().toISOString();

state.unreadCount = Math.max(0, state.unreadCount - 1);

}

},

markAllAsRead: (state) => {

state.notifications.forEach((notification) => {

if (!notification.read) {

notification.read = true;

notification.readAt = new Date().toISOString();

}

});

state.unreadCount = 0;

},

addNotification: (state, action) => {

state.notifications.unshift(action.payload);

if (!action.payload.read) {

state.unreadCount += 1;

}

},

updateFilters: (state, action) => {

state.filters = { ...state.filters, ...action.payload };

state.pagination.page = 1;

},

setPage: (state, action) => {

state.pagination.page = action.payload;

},

setLimit: (state, action) => {

state.pagination.limit = action.payload;

state.pagination.page = 1;

},

},

});

export const {

fetchNotificationsStart,

fetchNotificationsSuccess,

fetchNotificationsFailure,

markAsRead,

markAllAsRead,

addNotification,

updateFilters,

setPage,

setLimit,

} = notificationSlice.actions;

// Selectors

export const selectNotifications = (state) => state.notifications.notifications;

export const selectUnreadCount = (state) => state.notifications.unreadCount;

export const selectNotificationsLoading = (state) => state.notifications.isLoading;

export const selectNotificationsError = (state) => state.notifications.error;

export const selectNotificationFilters = (state) => state.notifications.filters;

export const selectNotificationPagination = (state) => state.notifications.pagination;

export default notificationSlice.reducer;

**API Integration with RTK Query**

RTK Query is used for data fetching, caching, and synchronization with the server. This approach separates server state from client state and provides optimized API calls with automatic caching.

// src/store/api/apiSlice.js

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

// Base query with auth header

const baseQuery = fetchBaseQuery({

baseUrl: process.env.REACT\_APP\_API\_URL || 'https://api.intered.com/v1',

prepareHeaders: (headers, { getState }) => {

const token = getState().auth.token;

if (token) {

headers.set('authorization', `Bearer ${token}`);

}

return headers;

},

});

// Handle token expiration and refresh

const baseQueryWithReauth = async (args, api, extraOptions) => {

let result = await baseQuery(args, api, extraOptions);

if (result.error && result.error.status === 401) {

// Try to get a new token

const refreshToken = api.getState().auth.refreshToken;

if (refreshToken) {

const refreshResult = await baseQuery(

{

url: '/auth/refresh',

method: 'POST',

body: { refreshToken },

},

api,

extraOptions

);

if (refreshResult.data) {

// Store the new token

api.dispatch(refreshTokenSuccess(refreshResult.data));

// Retry the original request

result = await baseQuery(args, api, extraOptions);

} else {

// Refresh failed, log out

api.dispatch(logout());

}

} else {

// No refresh token, log out

api.dispatch(logout());

}

}

return result;

};

// Define the API service

export const api = createApi({

reducerPath: 'api',

baseQuery: baseQueryWithReauth,

tagTypes: [

'Students',

'Student',

'Agents',

'Agent',

'Universities',

'University',

'Programs',

'Program',

'Applications',

'Application',

'Documents',

'Document',

'Commissions',

'Commission',

'Analytics',

'Notifications',

],

endpoints: (builder) => ({

// Auth endpoints

login: builder.mutation({

query: (credentials) => ({

url: '/auth/login',

method: 'POST',

body: credentials,

}),

}),

refreshToken: builder.mutation({

query: (refreshToken) => ({

url: '/auth/refresh',

method: 'POST',

body: { refreshToken },

}),

}),

// Students endpoints

getStudents: builder.query({

query: (params) => ({

url: '/students',

params,

}),

providesTags: (result) =>

result

? [

...result.students.map(({ id }) => ({ type: 'Students', id })),

{ type: 'Students', id: 'LIST' },

]

: [{ type: 'Students', id: 'LIST' }],

}),

getStudentById: builder.query({

query: (id) => `/students/${id}`,

providesTags: (result, error, id) => [{ type: 'Student', id }],

}),

createStudent: builder.mutation({

query: (student) => ({

url: '/students',

method: 'POST',

body: student,

}),

invalidatesTags: [{ type: 'Students', id: 'LIST' }],

}),

updateStudent: builder.mutation({

query: ({ id, ...patch }) => ({

url: `/students/${id}`,

method: 'PUT',

body: patch,

}),

invalidatesTags: (result, error, { id }) => [

{ type: 'Students', id: 'LIST' },

{ type: 'Student', id },

],

}),

deleteStudent: builder.mutation({

query: (id) => ({

url: `/students/${id}`,

method: 'DELETE',

}),

invalidatesTags: [{ type: 'Students', id: 'LIST' }],

} catch (error) { console.error('Reset password error:', error); return res.status(500).json({ success: false, error: { message: 'Internal server error' } }); } };

exports.logout = async (req, res) => { try { const authHeader = req.headers.authorization;

if (!authHeader || !authHeader.startsWith('Bearer ')) {

return res.status(400).json({

success: false,

error: {

message: 'Invalid authorization header'

}

});

}

// Invalidate refresh token

if (req.user && req.user.id) {

await invalidateAllRefreshTokens(req.user.id);

}

return res.status(200).json({

success: true,

message: 'Logged out successfully'

});

} catch (error) { console.error('Logout error:', error); return res.status(500).json({ success: false, error: { message: 'Internal server error' } }); } };

// Helper functions

// Get user permissions based on role const getUserPermissions = async (role) => { // Define permissions for each role const permissionsMap = { admin: [ 'read:all', 'write:all', 'delete:all', 'manage:users', 'manage:agents', 'manage:universities', 'manage:reports' ], staff: [ 'read:all', 'write:applications', 'write:students', 'write:documents', 'verify:documents', 'process:applications' ], agent: [ 'read:own\_students', 'read:own\_applications', 'write:own\_students', 'write:own\_applications', 'upload:documents' ], student: [ 'read:own\_profile', 'read:own\_applications', 'write:own\_profile', 'write:own\_applications', 'upload:own\_documents' ] };

return permissionsMap[role] || []; };

// Store refresh token in database const storeRefreshToken = async (userId, token) => { const { error } = await supabase .from('refresh\_tokens') .insert({ userId, token, createdAt: new Date(), expiresAt: new Date(Date.now() + 7 \* 24 \* 60 \* 60 \* 1000), // 7 days isRevoked: false });

if (error) { console.error('Store refresh token error:', error); throw new Error('Failed to store refresh token'); } };

// Validate stored refresh token const validateStoredRefreshToken = async (userId, token) => { const { data, error } = await supabase .from('refresh\_tokens') .select('\*') .eq('userId', userId) .eq('token', token) .eq('isRevoked', false) .single();

if (error || !data) { return false; }

// Check if token is expired const expiresAt = new Date(data.expiresAt); if (expiresAt < new Date()) { return false; }

return true; };

// Invalidate refresh token const invalidateRefreshToken = async (userId, token) => { const { error } = await supabase .from('refresh\_tokens') .update({ isRevoked: true }) .eq('userId', userId) .eq('token', token);

if (error) { console.error('Invalidate refresh token error:', error); throw new Error('Failed to invalidate refresh token'); } };

// Invalidate all refresh tokens for a user const invalidateAllRefreshTokens = async (userId) => { const { error } = await supabase .from('refresh\_tokens') .update({ isRevoked: true }) .eq('userId', userId) .eq('isRevoked', false);

if (error) { console.error('Invalidate all refresh tokens error:', error); throw new Error('Failed to invalidate refresh tokens'); } };

### 3. Auth Middleware

```javascript

// server/middleware/auth.js

const jwt = require('jsonwebtoken');

const supabase = require('../config/supabase');

const JWT\_SECRET = process.env.JWT\_SECRET;

exports.authenticate = async (req, res, next) => {

// Get token from header

const authHeader = req.headers.authorization;

if (!authHeader || !authHeader.startsWith('Bearer ')) {

return res.status(401).json({

success: false,

error: {

code: 'AUTHENTICATION\_ERROR',

message: 'No token provided'

}

});

}

const token = authHeader.split(' ')[1];

try {

// Verify token

const decoded = jwt.verify(token, JWT\_SECRET);

// Check if user exists

const { data: user, error } = await supabase.auth.admin.getUserById(decoded.sub);

if (error || !user) {

return res.status(401).json({

success: false,

error: {

code: 'AUTHENTICATION\_ERROR',

message: 'Invalid token'

}

});

}

// Get user profile

const { data: profile, error: profileError } = await supabase

.from('profiles')

.select('\*')

.eq('id', user.id)

.single();

if (profileError || !profile) {

return res.status(401).json({

success: false,

error: {

code: 'AUTHENTICATION\_ERROR',

message: 'User profile not found'

}

});

}

// Check if user is active

if (profile.status !== 'active') {

return res.status(401).json({

success: false,

error: {

code: 'AUTHENTICATION\_ERROR',

message: 'User account is not active'

}

});

}

// Set user data on request object

req.user = {

id: user.id,

email: user.email,

firstName: profile.firstName,

lastName: profile.lastName,

role: profile.role,

permissions: decoded.permissions

};

next();

} catch (error) {

if (error.name === 'JsonWebTokenError') {

return res.status(401).json({

success: false,

error: {

code: 'AUTHENTICATION\_ERROR',

message: 'Invalid token'

}

});

}

if (error.name === 'TokenExpiredError') {

return res.status(401).json({

success: false,

error: {

code: 'AUTHENTICATION\_ERROR',

message: 'Token expired'

}

});

}

console.error('Authentication error:', error);

return res.status(500).json({

success: false,

error: {

code: 'SERVER\_ERROR',

message: 'Internal server error'

}

});

}

};

exports.authorize = (requiredPermissions = []) => {

return (req, res, next) => {

// Check if user exists on request (set by authenticate middleware)

if (!req.user) {

return res.status(401).json({

success: false,

error: {

code: 'AUTHENTICATION\_ERROR',

message: 'Authentication required'

}

});

}

// Admin role has all permissions

if (req.user.role === 'admin') {

return next();

}

// Check if user has required permissions

const hasPermission = requiredPermissions.every(permission =>

req.user.permissions.includes(permission)

);

if (!hasPermission) {

return res.status(403).json({

success: false,

error: {

code: 'AUTHORIZATION\_ERROR',

message: 'Insufficient permissions'

}

});

}

next();

};

};

**4. Auth Routes**

// server/routes/authRoutes.js

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

const { authenticate } = require('../middleware/auth');

// Public routes

router.post('/login', authController.login);

router.post('/refresh', authController.refresh);

router.post('/forgot-password', authController.forgotPassword);

router.post('/reset-password', authController.resetPassword);

// Protected routes

router.post('/register', authenticate, authController.register);

router.post('/logout', authenticate, authController.logout);

module.exports = router;

**Database Schema for Authentication**

**1. Profiles Table**

CREATE TABLE profiles (

id UUID PRIMARY KEY REFERENCES auth.users(id),

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE,

role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'staff', 'agent', 'student')),

status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('active', 'inactive', 'suspended')),

phone VARCHAR(20),

avatar\_url VARCHAR(255),

last\_login TIMESTAMP WITH TIME ZONE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Enable Row-Level Security

ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;

-- RLS Policies

CREATE POLICY "Users can view their own profile"

ON profiles FOR SELECT

USING (auth.uid() = id);

CREATE POLICY "Users can update their own profile"

ON profiles FOR UPDATE

USING (auth.uid() = id);

CREATE POLICY "Admins can view all profiles"

ON profiles FOR SELECT

USING (

EXISTS (

SELECT 1 FROM profiles WHERE id = auth.uid() AND role = 'admin'

)

);

CREATE POLICY "Admins can update all profiles"

ON profiles FOR UPDATE

USING (

EXISTS (

SELECT 1 FROM profiles WHERE id = auth.uid() AND role = 'admin'

)

);

CREATE POLICY "Staff can view student and agent profiles"

ON profiles FOR SELECT

USING (

EXISTS (

SELECT 1 FROM profiles WHERE id = auth.uid() AND role = 'staff'

) AND role IN ('student', 'agent')

);

-- Triggers

CREATE TRIGGER update\_profiles\_updated\_at

BEFORE UPDATE ON profiles

FOR EACH ROW

EXECUTE FUNCTION update\_updated\_at();

**2. Refresh Tokens Table**

CREATE TABLE refresh\_tokens (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,

token TEXT NOT NULL UNIQUE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

expires\_at TIMESTAMP WITH TIME ZONE NOT NULL,

is\_revoked BOOLEAN DEFAULT FALSE,

revoked\_at TIMESTAMP WITH TIME ZONE,

revocation\_reason TEXT

);

-- Indexes

CREATE INDEX idx\_refresh\_tokens\_user\_id ON refresh\_tokens(user\_id);

CREATE INDEX idx\_refresh\_tokens\_token ON refresh\_tokens(token);

CREATE INDEX idx\_refresh\_tokens\_is\_revoked ON refresh\_tokens(is\_revoked);

-- Enable Row-Level Security

ALTER TABLE refresh\_tokens ENABLE ROW LEVEL SECURITY;

-- RLS Policies

CREATE POLICY "No direct access to refresh tokens"

ON refresh\_tokens

USING (false);

**Role-Based Access Control (RBAC)**

**1. Permission Structure**

| **Role** | **Permissions** |
| --- | --- |
| **Admin** | read:all, write:all, delete:all, manage:users, manage:agents, manage:universities, manage:reports |
| **Staff** | read:all, write:applications, write:students, write:documents, verify:documents, process:applications |
| **Agent** | read:own\_students, read:own\_applications, write:own\_students, write:own\_applications, upload:documents |
| **Student** | read:own\_profile, read:own\_applications, write:own\_profile, write:own\_applications, upload:own\_documents |

**2. Permission Checking Hook**

// src/hooks/usePermissions.js

import { useSelector } from 'react-redux';

import { selectUserPermissions } from '../store/slices/authSlice';

export const usePermissions = () => {

const permissions = useSelector(selectUserPermissions);

const hasPermission = (requiredPermission) => {

return permissions.includes(requiredPermission);

};

const hasAnyPermission = (requiredPermissions) => {

return requiredPermissions.some(permission => permissions.includes(permission));

};

const hasAllPermissions = (requiredPermissions) => {

return requiredPermissions.every(permission => permissions.includes(permission));

};

return {

permissions,

hasPermission,

hasAnyPermission,

hasAllPermissions

};

};

**3. Permission Guard Component**

// src/components/auth/PermissionGuard.js

import React from 'react';

import { usePermissions } from '../../hooks/usePermissions';

const PermissionGuard = ({

permissions = [],

requireAll = true,

fallback = null,

children

}) => {

const { hasAllPermissions, hasAnyPermission } = usePermissions();

const hasPermission = requireAll

? hasAllPermissions(permissions)

: hasAnyPermission(permissions);

if (!hasPermission) {

return fallback;

}

return children;

};

export default PermissionGuard;

**Multi-Factor Authentication (MFA)**

**1. MFA Setup Component**

// src/components/auth/MfaSetup.js

import React, { useState, useEffect } from 'react';

import { useAuth } from '../../contexts/AuthContext';

import AuthService from '../../services/authService';

import QRCode from 'react-qr-code';

import { Button } from '../ui/Button';

import { Input } from '../ui/Input';

import { Alert } from '../ui/Alert';

const MfaSetup = () => {

const { user } = useAuth();

const [qrCode, setQrCode] = useState(null);

const [secret, setSecret] = useState(null);

const [verificationCode, setVerificationCode] = useState('');

const [isLoading, setIsLoading] = useState(true);

const [error, setError] = useState(null);

const [success, setSuccess] = useState(false);

useEffect(() => {

const generateMfaSecret = async () => {

try {

setIsLoading(true);

const response = await AuthService.generateMfaSecret();

setQrCode(response.data.qrCode);

setSecret(response.data.secret);

} catch (err) {

setError('Failed to generate MFA secret');

} finally {

setIsLoading(false);

}

};

generateMfaSecret();

}, []);

const handleVerify = async () => {

try {

setIsLoading(true);

setError(null);

const response = await AuthService.verifyAndEnableMfa(verificationCode, secret);

if (response.data.success) {

setSuccess(true);

} else {

setError('Invalid verification code');

}

} catch (err) {

setError('Failed to verify MFA code');

} finally {

setIsLoading(false);

}

};

if (isLoading && !qrCode) {

return <div>Loading MFA setup...</div>;

}

if (success) {

return (

<Alert variant="success" title="MFA Enabled">

Two-factor authentication has been successfully enabled for your account.

</Alert>

);

}

return (

<div className="space-y-6">

<div className="text-center">

<h3 className="text-lg font-medium">Setup Two-Factor Authentication</h3>

<p className="mt-1 text-sm text-gray-500">

Scan the QR code with an authenticator app like Google Authenticator,

Microsoft Authenticator, or Authy.

</p>

</div>

{error && <Alert variant="error">{error}</Alert>}

<div className="flex justify-center">

{qrCode && <QRCode value={qrCode} size={200} />}

</div>

<div className="text-center">

<p className="text-sm font-medium">Manual Entry Code:</p>

<p className="font-mono text-sm">{secret}</p>

</div>

<div>

<label htmlFor="verificationCode" className="block text-sm font-medium text-gray-700">

Verification Code

</label>

<Input

id="verificationCode"

type="text"

className="mt-1"

placeholder="Enter 6-digit code"

value={verificationCode}

onChange={(e) => setVerificationCode(e.target.value)}

maxLength={6}

/>

</div>

<Button

type="button"

className="w-full"

onClick={handleVerify}

disabled={isLoading || verificationCode.length !== 6}

>

{isLoading ? 'Verifying...' : 'Verify and Enable'}

</Button>

</div>

);

};

export default MfaSetup;

**2. MFA Verification Component**

// src/components/auth/MfaVerification.js

import React, { useState } from 'react';

import { useAuth } from '../../contexts/AuthContext';

import AuthService from '../../services/authService';

import { Button } from '../ui/Button';

import { Input } from '../ui/Input';

import { Alert } from '../ui/Alert';

const MfaVerification = ({ email, password, onSuccess, onCancel }) => {

const [verificationCode, setVerificationCode] = useState('');

const [isLoading, setIsLoading] = useState(false);

const [error, setError] = useState(null);

const handleVerify = async () => {

try {

setIsLoading(true);

setError(null);

const response = await AuthService.verifyMfa({

email,

password,

code: verificationCode

});

onSuccess(response.data);

} catch (err) {

setError(err.response?.data?.message || 'Invalid verification code');

} finally {

setIsLoading(false);

}

};

return (

<div className="space-y-6">

<div className="text-center">

<h3 className="text-lg font-medium">Two-Factor Authentication</h3>

<p className="mt-1 text-sm text-gray-500">

Enter the verification code from your authenticator app

</p>

</div>

{error && <Alert variant="error">{error}</Alert>}

<div>

<label htmlFor="verificationCode" className="block text-sm font-medium text-gray-700">

Verification Code

</label>

<Input

id="verificationCode"

type="text"

className="mt-1"

placeholder="Enter 6-digit code"

value={verificationCode}

onChange={(e) => setVerificationCode(e.target.value)}

maxLength={6}

autoFocus

/>

</div>

<div className="flex space-x-3">

<Button

type="button"

variant="outline"

className="w-full"

onClick={onCancel}

>

Cancel

</Button>

<Button

type="button"

className="w-full"

onClick={handleVerify}

disabled={isLoading || verificationCode.length !== 6}

>

{isLoading ? 'Verifying...' : 'Verify'}

</Button>

</div>

</div>

);

};

export default MfaVerification;

**Security Considerations**

**1. Password Requirements**

// src/utils/passwordValidator.js

export const validatePassword = (password) => {

const errors = [];

// Minimum length

if (password.length < 8) {

errors.push('Password must be at least 8 characters long');

}

// Must contain at least one uppercase letter

if (!/[A-Z]/.test(password)) {

errors.push('Password must contain at least one uppercase letter');

}

// Must contain at least one lowercase letter

if (!/[a-z]/.test(password)) {

errors.push('Password must contain at least one lowercase letter');

}

// Must contain at least one number

if (!/\d/.test(password)) {

errors.push('Password must contain at least one number');

}

// Must contain at least one special character

if (!/[!@#$%^&\*(),.?":{}|<>]/.test(password)) {

errors.push('Password must contain at least one special character');

}

return {

isValid: errors.length === 0,

errors

};

};

export const passwordStrength = (password) => {

if (!password) {

return {

score: 0,

label: 'Very Weak',

color: 'bg-red-500'

};

}

let score = 0;

// Length

if (password.length >= 8) score += 1;

if (password.length >= 12) score += 1;

// Character types

if (/[A-Z]/.test(password)) score += 1;

if (/[a-z]/.test(password)) score += 1;

if (/\d/.test(password)) score += 1;

if (/[!@#$%^&\*(),.?":{}|<>]/.test(password)) score += 1;

// Variety

const uniqueChars = new Set(password).size;

if (uniqueChars >= password.length \* 0.7) score += 1;

// Map score to strength

let label, color;

switch (true) {

case (score <= 2):

label = 'Very Weak';

color = 'bg-red-500';

break;

case (score <= 4):

label = 'Weak';

color = 'bg-orange-500';

break;

case (score <= 6):

label = 'Moderate';

color = 'bg-yellow-500';

break;

case (score <= 8):

label = 'Strong';

color = 'bg-green-500';

break;

default:

label = 'Very Strong';

color = 'bg-green-700';

}

return {

score,

label,

color

};

};

**2. CSRF Protection**

// server/middleware/csrfProtection.js

const csrf = require('csurf');

const cookieParser = require('cookie-parser');

// Configure CSRF protection

const csrfProtection = csrf({

cookie: {

httpOnly: true,

secure: process.env.NODE\_ENV === 'production',

sameSite: 'strict'

}

});

// Middleware setup

const setupCsrfProtection = (app) => {

// Parse cookies first

app.use(cookieParser());

// Apply CSRF protection to all routes except API

app.use((req, res, next) => {

// Skip CSRF for API routes

if (req.path.startsWith('/api/')) {

next();

} else {

csrfProtection(req, res, next);

}

});

// Handle CSRF errors

app.use((err, req, res, next) => {

if (err.code === 'EBADCSRFTOKEN') {

return res.status(403).json({

success: false,

error: {

code: 'CSRF\_ERROR',

message: 'Invalid or missing CSRF token'

}

});

}

next(err);

});

// Provide CSRF token to client

app.get('/api/csrf-token', (req, res) => {

res.json({

success: true,

data: {

csrfToken: req.csrfToken()

}

});

});

};

module.exports = setupCsrfProtection;

**3. Rate Limiting**

// server/middleware/rateLimiter.js

const rateLimit = require('express-rate-limit');

const RedisStore = require('rate-limit-redis');

const redis = require('../config/redis');

// Create rate limiters

const authLimiter = rateLimit({

store: new RedisStore({

client: redis,

prefix: 'rl:auth:'

}),

windowMs: 15 \* 60 \* 1000, // 15 minutes

max: 10, // 10 requests per windowMs

message: {

success: false,

error: {

code: 'RATE\_LIMIT\_EXCEEDED',

message: 'Too many login attempts. Please try again later.'

}

},

standardHeaders: true,

legacyHeaders: false

});

const apiLimiter = rateLimit({

store: new RedisStore({

client: redis,

prefix: 'rl:api:'

}),

windowMs: 60 \* 1000, // 1 minute

max: 100, // 100 requests per minute

message: {

success: false,

error: {

code: 'RATE\_LIMIT\_EXCEEDED',

message: 'Too many requests. Please try again later.'

}

},

standardHeaders: true,

legacyHeaders: false

});

// Apply rate limiters

const setupRateLimiters = (app) => {

// Apply stricter rate limiting to auth endpoints

app.use('/api/auth/login', authLimiter);

app.use('/api/auth/forgot-password', authLimiter);

app.use('/api/auth/reset-password', authLimiter);

// Apply general API rate limiting

app.use('/api', apiLimiter);

};

module.exports = setupRateLimiters;

**Authentication Flow for the Admin Portal**

**1. Overall Authentication Flow**

1. User navigates to login page

2. User enters email and password

3. Client sends credentials to server

4. Server validates credentials with Supabase

5. If valid, server generates access and refresh tokens

6. Server sends tokens and user data to client

7. Client stores tokens in local storage

8. Client updates auth state with user information

9. Client redirects to appropriate dashboard based on role

**2. Token Refresh Flow**

1. Client makes request with access token

2. Server rejects request with 401 if token is expired

3. Client intercepts 401 response

4. Client sends refresh token to server

5. Server validates refresh token

6. If valid, server generates new access and refresh tokens

7. Server sends new tokens to client

8. Client updates tokens in local storage

9. Client retries original request with new access token

**3. Protected Route Access Flow**

1. User navigates to protected route

2. ProtectedRoute component checks authentication state

3. If not authenticated, redirects to login page

4. If authenticated, checks for required role/permissions

5. If authorized, renders requested component

6. If not authorized, redirects to unauthorized page

**Conclusion**

This authentication flow provides a comprehensive foundation for securing the InterEd Admin Portal. It implements:

1. **JWT-based authentication** with access and refresh tokens
2. **Role-based access control** with fine-grained permissions
3. **Secure token storage and transmission**
4. **Protection against common security vulnerabilities**
5. **Multi-factor authentication** support
6. **Password policies and strength validation**

This implementation can be completed at approximately 70-75% before the Vibathon, providing a solid security foundation for the application while leaving room for integration and refinement during the 48-hour development sprint.

The authentication system is designed to be scalable, maintainable, and secure, meeting industry best practices for web application security while providing a smooth user experience across different roles and access levels.

**InterEd Admin Portal Authentication Flow**

**Overview**

This document outlines the comprehensive authentication system for the InterEd Student Recruitment Admin Portal. The authentication flow is designed to be secure, scalable, and user-friendly, supporting multiple user types with different permission levels. This authentication system can be implemented at approximately 70-75% completion before the Vibathon.

**Authentication Architecture**

The InterEd Admin Portal uses a JWT (JSON Web Token) based authentication system with the following components:

1. **JWT Access Tokens**: Short-lived tokens for API access
2. **Refresh Tokens**: Long-lived tokens to obtain new access tokens
3. **Role-Based Access Control (RBAC)**: Permissions based on user roles
4. **Supabase Auth Integration**: Backend authentication provider

**Authentication Flow Diagram**

┌───────────────┐ ┌───────────────┐ ┌───────────────┐

│ Client App │ │ Auth API │ │ Supabase │

└───────┬───────┘ └───────┬───────┘ └───────┬───────┘

│ │ │

│ Login Request │ │

│─────────────────────>│ │

│ │ │

│ │ Verify Credentials │

│ │─────────────────────>│

│ │ │

│ │ Auth Confirmation │

│ │<─────────────────────│

│ │ │

│ JWT + Refresh Token │ │

│<─────────────────────│ │

│ │ │

┌───────┴───────┐ ┌───────┴───────┐ ┌───────┴───────┐

│ Client App │ │ Protected API│ │ Supabase │

└───────┬───────┘ └───────┬───────┘ └───────┬───────┘

│ │ │

│ API Request + JWT │ │

│─────────────────────>│ │

│ │ Validate Token │

│ │─────────────────────>│

│ │ │

│ │ Token Valid/Invalid │

│ │<─────────────────────│

│ │ │

│ API Response │ │

│<─────────────────────│ │

│ │ │

┌───────┴───────┐ ┌───────┴───────┐ ┌───────┴───────┐

│ Client App │ │ Auth API │ │ Supabase │

└───────┬───────┘ └───────┬───────┘ └───────┬───────┘

│ │ │

│ Token Expired │ │

│ │ │

│ Refresh Request │ │

│─────────────────────>│ │

│ │ Validate Refresh │

│ │─────────────────────>│

│ │ │

│ │ Refresh Valid │

│ │<─────────────────────│

│ │ │

│ New JWT + Refresh │ │

│<─────────────────────│ │

│ │ │

**Authentication Endpoints**

**1. Login Endpoint**

**Endpoint**: POST /api/auth/login

**Request**:

{

"email": "user@example.com",

"password": "securePassword"

}

**Response**:

{

"success": true,

"data": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"expiresIn": 3600,

"user": {

"id": "user\_id",

"email": "user@example.com",

"firstName": "John",

"lastName": "Doe",

"role": "admin",

"permissions": ["read:all", "write:all", "delete:all"]

}

}

}

**2. Refresh Token Endpoint**

**Endpoint**: POST /api/auth/refresh

**Request**:

{

"refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

}

**Response**:

{

"success": true,

"data": {

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"expiresIn": 3600

}

}

**3. Register User Endpoint (Admin Only)**

**Endpoint**: POST /api/auth/register

**Request**:

{

"email": "newuser@example.com",

"password": "securePassword",

"firstName": "Jane",

"lastName": "Smith",

"role": "staff"

}

**Response**:

{

"success": true,

"data": {

"user": {

"id": "user\_id",

"email": "newuser@example.com",

"firstName": "Jane",

"lastName": "Smith",

"role": "staff"

}

}

}

**4. Password Reset Request Endpoint**

**Endpoint**: POST /api/auth/forgot-password

**Request**:

{

"email": "user@example.com"

}

**Response**:

{

"success": true,

"message": "If an account with that email exists, a password reset link has been sent"

}

**5. Password Reset Endpoint**

**Endpoint**: POST /api/auth/reset-password

**Request**:

{

"token": "reset\_token\_from\_email",

"password": "newSecurePassword"

}

**Response**:

{

"success": true,

"message": "Password has been reset successfully"

}

**6. Logout Endpoint**

**Endpoint**: POST /api/auth/logout

**Request**: JWT token in Authorization header

**Response**:

{

"success": true,

"message": "Logged out successfully"

}

**Token Structure**

**JWT Access Token**

{

"sub": "user\_id",

"email": "user@example.com",

"role": "admin",

"permissions": ["read:all", "write:all", "delete:all"],

"iat": 1613743939,

"exp": 1613747539,

"iss": "intered-admin-portal"

}

**Refresh Token**

{

"sub": "user\_id",

"jti": "unique\_token\_id",

"iat": 1613743939,

"exp": 1614353539,

"iss": "intered-admin-portal"

}

**Frontend Implementation**

**1. Authentication Context**

// src/contexts/AuthContext.js

import React, { createContext, useContext, useEffect, useState } from 'react';

import { useDispatch } from 'react-redux';

import { loginSuccess, logout } from '../store/slices/authSlice';

import AuthService from '../services/authService';

import { setAuthToken, removeAuthToken } from '../utils/axiosConfig';

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {

const dispatch = useDispatch();

const [loading, setLoading] = useState(true);

// Initialize auth state from local storage

useEffect(() => {

const initializeAuth = async () => {

const storedUser = localStorage.getItem('user');

const storedToken = localStorage.getItem('token');

const storedRefreshToken = localStorage.getItem('refreshToken');

const expiresAt = localStorage.getItem('expiresAt');

if (storedUser && storedToken && storedRefreshToken && expiresAt) {

const user = JSON.parse(storedUser);

// Check if token is expired

if (new Date().getTime() > expiresAt) {

try {

// Try to refresh the token

const response = await AuthService.refreshToken(storedRefreshToken);

const { token, refreshToken, expiresIn } = response.data;

// Calculate new expiration time

const newExpiresAt = new Date().getTime() + expiresIn \* 1000;

// Update localStorage

localStorage.setItem('token', token);

localStorage.setItem('refreshToken', refreshToken);

localStorage.setItem('expiresAt', newExpiresAt);

// Update auth state

setAuthToken(token);

dispatch(loginSuccess({

user,

token,

refreshToken,

expiresAt: newExpiresAt

}));

} catch (error) {

// Refresh token failed, log out

handleLogout();

}

} else {

// Token still valid, restore auth state

setAuthToken(storedToken);

dispatch(loginSuccess({

user,

token: storedToken,

refreshToken: storedRefreshToken,

expiresAt: parseInt(expiresAt)

}));

}

}

setLoading(false);

};

initializeAuth();

}, [dispatch]);

const handleLogin = async (credentials) => {

try {

const response = await AuthService.login(credentials);

const { user, token, refreshToken, expiresIn } = response.data;

// Calculate expiration time

const expiresAt = new Date().getTime() + expiresIn \* 1000;

// Save to localStorage

localStorage.setItem('user', JSON.stringify(user));

localStorage.setItem('token', token);

localStorage.setItem('refreshToken', refreshToken);

localStorage.setItem('expiresAt', expiresAt);

// Update auth state

setAuthToken(token);

dispatch(loginSuccess({

user,

token,

refreshToken,

expiresAt

}));

return user;

} catch (error) {

throw error;

}

};

const handleLogout = async () => {

try {

await AuthService.logout();

} catch (error) {

console.error('Logout error:', error);

} finally {

// Clear localStorage

localStorage.removeItem('user');

localStorage.removeItem('token');

localStorage.removeItem('refreshToken');

localStorage.removeItem('expiresAt');

// Update auth state

removeAuthToken();

dispatch(logout());

}

};

const value = {

login: handleLogin,

logout: handleLogout,

loading,

};

return (

<AuthContext.Provider value={value}>

{children}

</AuthContext.Provider>

);

};

export const useAuth = () => {

const context = useContext(AuthContext);

if (!context) {

throw new Error('useAuth must be used within an AuthProvider');

}

return context;

};

**2. Auth Service**

// src/services/authService.js

import axios from 'axios';

const API\_URL = process.env.REACT\_APP\_API\_URL || 'https://api.intered.com/v1';

const AuthService = {

login: async (credentials) => {

return axios.post(`${API\_URL}/auth/login`, credentials);

},

refreshToken: async (refreshToken) => {

return axios.post(`${API\_URL}/auth/refresh`, { refreshToken });

},

register: async (userData) => {

return axios.post(`${API\_URL}/auth/register`, userData);

},

forgotPassword: async (email) => {

return axios.post(`${API\_URL}/auth/forgot-password`, { email });

},

resetPassword: async (token, password) => {

return axios.post(`${API\_URL}/auth/reset-password`, { token, password });

},

logout: async () => {

return axios.post(`${API\_URL}/auth/logout`);

},

getCurrentUser: () => {

const user = localStorage.getItem('user');

return user ? JSON.parse(user) : null;

}

};

export default AuthService;

**3. Axios Configuration with Auth Interceptors**

// src/utils/axiosConfig.js

import axios from 'axios';

import store from '../store';

import { refreshTokenSuccess, logout } from '../store/slices/authSlice';

import AuthService from '../services/authService';

const API\_URL = process.env.REACT\_APP\_API\_URL || 'https://api.intered.com/v1';

// Create axios instance

const api = axios.create({

baseURL: API\_URL,

headers: {

'Content-Type': 'application/json',

},

});

// Set auth token for all requests

export const setAuthToken = (token) => {

if (token) {

api.defaults.headers.common['Authorization'] = `Bearer ${token}`;

} else {

delete api.defaults.headers.common['Authorization'];

}

};

// Remove auth token

export const removeAuthToken = () => {

delete api.defaults.headers.common['Authorization'];

};

// Response interceptor for handling token refresh

api.interceptors.response.use(

(response) => response,

async (error) => {

const originalRequest = error.config;

// If error is not 401 or request has already been retried, reject

if (error.response?.status !== 401 || originalRequest.\_retry) {

return Promise.reject(error);

}

originalRequest.\_retry = true;

try {

// Get current state

const state = store.getState();

const refreshToken = state.auth.refreshToken;

if (!refreshToken) {

// No refresh token available, logout

store.dispatch(logout());

return Promise.reject(error);

}

// Attempt to refresh token

const response = await AuthService.refreshToken(refreshToken);

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

// Calculate new expiration time

const expiresAt = new Date().getTime() + expiresIn \* 1000;

// Update tokens in store

store.dispatch(refreshTokenSuccess({

token,

refreshToken: newRefreshToken,

expiresAt

}));

// Update localStorage

localStorage.setItem('token', token);

localStorage.setItem('refreshToken', newRefreshToken);

localStorage.setItem('expiresAt', expiresAt);

// Update auth header

setAuthToken(token);

// Retry original request with new token

originalRequest.headers['Authorization'] = `Bearer ${token}`;

return api(originalRequest);

} catch (refreshError) {

// Refresh failed, logout

store.dispatch(logout());

return Promise.reject(refreshError);

}

}

);

export default api;

**4. Protected Route Component**

// src/components/auth/ProtectedRoute.js

import React from 'react';

import { Navigate, useLocation } from 'react-router-dom';

import { useSelector } from 'react-redux';

import { selectIsAuthenticated, selectUserRole } from '../../store/slices/authSlice';

import { useAuth } from '../../contexts/AuthContext';

import LoadingScreen from '../common/LoadingScreen';

const ProtectedRoute = ({

element,

allowedRoles = [],

redirectPath = '/login'

}) => {

const location = useLocation();

const { loading } = useAuth();

const isAuthenticated = useSelector(selectIsAuthenticated);

const userRole = useSelector(selectUserRole);

// Show loading screen while auth state is being initialized

if (loading) {

return <LoadingScreen />;

}

// Check if user is authenticated

if (!isAuthenticated) {

return <Navigate to={redirectPath} state={{ from: location }} replace />;

}

// Check if user has required role (if specified)

if (allowedRoles.length > 0 && !allowedRoles.includes(userRole)) {

return <Navigate to="/unauthorized" replace />;

}

// User is authenticated and authorized, render the component

return element;

};

export default ProtectedRoute;

**5. Login Form Component**

// src/components/auth/LoginForm.js

import React, { useState } from 'react';

import { useNavigate, useLocation } from 'react-router-dom';

import { useForm } from 'react-hook-form';

import { yupResolver } from '@hookform/resolvers/yup';

import \* as yup from 'yup';

import { useAuth } from '../../contexts/AuthContext';

import { Button } from '../ui/Button';

import { Input } from '../ui/Input';

import { Alert } from '../ui/Alert';

const schema = yup.object().shape({

email: yup.string().email('Invalid email address').required('Email is required'),

password: yup.string().required('Password is required'),

});

const LoginForm = () => {

const navigate = useNavigate();

const location = useLocation();

const { login } = useAuth();

const [error, setError] = useState(null);

const [isLoading, setIsLoading] = useState(false);

// Get redirect path from location state or default to dashboard

const from = location.state?.from?.pathname || '/dashboard';

const {

register,

handleSubmit,

formState: { errors },

} = useForm({

resolver: yupResolver(schema),

});

const onSubmit = async (data) => {

try {

setIsLoading(true);

setError(null);

// Call login function from auth context

const user = await login(data);

// Redirect based on user role

if (user.role === 'admin' || user.role === 'staff') {

navigate('/dashboard');

} else if (user.role === 'agent') {

navigate('/agent/dashboard');

} else {

navigate(from);

}

} catch (err) {

setError(err.response?.data?.message || 'Failed to log in. Please check your credentials.');

} finally {

setIsLoading(false);

}

};

return (

<form onSubmit={handleSubmit(onSubmit)} className="space-y-6">

{error && <Alert variant="error">{error}</Alert>}

<div>

<label htmlFor="email" className="block text-sm font-medium text-gray-700">

Email

</label>

<Input

id="email"

type="email"

className="mt-1"

error={errors.email?.message}

{...register('email')}

/>

</div>

<div>

<label htmlFor="password" className="block text-sm font-medium text-gray-700">

Password

</label>

<Input

id="password"

type="password"

className="mt-1"

error={errors.password?.message}

{...register('password')}

/>

<div className="text-sm mt-1">

<a href="/forgot-password" className="text-indigo-600 hover:text-indigo-500">

Forgot your password?

</a>

</div>

</div>

<Button type="submit" className="w-full" disabled={isLoading}>

{isLoading ? 'Logging in...' : 'Sign In'}

</Button>

</form>

);

};

export default LoginForm;

**Backend Implementation with Supabase**

**1. Supabase Auth Configuration**

// server/config/supabase.js

const { createClient } = require('@supabase/supabase-js');

const supabaseUrl = process.env.SUPABASE\_URL;

const supabaseServiceKey = process.env.SUPABASE\_SERVICE\_KEY;

// Create Supabase client with service key for admin operations

const supabase = createClient(supabaseUrl, supabaseServiceKey);

module.exports = supabase;

**2. Auth Controller**

// server/controllers/authController.js

const jwt = require('jsonwebtoken');

const supabase = require('../config/supabase');

const { v4: uuidv4 } = require('uuid');

// JWT configuration

const JWT\_SECRET = process.env.JWT\_SECRET;

const JWT\_REFRESH\_SECRET = process.env.JWT\_REFRESH\_SECRET;

const JWT\_EXPIRES\_IN = '1h';

const JWT\_REFRESH\_EXPIRES\_IN = '7d';

// Helper to generate tokens

const generateTokens = (user) => {

// Create access token

const token = jwt.sign(

{

sub: user.id,

email: user.email,

role: user.role,

permissions: user.permissions

},

JWT\_SECRET,

{

expiresIn: JWT\_EXPIRES\_IN,

issuer: 'intered-admin-portal'

}

);

// Create refresh token

const refreshToken = jwt.sign(

{

sub: user.id,

jti: uuidv4() // Add unique token ID

},

JWT\_REFRESH\_SECRET,

{

expiresIn: JWT\_REFRESH\_EXPIRES\_IN,

issuer: 'intered-admin-portal'

}

);

return {

token,

refreshToken,

expiresIn: 3600 // 1 hour in seconds

};

};

exports.login = async (req, res) => {

try {

const { email, password } = req.body;

// Authenticate with Supabase

const { data, error } = await supabase.auth.signInWithPassword({

email,

password

});

if (error) {

return res.status(401).json({

success: false,

error: {

message: 'Invalid credentials'

}

});

}

// Get user profile data

const { data: profile, error: profileError } = await supabase

.from('profiles')

.select('\*')

.eq('id', data.user.id)

.single();

if (profileError) {

return res.status(500).json({

success: false,

error: {

message: 'Error fetching user profile'

}

});

}

// Get user permissions based on role

const permissions = await getUserPermissions(profile.role);

// Create user object with profile data

const user = {

id: data.user.id,

email: data.user.email,

firstName: profile.firstName,

lastName: profile.lastName,

role: profile.role,

permissions

};

// Generate JWT and refresh token

const tokens = generateTokens(user);

// Store refresh token in database

await storeRefreshToken(user.id, tokens.refreshToken);

// Update last login timestamp

await supabase

.from('profiles')

.update({ lastLogin: new Date() })

.eq('id', user.id);

return res.status(200).json({

success: true,

data: {

user,

...tokens

}

});

} catch (error) {

console.error('Login error:', error);

return res.status(500).json({

success: false,

error: {

message: 'Internal server error'

}

});

}

};

exports.refresh = async (req, res) => {

try {

const { refreshToken } = req.body;

if (!refreshToken) {

return res.status(400).json({

success: false,

error: {

message: 'Refresh token is required'

}

});

}

// Verify refresh token

let decoded;

try {

decoded = jwt.verify(refreshToken, JWT\_REFRESH\_SECRET);

} catch (error) {

return res.status(401).json({

success: false,

error: {

message: 'Invalid or expired refresh token'

}

});

}

// Check if token exists in database and is not revoked

const tokenExists = await validateStoredRefreshToken(decoded.sub, refreshToken);

if (!tokenExists) {

return res.status(401).json({

success: false,

error: {

message: 'Invalid refresh token'

}

});

}

// Get user data

const { data: user, error: userError } = await supabase.auth.admin.getUserById(decoded.sub);

if (userError || !user) {

return res.status(401).json({

success: false,

error: {

message: 'User not found'

}

});

}

// Get user profile

const { data: profile, error: profileError } = await supabase

.from('profiles')

.select('\*')

.eq('id', user.id)

.single();

if (profileError) {

return res.status(500).json({

success: false,

error: {

message: 'Error fetching user profile'

}

});

}

// Get user permissions

const permissions = await getUserPermissions(profile.role);

// Create user object with profile data

const userData = {

id: user.id,

email: user.email,

firstName: profile.firstName,

lastName: profile.lastName,

role: profile.role,

permissions

};

// Generate new tokens

const tokens = generateTokens(userData);

// Invalidate old refresh token and store new one

await invalidateRefreshToken(decoded.sub, refreshToken);

await storeRefreshToken(userData.id, tokens.refreshToken);

return res.status(200).json({

success: true,

data: tokens

});

} catch (error) {

console.error('Refresh token error:', error);

return res.status(500).json({

success: false,

error: {

message: 'Internal server error'

}

});

}

};

exports.register = async (req, res) => {

try {

const { email, password, firstName, lastName, role } = req.body;

// Only admins can register new users

if (req.user.role !== 'admin') {

return res.status(403).json({

success: false,

error: {

message: 'Unauthorized'

}

});

}

// Validate role

const validRoles = ['admin', 'staff', 'agent'];

if (!validRoles.includes(role)) {

return res.status(400).json({

success: false,

error: {

message: 'Invalid role'

}

});

}

// Create user in Supabase Auth

const { data, error } = await supabase.auth.admin.createUser({

email,

password,

email\_confirm: true

});

if (error) {

return res.status(400).json({

success: false,

error: {

message: error.message

}

});

}

// Create user profile

const { error: profileError } = await supabase

.from('profiles')

.insert({

id: data.user.id,

firstName,

lastName,

email,

role,

status: 'active',

createdAt: new Date(),

updatedAt: new Date()

});

if (profileError) {

// Rollback user creation if profile creation fails

await supabase.auth.admin.deleteUser(data.user.id);

return res.status(500).json({

success: false,

error: {

message: 'Error creating user profile'

}

});

}

return res.status(201).json({

success: true,

data: {

user: {

id: data.user.id,

email,

firstName,

lastName,

role

}

}

});

} catch (error) {

console.error('Register error:', error);

return res.status(500).json({

success: false,

error: {

message: 'Internal server error'

}

});

}

};

exports.forgotPassword = async (req, res) => {

try {

const { email } = req.body;

// Send password reset email

await supabase.auth.resetPasswordForEmail(email, {

redirectTo: `${process.env.FRONTEND\_URL}/reset-password`

});

// Always return success to prevent email enumeration

return res.status(200).json({

success: true,

message: 'If an account with that email exists, a password reset link has been sent'

});

} catch (error) {

console.error('Forgot password error:', error);

// Still return success to prevent email enumeration

return res.status(200).json({

success: true,

message: 'If an account with that email exists, a password reset link has been sent'

});

}

};

exports.resetPassword = async (req, res) => {

try {

const { token, password } = req.body;

const { error } = await supabase.auth.verifyOtp({

token\_hash: token,

type: 'recovery'

});

if (error) {

return res.status(400).json({

success: false,

error: {

message: 'Invalid or expired token'

}

});

}

// Update password

const { error: updateError } = await supabase.auth.updateUser({

password

});

if (updateError) {

return res.status(400).json({

success: false,

error: {

message: updateError.message

}

});

}

return res.status(200).json({

success: true,

message: 'Password has been reset successfully'

});

} catch (