Airiam RAG Service Admin Dashboard

This guide provides instructions for integrating the responsive Admin Dashboard components with your existing Next.js application.

Overview

The Admin Dashboard is built with:

- Next.js App Router architecture
- React components with hooks for state management
- Redux for global state
- TailwindCSS for styling
- Full responsive design (mobile, tablet, desktop)
- Lazy loading for optimal performance
- Role-based access control
- Mobile-optimized touch interfaces

Directory Structure

The code is organized as follows:

Copy

```
/app                    # Next.js App Router pages
  /admin                # Admin-specific pages
/components             # React components
  /admin                # Admin-specific components
/lib                    # Utility functions and hooks
  /admin                # Admin-specific utilities
/redux                  # Redux state management
  /features/admin       # Admin-specific Redux slices
```

Integration Steps

1. Install Required Dependencies

bash

Copy

```
npm install @heroicons/react @headlessui/react recharts
```

2. Copy the Component Files

Copy the provided component files to their respective directories in your project.

3. Configure Redux

Ensure your existing Redux store includes the admin slices:

javascript

Copy

```
// store.js
import { configureStore } from '@reduxjs/toolkit';
import adminReducer from './features/admin/adminSlice';
import tenantReducer from './features/admin/tenantSlice';
import userReducer from './features/admin/userSlice';
import settingsReducer from './features/admin/settingsSlice';
import logsReducer from './features/admin/logsSlice';
import rolesReducer from './features/admin/rolesSlice';

export const store = configureStore({
  reducer: {
    // Your existing reducers
    admin: adminReducer,
    tenants: tenantReducer,
    users: userReducer,
    settings: settingsReducer,
    logs: logsReducer,
    roles: rolesReducer
  }
});
```

4. Set Up Routes

Create the necessary route files in your Next.js App Router structure:

javascript

Copy

```javascript
// app/admin/layout.jsx
import { Suspense } from 'react';
import { AdminLayout } from '@/components/admin/AdminLayout';

export default function Layout({ children }) {
  return (
    <AdminLayout>
      <Suspense fallback={<div>Loading...</div>}>
        {children}
      </Suspense>
    </AdminLayout>
  );
}

// app/admin/page.jsx
import AdminDashboard from '@/components/admin/dashboard/AdminDashboard';

export default function AdminPage() {
  return <AdminDashboard />;
}

// Add similar files for other routes
```

5. Configure Access Control

Set up middleware for role-based access control:

javascript

Copy

```javascript
// middleware.js
import { NextResponse } from 'next/server';

export function middleware(request) {
  // Check if the user is authenticated and has admin role
  // This is a simplified example - adapt to your auth system
  const session = request.cookies.get('session')?.value;
  const isAdmin = checkIfUserIsAdmin(session); // Your implementation

  if (request.nextUrl.pathname.startsWith('/admin') && !isAdmin) {
    return NextResponse.redirect(new URL('/login', request.url));
  }

  return NextResponse.next();
}

export const config = {
  matcher: ['/admin/:path*']
};
```

6. Implement Lazy Loading

For optimal performance, use dynamic imports for admin components:

javascript

Copy

```javascript
// app/admin/page.jsx
import dynamic from 'next/dynamic';

const AdminDashboard = dynamic(
  () => import('@/components/admin/dashboard/AdminDashboard'),
  {
    loading: () => <div>Loading dashboard...</div>,
    ssr: true
  }
);

export default function AdminPage() {
  return <AdminDashboard />;
}
```

Component Features

Admin Dashboard

- System status cards with key metrics
- Recent activity widget with real-time updates
- Quick action buttons for common tasks
- Resource metrics charts

Tenant Management

- Grid/list view toggle with responsive layouts
- Detailed tenant information cards
- Tenant creation and editing forms
- Theme customization with live preview

User Management

- Comprehensive user listing with search and filters
- User detail side panel (desktop) or full screen (mobile)
- User creation and editing forms
- Password reset workflow

Role Management

- Permission matrix for fine-grained access control
- Mobile-optimized permission toggles
- Predefined permission templates
- Category-based permission organization

System Settings

- Configuration panels for various system aspects
- Security settings with password policies
- Integration settings for external services
- Mobile-friendly form inputs

Monitoring

- System logs with advanced filtering

- Performance metrics with historical trends
- Health status dashboard with real-time updates
- Alert configuration for system events

Mobile Optimizations

The admin interface includes the following mobile-specific optimizations:

- Larger touch targets for better tap accuracy
- Fixed action buttons for easy access
- Swipeable interfaces where appropriate
- Collapsible sections to conserve screen space
- Responsive grid layouts that adapt to screen size
- Bottom sheet modals for touch-friendly interactions
- Mobile-specific navigation patterns

Theming

The components use TailwindCSS with dark mode support. You can customize the theme by modifying your tailwind.config.js:

javascript Copy

```javascript
// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: {
          DEFAULT: '#6366f1', // Your primary color
          light: '#818cf8',
          dark: '#4f46e5',
        },
        // Add other custom colors as needed
      },
    },
  },
  darkMode: 'class', // or 'media' for system preference
}
```

Testing

The admin components include comprehensive test scenarios in the design documents. Implement tests using your preferred testing framework (Jest, React Testing Library, Cypress).

Performance Considerations

- Use the built-in lazy loading mechanisms
- The ActionTracker is designed for minimal performance impact
- Mobile optimizations improve performance on lower-end devices
- Component memoization is used where appropriate

Further Customization

- Add tenant-specific branding to the ThemeCustomizer
- Extend the permissions matrix with your custom permissions
- Add additional analytics to the ActionTracker
- Customize form validation rules for your specific requirements

Troubleshooting

If you encounter issues:

1. Check browser console for errors
2. Verify Redux DevTools to ensure state is updating correctly
3. Check network requests for API failures
4. Ensure all dependencies are installed correctly
5. Verify mobile responsive behavior with browser dev tools