

Zandaris Frontend – Architecture

System Overview

The Zandaris frontend is a **Next.js 14** App Router application serving as the web interface for the Municipal Police Bratislava's internal system. It is the sole consumer of the [Zandaris NestJS backend](#).

Users access the system via: - **Desktop web browser** – full-featured interface with sidebar navigation - **Android wrapper app** – wraps the web app with native features (GPS, camera); detected via zandardroid flag

The frontend handles: event management (violations, towings, blockers), penalty ticket tracking, reference data (indexes) administration, statistical reporting, real-time patrol map, and user profile management.

High-Level Architecture

Routing Map

All authenticated routes use the (app) layout group which provides the Header, MenuNavigation sidebar, and Footer. Unauthenticated users are redirected to /login.

Main Pages

Route	Page	Legacy Component	Required Role
/	Home/Dashboard	Home.jsx	all
/events	Event list	Events.jsx	eventRead
/violations	Violation events	Violations.jsx	eventRead
/blockerevents	Blocker (wheel clamp) events	BlockerEventsPage.jsx	eventRead
/towing	Towing events	Towings.jsx	eventRead
/stats	Statistics	Statistics .jsx	statRead
/map	Patrol positions map	Maps.jsx	mapsRead
/myprofile	User profile & substitutions	MyProfile.jsx	all
/diag	Diagnostic messages	DiagMessages.jsx	sysadmin

Ticket Management

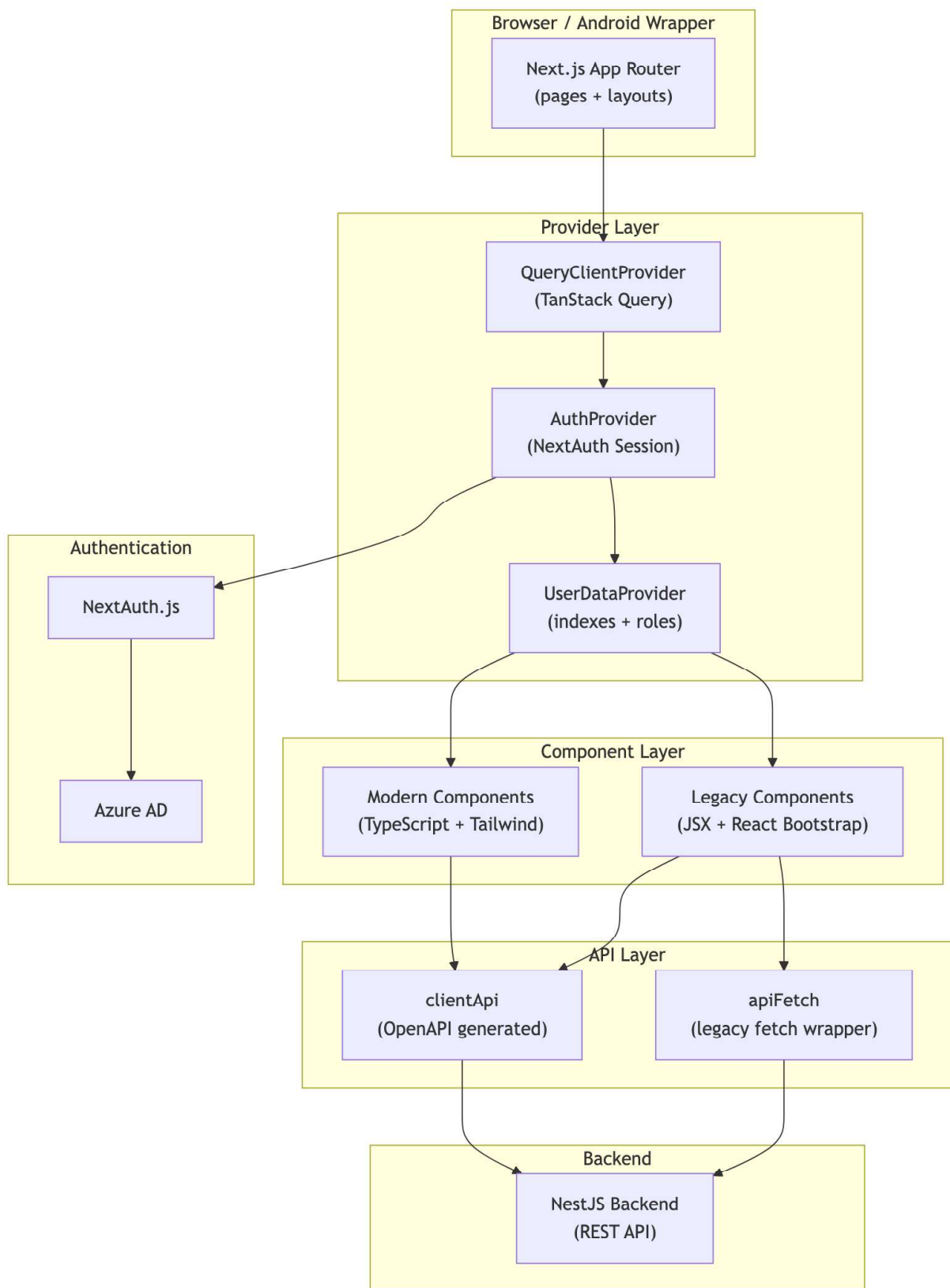


Figure 1: Diagram

Route	Page	Legacy Component	Required Role
/ticketbatch	Ticket batches (central)	TicketBatches.jsx	ticketRead
/ ticketlocal	Ticket batches (district)	Tickets.jsx	ticketLocalRead
/ticketelim	Eliminated tickets	TicketEliminations.jsx	ticketLocalRead
/ticketemp	Employee tickets & settlements	TicketsEmp.jsx	ticketLocalRead
/ticketterm	POS terminal tickets	TicketsTerm.jsx	ticketLocalRead

Dynamic Routes

Route Pattern	Config	Legacy Component	Required Role
/indexes/[path]	indexes/[path]/config	IndexBase.jsx	indexRead
/organization/[path]	organization/[path]/config	OrgBase.jsx	orgRead

Dynamic routes use a config object that maps the URL path to the backend API URL, list/edit components, and column definitions. This allows 20+ CRUD pages to share a single page component.

Other Routes

Route	Purpose
/login	Login page with Azure AD sign-in button
/event/[id]	Event detail (Android wrapper)
/auth / [...nextauth]	NextAuth API route handler

Layout Hierarchy

1. **RootLayout** (src/app/layout.tsx): Wraps everything with providers. Imports global CSS (Bootstrap, legacy styles, Tailwind).
2. **AppLayout** (src/app/(app)/layout.tsx): Checks authentication status, redirects to login if unauthenticated, renders the shell (Header, sidebar, Footer).
3. **Page**: Wraps a legacy component with PermissionRoleGuard for role-based access.

State Management

Provider Hierarchy

Providers are nested in this order (outermost first):

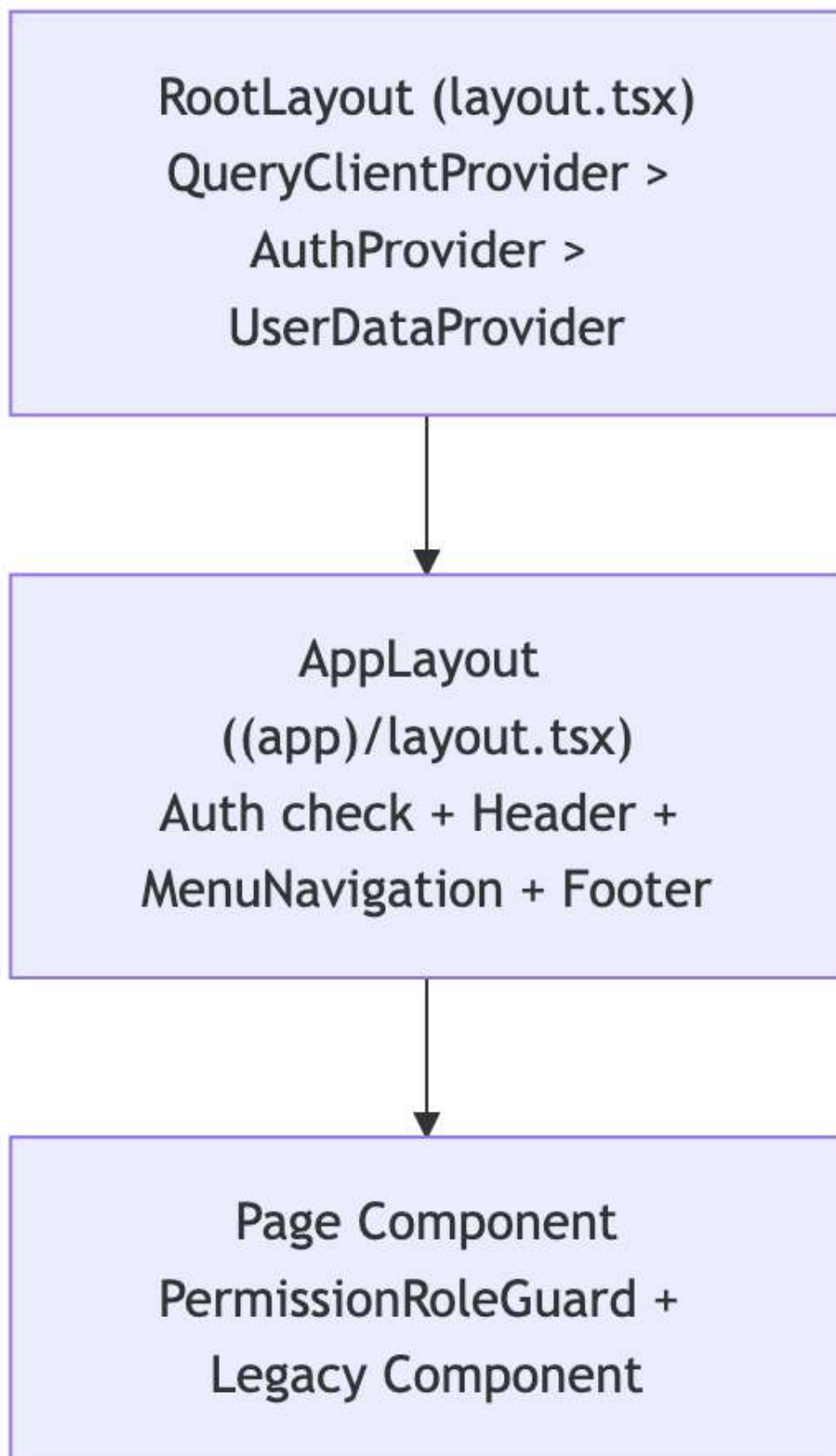


Figure 2: ₄Diagram

1. **QueryClientProvider** – TanStack Query client for server-state caching
2. **AuthProvider** – NextAuth session management
3. **UserDataProvider** – Loads all reference data (indexes) after authentication
4. **ExpandMenuProvider** – Sidebar menu accordion state (inside AppLayout)

Data Patterns

- **Server state:** TanStack Query (useQuery) for data fetched from the backend. Used in UserDataProvider and available for new code.
- **Global state:** React Context for user data (useUserData()) and menu state (useExpandMenu()).
- **Local state:** useState within components for forms, modals, and UI interactions.
- **Legacy pattern:** Legacy components fetch data directly via apiFetch() and manage state locally with useState.

Authentication Flow

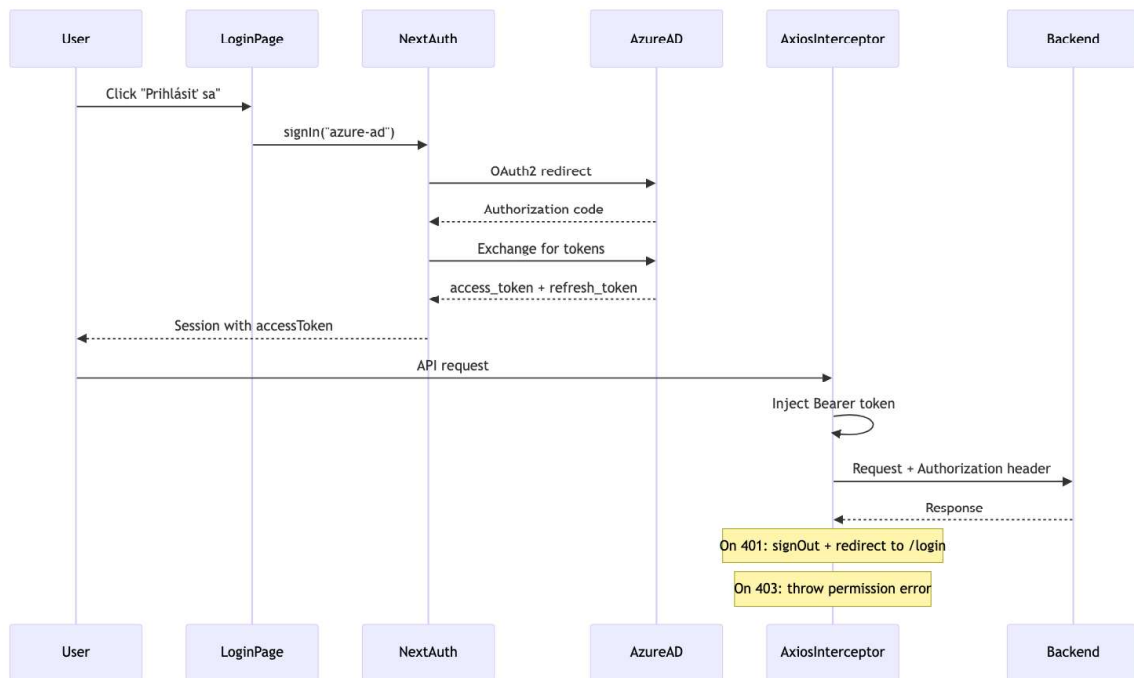


Figure 3: Diagram

- **Provider:** Azure AD via NextAuth
- **Token storage:** JWT session (server-side)
- **Token refresh:** Automatic via NextAuth JWT callback
- **Token injection:** Axios request interceptor in axios-instance.ts
- **Android bridge:** getToken() checks for zandardroid global before falling back to NextAuth session

API Consumption

Modern: clientApi (OpenAPI Generated)

```
import { clientApi } from '@services/clients/client-api'

const response = await clientApi.indexLoaderControllerGetIndexes()
```

- Auto-generated from the backend's Swagger spec (/api-json)
- Regenerate: `npm run generate-client`
- All 30+ API factory methods merged into a single `clientApi` object
- Uses `axiosInstance` with automatic Bearer token injection
- TypeScript types included for all request/response DTOs

Legacy: `apiFetch()`

```
import { apiFetch, fileUpload } from '@legacy/components/apiFetch'

const data = await apiFetch('api/events/basic', filterBody, 'POST')
await fileUpload('api/events/attach', formData)
```

- Custom `fetch()` wrapper with manual token handling
- Used by all legacy components
- Additional helpers: `fileUpload()`, `fileGet()`, `filePost()`
- Supports Android bridge token retrieval

When to Use Which

- **New code:** Always use `clientApi` – type-safe, consistent, auto-generated
- **Legacy code:** Uses `apiFetch` – being migrated incrementally
- **File operations:** `fileUpload()` / `fileGet()` from legacy (no `clientApi` equivalent yet)

Component Architecture

Base Component Pattern

Most pages are built on one of three “base” components that encapsulate the CRUD pattern:

IndexBase (`src/legacy/components/IndexBase.jsx`): - Generic CRUD base for all index and organization pages - Renders a list + Offcanvas edit panel - Props: API URL, List component, Edit component, column definitions - Used by all `/indexes/*` and `/organization/*` pages

EventBase (`src/legacy/components/Events/EventBase.jsx`): - Event list with fullscreen detail/edit modal - Advanced filtering (basic + advanced mode) - Export to Excel functionality - Used by: Events, Violations, Blockers, Towing pages

TicketBase (`src/legacy/components/Tickets/TicketBase.jsx`): - Ticket batch list with edit modal - Used by: TicketBatches, Tickets, TicketEmp, TicketTerm pages

Component Naming Conventions

- `*List.jsx` – Table/list view (e.g. `VehicleColorList.tsx`, `EventList.jsx`)
 - `*Edit.jsx` – Form/edit view (e.g. `VehicleColorEdit.jsx`, `TicketElimEdit.jsx`)
 - `*Page.jsx` – Full page combining list + edit (e.g. `EventGroupPage.jsx`)
-

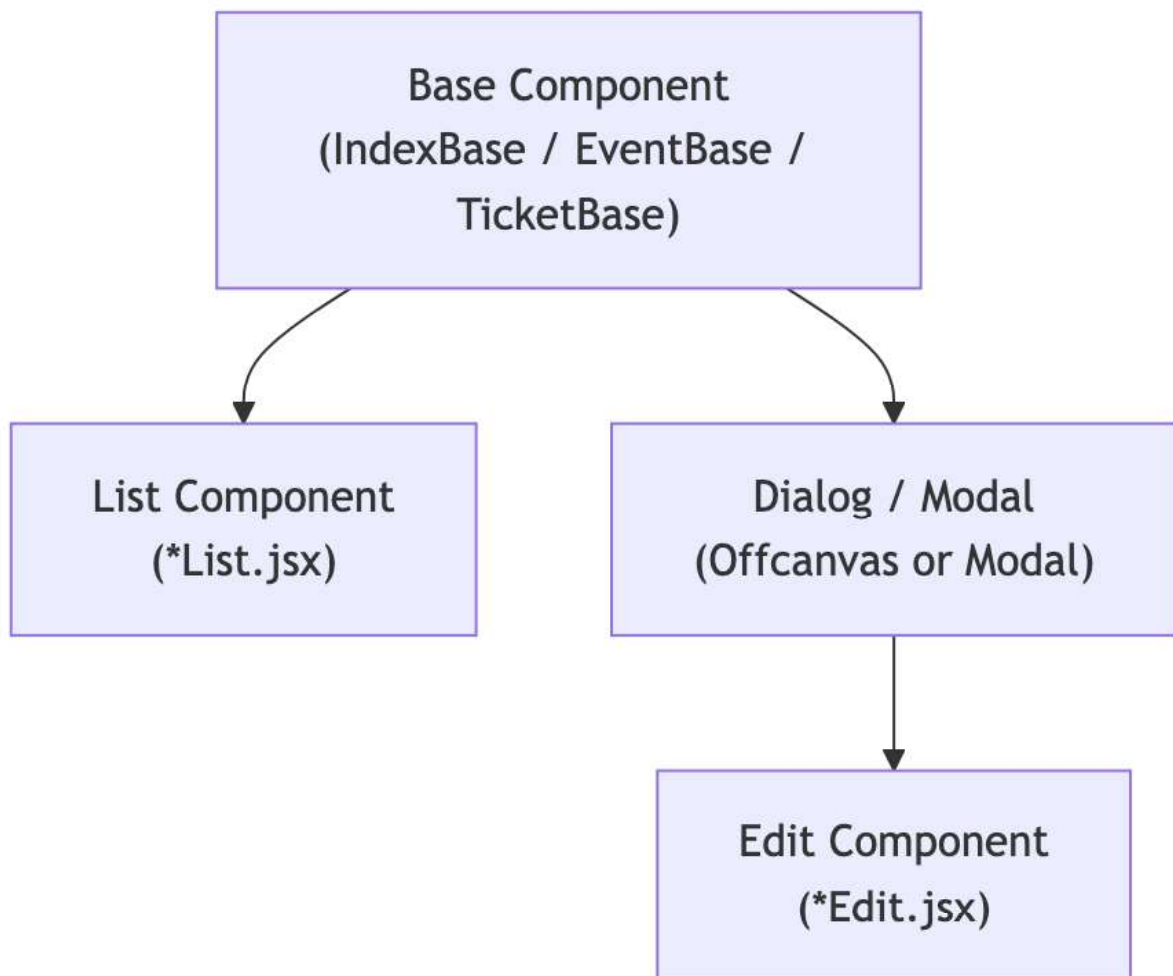


Figure 4: Diagram

Legacy Migration Status

The codebase is in a **hybrid state** between legacy (Pages Router era) and modern (App Router) code.

What Has Been Migrated

- Routing: Next.js App Router with layout groups
- Provider architecture: Modern React Context + TanStack Query
- Shared UI components: TypeScript + Tailwind (src/components/)
- API client: OpenAPI-generated TypeScript client
- Guards: PermissionRoleGuard component

What Remains Legacy

- All page content: JSX components in src/legacy/pages/
- All CRUD logic: IndexBase, EventBase, TicketBase in src/legacy/components/
- All form components: *Edit.jsx files
- All list components: *List.jsx files
- DTOs: 179 TypeScript files in src/legacy/dto/
- Styles: React Bootstrap + custom CSS (src/legacy/site.css)
- API calls: apiFetch() wrapper

Migration Pattern

Modern App Router pages are thin wrappers around legacy components:

```
// src/app/(app)/events/page.tsx
export default function Page() {
  return (
    <PermissionRoleGuard requiredRole={roles.eventRead}>
      <EventsPage /> { /* legacy component */ }
    </PermissionRoleGuard>
  )
}
```

Tailwind config explicitly **excludes** src/legacy/** to avoid conflicts with React Bootstrap styles.

Key Libraries

Library	Version	Purpose
Next.js	14.2	App Router, SSR, file-based routing
React	18.2	UI framework
NextAuth	4.24	Azure AD authentication
TanStack Query	5.76	Server state management, caching
Axios	1.9	HTTP client for API calls
React Bootstrap	2.9	UI components (legacy)
Tailwind CSS	3.4	Utility-first CSS (modern)
OpenLayers (rlayers)	9.1	Map rendering for patrol positions
Recharts	2.12	Statistical charts
FontAwesome	6.x	Icons

Library	Version	Purpose
react-datepicker	–	Date input fields

Deployment

- **Output mode:** Standalone (next.config.mjs: output: 'standalone')
 - **Container:** Dockerfile in project root
 - **Orchestration:** Kubernetes configs in kubernetes/envs/ (Dev, Staging, Prod)
 - **Environment variables:** NEXT_PUBLIC_API_URL, AZURE_AD_CLIENT_ID, AZURE_AD_CLIENT_SECRET, AZURE_AD_TENANT_ID, NEXTAUTH_SECRET
-

Related Documentation

- [Backend Architecture](#)
- Backend API reference: Swagger UI at /api on any running backend instance