NoTIP

NO TESTS IN PRODUCTION

Specifica tecnica - *Frontend*$_G$

Versione	1.1.0
Data Modifica	2026-04-17
Utilizzo	Esterno

Abstract dei contenuti

Specifica tecnica della WebApp *frontend*$_G$: architettura, gestione autenticazione, integrazione API tramite OpenAPI generator, streaming *SSE*$_G$ per *telemetria*$_G$ e gestione dell'impersonazione.

Changelog$_G$

Versione	Data	Autori	*Verificatore*$_G$	Descrizione
1.1.0	2026-04-17	Leonardo Preo	Alessandro Mazzariol	Aggiustamenti post meeting con Prof. Cardin: modifica descrizione architettura
1.0.0	2026-04-13	Leonardo Preo (responsabile)		Approvazione per ingresso in *baseline*$_G$ PB
0.3.0	2026-04-12	Leonardo Preo	Francesco Marcon	Revisione generale della specifica tecnica del *frontend*$_G$, con allineamento della struttura del documento e consolidamento delle sezioni descrittive
0.2.0	2026-04-10	Leonardo Preo	Alessandro Contarini	Integrazione delle informazioni su autenticazione, gestione degli errori e interazione con le API esposte dai servizi *backend*$_G$
0.1.0	2026-04-09	Leonardo Preo	Francesco Marcon	Prima stesura della specifica tecnica, con definizione dell'architettura generale e delle funzionalità principali della WebApp *frontend*$_G$
0.0.1	2026-04-05	Leonardo Preo	Alessandro Mazzariol	Creazione della prima bozza del documento e impostazione iniziale della specifica tecnica del *frontend*$_G$

Indice

Indice delle figure

Indice delle tabelle

1. Introduzione

La WebApp `notip-frontend`$_G$ è l'interfaccia utente del sistema NoTIP, sviluppata in *Angular*$_G$ 21 con architettura standalone. Il *frontend*$_G$ ha il compito di presentare all'utente i dati di *telemetria*$_G$ provenienti dai *gateway*$_G$ *IoT*$_G$, consentire la gestione di *tenant*$_G$, utenti, *gateway*$_G$ e sensori, configurare alert e soglie, consultare *audit*$_G$ log e costi, e inviare comandi ai *gateway*$_G$. Supporta tre ruoli utente (system_admin, tenant_admin, tenant_user) con autorizzazione basata su rotte e meccanismi di impersonazione che permettono ai system admin di operare nel contesto di un *tenant*$_G$.

Il *frontend*$_G$ comunica con due *backend*$_G$ distinti: `notip-management-api` per le operazioni di gestione e configurazione, e `notip-data-api` per le query e lo streaming di *telemetria*$_G$. L'autenticazione è gestita da *Keycloak*$_G$ tramite *PKCE*$_G$, con supporto al rinnovo automatico del token e all'impersonazione.

2. Dipendenze e configurazione

2.1. Variabili d'ambiente

La configurazione del *frontend*$_G$ avviene principalmente tramite `app.config.ts`. Non sono presenti variabili `.env` obbligatorie nel codice applicativo: i parametri runtime principali sono costanti di bootstrap e path del reverse proxy.

Campo	Variabile/Config	Default	Obbligatorio
KeycloakUrl	Configurato in `app.config.ts` come `/auth`	`/auth`	Si
KeycloakRealm	Configurato in `app.config.ts`	`notip`	Si
KeycloakClientId	Configurato in `app.config.ts`	`notip-frontend`$_G$	Si
SessionTimeout	Configurato in `app.config.ts` come `10 * 60 * 1000`	10 min	No
DataApiBasePath	`/api/data`	`/api/data`	Si
MgmtApiBasePath	`/api/mgmt`	`/api/mgmt`	Si

Tabella 1: Configurazione del *frontend*$_G$ `notip-frontend`$_G$

Il *frontend*$_G$ è servito attraverso un reverse proxy (Nginx) che instrada le chiamate ai path `/api/data`, `/api/mgmt` e `/auth` verso i rispettivi *backend*$_G$. Non esiste una directory `src/environments/`: la configurazione è interamente gestita in `app.config.ts` e i path del reverse proxy sono iniettati a livello di infrastruttura *Docker*$_G$.

2.2. Sequenza di avvio

La sequenza di inizializzazione dell'applicazione *Angular*$_G$ è la seguente:

Step	Componente	Azione	Bloccante?
0	Bootstrap *Angular*$_G$	Avvio del runtime *Angular*$_G$ e caricamento di `app.config.ts`	Si
1	KeycloakModule	Inizializza *Keycloak*$_G$ con `onLoad: 'login-required'` e *PKCE*$_G$ S256	Si

2	AutoRefreshTokenService	Configura il rinnovo automatico del token con timeout di 10 minuti	Si
3	HttpClientModule	Registra gli interceptor `authInterceptor` e `errorInterceptor`	Si
4	RouterModule	Configura le rotte con guardie (AuthGuard, RoleGuard, HomeRedirectGuard) e resolver (DashboardResolver)	Si
5	provideMgmtApi / provideDataApi	Registra i client OpenAPI generati con i base path configurati	Si
6	AuthService	Registra i provider per `SESSION_LIFECYCLE` e `IMPERSONATION_STATUS`	Si
7	AuthGuard	Al primo accesso a una rotta protetta: verifica inizializzazione *Keycloak*$_G$ e presenza token, avvia threshold prefetch	Si
8	RoleGuard	Verifica il ruolo dell'utente rispetto ai ruoli richiesti dalla rotta	Si
9	Browser rendering	Rendering del componente pagina risolto dalla route	No

Tabella 2: Sequenza di avvio dell'applicazione *frontend*$_G$

3. Architettura logica

L'applicazione adotta un'architettura **Model-View** (nello specifico Model-View-ViewModel), tipica dell'ecosistema *Angular*$_G$, implementata attraverso componenti standalone senza l'uso di NgModules. Questo paradigma architetturale separa nettamente la logica di business dalla presentazione, assegnando responsabilità precise:

- **Model:** Rappresenta i dati di dominio e la logica applicativa. È implementato attraverso i Service, le interfacce e le classi. Il Model è responsabile della gestione dei flussi di dati asincroni e della comunicazione con il *backend*$_G$ (es. chiamate HTTP e connessioni *SSE*$_G$), avvalendosi in modo massiccio della libreria RxJS.
- **View:** Costituisce l'interfaccia utente (*template*$_G$ HTML e fogli di stile CSS). È un layer puramente presentazionale, reattivo e completamente disaccoppiato dalla logica complessa di recupero o manipolazione dei dati.
- **ViewModel / Controller:** I componenti standalone agiscono da ponte (intermediari) tra il Model e la View. Interrogano i servizi per ottenere i dati e gestiscono lo stato locale avvalendosi dello state management basato sui segnali di *Angular*$_G$ (signal(), computed(), effect()). Questo permette di esporre alla View uno stato reattivo facilmente consumabile e di gestire l'interazione dell'utente (Subject e comunicazioni inter-componente).

A livello organizzativo e di file system, l'applicazione supporta questa architettura Model-View suddividendo il codice in tre aree orizzontali principali (Core, Features e Shared).

Questo garantisce che la logica di business (Model, contenuta soprattutto nel Core e nei servizi Feature) sia ben isolata dalla presentazione, mantenendo dipendenze unidirezionali e impedendo che la View conosca i dettagli di implementazione dei dati.

Lo state management è basato su **segnali *Angular*$_G$** (signal(), computed(), effect()) per lo stato locale dei componenti e dei servizi di feature, combinati con **RxJS** per la gestione di flussi asincroni (*SSE*$_G$, HTTP) e la comunicazione inter-componente tramite Subject.

3.1. Layout delle cartelle

```
notip-frontend_G/src/
├── app/
│   ├── core/
│   │   ├── auth/
│   │   │   └── contracts.ts          # Interfacce SESSION_LIFECYCLE, IMPERSONATION_STATUS
│   │   ├── guards/
│   │   │   ├── auth.guard.ts          # Guardia di autenticazione
│   │   │   ├── role.guard.ts          # Guardia basata sui ruoli
│   │   │   └── *.spec.ts
│   │   ├── interceptors/
│   │   │   ├── auth.interceptor.ts    # Iniezione bearer token_G
│   │   │   ├── error.interceptor.ts   # Gestione 401/403
│   │   │   └── *.spec.ts
│   │   ├── models/
│   │   │   ├── enums.ts               # UserRole, TenantStatus, GatewayStatus, ecc.
│   │   │   ├── gateway_G.ts            # Gateway_G, ObfuscatedGateway, AddGatewayParameters
│   │   │   ├── measure.ts             # TelemetryEnvelope, DecryptedEnvelope,
CheckedEnvelope
│   │   │   ├── sensor.ts              # Sensor
│   │   │   ├── tenant_G.ts             # Tenant_G, CreateTenantParameters
│   │   │   ├── user.ts                # ViewUser, CreatedUser, UserParameters
│   │   │   ├── alert.ts               # Alerts, AlertsConfig, AlertsFilter
│   │   │   ├── threshold.ts           # Threshold, SensorThreshold, TypeThreshold
│   │   │   ├── costs.ts               # Costs
│   │   │   ├── command.ts             # Command types
│   │   │   ├── client.ts              # API Client types
│   │   │   └── audit_G.ts              # Logs, LogsFilter
│   │   ├── resolvers/
│   │   │   └── dashboard_G.resolver.ts  # Risolve dataMode (clear/obfuscated)
│   │   └── services/
│   │       ├── auth.service.ts        # Gestione Keycloak_G, impersonazione, sessione
│   │       ├── threshold-prefetch.service.ts  # Prefetch periodico soglie
│   │       ├── threshold.service.ts   # Cache soglie sensori
│   │       ├── measure-bounds-evaluation.service.ts  # Valutazione bounds
│   │       └── obfuscated-stream-manager.service.ts # Gestione SSE_G
│   ├── features/
│   │   ├── admin/
│   │   │   ├── components/
│   │   │   │   ├── admin-gateway_G-form/    # Form creazione gateway_G admin
│   │   │   │   ├── admin-gateway_G-table/   # Tabella gateway_G admin
│   │   │   │   ├── impersonate-button/    # Bottone impersonazione
│   │   │   │   ├── tenant_G-form/           # Form creazione/modifica tenant_G
│   │   │   │   ├── tenant_G-table/          # Tabella tenant_G
│   │   │   │   └── tenant_G-user-list/      # Lista utenti tenant_G
│   │   │   ├── pages/
│   │   │   │   ├── tenant_G-manager/    # Gestione tenant_G (system_admin)
│   │   │   │   └── tenant_G-detail/     # Dettaglio tenant_G con utenti
│   │   │   └── pages/admin-gateway_G-list/  # Gateway_G admin
```

```
│   │   ├── alerts/
│   │   │   ├── components/
│   │   │   │   ├── alert-config-form/   # Form config timeout alert
│   │   │   │   └── alert-filter-panel/  # Pannello filtri alert
│   │   │   ├── pages/
│   │   │   │   ├── alert-list/        # Lista alert gateway_G offline
│   │   │   │   └── alert-config/      # Configurazione timeout alert
│   │   │   └── services/
│   │   │       └── alert.service.ts
│   │   ├── dashboard_G/
│   │   │   ├── pages/data-dashboard_G/  # Dashboard_G telemetria_G principale
│   │   │   ├── components/
│   │   │   │   ├── filter-panel/      # Pannello filtri multi-select
│   │   │   │   ├── telemetry-chart/   # Grafico Chart.js
│   │   │   │   └── telemetry-table/   # Tabella dati telemetria_G
│   │   │   └── services/
│   │   │       ├── decrypted-measure.service.ts      # Decrittografia SDK_G
│   │   │       ├── obfuscated-measure.service.ts     # Stream offuscato
│   │   │       └── validated-measure-facade.service.ts # Validazione bounds
│   │   ├── gateways/
│   │   │   ├── pages/
│   │   │   │   ├── gateway_G-list/      # Lista gateway_G tenant_G
│   │   │   │   └── gateway_G-detail/    # Dettaglio gateway_G con comandi
│   │   │   ├── components/
│   │   │   │   ├── gateway_G-card/
│   │   │   │   ├── gateway_G-actions/
│   │   │   │   ├── gateway_G-rename-modal/
│   │   │   │   └── command-modal/
│   │   │   └── services/
│   │   │       ├── gateway_G.service.ts
│   │   │       └── command.service.ts
│   │   ├── mgmt/
│   │   │   ├── api-clients/
│   │   │   │   ├── components/
│   │   │   │   │   └── api-client-table/    # Tabella client API
│   │   │   │   ├── pages/
│   │   │   │   │   └── api-client-list/     # Lista client API
│   │   │   │   └── services/
│   │   │   │       └── clients.service.ts
│   │   │   ├── audit_G/
│   │   │   │   ├── components/
│   │   │   │   │   ├── audit_G-filter-panel/  # Pannello filtri audit_G
│   │   │   │   │   └── audit_G-log-table/     # Tabella audit_G log
│   │   │   │   ├── pages/
│   │   │   │   │   └── audit_G-log/           # Pagina audit_G log
│   │   │   │   └── services/
│   │   │   │       └── audit_G.service.ts
│   │   │   ├── costs/
│   │   │   │   ├── components/
│   │   │   │   │   └── cost-card/           # Card costi tenant_G
│   │   │   │   ├── pages/
│   │   │   │   │   └── cost-dashboard_G/      # Dashboard_G costi
│   │   │   │   └── services/
│   │   │   │       └── costs.service.ts
│   │   │   ├── thresholds/
│   │   │   │   ├── components/
│   │   │   │   │   ├── threshold-form/      # Form soglie
│   │   │   │   │   └── threshold-table/     # Tabella soglie
```

```
│   │   │   │   └── pages/
│   │   │   │       └── threshold-settings/  # Configurazione soglie
│   │   │   └── users/
│   │   │       ├── components/
│   │   │       │   ├── user-form/           # Form utente
│   │   │       │   └── user-table/          # Tabella utenti
│   │   │       ├── pages/
│   │   │       │   └── user-list/           # Lista utenti tenant_G
│   │   │       └── services/
│   │   │           └── user.service.ts
│   │   ├── sensors/
│   │   │   ├── pages/
│   │   │   │   ├── sensor-list/       # Lista sensori
│   │   │   │   └── sensor-detail/     # Dettaglio sensore
│   │   │   └── services/
│   │   └── system/
│   │       └── pages/error/           # Pagina di errore generica
│   ├── generated/
│   │   └── openapi/
│   │       ├── notip-management-api-openapi/  # Client API management
│   │       └── notip-data-api-openapi/        # Client API data
│   └── shared/
│       ├── components/
│       │   ├── sidebar/               # Navigazione laterale role-based
│       │   ├── modal-layer/           # Overlay modale generico
│       │   ├── multi-select-dropdown/ # Dropdown multi-selezione con ricerca
│       │   ├── status-badge/          # Badge colorati per stato
│       │   ├── delete-confirm-modal/  # Modale conferma cancellazione
│       │   ├── impersonation-tag/     # Tag "OBFUSCATED MODE"
│       │   ├── logout-button/         # Bottone logout
│       │   └── profile-section/       # Sezione profilo utente
│       ├── pipes/
│       │   └── rome-date-time.pipe.ts  # Pipe per fuso orario Roma
│       └── utils/
│           └── rome-timezone.util.ts  # Utility conversione fuso orario
├── api-contracts/
│   └── openapi/
│       ├── notip-data-api-openapi.yaml_G
│       └── notip-management-api-openapi.yaml_G
├── scripts/
│   └── generate-openapi.sh
├── index.html
├── main_G.ts
└── styles.css
```

> **Nota:** Non esiste una directory `src/environments/`. La configurazione runtime è gestita interamente tramite `app.config.ts` e i path del reverse proxy Nginx (`/api/data`, `/api/mgmt`, `/auth`).

3.2. Strati architetturali

Strato	Package	Componenti	Responsabilità
Presentation	`src/app/features/*/pages/`, `src/app/features/*/components/`, `src/app/shared/components/`,	Componenti *Angular*$_G$ standalone, *Template*$_G$ HTML, CSS, Pipe custom	Rendering UI, gestione interazioni utente, composizione di componenti figli, binding con servizi tramite segnali e

	`src/app/shared/pipes/`		Observable. I componenti di pagina orchestrano la logica di feature, mentre i componenti shared sono riutilizzabili trasversalmente. Le pipe custom gestiscono la formattazione temporale con fuso orario Roma.
Feature Services	`src/app/features/*/services/`, `src/app/core/services/`	Servizi *Angular*$_G$ con stato locale (segnali), wrapper client API, gestione flussi di business logic di dominio	Astrazione sui client API generati, gestione stato locale con segnali (`signal()`, `asReadonly()`), orchestrazione chiamate HTTP, trasformazione *DTO*$_G$ in modelli di dominio, caching, polling per stato comandi. Servizi come `GatewayService`, `CommandService`, `DecryptedMeasureService` incapsulano la logica specifica di dominio.
Generated API Clients	`src/app/generated/openapi/notip-management-api-openapi/`, `src/app/generated/openapi/notip-data-api-openapi/`	Client *Angular*$_G$ auto-generati da OpenAPI Generator (`typescript`$_G$`-angular`$_G$)	Tipizzazione forte delle chiamate HTTP verso management-api e data-api. Generati automaticamente dagli specchi OpenAPI tramite script `generate-openapi.sh`. Forniscono classi come `GatewaysService`, `MeasuresService`, `AdminTenantsService`, ecc. con metodi tipizzati e gestione error integrata.
Core Infrastructure	`src/app/core/guards/`, `src/app/core/interceptors/`, `src/app/core/auth/`, `src/app/core/resolvers/`, `src/app/core/models/`	AuthGuard, RoleGuard, HomeRedirectGuard, authInterceptor, errorInterceptor, AuthService, DashboardResolver, modelli di dominio	Autenticazione *Keycloak*$_G$ con *PKCE*$_G$, iniezione automatica del *bearer token*$_G$, gestione errori HTTP (401 -> redirect /error, 403 -> forbidden), controllo accesso basato sui ruoli, risoluzione rotte condizionale, definizione contratti typed per l'intero dominio (*Gateway*$_G$, Sensor,

			Measure, *Tenant*$_G$, User, Alert, Threshold, Command, *Audit*$_G$, Cost).

Tabella 3: Strati architetturali del *frontend*$_G$ `notip-frontend`$_G$

3.3. Architettura dell'applicazione

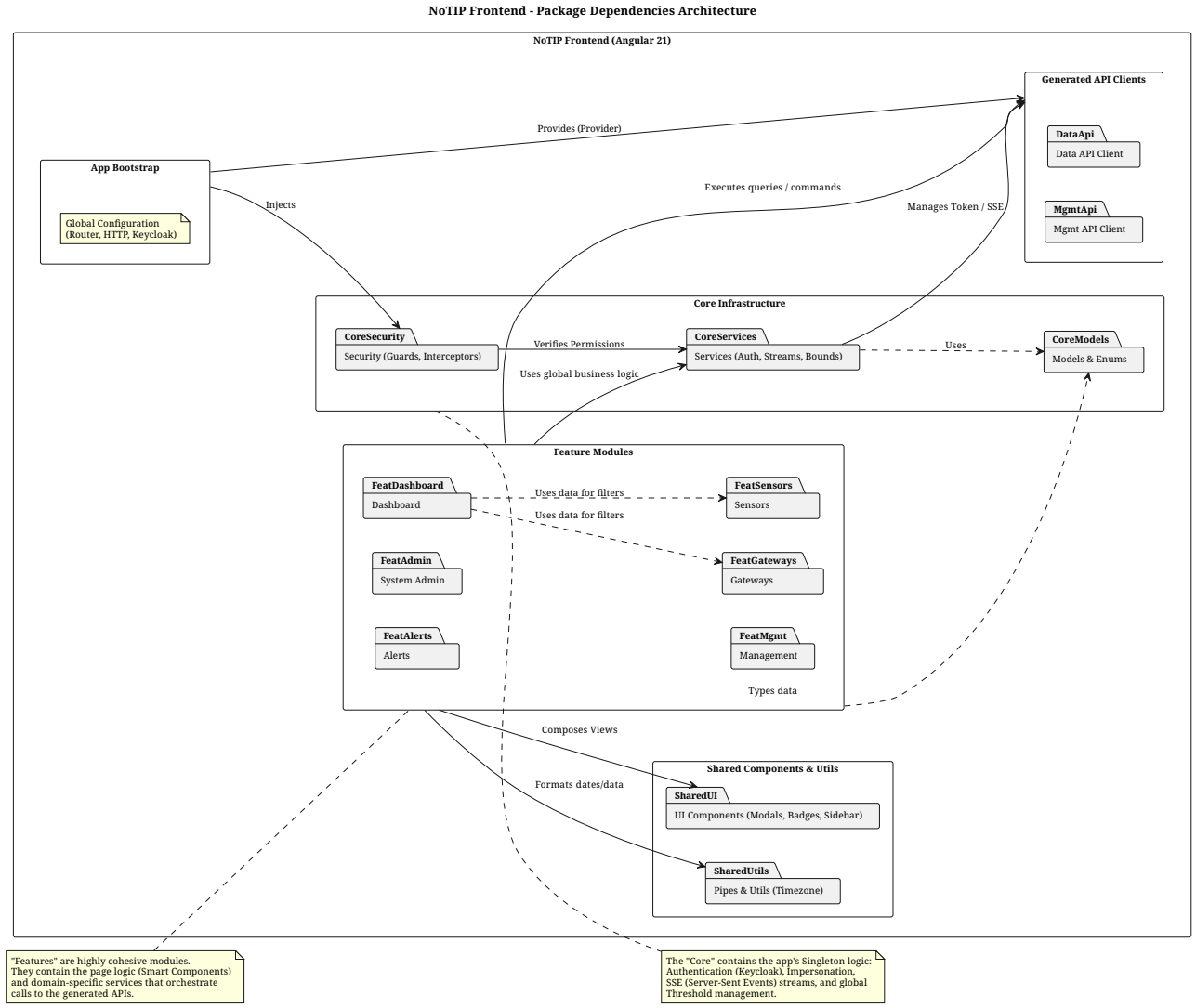

Figura 1: Architettura del *frontend*$_G$ `notip-frontend`$_G$

3.4. Struttura delle rotte

Path	Componente	Ruoli Richiesti	Descrizione
`/dashboard`$_G$	DataDashboardPageComponent	`tenant_user`, `tenant_admin`	*Dashboard*$_G$ *telemetria*$_G$ con modalità stream e query

`/gateways`	GatewayListPageComponent	`tenant_user`, `tenant_admin`	Lista *gateway*$_G$ del *tenant*$_G$ con card
`/gateways/:id`	GatewayDetailPageComponent	`tenant_user`, `tenant_admin`	Dettaglio *gateway*$_G$, sensori, comandi, *telemetria*$_G$ live
`/sensors`	SensorListPageComponent	`tenant_user`, `tenant_admin`	Lista sensori del *tenant*$_G$
`/sensors/:id`	SensorDetailPageComponent	`tenant_user`, `tenant_admin`	Dettaglio sensore con *telemetria*$_G$
`/alerts`	AlertListPageComponent	`tenant_user`, `tenant_admin`	Lista alert *gateway*$_G$ offline
`/alerts/config`	AlertConfigPageComponent	`tenant_user`, `tenant_admin`	Configurazione timeout alert per *gateway*$_G$/ *tenant*$_G$
`/mgmt/users`	UserListPageComponent	`tenant_admin`	Gestione utenti del *tenant*$_G$
`/mgmt/limits`	ThresholdSettingsPageComponent	`tenant_user`, `tenant_admin`	Configurazione soglie min/ max per sensori
`/mgmt/api`	ApiClientListPageComponent	`tenant_admin`	Gestione client API del *tenant*$_G$
`/mgmt/logs`	AuditLogPageComponent	`tenant_admin`	Consultazione *audit*$_G$ log *tenant*$_G$
`/mgmt/costs`	CostDashboardPageComponent	`tenant_admin`	*Dashboard*$_G$ costi (storage e banda)
`/admin/tenants`	TenantManagerPageComponent	`system_admin`	Gestione *tenant*$_G$ del sistema
`/admin/tenants/:id/ users`	TenantDetailPageComponent	`system_admin`	Dettaglio *tenant*$_G$ con lista utenti

`/admin/gateways`	AdminGatewayListPageComponent	`system_admin`	Gestione *gateway*$_G$ a livello sistema
`/error`	ErrorPageComponent	-	Pagina di errore con reason e retryUrl
**	redirect	-	Redirect a `/error?reason=not-found`

Tabella 4: Tabella delle rotte dell'applicazione

3.5. Route Guards

L'applicazione definisce tre route guards principali per proteggere le rotte e gestire l'accesso in base all'autenticazione e ai ruoli:

Route guard	Tipo	Responsabilità
AuthGuard	`CanActivate`	Verifica che *Keycloak*$_G$ sia inizializzato e che l'utente possieda un token *JWT*$_G$ valido. Se non autenticato, reindirizza al login di *Keycloak*$_G$. Al primo accesso valido, avvia il `ThresholdPrefetchService` per precaricare le soglie di validazione della *telemetria*$_G$.
RoleGuard	`CanActivate`	Confronta il ruolo dell'utente (estratto dal *JWT*$_G$) con i ruoli richiesti dalla rotta (`data['roles']`). Se il ruolo non è autorizzato, reindirizza: i `system_admin` verso `/admin/tenants`, gli altri verso `/dashboard`$_G$.
HomeRedirectGuard	`CanActivate`	Utilizzato sulla rotta vuota (`/`) per reindirizzare automaticamente in base al ruolo: `system_admin` verso `/admin/tenants`, tutti gli altri verso `/dashboard`$_G$.

Tabella 5: Route guards del *frontend*$_G$

3.5.1. AuthGuard

Campo	Tipo	Note
`auth`	`AuthService`	Servizio di autenticazione iniettato
`thresholdPrefetch`	`ThresholdPrefetchService`	Servizio di prefetch soglie iniettato

Tabella 6: Campi di AuthGuard

Metodo	Firma	Comportamento
`canActivate()`	`(): Promise<boolean>`	Verifica inizializzazione auth, richiede token valido e avvia `thresholdPrefetch.start()` al primo accesso valido. In caso di errore/autenticazione assente avvia `login()` e ritorna `false`.

Tabella 7: Metodi pubblici di AuthGuard

3.5.2. RoleGuard e HomeRedirectGuard

Campo	Tipo	Note
`auth`	`AuthService`	Risoluzione ruolo effettivo dell'utente
`router`	`Router`	Costruzione `UrlTree` di redirect

Tabella 8: Campi di RoleGuard e HomeRedirectGuard

Metodo	Firma	Comportamento
`RoleGuard.canActivate(route)`	`(route: ActivatedRouteSnapshot): boolean | UrlTree`	Confronta `auth.getRole()` con `route.data['roles']`. Se il ruolo non e autorizzato, redirige `system_admin` a `/admin/tenants` e gli altri a `/dashboard`$_G$.
`HomeRedirectGuard.canActivate()`	`(): UrlTree`	Redirezione automatica dalla home: `system_admin` su `/admin/tenants`, altri ruoli su `/dashboard`$_G$.

Tabella 9: Metodi pubblici di RoleGuard e HomeRedirectGuard

4. Autenticazione e autorizzazione

4.1. *Keycloak*$_G$ integration

L'autenticazione è gestita da *Keycloak*$_G$ tramite la libreria `keycloak`$_G$`-angular`$_G$. La configurazione prevede:

- ***PKCE*$_G$ method**: S256 (Proof Key for Code Exchange)
- **OnLoad**: `login-required` (l'utente deve autenticarsi per accedere)
- **Check login iframe**: disabilitato (`false`)
- **Auto-refresh token**: abilitato tramite `withAutoRefreshToken` con timeout di sessione di 10 minuti
- **Inactivity timeout**: logout automatico al superamento del timeout (gestito da `UserActivityService`)

I servizi `AutoRefreshTokenService` e `UserActivityService` sono registrati in `app.config.ts` come provider *Keycloak*$_G$ e gestiscono il rinnovo automatico del token e il monitoraggio dell'attività utente.

4.2. AuthService

Il servizio `AuthService` (`src/app/core/services/auth.service.ts`) è il fulcro della gestione identitaria e implementa le interfacce `SessionLifeCycle` e `ImpersonationStatus`. Le sue responsabilità includono:

Campo	Tipo	Note
`keycloak`$_G$	`Keycloak`$_G$ (via `inject()`)	Client *Keycloak*$_G$ iniettato
`keycloakEventSignal`	`Signal<KeycloakEvent>`	Signal eventi *Keycloak*$_G$

`authApi`	`AuthApiService` (via `inject()`)	Client OpenAPI per *endpoint*$_G$ auth management-api
`logoutSubject`	`Subject<void>`	Canale interno per lifecycle di logout
`impersonatingSignal`	`Signal<boolean>`	Stato impersonazione read-only verso i *consumer*$_G$
`impersonationTokenSignal`	`Signal<string | null>`	Token impersonato corrente
`impersonationPayloadSignal`	`Signal<JwtPayload | null>`	*Payload*$_G$ *JWT*$_G$ impersonato
`STORAGE_KEY`	`'impersonation'`	Chiave `sessionStorage` per persistenza contesto

Tabella 10: Campi principali di AuthService

Metodo	Firma	Comportamento
`init`$_G$`()`	`(): Promise<boolean>`	Verifica stato `keycloak`$_G$`.authenticated`
`login()`	`(): void`	Redirect verso login *Keycloak*$_G$
`logout()`	`(): void`	Pulizia contesto impersonazione, emit su `logout$`, logout *Keycloak*$_G$
`getToken()`	`(): Promise<string>`	Ritorna token impersonato se presente, altrimenti token *Keycloak*$_G$ con `updateToken(30)`
`getUsername()`	`(): Promise<string>`	Risoluzione nome utente da claims `preferred_username|username|name`
`getRole()`	`(): UserRole`	Mapping ruoli da *JWT*$_G$ (`realm_access`, `resource_access`) con fallback `tenant_user`
`getTenantId()`	`(): string`	Estrae claim `tenant_id`
`getUserId()`	`(): string`	Estrae claim `sub`
`setImpersonating(value)`	`(value: boolean): void`	Attiva/disattiva stato impersonazione e sincronizza storage
`stopImpersonation()`	`(): void`	Disattiva impersonazione
`startImpersonation(targetUserId)`	`(targetUserId: string): Observable<string>`	Chiama `/auth/impersonate`, salva token/*payload*$_G$ e attiva segnali

Tabella 11: Metodi pubblici di AuthService

Metodo	Comportamento
`saveImpersonationContext(token, payload`$_G$`)`	Persistenza *JSON*$_G$ in `sessionStorage`
`clearImpersonationStorage()`	Rimozione storage locale impersonazione
`loadImpersonating()`	Ripristina flag impersonazione da storage
`loadImpersonationToken()`	Ripristina token impersonazione
`loadImpersonationPayload()`	Ripristina *payload*$_G$ impersonazione
`collectRoles(payload`$_G$`)`	Accorpa ruoli da claim multipli
`decodeTokenPayload(token)`	Decodifica base64url del *payload*$_G$ *JWT*$_G$
`decodeJwtPayload()`	Risoluzione *payload*$_G$ corrente (impersonato o *keycloak*$_G$)
`clearImpersonationContext()`	Reset segnali e storage

Tabella 12: Metodi privati di AuthService

4.3. Impersonazione

Il sistema supporta l'impersonazione: un `system_admin` può assumere l'identità di un utente *tenant*$_G$ per operare nel suo contesto. Il flusso è:

1. Il system admin seleziona un utente *tenant*$_G$ dalla pagina di dettaglio del *tenant*$_G$;
2. `AuthService.startImpersonation()` scambia il token (tramite *OAuth2*$_G$ Token Exchange lato management-api) e salva il contesto impersonato in `sessionStorage`;
3. La pagina `TenantDetail` naviga verso `/dashboard`$_G$; `AuthService` ripristina lo stato da storage anche dopo refresh;
4. `DashboardResolver` rileva lo stato di impersonazione e imposta `dataMode: 'obfuscated'`;
5. Tutti i componenti che consumano *telemetria*$_G$ usano *endpoint*$_G$ e modalità offuscate;
6. Le azioni sensibili sono limitate: la rinomina *gateway*$_G$ è nascosta in UI e gli *endpoint*$_G$ *backend*$_G$ con `BlockImpersonationGuard` rifiutano operazioni bloccate;
7. L'UI mostra un banner «Impersonation mode» e un tag «OBFUSCATED MODE»;
8. Lo stop impersonazione dalla sidebar chiude il contesto e riporta l'utente su `/admin/tenants`.

5. Interceptor HTTP

L'applicazione registra due interceptors HTTP globali:

5.1. AuthInterceptor

`authInterceptor` (`src/app/core/interceptors/auth.interceptor.ts`) intercetta ogni richiesta HTTP in uscita e:

Elemento	Descrizione
Tipo	`HttpInterceptorFn`
Dipendenze	`AuthService` via `inject()`
Pipeline$_G$ RxJS	`from(auth.getToken()) -> switchMap(...)`
Header gestito	`Authorization: Bearer <token>`

Tabella 13: Struttura di AuthInterceptor

5.2. ErrorInterceptor

`errorInterceptor` (`src/app/core/interceptors/error.interceptor.ts`) gestisce le risposte HTTP con errore:

Caso	Comportamento
`Errore non HTTP`	Propaga errore originale con `throwError()`
`401 Unauthorized`	Redirect a `/error?reason=unauthorized&retryUrl=...` se non si e già sulla rotta `/error`
`403 Forbidden`	Redirect a `/error?reason=forbidden`
`Altri status`	Propagazione errore senza redirect aggiuntivo

Tabella 14: Regole di ErrorInterceptor

6. Modelli di dominio

I modelli di dominio sono definiti in `src/app/core/models/` e rappresentano i contratti typed tra i vari strati dell'applicazione.

6.1. Enums

Enum	Valori
UserRole	`system_admin`, `tenant_admin`, `tenant_user`
TenantStatus	`active`, `suspended`
CommandStatus	`queued`, `ack`$_G$, `nack`, `expired`, `timeout`
AlertsType	`GATEWAY_OFFLINE`
GatewayStatus	`online`, `paused`, `provisioning`$_G$, `offline`
CmdGatewayStatus	`online`, `paused`

Tabella 15: Enumerazioni del dominio

6.2. *Gateway*$_G$

Campo	Tipo	Note
`gatewayId`	`string`	Identificativo univoco del *gateway*$_G$
`name`	`string`	Nome visualizzato del *gateway*$_G$
`status`	`GatewayStatus`	Stato operativo corrente
`lastSeenAt`	`string | null`	Ultimo timestamp di contatto (opzionale)
`provisioned`	`boolean`	Indica se il *provisioning*$_G$ e stato completato
`firmwareVersion`	`string | null`	Versione firmware corrente (opzionale)
`sendFrequencyMs`	`number`	Intervallo di invio *telemetria*$_G$ in millisecondi

Tabella 16: Campi di *Gateway*$_G$

Campo	Tipo	Note
`gatewayId`	`string`	Identificativo *gateway*$_G$
`tenantId`	`string`	*Tenant*$_G$ proprietario del *gateway*$_G$
`model`	`string`	Modello hardware *gateway*$_G$
`firmware`	`string | null`	Versione firmware in modalita offuscata (opzionale)

provisioned	boolean	Stato di *provisioning*$_G$
factoryId	string	Identificativo fabbrica/*provisioning*$_G$
createdAt	string	Data di creazione del *gateway*$_G$

Tabella 17: Campi di ObfuscatedGateway

Campo	Tipo	Note
factoryId	string	Identificativo fabbrica del *gateway*$_G$ da registrare
tenantId	string	*Tenant*$_G$ a cui associare il *gateway*$_G$
factoryKey	string	Chiave di registrazione/*provisioning*$_G$
model	string	Modello del *gateway*$_G$ da registrare

Tabella 18: Campi di AddGatewayParameters

6.3. *Telemetria*$_G$

Campo	Tipo	Note
gatewayId	string	*Gateway*$_G$ sorgente della misura
sensorId	string	Sensore sorgente della misura
sensorType	string	Tipo sensore (es. temperatura, umidita)
timestamp	string	Timestamp *ISO 8601*$_G$ della rilevazione
keyVersion	number	Versione chiave usata per la cifratura
encryptedData	string	*Payload*$_G$ cifrato
iv$_G$	string	Initialization vector
authTag	string	Tag di autenticazione *GCM*$_G$

Tabella 19: Campi di TelemetryEnvelope

Campo	Tipo	Note
gatewayId	string	*Gateway*$_G$ sorgente
sensorId	string	Sensore sorgente
sensorType	string	Tipo del sensore
timestamp	string	Timestamp *ISO 8601*$_G$
value	number	Valore numerico in chiaro
unit	string	Unita di misura

Tabella 20: Campi di DecryptedEnvelope

Campo	Tipo	Note
gatewayId	string	Ereditato da DecryptedEnvelope
sensorId	string	Ereditato da DecryptedEnvelope
sensorType	string	Ereditato da DecryptedEnvelope
timestamp	string	Ereditato da DecryptedEnvelope
value	number	Ereditato da DecryptedEnvelope
unit	string	Ereditato da DecryptedEnvelope
isOutofBounds	boolean	Indica se il valore supera i bound configurati

Tabella 21: Campi di CheckedEnvelope

Campo	Tipo	Note
gatewayId	string	*Gateway*$_G$ sorgente
sensorId	string	Sensore sorgente
sensorType	string	Tipo del sensore
timestamp	string	Timestamp della misura in modalita offuscata

Tabella 22: Campi di ObfuscatedEnvelope

6.4. *Tenant*$_G$

Campo	Tipo	Note
tenantId	string	Identificativo univoco *tenant*$_G$
name	string	Nome *tenant*$_G$
status	TenantStatus	Stato *tenant*$_G$ (active/suspended)
suspensionIntervalDays	number | null	Giorni prima della sospensione automatica (opzionale)
createdAt	string	Data di creazione *tenant*$_G$

Tabella 23: Campi di *Tenant*$_G$

6.5. User

Campo	Tipo	Note
userId	string	Identificativo univoco utente
role	UserRole	Ruolo applicativo
username	string	Username visualizzato
email	string	Email utente
lastAccess	string | null	Ultimo accesso noto, null se assente

Tabella 24: Campi di ViewUser

7. Servizi core

I servizi singleton dell'applicazione forniscono funzionalità trasversali a tutte le feature.

7.1. ObfuscatedStreamManagerService

Campo	Tipo	Note
auth	AuthService	Recupero token e login fallback
abortController	AbortController | null	Gestione lifecycle stream *SSE*$_G$
channel	ObservableChannel<TelemetryEnvelope> | null	Canale di emissione per subscriber
streamSessionId	number	Protezione da stream stale concorrenti

Tabella 25: Campi di ObfuscatedStreamManagerService

Metodo	Firma	Comportamento

`openStream(sp)`	`(sp: StreamParameters): Observable<TelemetryEnvelope>`	Apre stream *SSE*$_G$, crea canale observabile e avvia sessione
`closeStream()`	`(): void`	Chiude stream corrente, abortisce fetch e completa il canale

Tabella 26: Metodi pubblici di ObfuscatedStreamManagerService

Metodo	Comportamento
`startStream(...)`	*Handshake*$_G$ *SSE*$_G$ con *bearer token*$_G$ e gestione callback `onopen/onmessage/onerror/onclose`
`buildQuery(sp)`	Costruzione querystring da filtri *gateway*$_G$/sensore/tipo
`parseTelemetryEnvelope(raw)`	Parsing *JSON*$_G$ evento *SSE*$_G$ e validazione *payload*$_G$
`handleMalformedMessage(...)`	Propaga errore e abort in caso *payload*$_G$ invalido
`isTokenExpiredEvent(value)`	Rilevamento evento applicativo `token_expired`
`isTelemetryEnvelope(value)`	Type guard runtime su envelope

Tabella 27: Metodi privati di ObfuscatedStreamManagerService

7.2. ThresholdPrefetchService

Campo	Tipo	Note
`REFRESH_INTERVAL_MS`	`5 * 60 * 1000`	Intervallo refresh soglie (5 minuti)
`thresholdService`	`ThresholdService`	Servizio cache soglie
`authService`	`AuthService`	Verifica ruolo/*tenant*$_G$ e lifecycle logout
`refreshSub`	`Subscription | null`	Subscription timer periodico

Tabella 28: Campi di ThresholdPrefetchService

Metodo	Firma	Comportamento
`start()`	`(): void`	Avvia timer immediato + periodico; evita duplicati e salta avvio se non eleggibile
`stop()`	`(): void`	Arresta timer e azzera subscription

Tabella 29: Metodi pubblici di ThresholdPrefetchService

Metodo	Comportamento
`shouldStart()`	Ritorna `false` per `system_admin` o *tenant*$_G$ assente

Tabella 30: Metodi privati di ThresholdPrefetchService

7.3. ThresholdService

Campo	Tipo	Note
`thresholdsApi`	`ThresholdsService (OpenAPI)`	Client management-api per *endpoint*$_G$ soglie
`cache`	`ThresholdConfig[]`	Cache locale in memoria

Tabella 31: Campi di ThresholdService

Metodo	Firma	Comportamento

`fetchThresholds()`	`(): Observable<ThresholdConfig[]>`	Carica soglie da API, normalizza *payload*$_G$ e aggiorna cache
`getCached()`	`(): ThresholdConfig[]`	Ritorna snapshot cache corrente
`invalidateCache()`	`(): void`	Azzera cache
`refreshThresholds()`	`(): Observable<ThresholdConfig[]>`	Invalidate + reload
`setDefaultThreshold(...)`	`(sensorType, minValue?, maxValue?): Observable<void>`	Upsert soglia di default per tipo sensore
`setSensorThreshold(...)`	`(sensorId, sensorType, minValue?, maxValue?): Observable<void>`	Upsert soglia specifica sensore
`deleteSensorThreshold(sensorId)`	`(sensorId: string): Observable<void>`	Delete soglia sensore
`deleteTypeThreshold(sensorType)`	`(sensorType: string): Observable<void>`	Delete soglia per tipo

Tabella 32: Metodi pubblici di ThresholdService

Metodo	Comportamento
`resolveBounds(...)`	Riutilizza bound cached in update parziale
`toThresholds(rows)`	Normalizza *payload*$_G$ API snake_case/camelCase in `ThresholdConfig[]`
`normalizeType(type)`	Mapping tipo in `sensorId|sensorType`
`toNullableNumber(value)`	Parsing robusto numeri
`toNonEmptyString(value)`	Validazione stringhe non vuote

Tabella 33: Metodi privati di ThresholdService

7.4. MeasureBoundsEvaluationService

Campo	Tipo	Note
`thresholds`	`ThresholdService`	Accesso cache soglie

Tabella 34: Campi di MeasureBoundsEvaluationService

Metodo	Firma	Comportamento
`evaluate(envelope)`	`(envelope: DecryptedEnvelope): boolean`	Priorita match su `sensorId`, fallback `sensorType`, `true` solo se fuori bound

Tabella 35: Metodi pubblici di MeasureBoundsEvaluationService

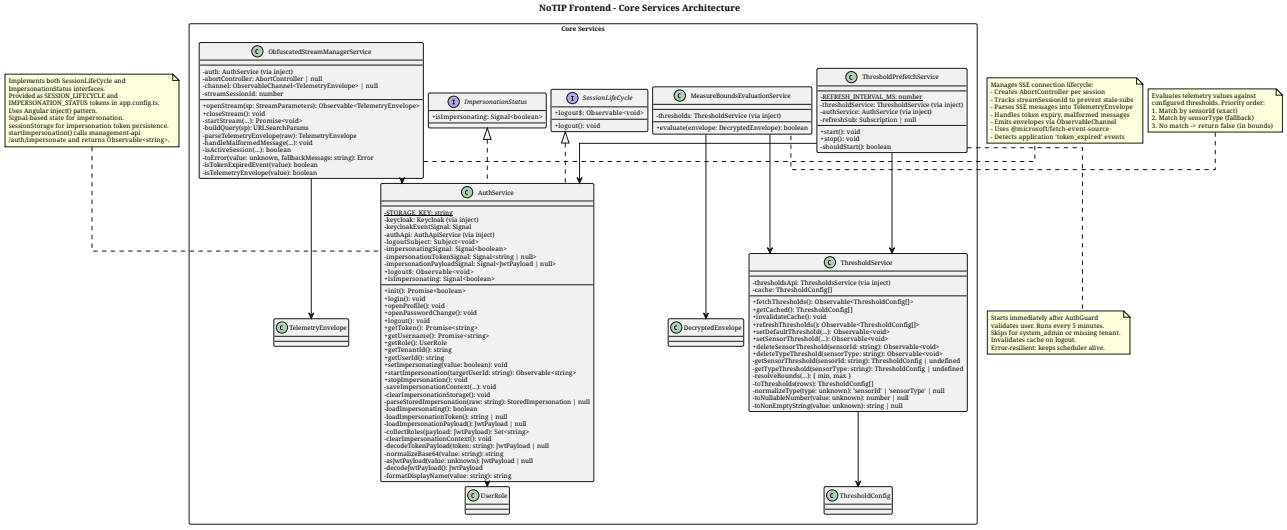

Figura 2: Diagramma dei servizi core

8. Generazione codice OpenAPI

I client API verso i *backend*$_G$ sono generati automaticamente tramite OpenAPI Generator. Il workflow è definito in `scripts/generate-openapi.sh`:

1. Recupera gli specchi OpenAPI (`openapi.json`$_G$) dai *repository*$_G$ dei servizi produttori tramite *GitHub*$_G$ API;
2. Salva le specifiche in `api-contracts/openapi/`;
3. Esegue OpenAPI Generator con generatore `typescript`$_G$`-angular`$_G$;
4. Output in `src/app/generated/openapi/{notip-management-api-openapi, notip-data-api-openapi}/`;
5. Applica Prettier al codice generato per uniformità di stile.

I client generati forniscono:

- **Management API** (`/api/mgmt`): classi come `GatewaysService`, `AdminTenantsService`, `AdminGatewaysService`, `AlertsService`, `AuditLogService`, `ApiClientService`, `CommandsService`, `CostsService`, `KeysService`, `ThresholdsService`, `UsersService`, `AuthService`;
- **Data API** (`/api/data`): classi come `MeasuresService` (query, export, stream), `SensorsService`, `AppService`.

Entrambi i set di client sono registrati in `app.config.ts` tramite `provideMgmtApi('/api/mgmt')` e `provideDataApi('/api/data')`, che configurano il base path per le chiamate.

9. Feature: *Dashboard*$_G$

La *dashboard*$_G$ *telemetria*$_G$ è il componente principale dell'applicazione per gli utenti *tenant*$_G$. Supporta due modalità operative e diverse funzionalità di visualizzazione e interazione.

9.1. DataDashboardPageComponent

`DataDashboardPageComponent` (`src/app/features/dashboard`$_G$`/pages/data-dashboard`$_G$`/`) è orchestrata dal resolver `DashboardResolver` che determina la modalità dati:

- **`dataMode: 'clear'`**: *telemetria*$_G$ decrittografata, modalità standard per utenti non in impersonazione;
- **`dataMode: 'obfuscated'`**: *telemetria*$_G$ offuscata, forzata durante l'impersonazione.

La pagina supporta due view mode:

- **Stream mode**: connessione *SSE*$_G$ in tempo reale tramite `ObfuscatedStreamManagerService`. I dati vengono decrittati da `DecryptedMeasureService` (che usa `@notip/crypto-sdk`$_G$) e validati da `ValidatedMeasureFacadeService` che aggiunge il flag `isOutofBounds`. Il grafico è aggiornato in tempo reale con un cap di 20 righe;
- **Query mode**: query paginata con cursore composito `(time, sensorId)`. Supporta filtri per gatewayIds, sensorTypes, sensorIds, e range temporale (con limite finestra 24h).

9.2. FilterPanelComponent

`FilterPanelComponent` (`src/app/features/dashboard`$_G$`/components/filter-panel/`) fornisce i controlli di filtro:

- Dropdown multi-select custom per `gatewayIds`, `sensorTypes`, `sensorIds` con ricerca locale;
- Input datetime-local per range temporale (solo in query mode);
- Validazione finestra temporale massima di 24h con clamping automatico;
- Conversione datetime nel fuso orario Roma tramite pipe e utility dedicate;
- Cross-filter dependency: la selezione di un *gateway*$_G$ aggiorna i cataloghi di sensori disponibili;
- Rimozione automatica delle selezioni incompatibili (es. sensore non appartenente al *gateway*$_G$ selezionato);
- Toggle singolo click per apertura/chiusura dropdown;
- Chiusura dropdown su click esterno o tasto Escape.

9.3. TelemetryChartComponent

`TelemetryChartComponent` (`src/app/features/dashboard`$_G$`/components/telemetry-chart/`) visualizza i dati telemetrici:

- Utilizza Chart.js per rendering di grafici lineari;
- Calcola sensori unici dai dati ricevuti;
- Crea un dataset per sensore con colorazione distinta;
- Cap a 120 punti per dataset per performance;
- Formattazione timestamp in locale italiano;
- Gestione fallback per timestamp invalidi;
- Rilevamento modalità «obfuscated-only» e visualizzazione conteggio.

9.4. TelemetryTableComponent

`TelemetryTableComponent` (`src/app/features/dashboard`$_G$`/components/telemetry-table/`) mostra i dati in formato tabella:

- Colonne: Timestamp (Rome timezone pipe), *Gateway*$_G$, Sensor, Type, Value;
- Ordine invertito (più recente in alto);
- Valori offuscati: mascheramento con stringa `*** OBFUSCATED ***`;
- Valori non numerici (`!Number.isFinite`): visualizzazione `n/a`;
- Righe con valori out-of-bounds: evidenziazione con classe CSS `row-alert`;
- Scroll orizzontale per tabelle larghe.

9.5. *Pipeline*$_G$ di decrittografia *telemetria*$_G$

Il flusso di elaborazione della *telemetria*$_G$ segue una *pipeline*$_G$ a tre stadi:

1. **ObfuscatedStreamManagerService**: riceve eventi *SSE*$_G$ grezzi (`TelemetryEnvelope` con `encryptedData`, `iv`$_G$, `authTag`);
2. **DecryptedMeasureService**: utilizza `@notip/crypto-sdk`$_G$ (`DataApiService`) per decrittografare con *AES-256*$_G$*-GCM*$_G$, producendo `DecryptedEnvelope` con `value` e `unit` in chiaro;
3. **ValidatedMeasureFacadeService**: valuta `isOutofBounds` tramite `MeasureBoundsEvaluationService` contro le soglie cached, producendo `CheckedEnvelope`.

La *pipeline*$_G$ opera in entrambe le modalità:

- **Stream (*SSE*$_G$)**: `ObfuscatedStreamManagerService.openStream()` -> `DecryptedMeasureService.openStream()` -> `ValidatedMeasureFacadeService.openStream()`;
- **Query (paginata)**: `ObfuscatedMeasureService.query()` oppure `ValidatedMeasureFacadeService.query()`, con cursore composito;
- **Export**: `ValidatedMeasureFacadeService.export()` in modalità clear, con limite finestra 24h.

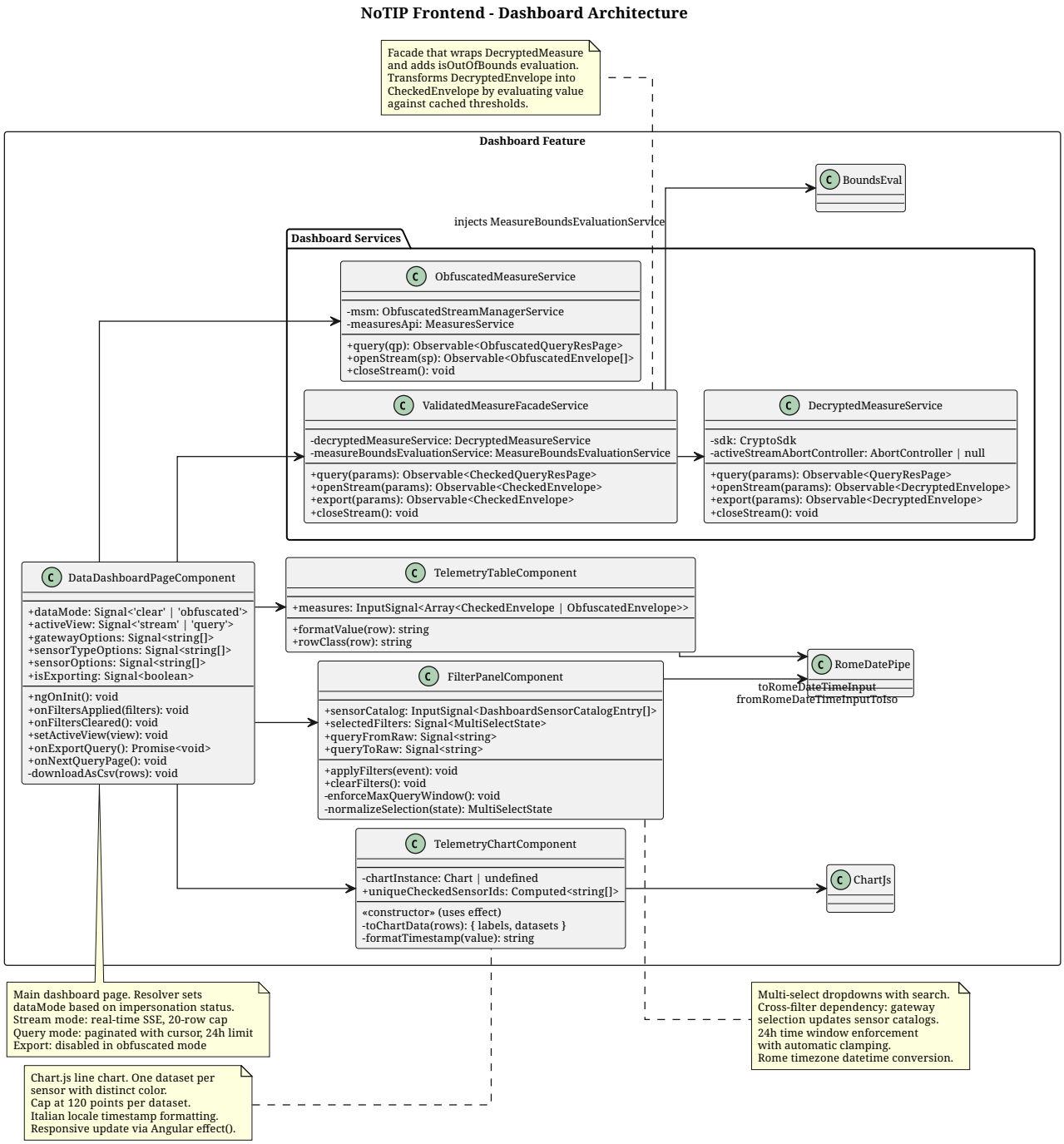

Figura 3: Diagramma della *dashboard*$_G$

10. Feature: *Gateway*$_G$

La gestione dei *gateway*$_G$ consente agli utenti *tenant*$_G$ di visualizzare, rinominare, configurare e inviare comandi ai propri *gateway*$_G$ *IoT*$_G$.

10.1. GatewayListPageComponent

`GatewayListPageComponent` carica la lista *gateway*$_G$ tramite `GatewayService` e renderizza card per ciascun *gateway*$_G$. La navigazione al dettaglio avviene tramite selezione della card.

10.2. GatewayDetailPageComponent

`GatewayDetailPageComponent` è la pagina più complessa dell'applicazione. Fornisce:

- **Riepilogo *gateway*$_G$**: nome, stato, ID, firmware, frequenza di invio;
- **Azioni *gateway*$_G$** (solo `tenant_admin` o se `canManage`):
  - Rinomina (GatewayRenameModalComponent);
  - Invio comando configurazione (CommandModalComponent mode `config`);
  - Invio comando firmware (CommandModalComponent mode `firmware`);
  - Eliminazione *gateway*$_G$ (DeleteConfirmModalComponent);
- **Lista sensori**: sensori associati al *gateway*$_G$ con navigazione al dettaglio;
- ***Telemetria*$_G$ live**: stream in tempo reale dei dati del *gateway*$_G$ selezionato;
- **Stato comandi**: polling dello stato dei comandi inviati con visualizzazione progresso.

10.3. GatewayService

`GatewayService` (`src/app/features/gateways/services/gateway`$_G$`.service.ts`) gestisce lo stato e le operazioni sui *gateway*$_G$:

Campo	Tipo	Note
`gatewaysApi`	`GatewaysApiService (OpenAPI)`	Client management-*api gateway*$_G$
`http`	`HttpClient`	Iniettato nel servizio
`listSignal`	`Signal<Gateway`$_G$`[]>`	Stato lista *gateway*$_G$
`selectedGatewaySignal`	`Signal<Gateway`$_G$` | null>`	*Gateway*$_G$ selezionato
`loadingSignal`	`Signal<boolean>`	Flag loading richieste

Tabella 36: Campi di GatewayService

Metodo	Firma	Comportamento
`getGateways()`	`(): Observable<Gateway`$_G$`[]>`	Fetch lista *gateway*$_G$, mapping *DTO*$_G$->model e update `listSignal`
`getGatewayDetail(gatewayId)`	`(gatewayId: string): Observable<Gateway`$_G$`>`	Fetch dettaglio *gateway*$_G$ e update `selectedGatewaySignal`
`updateGatewayName(gatewayId, name)`	`(gatewayId: string, name: string): Observable<GatewayUpdateResult>`	*Patch*$_G$ nome *gateway*$_G$ e mapping risposta
`deleteGateway(gatewayId)`	`(gatewayId: string): Observable<string>`	Delete *gateway*$_G$ e ritorno id eliminato
`list()`	`(): Signal<Gateway`$_G$`[]>`	Accessor read-only lista
`selectedGateway()`	`(): Signal<Gateway`$_G$` | null>`	Accessor read-only selezione
`isLoading()`	`(): Signal<boolean>`	Accessor read-only loading

Tabella 37: Metodi pubblici di GatewayService

Metodo	Comportamento
`toGateway(dto`$_G$`)`	Mapping robusto `GatewayResponseDto` in `Gateway`$_G$
`toGatewayStatus(status)`	Normalizzazione stato (uppercase/camelCase) con fallback `offline`
`pickString/pickOptionalString`	Utility estrazione stringhe
`pickBoolean`	Utility estrazione boolean

`pickNumber`	Utility coercion numeri con default `30000ms`

Tabella 38: Metodi privati di GatewayService

10.4. CommandService

`CommandService` (`src/app/features/gateways/services/command.service.ts`) gestisce l'invio e il monitoraggio dei comandi:

Elemento	Tipo	Note
`TERMINAL_STATUSES`	`Set<CommandStatus>`	Stati terminali (`ack`$_G$, `nack`, `expired`, `timeout`)
`ALLOWED_CMD_GATEWAY_STATUSES`	`Set<CmdGatewayStatus>`	Stati *gateway*$_G$ ammessi nei comandi config
`commandsApi`	`CommandsApiService (OpenAPI)`	Client API comandi

Tabella 39: Costanti e campi di CommandService

Metodo	Firma	Comportamento
`sendConfig(id, config)`	`(id: string, config: GatewayConfig): Observable<CommandStatusUpdate>`	Invio comando config con filtro campi non validi e chaining polling
`sendFirmware(id, firmware)`	`(id: string, firmware: GatewayFirmware): Observable<CommandStatusUpdate>`	Invio comando firmware e chaining polling
`pollStatus(gwId, cmdId, intervalMs?)`	`(gwId: string, cmdId: string, intervalMs = 2000): Observable<CommandStatusUpdate>`	Polling stato comando con timeout 5 minuti, gestione 304 e mapping 404/503 -> `timeout`

Tabella 40: Metodi pubblici di CommandService

Metodo	Comportamento
`toCommandStatus(status)`	Mapping string status API in enum `CommandStatus`
`mapCommandResponse(response)`	Validazione e mapping risposta invio comando
`mapCommandStatusResponse(response)`	Validazione e mapping risposta stato comando
`isCmdGatewayStatus(status)`	Type guard sugli stati *gateway*$_G$ consentiti

Tabella 41: Metodi privati di CommandService

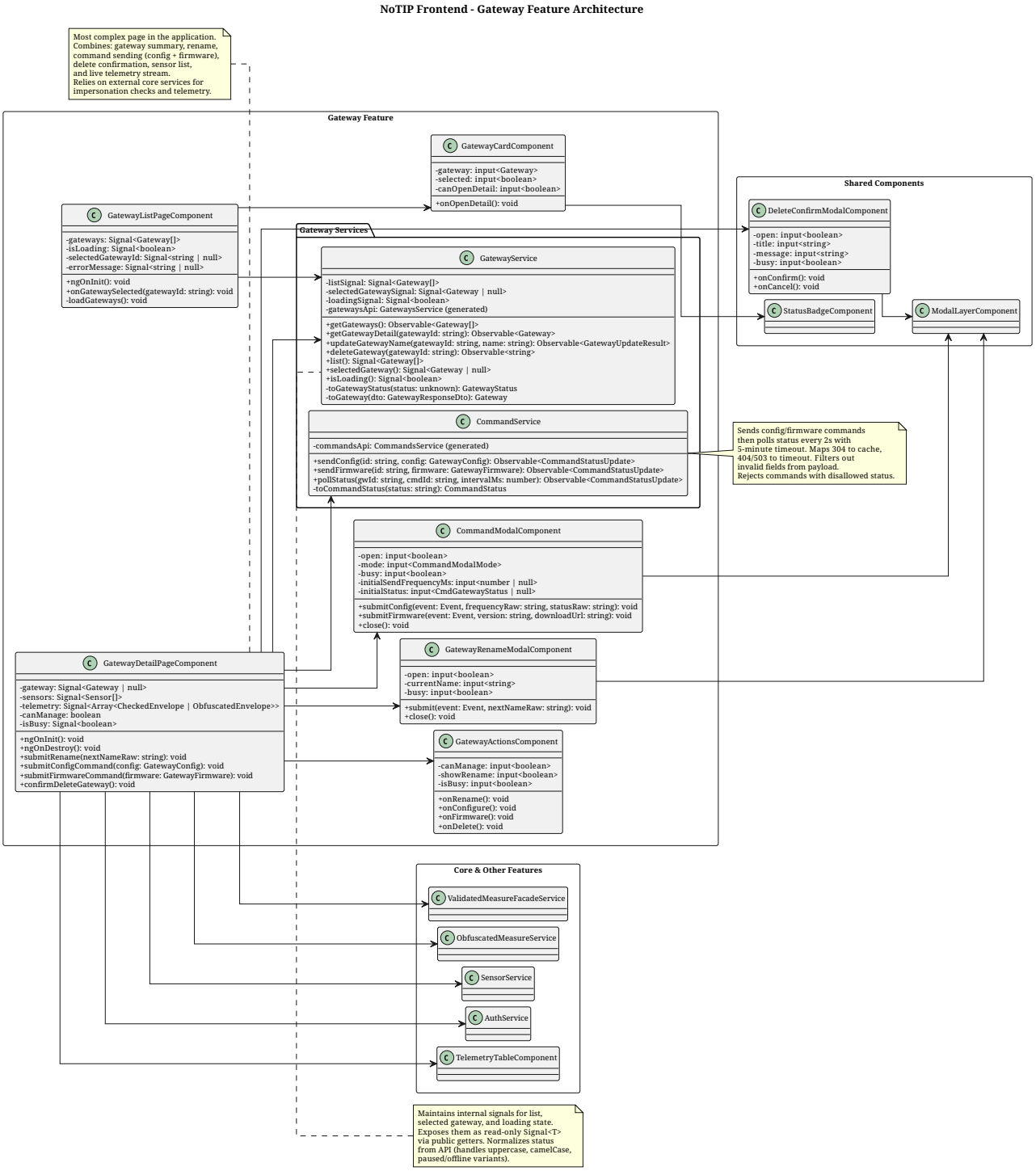

Figura 4: Diagramma della feature *Gateway*$_G$

11. Feature: Admin

Le funzionalità di amministrazione sistema sono accessibili esclusivamente ai `system_admin`.

La feature include anche componenti riutilizzabili sotto `admin/components/`: `AdminGatewayFormComponent`, `AdminGatewayTableComponent`, `ImpersonateButtonComponent`, `TenantFormComponent`, `TenantTableComponent`, `TenantUserListComponent`.

11.1. TenantManagerPageComponent

Campo	Tipo	Note
`tenantService`	`TenantService`	Servizio *tenant*$_G$ iniettato
`router`	`Router`	Navigazione verso dettaglio *tenant*$_G$
`tenants`	`Signal<Tenant_G[]>`	Lista *tenant*$_G$ caricata
`showCreateForm`	`Signal<boolean>`	Toggle form creazione/modifica
`editingTenant`	`Signal<Tenant_G | null>`	*Tenant*$_G$ in editing
`deletingTenantId`	`Signal<string | null>`	*Tenant*$_G$ in conferma cancellazione
`isLoading / isSaving`	`Signal<boolean>`	Stato caricamento/salvataggio
`errorMessage / formErrorMessage`	`Signal<string | null>`	Messaggi errore UI

Tabella 42: Campi di TenantManagerPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Innesca `loadTenants()`
`onTenantSelected(id)`	`(tenantId: string): void`	Naviga a `/admin/tenants/:id/users`
`onCreateRequested(payload_G)`	`(payload_G): void`	Crea *tenant*$_G$ e ricarica lista
`onUpdateRequested(payload_G)`	`(payload_G): void`	Aggiorna *tenant*$_G$ e ricarica lista
`confirmDeleteTenant()`	`(): void`	Elimina *tenant*$_G$ selezionato e aggiorna stato locale
`openCreateTenantForm()`	`(): void`	Toggle form con reset errori

Tabella 43: Metodi pubblici di TenantManagerPageComponent

Metodo	Comportamento
`loadTenants()`	Fetch *tenant*$_G$ via `TenantService.getTenants()` con gestione loading/error

Tabella 44: Metodi privati di TenantManagerPageComponent

11.2. TenantDetailPageComponent

Campo	Tipo	Note
`route / router`	`ActivatedRoute / Router`	Risoluzione parametro `id` e redirect
`adminUserService`	`AdminUserService`	Recupero utenti offuscati del *tenant*$_G$
`tenantService`	`TenantService`	Recupero dettagli *tenant*$_G$
`tenantId`	`Signal<string>`	*Tenant*$_G$ corrente da route params
`tenant_G`	`Signal<Tenant_G | null>`	Dettaglio *tenant*$_G$
`users`	`Signal<ObfuscatedUser[]>`	Utenti *tenant*$_G$
`feedbackMessage`	`Signal<string | null>`	Esito impersonazione

Tabella 45: Campi di TenantDetailPageComponent

Metodo	Firma	Comportamento

`ngOnInit()`	`(): void`	Sottoscrive route params e carica *tenant*$_G$ + utenti
`onImpersonationStarted(userId)`	`(userId: string): void`	Mostra feedback e naviga su `/dashboard`$_G$
`onImpersonationFailed(message)`	`(message: string): void`	Mostra messaggio errore impersonazione

Tabella 46: Metodi pubblici di TenantDetailPageComponent

11.3. AdminGatewayListPageComponent

Campo	Tipo	Note
`adminGatewayService`	`AdminGatewayService`	Servizio *gateway*$_G$ admin
`tenantService`	`TenantService`	Recupero tenantId disponibili per filtro
`gateways`	`Signal<ObfuscatedGateway[]>`	Lista *gateway*$_G$ sistema
`tenantOptions`	`Signal<string[]>`	*Tenant*$_G$ disponibili nel filtro
`selectedTenantIds / draftTenantIds`	`Signal<string[]>`	Filtro applicato vs bozza filtro
`filteredGateways`	`Computed<ObfuscatedGateway[]>`	Lista filtrata per *tenant*$_G$
`showCreateForm`	`Signal<boolean>`	Toggle creazione *gateway*$_G$

Tabella 47: Campi di AdminGatewayListPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Carica *gateway*$_G$ e opzioni *tenant*$_G$
`onApplyFilter(event)`	`(event: Event): void`	Applica bozza filtro ai *tenant*$_G$ selezionati
`onResetFilter()`	`(): void`	Reset filtro *tenant*$_G$
`onCreateGateway(payload`$_G$`)`	`(payload`$_G$`: CreateAdminGatewayPayload): void`	Crea *gateway*$_G$ admin e ricarica lista
`toggleCreateForm()`	`(): void`	Apre/chiude form creazione

Tabella 48: Metodi pubblici di AdminGatewayListPageComponent

11.4. Servizi Admin

Servizio	Metodo	Comportamento
`TenantService`	`getTenants()`	Recupera lista *tenant*$_G$ e mappa status/ suspension interval
`TenantService`	`createTenant(c)`	Crea *tenant*$_G$ con *payload*$_G$ admin bootstrap
`TenantService`	`updateTenant(id, u)`	Aggiorna nome/stato/intervallo con normalizzazione input
`TenantService`	`deleteTenant(id)`	Elimina *tenant*$_G$

`AdminUserService`	`getUsers(tenantId)`	Recupera utenti *tenant*$_G$ e mappa ruolo in `UserRole`
`AdminGatewayService`	`getGateways(tenantId?)`	Lista *gateway*$_G$ admin con mapping robusto dei campi
`AdminGatewayService`	`addGateway(params)`	*Provisioning*$_G$ *gateway*$_G$ con `factory_id/factory_key/model`

Tabella 49: Metodi pubblici dei servizi Admin

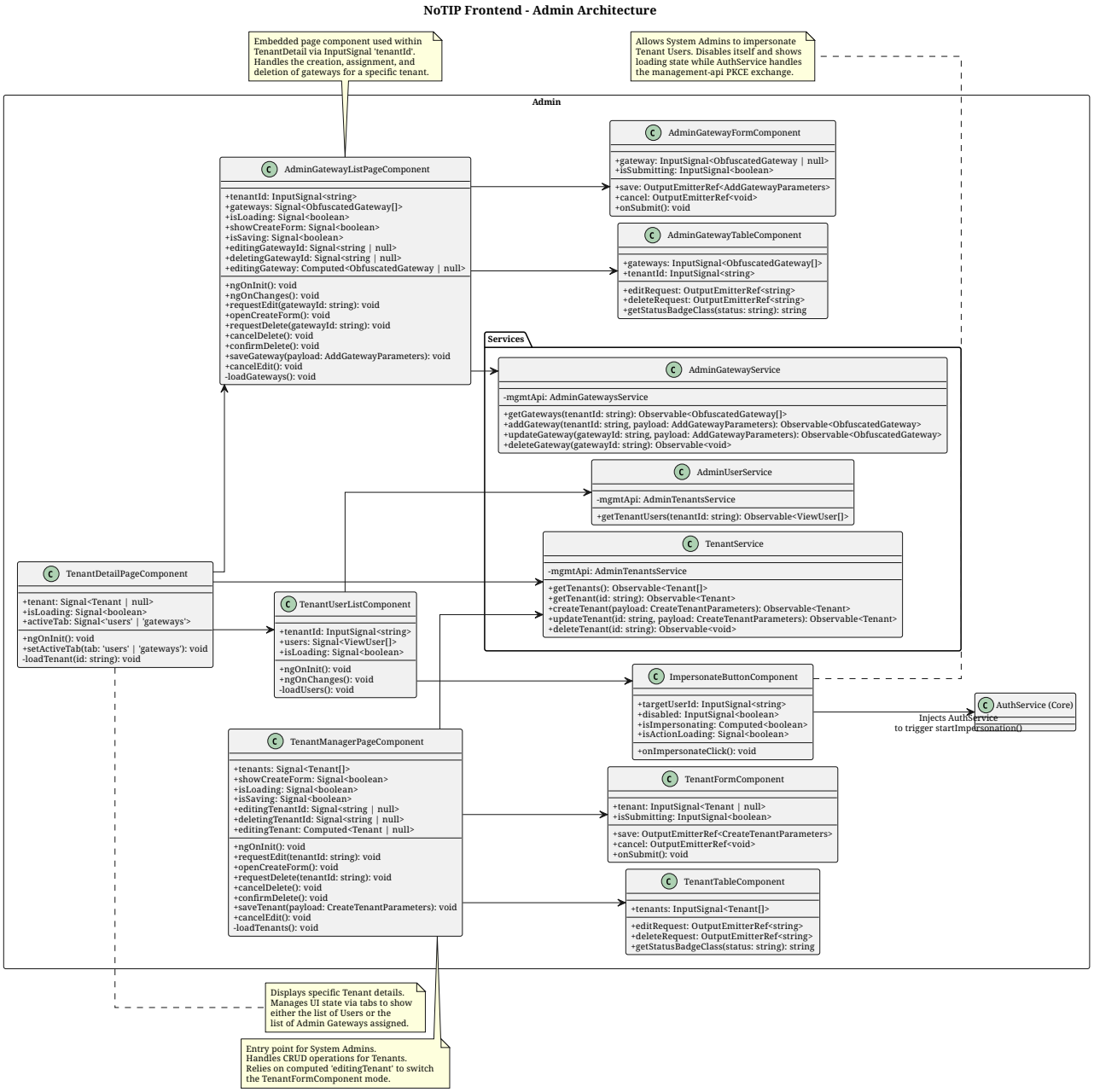

Figura 5: Diagramma della feature Admin

12. Feature: Alerts

La gestione degli alert consente di configurare e consultare gli alert di *gateway*$_G$ offline.

La feature include componenti sotto `alerts/components/`: `AlertConfigFormComponent`, `AlertFilterPanelComponent`.

12.1. AlertListPageComponent

Campo	Tipo	Note
`alertService`	`AlertService`	Caricamento alert con filtri
`gatewayService`	`GatewayService`	Costruzione opzioni filtro *gateway*$_G$
`authService`	`AuthService`	Valutazione permessi edit config
`alerts`	`Signal<Alerts[]>`	Lista alert correnti
`gatewayOptions`	`Signal<string[]>`	*Gateway*$_G$ disponibili nel filtro
`from / to`	`Signal<string>`	Intervallo temporale locale (Rome)
`gatewayIds`	`Signal<string[]>`	Filtro *gateway*$_G$ selezionato
`canEditAlertsConfig`	`boolean`	True solo per `tenant_admin`

Tabella 50: Campi di AlertListPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Carica opzioni *gateway*$_G$ e alert iniziali
`applyFilter(form)`	`(form: AlertFilterFormValue): void`	Applica filtri e ricarica alert
`resetFilter()`	`(): void`	Ripristina finestra 24h e ricarica

Tabella 51: Metodi pubblici di AlertListPageComponent

12.2. AlertConfigPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Carica config alert e *gateway*$_G$ disponibili
`saveDefault(payload`$_G$`)`	`(payload`$_G$`: DefaultTimeoutPayload): void`	Aggiorna timeout default *tenant*$_G$
`saveGateway(payload`$_G$`)`	`(payload`$_G$`: GatewayTimeoutPayload): void`	Salva override timeout per *gateway*$_G$
`deleteGateway(gatewayId)`	`(gatewayId: string): void`	Rimuove override *gateway*$_G$
`toggleForm()/closeForm()`	`(): void`	Gestione apertura/chiusura modale di configurazione

Tabella 52: Metodi pubblici di AlertConfigPageComponent

Metodo	Firma	Comportamento
`getAlertsConfig()`	`(): Observable<AlertsConfig>`	Carica default + *gateway*$_G$ overrides
`setDefaultConfig(timeoutMs)`	`(timeoutMs: number): Observable<DefaultAlertsConfig>`	Aggiorna timeout default
`sendGatewayConfig(gatewayId, timeoutMs)`	`(...): Observable<GatewayAlertsConfig>`	Imposta override *gateway*$_G$

`deleteGatewayConfig(gatewayId)`	`(gatewayId: string): Observable<void>`	Cancella override *gateway*$_G$
`getAlerts(filter)`	`(af: AlertsFilter): Observable<Alerts[]>`	Supporta multi-*gateway*$_G$ con `forkJoin`, dedup e sort desc

Tabella 53: Metodi pubblici di AlertService

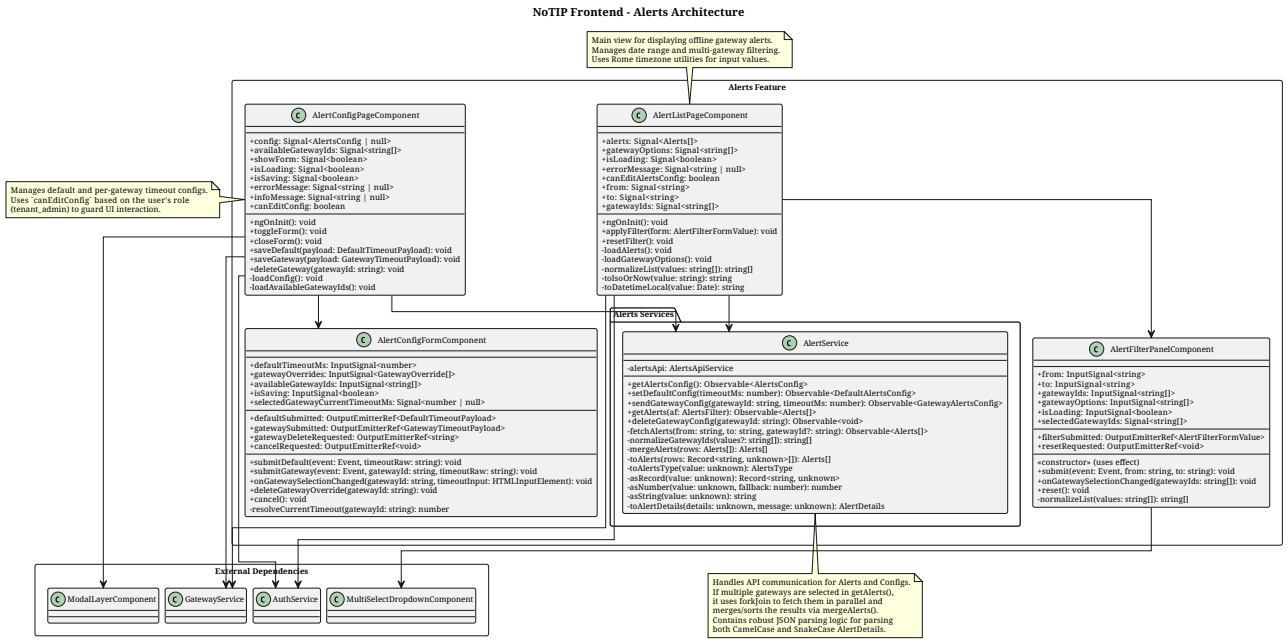

Figura 6: Diagramma della feature Alerts

13. Feature: Sensors

13.1. SensorListPageComponent

Elemento	Tipo/Firma	Comportamento
`sensors`	`Signal<Sensor[]>`	Lista sensori caricata
`gatewayFilters / typeFilters`	`Signal<string[]>`	Filtri attivi per *gateway*$_G$ e tipo
`gatewayOptions / sensorTypeOptions`	`Computed<string[]>`	Opzioni deduplicate e ordinate
`filteredSensors`	`Computed<Sensor[]>`	Filtro combinato su *gateway*$_G$ + tipo
`ngOnInit()`	`(): void`	Carica sensori
`openSensorDetail(id)`	`(sensorId: string): void`	Naviga a `/sensors/:id`
`onClearFilters()`	`(): void`	Reset filtri

Tabella 54: Campi e metodi pubblici di SensorListPageComponent

13.2. SensorDetailPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Legge `sensorId` da route, carica dettaglio e avvia stream *telemetria*$_G$

`ngOnDestroy()`	`(): void`	Ferma stream correnti e invalida run id

Tabella 55: Metodi pubblici di SensorDetailPageComponent

Metodo	Firma	Comportamento
`getAllSensors(refreshMs?)`	`(refreshMs = 10000): Observable<Sensor[]>`	Polling periodico sensori o fetch singolo
`getGatewaySensors(id, refreshMs?)`	`(id: string, refreshMs = 10000): Observable<Sensor[]>`	Polling/fetch sensori filtrati per *gateway*$_G$

Tabella 56: Metodi pubblici di SensorService

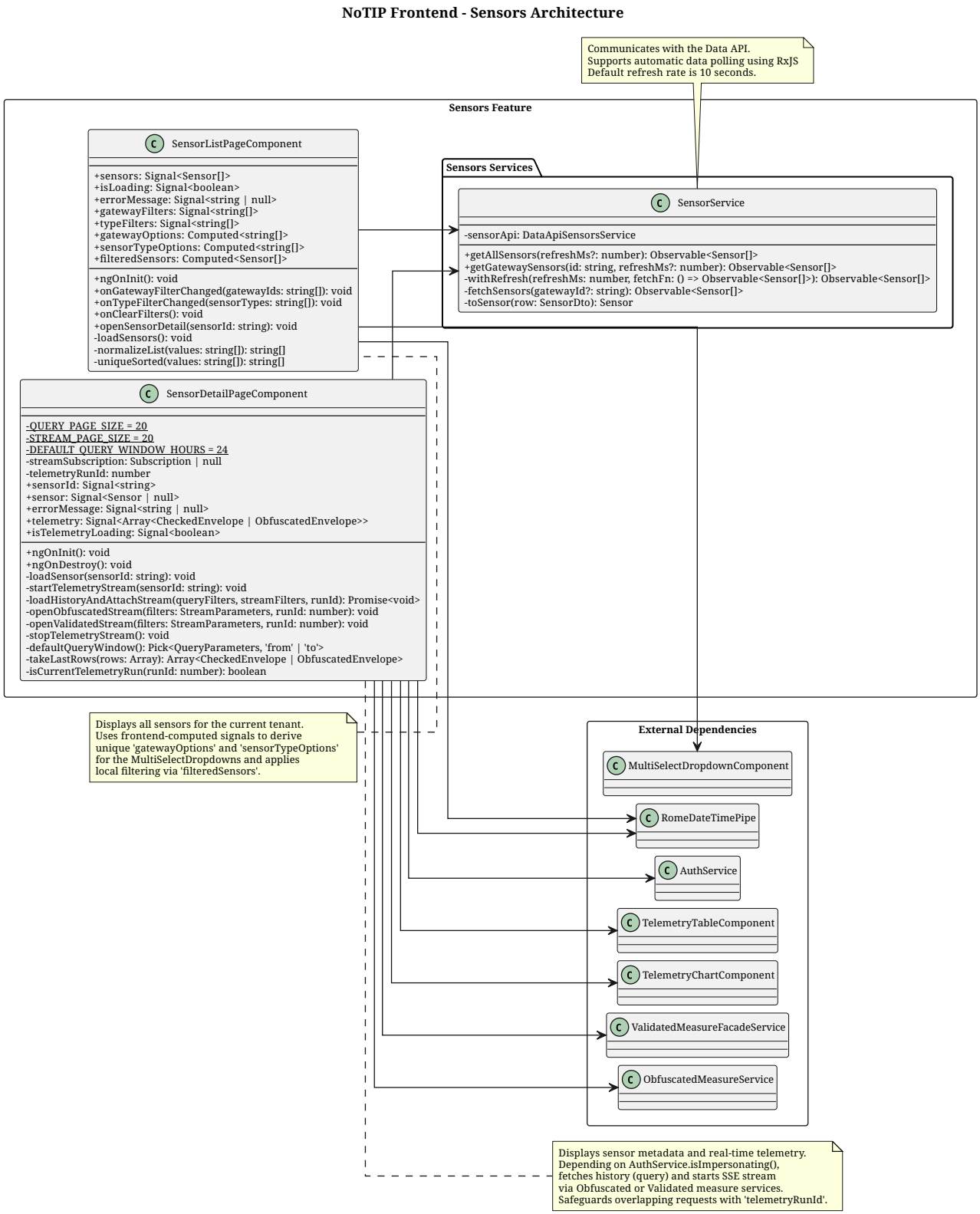

Figura 7: Diagramma della feature Sensors

14. Feature: Management (*Tenant*$_G$ Admin)

Le funzionalità di gestione *tenant*$_G$ sono accessibili ai `tenant_admin`.

14.1. UserListPageComponent

Elemento	Tipo/Firma	Comportamento
`users`	`Signal<ViewUser[]>`	Lista utenti *tenant*$_G$
`editingUserId / deletingUserId`	`Signal<string | null>`	Stato modali edit/delete
`editingUser`	`Computed<ViewUser | null>`	Utente in editing
`ngOnInit()`	`(): void`	Carica utenti
`createUser(payload`$_G$`)`	`(payload`$_G$`: CreateUserPayload): void`	Crea utente e ricarica lista
`updateUser(payload`$_G$`)`	`(payload`$_G$`: UpdateUserPayload): void`	Aggiorna utente e ricarica lista
`confirmDelete()`	`(): void`	Delete utente selezionato via `deleteUsers([id])`

Tabella 57: Campi e metodi pubblici di UserListPageComponent

Metodo	Firma	Comportamento
`getUsers()`	`(): Observable<ViewUser[]>`	Recupera utenti *tenant*$_G$ e mappa `lastAccess`
`createUser(params)`	`(up: UserParameters): Observable<CreatedUser>`	Crea utente con normalizzazione username e ruolo
`updateUser(id, params)`	`(userId: string, u: UpdateUserParameters): Observable<UpdatedUser>`	Aggiorna utente
`deleteUsers(ids)`	`(userIds: string[]): Observable<DeleteUserFeedback>`	Delete bulk con feedback deleted/failed

Tabella 58: Metodi pubblici di UserService

14.2. ThresholdSettingsPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Carica soglie e opzioni sensori/tipi
`saveTypeThreshold(payload`$_G$`)`	`(payload`$_G$`: TypeThresholdPayload): void`	Salva soglia per tipo
`saveSensorThreshold(payload`$_G$`)`	`(payload`$_G$`: SensorThresholdPayload): void`	Salva override soglia sensore
`confirmDeleteThreshold()`	`(): void`	Elimina soglia per tipo o sensore

Tabella 59: Metodi pubblici di ThresholdSettingsPageComponent

14.3. ApiClientListPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Carica lista client

`createClient(name)`	`(name: string): void`	Crea credenziali client e salva `lastCreated`
`confirmDelete()`	`(): void`	Elimina client selezionato
`toggleCreateForm()`	`(): void`	Apre/chiude form di creazione

Tabella 60: Metodi pubblici di ApiClientListPageComponent

Metodo	Firma	Comportamento
`getClients()`	`(): Observable<Client[]>`	Recupera lista client API *tenant*$_G$
`createClient(name)`	`(name: string): Observable<SecretClient>`	Crea client con secret iniziale
`deleteClient(clientId)`	`(clientId: string): Observable<void>`	Cancella client

Tabella 61: Metodi pubblici di ClientsService

14.4. AuditLogPageComponent

Metodo	Firma	Comportamento
`ngOnInit()`	`(): void`	Carica log iniziali
`applyFilters(form)`	`(form: AuditFilterFormValue): void`	Applica filtri e ricarica log
`resetFilters()`	`(): void`	Reset intervallo 24h e filtri
`onExportLogs()`	`(): void`	Export CSV lato client dei log correnti

Tabella 62: Metodi pubblici di AuditLogPageComponent

Metodo	Firma	Comportamento
`getLogs(filter)`	`(lf: LogsFilter): Observable<Logs[]>`	Fetch *audit*$_G$ log con join CSV di `userId/actions` e mapping *payload*$_G$

Tabella 63: Metodi pubblici di AuditService

14.5. CostDashboardPageComponent

Elemento	Firma	Comportamento
`CostDashboardPageComponent.ngOnInit()`	`(): void`	Carica costi *tenant*$_G$
`CostDashboardPageComponent.loadCosts()`	`private`	Aggiorna `costs`, `isLoading`, `errorMessage`
`CostsService.getTenantCosts()`	`(): Observable<Costs>`	Recupera `storage_gb` e `bandwidth_gb` con coercion numerica

Tabella 64: Metodi pubblici di CostDashboardPageComponent e CostsService

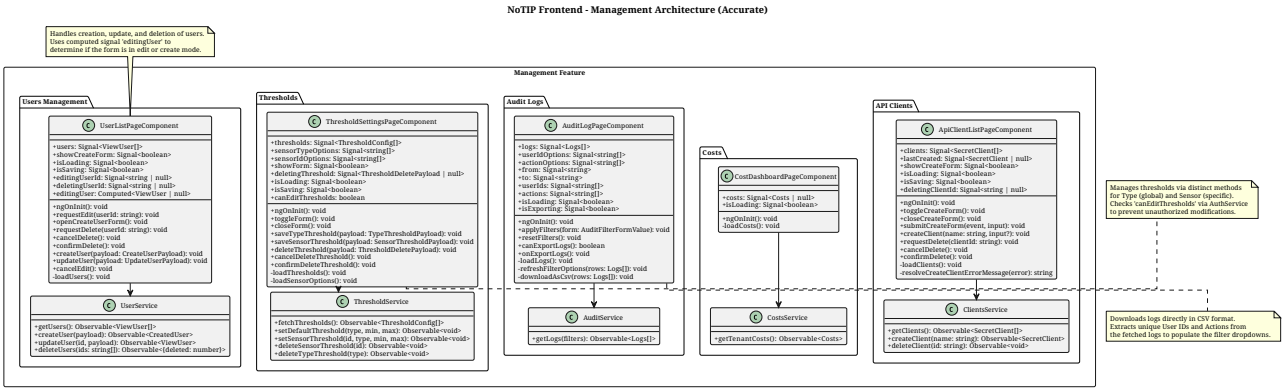

Figura 8: Diagramma della feature Management

15. Componenti shared

I componenti shared sono riutilizzati trasversalmente nell'applicazione.

Componente	Responsabilità
SidebarComponent	Navigazione laterale principale con voci di menu role-based. Mostra/rende disponibili le voci di menu in base al ruolo dell'utente (`tenant_user`, `tenant_admin`, `system_admin`). Durante l'impersonazione, mostra il banner e il bottone «Stop impersonation». Include ProfileSection e LogoutButton.
ModalLayerComponent	Overlay modale generico con backdrop. Utilizza elemento `<dialog>` HTML con proiezione `<ng-content>`. Supporta chiusura su click backdrop con rilevamento del target (previene chiusura accidentale su click all'interno).
MultiSelectDropdownComponent	Dropdown multi-selezione con ricerca integrata. Supporta apertura/chiusura tramite click su documento e tasto Escape. Opzioni filtrate per ricerca. Rimozione selezioni incompatibili. Stato disabled.
StatusBadgeComponent	Badge colorato per indicazione stato. Mappa valori di stato a classi CSS: `is-good` (verde), `is-warn` (giallo), `is-bad` (rosso), `is-neutral` (grigio).
DeleteConfirmModalComponent	Modale di conferma cancellazione con titolo e messaggio personalizzabili. Bottone confirm (pericolo, rosso) e cancel (ghost). Stato busy durante l'operazione.
ImpersonationTagComponent	Tag statico «OBFUSCATED MODE» visualizzato in contesti di dati offuscati.
ProfileSectionComponent	Sezione profilo utente con visualizzazione username e ruolo. Link opzionali per «Open profile» e «Change password».
LogoutButtonComponent	Bottone logout che emette evento click. Delega il logout ad AuthService.

Tabella 65: Componenti shared principali

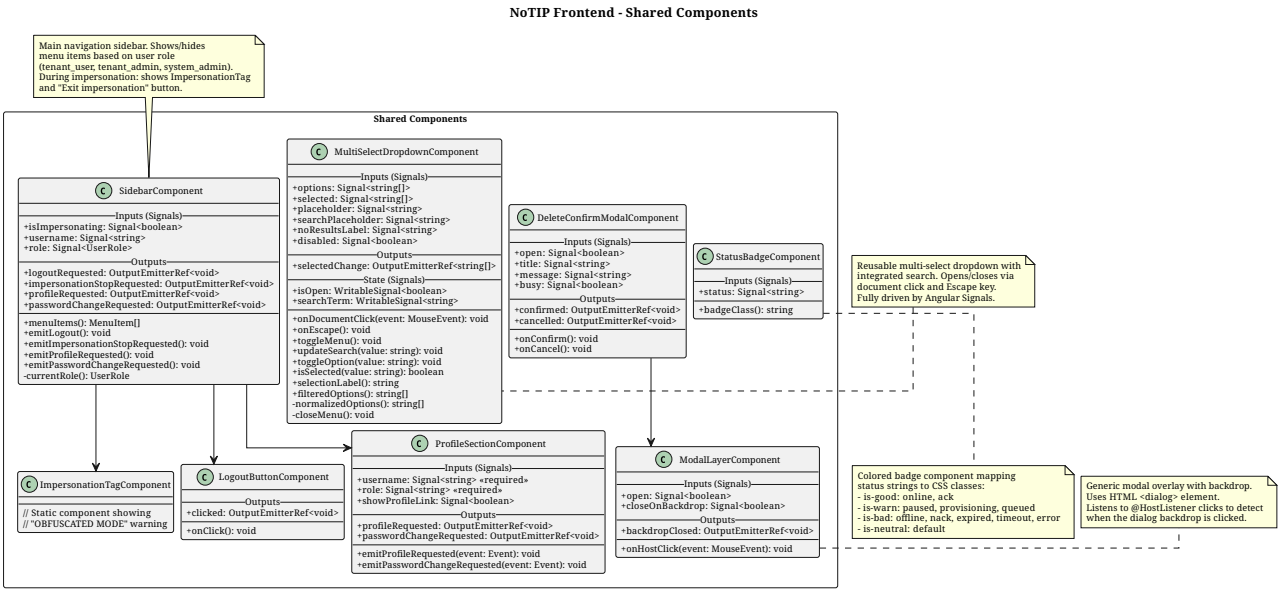

Figura 9: Diagramma dei componenti shared

16. Navigazione e sidebar

La sidebar è il componente di navigazione principale. La struttura delle voci di menu è organizzata per ruolo:

Voce	Ruoli Visibili	Path
Dashboard$_G$	`tenant_user`, `tenant_admin`	`/dashboard`$_G$
Gateways	`tenant_user`, `tenant_admin`	`/gateways`
Sensors	`tenant_user`, `tenant_admin`	`/sensors`
Alerts	`tenant_user`, `tenant_admin`	`/alerts`
Users	`tenant_admin`	`/mgmt/users`
Limits	`tenant_user`, `tenant_admin`	`/mgmt/limits`
API Clients	`tenant_admin`	`/mgmt/api`
Audit$_G$ Log	`tenant_admin`	`/mgmt/logs`
Costs	`tenant_admin`	`/mgmt/costs`
Tenants	`system_admin`	`/admin/tenants`
Gateways (Admin)	`system_admin`	`/admin/gateways`

Tabella 66: Voci di menu per ruolo

La configurazione alert non è una voce di sidebar dedicata: è accessibile dal link inline presente nella pagina di lista alert.

17. Testing

La strategia di testing del *frontend*$_G$ si articola su due livelli principali, con soglia di copertura target dell'80%.

17.1. Unit Test

I test unitari sono implementati con **Vitest** + **jsdom** + **@analogjs/vite-plugin-*angular*$_G$** e coprono:

- **Guards**: AuthGuard (inizializzazione *Keycloak*$_G$, token presence, login redirect, threshold prefetch trigger), RoleGuard (role matching, redirect per system_admin e altri ruoli, home redirect);
- **Interceptors**: authInterceptor (token injection, empty token passthrough), errorInterceptor (401 con retryUrl, 403, non-HTTP errors);
- **Core Services**:
  - `AuthService`: login/logout flows, token retrieval, *JWT*$_G$ decoding, role mapping, impersonation start/stop, sessionStorage restoration;
  - `ThresholdPrefetchService`: immediate + periodic fetch, admin/*tenant*$_G$ guards, error resilience, logout cleanup, duplicate start prevention;
  - `ThresholdService`: fetch/map/cache thresholds, type inference, refresh, null bounds, cached bound reuse, deletion;
  - `MeasureBoundsEvaluationService`: no match -> false, below-min -> true, above-max -> true, no bounds -> false, sensorId prioritized over sensorType;
  - `ObfuscatedStreamManagerService`: missing token -> login, valid envelopes emitted, malformed messages -> errors, *handshake*$_G$ failures, token-expired events, empty payloads ignored;
- **Feature Services**:
  - `GatewayService`: list loading, detail loading, obfuscated payloads, status normalization (uppercase/camelCase), paused/offline mapping, send frequency defaults, name update, deletion;
  - `CommandService`: config/firmware sending, disallowed status filtering, invalid field omission, polling until terminal status, 304 handling, 404/503 mapping, error rethrow, missing command ID rejection;
  - `DecryptedMeasureService`: query mapping, stream emission, export, empty filters, *SDK*$_G$ error propagation, stream abortion, no-store fetch configuration, close safety;
  - `ObfuscatedMeasureService`: stream mapping to obfuscated batches, query page mapping with/without nextCursor, array-wrapped response handling, closeStream delegation;
  - `ValidatedMeasureFacadeService`: stream/query/export mapping to checked envelopes, cursor preservation;
- ***Dashboard*$_G$ Components**:
  - `DataDashboardPageComponent`: stream mode (20-row cap), query mode with cursor pagination, impersonation forcing obfuscated endpoints and disabling export;
  - `FilterPanelComponent`: filter emission, 24h window clamping (from/to), clear/reset, stream mode ignoring dates, single-click toggle, search filtering, dropdown open/close, outside click dismissal, cross-filter dependency, incompatible selection removal;
  - `TelemetryChartComponent`: unique sensor ID computation, dataset creation, chart update reuse, invalid timestamp fallback sorting, format/withAlpha edge cases, obfuscated-only detection;
  - `TelemetryTableComponent`: obfuscated value masking, finite value formatting, non-finite n/a fallback, row-alert class application;

- ***Gateway*$_G$ Components**: GatewayCard (detail emission, disabled state), GatewayActions (all four event emissions), GatewayRenameModal (close event, trimmed submit), CommandModal (close, config full/partial/empty payloads, trimmed firmware submit);
- **Shared Components**: DeleteConfirmModal, MultiSelectDropdown, StatusBadge, ImpersonationBanner, ecc.

17.2. Configuration

La configurazione di test è definita in `vitest.config.ts`:

- Threshold di copertura: **80%** su branches, functions, lines, statements;
- Ambiente: **jsdom** per simulazione DOM;
- Plugin: **@analogjs/vite-plugin-*angular*$_G$** per compilazione *template*$_G$ *Angular*$_G$;
- Setup: configurazioni specifiche per test di componenti con TestBed *Angular*$_G$.

18. Design rationale

Di seguito le decisioni architetturali chiave e le relative motivazioni:

Decisione	Motivazione
Architettura Standalone (no NgModules)	*Angular*$_G$ 14+ introduce il modello standalone che semplifica la struttura dell'applicazione. Ogni componente, servizio e pipe è auto-contenuto con import espliciti. Questo riduce il boilerplate, migliora il tree-shaking e rende il codice più leggibile e manutenibile.
Signal-based state management (no NgRx)	I segnali *Angular*$_G$ forniscono state management reattivo nativo senza librerie esterne. Per uno stato locale di feature (es. lista *gateway*$_G$, *gateway*$_G$ selezionato, loading state), i segnali sono sufficienti e più semplici di NgRx. RxJS è mantenuto per flussi asincroni (*SSE*$_G$, HTTP) e Subject per comunicazione cross-componente.
OpenAPI Generator per client API	La generazione automatica dei client *Angular*$_G$ da spec OpenAPI garantisce strong typing, consistenza con le API *backend*$_G$, e aggiornamento automatico quando le API cambiano. Lo script `generate-openapi.sh` automatizza il recupero delle specifiche e la generazione.
SSE$_G$ per streaming *telemetria*$_G$	*Server-Sent Events*$_G$ (*SSE*$_G$) è preferita a *WebSocket*$_G$ per lo streaming di *telemetria*$_G$ perché: è unidirezionale (server -> client), adatta al caso d'uso; gestita nativamente dal browser; più semplice da implementare e debuggare; compatibile con i proxy HTTP e i meccanismi di autenticazione basati su header.
Decrittografia *telemetria*$_G$ lato client	La *telemetria*$_G$ viene crittografata *AES-256*$_G$*-GCM*$_G$ lato *gateway*$_G$ e decrittografata lato client tramite `@notip/crypto-sdk`$_G$. Questo approccio (Rule Zero) garantisce che il server non veda mai i dati in chiaro, preservando la privacy end-to-end. Il *frontend*$_G$ riceve `encryptedData`, `iv`$_G$, `authTag` e li decodifica localmente.
Data mode: clear vs obfuscated	Durante l'impersonazione, il *frontend*$_G$ opera in modalità `obfuscated`: i dati non vengono decrittografati e vengono

	mostrati solo *metadati*$_G$ (gatewayId, sensorId, timestamp). Questo protegge la privacy dei dati del *tenant*$_G$ durante il debug e il supporto da parte dei system admin.
Impersonation via sessionStorage	Il token di impersonazione è memorizzato in `sessionStorage` (non `localStorage`) per isolamento di sicurezza: viene cancellato alla chiusura del tab/browser e non persiste tra sessioni. Lo stato viene ripristinato in modo trasparente dal servizio auth leggendo il contesto salvato.
Cross-filter dependency nel FilterPanel	I dropdown del filtro sono interdipendenti: la selezione di un *gateway*$_G$ aggiorna i sensori disponibili. Questo previene selezioni incompatibili e migliora l'esperienza utente. Le selezioni incompatibili vengono rimosse automaticamente.
24h query window limit	Le query di *telemetria*$_G$ sono limitate a una finestra temporale di 24 ore per prevenire query troppo onerose sul database *TimescaleDB*$_G$. Il FilterPanel applica clamping automatico sia sul «from» che sul «to».
Rome timezone conversion	Tutti i timestamp sono convertiti nel fuso orario Europa/Roma tramite pipe custom (`RomeDateTimePipe`) e utility dedicate. Questo assicura coerenza temporale per gli utenti italiani.
Chart.js per visualizzazione dati	Chart.js è leggero, ben mantenuto e sufficiente per i grafici lineari richiesti dalla *dashboard*$_G$. Un dataset per sensore con colorazione distinta permette di distinguere facilmente le serie temporali. Il cap a 120 punti per dataset previene degrado delle performance.
Vitest over Karma/ Jasmine	Vitest è più veloce di Karma/Jasmine, supporta nativamente HMR, e si integra meglio con Vite (il bundler di *Angular*$_G$). jsdom fornisce un ambiente DOM headless sufficiente per i test di componenti.

Tabella 67: Decisioni architetturali del *frontend*$_G$

19. Error handling

La gestione degli errori nel *frontend*$_G$ segue questi principi:

Tipo Errore	Gestione
401 Unauthorized	L'errorInterceptor reindirizza a `/error?reason=unauthorized&retryUrl=<currentUrl>`. L'utente può effettuare il re-login e tornare alla pagina precedente tramite il `retryUrl`. Il token *Keycloak*$_G$ scaduto innesca il rinnovo automatico prima della scadenza.
403 Forbidden	L'errorInterceptor reindirizza a `/error?reason=forbidden`. Indica che l'utente non ha i permessi necessari per la risorsa richiesta.
Errori di rete / offline	Propagati come `HttpErrorResponse` con `status: 0`. Gestiti dai componenti feature con visualizzazione di messaggio di errore all'utente.

Errori *SSE*$_G$ (*handshake*$_G$, token expired)	L'ObfuscatedStreamManagerService termina lo stream con errore esplicito e abort della sessione *SSE*$_G$. In caso di token mancante all'apertura, viene richiesto il login.
Errori decrittografia *SDK*$_G$	Propagati da `DecryptedMeasureService` come errori Observable. Il componente *dashboard*$_G$ cattura l'errore e visualizza un messaggio appropriato.
Errori comando timeout (404, 503)	Il CommandService mappa `404` e `503` a `CommandStatus.timeout`. Il polling termina dopo 5 minuti di timeout con notifica all'utente.
Errori soglia cache fallback	Il `ThresholdPrefetchService` mantiene attivo lo scheduler anche in presenza di errori temporanei. La validazione utilizza lo stato cache corrente; se non sono presenti soglie, restituisce `false` (nessun out-of-bounds).
Error page generica	ErrorPageComponent visualizza messaggi di errore parametrizzati (`reason`, `retryUrl`). Supporta: `unauthorized`, `forbidden`, `not-found`. La rotta wildcard `**` reindirizza a `/error?reason=not-found`.

Tabella 68: Strategia di gestione errori

20. Osservabilità e metriche

Sebbene il *frontend*$_G$ non esponga *endpoint*$_G$ *Prometheus*$_G$, contribuisce all'osservabilità del sistema tramite:

- ***Keycloak*$_G$ *audit*$_G$ log**: ogni azione utente è registrata negli *audit*$_G$ log del sistema tramite il management-api;
- **Error tracking**: gli errori HTTP (401, 403, 5xx) sono registrati negli *audit*$_G$ log con contesto utente;
- ***Audit*$_G$ log impersonazione**: le azioni eseguite in modalità impersonazione hanno il nome utente offuscato per privacy, ma sono tracciate con timestamp e azione.

21. Sicurezza

Le misure di sicurezza implementate nel *frontend*$_G$ includono:

Misure	Dettaglio
PKCE$_G$ S256	L'autenticazione *Keycloak*$_G$ utilizza *PKCE*$_G$ (Proof Key for Code Exchange) con metodo S256 per prevenire attacchi di authorization code interception.
Auto-refresh token	Il token *JWT*$_G$ viene rinnovato automaticamente ogni 10 minuti tramite `withAutoRefreshToken` di `keycloak`$_G$`-angular`$_G$. I servizi `AutoRefreshTokenService` e `UserActivityService` monitorano l'attività utente e gestiscono il rinnovo. Il rinnovo fallito innesca il logout e il re-login.
SessionStorage per impersonazione	Il token di impersonazione è memorizzato in `sessionStorage`, non `localStorage`. Viene cancellato alla chiusura del tab e non persiste tra sessioni.

Role-based routing	Le guardie AuthGuard e RoleGuard prevengono l'accesso a rotte non autorizzate. I system admin vengono reindirizzati a `/admin/tenants`, gli altri a `/dashboard`$_G$.
Bearer token$_G$ injection	L'authInterceptor inietta il token *JWT*$_G$ in ogni richiesta HTTP. Le chiamate senza token sono permesse solo per *endpoint*$_G$ pubblici.
Impersonation guardrails	Durante l'impersonazione: la rinomina *gateway*$_G$ è nascosta in UI, le chiamate protette con `BlockImpersonationGuard` sono bloccate dal *backend*$_G$, la *telemetria*$_G$ è offuscata.
Whitelist validation	I client OpenAPI generati applicano validazione automatica sui *payload*$_G$ in ingresso/uscita. I form *Angular*$_G$ utilizzano validazione built-in (required, minlength, maxlength, pattern).
CSP (Content Security Policy)	Applicata dal reverse proxy Nginx che serve il *frontend*$_G$. Limita le sorgenti di script, stili e connessioni.

Tabella 69: Misure di sicurezza del *frontend*$_G$

22. Performance considerations

Aspetto	Strategia
Streaming *SSE*$_G$ cap	Lo stream *SSE*$_G$ è limitato a 20 righe nella *dashboard*$_G$ per prevenire accumulo di dati in memoria e degrado del rendering.
Chart.js dataset cap	I dataset Chart.js sono limitati a 120 punti per sensore per mantenere performance di rendering fluide.
Cursor-based pagination	Le query di *telemetria*$_G$ utilizzano pagination con cursore composito `(time, sensorId)` invece di offset per query più efficienti su *TimescaleDB*$_G$.
Threshold caching	Le soglie di validazione sono cached e aggiornate ogni 5 minuti dal ThresholdPrefetchService. In caso di errore temporaneo, il ciclo di refresh resta attivo senza interrompere la UI.
304 Not Modified caching	Il CommandService gestisce risposte 304 durante il polling dei comandi per evitare elaborazioni ridondanti.
Standalone tree-shaking	L'architettura standalone *Angular*$_G$ permette migliore tree-shaking rispetto ai NgModules, riducendo il bundle size finale.
Lazy loading rotte	Le rotte sono attualmente caricate in modo eager. La struttura feature-sliced è compatibile con una futura introduzione del lazy loading per ridurre ulteriormente il bundle iniziale.

Tabella 70: Considerazioni sulle performance