UNPLI Backend API

Sistema Laravel multi-tenant che espone API private per la gestione dei contenuti turistici e un'API pubblica read-only basata su token di accesso con scope.

Indice

Requisiti

- PHP 8.2+
- MySQL 8 (o database compatibile)
- Composer
- Node.js (per asset build, se necessario)

Setup locale

```
composer install
cp .env.example .env
# Configurare DB, cache, mail etc. in .envphp artisan key:generate
php artisan migrate --seed# oppure per rigenerare da zero con database popolato completo# php artisan migrate:fresh --seed
```

Gestione multi-tenant e domini

- La gerarchia è **tenant → domain → contenuti**. Ogni record gestito (venue, evento, store, ecc.) contiene `tenant_id` e `domain_id`.

- Il trait `App\Support\Concerns\BelongsToTenant` applica un global scope sul tenant corrente e valorizza automaticamente `tenant_id` in fase di creazione; `App\Support\Concerns\BelongsToDomain` replica la logica per `domain_id`, con fallback al tenant quando necessario.
- Ruoli supportati:
  - `superadmin`: accesso globale; impostare gli header `X-Tenant-Id` (obbligatorio per lavorare su un tenant) e, facoltativamente, `X-Domain-Id` per selezionare il dominio.
  - `admin`: controllo sull'intero tenant; può limitarsi a un dominio specifico tramite `X-Domain-Id`.
  - `user`: operatore di dominio, vincolato al proprio `domain_id`.
- Non sono previsti switch tra tenant o tra domini: ciascun utente appartiene a un unico tenant e a un unico dominio.

Contesto applicativo & middleware

- `App\Support\TenantContext` mantiene l'ID tenant per la durata di una richiesta; `App\Support\DomainContext` mantiene l'ID dominio (e replica il tenant).
- Il middleware `tenant.context` popola i contesti in base all'utente autenticato e pulisce gli scope a fine richiesta. Gli admin di tenant possono indicare `X-Domain-Id`; i superadmin scelgono il tenant da `X-Tenant-Id` e, opzionalmente, il dominio.
- Il middleware `public.token` valida i token dell'API pubblica, imposta i contesti e impone l'accesso in sola lettura; `public.scope` verifica gli scope dichiarati nel token.
- I modelli che usano i trait `BelongsToTenant` / `BelongsToDomain` applicano automaticamente i filtri anche sulle query Eloquent dirette (utile per test e comandi Artisan: ricordarsi di impostare i contesti prima di eseguire operazioni manuali).

Autenticazione & Sicurezza

Auth privata

- Login via `POST /api/auth/login` (richiede email/password). Restituisce personal access token Sanctum.
- Logout via `POST /api/auth/logout` (invalida il token corrente).
- Recupero password tramite `POST /api/auth/forgot-password` e `POST /api/auth/reset-password`.
- Cambio password self-service tramite `POST /api/auth/change-password` (richiede token e password attuale).
- Tutti gli endpoint privati usano `Bearer {{tenant_access_token}}` e sono protetti dai middleware:

- `auth:sanctum`
- `tenant.context`
- `throttle:api` (rate limiter globale 60 req/min per utente/IP)

API pubblica

- Accesso in sola lettura mediante token generati in `api_tokens`.
- Autenticazione via header `Authorization: Bearer <TOKEN>`.
- Ogni token è legato obbligatoriamente a un tenant (`tenant_id`) e può opzionalmente specificare un `domain_id`:
  - se `domain_id` è `null`, l'accesso riguarda tutte le entità del tenant;
  - se `domain_id` è valorizzato, le query restituiscono solo i contenuti di quel dominio (grazie al global scope `BelongsToDomain`).
- Formato risposta: default JSON; aggiungi `?format=xml` a qualsiasi rotta pubblica per ottenere XML.
- Ricerca per tag trasversale: `GET /api/public/search/tags?tag=<valore>` (rispetta gli scope del token).
- Middleware applicati:
  - `public.token` verifica token e abilita il tenant.
- `public.scope:<scope>` limita l'accesso alle risorse consentite.
- `throttle:api`.

Scope disponibili

Scope	Endpoint pubblico abilitato
`venues:read`	`/api/public/venues`
`events:read`	`/api/public/events`
`itineraries:read`	`/api/public/itineraries`
`routes:read`	`/api/public/routes`
`stores:read`	`/api/public/stores`
`restaurants:read`	`/api/public/restaurants`
`accommodations:read`	`/api/public/accommodations`
`promotions:read`	`/api/public/sales-promotions`
`media:read`	`/api/public/media-resources`

Riferimenti tra modelli (glossario E015)

Gli ID passati nei payload vengono validati sul tenant/dominio corrente. I principali campi di relazione che puoi popolare lato API private sono:
- `events`: `contact_ids`, `media_resource_ids`, `venue_ids`, `itinerary_ids` (+ `schedule` / `schedule_id`, **una sola schedule per evento**: quella indicata sostituisce eventuali altre)
- `itineraries`: `contact_ids`, `media_resource_ids`, `venue_ids`, `route_ids`, `event_ids` (+ `schedule` / `schedule_id`)
- `routes`: `contact_ids`, `media_resource_ids`, `itinerary_ids`, `venue_ids`, `event_ids` (+ `schedule` / `schedule_id`)
- `venues`: `contact_ids`, `media_resource_ids`, `event_ids`, `itinerary_ids`
- `stores`: `contact_ids`, `media_resource_ids`, `sales_promotion_ids`
- `restaurants` / `accommodations`: `contact_ids`, `media_resource_ids`
- `sales-promotions`: `contact_ids`, `media_resource_ids`, `store_ids` (oltre a `store_id` singolo)
- `itinerary steps`: `venue_id` / `event_id` / `route_id` singoli + `venue_ids`, `route_ids` (pivot aggiuntivi)
- Tag/categorie nei payload delle entità accettano forma localizzata (`tags.it[]` / `tags.en[]`, `categories.it[]` / `categories.en[]`); se invii un array semplice verrà replicato su entrambe le lingue.

API E015 aperta (senza token)

- Endpoint: `/api/e015/{addresses|geolocations|contacts|venues|events|itineraries|itinerary-steps|routes|stores|restaurants|accommodations|sales-promotions|media-resources|schedules|event-occurrences|recurrences}`.
- Parametri: `tenant_id` (obbligatorio), `domain_id` (opzionale).
- Formato: XML di default; `?format=json` per JSON.
- Filtri: temporali (`from`, `to`, `updated_since`, `period=last_24h|last_7d`), per data (`date_from|date_start`, `date_to|date_end` su `issue_datetime` se presente), spaziali (`lat` / `lon` / `radius<=50km` oppure bbox `minLat` / `minLon` / `maxLat` / `maxLon`), semantici (`status`, `categories`), testuali (`q`), per ID (`id=...`).
- Best practice: passare sempre `tenant_id` e un filtro temporale (default `last_24h`) per ridurre il payload.

Seeder di esempio

`php artisan migrate:fresh --seed` crea un dataset coerente con gli XSD E015:
- Tenant principale **UNPLI** con due domini (*Pro Loco Carate Brianza* e *Pro Loco Besana*).
- Per ogni dominio: 1 utente admin e 1 utente standard (password di default `password`).
- Token pubblico (`api_tokens`) con tutti gli scope attivi.
- Inoltre vengono creati automaticamente: un token pubblico “globale” (solo `tenant_id`) e un token dedicato per ciascun dominio, così da poter testare i due scenari di filtro.
- Contenuti completi (venue, eventi, itinerari, percorsi, negozi, ristoranti, strutture ricettive, promozioni) con campi multilingua (`it` / `en`), metadati amministrativi, riferimenti incrociati (`event_ids`, `route_ids`, ...) e payload complessi (es. `accessibility`, `product_promotions`). I master data di supporto (indirizzi, geolocalizzazioni, contatti) includono un campo `label`

per identificarli più rapidamente nelle relazioni.
- Token pubblico salvato nella tabella `api_tokens` (campo `token` generato casualmente).

Credenziali di esempio:
- Admin Carate Brianza: `admin@carate-brianza.unpli.test`
- User Carate Brianza: `user@carate-brianza.unpli.test`
- Admin Besana: `admin@besana.unpli.test`
- User Besana: `user@besana.unpli.test`
- Superadmin backoffice: `superadmin@teknet.it` (headers `X-Tenant-Id` obbligatorio e `X-Domain-Id` opzionale per impostare il contesto)
- Password (tutti): `password`

Seeder minimale (solo superadmin):
- `php artisan db:seed --class=SuperAdminOnlySeeder` crea solo l'utenza `superadmin@teknet.it` (password `password`) senza altri dati.

Guida operativa

1. Preparazione

- Configura `.env`, poi `php artisan migrate:fresh --seed` per una base dati coerente.
- Avvia l'app (`php artisan serve` o Sail) e importa `postman/unpli-api.postman_collection.json`.
- Nel workspace Postman imposta `base_url`, `superadmin_email`, `password_default`. Per gli endpoint E015 aperti usa anche `e015_tenant_id` (e facoltativamente `e015_domain_id`).

2. Accesso come superadmin

- Esegui la richiesta **Auth → Login - Superadmin** per ottenere `tenant_access_token`.
- Chiama **Tenant & Amministrazione → Lista tenant (superadmin)**: salva l'ID del tenant che vuoi gestire.
- Recupera i domini del tenant scelto con **Tenant & Amministrazione → Domini tenant (superadmin)** e salva `domain_id` per gli header successivi (quando necessari).
- Per le richieste successive aggiungi gli header:
  - `X-Tenant-Id: <id>` (obbligatorio); `X-Domain-Id: <id dominio>` solo se serve.

3. Gestione utenti

- Superadmin/admin: usa `GET /api/users` per elencare gli utenti nel tenant corrente.
- Crea operatori con `POST /api/users` indicando `role` (`admin` / `user`); il `domain_id` viene ereditato dal contesto o dai dati inviati.
- Per disattivare un account imposta `is_active=false` tramite `PUT /api/users/{id}`.

4. Token pubblici

- Crea token con `POST /api/api-tokens` specificando `scope[]` e, se necessario, `domain_id`.
- La `DELETE /api/api-tokens/{id}` disabilita il token (`enabled=false`); usa `PUT` per riattivarlo.
- Usa `DELETE /api/api-tokens/{id}/force` per eliminare definitivamente il record.
- I token restituiti nella sezione seed sono utili per testare subito le rotte pubbliche (`/api/public/*`).

5. Operazioni dominio/tenant

- Admin tenant: può cambiare dominio in corsa impostando l'header `X-Domain-Id`; senza header opera a livello tenant (dati cross-dominio).
- Utente di dominio: lavora solo nel proprio dominio, gli header extra vengono ignorati.
- Dopo ogni lavoro via Artisan/Tinker ricordati di settare i contesti manualmente:

```
TenantContext::setTenantId($tenantId);DomainContext::set($domainId, $tenantId); // opzionale// ... operazioni ...DomainContext::clear();TenantContext::clear();
```

6. Verifiche rapide

- `GET /api/admin/tenants` (superadmin) → verifica accesso e conteggio domini.
- `GET /api/admin/tenants/{tenant}/domains` (superadmin) → recupera elenco domini di un tenant e ID da riutilizzare negli header.
- `GET /api/public/events` con token pubblico → controlla il filtro dominio/tenant.
- `GET /api/e015/events?tenant_id=<id>&format=json` → verifica la risposta aperta (senza token) per il tenant indicato.
- `GET /api/me` → conferma ruolo e contesti caricati dal middleware.

7. Integrazione con applicazioni terze (token pubblico)

Scenario tipico: una web app o una piattaforma partner vuole mostrare eventi e contenuti UNPLI senza autenticarsi come tenant.

1. **Provisioning**
   - Crea o recupera un token in `api_tokens` con gli scope necessari (es. `events:read`, `itineraries:read`).
   - Se la terza parte deve vedere solo i contenuti di un dominio, valorizza `domain_id`; altrimenti lascialo `null` per copertura tenant.

- Condividi con il partner il token e le risorse disponibili, ricordando che le API sono read-only.

2. **Invocazione**
   - HTTP base:

     ```
     GET https://<host>/api/public/events?per_page=20
     Authorization: Bearer <TOKEN>
     ```

   - Per ricevere XML invece di JSON aggiungi `?format=xml` (es. `/api/public/events?format=xml`).
   - Ricerca trasversale per tag: `/api/public/search/tags?tag=storia` (opzionale `limit=...`, supporta `format=xml`).
   - Per i dettagli usa `/api/public/<resource>/{id}`. Le risposte includono JSON ricco (asset, relazioni, campi multilingua `it` / `en`).
3. **Gestione errori**
   - `401` → token mancante o errato.
   - `403` → scope insufficiente (`public.scope` blocca la rotta).
   - `429` → superato rate limit (default 60 req/min per token/IP).
4. **Rotazione**
   - Quando necessario disattiva il token con `DELETE /api/api-tokens/{id}` (salvo `enabled=false`).
   - Riattivalo con `PUT /api/api-tokens/{id}` impostando `enabled=true`.

Suggerimento: per ambienti pubblici usa un sottodominio dedicato (es. `api.unpli.it`) e limita gli scope al minimo indispensabile.

API Private (tenant)

1. Autenticazione

Metodo	URI	Descrizione	Payload	Note
POST	`/api/auth/login`	Login tenant	`{ email, password, device_name? }`	Ritorna token, `tenant_id` / `domain_id` e oggetti `tenant` / `domain` se presenti
POST	`/api/auth/logout`	Logout	-	Richiede `Authorization`

Metodo	URI	Descrizione	Payload	Note
POST	`/api/auth/change-password`	Cambio password utente autenticato	`{ current_password, password, password_confirmation }`	Richiede token; invalida la password precedente
POST	`/api/auth/forgot-password`	Richiede reset link	`{ email }`	Usa mailer configurato
POST	`/api/auth/reset-password`	Completa reset	`{ token, email, password, password_confirmation }`	Token inviato via email

2. User info & Tenant

Metodo	URI	Descrizione
GET	`/api/me`	Info utente corrente + tenant
GET	`/api/tenant`	Dettagli tenant
PUT	`/api/tenant`	Aggiorna dati tenant
GET	`/api/admin/tenants`	(solo superadmin) lista tenant
POST	`/api/admin/tenants`	(solo superadmin) crea tenant
PUT	`/api/admin/tenants/{tenant}`	(solo superadmin) aggiorna tenant
GET	`/api/admin/tenants/{tenant}/domains`	(superadmin o admin di quel tenant) lista domini del tenant
POST	`/api/admin/tenants/{tenant}/domains`	(superadmin o admin di quel tenant) crea dominio
PUT	`/api/admin/tenants/{tenant}/domains/{domain}`	(superadmin o admin di quel tenant) aggiorna dominio
DELETE	`/api/admin/tenants/{tenant}/domains/{domain}`	(superadmin o admin di quel tenant) elimina dominio
GET	`/api/domains`	(utente autenticato) domini del tenant corrente (context o tenant_id dell'utente)

3. Risorse CRUD (tutte `apiResource` senza `create/edit`)

> Payload sempre JSON. Le risposte includono i campi del modello e relazioni quando previste.
> Le chiamate GET /api/<resource> accettano parametri query per ogni campo previsto in creazione (es. ?label=Sede principale&city=Roma), con ricerca case-insensitive sui campi testuali; i campi multilingua fanno match su it ed en.

Risorsa	Endpoint base	Note principali
Users	`/api/users`	campi `name`, `email`, `password`, `role`, `is_active`, `phone` (opzionale), `domain_id` (opzionale, può essere null per utente tenant-wide)
API Tokens	`/api/api-tokens`	Campi `name`, `token?`, `scope[]`, `enabled`; `DELETE` disattiva (imposta `enabled=false`)
Addresses	`/api/addresses`	Dati civici con `label` opzionale per identificare l'indirizzo; rilegati da venue/store
Geolocations	`/api/geolocations`	Coord `x_coord`, `y_coord`, `altitude`, ecc., con `label` opzionale
Contacts	`/api/contacts`	`value`, `type` (phone/email) e `label` opzionale
Media Resources	`/api/media-resources`	`label` opzionale, multilingua `name` / `description`, `url`, `thumbnail_url`, contenuti base64
Tags	`/api/tags`	Dizionario tag deduplicato per tenant/dominio, campo `name` univoco per contesto
Categories	`/api/categories`	Dizionario categorie deduplicato per tenant/dominio, campo `name` univoco per contesto
Venues	`/api/venues`	tutti i campi previsti da `venueType` (metadati amministrativi, accessibilità, relazioni) + `address_id` / `geolocation_id`
Events	`/api/events`	full schema `eventType`: `scope`, `coverage`, `participants`, `price_info`, ecc.; schedule opzionale (`schedule` inline oppure `schedule_id`); `schedules` con `event_occurrences` e `recurrences` caricati
Itineraries	`/api/itineraries`	schema `itineraryType` con durate, contatti, servizi, riferimenti correlati; schedule opzionale (`schedule` inline oppure `schedule_id`); `schedules` con `event_occurrences` e `recurrences` caricati
Routes	`/api/routes`	schema `routeType`: durate, distanze, travel means, riferimenti cross-entity; schedule opzionale (`schedule` inline oppure `schedule_id`); `schedules` con `event_occurrences` e `recurrences` caricati
Stores	`/api/stores`	schema `storeType`: dati fiscali, commerciale, accessibilità, brand, servizi
Restaurants	`/api/restaurants`	eredita da store + `cuisine`, `diet`, `award_michelin`
Accommodations	`/api/accommodations`	schema `accomodationType`: stelle, stanze, disponibilità camere, checkin/out
Sales Promotions	`/api/sales-promotions`	schema `salePromotionType` con `product_promotions`, metadati e calendario
Classifications	`/api/classifications`	GET con tag e categorie deduplicate (per risorsa o combinato), filtrate per tenant/domain

Risorsa	Endpoint base	Note principali
Schedules	`/api/schedules`	`name` obbligatorio, `schedulable_type` (`event|itinerary|route`), `description` (JSON), `start_datetime`, `end_datetime`. Il tipo usa alias brevi (es. `itinerary`, non `App\Models\Itinerary`).

Nota schedule: per `events`, `itineraries` e `routes` puoi specificare una nuova schedule nel payload (blocco `schedule`), riutilizzare una esistente con `schedule_id`, oppure omettere la schedule quando non serve.

Rotte superadmin

- **GET** `/api/admin/tenants` → lista paginata dei tenant (solo utenti con ruolo `superadmin`)
- **GET** `/api/admin/tenants/{tenant}/domains` → elenco domini del tenant selezionato (solo `superadmin`)

Per ogni risorsa:
- **GET** `/resource` → lista paginata (`per_page` max 100)
- **POST** `/resource` → crea
- **GET** `/resource/{id}` → dettaglio
- **PUT** `/resource/{id}` → aggiorna
- **DELETE** `/resource/{id}` → elimina

4. Endpoint nidificati

Itinerary Steps

- GET `/api/itineraries/{itinerary}/steps`
- POST `/api/itineraries/{itinerary}/steps` (campi: `position`, `venue_id?`, `event_id?`, `route_id?`, `note` JSON)
- GET/PUT/DELETE `/api/itineraries/{itinerary}/steps/{step}`

Schedule Occurrences

- GET `/api/schedules/{schedule}/occurrences`
- POST `/api/schedules/{schedule}/occurrences` (`start_datetime`, `end_datetime`, `break_start?`, `break_end?`)
- GET/PUT/DELETE `/api/schedules/{schedule}/occurrences/{occurrence}`

Schedule Recurrences

- GET `/api/schedules/{schedule}/recurrences`
- POST `/api/schedules/{schedule}/recurrences` (`frequency`, `count`, `start_time`, `end_time`, ecc.)
- GET/PUT/DELETE `/api/schedules/{schedule}/recurrences/{recurrence}`

API Pubbliche (read-only)

Prefix `/api/public`, tutte le rotte sono `GET` e richiedono:
- Header `Authorization: Bearer {{public_api_token}}`
- Scope coerente (vedi tabella).

Risorsa	Endpoint	Query	Risposta
Venues	`/api/public/venues`	`per_page`	Lista venue (con relazioni principali)
Events	`/api/public/events`	`per_page`	Lista eventi con `schedules` + occorrenze/ricorrenze
Itineraries	`/api/public/itineraries`	`per_page`	Itinerari con steps e `schedules` + occorrenze/ricorrenze
Routes	`/api/public/routes`	`per_page`	Percorsi con `schedules` + occorrenze/ricorrenze
Stores	`/api/public/stores`	`per_page`	Negozi con restaurant/accommodation
Restaurants	`/api/public/restaurants`	`per_page`	Solo lettura
Accommodations	`/api/public/accommodations`	`per_page`	Solo lettura
Sales Promotions	`/api/public/sales-promotions`	`per_page`	Promozioni collegate allo store
Media Resources	`/api/public/media-resources`	`per_page`	Multimedia JSON

Ogni risorsa esponde anche `/api/public/<resource>/{id}` per il dettaglio.

Copertura XSD → Database

Tutti i tipi complessi definiti dagli XSD E015 sono mappati nelle tabelle. Panorama sintetico:

XSD Type	Tabelle/Colonne principali
`venueType`	`venues`: `alt_id`, campi multilingua (`name`, `description`, `abstract`, `price_info`), tassonomie (`tags`, `categories`), metadati (`administrative_contact_point`, `contact_point`, `acknowledgements`, `ratings`, `available_services`), logistica (`capacity`, `owner`, `opening_times`, `event_ids`, `route_ids`, `related`, `related_venues`, `more_info`, `building_year`, `building_floors`, `last_renew_building_year`, `accessibility`), relazioni con `addresses`, `geolocations`, `contacts`, `media_resources`, `events` (pivot `event_venue`).
`eventType`	`events`: oltre ai campi base, `scope`, `coverage`, `contact`, `organizer`, `participants`, `attendees`, `capacity`, `target_audience`, `acknowledgements`, `ratings`, `price_info`, `available_services`, `related_events`, `more_info`, `administrative_contact_point`, `contact_point`; relazioni con venue, media, schedules, contacts.

XSD Type	Tabelle/Colonne principali
`itineraryType`	`itineraries`: sezioni descrittive estese (`scope`, `coverage`, `contact`, `organizer`, `acknowledgements`, `ratings`, `price_info`, `available_services`, `event_ids`, `related_itineraries`, `more_info`, `administrative_contact_point`, `contact_point`), durate min/max, passi (`itinerary_steps`), media e scheduling.
`routeType`	`routes`: `scope`, `coverage`, `contact`, `acknowledgements`, `ratings`, `available_services`, `travel_means`, `direction`, `event_ids`, `venue_ids`, `itinerary_ids`, `related_routes`, `more_info`, `administrative_contact_point`, `contact_point`.
`itineraryStepType`	`itinerary_steps`: supporto multilingua (`name`, `description`, `category`), durate min/max, riferimenti multipli (media, venue/event/route) e note strutturate.
`resourceType`	`media_resources`: campi multilingua, `thumbnail_url`, `binary_content`, `thumbnail_binary_content` per asset binari e anteprime.
`storeType` / `restaurantType` / `accomodationType`	`stores` gestisce identificativi, descrizioni, tassonomie, servizi, rating, dati fiscali/commerciali e logistica. `restaurants` aggiunge `award_michelin`, `diet`, `cuisine`. `accommodations` include disponibilità camere (`available_room`), stelle, letti e informazioni di checkin/out.
`salePromotionType` / `saleProductPromotionType`	`sales_promotions`: metadati (`issue_datetime`, `data_source`, `license`), contatti amministrativi, calendario e `product_promotions` JSON; relazioni con negozi e media.

Per i riferimenti inter-tabellari (`EventId`, `RouteId`, `VenueId`, `ItineraryId`, ecc.) sono disponibili sia colonne JSON (es. `event_ids`, `route_ids`) sia tabelle ponte. Le colonne multilingua seguono sempre la struttura JSON con chiavi `it` ed `en`, validate dal rule `RequiredLocales`.

Errori & Rate limiting

- Rate limit nome `api`: 60 richieste/minuto per utente autenticato o IP (per token pubblici).
- Risposte errore standard Laravel:
  - 401 `{"message":"Unauthenticated."}`
  - 403 `{"message":"This action is unauthorized."}` o personalizzati (`tenant context unavailable`, `token invalid`).
  - 422 per errori di validazione (payload JSON con `errors`).

Postman collection

Percorso: `postman/unpli-api.postman_collection.json`.
- Contiene tutte le rotte private e pubbliche con script di salvataggio token/ID.
- Variabili richieste:

- `base_url` (es. `http://localhost`)
- `e015_tenant_id` (obbligatorio per gli endpoint E015 aperti)
- `e015_domain_id` (facoltativo per gli endpoint E015 aperti)
- `tenant_access_token` (popolata dal login)
- `public_api_token` (token tenant-wide da `api_tokens`)
- `domain_public_api_token` (token pubblico legato a uno dei domini, facoltativo)
- Vari ID (address_id, venue_id, ecc.) assegnati automaticamente dagli script la prima volta.

Importare la collezione, lanciare **Auth → Login (tenant admin)** e poi testare le sezioni *Tenant API*, *Public API* e *E015 Public (no token)*.