Travio API: A Functional, Data, and Architectural Analysis

Internal Engineering

November 13, 2025

Abstract

This paper analyzes the Travio API (v2.7.0) from a functional and data modeling perspective. We formalize the authentication model, the generic repository surface under `/rest`, and the multi-step booking workflow under `/booking/search`. We propose a normalized pseudo-schema, outline typical CRUD and query patterns, and discuss error and security concerns. We then identify modernization opportunities and conclude with a best-practice, scalable reference architecture including CRM integration.

1 Introduction

The Travio API exposes a bearer-authenticated set of endpoints for travel operations: generic repositories (accounting, catalogs, anagraphics), and a provider-agnostic booking interface that orchestrates availability and selection across different service types (hotels, flights, activities, and more). The design follows an OpenAPI 3.1.0 specification with a consistent approach to pagination, filtering, sorting, and link unfolding.

2 Authentication

Endpoint: `POST /auth` with JSON body {`id:int`, `key:string`}. On success, the server returns a JWT token used in the `Authorization: Bearer` header for all protected routes. Failures yield `401 Unauthorized`. Tokens should be short-lived and revocable.

2.1 Intuition

The API separates credential exchange from data access; the stateless JWT supports horizontal scalability of the API tier and avoids server-side session persistence.

3 Generic Repositories

Endpoints: `/rest/repository` and `/rest/repository/id`. Repositories include entities such as *services*, *subservices*, *master-data*, *reservations*, *payments*, *invoices*, *journal-entries*, *amenities*, *geo*, etc. The interface supports:

- List with pagination (`page`, `per_page`), filtering (JSON-encoded `filters`), sorting (JSON-encoded `sort_by`), and link unfolding (`unfold`).
- Create with `data` and optional `options` (e.g., `allow_id`, `check_vat`).
- Get one by id; optionally unfold linked fields.
- Update with `data` subset; read-only fields are ignored.
- Delete by id.

3.1 Intuition

The repository pattern provides a unified surface over heterogeneous domains. Link-unfolding shifts N+1 joins to the API layer when needed. JSON-encoded filters provide flexibility without proliferating query params, at the cost of discoverability.

4 Booking Workflow

Endpoint: `POST /booking/search` allows one or more searches in a single request. Each search defines service `type`, date ranges or transport `segments`, and `occupancy`. The response returns a stateful context `(cart, search_id, step)` and a list of `groups`, each requiring an action of type `pick`, `confirm`, or `input`. The contract includes `pick_type`, `allow_partial`, and `final` gating.

4.1 Intuition

The booking process is a guided multi-step state machine. The service aggregates diverse supplier semantics into a consistent set of group actions, deferring complex pricing, inventory, and selection logic behind a uniform interface.

5 Data Model (Pseudo-Schema)

This section proposes a relational pseudo-schema to capture major entities and links implied by the API.

Core Catalog

```
SERVICES(id PK, type INT, code TEXT, name JSONB, classification_id FK, category_id FK,
         typology_id FK, visibility JSONB, geo JSONB, descriptions JSONB, images JSONB,
         video JSONB, amenities JSONB, stop_sales JSONB, schedule JSONB,
         estimated_price_per_pax NUMERIC, supplier_hotel_id FK,
         show_prices TEXT, accept_other_prices TEXT, meta JSONB, notes JSONB, tags INT[])

SUBSERVICES(id PK, service_id FK, code TEXT, name JSONB, type INT, obsolete BOOL,
             visibility JSONB, descriptions JSONB, images JSONB, video JSONB,
             availability JSONB, amenities JSONB, cost JSONB, price JSONB,
             occupancy JSONB, show_prices TEXT, accept_other_prices TEXT,
             meta JSONB, notes JSONB, tags INT[])

SERVICES_CATEGORIES(id PK, code TEXT, name TEXT)
SERVICES_TYPOLOGIES(id PK, code TEXT, name TEXT, type INT)
AMENITIES(id PK, type TEXT, name JSONB)
CLASSIFICATIONS(id PK, code TEXT, name TEXT, rating NUMERIC)
GEO(id PK, parent_id FK NULL, type TEXT, name JSONB, description JSONB, suppliers JSONB)
TAGS(id PK, visible BOOL, reservation_default BOOL, parent_id FK NULL, name JSONB)

PACKAGES(id PK, code TEXT, name JSONB, type_id FK, category_id FK,
         shown_price NUMERIC, min_pax INT, max_pax INT,
         descriptions JSONB, images JSONB, video JSONB,
         visibility JSONB, geo JSONB, duration INT, schedule JSONB,
         rows JSONB, attachments JSONB, guides INT[], meta JSONB, tags INT[])

DEPARTURES(id PK, package_id FK, date DATE)
```

Anagraphics and Commerce

```
MASTER_DATA(id PK, profiles TEXT[], categories INT[], profile_type TEXT,
            honorific_id FK, name TEXT, surname TEXT, company_name TEXT,
            legal_form_id FK, commercial_name TEXT, full_name TEXT GENERATED,
            tax_code TEXT, vat_country TEXT, vat_number TEXT, language TEXT,
            nationality TEXT, birth DATE, birth_place TEXT, gender TEXT,
            doc JSONB, username TEXT, password_hash TEXT, enabled BOOL,
            pec TEXT, sdi_code TEXT, extra_ue BOOL, public_administration BOOL,
            website TEXT, promoter_id FK, network_id FK,
            logo JSONB, photo JSONB, invoice_master_data_id FK,
            inbound_payments_master_data_id FK, outbound_payments_master_data_id FK,
            contacts JSONB, addresses JSONB, price_lists TEXT[], meta JSONB,
            notes JSONB, tags INT[])

RESERVATIONS(id PK, heading_id FK, year INT, num TEXT,
             from DATE, to DATE, status INT, status_history JSONB,
             client_id FK, invoice_client_id FK, payment_client_id FK,
             promoter_id FK, user_id FK, description TEXT, reference TEXT,
             requested_by TEXT, first_pax TEXT, due TIMESTAMP, date TIMESTAMP,
             cancellation_date TIMESTAMP NULL, confirmation_date TIMESTAMP NULL,
             price JSONB, instalments JSONB, services JSONB,
             attachments JSONB, departures JSONB, price_modifiers JSONB,
             accounting_entries JSONB, api_key INT, meta JSONB, notes JSONB, tags INT[])

PAX(id PK, reservation_id FK, name TEXT, surname TEXT, phone TEXT, email TEXT,
    gender TEXT, doc JSONB, address TEXT, city TEXT, province TEXT, postal_code TEXT,
    country_id FK, language TEXT, nationality TEXT, tax_code TEXT, age INT, birth DATE,
    birth_place TEXT, notes TEXT, meta JSONB, tags INT[])

PAYMENTS(id PK, method_id FK, number INT, date DATE, currency_id FK,
         master_data_id FK, type TEXT, exchange_rate NUMERIC,
         print_description TEXT, rows JSONB, journal_entries INT[], meta JSONB)

INVOICES(id PK, year INT, number INT, full_number TEXT, date DATE, due DATE,
         type_id FK, currency_id FK, master_data_id FK, reference_date DATE,
         reference_number TEXT, journal_entries INT[], rows JSONB, meta JSONB)

JOURNAL_ENTRIES(id PK, date DATE, number INT, full_number TEXT, accounting_reason_id FK,
                vat_protocol_number INT, vat_protocol_suffix TEXT,
                description TEXT, invoices INT[], payments INT[], rows JSONB, meta JSONB)

CURRENCIES(id PK, code TEXT, name TEXT, symbol TEXT, decimals INT)
PAYMENT_METHODS(id PK, name TEXT, gateway TEXT, credentials JSONB, test BOOL,
                visible_in BOOL, visible_out BOOL,
                accounting_reason_in_id FK, accounting_reason_out_id FK, meta JSONB)
VAT_TYPES(id PK, code TEXT, name TEXT, percentage NUMERIC, type TEXT,
         partially_deductible NUMERIC, e_invoice_type_out TEXT, e_invoice_type_in TEXT,
         legal_reference TEXT, stamp_duty BOOL)
ACCOUNTING_ACCOUNTS(id PK, number INT, full_number TEXT, name TEXT)
ACCOUNTING_REASONS(id PK, name TEXT)
```

6 Query and CRUD Patterns

Listing

```
-- Pagination
SELECT * FROM services ORDER BY code ASC OFFSET (page-1)*per_page LIMIT per_page;

-- Filtering (example: type=1 AND preferred=true)
SELECT * FROM services WHERE type = 1 AND preferred = true;
```

```
-- Unfolding links (server-side join when requested)
SELECT s.*, c.* AS category FROM services s
JOIN services_categories c ON c.id = s.category_id
WHERE ... ORDER BY ... LIMIT ...;
```

Create

```
-- Read-only columns are computed/ignored on write.
INSERT INTO master_data(name, surname, language, email) VALUES (...);
```

Update

```
UPDATE master_data SET language='en', email='...' WHERE id = :id;
```

Delete

```
DELETE FROM master_data WHERE id = :id;
```

7 Error Handling and Security

- Standard responses: 401 (unauthorized), 403 (forbidden), 404 (not found), 500 (error).
- JWT bearer for all protected endpoints. Tokens should be audience-scoped and time-limited.
- Validate JSON-encoded `filters`/`sort_by`; reject overly complex or unsafe operators.
- Enforce read-only fields server-side; ignore or reject on write.

8 Small Diagrams (Mermaid snippets)

The repository includes a `diagrams.mmd` file with detailed Mermaid diagrams. Example:

```
flowchart TD
  Client -->|POST /auth| Auth
  Auth -->|200 {token}| Client
  Auth -->|401| Client
```

9 Modernization Opportunities

- OpenAPI-first development: generate server stubs, typed SDKs, and contract tests in CI.
- Typed filters DSL: replace free-form JSON with a composable, validated filter grammar.
- Idempotency keys: for POST/PUT to ensure safe retries.
- Observability: distributed tracing, RED/SRE metrics, per-repo SLIs/SLOs.
- Caching: per-repository read caching; booking availability edge caching with TTLs.
- Outbox + Event streaming: decouple write-side from CRM and accounting sync.
- Policy-as-code: authorization policies per repository and field-level guards.
- Pagination defaults and hard caps; keyset pagination for hot paths.
- Rate limiting and abuse prevention per token and IP.

10 Best-Practice Scalable Rebuild (with CRM Integration)

Architecture

- API Gateway (JWT validation, rate limit, request shaping, OpenAPI validation).
- Services: Auth, Repositories (per-bounded-context modules), Booking Orchestrator.
- Data: Postgres (primary), Redis (cache, rate-limit), S3-compatible storage (files).
- Async: Kafka (or NATS) for events; Outbox pattern from Postgres.
- CRM Integration: Event-driven (e.g., `reservation.created`, `payment.recorded`). Dedicated consumer pushes to CRM via reliable connectors.
- Telemetry: OpenTelemetry traces/metrics/logs, dashboards per repo.

Data and Throughput

- Read hot-paths (e.g., `/rest/services`) behind cache with stale-while-revalidate.
- Writes go to DB; emit events via outbox + change data capture.
- Booking: circuit-breakers and bulkheads per provider; parallel fan-out with timeouts and hedged requests.

API Design

- Strongly typed SDKs; pagination and filter helpers; idempotency headers; consistent error model with codes.
- Field masks or sparse fieldsets to control payload size.
- Standardized expand/unfold semantics with maximum depth and limits.

Operations

- CI: OpenAPI breaking-change detection; contract/integration tests per repo.
- CD: canary releases; error budgets per endpoint; autoscaling based on p95 latency and RPS.

11 Conclusion

The Travio API provides a uniform surface for heterogeneous travel operations. By formalizing filters, strengthening contracts, and adopting an event-driven, observable architecture with idempotent writes and robust caching, the platform can scale to tens of thousands of daily operations while integrating cleanly with CRM systems.