<https://gemini.google.com/u/1/app/5b5f3092ace62463>

\* \* \* \* \*

Module 8: Platform Content - Overview

Version: 1.0

Date: May 18, 2025

This document provides a master recap and architecture overview for Module 8: Platform Content, part of the Via di Francesco Pilgrimage Platform database. It details how tables, views, translation mechanisms, security policies, and API considerations interlock for managing general platform content like news and articles.

1\. Executive Summary

Module 8 establishes the framework for creating, managing, and displaying dynamic content such as news, blog posts, and informational articles on the pilgrimage platform. It supports rich text content, multimedia integration (featured images and galleries), author attribution, associations with other platform entities (trails, regions, towns), and robust multilingual capabilities. The module is designed with clear audit trails and Role-Based Access Control (RLS) to ensure content integrity and appropriate access. Key outputs include structured data readily consumable by frontend applications via a well-defined API, including denormalized views for common read patterns.

The primary business goals unlocked by this module include:

- Enhanced user engagement through timely news, articles, and stories.

- A flexible system for content managers to publish rich, multimedia content.

- Multilingual presentation of articles to cater to a global audience.

- Clear attribution and content lifecycle management.

2\. Group-Level Snapshot

| Group | Key Tables & Views | Primary Purpose | Top Inter-Group Links (Illustrative) |

| 8\. Platform Content Module | Tables: &lt;br> `public.articles` (v1.0) &lt;br> `public.media\_roles\_master` (v1.0) &lt;br> `public.article\_media` (v1.0) &lt;br>&lt;br> Views: &lt;br> `public.published\_articles\_view` (v1.0) &lt;br> `public.article\_media\_details\_view` (v1.0) | Manages textual and multimedia articles, defines semantic roles for media within articles, and links articles to specific media items. Views provide convenient, denormalized access to published content and article media. | `articles` to `profiles` (Module 1 - for authors, audit) &lt;br> `articles` to `media` (Module 1 - for featured image) &lt;br> `article\_media` to `media` (Module 1) &lt;br> `articles`, `media\_roles\_master`, `article\_media` to `translations` (Module 1 - for multilingual content) |

3\. Narrative Walkthrough

This module centers around the `public.articles` table, which stores the core information for each piece of content, such as its title, slug, body, excerpt, author, and publication status. Textual fields in `articles` designated as translatable store their content in the platform's primary reference language (English), with other language versions managed in the `public.translations` table (Module 1).

To support rich multimedia content:

- `public.media\_roles\_master` defines a controlled vocabulary for the roles media items can play (e.g., `gallery\_image`, `inline\_image`). This table's descriptive fields are also translatable.

- `public.article\_media` is a linking table connecting an article in `public.articles` to one or more media items in `public.media` (Module 1). Each link is assigned a role via `media\_role\_code` (from `media\_roles\_master`) and includes a `display\_order`. This table also allows for `caption\_override` and `alt\_text\_override` for linked media, specific to its use in that article; these overrides are translatable.

- The `public.articles` table itself can link to a single `featured\_image\_media\_id` directly.

Two primary views facilitate data retrieval:

- `public.published\_articles\_view`: Provides a denormalized look at articles that are published and not deleted, joining with `profiles` for author information and `media` for featured image details (in the primary reference language).

- `public.article\_media\_details\_view`: Offers a comprehensive view of all media items linked to articles, including their resolved role names, media file details, and any text overrides (in the primary reference language).

Cleanup triggers are implemented on `articles`, `media\_roles\_master`, and `article\_media` to remove orphaned translation entries from `public.translations` upon deletion of parent records.

4\. Cross-Cutting Concerns

- Translations / i18n:

- Translatable fields in `articles` (`title`, `body\_content`, `excerpt`), `media\_roles\_master` (`default\_display\_name`, `default\_description`), and `article\_media` (`caption\_override`, `alt\_text\_override`) store primary reference language (English) text directly.

- All other language versions are stored in `public.translations` (Module 1), linked by `table\_identifier`, `column\_identifier`, and `row\_foreign\_key`.

- `AFTER DELETE` triggers on these three tables ensure orphaned translation entries are removed.

- Media Handling:

- `articles.featured\_image\_media\_id` provides a direct link for a primary article image.

- The `article\_media` table, in conjunction with `media\_roles\_master` and `public.media` (Module 1), facilitates gallery functionality and the use of multiple, contextually defined images per article. This includes leveraging `media.image\_variants\_json` for responsive images.

- Audit / Lifecycle Management:

- All tables in this module (`articles`, `media\_roles\_master`, `article\_media`) include standard audit columns: `created\_at`, `updated\_at` (with `extensions.moddatetime` trigger), `created\_by\_profile\_id`, and `updated\_by\_profile\_id` (referencing `public.profiles`).

- `articles` uses a `deleted\_at` column for soft deletion and an `article\_status` field (linking to `content\_visibility\_status\_enum`) for its lifecycle.

- `media\_roles\_master` uses an `is\_active` flag. `article\_media` links are typically hard-deleted if the parent article is deleted (via `ON DELETE CASCADE`).

5\. Security & Access Control 🔐

- Authentication Provider: Supabase Auth is assumed, leveraging JWTs with roles derived from `public.profiles.roles`.

- RLS Overview: Row-Level Security is applied to all tables:

- `public.articles`:

- Public users can `SELECT` published, non-deleted articles.

- Authenticated authors can `INSERT`, and `UPDATE`/`DELETE` their own articles if in 'draft' or 'pending\_review' status.

- Users with roles like 'admin\_platform' or 'regional\_content\_manager' have broader `ALL` permissions (potentially with additional checks for regional managers).

- `public.media\_roles\_master`:

- Public users can `SELECT` active roles.

- 'admin\_platform' users can `ALL` (CRUD).

- `public.article\_media`:

- `SELECT` access is typically tied to the ability to view the parent article.

- `INSERT`/`UPDATE`/`DELETE` permissions are tied to edit rights on the parent article.

- Helper Functions for RLS: Policies will leverage existing helper functions like `public.has\_role(TEXT)` and may require new specific helpers like `public.can\_edit\_article(article\_id BIGINT)`.

6\. API Endpoints Summary (Conceptual)

Key conceptual API endpoints for Module 8 include:

- `GET /v1/articles`: Lists published articles with filtering (tags, associated entities), sorting, and pagination. Supports `lang` parameter for localized content.

- `POST /v1/articles`: Creates a new article. Requires appropriate authentication and role.

- `GET /v1/articles/{slug}`: Retrieves detailed information for a single published article by its slug, including full content, author, featured image, and media gallery. Supports `lang` parameter.

- `GET /v1/media-roles`: Lists active media roles. Supports `lang` parameter.

- `POST /v1/media-roles`: Creates a new media role (admin only).

- `GET /v1/articles/{article\_slug}/media`: Lists all media items associated with a specific article. \*(Refer to the "Module 8 OpenAPI 3.1 Specification" for detailed definitions).\*

7\. Prerequisite Objects & Build Order (for Module 8 components)

Assumes Module 1 (`profiles`, `media`, `translations`, `languages\_master`) and global ENUMs (e.g., `content\_visibility\_status\_enum`) are already in place.

1. Table: `public.media\_roles\_master` (defines roles for media)

2. Table: `public.articles` (core content table)

3. Table: `public.article\_media` (links articles to media)

4. Triggers: Apply `updated\_at` and translation cleanup triggers to all three tables.

5. Views:

- `public.published\_articles\_view`

- `public.article\_media\_details\_view`

6. RLS Policies: Apply to all tables and potentially views.

7. Seed Data: Populate `public.media\_roles\_master` with initial roles.

8\. Performance & Optimization Extras

- Key Indexes: All tables have appropriate primary keys, foreign key indexes, and specific indexes for common query patterns (e.g., `articles.slug`, `articles.article\_status`, GIN index on `articles.tags`, composite index on `article\_media(article\_id, media\_role\_code, display\_order)`).

- Views: The `published\_articles\_view` and `article\_media\_details\_view` are designed to denormalize data for common read operations.

- Database Functions: For API endpoints requiring complex data assembly (especially with dynamic translations for multiple fields and nested objects), creating PostgreSQL functions that return JSONB is highly recommended to optimize performance and simplify API gateway logic.

9\. Visuals (Conceptual ERD - Module 8 Key Tables & Links)

Code snippet

```

erDiagram

profiles ||--o{ articles : "authored\_by"

profiles ||--o{ articles : "audit\_created\_by"

profiles ||--o{ articles : "audit\_updated\_by"

media ||--o{ articles : "featured\_image"

media\_roles\_master {

TEXT role\_code PK

TEXT default\_display\_name "Translatable"

BOOLEAN is\_active

-- audit columns

}

articles {

BIGINT id PK

TEXT title "Translatable"

TEXT slug UK

TEXT body\_content "Translatable"

UUID author\_profile\_id FK

UUID featured\_image\_media\_id FK

content\_visibility\_status\_enum article\_status

-- audit columns

-- association FKs (trail, region, town)

}

article\_media {

BIGINT id PK

BIGINT article\_id FK

UUID media\_id FK

TEXT media\_role\_code FK

INTEGER display\_order

TEXT caption\_override "Translatable"

-- audit columns

}

translations {

BIGINT id PK

TEXT table\_identifier

TEXT column\_identifier

TEXT row\_foreign\_key

TEXT language\_code FK

-- audit columns

}

articles ||--|{ article\_media : "has\_gallery\_item"

media ||--|{ article\_media : "is\_linked\_via"

media\_roles\_master ||--|{ article\_media : "defines\_role\_for"

articles ..> translations : "content\_translated\_in"

media\_roles\_master ..> translations : "role\_name\_translated\_in"

article\_media ..> translations : "override\_translated\_in"

profiles ||--o{ media\_roles\_master : "audit\_created\_by"

profiles ||--o{ media\_roles\_master : "audit\_updated\_by"

profiles ||--o{ article\_media : "audit\_created\_by"

profiles ||--o{ article\_media : "audit\_updated\_by"

```

10\. Critical Gaps & Risks (for Module 8 implementation)

- 🔴 Application Logic for Slug Generation: Robust, unique slug generation for `articles` needs to be handled at the application layer or via a database trigger.

- 🟠 Content Ingestion Workflow: The workflow for authors creating content in a non-primary-reference language (e.g., Italian) and how it gets into `public.articles` (English primary) and `public.translations` (Italian version) needs clear definition at the application level.

- 🟠 RLS Helper Functions: If complex RLS logic is required (e.g., `regional\_content\_manager` access tied to `articles.associated\_region\_id`), dedicated helper functions must be securely implemented and tested.

- 🟠 Translation Quality & Management: While the schema supports translations, the process for generating, reviewing, and updating translations is external to this DB module's DDL.

11\. Scalability & Future-Proof Notes

- The design with `articles`, `media\_roles\_master`, and `article\_media` provides a scalable solution for managing rich content.

- The use of `TEXT` for `body\_content` in `articles` allows for flexibility (Markdown, HTML).

- Consideration for Full-Text Search (FTS) on `articles.title`, `articles.body\_content`, `articles.excerpt`, and `articles.tags` is a V2+ enhancement for improved searchability.

- The module can be extended with other content types if needed in the future.

12\. Next Steps (for Module 8 implementation)

- P1 🔴 Implement table DDL for `media\_roles\_master`, `articles`, and `article\_media`.

- P1 🔴 Implement all defined indexes and constraints.

- P1 🔴 Implement `updated\_at` and translation cleanup triggers for all three tables.

- P1 🔴 Implement RLS policies for all three tables, including necessary helper functions if any.

- P1 🟢 Seed initial data for `public.media\_roles\_master`.

- P1 🔴 Create the database views: `public.published\_articles\_view` and `public.article\_media\_details\_view`.

- P2 🟠 Develop and test application-layer logic for slug generation and content ingestion/translation workflow.

- P2 🟠 Define and implement any complex RLS helper functions.

- P2 🔵 Develop API endpoints based on the OpenAPI specification, ideally leveraging database functions for complex read operations.