

Task Management

Documentation

Auteur : Tiogars

Copyright © Tiogars 2026

Table of Contents

1. Task Management 4
1.1 What You Will Run 4
1.2 Modules 4
1.3 Supported Stack Versions 4
1.4 Tutorial: Run the Solution 5
1.4.1 1. Start the stack 5
1.4.2 2. Create a Task 5
1.4.3 3. Use the console 5
1.4.4 4. Stop the stack 5
1.5 Develop and Verify 5
2. Architecture 7
2.1 Public Object and Persistence Entity 7
2.2 Invariants and Ownership 7
2.3 Schema Evolution 8
3. Task and Taxonomy API Reference 9
3.1 Resource Schemas 9
3.1.1 Shared Fields 9
3.1.2 Task Fields 9
3.1.3 Reusable Taxonomy Domains 9
3.2 Endpoints 10
3.2.1 Authentication and Administration 10
3.2.2 User Settings 11
3.2.3 Tasks 11
3.2.4 Tags and Nomenclatures 12
3.3 PDF Printing 12
3.4 Create 12
3.5 Read 13
3.6 Update 13
3.7 Delete 13
3.8 Persistence Auditing and Required Migration 13
3.9 Seed and Import 14
3.10 Import and Export Formats 14
3.11 Response 15
3.12 Clients 15
4. Web Console 16
4.1 Run the Console 16
4.2 Routes 16

4.3 View Mode 16
4.4 List Tasks 16
4.5 Create or Update a Task 16
4.6 Manage Tags and Nomenclatures 17
4.7 Delete a Task 17
4.8 View and Print 17
4.9 Import and Export 17
4.10 Coverage 18
5. Deployment 19
5.1 Services 19
5.2 Deploy with Dokploy 19
5.2.1 Console runtime configuration 20
5.3 Local Test of the Stack 20
5.4 Persistence Behavior 20
5.5 Reset the Demo Database 21
5.6 Publish the Flutter Applications 21
5.7 Build Documentation and Slides 21
5.8 Production Considerations 21
6. Recommended Copilot Skills 22
6.1 Core Project Work 22
6.2 Java and Spring Boot 22
6.3 Frontend and TypeScript 22
6.4 Dart Client 22
6.5 Documentation, Diagrams, and Presentations 23
6.6 Delivery and Operations 23
6.7 Suggested Defaults 23

1. Task Management

This monorepo implements task management from persistence to user interface. A Spring Boot API stores tasks, reusable tags, and hierarchical nomenclatures in PostgreSQL, three reusable clients expose the same contract, and a React console provides day-to-day CRUD and task bulk data operations.

1.1 What You Will Run

The tutorial starts a complete local stack and creates one task. Tasks have a server-generated `Long` identifier, unique immutable `code`, required `label`, optional `description`, server-owned audit fields, status (`TODO` by default), priority (`NORMAL` by default), optional due date and assignee, tag IDs (`[]` by default), and an optional nomenclature ID.

Tags provide reusable flat labels. Nomenclatures provide a reusable tree through nullable `parentId`; tasks select one classification and any number of tags. The API rejects missing references, hierarchy cycles, and deletion of referenced taxonomy entries.

1.2 Modules

Module	Purpose
`api`	Spring Boot CRUD API backed by JPA and PostgreSQL.
`java-client`	Maven client library using Java `HttpClient`.
`react-client`	TypeScript client library for React or browser apps.
`backend`	React + Vite management console.
`dart-client`	Dart package for Flutter or Dart services.
`docs`	MkDocs project documentation.
`presentation`	Marp overview of the solution and its architecture.

See Recommended Copilot Skills for the skills to use when extending or maintaining each part of the monorepo.

1.3 Supported Stack Versions

Use these maintained baseline versions for local development and CI. Newer compatible patch releases are supported unless a module pins a version explicitly.

Technology	Version
Node.js	24
pnpm	11
Java	25
Maven	3.9.16
Dart SDK	3.12.2
Spring Boot	4.1.1
Vite	8.2.2
Vitest	4.1.11
Vite React plugin	6.1.1
Redux Toolkit / RTK Query	2.12

1.4 Tutorial: Run the Solution

1.4.1 1. Start the stack

From the `template-docker` repository:

On a new volume, PostgreSQL applies the task-domain database migration automatically; apply it manually on an existing database.

```
cp .env.example .env   # local login: admin / change-me
docker compose -f docker-compose.yml -f compose.local.yml up -d --wait
```

The local override publishes the API on `8989` and the console on `5173`.

1.4.2 2. Create a Task

Log in first (with `.env.example`, `admin` / `change-me`) and use the returned token:

```
TOKEN=$(curl -s -X POST http://localhost:8989/api/v1/auth/login \
    -H "Content-Type: application/json" \
    -d '{"username":"admin","password":"change-me"}' | jq -r .token)

curl -X POST http://localhost:8989/api/v1/task \
    -H "$(printf 'Authorization: %s %s' Bearer "$TOKEN")" \
    -H "Content-Type: application/json" \
    -d '{"code":"RELEASE-001","label":"Prepare release","description":"Review the release checklist"}'
```

The response resembles:

```
{
    "id": 1,
    "code": "RELEASE-001",
    "label": "Prepare release",
    "description": "Review the release checklist",
    "status": "TODO",
    "priority": "NORMAL",
    "dueDate": null,
    "assignedTo": null,
    "tagIds": [],
    "nomenclatureId": null,
    "createdDate": "2026-08-30T17:00:00Z",
    "lastModifiedDate": "2026-08-30T17:00:00Z",
    "createdBy": "admin",
    "lastModifiedBy": "admin"
}
```

The exact identifier and timestamps are assigned by the server.

1.4.3 3. Use the console

Open `http://localhost:5173`, select **Tasks**, and open the new task by its ID. From this page you can create, inspect, update, delete, seed, import, export, and print tasks. Use **Tags** and **Nomenclatures** to maintain reusable labels and classifications before assigning them to tasks.

Swagger UI is available at `http://localhost:8989/swagger-ui.html`.

1.4.4 4. Stop the stack

```
docker compose -f docker-compose.yml -f compose.local.yml down
```

The PostgreSQL volume is retained. Follow Reset the demo database when you need a clean database.

1.5 Develop and Verify

```
make setup
make test
make build
```

Generate Java, React, and Dart coverage reports:

```
make coverage
```

Run only the API locally against PostgreSQL on `localhost:5432`:

```
make java-run
```

Then open `http://localhost:8080/swagger-ui.html`.

Continue with the API reference, architecture explanation, web console guide, or deployment guide.

2. Architecture

```
flowchart LR
    Console[React Vite management console] --> ReactClient[React client library]
    ReactClient --> Api[Spring Boot API]
    JavaClient[Java client library] --> Api
    DartClient[Dart client library] --> Api
    Api --> Task[Task service /api/v1/task]
    Api --> Tag[Tag service /api/v1/tag]
    Api --> Nomenclature[Nomenclature service /api/v1/nomenclature]
    Task --> Tag
    Task --> Nomenclature
    Nomenclature --> Parent[Optional parent nomenclature]
    Task --> Database[(PostgreSQL)]
    Tag --> Database
    Nomenclature --> Database
    Docs[MkDocs] -. documents .-> Api
    Slides[Marp presentation] -. explains .-> Docs
```

The repository implements task management and two reusable taxonomy domains. HTTP controllers validate transport requests and delegate business operations to services; repositories isolate Spring Data JPA access. The Java, TypeScript, and Dart clients depend on the public HTTP contract, not on persistence classes. The application namespace stays `fr.tiogars.domaintemplate`; it is not the business-domain name.

2.1 Public Object and Persistence Entity

`Task` is the public business object returned by controllers and represented by each client. It carries identity (`id`, immutable `code`), `label`, nullable `description`, status, priority, nullable due date and assignee, tag IDs, an optional nomenclature ID, and server-owned audit timestamps and actors.

`Tag` is a reusable flat categorization domain. `Nomenclature` is a reusable hierarchical classification domain with an optional `parentId`. Both have generated identity, immutable code, mutable label and description, and the same server-owned audit fields as tasks. A task can reference many tags and one nomenclature without embedding their mutable definitions.

Internal mutable JPA entities handle persistence concerns such as generated identity, column constraints, and dirty tracking. Controllers never expose JPA entities.

The service maps between these types. This boundary keeps the public contract stable when persistence details change and prevents JPA behavior from leaking into client code.

2.2 Invariants and Ownership

Concern	Owner	Behavior
Persistent identity	Server and database	`id` is a generated `Long`.
Business identity	Service and database	`code` is required, unique, and immutable.
Mutable task data	Client request	Required `label`; nullable description, due date, assignee, and nomenclature; tags default to `[]`.
Workflow defaults	Service	Status defaults to `TODO`; priority defaults to `NORMAL`.
Taxonomy integrity	Service and database	Referenced IDs must exist; nomenclature parents cannot form cycles; referenced taxonomy deletion is rejected.
Audit metadata	Server auditing	Dates are `Instant` values; actors are authenticated username snapshots.
Bulk reconciliation	Service	Seed and import upsert by `code` and ignore imported IDs and dates.

2.3 Schema Evolution

Apply `src/main/resources/db/manual/task-domains.sql` from `template-java-api` before starting this version on an existing database; the Dokploy stack runs it automatically on a new volume. See the migration instructions. The older `domain-template-auditing-soft-delete.sql` script documents the legacy domain's auditing migration.

Hibernate is configured with `spring.jpa.hibernate.ddl-auto: update` for this runnable application. It does not migrate legacy business data or remove unmapped tables; existing volumes may retain `domain_templates` or `domain_records`.

This convenience setting is not a migration strategy for production. Use versioned database migrations in a real service. For the demo, reset the Compose volume when a clean schema is required, as described in the deployment guide.

3. Task and Taxonomy API Reference

The API manages tasks, reusable tags, and hierarchical nomenclatures at the singular paths `/api/v1/task`, `/api/v1/tag`, and `/api/v1/nomenclature`. Swagger UI is published at `/swagger-ui.html`.

All business endpoints require JWT authentication. Obtain a token with `POST /api/v1/auth/login`, then send it as `Authorization: Bearer <token>`. The forced administrator account is configured with `APP_ADMIN_USERNAME` and `APP_ADMIN_PASSWORD` at API startup.

3.1 Resource Schemas

3.1.1 Shared Fields

Tasks, tags, and nomenclatures share these identity, business, and audit fields.

Field	JSON type	Required	Rules
`id`	number	Response only	Server-generated Java `Long`.
`code`	string	Yes	Unique among active rows and immutable.
`label`	string	Yes	Non-blank and mutable.
`description`	string or `null`	No	Mutable.
`createdDate`	string	Response only	Server-owned `Instant` in ISO-8601 format.
`lastModifiedDate`	string	Response only	Server-owned `Instant` in ISO-8601 format.
`createdBy`	string	Response only	Authenticated username at creation.
`lastModifiedBy`	string	Response only	Authenticated username at the last persisted change.

3.1.2 Task Fields

Field	JSON type	Rules
`status`	string	`TODO`, `IN_PROGRESS`, `DONE`, or `CANCELLED`; default `TODO`.
`priority`	string	`LOW`, `NORMAL`, `HIGH`, or `URGENT`; default `NORMAL`.
`dueDate`	string or `null`	Nullable calendar date in `YYYY-MM-DD` format, without a timezone.
`assignedTo`	string or `null`	Nullable free-text assignee; not a user foreign key.
`tagIds`	number array	Existing tag IDs (`Long[]`); default `[]`.
`nomenclatureId`	number or `null`	Nullable existing nomenclature ID.

3.1.3 Reusable Taxonomy Domains

`Tag` provides flat labels that can be attached to multiple tasks, such as `RELEASE` or `DOCUMENTATION`. It has only the shared fields.

`Nomenclature` provides a reusable classification hierarchy, such as **Engineering → Release → Documentation**. It has the shared fields plus nullable `parentId` (a server-generated `Long` referencing another nomenclature). `null` denotes a root. Self-parenting and longer cycles are rejected.

Task tag/nomenclature references and nomenclature parent references must exist. Deleting a tag used by a task, a nomenclature used by a task, or a nomenclature with children is rejected rather than silently removing references. Invalid references and hierarchy cycles return `400 Bad Request`; deleting a referenced taxonomy entry returns `409 Conflict`.

Task and taxonomy reads require `ADMIN` or `USER`; writes require `ADMIN`.

3.2 Endpoints

3.2.1 Authentication and Administration

Method	Path	Description
`POST`	`/api/v1/auth/login`	Exchange username/password for a JWT.
`GET`	`/api/v1/auth/me`	Read the authenticated user.
`GET`	`/api/v1/auth/users`	List users. Requires `ADMIN`.
`POST`	`/api/v1/auth/users`	Create a user with a BCrypt-stored password. Requires `ADMIN`.
`GET`	`/api/v1/auth/users/{id}`	Read a user. Requires `ADMIN`.
`PUT`	`/api/v1/auth/users/{id}`	Update username, enabled flag, roles, and optionally password. Requires `ADMIN`.
`DELETE`	`/api/v1/auth/users/{id}`	Delete a user. Requires `ADMIN`.
`POST`	`/api/v1/auth/users/delete-selected`	Delete the users listed in `{"ids":[...]}`. Requires `ADMIN`.
`GET`	`/api/v1/auth/roles`	List roles. Requires `ADMIN`.
`POST`	`/api/v1/auth/roles`	Create a role. Requires `ADMIN`.
`GET`	`/api/v1/auth/roles/{id}`	Read a role. Requires `ADMIN`.
`PUT`	`/api/v1/auth/roles/{id}`	Update a role label. Requires `ADMIN`.
`DELETE`	`/api/v1/auth/roles/{id}`	Delete a non-system role. Requires `ADMIN`.
`POST`	`/api/v1/auth/roles/delete-selected`	Delete the roles listed in `{"ids":[...]}`; system roles are skipped. Requires `ADMIN`.

The `ANONYMOUS`, `ADMIN`, and `USER` roles are seeded at first startup and cannot be deleted. Bulk delete endpoints respond `200 OK` with `{"deletedIds":[...],"skippedIds":[...]}`; unknown IDs and protected rows are reported as skipped. User responses never include passwords or password hashes.

3.2.2 User Settings

User settings are implemented in the `fr.tiogars.domaintemplate.settings.user` package. The first setting is `viewMode` (`light`, `dark`, or `system`; default `system`).

Method	Path	Description
`GET`	`/api/v1/settings/user/me`	Read the authenticated user's settings. Requires `USER` or `ADMIN`.
`PUT`	`/api/v1/settings/user/me`	Update the authenticated user's settings. Requires `USER` or `ADMIN`.
`GET`	`/api/v1/settings/user/users/{userId}`	Read the settings of a user with the `USER` role. Requires `ADMIN`; returns `404` for other users.
`PUT`	`/api/v1/settings/user/users/{userId}`	Update the settings of a user with the `USER` role. Requires `ADMIN`; returns `404` for other users.
`GET`	`/api/v1/settings/user/anonymous`	Read the `ANONYMOUS` role settings. Public.
`PUT`	`/api/v1/settings/user/anonymous`	Update the `ANONYMOUS` role settings. Requires `ADMIN`.

Settings that were never saved are returned with default values; they are stored on first update and deleted with their user.

3.2.3 Tasks

Method	Path	Description
`GET`	`/api/v1/task`	List all tasks, ordered by code.
`GET`	`/api/v1/task/{id}`	Read one task by server ID.
`POST`	`/api/v1/task`	Create a task.
`PUT`	`/api/v1/task/{id}`	Update its mutable fields.
`DELETE`	`/api/v1/task/{id}`	Archive one task without removing its database row.
`POST`	`/api/v1/task/delete-selected`	Archive the tasks listed in `{"ids":[...]}`. Requires `ADMIN`.
`POST`	`/api/v1/task/seed`	Upsert an array by code.
`POST`	`/api/v1/task/import.json`	Import and upsert a JSON array by code.
`POST`	`/api/v1/task/import.csv`	Import and upsert CSV rows by code.
`GET`	`/api/v1/task/export.json`	Export all tasks as JSON.
`GET`	`/api/v1/task/export.csv`	Export all tasks as CSV.
`GET`	`/api/v1/task/{id}/print.pdf`	Print one active task as PDF.
`GET`	`/api/v1/task/print.pdf`	Print all active tasks as PDF, ordered by code.
`POST`	`/api/v1/task/print-selected.pdf`	Print the active tasks listed in `{"ids":[...]}` as PDF, ordered by code.

3.2.4 Tags and Nomenclatures

Both taxonomy resources expose the following CRUD operations, where `{domain}` is `tag` or `nomenclature`:

Method	Path	Description
`GET`	`/api/v1/{domain}`	List entries.
`GET`	`/api/v1/{domain}/{id}`	Read an entry.
`POST`	`/api/v1/{domain}`	Create with `code`, `label`, optional `description`, and optional `parentId` for nomenclatures.
`PUT`	`/api/v1/{domain}/{id}`	Update `label`, `description`, and optional `parentId` for nomenclatures; exclude `code`.
`DELETE`	`/api/v1/{domain}/{id}`	Delete an unreferenced entry.

Create taxonomy entries before tasks that reference them. For example, create a tag with `{"code":"RELEASE","label":"Release","description":null}` and a root nomenclature with `{"code":"ENGINEERING","label":"Engineering","description":null,"parentId":null}`. Use the returned IDs, not codes, in task references and child `parentId` values.

3.3 PDF Printing

PDF printing uses Apache PDFBox 3.0.7 and three dedicated services for one, all, and selected tasks. All three endpoints require the same JWT authentication and `ADMIN` or `USER` role as reading tasks and return `application/pdf`, inline `Content-Disposition`, and `Cache-Control: no-store`.

Each record starts on a new A4 page and includes its ID, code, label, description, and audit fields. Long fields wrap and continue onto additional numbered pages. The embedded Liberation Sans font supplied by PDFBox supports accented Latin, Greek, and Cyrillic text; unsupported characters return `422 Unprocessable Content` instead of silently changing the content. Printing an empty collection returns a PDF stating that there are no tasks.

The selected request accepts 1 to 1000 positive IDs. Duplicate IDs print only once. Invalid selections return `400 Bad Request`; an unknown or archived ID returns `404 Not Found` for the entire request, so a PDF never silently omits selected records. Archived records are excluded from all PDF queries.

In the React backend console, **Print all** prints every active record, independently of grid pagination or filtering. **Print selected** prints only the checkbox-selected grid rows, including selections across pages, and is disabled when nothing is selected. **Print** on the detail page prints the current saved record. Each action opens the generated PDF in a new browser tab; use the PDF viewer's native Print control to choose a printer. Popups must be allowed for the site. Failed requests close the blank viewer and display an error in the console.

The React client's `print(id)`, `printAll()`, and `printSelected(ids)` methods return `Promise<Blob>`. Binary requests use the configured API URL, fetcher, and token provider directly, outside the JSON-oriented RTK Query cache, avoiding non-serializable PDF objects in Redux state.

3.4 Create

`POST /api/v1/task` accepts only business fields:

```
{
    "code": "RELEASE-001",
    "label": "Prepare release",
    "description": "Review the release checklist",
    "status": "TODO",
    "priority": "HIGH",
    "dueDate": "2026-11-01",
    "assignedTo": "alice",
    "tagIds": [],
    "nomenclatureId": null
}
```

The response is `201 Created`, includes the complete resource, and sets `Location` to `/api/v1/task/{id}`. A duplicate code returns `409 Conflict`. Blank required fields return `400 Bad Request`.

3.5 Read

`GET /api/v1/task` returns a JSON array. `GET /api/v1/task/{id}` returns one task or `404 Not Found`.

3.6 Update

`PUT /api/v1/task/{id}` accepts the mutable fields only; `code` is excluded:

```
{
    "label": "Prepare release",
    "description": null,
    "status": "IN_PROGRESS",
    "priority": "HIGH",
    "dueDate": "2026-11-01",
    "assignedTo": "alice",
    "tagIds": [],
    "nomenclatureId": null
}
```

The server preserves `id`, `code`, and `createdDate`, and replaces `lastModifiedDate`. The response is `200 OK`, or `404 Not Found` for an unknown ID.

3.7 Delete

`DELETE /api/v1/task/{id}` returns `204 No Content`, or `404 Not Found` for an unknown or already archived ID.

`POST /api/v1/task/delete-selected` archives several tasks in one transaction. The body is `{"ids":[1,2,3]}` and the response is `{"deletedIds":[1,2],"skippedIds":[3]}`, where skipped IDs were unknown or already archived. The admin console uses it for the grid's "Delete selected" action.

Deletion sets `deleted_date` and `deleted_by`; the original ID, code, business fields, and creation metadata remain in PostgreSQL. Archived rows are excluded from all normal JPA queries, including read, update, list, export, duplicate checks, and seed/import lookups. They cannot be updated or restored through the API.

After deletion, creating the same code produces a new row with a new ID. Seed/import also creates a new row rather than reviving the archived row. Multiple archived rows may share a code, but a PostgreSQL partial unique index permits at most one active row per code. Optimistic versioning prevents stale writers from overwriting a deletion or another update.

Concurrent writes that lose an optimistic-lock race, or violate active-code uniqueness, return `409 Conflict`; other database integrity errors are not treated as duplicate-code conflicts. Refresh the resource before retrying.

3.8 Persistence Auditing and Required Migration

Tasks, tags, and nomenclatures have server-owned auditing. Task deletion archives the task; taxonomy deletion is subject to reference checks. Users, roles, and user settings retain their existing behavior. Spring Data JPA supplies creation and modification timestamps, truncated to PostgreSQL microsecond precision, and records the authenticated username:

Database column	Meaning
`created_by`	Username at creation; never changed.
`last_modified_by`	Username at the last persisted change, including deletion.
`deleted_by`	Username that archived the row, or `NULL` while active.
`deleted_date`	Archival timestamp, or `NULL` while active.
`version`	Optimistic concurrency token managed by JPA.

Actors are username snapshots, not foreign keys: changing or deleting a user does not erase the recorded name. Writes without an authenticated security context fail; scheduled jobs must establish an explicit service identity. JSON responses (including JSON exports) expose `createdBy` and `lastModifiedBy` as read-only fields, displayed alongside their timestamps in the backend console detail view. Historical actors backfilled by the migration appear as `unknown`. Deletion metadata and the optimistic-lock version remain persistence-only; the CSV fields are described below. Spring Data auditing records the creator and most recent editor, not a full revision history.

Before deploying this API version, stop API writers, back up the database, and run the manual PostgreSQL task-domain migration. Run it for both existing and new databases, using the same schema/search path as the API. For example, with PostgreSQL connection variables configured:

```
psql -v ON_ERROR_STOP=1 -f java\api\src\main\resources\db\manual\task-domains.sql
```

The task-domain migration defines the task and taxonomy persistence schema. Hibernate `ddl-auto: update` is not a replacement for this migration and does not remove obsolete tables or migrate legacy business data automatically. Review the script against a database backup before upgrading.

This script replaces the former auditing migration for the legacy `domain_templates` schema. Archived task history is persistence-only and is not exposed by normal API endpoints.

The PostgreSQL lifecycle tests require Docker and run with the normal API test suite:

```
mvn -B -f java\pom.xml -pl api test
```

3.9 Seed and Import

Seed and both import formats use upsert semantics keyed by `code`: a missing code creates a task and an existing code updates all mutable task fields. Client-supplied `id`, `createdDate`, `lastModifiedDate`, `createdBy`, and `lastModifiedBy` values are ignored. The server keeps ownership of identifiers, timestamps, and audit actors.

`POST /api/v1/task/seed` and `POST /api/v1/task/import.json` accept an array. A minimal document is:

```
[
    {
        "code": "RELEASE-001",
        "label": "Prepare release",
        "description": null
    }
]
```

Both return `201 Created` with the stored resources.

Omitted workflow fields use the task defaults; optional date, assignee, and nomenclature values may be `null`. Imports may include all mutable fields, including `tagIds`; taxonomy references must already exist in the same database. Exported numeric taxonomy IDs are not portable to another database unless the same referenced entries exist there with matching IDs.

3.10 Import and Export Formats

JSON export returns the complete resource schema. The same exported JSON can be imported: server-owned fields are accepted only as transport data and ignored during the upsert.

```
[
    {
        "id": 42,
        "code": "RELEASE-001",
        "label": "Prepare release",
        "description": "Review the release checklist",
        "status": "TODO",
        "priority": "HIGH",
        "dueDate": "2026-11-01",
        "assignedTo": "alice",
        "tagIds": [],
        "nomenclatureId": null,
        "createdDate": "2026-08-30T17:00:00Z",
        "lastModifiedDate": "2026-08-30T17:00:00Z",
        "createdBy": "admin",
        "lastModifiedBy": "admin"
    }
]
```

CSV import and export use this header:

```
id,code,label,description,createdDate,lastModifiedDate,status,priority,dueDate,assignedTo,tagIds,nomenclatureId
```

The 12-column header and column order above round-trip all task business fields. `tagIds` is a semicolon-separated list of positive IDs (for example `3;7`); an empty cell means `[]`. Empty `description`, `dueDate`, `assignedTo`, and `nomenclatureId` cells become `null`. Blank status and priority use `TODO` and `NORMAL`. Date values use `YYYY-MM-DD`; referenced IDs must already exist. On import, `id`, `createdDate`, and `lastModifiedDate` are ignored. Audit actors are not included in CSV.

```
id,code,label,description,createdDate,lastModifiedDate,status,priority,dueDate,assignedTo,tagIds,nomenclatureId
42,RELEASE-001,Prepare release,Review the release checklist,2026-08-30T17:00:00Z,2026-08-30T17:00:00Z,TODO,HIGH,2026-11-01,alice,,
```

For backward-compatible import, the exact legacy header `id,code,label,description,createdDate,lastModifiedDate` is also accepted. Its rows use the new task-field defaults; export always writes the extended header.

3.11 Response

```
{
    "id": 42,
    "code": "RELEASE-001",
    "label": "Prepare release",
    "description": "Review the release checklist",
    "status": "TODO",
    "priority": "HIGH",
    "dueDate": "2026-11-01",
    "assignedTo": "alice",
    "tagIds": [],
    "nomenclatureId": null,
    "createdDate": "2026-08-30T17:00:00Z",
    "lastModifiedDate": "2026-08-30T17:05:00Z",
    "createdBy": "admin",
    "lastModifiedBy": "editor"
}
```

3.12 Clients

The `java-client`, `react-client`, and `dart-client` modules provide typed Java, TypeScript, and Dart access to CRUD, seed, import, and export operations. Construct each client with the service origin; the client appends `/api/v1/task`. Taxonomy clients use `/api/v1/tag` and `/api/v1/nomenclature`. Application package names and artifact coordinates remain unchanged.

The `react-client` also generates RTK Query endpoints and exports an `AuthClient` for login, current-user lookup, and user and role administration under `/api/v1/auth`, and a `UserSettingsClient` for user settings under `/api/v1/settings/user`.

4. Web Console

The `backend` webapp is a React, Vite, and MUI console for tasks, reusable tags, and hierarchical nomenclatures. It identifies resources by server-generated `id` and presents business fields directly instead of editing a generic JSON payload.

4.1 Run the Console

Start the API and the webapp from the repository root:

```
make java-run
make react-run
```

Open `http://localhost:5173`. The webapp reads the API base URL from `VITE_API_BASE_URL`; the Docker stack builds it with `http://localhost:8989`, while direct local development defaults to `http://localhost:8080`.

4.2 Routes

Route	Purpose
`/`	Dashboard with task metrics and recent tasks.
`/tasks`	List, import, export, print, and delete tasks.
`/tasks/new`	Create a task with a structured form.
`/tasks/{id}`	View and print one task by ID.
`/tasks/{id}/edit`	Update its mutable fields by ID.
`/tags`	Manage reusable task tags.
`/nomenclatures`	Manage reusable hierarchical classifications.
`/settings`	Manage your user settings; administrators also manage the `ANONYMOUS` role settings.
`/admin/users/{id}/edit`	Edit a user in the **Account** tab and its user settings in the **Settings** tab (administrators only).
`/about`	Show API, documentation, and Swagger UI links.

4.3 View Mode

Use the icons in the top bar or the `/settings` page to choose bright, dark, or system mode. The selected mode is saved in local storage and, when signed in, in the user settings through `/api/v1/settings/user/me`. Anonymous visitors start with the `ANONYMOUS` role settings.

The **Settings** tab of the user edit page is only available for users with the `USER` role.

4.4 List Tasks

The Tasks page presents task identity, workflow status, priority, and business fields. Desktop and tablet layouts use MUI DataGrid. Mobile layouts provide the same view, edit, and delete actions in a compact list.

4.5 Create or Update a Task

For creation, enter a required unique `code`, a required `label`, and an optional `description`. Choose status (`TODO`, `IN_PROGRESS`, `DONE`, or `CANCELLED`) and priority (`LOW`, `NORMAL`, `HIGH`, or `URGENT`); defaults are `TODO` and `NORMAL`. Due date is an optional `YYYY-MM-DD` date, and assignee is optional free text, not a user foreign key.

Select existing tags and an optional nomenclature. Tasks store their IDs as `tagIds` (default `[]`) and nullable `nomenclatureId`; create these reusable entries first. The server assigns ID, audit timestamps, and audit actors. Create calls `POST /api/v1/task`.

For update, the console loads `/tasks/{id}` and disables `code` because it is immutable. Only mutable fields are sent to `PUT /api/v1/task/{id}`; `code` is excluded.

4.6 Manage Tags and Nomenclatures

Tags are flat, reusable task labels. Nomenclatures are reusable classifications with an optional parent, for example **Engineering → Release**. Both have `id`, `code`, `label`, `description`, `createdDate`, `lastModifiedDate`, `createdBy`, and `lastModifiedBy`; nomenclatures additionally have nullable `parentId`. Creation accepts code, label, description, and optional parent; updates never send code.

The API requires references to exist and rejects self-parenting and hierarchy cycles. A tag or nomenclature referenced by a task cannot be deleted, nor can a nomenclature with children. Remove or change references first.

4.7 Delete a Task

The delete action asks for confirmation, then calls `DELETE /api/v1/task/{id}`. Task deletion archives the record; archived tasks disappear from normal views.

4.8 View and Print

The detail page shows task fields and audit metadata. Print actions request PDF output from the API and open it in a new browser tab. Use the PDF viewer's Print control to choose a printer. The list also supports printing all or selected tasks; see PDF printing.

4.9 Import and Export

The Tasks page can seed sample tasks and import or export JSON and CSV. Seed and import upsert by `code`: existing tasks are updated and missing ones are created. Imported IDs, timestamps, and audit actors never replace server-owned values.

JSON import accepts an array of records:

```
[
  {
    "id": 42,
    "code": "RELEASE-001",
    "label": "Prepare release",
    "description": "Review the release checklist",
    "status": "TODO",
    "priority": "HIGH",
    "dueDate": "2026-11-01",
    "assignedTo": "alice",
    "tagIds": [],
    "nomenclatureId": null,
    "createdDate": "2026-08-30T17:00:00Z",
    "lastModifiedDate": "2026-08-30T17:00:00Z"
  }
]
```

CSV import uses this header:

```
id,code,label,description,createdDate,lastModifiedDate,status,priority,dueDate,assignedTo,tagIds,nomenclatureId
42,RELEASE-001,Prepare release,Review the release checklist,2026-08-30T17:00:00Z,2026-08-30T17:00:00Z,TODO,HIGH,2026-11-01,alice,,
```

Tag IDs use semicolons within their CSV cell (for example `3;7`), not a JSON array. Empty tag IDs mean `[]`; empty due date, assignee, and nomenclature mean `null`. Blank status and priority use their defaults. The legacy six-column header is accepted for import; exports use the extended header.

Both formats round-trip exports. Import uses code, label, description, and task workflow/assignment/taxonomy fields; server-owned identity and audit values are ignored. Referenced tags and nomenclatures must already exist in the same database with matching IDs; taxonomy IDs are not automatically remapped between environments. See the API format reference for CSV encoding.

4.10 Coverage

Generate coverage reports for the monorepo:

```
make coverage
```

Reports are written to:

Stack	Report location
Java API	`template-java-api/target/site/jacoco/index.html`
Java client	`template-java-api-client/target/site/jacoco/index.html`
React backend	`template-react-backend/coverage/index.html`
React client	`template-react-api-client/coverage/index.html`
Dart client	`template-flutter-api-client/coverage/lcov.info`

5. Deployment

The application is split across several repositories. Each one publishes its own artifact; the `template-docker` repository assembles them in a single `docker-compose.yml` deployed by Dokploy.

Repository	Published artifact	Workflow
`template-java-api`	`ghcr.io/tiogars/template-java-api` image	`Docker Publish`
`template-react-backend`	`ghcr.io/tiogars/template-react-backend` image	`Build and publish Docker image`
`template-flutter`	GitHub Release with Android, Linux, Windows builds	`Flutter releases` (on `v*` tags)
`template-docs-mkdocs`	Documentation on GitHub Pages	`Build and deploy docs`
`template-presentation`	Slides on GitHub Pages	`Publish presentation`

Images are tagged `latest` (default branch), `sha-<commit>` and, for `v*` tags, with the version.

5.1 Services

`docker-compose.yml` declares three services and publishes no host port:

- `postgres` : PostgreSQL 17 with the persistent `postgres-data` volume. On the first start of an empty volume, it runs `postgres/initdb/01-task-domains.sql` , a copy of `template-java-api/src/main/resources/db/manual/task-domains.sql` .
- `api` : Spring Boot API on container port `8080` ; healthy once `/actuator/health` answers `UP` .
- `backend` : web console served by Nginx on container port `8080` . It starts after the API is healthy.

5.2 Deploy with Dokploy

1. Create a **Docker Compose** service with the GitHub provider, repository `tiogars/template-docker` , branch `main` and compose path `./docker-compose.yml` .
2. In **Environment**, set the variables listed below (see `.env.example`).
3. In **Domains**, assign one domain to `backend` (port `8080`) and one to `api` (port `8080`), with HTTPS.
4. Deploy. Dokploy pulls the images from GHCR; they must be public, otherwise register `ghcr.io` credentials (token with `read:packages`) in Dokploy.

To deploy a new image, rerun the deployment (or enable auto-deploy). Pin `API_IMAGE_TAG` and `BACKEND_IMAGE_TAG` to `sha-<commit>` or a version for reproducible deployments.

Variable	Required	Purpose
`POSTGRES_PASSWORD`	Yes	Database password.
`POSTGRES_DB`, `POSTGRES_USER`	No	Database name and user; default `template`.
`APP_ADMIN_USERNAME`	No	Forced administrator username; defaults to `admin`.
`APP_ADMIN_PASSWORD`	Yes	Forced administrator password, stored only as a BCrypt hash.
`APP_JWT_SECRET`	Yes	JWT HMAC signing secret, at least 32 characters.
`APP_JWT_EXPIRATION_SECONDS`	No	Access token lifetime; defaults to 3600 seconds.
`APP_CORS_ALLOWED_ORIGINS`	Yes	Comma-separated console origins, e.g. `https://console.example.com`.
`API_BASE_URL`	Yes	Public API URL used by the console, e.g. `https://api.example.com`.
`DOCS_BASE_URL`	No	Documentation URL shown in the console.
`GITHUB_BASE_URL`	No	Repository URL shown in the console.
`API_IMAGE_TAG`, `BACKEND_IMAGE_TAG`	No	Image tags; default `latest`.

Compose refuses to start when a required variable is missing.

5.2.1 Console runtime configuration

The console image serves `/config.js`, generated at container start from `API_BASE_URL`, `DOCS_BASE_URL` and `GITHUB_BASE_URL`. The same image can therefore target any environment without being rebuilt. An empty value falls back to the `VITE_*` value compiled into the bundle.

```
docker run --rm -p 5173:8080 -e API_BASE_URL=https://api.example.com \
  ghcr.io/tiogars/template-react-backend:latest
curl http://localhost:5173/config.js
# window.__APP_CONFIG__ = { apiBaseUrl: "https://api.example.com", docsBaseUrl: "", githubBaseUrl: "" };
```

5.3 Local Test of the Stack

```
cd template-docker
cp .env.example .env
docker compose -f docker-compose.yml -f compose.local.yml up -d --wait
```

`compose.local.yml` publishes the API on `http://localhost:8989` and the console on `http://localhost:5173`, and sets the matching URLs. Dokploy only uses `docker-compose.yml`.

5.4 Persistence Behavior

The API persists tasks, reusable tags, and hierarchical nomenclatures. The Compose volume is named from the project and the declared `postgres-data` volume; Docker Compose resolves the final volume name automatically.

The API also persists local authentication data in `auth_roles`, `auth_users`, and `auth_user_roles`. At startup it seeds the `ANONYMOUS`, `ADMIN`, and `USER` roles, then forces the administrator account from `APP_ADMIN_USERNAME` and `APP_ADMIN_PASSWORD`. If that username already exists, the account is enabled, its password hash is replaced from the environment value, and the `ADMIN` role is guaranteed.

Hibernate uses `ddl-auto: update`, but automatic schema updates do not replace the task-domain migration, migrate legacy business data, or remove obsolete schema objects. The init script only runs on an empty volume: apply `task-domains.sql` manually on an existing database before upgrading. Follow the migration instructions.

5.5 Reset the Demo Database

Reset the volume when you want to remove all demo data and obsolete tables. This operation is destructive: it removes all task, taxonomy, authentication, and settings data; it is not an upgrade procedure.

```
docker compose down --volumes
docker compose up -d --wait
```

The init script recreates the task-domain schema on the new volume. In Dokploy, stop the service and delete the `postgres-data` volume before redeploying.

5.6 Publish the Flutter Applications

The console download page (`/downloads`) lists the files served from `/releases/`, described by `/releases/manifest.json` (version, size, SHA-256, default API URL). Missing platforms are shown as unavailable.

1. In `template-flutter`, set the `API_BASE_URL` Actions variable to the public HTTPS API URL compiled into the applications (users can still change it).
2. Push a `v*` tag. The `Flutter releases` workflow builds Android and Linux on Ubuntu and Windows on a Windows runner, then publishes a GitHub Release with the packages and `build-info-<platform>.json` files.
3. Rerun the `template-react-backend` image workflow. It downloads the latest `template-flutter` release, generates the manifest, and embeds the files in the image. Without any release, the image is published without downloads.
4. Redeploy in Dokploy.

Android release builds block clear-text HTTP traffic: use an HTTPS API URL for the APK. The APK is signed with the debug key until a release signing configuration is added to `android/app/build.gradle.kts`.

5.7 Build Documentation and Slides

The documentation (`template-docs` sources, `template-docs-mkdocs` configuration) and the presentation (`template-presentation`) are published on GitHub Pages by their own workflows; they are not part of the Compose stack. Set `DOCS_BASE_URL` to the published documentation URL.

5.8 Production Considerations

Keep the API, database, console, and documentation as independently deployable units. Replace `ddl-auto: update` with versioned migrations, provide credentials through the Dokploy environment, terminate TLS at the platform edge, and back up the `postgres-data` volume.

6. Recommended Copilot Skills

Use these Copilot skills when evolving this monorepo. They help keep work focused, validated, and aligned with the technology in each module.

6.1 Core Project Work

Skill	Use When
`project-setup-info-local`	Creating or reshaping the monorepo structure, adding a new top-level module, or bootstrapping a new framework area.
`documentation-writer`	Writing or reorganizing MkDocs content, README sections, tutorials, how-to guides, reference pages, or explanations.
`refactor`	Improving maintainability without changing behavior, such as extracting methods, simplifying clients, or reducing duplication.

6.2 Java and Spring Boot

Skill	Use When
`java-springboot`	Adding API endpoints, configuration, validation, controllers, services, or Spring Boot production conventions in `api`.
`java-junit`	Adding or improving unit and integration-style tests for `java/api` and `java/java-client`.
`java-lsp-tools`	Navigating Java symbols precisely before changing Java classes, methods, or package structure.

6.3 Frontend and TypeScript

Skill	Use When
`vite`	Changing Vite configuration, build behavior, dev server settings, or frontend packaging in `backend`.
`typescript-dependencies-upgrade`	Updating npm dependencies for `react/backend`, `react/react-client`, or `docs/presentation`, except the TypeScript compiler itself.
`typescript-compiler-upgrade`	Upgrading the `typescript` package or testing a major compiler migration.
`web-design-guidelines`	Reviewing the management console UI for usability, accessibility, responsive behavior, and visual quality.

6.4 Dart Client

Skill	Use When
`flutter-use-http-package`	Extending the Dart client API methods or HTTP behavior.
`flutter-implement-json-serialization`	Adding typed Dart models or changing JSON mapping rules in `flutter/dart-client`.

6.5 Documentation, Diagrams, and Presentations

Skill	Use When
`archify`	Creating architecture, sequence, workflow, data-flow, or lifecycle diagrams from the implementation.
`md-to-docx`	Exporting Markdown documentation into a polished Word document.

6.6 Delivery and Operations

Skill	Use When
`cve-remediation`	Scanning and remediating dependency or container vulnerabilities.
`runtime-validation`	Designing or executing smoke, integration, startup, or end-to-end validation for the deployable stack.
`github-actions-hardening`	Creating or reviewing GitHub Actions workflows for secure CI/CD.
`github-actions-efficiency`	Optimizing GitHub Actions runtime, caching, and cost.

6.7 Suggested Defaults

- For API behavior changes, start with `java-springboot`, then use `java-junit` for tests.
- For frontend build or UI work, start with `vite`; add `web-design-guidelines` for user-facing screens.
- For client model changes, update Java, TypeScript, and Dart clients together, then run all module tests.
- For deployment changes, combine `runtime-validation` with `cve-remediation` before publishing images.



https://docs.template.tiogars.fr/

Copyright © Tiogars 2026