



# DomainTemplate CRUD

## Documentation

Auteur : Tiogars

Copyright © Tiogars 2026

# Table of Contents

---

|                                                 |    |
|-------------------------------------------------|----|
| 1. DomainTemplate CRUD                          | 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 DomainTemplate                | 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                                 | 6  |
| 2.1 Public Object and Persistence Entity        | 6  |
| 2.2 Invariants and Ownership                    | 6  |
| 2.3 Schema Evolution                            | 6  |
| 3. DomainTemplate API Reference                 | 7  |
| 3.1 Resource Schema                             | 7  |
| 3.2 Endpoints                                   | 8  |
| 3.2.1 Authentication and Administration         | 8  |
| 3.2.2 User Settings                             | 8  |
| 3.2.3 DomainTemplate                            | 9  |
| 3.3 PDF Printing                                | 9  |
| 3.4 Create                                      | 10 |
| 3.5 Read                                        | 10 |
| 3.6 Update                                      | 10 |
| 3.7 Delete                                      | 10 |
| 3.8 Persistence Auditing and Required Migration | 11 |
| 3.9 Seed and Import                             | 11 |
| 3.10 Import and Export Formats                  | 12 |
| 3.11 Response                                   | 12 |
| 3.12 Clients                                    | 12 |
| 4. Web Console                                  | 13 |
| 4.1 Run the Console                             | 13 |
| 4.2 Routes                                      | 13 |
| 4.3 View Mode                                   | 13 |
| 4.4 List DomainTemplates                        | 13 |
| 4.5 Create or Update a DomainTemplate           | 13 |
| 4.6 Delete a DomainTemplate                     | 14 |

|                                                |    |
|------------------------------------------------|----|
| 4.7 View and Print                             | 14 |
| 4.8 Import and Export                          | 14 |
| 4.9 Coverage                                   | 14 |
| 5. Deployment                                  | 15 |
| 5.1 Start the Stack                            | 15 |
| 5.2 Persistence Behavior                       | 15 |
| 5.3 Reset the Demo Database                    | 16 |
| 5.4 Build Documentation and Slides             | 16 |
| 5.5 Production Considerations                  | 16 |
| 6. Recommended Copilot Skills                  | 17 |
| 6.1 Core Project Work                          | 17 |
| 6.2 Java and Spring Boot                       | 17 |
| 6.3 Frontend and TypeScript                    | 17 |
| 6.4 Dart Client                                | 17 |
| 6.5 Documentation, Diagrams, and Presentations | 18 |
| 6.6 Delivery and Operations                    | 18 |
| 6.7 Suggested Defaults                         | 18 |

# 1. DomainTemplate CRUD

This monorepo demonstrates one business object from persistence to user interface. A Spring Boot API stores `DomainTemplate` objects in PostgreSQL, three reusable clients expose the same contract, and a React console provides day-to-day CRUD and bulk data operations.

## 1.1 What You Will Run

The tutorial starts a complete local stack and creates one `DomainTemplate`. The resulting object has a server-generated `Long` identifier, a unique immutable `code`, a required `label`, an optional `description`, and server-owned `Instant` timestamps.

## 1.2 Modules

| Module                    | Purpose                                                   |
|---------------------------|-----------------------------------------------------------|
| <code>api</code>          | Spring Boot CRUD API backed by JPA and PostgreSQL.        |
| <code>java-client</code>  | Maven client library using Java <code>HttpClient</code> . |
| <code>react-client</code> | TypeScript client library for React or browser apps.      |
| <code>backend</code>      | React + Vite management console.                          |
| <code>dart-client</code>  | Dart package for Flutter or Dart services.                |
| <code>docs</code>         | MkDocs project documentation.                             |
| <code>presentation</code> | 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 repository root:

```
make docker-up
```

Compose publishes the documentation on `8080`, the API on `8989`, and the console on `5173`.

### 1.4.2 2. Create a DomainTemplate

```
curl -X POST http://localhost:8989/api/v1/domaintemplate \
-H "Content-Type: application/json" \
-d '{"code":"CUSTOMER","label":"Customer","description":"Customer master data"}
```

The response resembles:

```
{
  "id": 1,
  "code": "CUSTOMER",
  "label": "Customer",
  "description": "Customer master data",
  "createdAt": "2026-08-30T17:00:00Z",
  "lastModifiedDate": "2026-08-30T17:00:00Z"
}
```

The exact identifier and timestamps are assigned by the server.

### 1.4.3 3. Use the console

Open `http://localhost:5173`, select **DomainTemplates**, and open the new object by its ID. From this page you can create, inspect, update, delete, seed, import, and export DomainTemplates.

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

### 1.4.4 4. Stop the stack

```
make docker-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 /api/v1/domaintemplate]
    JavaClient[Java client library] --> Api
    DartClient[Dart client library] --> Api
    Api --> Service[DomainTemplateService]
    Service --> Repository[DomainTemplateRepository]
    Repository --> Database[(PostgreSQL domain_templates)]
    Docs[MkDocs] -- documents --> Api
    Slides[Marp presentation] -- explains --> Docs
```

The repository demonstrates one vertical CRUD feature. HTTP controllers validate transport requests and delegate business operations to `DomainTemplateService`. The service owns timestamps and upsert behavior, while `DomainTemplateRepository` isolates Spring Data JPA access. The Java, TypeScript, and Dart clients depend on the public HTTP contract, not on persistence classes.

### 2.1 Public Object and Persistence Entity

`DomainTemplate` is the public business object returned by controllers and represented by each client. It is an immutable Java record with `id`, `code`, `label`, `description`, `createdAt`, and `lastModifiedDate`. This is the model API consumers should understand.

`DomainTemplateEntity` is the internal mutable JPA representation. It maps to the PostgreSQL table `domain_templates` and exists to satisfy persistence concerns such as a protected no-argument constructor, generated identity, column constraints, and dirty tracking. Controllers never expose the JPA entity.

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  | <code>id</code> is a generated <code>Long</code> .                                  |
| Business identity     | Service and database | <code>code</code> is required, unique, and immutable.                               |
| Mutable business data | Client request       | <code>label</code> is required; <code>description</code> may be <code>null</code> . |
| Audit timestamps      | Service              | Both dates are <code>Instant</code> values assigned by the server.                  |
| Bulk reconciliation   | Service              | Seed and import upsert by <code>code</code> and ignore imported IDs and dates.      |

### 2.3 Schema Evolution

Hibernate is configured with `spring.jpa.hibernate.ddl-auto: update` for this runnable application. It can add or adjust the `domain_templates` schema, but it does not remove tables that are no longer mapped. In particular, an existing demo volume may retain the obsolete `domain_records` table. The application does not use that table.

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. DomainTemplate API Reference

---

The API manages DomainTemplate business object. Its singular base path is `/api/v1/domaintemplate`. 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 Schema

---

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

## 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 <code>ADMIN</code> .                                                                          |
| POST   | /api/v1/auth/users                 | Create a user with a BCrypt-stored password. Requires <code>ADMIN</code> .                                         |
| GET    | /api/v1/auth/users/{id}            | Read a user. Requires <code>ADMIN</code> .                                                                         |
| PUT    | /api/v1/auth/users/{id}            | Update username, enabled flag, roles, and optionally password. Requires <code>ADMIN</code> .                       |
| DELETE | /api/v1/auth/users/{id}            | Delete a user. Requires <code>ADMIN</code> .                                                                       |
| POST   | /api/v1/auth/users/delete-selected | Delete the users listed in <code>{ "ids": [...] }</code> . Requires <code>ADMIN</code> .                           |
| GET    | /api/v1/auth/roles                 | List roles. Requires <code>ADMIN</code> .                                                                          |
| POST   | /api/v1/auth/roles                 | Create a role. Requires <code>ADMIN</code> .                                                                       |
| GET    | /api/v1/auth/roles/{id}            | Read a role. Requires <code>ADMIN</code> .                                                                         |
| PUT    | /api/v1/auth/roles/{id}            | Update a role label. Requires <code>ADMIN</code> .                                                                 |
| DELETE | /api/v1/auth/roles/{id}            | Delete a non-system role. Requires <code>ADMIN</code> .                                                            |
| POST   | /api/v1/auth/roles/delete-selected | Delete the roles listed in <code>{ "ids": [...] }</code> ; system roles are skipped. Requires <code>ADMIN</code> . |

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

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

### 3.2.3 DomainTemplate

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

## 3.3 PDF Printing

PDF printing uses Apache PDFBox 3.0.7 and three dedicated services for one, all, and selected DomainTemplates. All three endpoints require the same JWT authentication and `ADMIN` or `USER` role as reading DomainTemplates 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 DomainTemplates.

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/domaintemplate` accepts only business fields:

```
{
  "code": "CUSTOMER",
  "label": "Customer",
  "description": "Customer master data"
}
```

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

## 3.5 Read

---

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

## 3.6 Update

---

PUT `/api/v1/domaintemplate/{id}` accepts the mutable fields only:

```
{
  "label": "Customer record",
  "description": 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/domaintemplate/{id}` returns `204 No Content`, or `404 Not Found` for an unknown or already archived ID.

POST `/api/v1/domaintemplate/delete-selected` archives several objects 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

Auditing and soft deletion apply to `DomainTemplateEntity` only; 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                                                            |
|-------------------------------|--------------------------------------------------------------------|
| <code>created_by</code>       | Username at creation; never changed.                               |
| <code>last_modified_by</code> | Username at the last persisted change, including deletion.         |
| <code>deleted_by</code>       | Username that archived the row, or <code>NULL</code> while active. |
| <code>deleted_date</code>     | Archival timestamp, or <code>NULL</code> while active.             |
| <code>version</code>          | 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 contract remains unchanged. 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 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\domain-template-auditing-soft-delete.sql
```

The migration is transactional and rerunnable. It retains all rows and existing timestamps, backfills unavailable historical actors as `unknown`, adds the audit/deletion/version columns, removes the old Hibernate-generated unique constraint on `code`, and creates the active-only unique index. It can also create the `DomainTemplate` table in an empty database. `ddl-auto: update` alone does **not** remove the old constraint or create the partial index; the script is deliberately not executed automatically.

Inspect retained history with a privileged SQL query (it is not exposed by normal API endpoints):

```
SELECT id, code, created_by, created_date, last_modified_by,
       last_modified_date, deleted_by, deleted_date
FROM domain_templates
WHERE code = 'CUSTOMER'
ORDER BY id;
```

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 an object and an existing code updates `label` and `description`. Client-supplied `id`, `createdDate`, `lastModifiedDate`, `createdBy`, and `lastModifiedBy` values are ignored. The server keeps ownership of identifiers, timestamps, and audit actors.

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

```
[
  {
    "code": "CUSTOMER",
    "label": "Customer",
    "description": null
  }
]
```

Both return `201 Created` with the stored resources.

## 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": "CUSTOMER",
    "label": "Customer",
    "description": "Customer master data",
    "createdAt": "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,createdAt,lastModifiedDate
```

All six columns are required in the CSV header so an export can round-trip. On import, `id`, `createdAt`, and `lastModifiedDate` are ignored; empty `description` becomes `null`.

```
id,code,label,description,createdAt,lastModifiedDate
42,CUSTOMER,Customer,Customer master data,2026-08-30T17:00:00Z,2026-08-30T17:00:00Z
```

## 3.11 Response

```
{
  "id": 42,
  "code": "CUSTOMER",
  "label": "Customer",
  "description": "Customer master data",
  "createdAt": "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/domaintemplate`.

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 the `DomainTemplate` CRUD API. It identifies resources by server-generated `id` and presents the 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                                                                                                       |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>/</code>                          | Dashboard with metrics and the last 10 stored <code>DomainTemplates</code> .                                  |
| <code>/domainTemplates</code>           | List, import, export, and delete <code>DomainTemplates</code> .                                               |
| <code>/domainTemplates/new</code>       | Create a <code>DomainTemplate</code> with a JSON payload form.                                                |
| <code>/domainTemplates/{id}</code>      | View and print one <code>DomainTemplate</code> by ID.                                                         |
| <code>/domainTemplates/{id}/edit</code> | Update its label and description by ID.                                                                       |
| <code>/settings</code>                  | Manage your user settings; administrators also manage the <code>ANONYMOUS</code> role settings.               |
| <code>/admin/users/{id}/edit</code>     | Edit a user in the <b>Account</b> tab and its user settings in the <b>Settings</b> tab (administrators only). |
| <code>/about</code>                     | 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 DomainTemplates

The `DomainTemplates` page lists `id`, `code`, `label`, `description`, `createdDate`, and `lastModifiedDate`. 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 DomainTemplate

For creation, enter a required unique `code`, a required `label`, and an optional `description`. The server assigns the ID and dates. Create calls `POST /api/v1/domaintemplate`.

For update, the console loads `/domainTemplates/{id}` and disables `code` because it is immutable. Only `label` and `description` are sent to `PUT /api/v1/domaintemplate/{id}`.

## 4.6 Delete a DomainTemplate

The delete action asks for confirmation, then calls `DELETE /api/v1/domaintemplate/{id}`.

## 4.7 View and Print

The detail page shows all six fields. Use the print action to print the detail view; navigation and action controls are hidden in print output.

## 4.8 Import and Export

The DomainTemplates page can seed the standard DomainTemplate records and import or export JSON and CSV. Seed and import upsert by `code`: existing business objects are updated and missing ones are created. Imported IDs and dates never replace server-owned values.

JSON import accepts an array of records:

```
[
  {
    "id": 42,
    "code": "CUSTOMER",
    "label": "Customer",
    "description": "Customer master data",
    "createdDate": "2026-08-30T17:00:00Z",
    "lastModifiedDate": "2026-08-30T17:00:00Z"
  }
]
```

CSV import uses this header:

```
id,code,label,description,createdDate,lastModifiedDate
42,CUSTOMER,Customer,Customer master data,2026-08-30T17:00:00Z,2026-08-30T17:00:00Z
```

Both formats round-trip exports. During import, `code`, `label`, and `description` are the effective business input; `id`, `createdDate`, and `lastModifiedDate` are ignored.

## 4.9 Coverage

Generate coverage reports for the monorepo:

```
make coverage
```

Reports are written to:

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

## 5. Deployment

---

The local deployment uses six services:

- `postgres` : PostgreSQL 17 with a persistent named volume;
- `api` : Spring Boot application available on port `8989` locally. The base Compose file does not publish the API port; `make docker-up` adds the local override that publishes it;
- `backend` : static Vite build served by Nginx on port `5173` .
- `docs` : MkDocs site published on port `8000` .
- `java-site` : Maven Site with aggregated Javadocs published on port `8083` .
- `presentation` : Marp presentation published on port `8082` .

### 5.1 Start the Stack

---

```
make docker-up
```

After startup:

- Documentation: `http://localhost:8000`
- Java Maven Site: `http://localhost:8083`
- Java API documentation: `http://localhost:8083/apidocs/`
- API health: `http://localhost:8989/actuator/health`
- Swagger UI: `http://localhost:8989/swagger-ui.html`
- Management console: `http://localhost:5173`

The local Compose file supplies a development administrator account and JWT secret. The default console login is `admin / admin` .

Deployments on platforms such as Dokploy should use `docker/compose.yml` alone. The API remains available to other Compose services on port `8080` , with no host port published. The local development override is `docker/compose.local.yml` .

Inspect service logs with:

```
make docker-logs
```

Stop containers while retaining PostgreSQL data:

```
make docker-down
```

### 5.2 Persistence Behavior

---

The API persists `DomainTemplateEntity` rows in `domain_templates` . 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` . This is convenient for this solution because it initializes `domain_templates` , but it does not remove obsolete schema objects. A volume created by an earlier version may therefore still contain `domain_records` . That table is unused and its presence does not indicate that the current application writes to it.

## 5.3 Reset the Demo Database

Reset the volume when you want to remove all demo data and obsolete tables. This operation is destructive.

```
docker compose -f docker/compose.yml -f docker/compose.local.yml down --volumes
make docker-up
```

The second command creates a fresh PostgreSQL volume and Hibernate recreates `domain_templates` on API startup.

## 5.4 Build Documentation and Slides

```
make docs-build
make presentation-build
```

The host commands `make docs-build` and `make docs-serve` use the lightweight `docs/mkdocs.yml` configuration with MkDocs Material. Both read `docs/src` and write generated output to `site_output`.

The Docker documentation service uses `ghcr.io/tiogars/mkdocs-docker-image:latest` with the richer `docs/mkdocs/mkdocs.yml` configuration, including PDF generation. The full `docs` directory is mounted read-only at `/server/docs` so the configuration, hooks, assets, overrides, templates, and Markdown sources are all available. `site_output` is mounted separately at `/server/site_output` for generated files. The server watches both the sources and the configuration directory.

Restart only the documentation service after changing its Compose configuration:

```
make docker-up-docs
```

To build HTML and PDF output with the image's bundled plugins:

```
docker compose -f docker/compose.yml run --rm --no-deps docs build --config-file docs/mkdocs/mkdocs.yml --strict
```

`MKDOCS_CONFIG` overrides the server configuration path inside the container. Use a path under the mounted tree, such as `docs/mkdocs/mkdocs.yml`, rather than a host path or `../docs/...`.

## 5.5 Production Considerations

Keep the API, database, console, and documentation as independently deployable units. Replace `ddl-auto: update` with versioned migrations, provide managed PostgreSQL credentials through secrets, terminate TLS at the platform edge, and set the console API base URL to the public API origin.

Provide these security variables through the deployment secret store:

| Variable                                | Required | Purpose                                                      |
|-----------------------------------------|----------|--------------------------------------------------------------|
| <code>APP_ADMIN_USERNAME</code>         | Yes      | Forced administrator username.                               |
| <code>APP_ADMIN_PASSWORD</code>         | Yes      | Forced administrator password, stored only as a BCrypt hash. |
| <code>APP_JWT_SECRET</code>             | Yes      | JWT HMAC signing secret, at least 32 characters.             |
| <code>APP_JWT_EXPIRATION_SECONDS</code> | No       | Access token lifetime; defaults to 3600 seconds.             |

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

### 6.2 Java and Spring Boot

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

### 6.3 Frontend and TypeScript

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

### 6.4 Dart Client

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

## 6.5 Documentation, Diagrams, and Presentations

---

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

## 6.6 Delivery and Operations

---

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



<http://0.0.0.0:8000/>