

# Configuration and data models

JasperWho? is built around a small set of interconnected models. Understanding them is the key to understanding everything else — from how reports are set up, to how renderings are stored, to how print jobs are dispatched.

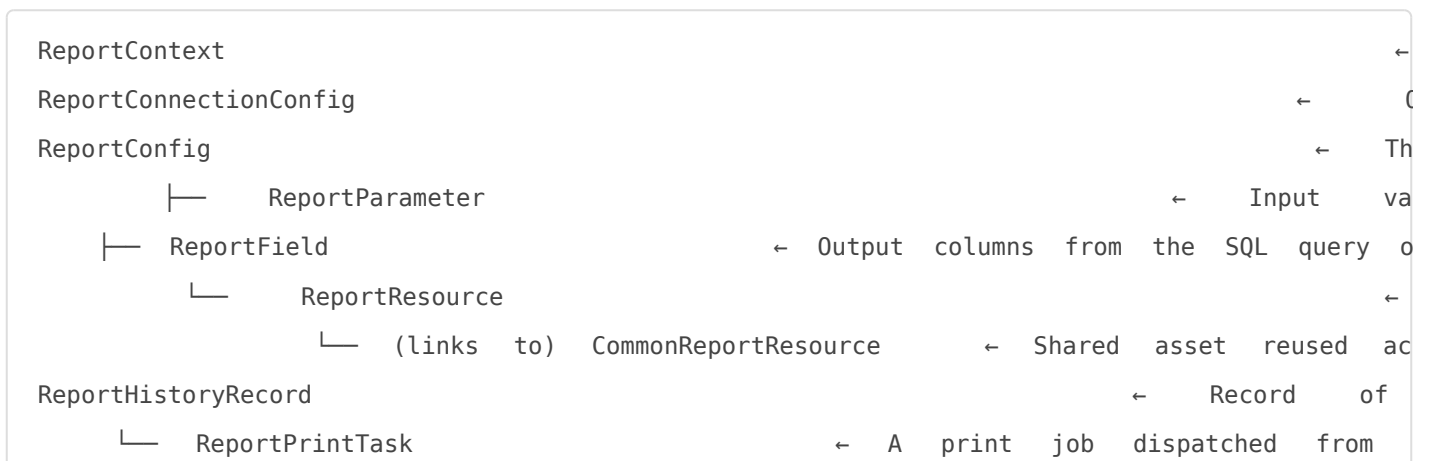
## Field Naming Conventions

JasperWho? uses two naming styles consistently throughout the system. Database columns and Laravel model attributes are always `snake_case` — for example `broadcast_id`, `report_config_id`, `created_by_token_id`. The API and frontend use `camelCase` for all request and response fields — the same fields become `broadcastId`, `reportConfigId`, `createdByTokenId`.

This split is consistent without exception: whenever you are working with the API or the frontend, use `camelCase`. Whenever you are looking at raw database records, migration files, or server-side model attributes, expect `snake_case`. Throughout this documentation, all field names and JSON examples follow the API convention — `camelCase`.

## The Model Hierarchy

Every piece of data in JasperWho? fits into a clear hierarchy. At the top sits the **ReportConfig** — the central entity. Everything else either belongs to it, describes it, or records what happened when it was used.



---

# ReportContext

A context is a **visual label** you assign to report configurations to group and identify them at a glance. It carries no functional logic — it is purely organisational.

Field	Description
<code>context_name</code>	Display name of the context
<code>context_description</code>	Short description
<code>context_text_color</code>	Hex color for the label text
<code>context_badge_color</code>	Hex color for the badge background
<code>context_border_color</code>	Hex color for the badge border

Every `ReportConfig` requires a context. A single context can be shared across any number of report configurations.

---

# ReportConnectionConfig

A connection config represents a **live database connection** that JasperWho? can use as a data source when rendering a report. When assigned to a `ReportConfig`, JasperWho? executes the report's SQL query against this connection at render time and feeds the result rows into the report as field data.

Field	Description
<code>connection_name</code>	Friendly name for this connection
<code>connection_driver</code>	Database driver (see table below)
<code>connection_host</code>	IP address of the database server
<code>connection_port</code>	Port — required
<code>connection_database</code>	Database / schema name
<code>connection_username</code>	Username (stored encrypted)
<code>connection_password</code>	Password (stored encrypted)
<code>connection_test_query</code>	SQL query used to verify the connection
<code>connection_tested</code>	Whether the connection has been successfully tested

## Supported drivers:

Driver	Database
mysql	MySQL
mariadb	MariaDB
pgsql	PostgreSQL
sqlsrv	Microsoft SQL Server

**Connection status** is derived from `connection_tested`:

Status	Meaning
approved	Connection has been tested successfully
unapproved	Never tested or last test failed

⚠ **A report can only be rendered with a live connection if its status is *approved*.** JasperWho? will reject render requests for reports whose connection has not been successfully tested.

## Network Requirements

JasperWho? establishes the database connection directly from the server it runs on. The target database must therefore be **reachable from that host** — ideally within the same network or at minimum via a secured private channel.

⚠ **Do not expose your database to the public internet.** Configuring a publicly accessible database — or opening firewall ports to make one reachable — is a significant security risk and is strongly discouraged. If JasperWho? and your database run in separate networks, use an encrypted VPN tunnel instead: **WireGuard** or **OpenVPN** are both well-suited for this purpose.

## When is a ReportConnectionConfig needed?

A `ReportConnectionConfig` is **optional** per `ReportConfig`. Whether you need one depends on how your report gets its data:

Scenario	Connection needed?
Report has no detail band — purely static layout	No
Report has a detail band, data delivered via API at render time	No

Scenario	Connection needed?
Report has a detail band and fetches data via SQL	<b>Yes</b>

# ReportConfig

The `ReportConfig` is the **core entity** of JasperWho?. It represents a single JasperReports template — the `.jrxml` file — together with all the metadata JasperWho? maintains about it.

Field	Description
<code>report_name</code>	Display name of the report
<code>report_description</code>	Optional description
<code>report_file_name</code>	Internal filename of the stored <code>.jrxml</code>
<code>report_width</code>	Page width in mm (extracted from the <code>.jrxml</code> on upload)
<code>report_height</code>	Page height in mm (extracted from the <code>.jrxml</code> on upload)
<code>report_query</code>	SQL query — defined in JasperWho? and stored in the database
<code>report_has_detail_band</code>	Whether the template contains a detail band (extracted on upload)
<code>report_context_id</code>	FK → <code>ReportContext</code>
<code>report_connection_config_id</code>	FK → <code>ReportConnectionConfig</code> (nullable)
<code>report_preview_base64</code>	Base64-encoded preview image
<code>report_thumbnail_base64</code>	Base64-encoded thumbnail image

**i The SQL query is not defined in the `.jrxml` file.** It is written and managed directly in JasperWho? and stored in the database as part of the `ReportConfig`. The `.jrxml` only defines which fields the query result maps to.

When a `.jrxml` file is uploaded, JasperWho? **automatically analyses it** and creates the associated `ReportParameter`, `ReportField`, and `ReportResource` records. You then review and complete the auto-generated data — for example setting example values or uploading resource files.

## ReportParameter

Parameters are the **inputs** passed into a report at render time — dates, IDs, filter values, flags, and so on.

Field	Description
<code>parameter_name</code>	Parameter name as defined in the <code>.jrxml</code>
<code>parameter_data_type</code>	Java class name (e.g. <code>java.lang.String</code> , <code>java.lang.Integer</code> )
<code>parameter_required</code>	Read from the <code>required</code> custom property in the <code>.jrxml</code>
<code>parameter_evaluation</code>	Evaluation time, extracted from the <code>.jrxml</code>
<code>parameter_example_value</code>	Read from the <code>exampleValue</code> custom property in the <code>.jrxml</code>

Both `parameter_required` and `parameter_example_value` are sourced from **custom properties** embedded in the `.jrxml` parameter definition. They can also be set manually in JasperWho? after upload.

## ReportField

Fields represent the **data columns** that populate the report's detail band — either from an SQL query result or from a data array delivered at render time.

Field	Description
<code>field_name</code>	Field name as defined in the <code>.jrxml</code>
<code>field_data_type</code>	Java class name (e.g. <code>java.lang.String</code> , <code>java.math.BigDecimal</code> )
<code>field_example_value</code>	Read from the <code>exampleValue</code> custom property in the <code>.jrxml</code>

`field_example_value` is used when rendering a preview without a live database connection.

## ReportResource

Resources are **graphic file assets** — images and logos — embedded in the report template. They are referenced in the `.jrxml` via parameters following the `P_RESOURCE_` naming convention.

Field	Description
<code>parameter_name</code>	The <code>P_RESOURCE_</code> parameter name as referenced in the <code>.jrxml</code>
<code>resource_file_name</code>	Filename of the directly uploaded file (nullable)
<code>common_report_resource_id</code>	FK → <code>CommonReportResource</code> (nullable)

A `ReportResource` either holds its own uploaded file **or** it is linked to a `CommonReportResource` — never both at the same time. When linking to a common resource, the resource's own file is deleted and the common file is used in its place.

## CommonReportResource

A `CommonReportResource` is a **shared graphic asset** — a company logo, a standard header image — that multiple report configurations can reference. Instead of uploading the same file to each report individually, you upload it once and link individual `ReportResource` records to it.

Field	Description
<code>resource_name</code>	Display name
<code>resource_description</code>	Optional description
<code>resource_file_name</code>	Internal filename of the stored file

**i Linking is a one-way action.** When a `ReportResource` is linked to a `CommonReportResource`, its own file is permanently deleted. Unlinking removes the reference but does not restore the file — you will need to re-upload it.

## ReportHistoryRecord

A `ReportHistoryRecord` captures the **full context of a rendering** — what was requested, what was returned, and whether it succeeded. Creating a history record is **optional** and controlled by the `createHistoryRecord` flag in the render request.

Field	Description
<code>report_config_id</code>	FK → <code>ReportConfig</code>
<code>trace_id</code>	Unique identifier for this rendering run
<code>output_type</code>	How the PDF was returned (see below)
<code>report_api_payload</code>	The exact request payload sent to the render call
<code>report_api_response</code>	The full API response, stored for traceability
<code>report_pdf_base64</code>	Base64-encoded PDF content

Field	Description
<code>report_pdf_file_name</code>	Filename on disk
<code>report_thumbnail_base64</code>	Base64-encoded thumbnail of the first page (generated asynchronously)
<code>status</code>	Outcome of the rendering (see below)

**Output types** (`output_type`):

Value	Description
<code>base64</code>	PDF returned inline as a Base64 string
<code>url</code>	PDF stored as a file, a URL is returned
<code>preview</code>	Rendered for preview; file is not persisted
<code>none</code>	No PDF output — used for print-only flows

**Status values** (`status`):

Value	Description
<b>ok</b>	Rendering succeeded, PDF received
<code>render_fail</code>	No errors reported, but no PDF received
<b>error</b>	JasperReports returned one or more errors
<code>unknown</code>	Status cannot be determined

History records are retained for a configurable number of days — see [Environment Configuration](#) below. Thumbnails are generated asynchronously after rendering completes.

## ReportPrintTask

A `ReportPrintTask` represents a **print job** dispatched to a physical printer. It is always linked to a `ReportHistoryRecord` — you always print a specific past rendering, not a report config directly.

Field	Description
<code>report_config_id</code>	FK → <code>ReportConfig</code>
<code>report_history_record_id</code>	FK → <code>ReportHistoryRecord</code>
<code>trace_id</code>	Unique identifier for this print run

Field	Description
<code>broadcast_id</code>	WebSocket channel ID for real-time status updates (nullable)
<code>printer_name</code>	Target printer name
<code>copies</code>	Number of copies to print
<code>output_file_name</code>	Filename of the PDF sent to the printer
<code>output_base64_string</code>	Base64-encoded PDF (consumed by the print service)
<code>error_message</code>	Error detail if printing failed
<code>status</code>	Current print status (see below)

**Status values** (`status`):

Value	Description
<b>pending</b>	Created, waiting for the print service
<b>printed</b>	Successfully printed and confirmed
<b>error</b>	Printing failed
unknown	Status cannot be determined

**i Real-time updates via WebSocket** only apply when `broadcastId` is provided in the render request. The C# print service subscribes to that channel, picks up the task, executes the print job, and reports status back. Without a `broadcastId`, the task is created silently — the print service must poll for new tasks.

## Audit Trail

Every model in JasperWho? tracks who created and last updated a record, and which API token was used. This information is available on all records via the `withAudit=true` query parameter in the API.

Field	Description
<code>created_at</code>	Timestamp of creation
<code>created_by</code>	User ID of the creator
<code>created_by_token_id</code>	API token ID used (if created via API)
<code>updated_at</code>	Timestamp of last update
<code>updated_by</code>	User ID of the last updater

Field	Description
<code>updated_by_token_id</code>	API token ID used (if updated via API)

The `creationMethod` and `updateMethod` fields in the API response (`"Frontend"` vs. `"API"`) are derived automatically — based on whether a token was present on the request.

---

## Environment Configuration

JasperWho?'s runtime behaviour is controlled via environment variables in `.env` or as container environment variables.

Variable	Default	Description
<code>APP_SCHEME</code>	<code>http</code>	URL scheme for generated links ( <code>http</code> or <code>https</code> )
<code>API_RATE_LIMIT_PER_MINUTE</code>	<code>10</code>	Max API requests per minute per token
<code>PURGE_HISTORY_DAYS</code>	<code>30</code>	Age in days after which history records are purged
<code>PURGE_PRINTTASKS_DAYS</code>	<code>30</code>	Age in days after which print tasks are purged
<code>PURGE_ORPHANED_FILES_DAYS</code>	<code>30</code>	Age in days after which orphaned files on disk are purged
<code>PAGINATION_DEFAULT_COUNT</code>	<code>25</code>	Default number of results per API response

---

Revision #13

Created 2026-05-10 10:46:59 UTC by Benjamin Fischer

Updated 2026-05-23 09:45:26 UTC by Benjamin Fischer