Meet the frontend

VeloxFactory comes with a built-in web frontend that gives you full access to every feature without writing a single line of code. It is built with Laravel Livewire, a reactive framework that delivers a dynamic, app-like feel while keeping everything server-rendered. No separate JavaScript build, no SPA complexity.

Screenshot placeholder: `report-config.index.png`
Report Config overview, the landing page after login, with context badges and thumbnails.

Navigation

After logging in you land directly on the **Report Configs** overview, the central workspace. Dedicated scan users without edit rights for report configurations land on **Scan2Print** instead. All other sections are reachable from the main navigation.

Section	What you manage here
Report Configs	Your report templates, the heart of VeloxFactory
Report History Records	Every past rendering with status, payload, and PDF
Report Print Tasks	Print jobs dispatched to the print service
Mail Queue	Every mail that was sent or is waiting to go out, see Report Mailing
Job Scheduler	Recurring renders and the cleanup run, plus the log of every run, see The Job Scheduler
Scan2Print	Mobile scan station that turns every barcode scan into a print job, see Scan2Print: Printing labels by scanning
Report Contexts	Organisational labels and visual tags for reports
Report Connection Configs	Live database connections for SQL-driven reports
Common Report Resources	Shared graphic assets, used by reports and as inline graphics in mails
Printers	Master data for your physical printers, see Printers

Section	What you manage here
Mailers	Master data for your SMTP accounts, see Mailers
Mail Templates	Subject, body and whitelabeling of your mails, see Mail Templates
Users	User accounts and API token management

The last seven entries are grouped under **Configuration** in the main navigation. They are master data you set up once and then pick by name everywhere else.

The Lookup Pattern

Every section opens with a **list view**, powered by a Livewire Lookup component. These views work the same way across all sections, so once you know one, you know them all.

Screenshot placeholder: `report-config.index.filtered.png`
Report Config overview with the filter bar open and active filter badges.

Full-Text Search

A search bar at the top filters records instantly as you type, no page reload, no submit button. Depending on the section, the search covers names, descriptions, file names, and query text.

Column Filters

Each list view offers contextual filter dropdowns tailored to the entity. For Report Configs, for example, you can filter by Context, Data Adapter, Creator, or date ranges for creation and last update. Active filters are indicated by a badge count on the respective dropdown so you always see at a glance which filters are in play.

Pagination

Results are paginated. The default page size is controlled by the `PAGINATION_DEFAULT_COUNT` environment variable (default: 25). See Configuration and Data Models for all available environment settings.

Persistent Filters and Paging

Every list view remembers its state. The search term, all column filters (checkboxes, dropdown selections, date ranges) and the current page are stored in your session as soon as you change them. You can open a record, switch to another section and come back later, the list shows exactly the same filters and the same page as before.

The state is kept separately for each list view. Filters set on History Records do not affect Report Configs or any other section.

To start over, click the **reset button** of the respective list view, the dark button with the curved arrow icon between the search bar and the green **Filter** button. It clears the search term and all filters of this view and returns to the first page. All other views keep their filters.

> **i Filters are stored per user session.** They survive page reloads and navigation, but are cleared on logout or when the session expires. Filters are never shared with other users or other browser sessions.

Working with Report Configs

The Report Config section has the richest set of actions and is where most of your day-to-day work happens.

Screenshot placeholder: `report-config.edit.png`
Report Config edit view with parameters, fields and resources.

Uploading a Template

Report templates are designed in **Jaspersoft Studio** (compatible version: **6.21.5**) and exported as `.jrxml` files. Once you have a `.jrxml` ready, you upload it to VeloxFactory to create a new Report Config. VeloxFactory immediately analyses the file and automatically creates all associated Parameters, Fields, and Resources, based on what is defined in the template. You do not need to add these manually.

After upload you review the auto-generated records: set example values where missing, assign a Data Adapter if the report uses SQL, and upload any resource files that were detected.

Managing Parameters, Fields, and Resources

Parameters, Fields, and Resources each have their own section within the Report Config edit view. You can edit example values, mark parameters as required, upload resource files, or link a resource to a Common Report Resource, all from the same screen.

Screenshot placeholder: `report-resource.edit.png`
Report Resource edit view with the file upload and the link to a Common Report Resource.

Generating Previews

Once all example values are set and all resource files are uploaded, you can generate a preview rendering directly from the edit view. VeloxFactory renders the report using the stored example values and stores the result as a preview image and thumbnail on the Report Config.

Rendering from the Frontend

You can trigger a full render directly from the frontend, without touching the API. A render dialog lets you fill in parameter values, choose an output type, and optionally dispatch a print job or a mail in the same step. The result is shown inline and logged as a History Record if desired.

Screenshot placeholder: `generate-pdf.create.png`
Generate PDF page with the parameter inputs and the output settings.

Report Connection Configs

When creating or editing a Connection Config, the form includes a **Test Connection** button. Use it before saving, a connection must be in status **approved** before VeloxFactory will use it for rendering. An untested or failed connection will cause render requests to be rejected.

Screenshot placeholder: `report-connection-config.edit.png`
Report Connection Config edit form with the Test Connection button.

Report History Records

The History Records section gives you a full log of every rendering that was saved. For each record you can see the status, the exact parameters and data that were submitted, the raw JasperReports response, and, if the rendering succeeded, the resulting PDF.

From a History Record you can:

- **Download** the PDF directly to your browser, and the xlsx export if the record has one
- **Trigger a print job**, creates a new Print Task for this specific rendering and dispatches it to the print service
- **Send it by mail**, opens a dialog for mailer, template, recipients and attachments and creates a new Mail Task

Screenshot placeholder: `report-history-record.show.png`
Report History Record detail page with status, payload, PDF preview and the actions.

> ℹ **Downloads follow the read permission.** Downloading a rendered PDF, an xlsx export, a report template or a resource file requires the read permission for that resource. Being signed in is not enough.

Report Print Tasks

The Print Tasks section shows all dispatched print jobs and their current status. For each task you can inspect the target printer, number of copies, and any error messages if printing failed.

If a task ended up in error state, you can **reset** it, the task returns to **pending** and the print service will pick it up again.

Screenshot placeholder: `report-print-task.index.png`
Report Print Task overview with status badges, printer and the reset action.

Mailing

A rendered document can be mailed as well as printed. Three sections work together, and all three behave like every other list view described above.

Mailers hold your SMTP accounts: host, port, credentials, sender address and the dispatch rate limits. The form has a **Test Mailer** button and a **Send test mail** action, the same idea as Test Connection on a Connection Config. A mailer has to be active before it is offered anywhere.

Mail Templates hold subject and body. The body is written in a built-in rich text editor, so no HTML knowledge is needed. A button opens the **placeholder catalog**, a searchable table of every token the template can use, each one copyable with a click. A preview shows the finished mail exactly as a recipient will see it, including the colours and the logo, which can be overridden per template.

Mail Queue lists every mail task with recipients, subject, attachments, mode and status. Opening one shows the mail that was actually sent, rendered in place. From there a mail can be **repeated**: a dialog lets you correct the mailer or the recipients before it goes out again, so a mail sent to the wrong address is fixed rather than cloned.

Screenshot placeholder: `mail-template.edit.png`
Mail Template edit view with the rich text editor, the placeholder button and the whitelabel toggles.

Mailing itself is never configured here. It is requested per rendering, in the Generate PDF page, in the Scan2Print settings, in the mail dialog of a History Record, in a render job of the Job Scheduler, or in the `mailing` segment of a render request. The three sections above only provide the building blocks.

Screenshot placeholder: `mailing.history-record-modal.png`
The mail dialog of a Report History Record with mailer, template, recipients and attachments.

> ℹ Full detail is on the dedicated pages: Report Mailing, Mailers and Mail Templates.

Scheduling

Work that repeats has its own section. **Job Scheduler** lists every schedule with its type, its crontab expression, the next and the last run, and how that run ended. A **Run Log** button opens the full history of every run of every schedule, each with its trace id.

Creating a schedule starts with the type, a render job or the cleanup run, and the form follows from there. The expression is entered either in a guided mode with frequency, time and weekday pickers or as plain crontab syntax; both write the same field, and the next five run times are shown while you type. A render job additionally takes the report, an API token, the output and print settings, and the same parameter and data row tables as the Generate PDF page, here with a switch per field for entering a placeholder such as `[now.date]` instead of a fixed value.

Run now starts a schedule immediately without moving its next slot, which is how a new one is tried before it is switched on.

Screenshot placeholder: `scheduler.overview.png`
The Job Scheduler overview with type, cron expression, next run, last run and status per schedule.

> ℹ Full detail is on the dedicated pages: The Job Scheduler, Render Jobs and The Purge Job.

Users and API Tokens

User accounts are managed under the **Users** section, accessible to administrators only. Each user can hold one or more named API tokens, which grant API access under that user's identity.

From the user edit view you can:

- Create a new token and copy it immediately after generation (it is only shown once)
- Revoke any existing token with immediate effect
- Reset another user's password, or change your own after confirming the current one

Screenshot placeholder: `user.edit1.png`
User edit view, account data and API token management.

Screenshot placeholder: `user.edit2.png`
User edit view, the permission sections.

Screenshot placeholder: `create-token.modal.png`
Create API token dialog, the token is shown once.

> ⚠ **Copy your token immediately after creation.** VeloxFactory only displays the token value once. If you lose it, you will need to revoke the old token and create a new one.

> ⚠ **Tokens are issued and revoked here, never through the API.** A request that authenticates with an API token cannot create or revoke tokens and is answered with `403 API tokens can only be managed from the web interface`. That is deliberate: a token that could issue another one would survive its own revocation, and revoking a leaked token would not actually lock it out.

Permissions

Every user in VeloxFactory has a set of permissions that controls what they can see and do, both in the frontend and via the API. Since both use the same underlying controllers, **a permission that restricts an action in the frontend restricts the exact same action via API, and vice versa.** There is no way to grant API-only or frontend-only access to a resource.

Permissions fall into two categories: **global** and **per-resource**.

Global permissions:

Permission	Effect

`global:admin`	Full access to everything, including user management. Only an administrator may grant or withdraw permissions, activate or deactivate accounts, and manage other users' API tokens
`global:use-api`	**Required for every API request.** Without it, each call answers `403 This user is not permitted to use the API`. Also allows managing one's own API tokens in the web interface

Per-resource permissions, available for each of the following resources: `report-config`, `report-connection-config`, `report-context`, `report-history-record`, `report-print-task`, `report-mail-task`, `common-report-resource`, `printer`, `mailer`, `mail-template`, `scheduled-job`:

Permission	Effect
`<resource>:full`	Full read/create/update/delete access to this resource
`<resource>:read`	View records only
`<resource>:create`	Create new records
`<resource>:update`	Edit existing records
`<resource>:delete`	Delete records

`global:admin` always implies full access to all resources and overrides any per-resource setting.

`scheduled-job:*` also covers the run log. Changing or starting a render job is limited to its owner and to administrators, because it acts with the owner's permissions; reading it is not.

Feature permissions:

Permission	Effect
`scan2print:use`	Access to Scan2Print. Implicitly includes `report-config:read`, `report-print-task:read` and `:create`, `report-history-record:create` and `:update`
`scheduled-job:create`	Creating a schedule means rendering, printing and mailing with it, so it implicitly includes `report-config:read`, `report-print-task:create`, `report-mail-task:create` and `report-history-record:create` and `:update`. `scheduled-job:full` does the same

> ℹ **Permissions are administrator territory.** Users can edit their own name and e-mail address, but not their own permissions. The permission section is not shown to them, and a request that carries a permission list is answered with the user's current set instead. The same applies to activating and deactivating an account. VeloxFactory also refuses to let the last remaining administrator give up `global:admin` or be deactivated, so an instance can never end up without one.

> ℹ **API token permissions are inherited from the user.** A token does not have its own permission set, it acts with exactly the permissions of the user it belongs to, evaluated on every single request. Withdrawing a permission takes effect on the next call, there is nothing to re-issue. Deactivating a user revokes all of their tokens at once, and tokens expire after the period configured in `SANCTUM_EXPIRATION` (one year by default).

Signing in

The login form answers every failed attempt the same way, with `Invalid credentials.` Whether the address belongs to an account, whether that account is deactivated and whether the password was wrong are deliberately indistinguishable, so the form cannot be used to find out which addresses exist on an instance.

Attempts are limited to five per minute, counted per source address and per e-mail address. Beyond that the form answers `429 Too Many Requests` until the minute is over. Successful and failed sign-ins, sign-outs, password changes, permission changes and token operations are written to a dedicated `security` log channel.

> ℹ **Changing your own password needs the current one.** An administrator resetting someone else's password does not. Either way, every API token of that account is revoked afterwards, so a password change really does end all access that was granted before it.

Frontend vs. API - What is the Difference?

The short answer: there is no feature difference. The frontend calls the exact same controller methods as the API. Everything you can do in the UI, you can also automate via API.

The frontend is optimised for interactive, human-driven workflows, exploring reports, reviewing history records, testing connections. The API is the right choice when you want to integrate rendering into external systems, automate repetitive tasks, or process results programmatically.

> ℹ Ready to explore the API? See Meet the API for authentication, query parameters, and a practical render example.

Revision #33
Created 2026-05-10 10:42:51 UTC by Benjamin Fischer
Updated 2026-10-02 18:37:36 UTC by Benjamin Fischer