



CONTENT MANAGEMENT SYSTEM

SHIO

Headless CMS & Website Engine

D O C U M E N T A T I O N

v 2026.1

May 11, 2026



viglet

viglet.org

Shio CMS

Content Management System

PRODUCT	Shio CMS — Headless Content Management System
VERSION	2026.1
GENERATED	May 11, 2026
WEBSITE	viglet.org
SOURCE	github.com/openviglet/shio
LICENSE	GNU General Public License v3.0

Shio CMS is a headless content management system that lets you model content with custom Post Types, query it via GraphQL and REST API, and build websites using server-side JavaScript with Hazelcast distributed caching and Viglet Turing ES search integration. This document covers architecture, installation, configuration, content modeling, website development, and the developer guide.

Table of Contents

What is Shio CMS?	1
Core Concepts	5
Shio CMS — Architecture Overview	11
Viglet Shio CMS: Installation Guide	20
Configuration Reference	31
Content Modeling	41
Website Development	48
Search & Caching	53
Administration Guide	58
Developer Guide	64
REST API Reference	71
GraphQL	81
Security	85
Import & Export	88

What is Shio CMS?

Viglet Shio CMS is an open-source headless Content Management System. It allows you to model content with custom Post Types, query it via GraphQL or REST API, and build websites using server-side JavaScript with native caching and integrated full-text search.

Whether you need a traditional CMS with page rendering, a headless API for a single-page application, or a hybrid approach, Shio CMS adapts to your architecture.

What can you do with Shio CMS?

Model your content

Define custom **Post Types** with flexible fields — text, HTML, file uploads, relationships, date pickers, code editors, and more. Each Post Type becomes a reusable content template that editors can populate through the admin console.

Build websites with JavaScript

Create **Page Layouts** and **Regions** using server-side JavaScript. The Nashorn/Node.js engine renders pages dynamically, and the `shObject` API gives you access to content, navigation, and URL generation directly from your templates.

Query content via GraphQL

Use the built-in **GraphQL** endpoint with an interactive **GraphiQL** console to query posts, folders, and sites. Perfect for decoupled frontends and mobile applications.

Integrate with any stack

Consume Shio CMS from your application via **REST API** or **GraphQL**. All content operations — CRUD, search, file upload, user management — are available as JSON endpoints.

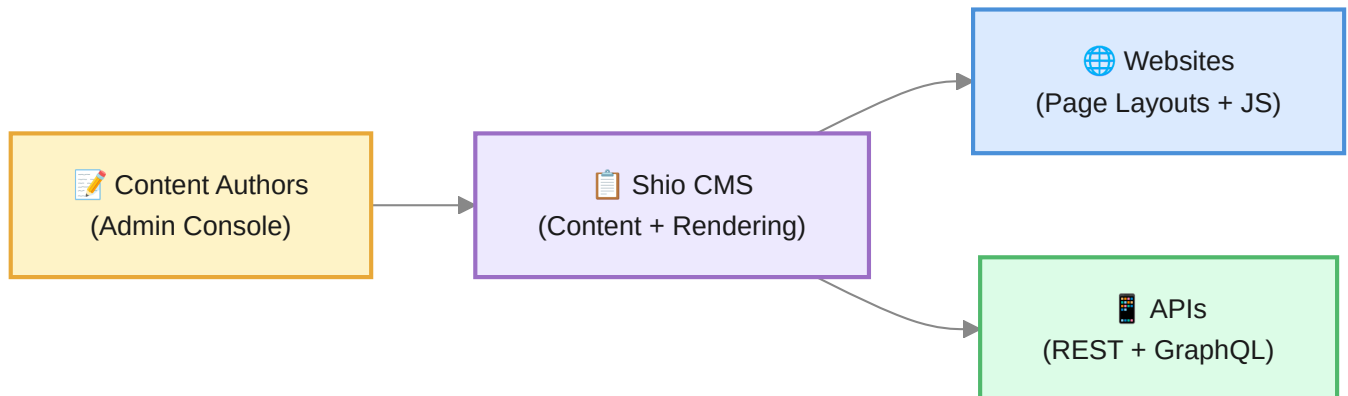
Search your content

Contents are indexed automatically and searchable through the embedded search engine. For advanced search, Shio integrates with **Viglet Turing ES** for faceted search, semantic navigation, and generative AI.

Cache for performance

Hazelcast distributed caching ensures your rendered pages and content objects are served fast. Cache invalidation happens automatically when content is updated.

How it works at a glance



Content authors create and manage content through the admin console. Shio CMS stores it, renders websites using JavaScript templates, and exposes it via REST and GraphQL APIs.

Key concepts

These are the main building blocks you will work with in Shio CMS. You do not need to understand all of them before getting started — come back to each one as you need it.

CONCEPT	WHAT IT IS	LEARN MORE
Site	The top-level container for a website. Defines URL, theme, and Post Type associations.	Core Concepts
Folder	Organizes content hierarchically within a site. Folders can be nested.	Core Concepts
Post	A content item — an instance of a Post Type with field values.	Core Concepts
Post Type	A content model that defines fields, labels, and publishing rules.	Content Modeling
Page Layout	A template that controls how a page is structured — defines regions and rendering logic.	Website Development
Region	A section within a Page Layout that renders content using component APIs.	Website Development
Component API	Reusable content source within a Region — Navigation Component, Query Component, etc.	Website Development
Publishing	Content lifecycle: Draft, Published, Stale, Unpublished.	Content Modeling
GraphQL	Query interface for accessing content programmatically.	GraphQL
Hazelcast Cache	Distributed cache for rendered pages and objects.	Search & Caching

Where to go next

I want to understand how Shio CMS works

Read [Core Concepts](#) first, then [Architecture Overview](#).

I want to set up Shio CMS

Go to the [Installation Guide](#).

I want to model content

Start with [Core Concepts](#) then go to [Content Modeling](#).

I want to build a website

Go to [Website Development](#) for Page Layouts, Regions, and JavaScript API.

I want to query content from an external app

See [GraphQL](#) or [REST API](#).

I want to manage users and permissions

Go to [Administration Guide](#).

Core Concepts

This page introduces the fundamental building blocks of Shio CMS. Understanding these concepts will help you navigate the admin console and make informed decisions about how to structure your content.

Sites

A **Site** is the top-level container in Shio CMS. Each site represents a distinct website with its own URL, theme, and content hierarchy.

A site defines:

- Name and description** — human-readable identifiers

- URL** — the base URL for the published site

- Post Type / Page Layout associations** — which Page Layout renders each Post Type

- Searchable content** — which content is indexed for search

- Form folder** — where form submissions are stored

You can host **multiple sites** in a single Shio CMS instance. Each site has its own folder tree, published URL, and configuration.

Folders

Folders organize content hierarchically within a site. They work like directories in a file system — you can nest folders to create a tree structure.

Folders serve as:

- Navigation structure** — regions can render folder hierarchies as menus

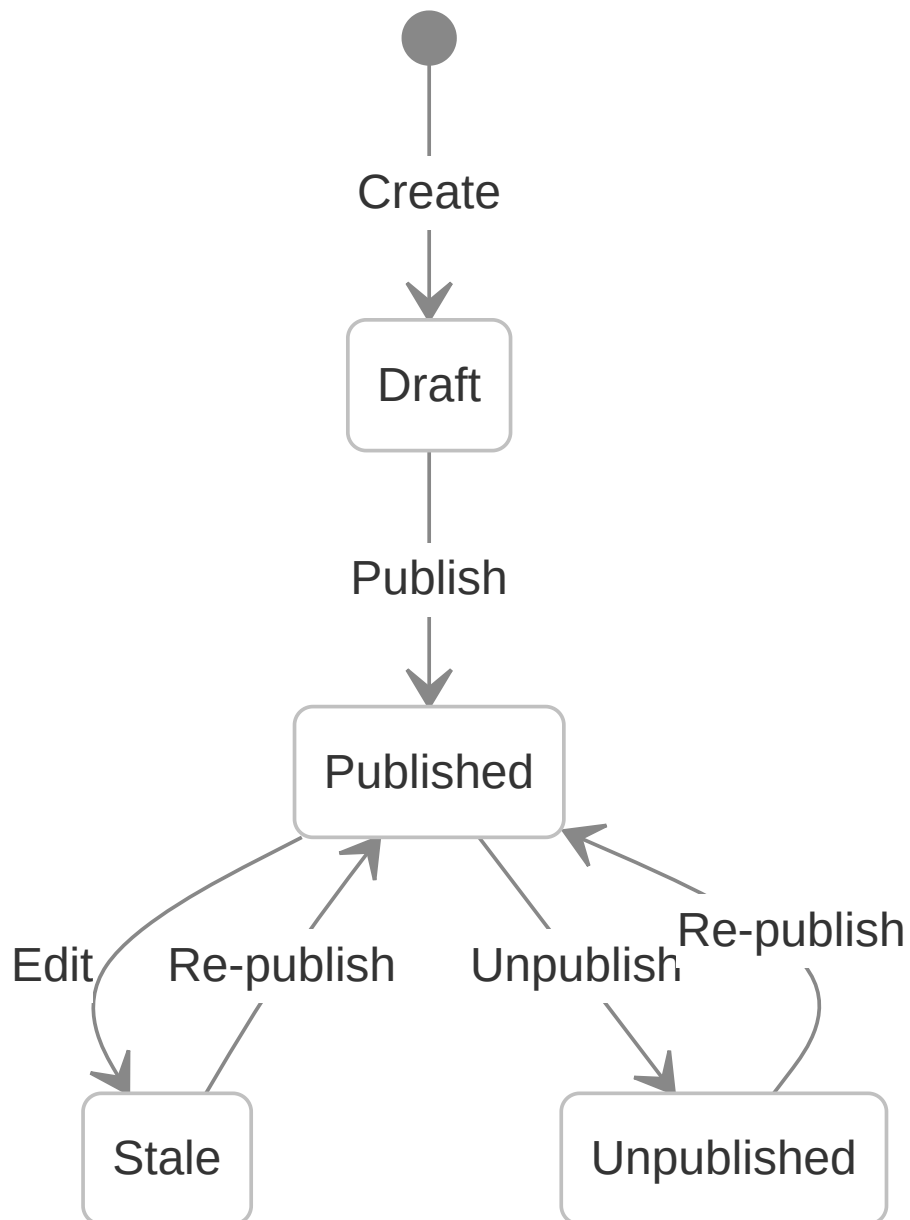
- Content containers** — posts live inside folders

- URL segments** — each folder contributes a segment to the content URL

Posts

A **Post** is a content item — a single piece of content that follows the structure defined by its Post Type. Posts live inside Folders and contain field values (text, images, files, relationships, etc.).

Posts have a **publishing lifecycle**:



STATUS	DESCRIPTION
Draft	Newly created, not yet visible on the published site
Published	Live on the published site
Stale	Published but has been modified — re-publish to update the live version
Unpublished	Removed from the published site but still in the repository

Post Types

A **Post Type** defines the structure of a content item — its fields, labels, and behavior. Think of it as a content template that editors populate when creating Posts.

Shio CMS includes built-in Post Types:

POST TYPE	PURPOSE
Text	General-purpose text content
Photo	Image content with metadata
Video	Video content
Quote	Quoted text
Link	External URL reference
File	Downloadable file
Region	Reusable page section
Theme	Site theme definition
Page Layout	Page template
Alias	URL redirect
Folder Index	Default content for a folder

You can create **custom Post Types** with any combination of field types. See [Content Modeling](#) for details.

Page Layouts

A **Page Layout** is the template that controls how a page is structured and rendered. Each page generated by Shio CMS is associated with exactly one Page Layout.

Page Layouts define:

Regions — sections of the page (header, navigation, content, footer)

JavaScript code — server-side rendering logic

HTML template — the rendering template used by the JavaScript code

The same Page Layout can be applied to multiple folders and posts, simplifying development and management.

Regions

A **Region** is a section within a Page Layout. Each region can render content from the Shio CMS repository using one or more **Component APIs**.

Common region patterns:

REGION	TYPICAL COMPONENT
Header	Navigation Component
Navigation	Navigation Component
Content	Query Component
Footer	Navigation Component

Regions support caching — rendered output is cached by Hazelcast unless the TTL is set to zero.

Component APIs

Component APIs are reusable content sources that regions use to render content. They abstract common content retrieval patterns:

Navigation Component — renders folder hierarchies as menus

Query Component — filters and lists posts from a folder

Components use the `shObject` JavaScript API to access content, generate URLs, and produce rendered HTML.

Views: Management vs Published

Shio CMS provides two views of your site:

VIEW	CONTENT SHOWN	PURPOSE
Management	All content — published, unpublished, drafts	Content editing and administration
Published	Only published content	The live website seen by visitors

Search Integration

Content in Shio CMS is **automatically indexed** for search. The built-in search engine enables basic full-text search on your site.

For advanced search capabilities — faceted navigation, autocomplete, semantic search, and generative AI — Shio integrates with **Viglet Turing ES**. Post Type fields can be mapped to Turing Semantic Navigation fields for precise control over what is indexed and how. See [Search & Caching](#) for details.

Related Pages

PAGE	DESCRIPTION
Architecture Overview	Component diagram and technology stack
Content Modeling	Creating custom Post Types and fields
Website Development	Page Layouts, Regions, and JavaScript API
Installation Guide	Setup with Docker, JAR, or build from source

Shio CMS — Architecture Overview

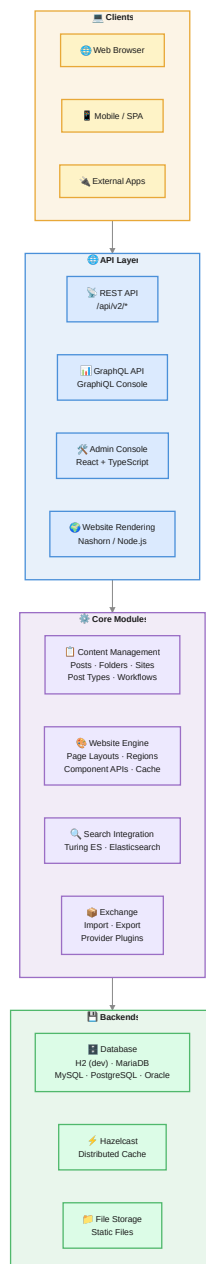
Introduction

Viglet Shio CMS is an open-source headless Content Management System built on **Java 21** and **Spring Boot 4**. It allows organizations to model content with custom Post Types, render websites using server-side JavaScript, and expose content via REST and GraphQL APIs.

The system provides a modern **React** admin console, a **Hazelcast** distributed cache for website rendering, and integration with **Viglet Turing ES** for advanced search capabilities.

This document describes the system's components, internal modules, and the core data flows.

High-Level Component Diagram



API Layer

COMPONENT	DESCRIPTION
REST API	Controllers for posts, folders, sites, post types, users, groups, search, widgets, workflows, and file management (<code>/api/v2/*</code>)
GraphQL API	Content query interface with built-in GraphQL console
Admin Console	React + TypeScript + Radix UI + TailwindCSS admin interface for content management
Website Rendering	Server-side JavaScript engine (Nashorn/Node.js) renders Page Layouts and Regions into HTML

Core Modules

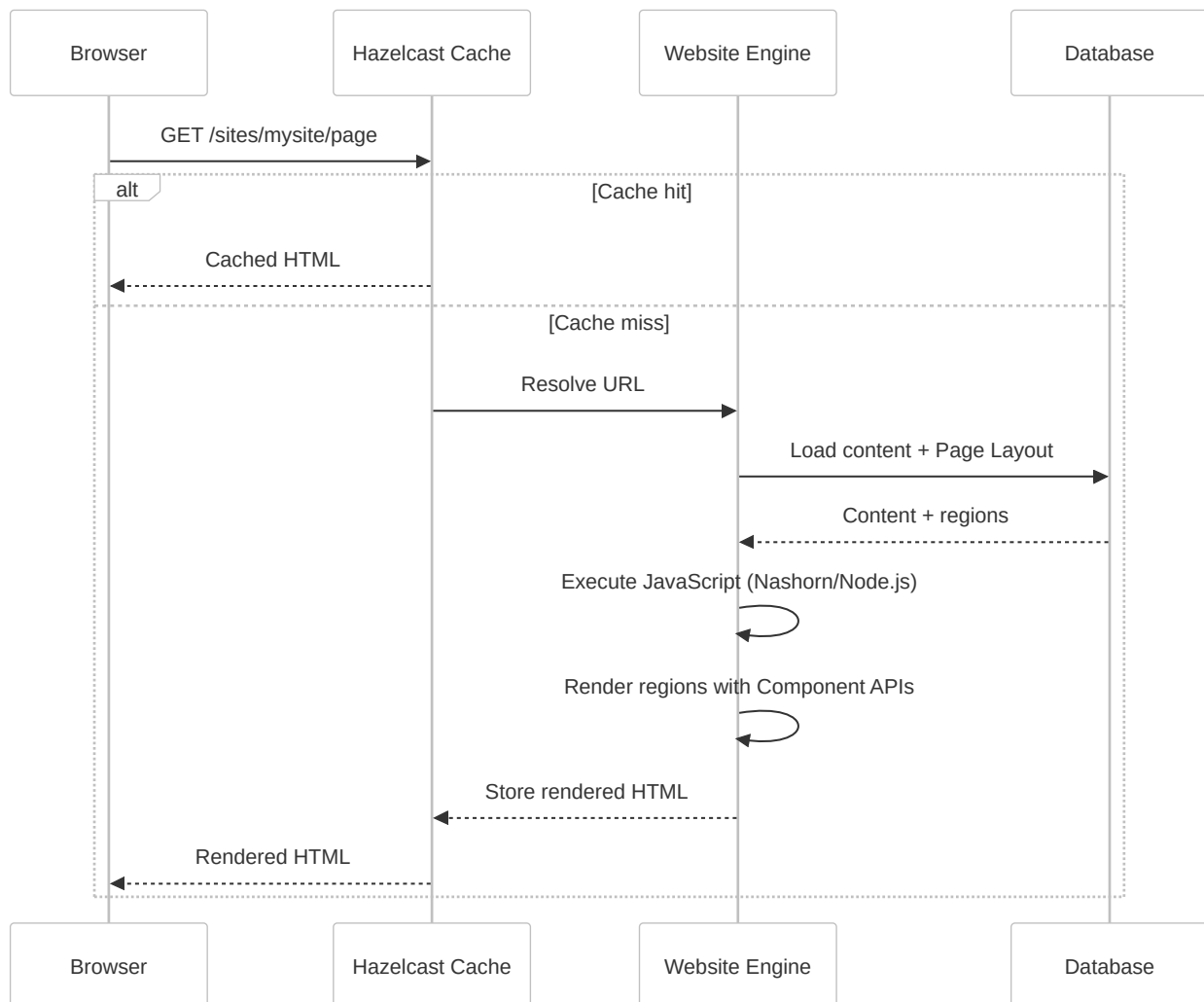
MODULE	PACKAGE	RESPONSIBILITY
Content Management	<code>api</code> , <code>persistence</code>	CRUD for Posts, Folders, Sites, Post Types; publishing workflow; change history; references
Website Engine	<code>website</code>	Page Layout rendering, Region evaluation, component API execution, Hazelcast cache integration
Search Integration	<code>turing</code>	Automatic content indexing, Turing ES REST API integration, search field mapping
Exchange	<code>exchange</code> , <code>provider</code>	Import/export of sites, folders, and posts; provider plugins for Blogger, OTCS, OTMM
Security	<code>spring/security</code>	User, role, and group management; Spring Security integration; CSRF protection
Webhook	<code>webhook</code>	Incoming webhook handling for external integrations

Backends & Storage

BACKEND	PURPOSE	NOTES
Database	Content, metadata, configuration, users	H2 for dev; MariaDB/MySQL recommended for production
Hazelcast	Distributed page and object cache	TTL-based expiration (24h default); automatic invalidation on content change
File Storage	Static files and uploads	Local filesystem with configurable paths; max upload 1 GB

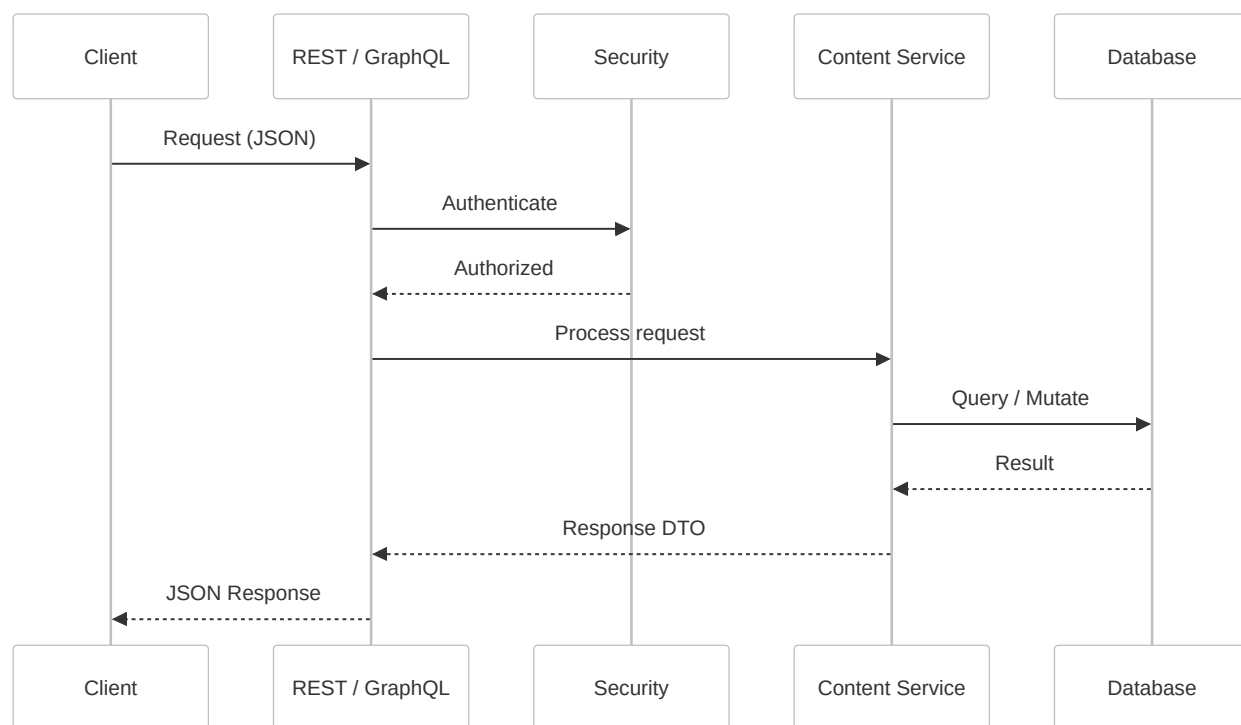
Website Rendering Flow

When a visitor requests a page, the website engine resolves the URL to a content object, loads the associated Page Layout, evaluates each Region with its Component APIs, and assembles the final HTML. Hazelcast caches the rendered output for subsequent requests.



Content API Flow

External applications access content through the REST API or GraphQL endpoint. Both require authentication (except public site endpoints).



Technology Stack

LAYER	TECHNOLOGY	NOTES
Runtime	Java 21	Minimum supported version
Framework	Spring Boot 4.0.4	Application container with auto-configuration
Database	H2 / MariaDB / MySQL / PostgreSQL / Oracle	H2 for development; MariaDB/MySQL recommended for production
Cache	Hazelcast	Distributed cache for website rendering
Search	Elasticsearch 9.3.3 via Viglet Turing SDK	Full-text search integration
JavaScript Engine	Nashorn / Node.js	Server-side rendering of Page Layouts and Regions
Frontend	React 19 + TypeScript + Radix UI + TailwindCSS + Vite	Admin console (<code>shio-react</code>)
Build	Maven (backend) + npm (frontend)	Multi-module project
CI/CD	GitHub Actions	Automated builds and code quality
Containerization	Docker / Docker Compose	Available in project root and <code>containers/</code> directory
Orchestration	Kubernetes	Manifests available in <code>k8s/</code> directory
API Protocols	REST + GraphQL	GraphQL console available in development mode

Deployment Topologies

Development

Minimal setup for local development and evaluation.

Shio CMS (H2 embedded)

Shio CMS starts with an embedded H2 database. No external services are needed. Not suitable for production.

Simple Production

Recommended baseline for production environments.

Shio CMS + MariaDB / MySQL

MariaDB or MySQL provides durable persistence for content, users, and configuration. Hazelcast runs embedded within Shio CMS.

Docker Compose

Complete environment with all dependencies.

Nginx (reverse proxy, ports 80/443)
└─ Shio CMS (port 2710)
└─ MariaDB (port 3306)

All services are defined in the project's `docker-compose.yml`. Volume mounts provide persistent storage for the database and Shio CMS store directory.

Kubernetes

For cloud deployments requiring horizontal scaling.

```
Nginx (reverse proxy)
└─ Shio CMS (1..N replicas)
MariaDB
```

Kubernetes manifests are available in the [k8s/](#) directory.

Related Pages

PAGE	DESCRIPTION
Installation Guide	Setup with Docker, JAR, or build from source
Configuration Reference	All application.properties settings
Developer Guide	Tech stack, dev environment, and contribution guide
Website Development	Page Layouts, Regions, and JavaScript API

Viglet Shio CMS: Installation Guide

Viglet Shio CMS (<https://viglet.org/shio>) is an open-source headless CMS (<https://github.com/openviglet/shio-ce>) that lets you model content, query it via GraphQL, and build websites using JavaScript with native caching and search.

Installing Java

Shio CMS requires Java.

NOTE

Shio CMS only supports Java 21.

During installation, Shio CMS reads the JAVA_HOME and PATH variables to install the necessary services. Therefore, you must modify the variables as specified here:

On Windows:

Set the JAVA_HOME system variable to point to the JAVA location.

Attach the PATH system variable with `%JAVA_HOME%/bin`.

On Unix:

Create an environment variable called JAVA_HOME.

Provide the JAVA bin path in the PATH variable.

Shio CMS Requirements

The information below describes the resource modes available to run Shio CMS.

NOTE

Use a screen resolution of at least 1024 x 768 for optimum clarity.

Hardware Requirements

MINIMUM	RECOMMENDED
Single 1.5 GHz processor, 1 GB available memory	Dual 2 GHz processors with Hyper-Threading enabled, 2 GB available memory

In addition, the following is required:

- 1GB of disk space for software.

- A compatible version of a database.

Installing and Configuring Databases

Shio CMS supports multiple database platforms, each with specific unique requirements to maximize effectiveness with Shio CMS.

Shio CMS Database Table Requirements

Shio CMS uses multiple tables in multiple databases (also called database schemas by some vendors such as Oracle). You must create new databases / schemas that will contain the sets of tables described below.

Oracle Database Requirements

This section describes configuration requirements for an existing database.

To set up an Oracle database:

- Install and configure a single database instance for use by Shio CMS.

- Create a database schema to contain the Shio CMS system tables.

- Configure the Shio CMS admin user for the database. For Oracle, the database user with the grant of the CONNECT and RESOURCE roles. Make sure the user has UNLIMITED TABLESPACE and CREATE VIEW privileges.

Microsoft SQL Server Database Requirements

When creating the Shio CMS database, use the following options:

- You must install the Database Engine feature and Management Tools are recommended.

Set the Collation configuration variable to `SQL_Latin1_General_CP1_CS_AS` or `Latin1_General / Case-sensitive / Accent-Sensitive`.

Set authentication mode to mixed mode.

Shio CMS system user cannot be sa.

Configuring the SQL Server Database and Schemas

NOTE

The database and usernames provided are examples only.

To create the SQL Server database:

Select Databases > New Database.

Enter the database name (eg shio) and the size of the data and log files according to the environment.

Go to the Options page and set the following values:

Collation = `Latin1_General_CS_AS` or `SQL_Latin1_General_CP1_CS_AS`

Miscellaneous > Allow Snapshot Isolation = True

Miscellaneous > Is Read Committed Snapshot On = True

```
USE master
GO
ALTER DATABASE shio
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE
GO
ALTER DATABASE shio
SET ALLOW_SNAPSHOT_ISOLATION ON
GO
ALTER DATABASE shio
SET READ_COMMITTED_SNAPSHOT ON
GO
ALTER DATABASE shio
SET MULTI_USER
GO
```

Under Security > Logins, select New Login:

Enter the Login name (for example, shio)

Deselect the Enforce Password Expiration option

Set the Default database

On the User Mapping page:

Default user = shio

Default schema = shio

Select the option for db_owner

PostgreSQL Database Requirements

Installing and configuring the PostgreSQL Database

To install the PostgreSQL database:

Navigate to <https://www.postgresql.org/docs/manuals/> and follow the instructions.

To configure the PostgreSQL instance, open `postgresql.conf` and set:

```
max_connections = 10
max_prepared_transactions = 10
shared_buffers = 512MB
autovacuum = off
```

! IMPORTANT

The autovacuum daemon must be re-enabled after installing and configuring Shio CMS.

Configuring the Shio user, database, and schemas

! IMPORTANT

When you create a database table in PostgreSQL, the field name must be lowercase.

Create a user:

```
CREATE USER shio PASSWORD <password> with createuser;
```

Create the database:

```
CREATE DATABASE shio_db;  
ALTER DATABASE shio_db OWNER TO shio;
```

Log in and create the schema:

```
$psql -U shio -d shio_db
```

```
CREATE SCHEMA shio;  
ALTER SCHEMA shio OWNER TO shio;  
ALTER USER shio SET SEARCH_PATH = shio;
```

Installing Shio CMS

Option 1 — Docker (fastest)

```
docker pull openviglet/shio:2026.1
docker run -p 2710:2710 openviglet/shio:2026.1
```

Option 2 — JAR download

Go to <https://viglet.org/shio/download/> and click on "Download Shio CMS" button to download the `viglet-shio.jar` executable.

Copy the `viglet-shio.jar` to `/appl/viglet/shio/server`:

```
mkdir -p /appl/viglet/shio/server
cp viglet-shio.jar /appl/viglet/shio/server
```

Option 3 — Build from source

```
git clone https://github.com/openviglet/shio-ce.git
cd shio
mvn clean package -pl shio-app
cp shio-app/target/viglet-shio.jar /appl/viglet/shio/server/
```

Database

By default, Shio uses H2 as its embedded database, but if you prefer another database you can create an external properties file and reference it at startup using `--spring.config.additional-location`.

Create `/appl/viglet/shio/server/viglet-shio.properties`:

MySQL

```
spring.datasource.url=jdbc:mariadb://localhost:3306/shio
spring.datasource.username=shio
spring.datasource.password=shio
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

Oracle Database

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/shio
spring.datasource.username=shio
spring.datasource.password=shio
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

PostgreSQL

```
spring.datasource.url=jdbc:postgresql://localhost:5432/shio
spring.datasource.username=shio
spring.datasource.password=shio
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL94Dialect
```

Then start Shio CMS referencing the properties file:

```
java -Xmx1g -Xms1g -jar viglet-shio.jar \
  --spring.config.additional-location=file:/appl/viglet/shio/server/viglet-shio.properties
```

Creating Shio CMS Service on Linux

As root, create a `/etc/systemd/system/shio.service` file with the following lines:

```
[Unit]
Description=Viglet Shio CMS
After=syslog.target

[Service]
User=viglet
Group=viglet
WorkingDirectory=/appl/viglet/shio/server
ExecStart=/appl/java/jdk21/bin/java -Xmx1g -Xms1g \
    -jar /appl/viglet/shio/server/viglet-shio.jar \
    --spring.config.additional-location=file:/appl/viglet/shio/server/viglet-
shio.properties
SuccessExitStatus=143

[Install]
WantedBy=multi-user.target
```

SPRING BOOT 4

Starting with Spring Boot 4, JAR files are no longer directly executable (`./viglet-shio.jar`). You must launch Shio CMS using `java -jar` explicitly. The `.conf` file pattern used in older versions is no longer supported — use `--spring.config.additional-location` to load external properties instead.

Enable and start the service:

```
systemctl daemon-reload
systemctl enable shio.service
systemctl start shio.service
```

Other useful commands:

```
systemctl stop shio.service
systemctl restart shio.service
systemctl status shio.service
```

Accessing the Shio Console

Shio provides remote access through its Web application interfaces at: <http://localhost:2710>.

By default the login/password are: **admin/admin**

Appendix A: Installation Modes

Simple

Minimal setup using an embedded H2 database. Suitable for local development and evaluation only — not for production.

Prerequisites:

- Linux server
- Java 21
- 50 GB HDD
- 2 GB RAM

Target Audience: Development and testing environments.

Docker Compose

Shio CMS and its dependencies installed via Docker Compose, including:

- MariaDB — Shio CMS system tables
- Nginx — reverse proxy on port 80
- Shio CMS

Prerequisites:

- Linux server with Docker and Docker Compose installed

50 GB HDD

4 GB RAM

Target Audience: Teams that want a complete environment without manual service installation. Suitable for QA and production.

Kubernetes

Shio CMS and its dependencies deployed via Kubernetes manifests (available in the [k8s/](#) directory), including:

MariaDB — Shio CMS system tables

Nginx — reverse proxy

Shio CMS

Prerequisites:

Linux server with Kubernetes, or a cloud Kubernetes service (GKE, EKS, AKS)

100 GB storage

4 GB RAM

Target Audience: Cloud deployments requiring horizontal scaling and infrastructure-as-code.

Manual Installation

Each service installed individually following this guide:

MariaDB — Shio CMS system tables

Shio CMS

Prerequisites:

One to two Linux servers

50–100 GB storage per server

Minimum 2 GB RAM per server

Appendix B: Docker Compose

You can start Shio CMS using MariaDB and Nginx.

```
mvn clean package -pl shio-app  
docker-compose up
```

NOTE

If you have problems with permissions on directories, run `chmod -R 777 volumes`

Docker Commands

Shio

```
docker exec -it shio /bin/bash
```

MariaDB

```
docker exec -it shio-mariadb /bin/bash
```

Nginx

```
docker exec -it shio-nginx /bin/bash
```

Configuration Reference

This document lists all configuration properties available in Shio CMS. Properties are set in `application.properties` (embedded) or in an external properties file loaded via `--spring.config.additional-location`.

Shio CMS Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>shio.mgmt.enabled</code>	<code>true</code>	Enable the management interface
<code>shio.git.url</code>	—	Git repository URL for version control
<code>shio.git.token</code>	—	Authentication token for Git repository
<code>shio.website.nashorn</code>	<code>--persistent-code-cache,--optimistic-types=true,-pcc,--class-cache-size=50000</code>	Nashorn JavaScript engine options
<code>shio.website.java.script.engine</code>	<code>nashorn</code>	JavaScript engine: <code>nashorn</code> or <code>nodejs</code>
<code>shio.allowedOrigins</code>	<code>localhost</code>	CORS allowed origins
<code>shio.plugin.blogger</code>	<code>com.viglet.shio.plugin.ShImporterBloggerPlugin</code>	Blogger import plugin class
<code>shio.config.system</code>	<code>/system</code>	System configuration path
<code>shio.config.auth</code>	<code>/provider/auth/%s</code>	Auth provider path pattern
<code>shio.config.exchange</code>	<code>/provider/exchange/%s</code>	Exchange provider path pattern

Server Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>server.port</code>	<code>2710</code>	HTTP listening port (overridable via <code>PORT</code> env var)
<code>server.compression.enabled</code>	<code>true</code>	Enable HTTP response compression
<code>server.compression.mime-types</code>	<code>application/json,text/css,application/javascript</code>	MIME types to compress
<code>server.compression.min-response-size</code>	<code>2048</code>	Minimum response size (bytes) to compress
<code>server.tomcat.use-relative-redirects</code>	<code>true</code>	Use relative redirects in Tomcat

Database Properties

H2 (Default — Development)

PROPERTY	DEFAULT	DESCRIPTION
<code>spring.datasource.url</code>	<code>jdbc:h2:file:./store/db/testingDB</code>	H2 file-based database URL
<code>spring.datasource.username</code>	<code>sa</code>	Database username
<code>spring.datasource.password</code>	<i>(empty)</i>	Database password
<code>spring.datasource.driver-class-name</code>	<code>org.h2.Driver</code>	JDBC driver class
<code>spring.jpa.properties.hibernate.dialect</code>	<code>org.hibernate.dialect.H2Dialect</code>	Hibernate dialect
<code>spring.h2.console.enabled</code>	<code>false</code>	Enable H2 web console
<code>spring.h2.console.path</code>	<code>/h2</code>	H2 console URL path

MariaDB / MySQL (Production)

```
spring.datasource.url=jdbc:mariadb://localhost:3306/shio
spring.datasource.username=shio
spring.datasource.password=shio
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

PostgreSQL (Production)

```
spring.datasource.url=jdbc:postgresql://localhost:5432/shio
spring.datasource.username=shio
spring.datasource.password=shio
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL94Dialect
```

Oracle (Enterprise)

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/shio
spring.datasource.username=shio
spring.datasource.password=shio
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

JPA Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>spring.jpa.hibernate.ddl-auto</code>	<code>update</code>	Schema management strategy: <code>update</code> , <code>validate</code> , <code>create</code> , <code>none</code>
<code>spring.jpa.show-sql</code>	<code>false</code>	Log SQL statements
<code>spring.jpa.properties.hibernate.format_sql</code>	<code>false</code>	Format logged SQL for readability
<code>spring.jpa.properties.hibernate.enable_lazy_load_no_trans</code>	<code>true</code>	Allow lazy loading outside transactions
<code>spring.jpa.properties.jakarta.persistence.sharedCache.mode</code>	<code>ALL</code>	JPA shared cache mode
<code>spring.jpa.properties.hibernate.generate_statistics</code>	<code>false</code>	Hibernate statistics
<code>spring.jpa.properties.hibernate.globally_quoted_identifiers</code>	<code>true</code>	Quote all identifiers in generated SQL

Logging Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>logging.level.org.springframework</code>	<code>INFO</code>	Spring Framework log level
<code>logging.level.com.viglet</code>	<code>INFO</code>	Viglet application log level
<code>logging.level.org.hibernate.stat</code>	<code>INFO</code>	Hibernate statistics log level
<code>logging.level.org.hibernate.SQL</code>	<code>INFO</code>	Hibernate SQL log level
<code>logging.file.name</code>	<code>store/logs/shio.log</code>	Log file path
<code>logging.logback.rollingpolicy.max-file-size</code>	<code>25MB</code>	Maximum log file size before rotation
<code>logging.logback.rollingpolicy.max-history</code>	<code>10</code>	Number of rotated log files to keep

File Upload Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>spring.servlet.multipart.max-file-size</code>	<code>1024MB</code>	Maximum file upload size
<code>spring.servlet.multipart.max-request-size</code>	<code>1024MB</code>	Maximum request size
<code>spring.mvc.async.request-timeout</code>	<code>3600000</code> <code>0</code>	Async request timeout (ms) — 1 hour

UI Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>spring.web.resources.static-locations</code>	<code>classpath:/ui/public/</code>	Static resource locations
<code>spring.thymeleaf.prefix</code>	<code>classpath:/ui/templates/</code>	Thymeleaf template prefix
<code>spring.output.ansi.enabled</code>	<code>always</code>	ANSI color output in console
<code>spring.web.resources.chain.strategy.content.enabled</code>	<code>true</code>	Content-based versioning for static resources
<code>spring.web.resources.chain.strategy.content.paths</code>	<code>/js/**,/css/**,/img/**,/*.png</code>	Paths for content versioning

Mail Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>spring.mail.host</code>	<code>localhost</code>	SMTP server hostname
<code>spring.mail.port</code>	<code>587</code>	SMTP port
<code>spring.mail.username</code>	<code>localhost@localhost</code>	SMTP username
<code>spring.mail.password</code>	<code>password</code>	SMTP password
<code>spring.mail.properties.mail.smtp.auth</code>	<code>true</code>	Enable SMTP authentication
<code>spring.mail.properties.mail.smtp.starttls.enable</code>	<code>true</code>	Enable STARTTLS
<code>spring.mail.properties.mail.smtp.starttls.required</code>	<code>true</code>	Require STARTTLS
<code>spring.mail.properties.mail.smtp.ssl.enable</code>	<code>false</code>	Enable SSL
<code>spring.mail.test-connection</code>	<code>false</code>	Test mail connection on startup

Management Properties

PROPERTY	DEFAULT	DESCRIPTION
<code>management.endpoints.web.exposure.include</code>	<code>*</code>	Exposed Spring Boot Actuator endpoints

Profiles

PROFILE	PURPOSE
<code>production</code>	Default profile — optimized for production
<code>development</code>	Development mode with additional logging
<code>ui-dev</code>	Headless mode for frontend development with separate Vite dev server

Set the active profile:

```
spring.profiles.active=production
```

Related Pages

PAGE	DESCRIPTION
Installation Guide	Setup with Docker, JAR, or build from source
Architecture Overview	Component diagram and deployment topologies
Developer Guide	Dev environment setup and contribution guide

Content Modeling

Content modeling in Shio CMS revolves around **Post Types** — reusable templates that define the structure of content items. This guide covers how to create and configure Post Types, the available field types, the publishing workflow, and content relationships.

Post Types

A **Post Type** defines:

Name — human-readable label

Description — purpose of the content type

Identifier — unique system key

Publishing responsibility — who can approve and publish

System Post Types

Shio CMS ships with built-in Post Types for common content patterns:

POST TYPE	PURPOSE
Text	General-purpose text content
Photo	Image content with metadata
Video	Video content
Quote	Quoted text
Link	External URL reference
File	Downloadable file
Region	Reusable page section
Theme	Site theme definition
Page Layout	Page template
Alias	URL redirect to another object
Folder Index	Default content for a folder

Custom Post Types

You can create **custom Post Types** with any combination of fields. To create a new Post Type:

Navigate to **Administration > Post Types** in the admin console

Click **New Post Type**

Define the Name, Description, and Identifier

Add fields (see [Field Types](#) below)

Configure publishing responsibility

Field Types

Fields are the building blocks of Post Types. Each field has a **name**, **description**, **identifier**, and **type**. You can configure whether a field serves as the **title** and/or **description** of the Post Type.

FIELD TYPE	DESCRIPTION
Text	Single-line text input
Text Area	Multi-line plain text
HTML Editor	Rich text editor with formatting
Ace Editor (HTML)	Code editor with HTML syntax highlighting
Ace Editor (Javascript)	Code editor with JavaScript syntax highlighting
Content Select	Reference to another content item
Relator	Relationship to one or more content items
Combo Box	Dropdown selection
Multi Select	Multi-value selection
Check Box	Boolean checkbox
Date	Date picker
Hidden	Hidden field (not shown in editor)
Tab	Visual tab separator for organizing fields
Recaptcha	CAPTCHA verification field
Form Configuration	Form submission settings

Field Configuration

For each field you can:

Order fields within the Post Type

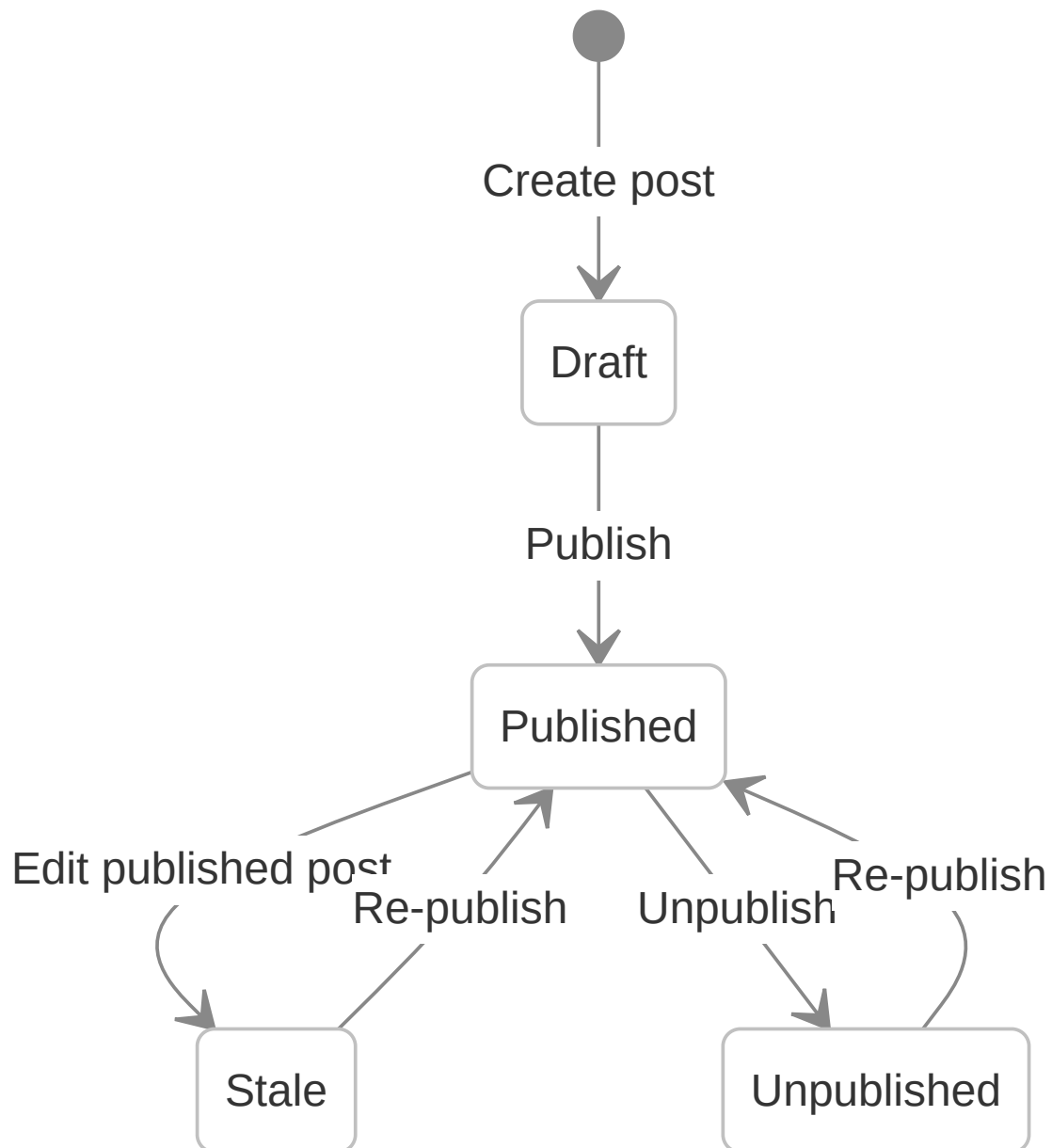
Set as Title — marks the field as the Post Type's title

Set as Description — marks the field as the Post Type's description

Configure search mapping — define how the field is indexed in Viglet Turing ES

Publishing Workflow

Every post follows a lifecycle from creation to publication:



STATUS	VISIBLE ON SITE?	DESCRIPTION
Draft	No	Newly created, only visible in management view
Published	Yes	Live on the published site
Stale	Yes (old version)	Published but modified — re-publish to update
Unpublished	No	Removed from published site, still in repository

Workflow Tasks

Shio CMS supports **workflow tasks** for content approval. Administrators can configure workflows that require approval before content is published.

Content Relationships

References

Posts can reference other posts using **Content Select** and **Relator** fields. References create navigable relationships between content items.

Content Select — references a single content item

Relator — references one or more content items with ordering

Folder Hierarchy

Content is organized in a hierarchical folder structure within each site. Folders can be nested to create a tree. Posts live inside folders.

```
Site
├── Home (Folder)
│   ├── About (Post)
│   └── Contact (Post)
├── Blog (Folder)
│   ├── Post 1
│   └── Post 2
└── Products (Folder)
    ├── Category A (Folder)
    │   ├── Product 1
    │   └── Product 2
    └── Category B (Folder)
```

Content Ordering

Posts within a folder can be **reordered** via drag-and-drop in the admin console. The order is reflected immediately on the published site pages.

Search Navigation Integration

Post Type fields can be mapped to **Viglet Turing ES** Semantic Navigation fields for precise search indexing:

Search Field Association — map a field to a default Turing SN field (title, description, text, date, URL, image)

Create Additional Search Field — map a field to a custom Turing SN field

This allows you to control exactly what content is searchable and how it appears in search results. See [Search & Caching](#) for details.

Spreadsheet Export

You can generate a **spreadsheet** of a folder's contents. Each Post Type in the folder becomes a separate sheet, with columns for each field. This is useful for bulk content review and reporting.

Related Pages

PAGE	DESCRIPTION
Core Concepts	Sites, Folders, Posts, and Page Layouts
Website Development	Page Layouts, Regions, and JavaScript API
Search & Caching	Turing ES integration and Hazelcast cache
REST API	Post, Post Type, and folder API endpoints

Website Development

Shio CMS renders websites using server-side JavaScript. This guide covers how to create Page Layouts and Regions, use the component API, and work with the `shObject` JavaScript API.

Page Layouts

A **Page Layout** is the top-level template for rendering a page. Each page is associated with exactly one Page Layout, which defines:

- Regions** — sections of the page (header, navigation, content, footer)
- JavaScript Code** — server-side processing that returns a rendered view
- HTML Code** — the rendering template used by the JavaScript code
- JavaScript Libraries** — custom JavaScript files to include

The same Page Layout can be applied to various folders and posts, greatly simplifying development and site management.

Creating a Page Layout

- Navigate to the **Themes** area in the admin console
- Create a new Page Layout post
- Define the regions, JavaScript code, and HTML template
- Associate the Page Layout with Post Types in the site properties

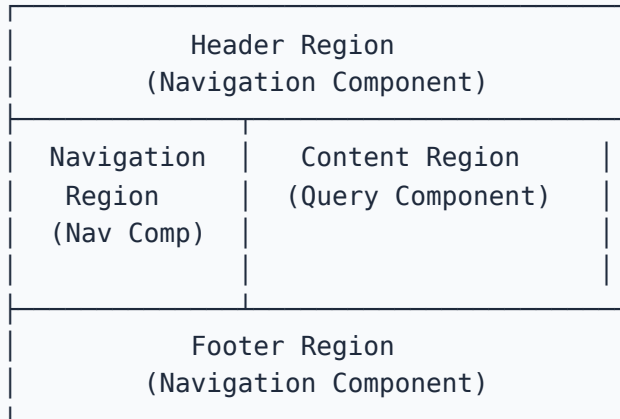
Page Layout / Post Type Association

In the **Site Properties**, you map each Post Type to a Page Layout. This tells Shio CMS which template to use when rendering content of that type.

Regions

A **Region** is a section within a Page Layout. Regions are the building blocks of the page — each region renders a specific area (header, navigation, content, footer).

Each region can call one or more **Component APIs** to fetch and render content from the repository.



Region Caching

Region output is **cached by Hazelcast** by default. Subsequent requests serve the cached HTML until:

- The cache TTL expires (default: 24 hours)

- Content referenced by the region is modified

- The TTL is set to zero (disabling cache for that region)

Component APIs

Component APIs are reusable content sources that regions use to render content. They abstract common retrieval patterns:

Navigation Component

Renders folder hierarchies as menus. Useful for site navigation, breadcrumbs, and sidebar menus.

Query Component

Filters and lists posts from a folder. Useful for content listings, blog rolls, and category pages. You can define sorting, filtering, and pagination.

JavaScript Engine

Shio CMS supports two JavaScript engines for server-side rendering:

ENGINE	CONFIGURATION	NOTES
Nashorn	<code>shio.website.javascript.engine=nashorn</code>	Default engine, built into the JVM
Node.js	<code>shio.website.javascript.engine=nodejs</code>	External Node.js runtime

Nashorn Options

The Nashorn engine can be tuned via the `shio.website.nashorn` property:

```
shio.website.nashorn=-persistent-code-cache,-optimistic-types=true,-pcc,-class-cache-size=50000
```

The `shObject` API

The `shObject` API is available in Page Layouts and Regions for accessing content and generating URLs. It provides methods to interact with the Shio CMS repository from server-side JavaScript.

URL Generation Methods

METHOD	DESCRIPTION
<code>shObject.generateFolderLink(id)</code>	Generates a URL for a Folder
<code>shObject.generatePostLink(id)</code>	Generates a URL for a Post or File
<code>shObject.generateObjectLink(id)</code>	Generates a URL for any object (Folder, Post, or File)

Usage in JavaScript Code

Page Layout JavaScript code uses `shObject` to access the current content item, query related content, and generate the rendered HTML:

```
// Access current post attributes
var title = shObject.post.title;
var content = shObject.post.summary;

// Generate links
var folderUrl = shObject.generateFolderLink(folderId);
var postUrl = shObject.generatePostLink(postId);

// Return rendered HTML
var html = '<h1>' + title + '</h1>';
html += '<div>' + content + '</div>';
html += '<a href="' + folderUrl + '">Back to folder</a>';
```

! INFO

For the complete JavaScript API documentation, see the [JavaScript API reference](#).

Development Workflow

Creating a Site Theme

Create Page Layouts — define templates for different content types

Create Regions — define reusable sections (header, footer, navigation, content)

Associate Components — add Navigation and Query components to regions

Write JavaScript — implement rendering logic using the `shObject` API

Configure Site — associate Post Types with Page Layouts in site properties

Preview

Use the **View Site** button in the admin console to preview the published site. The management view shows all content (including drafts), while the published view shows only published content.

Folder Views

The admin console supports two folder views:

List View — table layout with columns

Thumbnail View — grid layout with thumbnails (useful for media folders)

Static Files

Shio CMS provides a **Static File API** for uploading and managing static assets (images, CSS, JavaScript files). Files can be uploaded individually or in bulk via the admin console.

Static files are served directly by the embedded web server and can be referenced from Page Layouts and Regions.

Related Pages

PAGE	DESCRIPTION
Core Concepts	Sites, Folders, Posts, and Page Layouts overview
Content Modeling	Post Types, fields, and publishing workflow
Search & Caching	Hazelcast cache configuration and search integration
REST API	API endpoints for content operations

Search & Caching

Shio CMS provides automatic content indexing with full-text search capabilities, and a distributed caching layer powered by Hazelcast for high-performance website delivery.

Full-Text Search

Automatic Indexing

Content in Shio CMS is **automatically indexed** when created or updated. This means your site has search functionality out of the box — no additional configuration required.

Built-in Search

The embedded search engine provides:

- Full-text search across all content

- Search API endpoint (`/api/v2/search`)

- Site-scoped search results

Viglet Turing ES Integration

For advanced search capabilities, Shio CMS integrates with **Viglet Turing ES** — an enterprise search platform with semantic navigation and generative AI.

What Turing ES adds

FEATURE	DESCRIPTION
Faceted Search	Filter results by category, date, author, or any attribute
Autocomplete	Type-ahead suggestions as users type
Semantic Navigation	Configurable search experience with result ranking
Generative AI	RAG-powered AI answers grounded in your content
Spotlights	Curated results pinned to specific search terms

Search Field Mapping

When configuring Post Type fields, you can map each field to a Turing Semantic Navigation field:

Search Field Association — map to a default Turing SN field:

Title, Description, Text, Date, URL, Image

Create Additional Search Field — map to a custom field for facets or advanced filtering

Turing Search Endpoint

Shio CMS exposes a Turing search context at:

```
/__tur/sn/{siteName}
```

This endpoint proxies search requests to the configured Turing ES instance, allowing your site to use Turing's full search capabilities without direct client-server communication.

Configuration

Search providers are configured in the **Configuration Console > Search Providers** section. Define:

Turing ES server URL

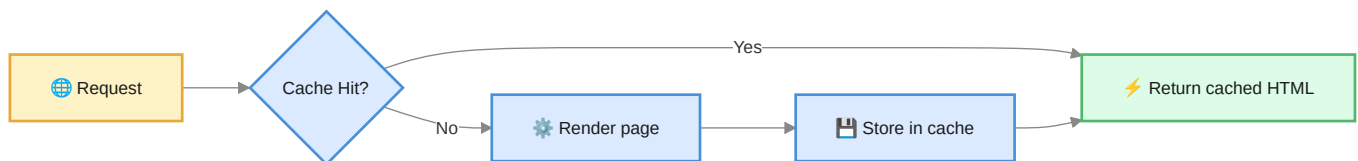
Search site name

Authentication credentials

Hazelcast Cache

Shio CMS uses **Hazelcast** as a distributed caching layer for website rendering. The cache dramatically improves response times by storing rendered page HTML and content objects.

How it works



Cache Behavior

ASPECT	DETAILS
Scope	Page-level and object-level caching
TTL	Default 24 hours (configurable per region)
Invalidation	Automatic when content is created, updated, or deleted
Distribution	Hazelcast distributes cache across multiple Shio CMS nodes
Disable	Set TTL to <code>0</code> on a region to disable caching for that region

Cache Configuration

Hazelcast is configured via `hazelcast.xml` in the application resources. Key settings:

Cluster name — identifies the Hazelcast cluster

Network — multicast or TCP/IP discovery for multi-node setups

Map configuration — TTL, max size, and eviction policies per cache map

Multi-Node Caching

In a multi-node deployment, Hazelcast automatically discovers other nodes and distributes the cache. When content is updated on one node, the cache is invalidated across all nodes.

Related Pages

PAGE	DESCRIPTION
Content Modeling	Search field mapping in Post Types
Website Development	Region caching and TTL configuration
Configuration Reference	Application properties
Architecture Overview	System components and deployment topologies

Administration Guide

This guide covers the administration features of Shio CMS — user management, permissions, site configuration, and system settings.

Users

Administrators can **create, modify, and delete** users through the admin console. Users can access the admin console and/or protected pages on the published site.

Default User

On first startup, Shio CMS creates a default administrator account:

FIELD	VALUE
Username	admin
Password	admin

WARNING

Change the default password immediately after first login.

User Properties

Each user has:

- Username and password
 - Email address
 - Full name
 - Group memberships
-

Groups

Groups organize users into logical sets for permission management. You can:

- Create, modify, and delete groups
 - Add or remove users from groups
 - Assign groups to console or page permissions
-

Permissions

Shio CMS provides two levels of permissions:

Console Permissions

Control who can access the admin console and its features. Assign **groups and users** to console permissions to restrict access to administration functions.

Page Permissions

Control who can view specific pages on the published site. Assign **groups and users** to page permissions to create **protected pages** that require authentication.

! INFO

Protected pages use the users defined in the Administration section for authentication. Visitors must log in with valid credentials to access protected content.

Site Properties

Each site has configurable properties accessible through the admin console:

PROPERTY	DESCRIPTION
Name	Site display name
Description	Site description
URL	Base URL for the published site
Post Type / Page Layout	Association between Post Types and their rendering Page Layouts
Searchable content	Which content is indexed for search
Form folder	Folder where form submissions are saved

Site Management

Creating a Site

- Navigate to **Sites** in the admin console
- Click **New Site**
- Configure site properties
- Choose a pre-defined theme or create a custom one

Importing a Site

Import a previously exported site package:

- Navigate to **Sites**
- Click **Import Site**
- Upload the site package file

Downloading a Site

Export a site as a package for backup or migration:

- Navigate to the site in the admin console
- Click **Download Site**

The site package includes all content, folders, themes, and configuration

Content Administration

Folder Management

Create folders — organize content hierarchically

Reorder posts — drag-and-drop reordering with instant results on site pages

Change folder view — switch between List and Thumbnail views

Navigate via breadcrumb — browse folders through the breadcrumb navigation

Upload files — upload multiple files into a folder at once

Spreadsheet Export

Generate a **spreadsheet** of a folder's contents. Each Post Type in the folder becomes a separate sheet with columns for each field. Useful for bulk content review and reporting.

Version Control

Shio CMS integrates with **Git** for content version control. You can see all commit changes of a site and track content modifications over time.

Configure Git integration in the [Configuration Reference](#):

```
shio.git.url=https://github.com/your-org/your-repo.git  
shio.git.token=YOUR_TOKEN
```

Workflows

Create **workflows** for content approval processes. Workflows define approval steps that content must pass through before publication.

Configuration Console

The Configuration Console provides access to system-wide settings:

Auth Providers

Configure authentication providers:

PROVIDER	DESCRIPTION
Shio Native	Built-in username/password authentication
OTDS	OpenText Directory Service integration

Exchange Providers

Configure content exchange providers for import/export:

PROVIDER	DESCRIPTION
OTCS	OpenText Content Services — import documents
OTMM	OpenText Media Management — import media files

Search Providers

Configure how site search works — connect to Viglet Turing ES for advanced search capabilities.

Email

Configure the **email service** for notifications:

```
spring.mail.host=smtp.example.com
spring.mail.port=587
spring.mail.username=shio@example.com
spring.mail.password=secret
```

Related Pages

PAGE	DESCRIPTION
Security	Authentication and authorization details
Configuration Reference	All application.properties settings
Import & Export	Exchange providers and content migration
Installation Guide	Setup and first-time configuration

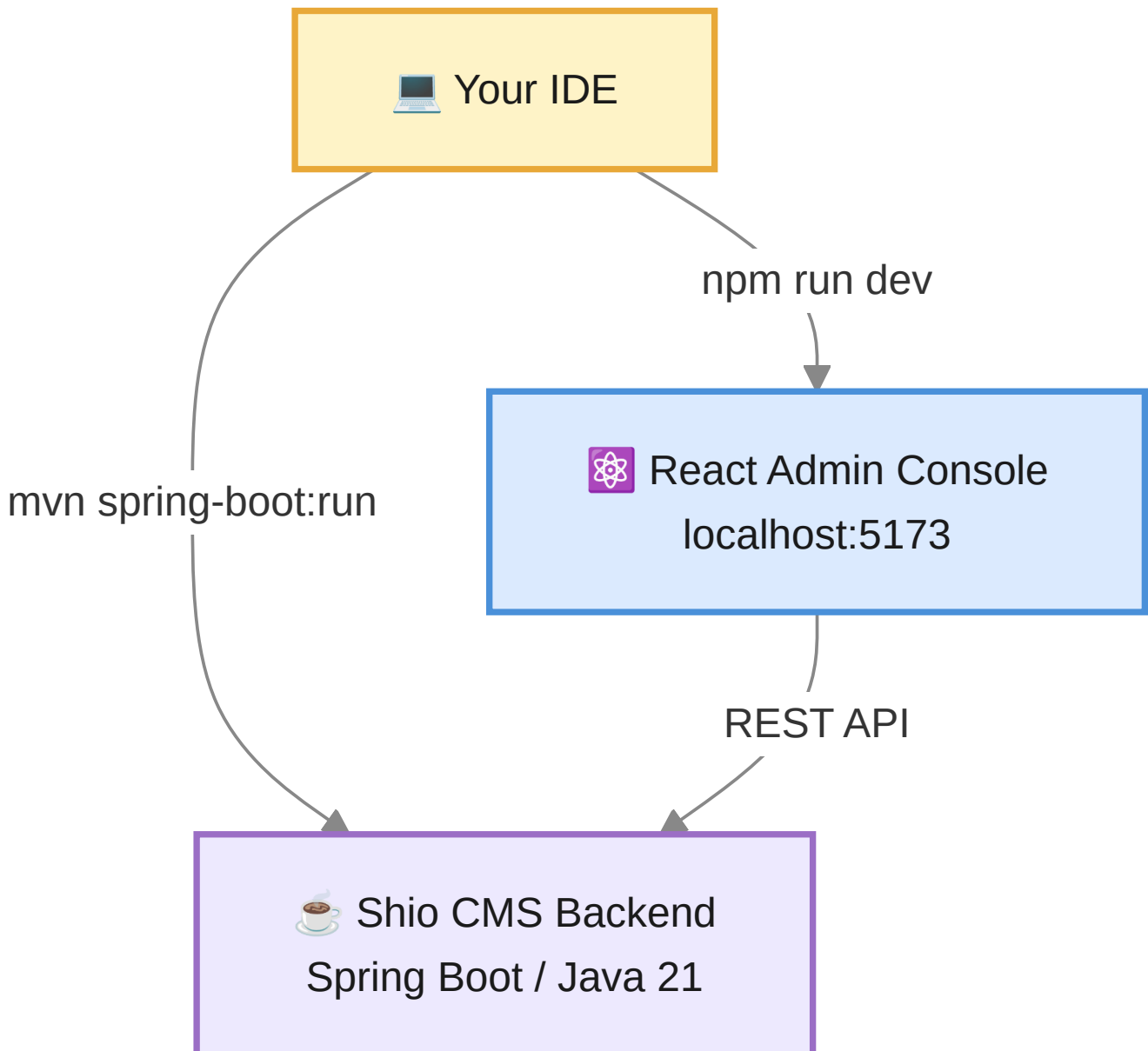
Developer Guide

Whether you're **building a website** on top of Shio CMS or **contributing to the project itself**, this guide has everything you need to get up and running.

Shio CMS is a fully open-source headless CMS with JavaScript-based website rendering, GraphQL, and native caching. The source code lives at github.com/openviglet/shio and all contributions are welcome.

Tech Stack

LAYER	TECHNOLOGY
Backend	Java 21 · Spring Boot 4.0.4
Database	H2 (dev) · MariaDB / MySQL (prod) · PostgreSQL · Oracle
Cache	Hazelcast
Search	Elasticsearch 9.3.3 via Viglet Turing SDK
JavaScript Engine	Nashorn · Node.js
Frontend	React 19 · TypeScript · Radix UI · TailwindCSS · Vite
Build	Maven (backend) · npm (frontend)
CI/CD	GitHub Actions



Setting Up Your Dev Environment

Prerequisites

Before you begin, make sure you have these installed:

Java 21 (Temurin recommended)

Maven 3.9+

Node.js 20+ and npm

Git

Clone the Repository

```
git clone https://github.com/openviglet/shio-ce.git
cd shio
```

Running Shio CMS

Backend (Spring Boot)

Run the full backend including the bundled React console:

```
cd shio
mvn spring-boot:run -pl shio-app
```

The backend starts at <http://localhost:2710>.

Frontend — React Admin Console

For active frontend development, run the React dev server separately:

```
cd shio/shio-react
npm install
npm run dev
```

The Vite dev server starts at <http://localhost:5173> with hot-reload enabled.

Production Build

```
cd shio
mvn clean package -pl shio-app
```

The resulting JAR in `shio-app/target/` bundles both the backend and the compiled React assets.

Development URLs

SERVICE	URL	NOTES
Admin Console	<code>http://localhost:2710</code>	Backend-served
React Dev Server	<code>http://localhost:5173</code>	Vite hot-reload
Sample Site	<code>http://localhost:2710/sites/viglet/default/en-us</code>	Default sample site
GraphiQL Console	<code>http://localhost:2710/graphiql</code>	Interactive GraphQL

ⓘ DEFAULT CREDENTIALS

On first startup the default login/password are: **admin/admin**.

Project Structure

```
shio/
├── shio-app/           # Main Spring Boot application
│   └── src/main/java/com/viglet/shio/
│       ├── api/        # REST API controllers
│       ├── persistence/ # JPA entities and repositories
│       ├── website/     # Website rendering engine
│       ├── exchange/    # Import/export
│       ├── provider/    # Auth and exchange providers
│       ├── graphql/     # GraphQL support
│       ├── turing/      # Turing ES integration
│       ├── spring/      # Spring Security config
│       └── onstartup/   # Initialization
├── shio-react/         # React admin console
├── k8s/                # Kubernetes manifests
├── containers/         # Docker configurations
├── docker-compose.yaml
├── Dockerfile
└── pom.xml
```

Deploy Table

DIRECTORY	DEPLOYED FILE	PROVIDED BY
<code><SHIO_DIR>/</code>	<code>viglet-shio.jar</code>	Shio CMS build
<code><SHIO_DIR>/</code>	<code>viglet-shio.properties</code>	You (optional)

`viglet-shio.jar`

The application JAR is produced by the Maven build and contains the backend, frontend assets, and all dependencies.

`viglet-shio.properties`

Optional external properties file for database configuration and other overrides. See [Configuration Reference](#) for all available properties.

SPRING BOOT 4

Starting with Spring Boot 4, JAR files are no longer directly executable (`./viglet-shio.jar`). You must launch Shio CMS using `java -jar` explicitly. The `.conf` file pattern used in older versions is no longer supported — use `--spring.config.additional-location` to load external properties instead.

REST API

Shio CMS exposes a REST API for all content and administration operations. All endpoints use **JSON** and require authentication (except public site endpoints).

The API base path is `/api/v2`. For the full endpoint reference, see [REST API Reference](#).

Code Quality

TOOL	LINK
SonarCloud	sonarcloud.io/organizations/viglet
GitHub Actions	openviglet/shio/actions

Contributing

We'd love your help making Shio CMS better. Here's how to get involved:

Fork the [openviglet/shio](https://github.com/openviglet/shio) repository.

Create a branch for your feature or fix: `git checkout -b feature/my-improvement`

Commit your changes with clear, descriptive messages.

Open a Pull Request — describe what you changed and why.

For larger contributions, open an issue first to discuss the approach before writing code.



TIP

Check the open [GitHub Issues](#) for good first issues tagged with `good first issue` or `help wanted`.

REST API Reference

Shio CMS exposes a comprehensive REST API for content management, search, and administration. All endpoints use **JSON** for request and response bodies.

Authentication

Most API endpoints require authentication. Shio CMS uses **session-based authentication** with CSRF token protection.

Login

```
POST /api/v2/auth/login
Content-Type: application/json

{
  "username": "admin",
  "password": "admin"
}
```

CSRF Token

For write operations (POST, PUT, DELETE), include the CSRF token in the `X-XSRF-TOKEN` header. The token is returned as a cookie (`XSRF-TOKEN`) after successful authentication.

API Base Path

All endpoints are prefixed with `/api/v2`.

Content Endpoints

Posts

METHOD	PATH	DESCRIPTION
GET	/api/v2/post/{id}	Get a post by ID
POST	/api/v2/post	Create a new post
PUT	/api/v2/post/{id}	Update a post
DELETE	/api/v2/post/{id}	Delete a post

Get Post

```
GET /api/v2/post/{id}
```

Response:

```
{
  "id": "post-uuid",
  "title": "My Post",
  "summary": "Post summary",
  "postType": "Text",
  "folder": "folder-uuid",
  "attributes": {
    "title": "My Post",
    "description": "Post content..."
  },
  "publishStatus": "PUBLISHED",
  "date": "2026-03-27T10:00:00"
}
```

Post Types

METHOD	PATH	DESCRIPTION
GET	/api/v2/post-type	List all Post Types
GET	/api/v2/post-type/{id}	Get a Post Type by ID
POST	/api/v2/post-type	Create a new Post Type
PUT	/api/v2/post-type/{id}	Update a Post Type
DELETE	/api/v2/post-type/{id}	Delete a Post Type

Post Type Attributes

METHOD	PATH	DESCRIPTION
GET	/api/v2/post-type/{id}/attr	List attributes for a Post Type
POST	/api/v2/post-type/{id}/attr	Add an attribute to a Post Type
PUT	/api/v2/post-type/{id}/attr/{attrId}	Update an attribute
DELETE	/api/v2/post-type/{id}/attr/{attrId}	Delete an attribute

Folder Endpoints

METHOD	PATH	DESCRIPTION
GET	/api/v2/folder/{id}	Get a folder by ID
POST	/api/v2/folder	Create a new folder
PUT	/api/v2/folder/{id}	Update a folder
DELETE	/api/v2/folder/{id}	Delete a folder

Site Endpoints

METHOD	PATH	DESCRIPTION
GET	/api/v2/site	List all sites
GET	/api/v2/site/{id}	Get a site by ID
POST	/api/v2/site	Create a new site
PUT	/api/v2/site/{id}	Update a site
DELETE	/api/v2/site/{id}	Delete a site

Search Endpoint

METHOD	PATH	DESCRIPTION
GET	/api/v2/search	Full-text search across content

Query Parameters:

PARAMETER	DESCRIPTION
q	Search query string
site	Site ID to scope the search

User Management Endpoints

Users

METHOD	PATH	DESCRIPTION
GET	/api/v2/user	List all users
GET	/api/v2/user/{username}	Get a user
POST	/api/v2/user	Create a new user
PUT	/api/v2/user/{username}	Update a user
DELETE	/api/v2/user/{username}	Delete a user

Groups

METHOD	PATH	DESCRIPTION
GET	/api/v2/group	List all groups
GET	/api/v2/group/{id}	Get a group
POST	/api/v2/group	Create a new group
PUT	/api/v2/group/{id}	Update a group
DELETE	/api/v2/group/{id}	Delete a group

Roles

METHOD	PATH	DESCRIPTION
GET	/api/v2/role	List all roles
GET	/api/v2/role/{id}	Get a role
POST	/api/v2/role	Create a new role
PUT	/api/v2/role/{id}	Update a role
DELETE	/api/v2/role/{id}	Delete a role

Widget Endpoints

METHOD	PATH	DESCRIPTION
GET	/api/v2/widget	List all widgets
GET	/api/v2/widget/{id}	Get a widget
POST	/api/v2/widget	Create a widget
PUT	/api/v2/widget/{id}	Update a widget
DELETE	/api/v2/widget/{id}	Delete a widget

Workflow Endpoints

METHOD	PATH	DESCRIPTION
GET	/api/v2/workflow	List workflow tasks
GET	/api/v2/workflow/{id}	Get a workflow task
POST	/api/v2/workflow	Create a workflow task
PUT	/api/v2/workflow/{id}	Update a workflow task

File Upload Endpoints

METHOD	PATH	DESCRIPTION
POST	/api/v2/static-file/{folderId}	Upload a static file to a folder

Request: multipart/form-data with file attachment.

NOTE

Maximum file upload size is 1024 MB (configurable via `spring.servlet.multipart.max-file-size`).

History Endpoint

METHOD	PATH	DESCRIPTION
GET	/api/v2/history/{objectId}	Get change history for an object

Reference Endpoints

METHOD	PATH	DESCRIPTION
GET	/api/v2/reference/{id}	Get references for an object

Object Endpoints

METHOD	PATH	DESCRIPTION
GET	/api/v2/object/{id}	Get any object by ID (folder, post, or file)

Preview Endpoint

METHOD	PATH	DESCRIPTION
GET	/api/v2/preview/{id}	Preview content rendering

Import / Export Endpoints

METHOD	PATH	DESCRIPTION
POST	/api/v2/import	Import a site or content package

Provider Endpoints

Exchange Providers

METHOD	PATH	DESCRIPTION
GET	/api/v2/provider/exchange	List exchange providers
GET	/api/v2/provider/exchange/{id}	Get an exchange provider

Auth Providers

METHOD	PATH	DESCRIPTION
GET	/api/v2/provider/auth	List auth providers
GET	/api/v2/provider/auth/{id}	Get an auth provider

Stock Photo Endpoint

METHOD	PATH	DESCRIPTION
GET	/api/v2/stock-photo	Search stock photos

Website Endpoints

These endpoints are publicly accessible (no authentication required):

PATH	DESCRIPTION
/sites/{siteName}/{path}	Access the published website
/__tur/sn/{siteName}	Turing ES search proxy

Related Pages

PAGE	DESCRIPTION
GraphQL	Query content using GraphQL
Developer Guide	Dev environment and API usage
Security	Authentication and authorization

GraphQL

Shio CMS provides a **GraphQL** API for querying content. GraphQL is ideal for decoupled frontends, mobile applications, and any scenario where you need to request exactly the data you need in a single request.

GraphiQL Console

Shio CMS includes a built-in **GraphiQL** interactive console for exploring and testing GraphQL queries:

```
http://localhost:2710/graphiql
```

The console provides:

- Schema explorer** — browse available types and fields
 - Auto-complete** — type-ahead suggestions for queries
 - Query history** — review and re-run previous queries
 - Documentation** — inline documentation for all types
-

Querying Content

Query Posts

Retrieve posts from a specific folder:

```
{
  posts(folderId: "folder-uuid") {
    id
    title
    summary
    postType
    publishStatus
    date
    attributes {
      name
      value
    }
  }
}
```

Query a Single Post

```
{
  post(id: "post-uuid") {
    id
    title
    summary
    postType
    publishStatus
    attributes {
      name
      value
    }
  }
}
```

Query Folders

```
{
  folders(siteId: "site-uuid") {
    id
    name
    parentFolder
    children {
      id
      name
    }
  }
}
```

Query Sites

```
{
  sites {
    id
    name
    description
    url
  }
}
```

Use Cases

USE CASE	WHY GRAPHQL
Single-Page Applications	Fetch exactly the fields you need in one request
Mobile Apps	Minimize payload size by selecting specific fields
Content Aggregation	Combine posts, folders, and site data in a single query
Prototyping	Use GraphQL to explore the content model interactively

Authentication

GraphQL queries require authentication. Use session-based authentication (same as the REST API) or configure API access as needed.

Related Pages

PAGE	DESCRIPTION
REST API Reference	Full REST API endpoint documentation
Content Modeling	Post Types and fields
Developer Guide	Dev environment and project structure

Security

Shio CMS provides a comprehensive security model built on **Spring Security**. This guide covers authentication, authorization, CSRF protection, and access control.

Authentication

Session-Based Authentication

Shio CMS uses **session-based authentication** for the admin console and API. Users log in with username and password, and the server creates a session cookie.

ASPECT	DETAILS
Method	HTTP Basic Authentication
Session	Server-side session with cookie
CSRF	Cookie-based CSRF token (<code>XSRF - TOKEN</code>)
Frame Options	<code>SAME_ORIGIN</code> — allows embedding within the same domain

Auth Providers

Shio CMS supports multiple authentication providers:

PROVIDER	DESCRIPTION
Shio Native	Built-in username/password authentication with BCrypt password encoding
OTDS	OpenText Directory Service — enterprise directory integration

Auth providers are configured in **Configuration Console > Auth Providers**.

Authorization

Protected Paths

PATH PATTERN	ACCESS
<code>/api/v2/*</code>	Requires authentication
<code>/content/*</code>	Requires authentication (admin console)
<code>/sites/**</code>	Public access
<code>/welcome/**</code>	Public access

User Roles

Users are assigned to **groups** which determine their permissions. The permission model supports:

Console access — who can log in to the admin console

Page-level permissions — who can view specific pages on the published site

Protected Pages

Through **Page Permissions**, you can create pages on the published site that require authentication. Only users in the authorized groups can view protected pages.

CSRF Protection

Shio CMS implements **CSRF protection** using a cookie-based token:

The server sets an `XSRF-TOKEN` cookie on login

Clients must include the token value in the `X-XSRF-TOKEN` header for write operations (POST, PUT, DELETE)



TIP

When building API clients, read the `XSRF-TOKEN` cookie value and include it as the `X-XSRF-TOKEN` header in all mutating requests.

CORS Configuration

Cross-Origin Resource Sharing is configured via the `shio.allowedOrigins` property:

```
shio.allowedOrigins=localhost,yourdomain.com
```

HTTP Firewall

Shio CMS uses a **strict HTTP firewall** configuration that blocks:

- Encoded path separators
- Non-normalized URLs
- Suspicious URL patterns

Password Encoding

User passwords are stored using **BCrypt** encoding. The default admin password (`admin`) should be changed immediately after first login.

Related Pages

PAGE	DESCRIPTION
Administration Guide	User, group, and permission management
REST API Reference	API authentication and endpoints
Configuration Reference	Security-related properties

Import & Export

Shio CMS provides built-in import/export functionality for content migration, backup, and integration with external content management systems.

Site Export

Export a complete site as a downloadable package:

- Navigate to the site in the admin console

- Click **Download Site**

The package includes:

- All folders and posts

- Post Types and their attributes

- Page Layouts and Regions

- Themes and static files

- Site configuration

The exported package can be used for backup, migration, or duplication.

Site Import

Import a previously exported site package:

- Navigate to **Sites** in the admin console

- Click **Import Site**

- Upload the site package file

The import process creates all folders, posts, Post Types, and configuration from the package.

API Import

You can also import programmatically via the REST API:

```
POST /api/v2/import
Content-Type: multipart/form-data
```

Exchange Providers

Exchange providers enable content import from external content management systems. Providers are configured in **Configuration Console > Exchange Providers**.

OTCS — OpenText Content Services

Import documents from **OpenText Content Services**:

FEATURE	DETAILS
Source	OpenText Content Services (OTCS)
Content	Documents, folders, and metadata
Authentication	OTCS credentials

OTMM — OpenText Media Management

Import media files from **OpenText Media Management**:

FEATURE	DETAILS
Source	OpenText Media Management (OTMM)
Content	Images, videos, and media assets
Authentication	OTMM credentials

Blogger

Import blog posts from **Google Blogger**:

FEATURE	DETAILS
Source	Blogger blog export
Content	Blog posts, titles, dates, and content
Plugin	<code>com.viglet.shio.plugin.ShImporterBloggerPlugin</code>

Content Operations

Folder Export

Export individual folders and their contents as spreadsheets. Each Post Type in the folder becomes a separate sheet with columns for each field.

Bulk Upload

Upload multiple files to a folder at once through the admin console. Files are automatically stored and accessible via the Static File API.

Related Pages

PAGE	DESCRIPTION
Administration Guide	Site management and configuration
REST API Reference	Import API endpoint
Content Modeling	Post Types and fields