SpringBootLibrarySphinx

Technical Documentation

Generated automatically by GitHub Actions · dipina

SpringBootLibrary Documentation

Welcome to the **SpringBootLibrary** technical documentation portal. This site aggregates all documentation artifacts generated for the project, including architectural guides, Javadoc API reference, test reports, and code quality reports.

Indices and tables

- Index
- Search Page

Project Overview

Purpose

SpringBootLibrarySphinx is an exemplary Spring Boot application developed for the **SPQ (Software Process and Quality)** subject. It demonstrates best practices in:

- RESTful API design with Spring Boot
- Layered MVC architecture
- Automated testing (unit, integration, performance)
- CI/CD pipelines with GitHub Actions
- Code quality tooling (JaCoCo, PMD, Checkstyle, JDepend)
- Technical documentation (Doxygen, Maven Site, **Sphinx**)

The application models a **library borrowing system** where users can borrow and return books through a shared platform.

Key Features

Feature	Technology
REST API backend	Spring Boot 3.4, Spring MVC
Database ORM	Spring Data JPA + Hibernate
Database	MySQL (prod), H2 (test)
API docs (interactive)	Springdoc OpenAPI / Swagger UI
Unit tests	JUnit 5 + Mockito
Integration tests	Spring Boot Test + MySQL
Performance tests	JUnitPerf
Code coverage	JaCoCo
Static analysis	PMD, Checkstyle, JDepend
Containerisation	Docker + Docker Compose
Technical docs	Doxygen, Maven Site, Sphinx

Live Documentation Links

Document	URL
Landing page	https://dipina.github.io/SpringBootLibrarySphinx/
This Sphinx hub	https://dipina.github.io/SpringBootLibrarySphinx/sphinx/index.html
Sphinx PDF	https://dipina.github.io/SpringBootLibrarySphinx/sphinx/SpringBootLibrarySphinx.pdf
Javadoc	https://dipina.github.io/SpringBootLibrarySphinx/site/apidocs/index.html
JaCoCo coverage	https://dipina.github.io/SpringBootLibrarySphinx/site/jacoco/index.html
Surefire unit tests	https://dipina.github.io/SpringBootLibrarySphinx/site/surefire-report.html
Performance tests	https://dipina.github.io/SpringBootLibrarySphinx/site/reports/perf-report.html
Checkstyle	https://dipina.github.io/SpringBootLibrarySphinx/site/checkstyle.html
PMD	https://dipina.github.io/SpringBootLibrarySphinx/site/pmd.html
Doxygen	https://dipina.github.io/SpringBootLibrarySphinx/doxygen/html/index.html
Swagger UI	http://localhost:8080/swagger-ui.html (local only)

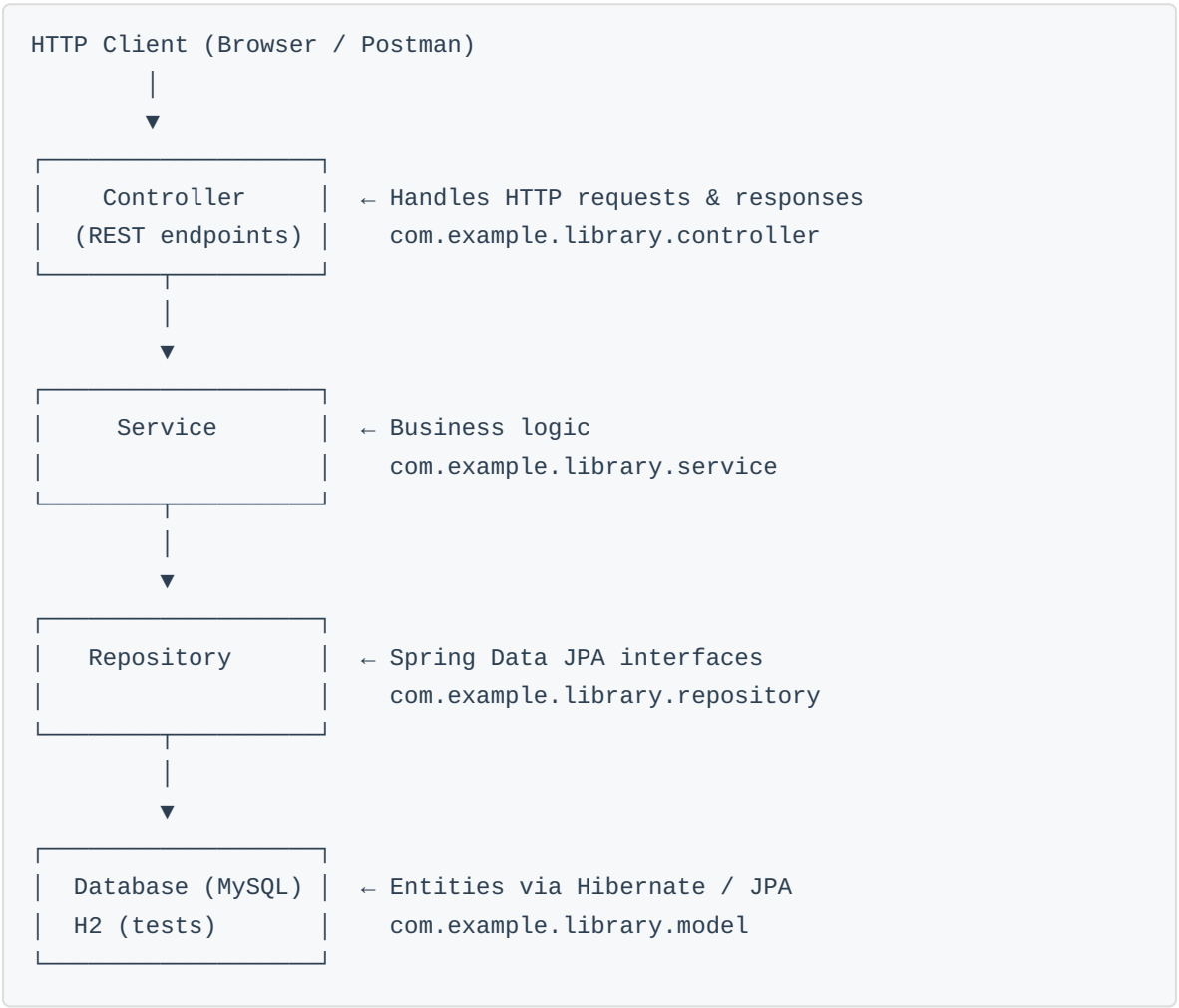
Repository Structure

```
SpringBootLibrarySphinx/
├── .github/workflows/
│   ├── sphinx-docs.yml              ← Sphinx + PDF → GitHub Pages
│   └── maven-site-integration.yml  ← Build, test, Maven site → GitHub Pages
├── docs-sphinx/                     ← All Sphinx source files
│   ├── index.html                   ← Landing page (committed, readable
HTML)
│   ├── Makefile                     ← Linux/macOS build
│   ├── make.bat                     ← Windows build
│   ├── requirements.txt             ← Python dependencies
│   └── source/
│       ├── conf.py
│       ├── index.rst
│       ├── overview.md
│       ├── architecture.md
│       ├── getting_started.md
│       ├── testing.md
│       ├── reports.md
│       ├── api_rest.md
│       ├── javadoc_link.md
│       ├── cicd.md
│       ├── docker.md
│       └── sphinx_101.md
├── src/                             ← Java source code
├── pom.xml                          ← Maven build
└── README.md
```

Architecture

MVC Pattern

SpringBootLibrary follows the classic **Model-View-Controller (MVC)** pattern, adapted for a REST API:



Package Structure

Package	Responsibility
`controller/`	REST endpoints — `BookController`, `BorrowingController`, `UserController`
`service/`	Business logic — `BookService`, `BorrowingService`, `UserService`
`repository/`	JPA repositories — CRUD + custom queries
`model/`	JPA entities — `Book`, `Borrowing`, `User`

Key Design Decisions

Spring Data JPA is used for the persistence layer. Repository interfaces extend `JpaRepository`, providing standard CRUD operations without boilerplate code.

H2 in-memory database is activated automatically in the `test` Spring profile, so unit and integration tests run without requiring a real MySQL instance (except when using `@SpringBootTest` with the `integration` Maven profile, which spins up a MySQL Docker container via GitHub Actions).

Springdoc OpenAPI auto-generates an interactive Swagger UI from the controller annotations at runtime, always reflecting the current API surface.

Static Resources

The frontend is a plain HTML + JavaScript + CSS single-page application located at:

```
src/main/resources/static/
├── index.html
├── app.js
└── style.css
```

It communicates with the backend over REST, keeping the view layer decoupled from the server.

Getting Started

Prerequisites

Tool	Minimum Version
Java (JDK)	17
Maven	3.9
MySQL	8.0
Docker & Docker Compose	24+ (optional)

Database Setup

```
mysql -u root -p < src/main/resources/dbsetup.sql
```

Then update `src/main/resources/application.properties` with your database credentials:

```
spring.datasource.url=jdbc:mysql://localhost:3306/libraryapidb
spring.datasource.username=<your_user>
spring.datasource.password=<your_password>
```

Running the Application

```
# Build and run (all checks)
mvn clean install
mvn spring-boot:run

# Skip tests for a fast start
mvn -DskipTests spring-boot:run
```

Access points once running:

Endpoint	URL
Frontend UI	http://localhost:8080
Swagger UI	http://localhost:8080/swagger-ui.html
OpenAPI JSON	http://localhost:8080/v3/api-docs

Running with Docker

```
# Start all services (app + MySQL)
docker-compose up

# Rebuild after Dockerfile changes
docker-compose up --build
```

Quick Reference: Maven Commands

Goal	Command
Unit tests	`mvn test`
Integration tests	`mvn -Pintegration integration-test`
Performance tests	`mvn -Pperformance integration-test`
JaCoCo coverage report	`mvn clean test jacoco:report`
Full Maven site	`mvn clean verify site`
Publish docs to `docs/`	`mvn post-site`
Build Sphinx docs	`cd docs-sphinx && make html`

Testing Strategy

The project applies a three-tier testing strategy, each with its own Maven profile and tooling.

Unit Tests

Framework: JUnit 5 + Mockito
Location: `src/test/java/com/example/library/` (excluding `integration/` and `performance/`)
Command:

```
mvn test
```

Unit tests mock all external dependencies (database, other services) and run in-process using the H2 in-memory database. They are fast and should run on every commit.

Key test classes:

- `BookServiceTest` — validates book availability, add/remove logic
- `UserServiceTest` — validates user registration and lookup
- `BorrowingServiceTest` — validates borrow/return business rules

Integration Tests

Framework: Spring Boot Test (`@SpringBootTest`) + real MySQL
Location: `src/test/java/com/example/library/integration/`
Maven profile: `integration`
Command:

```
mvn -Pintegration integration-test
```

Integration tests start the full Spring context against a live database. In CI they rely on the MySQL Docker service defined in the GitHub Actions workflow.

Performance Tests

Framework: JUnitPerf
Location: `src/test/java/com/example/library/performance/`
Maven profile: `performance`
Command:

```
mvn -Pperformance integration-test
```

Performance tests annotate methods with `@JUnitPerfTest` to define throughput and latency thresholds. The generated HTML report is saved to `target/reports/perf-report.html`.

Code Coverage

Tool: JaCoCo
Minimum line coverage enforced: 25 %

```
mvn clean test jacoco:report   # generates target/site/jacoco/index.html
mvn clean verify               # enforces the coverage threshold
```

The JaCoCo report is included in the Maven Site and embedded in the Sphinx portal.

Static Analysis

Tool	Purpose	Report
Checkstyle	Code style conformance	`target/site/checkstyle.html`
PMD	Static bug-pattern detection	`target/site/pmd.html`
CPD (via PMD)	Copy-paste detection	`target/site/cpd.html`
JDepend	Package coupling metrics	`target/site/jdepend-report.html`

All static-analysis reports are produced during `mvn site`.

Test & Quality Reports

All reports are generated automatically by the CI/CD pipeline on every push to `main` and published to GitHub Pages alongside the Sphinx portal.

Javadoc API Reference

Generated by `maven-javadoc-plugin` during `mvn site`.

> Note
>
> → **Open Javadoc**

JaCoCo — Code Coverage

> Note
>
> → **JaCoCo Coverage Report**

Minimum enforced: **25 % line coverage** via `mvn verify`.

Surefire — Unit Test Report

> Note
>
> → **Surefire Unit Test Report**

Performance Tests

> Note
>
> → **Performance Report**

Generated by the `performance` Maven profile (`mvn -Pperformance integration-test`), then copied into `target/site/reports/` before `mvn site` runs.

Checkstyle

> Note
>
> → **Checkstyle Report**

PMD — Static Analysis

> Note
>
> → **PMD Report**
>
> → **CPD (Copy-Paste Detector)**

JDepend — Package Metrics

> Note
>
> → **JDepend Report**

Doxygen

> Note
>
> → **Doxygen HTML Reference**

Complete path map

Report	Published path on GitHub Pages	Generated by
Javadoc	`site/apidocs/index.html`	`maven-javadoc-plugin` in `<reporting>`
JaCoCo	`site/jacoco/index.html`	`mvn test jacoco:report`
Surefire	`site/surefire-report.html`	`surefire-report-plugin` `report` goal
Performance	`site/reports/perf-report.html`	`mvn -Pperformance integration-test`
Checkstyle	`site/checkstyle.html`	`mvn site`
PMD	`site/pmd.html`	`mvn site`
CPD	`site/cpd.html`	`mvn site`
JDepend	`site/jdepend-report.html`	`mvn site`
Doxygen	`doxygen/html/index.html`	`doxygen-maven-plugin` via `mvn site`

Generating All Reports Locally

```
# 1. Unit tests + JaCoCo
mvn test jacoco:report

# 2. Performance tests + copy report into site/reports/
mvn -Pperformance integration-test
mvn -Pperformance resources:copy-resources@copy-perf-report

# 3. Full Maven site (Javadoc, Checkstyle, PMD, JDepend, Surefire, JaCoCo,
Doxygen)
mvn site

# 4. Copy everything to docs/ (local preview only — not committed)
mvn post-site
```

REST API Reference

The API is documented interactively via **Springdoc OpenAPI** and is always in sync with the source code.

Swagger UI (Live)

When the application is running locally:

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

The OpenAPI JSON specification can be downloaded from:

```
http://localhost:8080/v3/api-docs
```

Endpoints Summary

Books

Method	Path	Description
`GET`	`/api/books`	List all books
`GET`	`/api/books/{id}`	Get book by ID
`POST`	`/api/books`	Add a new book
`PUT`	`/api/books/{id}`	Update a book
`DELETE`	`/api/books/{id}`	Delete a book

Users

Method	Path	Description
`GET`	`/api/users`	List all users
`GET`	`/api/users/{id}`	Get user by ID
`POST`	`/api/users`	Register a new user
`DELETE`	`/api/users/{id}`	Delete a user

Borrowings

Method	Path	Description
`GET`	`/api/borrowings`	List all borrowings
`POST`	`/api/borrowings`	Borrow a book
`PUT`	`/api/borrowings/{id}/return`	Return a borrowed book
`GET`	`/api/borrowings/user/{userId}`	Get borrowings for a user

Request / Response Examples

Borrow a book

Request:

```
POST /api/borrowings
Content-Type: application/json

{
  "userId": 1,
  "bookId": 42
}
```

Response `201 Created`:

```
{
  "id": 7,
  "userId": 1,
  "bookId": 42,
  "borrowDate": "2025-04-20",
  "returnDate": null,
  "status": "BORROWED"
}
```

Return a book

```
PUT /api/borrowings/7/return
```

Response `200 OK`:

```
{
  "id": 7,
  "returnDate": "2025-04-22",
  "status": "RETURNED"
}
```

Javadoc API Documentation

Full Javadoc is generated by `maven-javadoc-plugin` during `mvn site`.

> Important
>
> → **Open Javadoc**

Why Javadoc was missing — and the fix

Javadoc is **only generated** when `maven-javadoc-plugin` is declared inside the `<reporting><plugins>` section of `pom.xml`. Without it, `mvn site` silently skips Javadoc and `site/apidocs/` is never created.

The plugin has been added to `pom.xml` in this project. The critical block is:

```
<reporting>
  <plugins>
    <!-- This block was missing — without it mvn site skips Javadoc -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.6.3</version>
      <reportSets>
        <reportSet>
          <reports>
            <report>javadoc</report>
          </reports>
        </reportSet>
      </reportSets>
      <configuration>
        <doclint>none</doclint>
      </configuration>
    </plugin>
    <!-- ... other reporting plugins ... -->
  </plugins>
</reporting>
```

The `<doclint>none</doclint>` flag is important — without it, any comment that doesn't conform strictly to Javadoc syntax causes the build to fail.

Key Documented Classes

Class	Package	Description
`BorrowingController`	`controller`	REST endpoints for borrowing — includes `@Operation` (OpenAPI) annotations
`BookService`	`service`	Business logic for book inventory management
`Borrowing`	`model`	JPA entity modelling a borrowing transaction
`BookRepository`	`repository`	Spring Data JPA repository with custom query methods
`UserService`	`service`	User management operations

Generating Javadoc Locally

```
# As part of the full Maven site (recommended — also runs all other reports)
mvn site
# Open: target/site/apidocs/index.html

# Standalone
mvn javadoc:javadoc
# Open: target/site/apidocs/index.html
```

Javadoc vs Doxygen

Tool	Output	Strength
Javadoc	`site/apidocs/`	Standard Java API reference; IDE integration
Doxygen	`doxygen/html/`	Cross-referenced call graphs, class diagrams, richer navigation

CI/CD with GitHub Actions

The project ships with two GitHub Actions workflows:

Workflow file	Trigger	What it does
`maven-site-integration.yml`	Push to any branch + cron	Build, test, Maven site, deploy to Pages
`sphinx-docs.yml`	Push to `main`	Build Sphinx docs + merge into GitHub Pages

Workflow 1: `maven-site-integration.yml`

This is the main CI/CD workflow. It:

1. Starts a MySQL 8 Docker service
2. Waits for the database to be healthy
3. Runs the DB initialisation script
4. Executes integration tests (`-Pintegration`)
5. Generates the full Maven site (Javadoc, JaCoCo, PMD, Checkstyle, Surefire)
6. Copies all Maven site output + Doxygen into the `docs/site/` and `docs/doxygen/` sub-folders
7. Deploys the `docs/` directory to **GitHub Pages**

```
name: Maven Site & Integration Tests

on:
  push:
    branches: ['**']
  schedule:
    - cron: '0 18-23/2 * * *'
    - cron: '0 0-6/2 * * *'

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    services:
      mysql:
        image: mysql:8.0
        env:
          MYSQL_ROOT_PASSWORD: root
          MYSQL_DATABASE: libraryapidb
        ports:
          - 3306:3306
        options: >-
          --health-cmd="mysqladmin ping -h 127.0.0.1 -uroot -proot"
          --health-interval=10s
          --health-timeout=5s
          --health-retries=5

    steps:
      - uses: actions/checkout@v4

      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          java-version: '17'
          distribution: 'temurin'

      - name: Cache Maven
        uses: actions/cache@v4
        with:
          path: ~/.m2/repository
          key: ${{ runner.os }}-maven-${{ hashFiles('**/pom.xml') }}

      - name: Wait for MySQL
        run: |

until mysqladmin ping -h 127.0.0.1 -uroot -proot --silent; do sleep 2; done

      - name: Init database
```

```
        run: mysql -h 127.0.0.1 -uroot -proot < src/main/resources/
dbsetup.sql

      - name: Integration tests
        run: mvn -Pintegration integration-test

      - name: Maven site
        run: mvn site

      - name: Publish docs to GitHub Pages
        uses: peaceiris/actions-gh-pages@v4
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./docs
          destination_dir: .    # serve from root of gh-pages branch
          keep_files: true       # keep Sphinx output from the other workflow
```

Workflow 2: `sphinx-docs.yml`

This workflow builds the Sphinx documentation and deploys it to the `gh-pages` branch **alongside** the Maven site output, without overwriting it.

```yaml
name: Build and Deploy Sphinx Docs

on:
  push:
    branches: [main]
  workflow_dispatch:

jobs:
  sphinx:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.12'

      - name: Install Sphinx dependencies
        run: |
          pip install -r docs-sphinx/requirements.txt

      - name: Build Sphinx HTML
        run: |
          cd docs-sphinx
          make html

      - name: Deploy Sphinx to GitHub Pages (sphinx/ subfolder)
        uses: peaceiris/actions-gh-pages@v4
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./docs-sphinx/_build/html
          destination_dir: sphinx   # → gh-pages/sphinx/
          keep_files: true
```

GitHub Pages Setup

To enable GitHub Pages for this repository:

1. Go to **Settings** → **Pages**
2. Set **Source** to `gh-pages` branch, root `/`
3. Save — the site will be live at `https://<owner>.github.io/SpringBootLibrary/`

> **Permissions:** Go to **Settings** → **Actions** → **General** → **Workflow permissions** and set **Read and write permissions** so that the `GITHUB_TOKEN` can push to the `gh-pages` branch.

Triggering Workflows via API

```
curl -X POST \
  https://api.github.com/repos/<owner>/SpringBootLibrary/actions/workflows/
sphinx-docs.yml/dispatches \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer $PAT_TOKEN" \
  -H "X-GitHub-Api-Version: 2022-11-28" \
  -d '{"ref":"main"}'
```

Docker & Docker Compose

Quick Start

```
# Start application + MySQL (uses cached images)
docker-compose up

# Rebuild after code or Dockerfile changes
docker-compose up --build

# Run in background
docker-compose up -d

# Stop all services
docker-compose down
```

Dockerfile

The project's `Dockerfile` packages the Spring Boot fat JAR into a lightweight JRE image:

```
FROM eclipse-temurin:17-jre-alpine
WORKDIR /app
COPY target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

docker-compose.yml

The compose file wires the app container to a MySQL service, passing environment variables for the database connection:

```
version: '3.8'
services:
  app:
    build: .
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://db:3306/libraryapidb
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
    depends_on:
      db:
        condition: service_healthy

  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: libraryapidb
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5
```

VisualVM Profiling

To enable remote profiling with VisualVM, the `spring-boot-maven-plugin` is configured with:

```
<jvmArguments>-Xverify:none</jvmArguments>
```

Then run `mvn spring-boot:run` and connect VisualVM to the local JVM from the **Local** tab.

Sphinx 101 — Complete Guide

What Is Sphinx?

Sphinx is an open-source documentation generator originally created for the Python language reference documentation (it still powers the official Python docs). Today it is widely used across languages — including Java, C++, and JavaScript projects — whenever developers need to produce structured, professional, multi-page technical documentation from plain text source files.

Sphinx takes plain text files written in **reStructuredText (`.rst`)** or **Markdown (`.md` , via the MyST parser extension)** and generates:

- Static HTML websites (the most common output)
- PDF documents (via LaTeX)
- ePub e-books
- Man pages

The result is a maintainable, version-controlled documentation site that lives alongside the source code and can be automatically published through CI/CD.

What Is Sphinx Useful For?

Use Case	Why Sphinx
Project portals	Aggregates API docs, guides, reports, and tutorials in one navigable site
API reference	Can auto-import Python docstrings; for Java projects, embeds Javadoc/ Doxygen links
Tutorials & how-tos	Numbered sections, admonition boxes, code blocks with syntax highlighting
Multi-version docs	Tools like `sphinx-multiversion` serve docs for every tagged release
Search	Built-in full-text client-side search (no server needed)
Cross-references	`:ref:`, `:doc:` directives create typed links that break at build time if the target disappears
Theming	Dozens of themes (`sphinx_rtd_theme`, `furo`, `pydata_sphinx_theme`) give professional results instantly

Sphinx vs. Alternatives

Tool	Primary audience	Markup	Auto-API	Best for
Sphinx	Python / polyglot	RST / Markdown	Python (autodoc)	Comprehensive project portals
Javadoc	Java	Javadoc tags	Yes (Java)	API reference only
Doxygen	C/C++/Java	Doxygen tags	Yes	Cross-referenced code ref
MkDocs	Any	Markdown	No	Simple Markdown-first sites
Docusaurus	JS / React	Markdown/ MDX	No	Product/SaaS documentation

Sphinx shines when you need a **single portal** that combines hand-written narrative documentation, auto-generated API references, test reports, and external links — exactly the goal of this project.

How Is Sphinx Used?

1. Install Sphinx

```
pip install sphinx sphinx-rtd-theme myst-parser
```

Or pin versions in a `requirements.txt`:

```
sphinx==7.4.7
sphinx-rtd-theme==2.0.0
myst-parser==3.0.1
```

2. Initialise a Project

```
sphinx-quickstart docs-sphinx
```

This interactive wizard creates `conf.py` (the configuration file), `index.rst` (the master table of contents), and a `Makefile`.

3. Source File Structure

```
docs-sphinx/
├── source/
│   ├── conf.py        ← Sphinx configuration
│   ├── index.rst      ← Master table of contents (toctree)
│   ├── overview.md    ← A page written in Markdown
│   ├── api.rst        ← A page written in reStructuredText
│   └── _static/       ← Custom CSS / images
├── Makefile           ← make html, make pdf, …
└── requirements.txt   ← Pinned Python dependencies
```

4. The `toctree` Directive

The `.. toctree::` directive in `index.rst` defines the site's navigation hierarchy:

```
.. toctree::
   :maxdepth: 2
   :caption: Project Overview

   overview
   architecture
   getting_started
```

Each entry maps to a `.rst` or `.md` file. Sphinx resolves them recursively.

5. Build the Documentation

```
cd docs-sphinx
make html          # output → _build/html/index.html
make latexpdf      # output → _build/latex/*.pdf
make clean         # remove previous build artefacts
```

6. Key reStructuredText Syntax

```
Section Heading
===============

Sub-heading
-----------

A paragraph with **bold**, *italic*, and ``inline code``.

.. code-block:: java

   public class Hello {
       public static void main(String[] args) {
           System.out.println("Hello from Sphinx!");
       }
   }

.. note::
   This is an admonition — a highlighted information box.

.. warning::
   This is a warning box.
```

7. Markdown with MyST

With the `myst-parser` extension enabled in `conf.py`, you can write pages in Markdown:

````
# My Page

A paragraph with **bold** and *italic*.

```java
public class Hello { }
```

```{note}
This is a MyST admonition.
```

```{toctree}
:maxdepth: 2
overview
architecture
```
````

How to Integrate Sphinx with GitHub Actions

The workflow below builds the Sphinx HTML and publishes it to the `gh-pages` branch using the `peaceiris/actions-gh-pages` action. It preserves other content already on `gh-pages` (e.g., Maven Site) via `keep_files: true`.

```
# .github/workflows/sphinx-docs.yml
name: Build and Deploy Sphinx Docs

on:
  push:
    branches: [main]
  workflow_dispatch:       # allow manual trigger

jobs:
  sphinx:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Python 3.12
        uses: actions/setup-python@v5
        with:
          python-version: '3.12'

      - name: Cache pip packages
        uses: actions/cache@v4
        with:
          path: ~/.cache/pip
          key: ${{ runner.os }}-pip-${{ hashFiles('docs-sphinx/
requirements.txt') }}
          restore-keys: ${{ runner.os }}-pip-

      - name: Install Sphinx & extensions
        run: pip install -r docs-sphinx/requirements.txt

      - name: Build Sphinx HTML
        run: |
          cd docs-sphinx
          make html

      - name: Deploy to GitHub Pages (sphinx/ subfolder)
        uses: peaceiris/actions-gh-pages@v4
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./docs-sphinx/_build/html
          destination_dir: sphinx     # published at /sphinx/ URL path
          keep_files: true            # preserve Maven Site & Doxygen content
```

Required Repository Settings

1. **Settings** → **Actions** → **General** → **Workflow permissions** → *Read and write permissions*
2. **Settings** → **Pages** → Source: `gh-pages` branch, folder: `/`

The Sphinx portal will then be live at:

```
https://<owner>.github.io/<repo>/sphinx/
```

How This Project Uses Sphinx

Documentation Architecture

The Sphinx portal for SpringBootLibrary is a **meta-documentation hub** — it does not duplicate content that already lives in other tools; instead it links out to the generated artefacts:

```
GitHub Pages (gh-pages branch)
├── index.html              ← Landing page redirecting to Sphinx or Maven
site
├── sphinx/                 ← Sphinx portal (this site)
│   ├── index.html
│   ├── overview.html
│   ├── testing.html
│   ├── reports.html        ← Links to JaCoCo, Surefire, PMD reports
│   ├── javadoc_link.html   ← Links to Javadoc in site/apidocs/
│   └── sphinx_101.html     ← This page
├── site/                   ← Maven Site (Javadoc, JaCoCo, PMD, Checkstyle …)
│   ├── apidocs/
│   ├── jacoco/
│   ├── surefire-report.html
│   └── …
└── doxygen/                ← Doxygen cross-referenced HTML
    └── html/
```

Integration Points

External Artefact	Where Sphinx links to it
Javadoc	`javadoc_link.md` → `../site/apidocs/index.html`
JaCoCo	`reports.md` → `../site/jacoco/index.html`
Surefire	`reports.md` → `../site/surefire-report.html`
PMD / Checkstyle	`reports.md` → `../site/pmd.html`
Performance report	`reports.md` → `../site/reports/perf-report.html`
Doxygen	`javadoc_link.md` → `../doxygen/html/index.html`

Adding a New Documentation Page

1. Create `docs-sphinx/source/my_page.md`
2. Add `my_page` to the appropriate `.. toctree::` in `index.rst`
3. Push to `main` — the GitHub Actions workflow rebuilds and redeploys automatically

Adding the `conf.py` Extensions for This Project

```
extensions = [
    'sphinx.ext.autodoc',    # auto-import Python docstrings (optional here)
    'sphinx.ext.viewcode',   # adds [source] links
    'sphinx.ext.napoleon',   # Google/NumPy style docstrings
    'myst_parser',           # Markdown .md files
    'sphinx_rtd_theme',      # Read the Docs theme
]
```

Useful Sphinx Resources

Resource	URL
Official Sphinx docs	https://www.sphinx-doc.org
MyST Markdown parser	https://myst-parser.readthedocs.io
Read the Docs theme	https://sphinx-rtd-theme.readthedocs.io
Furo theme (modern)	https://pradyunsg.me/furo
sphinx-multiversion	https://holzhaus.github.io/sphinx-multiversion
peaceiris/actions-gh-pages	https://github.com/peaceiris/actions-gh-pages