SpringERP — Detailed Software Requirements Specification

Generated by ChatGPT

July 26, 2025

1. Project Overview

SpringERP is a modular, open-source ERP platform inspired by metasfresh, built using Spring Boot (backend) and React/Redux (frontend). This project targets enterprise-scale architecture, business workflows, messaging orchestration, observability, and deep exploration of Spring Boot internals (auto-configuration, bean lifecycle, Actuator extension). Metasfresh is itself Java-based, with REST APIs, React/Redux frontend, PostgreSQL database, messaging via RabbitMQ/ActiveMQ, Swagger/OpenAPI, Hazelcast caching, and reporting via JasperReports :contentReference[oaicite:2]index=2.

2. Functional Modules & Features

2.1 CRM

- Entities: BusinessPartner (customer/vendor), Leads, Contacts, Activity Logs, Task pipelines.
- Workflow: Prospect → Lead → Customer transitions.

2.2 Sales Management

- Quotation, Sales Orders, Shipment Planning, Invoicing, Credit Limits, Incoterms, Sales Analytics.

2.3 Procurement

- Requisition Planning, Purchase Orders, Goods Receipts, Supplier Performance Tracking.

2.4 Inventory & Warehouse

- Product Attributes, Batch/Serial Numbers, Units of Measure, Warehouse Zones, Traceability, Handling Units.

2.5 Manufacturing (Optional)

- Bills of Materials (BOM), Work Orders, MRP Planning, Scheduling, Resource Allocation.

2.6 Financial Accounting

- Ledger Management, Invoice Payment Handling (Debtors/Creditors), Dunning, Multi-Currency, Reporting.

2.7 Logistics & Supply Chain

- Packing, Order Picking, Tour Planning, EDI Shipping Advice (DESADV), Container / Empties Handling.

2.8 Multi-tenancy & Support Features

Supports multi-organization, multi-tenant architecture, multi-language, multi-currency and scalable mass-processing operations :contentReference[oaicite:3]index=3.

3. Technical Architecture

3.1 Tiered Layers

- **WebUI:** React/Redux single-page application served via Nginx or Apache.
- **WebAPI:** Spring Boot services exposing REST JSON APIs, Swagger/OpenAPI endpoints, optional WebSocket.
- **Application Core:** Domain logic modules, JasperReports integration.
- **Infrastructure:** PostgreSQL database, RabbitMQ/ActiveMQ messaging, Elasticsearch/Kibana search, Hazelcast caching cluster.

3.2 Spring Internals Explored

- Custom Spring Boot Starter modules using `spring.factories`.
- Lifecycle instrumentation via `BeanFactoryPostProcessor` and `BeanPostProcessor`.
- Application events and domain-event driven workflows.

4. Data Model

4.1 Core Entities

BusinessPartner, Product, Quote, Order, OrderLine, Shipment, Invoice, Payment, BOM, ProcurementRequest, GoodsReceipt.

Each entity includes audit fields, organization/tenant association, status codes, and versioning.

4.2 Relationships

One-to-many relations (e.g. Order → OrderLines), multi-tenant scoping via shared schema + tenant ID or separate schemas.

5. API Contracts & Workflows

5.1 Sales Workflow

1. POST `/api/sales/quotes` to create a quotation.
2. POST `/api/sales/orders` to convert accepted quotes into orders.
3. POST `/api/sales/shipments` to dispatch deliveries.
4. POST `/api/financial/invoices` to generate invoices, optionally linking payments.

5.2 Procurement Flow

1. Plan via `/api/procurement/requests`
2. Confirm via `/api/procurement/orders`
3. Log goods receipt via `/api/procurement/receipts`
4. Inventory module reacts to receipts asynchronously.

5.3 Domain Event Dispatch

OrderPlacedEvent, GoodsReceivedEvent, InvoiceGeneratedEvent published via messaging; other modules subscribe to update inventory, ledger entries, etc.

6. Observability & Scheduling

- Spring Actuator endpoints: /actuator/health, /metrics, /httptrace, custom admin probes.
- Micrometer metrics forwarding to Prometheus; dashboards visualized via React UI.
- Scheduled tasks for credit limit checks, dunning notices, stock alerts (via `@Scheduled`).

7. Spring Internals Focus

- Auto-configuration: create Spring Boot starters for modules, conditionals, configuration annotation.
- Bean lifecycle inspection: inject custom post-processors to log bean creation order.
- Request pipeline instrumentation: filters, AOP advices for performance tracing.

8. Testing Strategy

- **Unit Testing:** JUnit + Mockito for service and repository layers.
- **Integration Testing:** Spring Boot Test with embedded DB and MockMVC/TestRestTemplate.
- **E2E Testing:** Cypress or Selenium for user flows.
- **Load Testing:** simulate high concurrency and Hazelcast cache cluster behavior.
- **Fault Injection:** test resilience, actuator health endpoints under service failure.

9. Implementation Roadmap & Milestones

1. Core module: authentication (JWT), tenant context, BusinessPartner CRUD.
2. Sales module: implement full quote $\rightarrow$ order $\rightarrow$ shipment $\rightarrow$ invoice flow.
3. Procurement + Inventory: integrate via messaging.
4. Financial module: invoice creation, payments, dunning dashboard.
5. Front-end: React UI, tables, forms, dashboards, report viewer.
6. Monitoring stack: integrate custom actuator endpoints, Prometheus metrics, and React dashboards.
7. Deep Spring internals: inject starters, lifecycle tracing, profiling instrumentation.

10. References

- Metasfresh technology and architecture: REST API, WebReact, messaging, Elasticsearch, Hazelcast, JasperReports :contentReference[oaicite:4]index=4.
- Functional module coverage and ERP scope from metasfresh documentation :contentReference[oaicite:5]index=5.