Architektur-, Muster- und Integrationsdokumentation

Die folgende Dokumentation beschreibt die zentralen Architekturprinzipien, Entwurfsmuster sowie externen Integrationen der Anwendung „Auftragscockpit“

---

1. Softwarearchitektur & Schichtenaufbau

Die Anwendung folgt einer klassischen Spring-Boot-Schichtarchitektur aus Controller-, Service- und Repository-Schicht.

REST-Endpunkte – etwa im ArticleController – exponieren alle relevanten Geschäftsoperationen.

DTOs schirmen Persistenzdetails konsequent ab und sorgen dafür, dass Controller rein transportorientiert arbeiten.

Die Trennung zwischen DTOs und Entitäten wird über mehrere dedizierte Factory-Klassen umgesetzt (z. B. OrderDtoFactory, DefaultArticleDtoFactory).
Diese Factories erzeugen sowohl Transportobjekte als auch persistierbare Domänenobjekte und laden dabei benötigte abhängige Entitäten (z. B. Benutzer) aus Repositories.

Eine zentrale FactoryConfig registriert alle Factory-Implementierungen als Spring Beans.

Die Repository-Schicht ist über Spring Data JPA realisiert und folgt damit dem Repository Pattern (z. B. OrderRepository).

Spezifische Fachlogik, etwa Margenberechnung oder Lagerprüfungen, ist in dedizierten Services wie ArticleService, PurchaseOrderService oder UserService gekapselt.

---

2. Entwurfsmuster (Design Patterns)

2.1 Factory Pattern

Factories erzeugen DTOs und Entities strukturiert und ohne redundante Konstruktionslogik.
Beispiele: DefaultOrderDtoFactory, DefaultUserDtoFactory, DefaultSupplierDtoFactory.

2.2 Null-Object Pattern

Null-Objekte (NullOrder, NullOrderItem, NullUser, NullArticle, NullSupplier, NullPurchaseOrder, NullPurchaseOrderItem) verhindern NullPointer-Probleme und liefern stabile Defaultwerte.

2.3 Chain of Responsibility

Validierungen laufen über kaskadierte Validierungs-Handler wie:

- StockLevelValidationHandler
- MinimumQuantityValidationHandler
- UniqueArticleValidationHandler

Die Verkettung wird über OrderValidationConfig definiert.

2.4 Strategy Pattern

Die Preisberechnung basiert auf austauschbaren Strategien (OrderPricingStrategy).
Ein Resolver wählt über Prioritäten (z. B. @Order) automatisch die beste Strategie aus.
Beispiele:

- BulkOrderDiscountStrategy
- StandardOrderPricingStrategy

2.5 Decorator Pattern

LoggingOrderServiceDecorator erweitert den OrderService um Logging, ohne dessen Logik zu verändern.
Über @Primary wird er automatisch statt der Originalimplementierung verwendet.

2.6 Template-Method Pattern

Komplexe Abläufe werden durch Template-Method strukturiert, z. B.:

- Bestell-Workflows (OrderWorkflowTemplate, OrderDeliveryWorkflow)
- PDF-Erstellung (AbstractPdfExporter, ArticlePdfExporter)

2.7 Observer / Event Pattern

Ereignisse wie das Ausliefern einer Bestellung werden über OrderEventPublisher an Listener verteilt, z. B.:

- LoggingOrderEventListener
- InventoryAdjustingOrderEventListener

Dieses Muster entkoppelt Kernlogik und Seiteneffekte (z. B. Lageranpassung).

2.8 Dependency Injection (IoC)

Alle Abhängigkeiten (Repositories, Strategien, Workflows etc.) werden im OrderServiceImpl via Konstruktor injiziert.

2.9 Mock-Object / Test-Double Pattern (Unit Tests)

OrderServiceImplTest verwendet Mockito zur Simulation abhängiger Komponenten (Stubs, Mocks), um isoliert testen zu können.

2.10 Service Layer Pattern

OrderService kapselt alle Geschäftsoperationen und trennt die Domänelogik strikt von Controllern und Repositories.

2.11 Singleton Pattern

OrderNumberGenerator erzwingt eine einzelne Instanz zur konsistenten Nummernvergabe.

2.12 Builder Pattern

DTOs wie OrderItemDto und OrderDto verwenden Builder zur sicheren und fluenteren Konstruktion unveränderlicher Objekte.

2.13 Repository Pattern

Über Spring Data JPA werden Repositories wie OrderRepository, UserRepository, ArticleRepository bereitgestellt.

2.14 Record / Value Object Pattern

Records wie OrderWorkflowTemplateDependencies dienen als unveränderliche Datencontainer.

2.15 Strategy-Orchestration / Metadata Pattern

StrategyOrderExtractor nutzt Annotationen, um Strategien reflektiv auszuwählen und zu sortieren.

2.16 Layered Architecture Pattern

Die Paketstruktur folgt der klassischen Layered Architecture:
controller, service, repository, validation, workflow, factory, strategy.

2.17 Adapter Pattern (DTO als Adapter)

DTOs wie OrderDto, UserDto und ArticleDto fungieren als Adapter zwischen internen Domain-Entities und externen API-Modellen.

2.18 Mapper Pattern (Data Mapper)

UserMapper übersetzt zwischen User-Entity und UserDto, ohne Geschäftslogik zu vermischen.

2.19 Template Hook Pattern

Workflow-Methoden (validate(), mutate(), afterSave()) dienen als Hooks zur Erweiterung, ohne die Hauptlogik zu überschreiben.

2.20 Composite-ähnliche Struktur (Bestellung → Positionen)

Bestellungen und Beschaffungsaufträge bestehen aus kompositorisch verschachtelten Items — ein leichtgewichtiges Composite-Muster.

2.21 : Façade Pattern

OrderServiceImpl fungiert explizit als Fassade, da es:

- Validierungen
- Repositories
- Strategien
- Workflows
- Events
- Logging (via Decorator)

orchestriert.

2.22: Transaction Script Pattern (teilweise)

Methoden wie createOrder() und deliverOrder() enthalten sequentielle Geschäftslogikschritte.
Dies ist eine Mischform aus Fassade + Transaction Script.

2.23: Domain Separation Pattern

Durch separate Services & Factories für Kunden- und Beschaffungsaufträge wird ein Domain-Driven-Design-ähnliches Muster sichtbar.

2.24: Data Transfer Aggregation Pattern

OrderDto fasst komplexe Strukturen (Order + Items + User) aggregiert zusammen.

---

3. Geschäftslogik & Workflows

OrderServiceImpl fungiert als zentrale Fassade der Geschäftslogik.
Der Service:

- orchestriert Repositories
- wählt Preisstrategien

- führt Validierungsketten aus
- delegiert Workflow-Schritte
- publiziert Events
- sichert atomare Operationen über @Transactional

Der Auslieferungsworkflow sorgt für Statuswechsel, Event-Publishing und verhindert doppelte Auslieferungen.
StockExceededException sichert Konsistenzregeln zu Lagerbeständen.

Der OrderNumberGenerator stellt eine konsistente Vergabe von Auftragsnummern sicher.

---

4. PDF-Export & Reporting

Die PDF-Generierung nutzt OpenPDF und basiert auf Template Method:

- AbstractPdfExporter definiert Struktur und Ressourcenmanagement
- ArticlePdfExporter implementiert die eigentlichen Tabelleninhalte
- PdfExportService lädt Daten und initiiert den Export

Der Endpunkt /api/articles/export/pdf stellt die Datei zum Download bereit.

---

5. REST-API & Funktionsumfang

Der ArticleController bietet u. a.:

- Lagerbestandsprüfung (checkStockLevel)
- Margenberechnung (getArticleMargin)
- Bestandsänderungen via PATCH (updateInventory)
- PDF-Export (exportPdf)

Alle Controller erlauben Cross-Origin-Zugriffe (@CrossOrigin) für SPA-Frontends.

---

6. Frontend-Integrationen (YouTube, Google Maps)

Die statische Oberfläche (aboutus.html) integriert externe Systeme über <iframe>:

- YouTube-Video
- Google Maps Standortansicht

Zusätzlich sorgen:

- erklärende Texte,
- Barrierefreiheits-Labels und
- ein modernes Grid-Layout

für eine klare UI-Struktur.

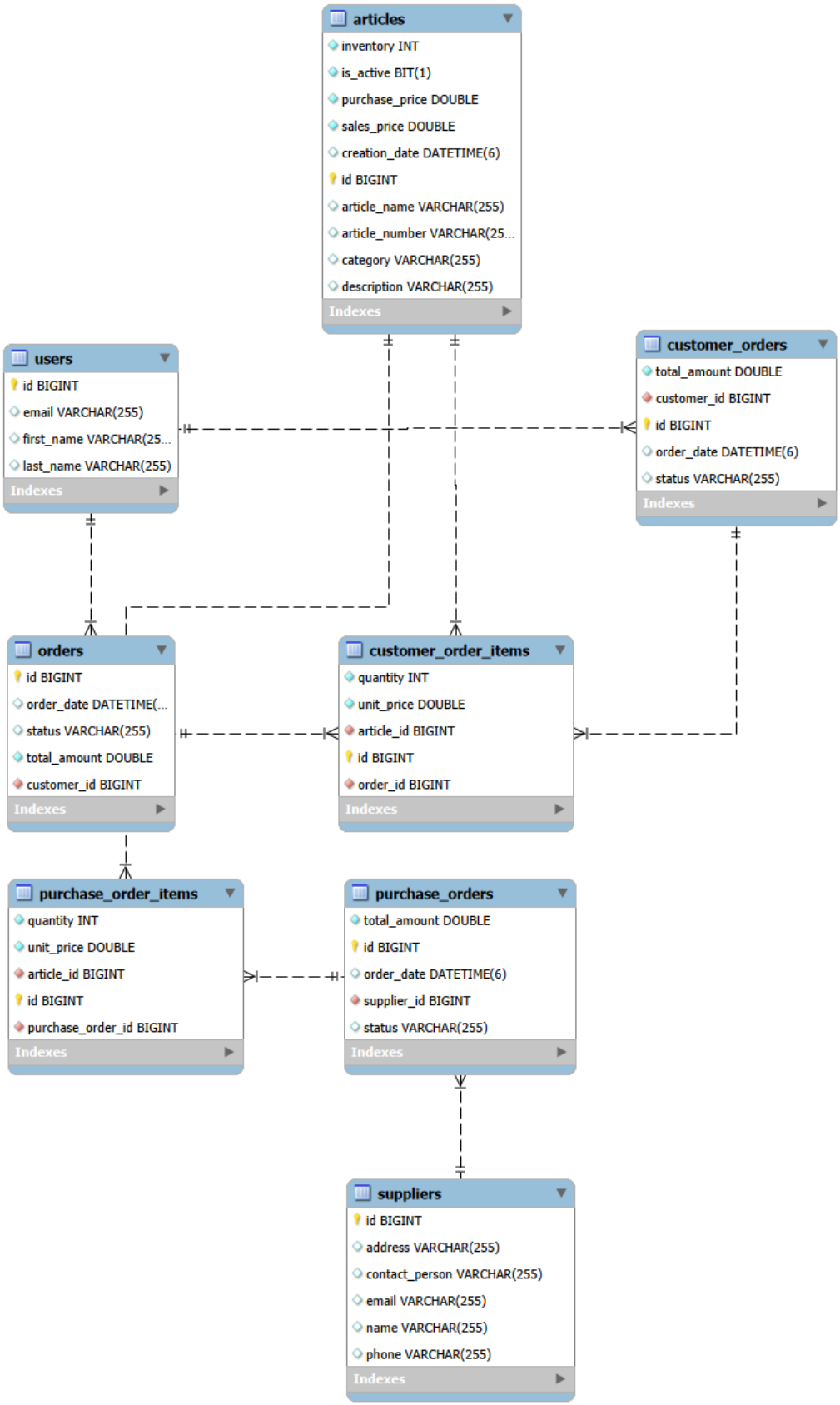
---

7. Zusatztechniken & Architekturprinzipien

- Records stärken Immutability.
- Validierungen bewusst vor Persistenz.
- Kopierte Listenerliste (Immutability-Konzept).
- Reflektive Strategiewahl.
- Konsistente Paketstruktur.
- strikte Single Responsibility pro Modul (z. B. OrderNumberGenerator, Factories).
- API-Design folgt REST-Prinzipien: klare Ressourcen, HTTP-Statuscodes, Content-Type-Steuerung.
- saubere Trennung zwischen Query- und Command-Endpunkten (z. B. Bestand prüfen vs. Bestand ändern).
- Verwendung von Optional in Repositories verhindert NullPointer-Szenarien im Service.

8. Entity-Relationship-Diagramm

Dieses ER-Diagramm zeigt das Datenfundament für die Verwaltung von Artikeln, Kunden, Lieferanten sowie den gesamten Prozess von der Bestellung beim Lieferanten (purchase_orders) bis zum Verkauf an den Kunden (orders).



9. Installation & Setup

Repository klonen:

```
git clone https://github.com/SMABFWS124A/Auftragsmanagement
```

Mit einer IDE der Wahl (IntelliJ, Eclipse, VS Code) öffnen,

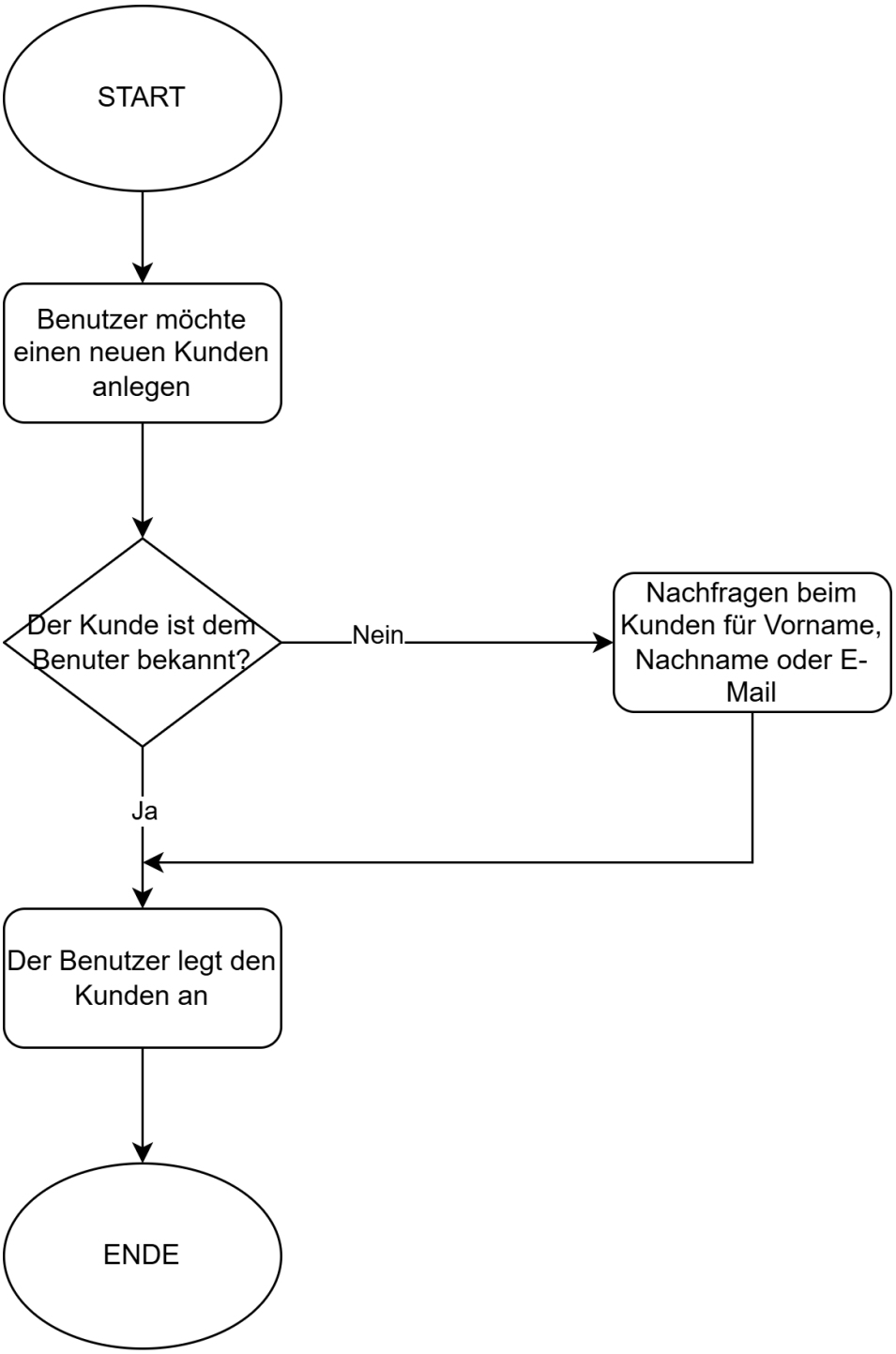
Maven Dependencies laden, Spring Boot Anwendung starten,

Abschließend den Browser (bspw. Chrome) öffnen und die Schnittstellen bzw. HTML-Oberfläche aufrufen über: http://localhost:8080/login.html

Anmeldedaten:

Rolle	E-Mail	Passwort	Hinweis
Testbenutzer	**test@test.de**	**Test123**	**Vollständiger Zugriff auf alle Verwaltungsbereiche für Testzwecke.**

10. Bsp.: Flussdiagramm zum Prozess: Kunden anlegen



11. Use Cases

Die Use Cases lassen sich hier finden:

Auftragsmanagement/src/main/resources/static/Use Cases - Auftragscockpit.pdf at main · SMABFWS124A/Auftragsmanagement

Bzw. unter diesem Pfad im GitHub Repository:

src/main/resources/static/Use Cases - Auftragscockpit.pdf

12. Aufteilung der Arbeiten im Projekt

Die gesamte Projektarbeit, umfassend die Programmierung der Software, die schriftliche Ausarbeitung sowie die Erstellung der Use Cases und weiterer Dokumentationen, wurde von **Nikolas Pering** und **Arda Arif Ermaya** gleichberechtigt im Verhältnis **50:50** erbracht.

Unterschriften:

Gez. Nikolas Pering

Gez. Arda Arif Ermaya

Bergisch Gladbach, 01.12.2025