Desktop-Fakturierungsanwendung

Schlanke, lokal betriebene Fakturierung für Kleinstunternehmen

Agenda

1. **Einleitung** — Team, Kontext & Ziel, Architektur
2. **Das Programm**
    - Gruppe C — Kundenverwaltung
    - Gruppe B — Produktverwaltung
    - Gruppe A — Dokumentenzyklus
    - Gruppe D — Programmoberfläche
3. **Das Entwicklungsprojekt**
    - Ergebnisse der Modultestpläne
    - Wo wurde KI eingesetzt – und wie gut?
    - Was würden wir nächstes Mal anders machen?

Team & Aufgabenverteilung

12 Personen, vier fachliche Komponenten:

Gruppe	Komponente	Leitung	Mitglieder
A	Prozess / Dokumentenzyklus	Lucas Strubel	Luca Kaiser
B	Produktverwaltung	Meron Berhane	Jan-Micah SchulzeAmeling · Jessica Volz
C	Kundenverwaltung	Mahsuna Ahadyar	Kübra Kilic · Mara Weidmann
D	Programmoberfläche	Mirkan Güngör	Moritz König · Mohammed Bouhki

Querschnitt (`gemeinsam`): EreignisBus, JSON-Persistenz, CSV-Hilfe – von allen Gruppen genutzt.

Kontext & Ziel

Problem: SaaS-Fakturierung kostet laufend Lizenzgebühren; Open-Source-Alternativen (z. B. *Fakturama*) sind im Betrieb anspruchsvoll.

Unsere Lösung: schlanke Desktop-Anwendung mit voller Datensouveränität – ohne Cloud.

Ziel	Inhalt
PZ-01	Digitale Verwaltung von Kunden- & Produktstammdaten (CRUD)
PZ-02	Vollständiger Dokumentenzyklus: Angebot → AB → Lieferschein → Rechnung
PZ-03	Bedienbar ohne Vorkenntnisse (Rechnung in < 10 Min)
PZ-04	100 % lokale, DSGVO-konforme Datenhaltung

Nichtziele: Mehrbenutzer/Netzwerk, vollständige Buchhaltung, E-Rechnung (ZUGFeRD/XRechnung).

Architektur im Überblick

Vier Fachmodule + gemeinsame Infrastruktur, manuelle Dependency Injection in `Main.java` (kein Framework).

- **Repository-Interfaces** je Domäne mit JSON-Implementierungen; atomare Schreibvorgänge (`JsonPersistenz.schreibeAtomar`)
- **EreignisBus** (Observer): Services melden Datenänderungen, GUI-Panels aktualisieren sich selbst
- **Snapshot-Prinzip** in Belegen + **Löschsperren** (referenzielle Integrität)

Das Programm

Vier Komponenten

Gruppe C — Kundenverwaltung

Ziel *(Lasten-/Pflichtenheft)*

Kundenstammdaten anlegen, ändern, suchen, löschen *(BA-01–04)* · eindeutige Kundennummer · **keine Löschung bei verknüpften Dokumenten** *(GR-04)*

Technische Umsetzung

- **Referenzielle Integrität** ohne Datenbank: Löschsperre über eine Referenzprüfung
- Automatische Nummernvergabe & Pflichtfeld-/E-Mail-Validierung

  `KundenRepository` · `EinfacherKundennummernGenerator` · `KundenReferenzPruefung`

Live-Demo

Kunde anlegen → `K-000017` · suchen / sortieren · ändern · **Löschsperre** bei verknüpftem Kunden (Hinweis mit Anzahl)

Gruppe B — Produktverwaltung

Ziel *(Lasten-/Pflichtenheft)*

Produkte anlegen, ändern, suchen, löschen *(BA-05–08)* · Nettopreis + Steuersatz · **nur gültige Eingaben**, Löschsperre

Technische Umsetzung

- **Validierung** lehnt negative Preise & unzulässige Steuersätze ab; Produktnummer unveränderlich
- Gleiches Repository-/Generator-Muster wie Gruppe C

  `ProduktRepository` · `EinfacherProduktnummernGenerator` · `ProduktReferenzPruefung`

Live-Demo

Produkt anlegen → `P-000042` · neg. Preis & Steuersatz `0.15` **abgelehnt** · ändern · suchen · Löschsperre

Gruppe A — Prozess / Dokumentenzyklus

Ziel *(Lasten-/Pflichtenheft)*

Angebot → Auftragsbestätigung → Lieferschein → Rechnung *(BA-09–12)* · lückenlose Belegnummern *(GR-01)* · Steuer & Zahlungsziel · § 14 UStG · Storno *(BA-14)*

Technische Umsetzung

- **Snapshot-Prinzip**: alte Rechnungen bleiben preisstabil, auch wenn sich Produktpreise ändern *(GR-03)*
- **Lückenlose** Belegnummern (GoBD) · Folgebeleg übernimmt Daten + Rückreferenz *(GR-05)*

  `DokumentRepository` · `BelegnummernGenerator` · `PdfBoxPdfExporter`

Live-Demo

Angebot erstellen → Folgebeleg zu Rechnung → **PDF-Export**

Gruppe D — Programmoberfläche

Ziel *(Lasten-/Pflichtenheft)*

Geführte Rechnungserstellung *(BA-13)* · Navigation, Pflichtfeldhinweise *(Q-09)*, Statusanzeige · bedienbar **ohne Vorkenntnisse** *(PZ-03)*

Technische Umsetzung

- **EreignisBus (Observer)**: Panels aktualisieren sich automatisch nach Datenänderung
- Fachlogik strikt getrennt — GUI ruft nur die Services der Komponenten A–C

  Swing + FlatLaf · `HauptFenster` · `RechnungsWizardController`

Live-Demo

Wizard: Kunde → Positionen → Bestätigen → Zusammenfassung → Speichern · Statusfilter „offen" · Pflichtfeldhinweis · PDF-/CSV-Export

Das Entwicklungsprojekt

Modultests · KI-Einsatz · Lessons Learned

Ergebnisse der Modultestpläne

71 / 71 Testfälle bestanden — 100 %, 0 Fehler

Komponente	Tests
A — Dokumentenzyklus	18
B — Produktverwaltung	14
C — Kundenverwaltung	14
D — Programmoberfläche	15
Infrastruktur · Performance	6 · 4

Performance	gemessen	Schranke
Start	0,039 s	≤ 5 s
Suche	0,025 s	≤ 1 s
PDF	0,011 s	≤ 2 s
Export	0,181 s	≤ 30 s

Wo wurde KI eingesetzt – und wie gut?

Eingesetzt für

- **Dokumentation:** Entwürfe für Lasten-/Pflichtenhefte, Modultestplan, Traceability-Matrix
- **Code:** Boilerplate der Repository-/Service-Schicht, JSON-Persistenz, CSV-Export, JUnit-Testgerüste, Verwendung von Design-Patters, Optimierungen
- **Diagramme & Tooling:** PlantUML-Quellen, Maven-Konfiguration, pandoc-/PDF-Workflow

Bewertung

- 👍 Hohes Tempo bei Produktion, Tests & Doku; konsistente deutsche Fachsprache; schnelle Iteration nach Feedback
- 👎 Vorschläge teils zu generisch

Was würden wir nächstes Mal anders machen?

- **Früher integrieren:** modulübergreifende Tests nicht erst spät im V-Modell
- **Schnittstellen-Verträge** zwischen Gruppen A–D verbindlich zu Projektbeginn festlegen
- **Kleinere, häufigere Commits** + einheitliche Anforderungen an commits
- **KI-Output gezielter prüfen** statt übernehmen – besonders bei Fachlogik
- Aufgaben-/Lastverteilung über die Git-History früh sichtbar machen

Zusammenfassung

- Vollständiger **Dokumentenzyklus** + **Stammdatenverwaltung** lauffähig umgesetzt
- **Lokale, DSGVO-konforme** Einzelplatzanwendung (Java 21, Swing/FlatLaf)
- **71 grüne Modultests** mit vollständiger **Traceability** zu den Anforderungen
- Alle Performance-Anforderungen erfüllt

Vielen Dank!