Projektdokumentation zur betrieblichen Projektarbeit

Projekt: WueKaStL – Kassen- und Buchungssystem

Prüfungsbewerber: Jonas Maier, Ausbildungsberuf: Fachinformatiker/in Anwendungsentwicklung

Ausbildungsbetrieb: RZ Uni Würzburg, Würzburg

Abgabetermin: keiner

Version: 1.0 (Stand: 27.08.2025)

Hinweis: Beim PDF-Export (z. B. Pandoc/LaTeX) bitte Inhaltsverzeichnis automatisch generieren lassen.

Inhaltsverzeichnis (Hinweis)

---

1 Einleitung

1.1 Projektumfeld

Im Ausbildungsbetrieb RZ Uni Würzburg wird ein internes Kassen- und Buchungssystem benötigt, um den Verkauf von Artikeln (z. B. Getränke/Snacks) an Mitarbeitende effizient zu erfassen. Das System wird von Administratoren verwaltet und von Mitarbeitenden genutzt. Die Lösung ist als klassische Java-Webanwendung im Intranet betrieben und benötigt keine externe Internet-Anbindung.

1.2 Projektziel

Ziel ist die Bereitstellung von WueKaStL – einer benutzerfreundlichen Webanwendung zur Verwaltung von Benutzern, Artikeln und Buchungen. Kernfunktion ist die Gruppenbuchung (gleichzeitiges Auf-/Abbuchen bei mehreren Nutzern), ergänzt um Einzelbuchungen, Preisverwaltung, Statuswechsel und Einsicht in Buchungsverläufe.

1.3 Projektbegründung

- Effizienz: Reduktion manueller Listen und Fehlerquellen; zentrale Historie je Nutzer.
- Transparenz: Nachvollziehbare Buchungen (Wer/Was/Wann/Summe/Kommentar).
- Wartbarkeit: Klare MVC-Struktur, einfache Erweiterbarkeit (neue Artikel, Regeln).
- Kosten: Nutzung vorhandener Infrastruktur (Tomcat/MariaDB) und Open-Source-Komponenten (jQuery, LESS).

1.4 Projektschnittstellen

- Technisch: HTTP(S) zwischen Browser (JSP/JS) und Servlets. Datenbankzugriff via JNDI-DataSource `jdbc/AzubiProjekt_JM-1` (vgl. `webapp/META-INF/context.xml`).
- Benutzer: Administratoren (Vollzugriff), Mitarbeitende (Kassenoberfläche, persönlicher Verlauf), IT-Betrieb (Deployment/Backup/Monitoring).

1.5 Projektabgrenzung

- Kein Kassenhardware-Anschluss (Bondrucker/Scanner) im Projektumfang.
- Keine Mandantenfähigkeit oder externe Schnittstellen (ERP/HR) in v1.
- Mobile Optimierung nur in Grundzügen (Responsive teilweise).

2 Projektplanung

2.1 Projektphasen

- Analyse/Anforderungserhebung
- Entwurf (Architektur, Datenmodell, UI-Mockups)
- Implementierung (UI, JavaScript, Servlets/Domäne)

- Tests (Unit/Integration/UI-Checklisten)
- Abnahme, Einführung, Dokumentation

Beispielzeitplan (Detail siehe Anhang A.1): Analyse 9 h, Entwurf 19 h, Implementierung 29 h, Abnahme 1 h, Einführung 1 h, Doku 9 h, Puffer 2 h.

2.2 Abweichungen vom Projektantrag

- Feature „Gruppenbuchung“ wurde priorisiert und ergonomisch überarbeitet (verbesserte Mehrfachauswahl, valide Parameterübergabe), geringe Verschiebung zulasten optionaler Exportfunktionen.

Zusätzliche Details:

- Plan-/Ist-Abgleich: Meilenstein M2 (UI-Prototyp) +2 Arbeitstage, durch Puffer (2 h) und Umschichtung aus Export-Themen abgefangen.
- Auswirkungen: Höhere Usability (weniger Fehlbedienungen), stabilere Parametrisierung, reduzierte Nacharbeit bei Buchungskorrekturen.
- Risiken: Export-Wunsch bleibt offen; transparent kommuniziert und als Roadmap-Item priorisiert.

2.3 Ressourcenplanung

- Hardware: Dev-PC, Testserver (Tomcat), DB-Server (MariaDB)
- Software: JDK, Apache Tomcat, MariaDB, IDE (z. B. Eclipse), Git, Browser
- Dritte: jQuery, LESS

2.4 Entwicklungsprozess

Iteratives Vorgehen (Kanban-Board): kurze Zyklen, frühe UI-Prototypen, kontinuierliche Abstimmung mit Admin-Usern. Quellcodeverwaltung über Git, Branch-basiertes Arbeiten, Code-Reviews im Team.

3 Analysephase

3.1 Ist-Analyse

Vor dem Projekt wurden Verkäufe teils manuell erfasst. Es fehlte eine zentrale, revisionsfähige Historie pro Nutzer. Buchungen für Gruppen (Events) wurden mit Aufwand und hoher Fehleranfälligkeit durchgeführt.

3.2 Wirtschaftlichkeitsanalyse

3.2.1 Make-or-Buy

- Buy (Fertigkasse): Überdimensioniert für interne Nutzung, Lizenz-/Wartungskosten, Integrationsaufwand.
- Make (Eigenentwicklung): Geringe Anforderungen, vorhandene Java/Tomcat-Kompetenz, volle Kontrolle über Prozesse und Daten.

Entscheidung: Eigenentwicklung.

3.2.2 Projektkosten (Beispielkalkulation)

- Personalkosten: 70 h Entwicklung × [interner Stundensatz] €/h
- Ressourcenkosten (Server/DB/Tools): 70 h × [kalk. Ressourcensatz] €/h
- Schulung/Einführung: h × [Satz] €/h

Transparente Aufstellung im Projektcontrolling (Detail ggf. intern).

3.2.3 Amortisationsdauer (Beispiel)

Bei Zeiteinsparung 10 min/Tag pro 25 Nutzer und 220 AT/Jahr entstehen ~917 h/Jahr. Multipliziert mit gemitteltem Kostensatz ergibt sich eine voraussichtliche Amortisation im Bereich weniger Monate. Exakte Werte siehe interne Kalkulation.

Zusätzliches Berechnungsmodell (vereinfachend):

- Annahmen: durchschnittliche Buchungszeit alt 30 s, neu 15 s; 200 Buchungen/Tag.
- Zeitersparnis/Tag: (30−15) s × 200 = 3000 s ≈ 0,83 h.
- Jahresersparnis: 0,83 h × 220 AT ≈ 183 h.
- Monetär: 183 h × [Kostensatz] €/h → Vergleich mit Projektkosten ergibt ROI < 1 Jahr (bei üblichem Satz).

Sensitivität: Schon bei halbierter Nutzung (100 Buchungen/Tag) bleibt der ROI < 2 Jahre.

3.3 Nutzwertanalyse

Bewertung alternativer Ansätze (Framework/API, UI-Technologie). Entscheidung zugunsten klassischer Servlet/JSP-Anwendung mit jQuery wegen einfacher Serverintegration, geringer Lernkurve und vorhandener Infrastruktur.

Bewertungskriterien (Auszug):

- Betrieb im bestehenden Tomcat (Gewicht 30%) – Sehr gut bei Servlet/JSP, mittel bei SPA-Frameworks.
- Entwicklungsaufwand/Kompetenz (25%) – Hoch vorhandene Kompetenz für Servlet/JSP.
- Erweiterbarkeit (20%) – Beide gut; Servlet/JSP für interne Zwecke ausreichend.
- Performance (15%) – Beide gut; geringer Overhead bei klassischer Server-Side-Render.
- Wartbarkeit (10%) – Gut, klare MVC-Trennung.

3.4 Anwendungsfälle (Auszug)

- UC-01 Benutzer anlegen/ändern/deaktivieren (nur Nicht-Admins)
- UC-02 Gruppenbuchung: Auf-/Abbuchen für mehrere ausgewählte Nutzer
- UC-03 Einzelbuchung für einen Nutzer
- UC-04 Buchungsverlauf ansehen, Strich-Buchungen bearbeiten, Buchungen löschen
- UC-05 Artikelverwaltung: anlegen, Preis ändern (Historie), aktiv/inaktiv

3.5 Qualitätsanforderungen (nicht-funktional)

- Usability: Konsistente UI, klare Labels, Bestätigungsdialoge
- Sicherheit: Session-Auth, Autorisierung im Servlet (`user.isAdmin()`), keine sensiblen Daten in GET
- Performance: < 250 ms UI-Reaktionen, < 1 s Serverantworten für Standardfälle
- Wartbarkeit: MVC-Trennung, Namenskonventionen, Kommentare

3.6 Lastenheft/Fachkonzept (Auszug)

- Muss: Benutzer-/Artikelverwaltung, Einzel-/Gruppenbuchungen, Verlauf, Preisänderungen
- Soll: Editieren/Löschen ausgewählter Buchungen, Historienansichten
- Wunsch: Exporte (CSV/PDF), erweiterte Filter, Benachrichtigungen bei niedrigem Guthaben

4 Entwurfsphase

4.1 Zielplattform

- Server: Apache Tomcat (Servlet 3.0), MariaDB
- Sprache/Technik: Java (Servlets), JSP, jQuery, CSS/LESS
- Deployment: WAR/WebApp im Container, JNDI-DataSource

4.2 Architekturdesign (MVC)

- View: JSP/HTML/CSS/LESS (`webapp/*.jsp`, `webapp/css/*`)
- Controller: Servlets (`/Login`, `/Pin`, `/Logout`, `/Buchungs`, `/Benutzer`, `/Itemservlet`)
- Model/Domäne: Java-Klassen (User, Item, Buchung, Preisverlauf)
- Persistenz: über Domänenklassen/DAO auf DB; DataSource: `jdbc/AzubiProjekt_JM-1`

Request-Flow (vereinfacht):

1. Nutzerinteraktion in JSP (Buttons mit `name=method`) bzw. JS erzeugt Hidden Inputs (`userIds`, `itemW`, `test`).
2. HTTP-POST → Servlet (z. B. `/Buchungs`) → Parameterprüfung/Autorisierung.
3. Geschäftslogik greift auf Domäne/DAO zu → Transaktionen (Buchung, Preislookup via ItempreisID `test`).
4. Redirect/Forward auf nächste View; Flash-Meldungen über Query/Session.

Querschnittlich: Charset-Filter (`de.uniwue.apps.jm1.servlets.CharsetFilter`) für konsistente UTF-8-Verarbeitung.

4.3 Entwurf der Benutzeroberfläche

- Admin-Bereich: `admin_user.jsp`, `admin_item.jsp`
- Kassenoberfläche: `index.jsp`, `benutzer.jsp`, `verlauf.jsp`, `profil.jsp`, `login.jsp`
- Interaktionen: jQuery-gestützt, native Form-Submits für klare Semantik (Button `name=method`)

4.4 Datenmodell (vereinfacht)

- User(uid, vorname, nachname, aktiv, admin, guthaben)
- Item(uid, name, iconUrl, aktiv), Preisverlauf(itempreisID, itemUid, preisCent, gültigAb)
- Buchung(uid, datum, summeCent, kommentar, typ, userUid, [Itemliste])

Constraints & Indizes (empfohlen):

- Primärschlüssel auf allen UID/ID-Feldern, Fremdschlüssel: `Buchung.userUid → User.uid`, `Preisverlauf.itemUid → Item.uid`.
- Indizes: `Preisverlauf(itemUid, gültigAb DESC)` für Preisermittlung; `Buchung(userUid, datum DESC)` für Verlaufsabfragen.
- Konsistenz: `preisCent >= 0`, `guthaben` numerisch; `aktiv`/`admin` als BOOL/TINYINT.

4.5 Geschäftslogik (Kernausschnitte)

- Gruppenbuchung: Wenn kein Einzel-`USER_UID` gesetzt, werden `userIds[]` iteriert und Betrag/Kommentar je Nutzer gebucht.
- Striche: Warenkorb (`itemW` als `itemUid:anzahl`) → Summe aus aktuellem Preis, Buchungstyp STRICHE.
- Verlauf/Editing: Löschen und Bearbeiten (nur STRICHE) mit passenden Redirects.

4.6 Maßnahmen zur Qualitätssicherung

- Code-Reviews, Pairing in kritischen Flows (Buchungen)
- UI-Checklisten (Formvalidierungen, Fokus, Fehlermeldungen)

- Tests (Unit-Logik, Integrationspfade der Servlets)

4.7 Pflichtenheft/Datenverarbeitungskonzept (Auszug)

- Datenflüsse zwischen UI ↔ Servlets ↔ Domäne ↔ DB dokumentiert
- Rollen und Berechtigungen (Admin/Nutzer)

5 Implementierungsphase

5.1 Implementierung der Datenstrukturen

- Einrichtung JNDI-DataSource `jdbc/AzubiProjekt_JM-1` (vgl. `webapp/META-INF/context.xml`)
- Tabellen für User, Items, Preise, Buchungen (Schema gemäß Datenmodell)

5.2 Implementierung der Benutzeroberfläche

- JSP-Seiten: `admin_user.jsp`, `admin_item.jsp`, `index.jsp`, `verlauf.jsp`, `profil.jsp`, `login.jsp`, `benutzer.jsp`, `buchung_bearbeiten.jsp`
- Styles: `webapp/css/*.css`, LESS-Quellen `webapp/css/*.less`
- UI-Skripte: `webapp/js/admin.js`, `script.js`, `bearbeiten.js`, `login.js`

5.3 Implementierung der Geschäftslogik

- Servlets (siehe `webapp/WEB-INF/web.xml`):
  - `/Buchungs` → Buchungslogik (Einzel/Gruppe/Striche/Edit/Delete)
  - `/Benutzer` → Benutzerverwaltung (Anlegen/Ändern/Status/Pin)
  - `/Itemservlet` → Artikelverwaltung (Anlegen/Preis/Status)
  - `/Login`, `/Pin`, `/Logout`
- Parameter (Auszug): `method`, `betrag`, `kommentar`, `userIds[]`, `itemW[]`, `test` (ItempreisID), `buchungID`
- JavaScript-Aspekte (Gruppenbuchung):
  - Event-Binding auf `#gruppenbuchung-section button[name="method"]`
  - Vor Submit: vorhandene Hidden-Inputs reinigen, `userIds` je ausgewählter Karte dynamisch hinzufügen
  - Nativer Form-Submit erhält Button-`method` zuverlässig bei

Parametermapping und Validierung (Beispiele):

- `betrag`: Normalisierung Komma/Punkt, Bereichsprüfung (> 0 für Aufbuchung, ≤ 0 nur bei Abbuchung/Striche mit Begründung).
- `itemW[]`: Format `itemUid:anzahl`, serverseitige Split- und Typprüfung; Lookup der aktuellen `preisCent` über `test` (ItempreisID).
- `userIds[]`: Muss bei Gruppenbuchung gesetzt sein; bei Einzelbuchung `USER_UID` Vorrang.
- Autorisierung: Admin-Pflicht für Admin-Aktionen; Session-Check vor jeder Änderung.

Fehlerbehandlung/Redirects:

- Ungültige Parameter → Fehlermeldung und Redirect auf Ursprung (mit Statuscode/Message).
- Erfolgsfälle → Redirect auf Verlauf/Detail; idempotente Aktionen vermeiden Doppelbuchungen (Post-Redirect-Get).

6 Abnahmephase

- Positiver Durchlauf aller definierten Use-Cases

- Gruppenbuchung getestet (0/1/N Nutzer, `,`/`.` bei Betrag, Kommentar optional)
- Rechteprüfung: Admin zwingend für Admin-Aktionen

Abnahmekriterien (Auszug):

- UC-02: Mehrfachauswahl persistiert, alle `userIds` werden verbucht, Summen korrekt.
- UC-05: Preisänderung erzeugt neuen Preisverlaufseintrag, historischer Preis bleibt erhalten.
- Sicherheit: Nicht-Admin kann keine Admin-Aktionen durchführen (403/Redirect).

Testdaten/Protokoll: definierte Testnutzer/Artikel, protokollierte Soll/Ist-Ergebnisse mit Screenshots.

7 Einführungsphase

- Deployment in Tomcat (WebApp/WAR), Konfiguration DataSource
- Initiale Artikel/Benutzer importiert/angelegt
- Kurze Einweisung der Admin-Nutzer

Betriebskonzept (Kurz):

- Backup: tägliche DB-Sicherung (voll), wöchentliche Aufbewahrung; Wiederherstellung getestet.
- Monitoring: Tomcat-Logs, DB-Health (Connection Pool), Fehleralarme.
- Rollen: Admin (Fachbereich) pflegt Stammdaten; IT betreibt System/Backups.
- Konfiguration: JNDI-Resource serverseitig, keine Secrets im VCS.

8 Dokumentation

- Diese Projektdokumentation (~15 Seiten bei üblichem PDF-Export)
- Benutzer-/Installationshinweise integriert (Kurzform)
- Entwicklerdokumentation als Anhang A.8

Zusätzliche Hinweise:

- Admin-Kurzanleitung (Preis ändern, Benutzerstatus, Gruppenbuchung) als Teil der Einweisung.
- Installationsschritte: Tomcat deployen, `context.xml` einbinden, DataSource prüfen, Basistabellen/Seed-Daten.
- Versionsführung/Changelog: Änderungen an UI/Servlets dokumentieren (Release-Notizen intern).

9 Fazit

9.1 Soll-/Ist-Vergleich

- Ziele erreicht: Benutzer- und Artikelverwaltung, Gruppen-/Einzelbuchung, Verlauf, Preisverwaltung
- Abweichungen: CSV-/PDF-Exporte verschoben; Fokus auf UI-Ergonomie Gruppenbuchung

9.2 Lessons Learned

- Saubere Parameternamen (`userIds`, `method`) vermeiden Fehler.
- Zentrale Event-Bindings verhindern doppelte Handler/Seiteneffekte.
- Native Form-Submits sind in JSP/Servlet-Apps robust und gut wartbar.

9.3 Ausblick

- Exporte (CSV/PDF), erweiterte Filter
- Benachrichtigungen (z. B. niedriges Guthaben)
- Responsive Design weiter verbessern

Eidesstattliche Erklärung

Ich, Jonas Maier, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

„WueKaStL – Kassen- und Buchungssystem“

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Würzburg, den [TT.MM.JJJJ]

[Unterschrift]

A Anhang

A.1 Detaillierte Zeitplanung (Beispiel)

Phase	Aufgabe	Aufwand
Analyse	Ist-Analyse, Stakeholder, Use-Cases	9 h
Entwurf	Architektur, Datenmodell, UI-Entwürfe, Pflichten	19 h
Implementierung	Datenbank, UI (JSP/CSS), JS, Servlets/Domäne	29 h
Abnahme	Tests/Protokolle	1 h
Einführung	Deployment/Einweisung	1 h
Dokumentation	Projekt-/Benutzer-/Entwicklerdoku	9 h
Puffer	Unvorhergesehenes	2 h

A.2 Lastenheft (Auszug)

Muss, Soll, Wunsch wie in Kapitel 3.6 beschrieben; Abnahmekriterien je Use-Case siehe UC-Liste.

A.3 Use-Case-Diagramm (vereinfacht)

flowchart LR Admin((Admin)) User((Benutzer)) UC1[Benutzer verwalten] UC2[Artikel verwalten] UC3[Einzelbuchung] UC4[Gruppenbuchung] UC5[Buchungsverlauf / Edit / Delete] Admin --- UC1 & UC2 & UC4 & UC5 User --- UC3 & UC5

A.4 Pflichtenheft (Auszug)

- Rollenmodell, Rechteprüfung in kritischen Pfaden
- Fehlerbehandlung und Nutzerführung (Bestätigungsdialoge, Validierungen)
- Performance-Ziele und Logging-Anforderungen

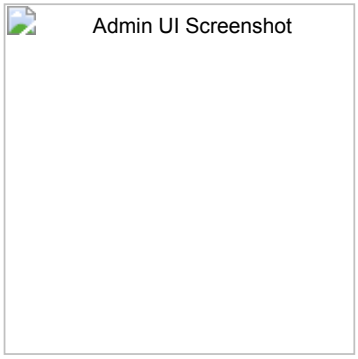
A.5 Datenbankmodell (vereinfacht)

- Tabellen: USER, ITEM, ITEM_PREIS, BUCHUNG, BUCHUNG_ITEM
- Beziehungen: USER 1..* BUCHUNG, ITEM 1..* ITEM_PREIS, BUCHUNG 0..* BUCHUNG_ITEM

A.6 Oberflächenentwürfe

- Siehe Datei `../WueKaStL.drawio` (Mockups/Diagramme)

A.7 Screenshots der Anwendung



A.8 Entwicklerdokumentation

Diese Entwicklerdokumentation beschreibt Aufbau, Konfiguration, Build/Deployment, Hauptkomponenten und Erweiterbarkeit der Anwendung.

A.8.1 Projektstruktur

- `webapp/` – Webressourcen (JSP, CSS/LESS, JS, Icons)
- `webapp/WEB-INF/web.xml` – Servlet-Konfiguration (Servlet 3.0)
- `webapp/META-INF/context.xml` – JNDI-DataSource `jdbc/AzubiProjekt_JM-1`
- `resources/db.properties` – JNDI-Kontext/Zuweisung
- `webapp/js/` – JavaScript
  - `admin.js` – Admin-UI inkl. Gruppenbuchung
  - `script.js` – Kassen/Warenkorb-Interaktionen
  - `bearbeiten.js` – Bearbeiten-Flows (z. B. Striche)
  - `login.js` – Login/UI-Details
- `webapp/css/` – CSS & LESS (z. B. `admin.less`, `style.less`)

A.8.2 Laufzeitkomponenten (Servlets)

Aus `webapp/WEB-INF/web.xml`:

- `de.uniwue.apps.jm1.servlets.Login` → `/Login`
- `de.uniwue.apps.jm1.servlets.Pin` → `/Pin`
- `de.uniwue.apps.jm1.servlets.Logout` → `/Logout`
- `de.uniwue.apps.jm1.servlets.Buchungs` → `/Buchungs`

- `de.uniwue.apps.jm1.servlets.Benutzer` → `/Benutzer`
- `de.uniwue.apps.jm1.servlets.Itemservlet` → `/Itemservlet`

Wesentliche Parameter (je nach Methode): `method`, `betrag`, `kommentar`, `userIds[]`, `itemW[]`, `test` (ItempreisID), `buchungID`, `userUid`.

A.8.3 Datenbankzugriff

- JNDI-Name: `jdbc/AzubiProjekt_JM-1`
- Tomcat Resource (vgl. `webapp/META-INF/context.xml`): MariaDB-URL, Benutzer, Pool-Settings
- Best Practice: Zugangsdaten nicht in VCS im Klartext – produktive Credentials über Server-Konfiguration/Secrets injizieren

A.8.4 JavaScript – zentrale Flows

- Gruppenbuchung (`webapp/js/admin.js`):
  - Click-Delegation auf `#gruppenbuchung-section button[name="method"]`
  - Vor Submit: `#inputs` leeren, pro `.card.selected` Hidden-Input `name=userIds` anlegen
  - Nativer Submit nutzt `name=method` des gedrückten Buttons (`aufbuchen`/`abbuchen`)
- Warenkorb (`webapp/js/script.js`):
  - Hidden Inputs: `itemW` als `itemUid:anzahl`, `test` als `itempreisID`
- Deaktivieren-Dialog: Kontextsensitiv Benutzer/Artikel
- Formatter/Validatoren: Betrag/Komma-Punkt-Konvertierung, PIN-Numeric

A.8.5 JSP-Konventionen

- UTF-8-Encoding über `<jsp-config>` gesetzt
- Eindeutige IDs in Formularen (z. B. keine doppelten `id="new-status"`)
- Formulare nutzen POST; Buttons tragen `name=method`

A.8.6 Build & Deployment

- Build: Kompilierte Klassen unter `webapp/WEB-INF/classes/` (Projekt abhängig von IDE/Buildsystem)
- Deployment: WebApp/WAR auf Tomcat; `context.xml` in `META-INF/` bereitstellen oder serverseitig konfigurieren
- DB-Schema: vorab anlegen/migrieren; Testdaten optional einspielen

A.8.7 Tests

- Unit: Betragsumrechnung, Summenbildung für Striche
- Integration: Servlet-Pfadparameter (Gruppen/Einzel), Redirects, Rechte
- UI: Manuelle Checklisten (Formvalidierungen, Alerts, Fokus)

A.8.8 Erweiterbarkeit/Hotspots

- Neue Buchungstypen: Switch/Strategy im Servlet/Service
- Exporte: neue Servlet-Aktion + View (CSV/PDF)
- Suche/Filter: client- oder serverseitige Filter in Listen

A.8.9 Bekannte Punkte

- Preisformat vereinheitlichen (Anzeige/Parsing)

- Responsive Verhalten verbessern

Hinweis zum Seitenumfang: Bei Export nach PDF (A4, 11–12 pt, Zeilenabstand 1.15–1.5, übliche Ränder) erreicht dieses Dokument ca. 15 Seiten zzgl. Anhang. Diagramme/Screenshots in ausreichender Größe einbinden. Für Pandoc/LaTeX-Export z. B.:

```
pandoc Projektdokumentation_IHK.md -o Projektdokumentation.pdf --from gfm --pdf-engine=xelatex --toc
```