Institut für Informatik
Datenbanken und Informationssysteme
Prof. Dr. Stefan Conrad
Sergej Korlakov
Nina Liebrand

Wintersemester 25/26
Abschlussprojekt
Abgabe bis 26.01.2026 8:00 Uhr

Datenbanken: Weiterführende Konzepte

- Die Aufgaben sind selbstständig und eigenständig zu bearbeiten. Die Nutzung von ChatGPT oder anderen generativen KI-Modellen ist erlaubt, erfolgt jedoch auf eigene Verantwortung. Inhalte, die mithilfe von KI erstellt wurden, müssen fachlich korrekt sein; daraus resultierende Fehler werden wie reguläre Fehler bewertet.
- Das Projekt muss auf einem Unix-System lauffähig sein. Kann das Projekt nicht ausgeführt werden, wird die Abgabe mit 0 Punkten bewertet.
- Komprimieren Sie Ihre Abgabe als **ZIP-Datei (.zip)**. Laden Sie diese ZIP-Datei auf ILIAS hoch.
- Teilen Sie sich die Bearbeitungszeit gut ein, um bei möglichen Problemen genügend Zeit zu haben. Bei Fragen nutzen Sie bitte die Fragestunden am 13.01.2026 und 15.01.2026 oder das ILIAS-Forum.
- Die Korrektur wird zum Teil automatisiert ablaufen. Es ist folglich zwingend erforderlich, sich genauestens an die Anforderungen zu halten!

Aufgabe 1 *Tutorial I: Installation von Java, Maven und Docker* (1 Punkte)

Für die Bearbeitung erhalten Sie eine vorgegebene Struktur des Projekts. Für das Projekt werden **Java 17**, **Maven** und **Docker** benötigt. **Bei den folgenden Tutorials setzen wir voraus, dass Sie ein Debian-basiertes System haben wie Debian, Ubuntu, Kubuntu, Xubuntu oder Manjaro.**

(a) Installieren Sie Java 17 wie hier angegeben. Für Ubuntu installieren Sie die passende Java-Version mit:

```
$ sudo apt-get install openjdk-17-jre openjdk-17-jdk
```

Überprüfen Sie Ihre Java Installation mit:

```
$ java -version
```

Mögliches Problem: Java-Version ist nicht 17

- Sie haben wahrscheinlich mehrere Java-Versionen installiert. Mit dem folgenden Befehl:

  ```
  $ sudo update-alternatives --config java
  ```

  können Sie die passende Java-Version (17) auswählen.

(b) Installieren Sie Maven mit:

```
$ sudo apt install maven
```

(c) Installieren Sie Docker Desktop. Wenn Sie Docker Desktop starten, wird gefragt, ob Sie einen Account haben. Sie benötigen keinen Account. Sie können die Schritte für das Erstellen eines Accounts überspringen.

Aufgabe 2 *Tutorial II: Projekt aufsetzen* (1 Punkte)

(a) Öffnen Sie das Projekt in einer IDE (Wir empfehlen IntelliJ IDEA oder VSCode). Bei VSCode sollten Sie passende Plugins wie *Gradle for Java*, *Language Support for Java(TM) by Red Hat* und *Extension Pack for Java* installieren.

(b) Öffnen Sie eine Console und navigieren Sie zum Ordner des Projekts. Starten Sie die Datenbank mit folgendem Befehl:

```
$ docker compose up -d
```

Nun wird die Postgresql-Datenbank namens *abschlussprojekt* gestartet. **Achtung:** Wenn Sie den Container oder den Rechner nun herunterfahren, dann können alle Änderungen verloren gehen, die Sie direkt in der Datenbank selbst vorgenommen haben.

Falls Sie den Datenbank-Container im persistenten Modus starten wollen (d.h. so, dass die Änderungen in der Datenbank nach dem Herunterfahren nicht verloren gehen) dann führen Sie folgenden Befehl aus:

```
$  docker compose -f docker-compose-persistent.yml up -d
```

Achtung: Prüfen Sie vor der Abgabe, ob alle Tabellen, deren Inhalte, Trigger, Views etc. mit dem Befehl *docker compose up -d* in die Datenbank geladen werden! Hierfür werden im weiteren Verlauf der Projektbeschreibung die Dateien *schema.sql* und *data.sql* eingeführt.
Es werden ausschließlich die Ergebnisse berücksichtigt, die durch *docker compose up -d* erzeugt werden.

(c) Um sich mit der Datenbank zu verbinden, führen Sie folgende Befehle in der Console aus:

```
$ docker exec -it dbw25-26-abschlussprojekt-db-1 bash
psql -U lordoftherows -d abschlussprojekt
```

(d) Wenn die IDE Ihnen anzeigt, dass das Projekt fehlerfrei aufgesetzt wurde, können Sie es über das Terminal mit dem folgenden Befehl starten:

```
$ cd spring-boot/
$ mvn clean install
$ mvn spring-boot:run
```

Sie sollten die Rückmeldung `BUILD SUCCESS` erhalten.

Mögliches Problem:

- Bei dem Fehler "Failed to read artifact descriptor for..." Führen Sie den Befehl erneut durch:

```
$ mvn clean install
```

(e) Rufen Sie die Startseite der API im Webbrowser auf, indem Sie die URL http://localhost:8080 besuchen.

(f) Wenn Sie bei der Bearbeitung der Aufgaben *schema.sql* oder *data.sql* verändern, dann müssen Sie den entsprechenden Datenbank-Container neu starten, damit diese Änderungen in der Datenbank wirksam werden. Hierfür müssen Sie im Projektordner folgende Befehle ausführen:

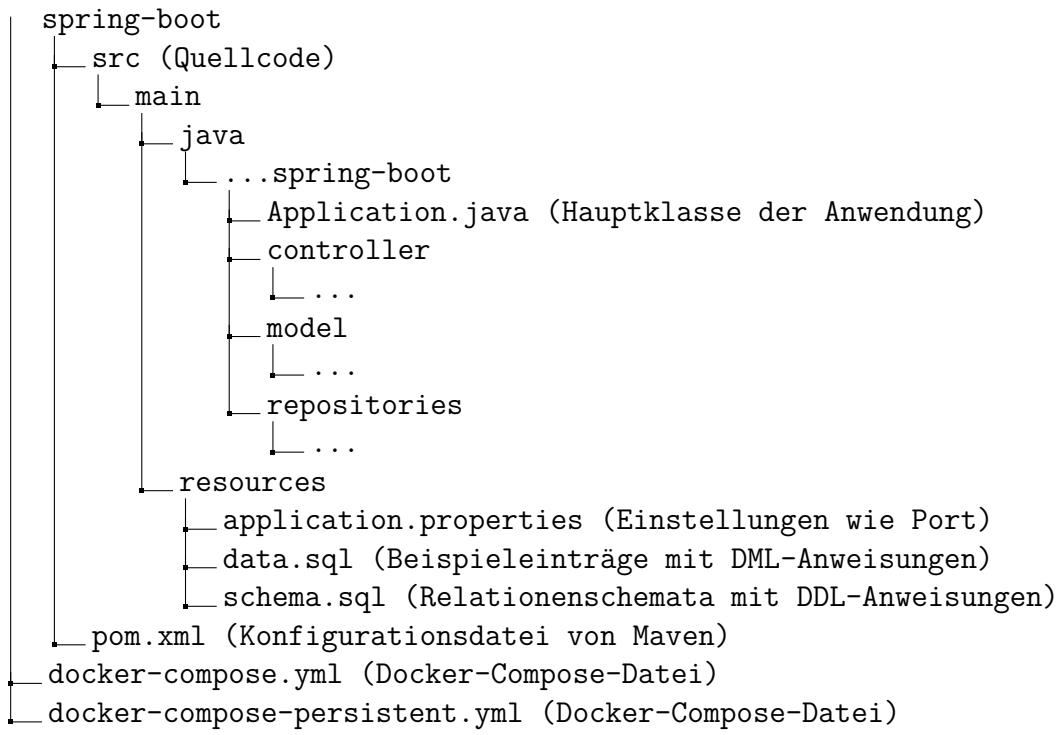
```
$ docker compose down
$ docker compose up -d
```

Mögliches Problem: Nach *docker compose up* sind die Änderungen an *schema.sql* weiterhin nicht übernommen worden. Führen Sie dann den Befehl wie in Aufgabenteil (c) aus, um sich mit der Datenbank zu verbinden. Anschließend können Sie an der Datenbank Änderungen vornehmen. Mit `DROP SCHEMA IF EXISTS public CASCADE;` löschen Sie alle alten Relationen der Datenbank. Anschließend können Sie die veränderten Relationen und Trigger, sowie die Daten neu in die Datenbank laden, indem Sie die sql-Statements der *schema.sql* und *data.sql* ausführen.

Achtung: Auch hier gilt: wenn Sie den Container oder den Rechner herunterfahren, dann können alle Änderungen verloren gehen, die Sie in dem nicht-persistenten Datenbank-Container selbst und nicht über *schema.sql* vorgenommen haben.

Aufgabe 3 *Tutorial III: Projektstruktur* (0 Punkte)

Im Projekt sind bereits wichtige Dateien vorhanden, welche nicht gelöscht werden dürfen! Im Folgenden werden die Funktionalitäten erklärt:



- `pom.xml` ist die zentrale Projekt-Konfigurationsdatei von Maven und definiert Abhängigkeiten, Build-Einstellungen, Plugins, Java-Version sowie Projekt-Metadaten.
- `docker-compose.yml` beschreibt die Datenbank-Infrastruktur der Anwendung und definiert in dem Projekt die Nutzung einer PostgreSQL-Datenbank inklusive Benutzer, Passwort und Datenbankname.
- `docker-compose-persistent.yml`: hiermit wird der Datenbank-Container im persistenten Modus gestartet.
- `spring-boot/src/main/resources/schema.sql` muss von Ihnen gemäß der Aufgabenstellung mit den Anweisungen zur Erstellung der Datenbank befüllt werden und fehlerfrei ausführbar sein.
- `spring-boot/src/main/resources/data.sql` enthält zunächst auskommentierte Statements zum Testen Ihres Projekts, die abhängig vom Projektfortschritt schrittweise wieder einkommentiert werden können.

Für die Umsetzung des Projekts ist zudem unter anderem die Erstellung der folgenden Ordner/-Dateien wichtig:

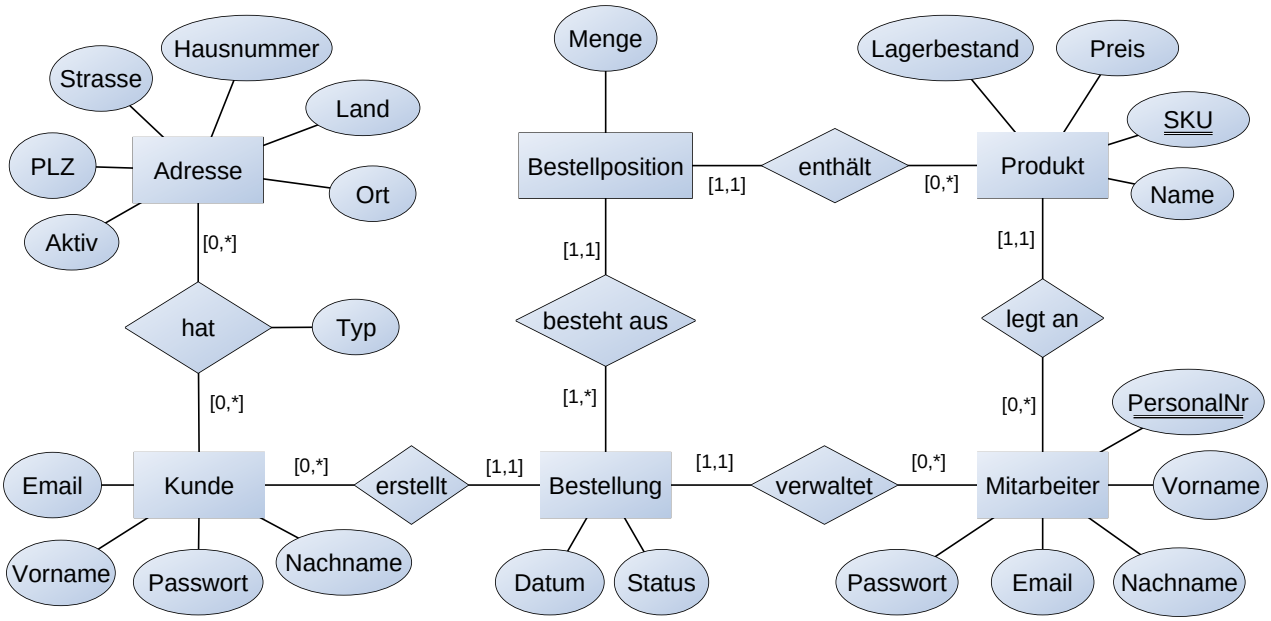
1. model: wird verwendet, um Daten zwischen der Datenbank und der Benutzeroberfläche zu übertragen. In unserer Anwendung repräsentiert das Objekt eine Entität in unserer Datenbank, welches Attribute besitzt.
2. repository: Ist eine Klasse, die verwendet wird, um die Datenbankzugriffe in einer Anwendung zu implementieren.

3. controller: Ein Controller steuert die gesamte Webanwendung, indem er festlegt, welche URLs erreichbar sind und welche Aktionen daraufhin ausgeführt werden sollen.

ACHTUNG: keine der vorgegeben Dateien außer *schema.sql* und *data.sql* darf verändert werden!

Aufgabe 4 *Abschlussprojekt: Datenbankerstellung, Trigger und Checks* (38 Punkte)

Für das Abschlussprojekt sei das folgende ER-Modell zu einem Online-Handel gegeben:



Zu diesem ER-Modell ist auch das dazugehörige Relationenmodell gegeben:

adresse(adresse_id, aktiv, strasse, hausnummer, plz, ort, land)

kunde(kunde_id, email, vorname, nachname, passwort)

kunde_hat_adressen(adresse_id, kunde_id, typ)

mitarbeiter(personal_nr, vorname, nachname, email, passwort)

bestellung(bestellung_id, datum, status, mitarbeiterzuweis, kunde_id)

produkt(sku, name, preis, lagerbestand, angelegt_von)

bestellposition(position_id, menge, sku, bestellung_id)

Das Attribut `bestellung.mitarbeiterzuweis` zeigt dabei auf `mitarbeiter.personal_nr` und `produkt.angelegt_von` zeigt ebenfalls auf `mitarbeiter.personal_nr`.

Erstellen Sie die Datenbank zu dem gegebenen Relationenmodell in postgresql . Beachten Sie dabei die zusätzlichen Anforderungen in den folgenden Aufgabenstellungen. **Schreiben Sie Ihre Lösung in die Datei `schema.sql`**. Beachten Sie zudem:

- Alle Attribute müssen exakt so benannt sein wie im Relationenmodell.
- Passende Datentypen für die Attribute müssen selbst gewählt werden.
- Alle Attribute sollen `not null` sein; es sei denn, in der entsprechenden Teilaufgabe steht etwas anderes.
- Alle ids, sowie `personal_nr` sollen in sql automatisch erstellt werden.
- Für nicht genannte Attribute in den folgenden Anforderungen liegen keine zusätzlichen Bedingungen vor.

- Die Dokumentation von Postgresql zu Regular Expressions, sowie ein Online-Tool zur Überprüfung der Korrektheit von regulären Ausdrücken.
- Wählen Sie selbst die Reihenfolge der Tabellen und beachten Sie dabei Fremdschlüssel.
- Die Kompilierbarkeit Ihrer Datenbank können Sie sicher mit einem Online-Compiler überprüfen. Alternativ können Sie in dem Log von *docker-compose up* darauf achten, ob Ihre Datenbank hochgefahren wird.
- `data.sql` wurde bereits für Sie vorbereitet. Die enthaltenen Einträge sind zunächst auskommentiert und können abhängig vom Projektfortschritt schrittweise wieder einkommentiert werden. Zusätzlich darf die Datei zu eigenen Testzwecken um weitere Einträge ergänzt werden; die ursprünglichen Einträge dürfen jedoch nicht gelöscht werden!

(a) Erstellen Sie die Tabelle zu der Relation `adresse` mit folgenden zusätzlichen Informationen.

- `aktiv` ist ein boolean.
- `strasse` ist maximal 60 Zeichen lang und besteht nur aus Buchstaben des lateinischen Alphabetes (inkl. Umlaute (ÄäÜüÖöß)), wobei nur der erste Buchstabe ein Großbuchstabe ist.
- `hausnummer` besteht nur aus Zahlen, aber kann mit einem einzelnen Kleinbuchstaben abschließen (z.B. 70 oder 12a)
- `plz` wird als Zahl gespeichert und kann bis zu 12 Ziffern lang sein (z.b. 112233445566).

(b) Erstellen Sie die Tabelle zu der Relation `kunde` mit folgenden zusätzlichen Informationen:

- `vorname` und `nachname` sind jeweils maximal 32 Zeichen lang und dürfen nur Buchstaben aus dem lateinischen Alphabet (inkl. Umlaute (ÄäÜüÖöß)) enthalten.
- Für `email` soll Folgendes gelten:
  - Sie ist einzigartig.
  - Sie darf maximal 256 Zeichen lang sein.
  - Sie muss die Zeichen `@` und `.` enthalten. Das `@` muss vor dem `.` stehen.
- `passwort` muss zwischen fünf und 20 Zeichen lang sein und mindestens einen Buchstaben, eine Zahl sowie ein Sonderzeichen enthalten.

(c) Erstellen Sie die Tabelle zu der Relation `kunde_hat_adressen` mit folgenden zusätzlichen Informationen.

- Das Attribut `typ` kann ausschließlich die Werte "Lieferadresse" oder "Rechnungsadresse" annehmen.

(d) Erstellen Sie die Tabelle zu der Relation `bestellung` mit folgenden zusätzlichen Informationen.

- `status` darf auschließlich die Werte "neu", "bezahlt", "versendet", "abgeschlossen" und "storniert" annehmen.
- `status` darf nur auf "storniert" geändert werden, wenn der `status` vorher den Wert "neu" oder "bezahlt" hatte.
- `datum` wird als Datum mit Uhrzeit angegeben.

(e) Erstellen Sie die Tabelle zu der Relation `mitarbeiter` mit folgenden zusätzlichen Informationen.

- `passwort` muss zwischen fünf und 20 Zeichen lang sein und mindestens einen Buchstaben, eine Zahl sowie ein Sonderzeichen enthalten.

(f) Erstellen Sie die Tabelle zu der Relation `produkt` mit folgenden zusätzlichen Informationen.

- `lagerbestand` darf zu keinem Zeitpunkt einen negativen Wert haben.
- `sku` ist ein eindeutiger, interner alphanumerischer Code (Dieser enthält beispielsweise Informationen wie Hersteller, Farbe und Größe).
- `preis` soll maximal 8 Stellen vor dem Komma und zwei Stellen nach dem Komma haben.

(g) Erstellen Sie die Tabelle zu der Relation `bestellposition` mit folgenden zusätzlichen Informationen.

- `menge` muss größer als 0 sein.
- Beim Einfügen (Insert) einer neuen `bestellposition`:
  - Prüfe, ob der `lagerbestand` des zugehörigen Produkts mindestens der `menge` entspricht, die bestellt wird.
  - Falls nicht, soll die Insert Operation mit einer Fehlermeldung abgebrochen werden.
  - Andernfalls soll der `lagerbestand` um die bestellte `menge` reduziert werden.
  - Diese Überprüfung und Änderung des `lagerbestand` eines Produktes sollen nur erfolgen, wenn der `status` der zugehörigen `bestellung` nicht "storniert" ist.
- Wird die `menge` einer bestehenden `bestellposition` geändert (update), soll Folgendes gelten:
  - Falls die Erhöhung der `menge` den `lagerbestand` des Produktes unterschreiten würde, soll das Update mit Ausgabe einer Fehlermeldung verhindert werden.
  - Ansonsten soll der `lagerbestand` entsprechend der neuen `menge` erhöht oder verringert werden.
  - Diese Überprüfung und Änderung des `lagerbestand` eines Produktes sollen nur erfolgen, wenn der `status` der zugehörigen `bestellung` nicht "storniert" ist.
- Wird eine `bestellposition` gelöscht (delete), soll Folgendes gelten:
  - Stelle sicher, dass die zuvor reservierte `menge` wieder zum `lagerbestand` des Produkts hinzugefügt wird.
  - Diese Änderungen des `lagerbestand` eines Produktes soll nur erfolgen, wenn der `status` der zugehörigen `bestellung` nicht "storniert" ist.
- Wird der `status` einer `bestellung` auf "storniert" geändert (update), soll Folgendes gelten:
  - Stelle sicher, dass die `menge` aller Produkte von zugehörigen Bestellpositionen wieder dem `lagerbestand` des Produktes hinzugefügt werden.

Aufgabe 5 *Abschlussprojekt: API mit Datenbankanbindung* (100 Punkte)

Programmieren Sie eine REST-API, die über JDBC mit Ihrer PostgreSQL-Datenbank kommuniziert. `Wichtig`: Achten Sie bei der Implementierung der Endpoints auf den korrekten Einsatz der HTTP-Request-Methoden passend zum CRUD-REST-Prinzip. Endpoints, die dieses Prinzip nicht erfüllen, werden nicht bewertet. Informationen zu Request finden Sie unter anderem hier, sowie ein kleines Tutorial zu Get-Requests. **JPA darf nicht verwendet werden!**

Für die Ausführung von Anfragen können Sie `Swagger-ui` verwenden, die über `http://localhost:8080/swagger-ui/index.html` erreicht werden kann. Beachten Sie, dass `Swagger-ui` keine Anfragen mit voneinander abhängigen oder sich gegenseitig ausschließenden Parametern unterstützt. Hierfür sollten Sie Tools wie Postman oder das Terminal verwenden.

(a) Alle Mitarbeiter sollen unter `http://localhost:8080/mitarbeiter` ausgegeben werden. Die Ausgabe für einen einzelnen Mitarbeiter soll exakt die Attribute des folgenden Beispiels enthalten:

```
{
    "personalNr":1,
    "passwort":"a#123",
    "email":"anna.meier@firma.db",
    "vorname":"Anna",
    "nachname":"Meier"
}
```

- Zudem soll ein einzelner Mitarbeiter anhand der `personal_nr` über den Parameter id ausgegeben werden (z.B. `http://localhost:8080/mitarbeiter?id=1`).
- Ein neuer Mitarbeiter soll ohne die Angabe der `personal_nr` erstellt werden können.
- Ein Mitarbeiter soll anhand der `personal_nr` gelöscht werden können.

(b) Alle Kunden sollen unter `http://localhost:8080/kunden` ausgegeben werden. Kunden sollen mit all Ihren Adressen und dem `typ` der Adressen ausgegeben werden. Die Ausgabe für einen einzelnen Kunden soll exakt die Attribute des folgenden Beispiels enthalten:

```
{
    "kundeId":1,
    "email":"123@test.db",
    "vorname":"Max",
    "nachname":"Muster",
    "passwort":"max123",
    "adressen":[
        {"adresse":{
            "adresseId":1,
            "aktiv":true,
            "strasse":"Hauptstrasse",
            "hausnummer":"12",
            "plz":10115,
            "ort":"Berlin",
            "land":"Deutschland"},
        "typ":"Lieferadresse"},
        {"adresse":{
            "adresseId":2,
            "aktiv":true,
            "strasse":"Bahnhofweg",
            "hausnummer":"7a",
            "plz":80331,
            "ort":"Muenchen",
```

```
            "land":"Deutschland"},
        "typ":"Rechnungsadresse"}
        ]
}
```

- Zudem soll ein einzelner Kunde anhand der `kunde_id` über den Parameter id ausgegeben werden (z.B. `http://localhost:8080/kunden?id=1`). Auch hier sollen die Adressen des Kunden und der `typ` der Adressen ausgegeben werden.
- Ein einzelner Kunde soll anhand der `email` über den Parameter email ausgegeben werden (z.B. `http://localhost:8080/kunden?email=25@dbwistsuper.db`). Auch hier sollen die Adressen des Kunden und der `typ` der Adressen ausgegeben werden.
- Ein neuer Kunde soll erstellt werden können, wobei der Nutzer bei der Erstellung nur die Attribute `email`, `vorname`, `nachname` und `passwort` angeben muss.
- Alle Attribute eines Kunden sollen anhand der `kunde_id` verändert werden können.

(c) Alle Adressen sollen unter `http://localhost:8080/adressen` ausgegeben werden. Die Ausgabe für eine einzelne Adresse soll exakt die Attribute des folgenden Beispiels enthalten:

```
{
    "adresseId":1,
    "aktiv":true,
    "strasse":"Hauptstrasse",
    "hausnummer":"12",
    "plz":10115,
    "ort":"Berlin",
    "land":"Deutschland"
}
```

- Eine einzelne Adresse soll anhand der `adresse_id` über den Parameter id ausgegeben werden (z.B. `http://localhost:8080/adressen?id=1`).
- Ein Nutzer soll eine neue Adresse erstellen können ohne eine `adresse_id` anzugeben.
- Alle Attribute einer Adresse sollen anhand der `adresse_id` verändert werden können.

(d) Alle Produkte sollen unter `http://localhost:8080/produkte` ausgegeben werden. Zu Produkten soll unter `angelegtVon` die PersonalNr angegeben werden, jedoch keine weiteren Informationen zum Mitarbeiter. Die Ausgabe für ein einzelnes Produkt soll exakt die Attribute des folgenden Beispiels enthalten:

```
{
    "sku":"SKU-1002",
    "name":"Notebook Pro",
    "preis":1299.00,
    "lagerbestand":7,
    "angelegtVon":2
}
```

- Ein einzelnes Produkt soll anhand der `sku` über einen Parameter sku ausgegeben werden (z.B. `http://localhost:8080/produkte?sku=1`) .
- Ein Nutzer soll ein neues Produkt unter Angabe aller obenstehender Attribute erstellen können.

- Ein Produkt soll anhand der `sku` gelöscht werden können.
- Der `lagerbestand` eines Produktes sollen anhand der `sku` verändert werden können, andere Attribute können jedoch nicht verändert werden.

(e) Alle Bestellpositionen sollen unter `http://localhost:8080/bestellpositionen` ausgegeben werden. Die Ausgabe für eine einzelne Bestellposition soll exakt die Attribute des folgenden Beispiels enthalten:

```
{
    "positionsId":14,
    "bestellungId":1,
    "produktSku":"SKU-1004",
    "menge":2,
    "gesamtpreis":20.00
}
```

- Eine einzelne Bestellposition soll anhand der `position_id` über den Parameter id ausgegeben werden (z.B. `http://localhost:8080/bestellpositionen?id=14`).
- Eine neue Bestellposition soll erstellt werden können, wobei nur die Attribute `bestellung_id`, `sku` und `menge` bei der Erstellung angegeben werden. Der Gesamtpreis für die Rückgabe soll automatisch anhand der bestellten Menge und des Produktpreises berechnet werden.
- Eine Bestellposition soll anhand der `position_id` gelöscht werden können.

(f) Alle Bestellungen sollen unter `http://localhost:8080/bestellungen` ausgegeben werden. Die Ausgabe für eine einzelne Bestellung soll exakt die Attribute des folgenden Beispiels enthalten:

```
{
  "bestellungId": 10,
  "kundeId": 10,
  "personalnr": 10,
  "datum": "2024-01-19T14:05:00Z",
  "status": "abgeschlossen",
  "positionen": [
     {"positionsId": 10,
      "bestellungId": 10,
      "produkt": {
        "sku": "SKU-1010",
        "name": "Headset Noise Cancel",
        "preis": 229.99,
        "lagerbestand": 5,
        "angelegtVon": 1},
      "menge": 1,
      "gesamtpreis": 229.99
    }
  ]
}
```

- Eine einzelne Bestellung soll anhand der `bestellung_id` über den Parameter id ausgegeben werden (z.B. `http://localhost:8080/bestellungen?id=10`).
- Eine neue Bestellung soll erstellt werden können, wobei nur die Attribute `kunde_id`, `personal_nr`, `datum` und `status` bei der Erstellung angegeben werden.

- Eine Bestellung soll anhand der `bestellung_id` gelöscht werden können.

(g) Unter der URL http://localhost:8080/login/mitarbeiter sollen sich Mitarbeitende per POST-Request mit ihrer Personalnummer und ihrem Passwort anmelden können. Die Anmeldedaten sind dabei im Request-Body und nicht in der URL zu übermitteln. Der Aufbau des Bodys soll exakt dem im folgenden Beispiel gezeigten Format entsprechen:

```
{
  "personalNr": 1,
  "passwort": "a#123"
}
```

Ist der Login erfolgreich, soll der HTTP-Statuscode 200 zurückgegeben werden, zusammen mit den zugehörigen Attributen, wie im folgenden Beispiel dargestellt:

```
{
  "personalNr": 1,
  "passwort":null,
  "email": "anna.meier@firma.db",
  "vorname": "Anna",
  "nachname": "Meier"
}
```

Schlägt der Login fehl, ist der HTTP-Statuscode 401 zurückzugeben. Der Response-Body soll dabei wie folgt aufgebaut sein:

```
{
    "personalNr": null,
    "passwort": null,
    "email": null,
    "vorname": null,
    "nachname": null
}
```

(h) Unter der URL http://localhost:8080/login/kunde sollen sich Kunden per POST-Request mit ihrer E-Mail-Adresse und ihrem Passwort anmelden können. Die Zugangsdaten sind im Request-Body und nicht in der URL zu übermitteln. Der Aufbau des Bodys soll dem im folgenden Beispiel dargestellten Format entsprechen:

```
{
  "email": "123@test.db",
  "passwort": "max12!"
}
```

Ist der Login erfolgreich, soll der HTTP-Statuscode 200 zurückgegeben werden, zusammen mit den zugehörigen Attributen, wie im folgenden Beispiel dargestellt:

```
{
  "kundeId": 1,
  "email": "123@test.db",
  "vorname": "Max",
  "nachname": "Muster",
  "passwort": null,
  "adressen": []
}
```

Schlägt der Login fehl, ist der HTTP-Statuscode 401 zurückzugeben. Der Response-Body soll dabei wie folgt aufgebaut sein:

```
{
    "kundeId": null,
    "email": null,
    "vorname": null,
    "nachname": null,
    "passwort": null,
    "adressen": []
}
```

`Wichtig:` In der Praxis wird aus einem Passwort und einem Salt ein Hashwert gebildet, der bei der Registrierung gespeichert wird. Beim Anmeldevorgang wird das eingegebene Passwort auf die gleiche Weise gehasht und mit dem gespeicherten Hashwert verglichen. Die Teilaufgaben, die das Anlegen der Kunden, Mitarbeiter und die entsprechenden Anmeldevorgänge wurden jedoch zu Ihren Gunsten vereinfacht.

Aufgabe 6 *Abschlussprojekt: Views und API-Erweiterung* (22 Punkte)
Das Controlling der Muttergesellschaft des Shops möchte aufgrund interner Gegebenheiten mehrere Sichten (Views) bereitgestellt bekommen, auf die es über `http://localhost:8080/report/` zugreifen kann. Speichern Sie die Views in schema.sql, sodass beim Hochfahren des Projekts alle Views mit erstellt werden.

(a) Erstellen Sie einen SQL-View mit dem Namen *v_kunde_summe_anzahl_bestellungen*.

Der View soll für jeden Kunden eine Übersicht über seine Bestellungen liefern und folgende Informationen enthalten:

- die Kundennummer (kunde_id),
- die E-Mail-Adresse des Kunden,
- die Anzahl der getätigten Bestellungen,
- die Gesamtsumme aller Bestellungen des Kunden.

Dabei gelten folgende Anforderungen:

- Kunden ohne Bestellungen sollen ebenfalls im View enthalten sein.
- Die Gesamtsumme ergibt sich als Summe von Preis × Menge aller Bestellpositionen.
- Hat ein Kunde noch keine Bestellungen getätigt, soll die Gesamtsumme 0 betragen.
- Der View soll nach der kunde_id aufsteigend sortiert sein.

Im nächsten Schritt soll ein Endpoint erstellt werden, über den der View unter `http://localhost:8080/report/kunde/summe-anzahl-bestellungen` abgerufen werden kann. Die Ausgabe soll exakt die Attribute des Beispiels enthalten:

```
[
    {
        "kundeId": 1,
        "email": "123@test.db",
        "anzahlBestellungen": 2,
        "gesamtsumme": 6997.94
    }
]
```

(b) Erstellen Sie einen SQL-View mit dem Namen `v_produkt_verkaufszahlen`.

Der View soll für jedes Produkt eine Übersicht über dessen Verkaufszahlen liefern und die folgenden Informationen enthalten:

- die Produktnummer (sku),
- den Namen des Produkts,
- die insgesamt verkaufte Menge des Produkts,
- den Umsatz des Produkts,
- die Anzahl der Bestellungen, in denen das Produkt enthalten ist.

Dabei gelten folgende Anforderungen:

- Produkte, die noch nie bestellt wurden, sollen ebenfalls im View enthalten sein.
- Der Umsatz eines Produkts berechnet sich aus der mit dem Produktpreis multiplizierten bestellten Menge, sofern der Bestellstatus weder "neu" noch "storniert" ist.
- Für alle anderen Produkte sollen Verkaufsmenge und Umsatz den Wert 0 annehmen.
- Der View soll nach der insgesamt verkauften Menge absteigend sortiert sein.

Im nächsten Schritt soll ein Endpoint erstellt werden, über den der View unter `http://localhost:8080/report/produkt/verkaufszahlen` abgerufen werden kann. Die Ausgabe soll exakt die Attribute des Beispiels enthalten:

```
[
    {
        "sku": "SKU-1001",
        "name": "Notebook Basic",
        "gesamtVerkaufteMenge": 7,
        "umsatz": 4899.93,
        "anzahlBestellungen": 2
    }
]
```

(c) Erstellen Sie einen SQL-View mit dem Namen `v_mitarbeiter_uebersicht`.

Der View soll für jeden Mitarbeiter eine Übersicht über dessen Aktivitäten im System liefern und die folgenden Informationen enthalten:

- die Personalnummer des Mitarbeiters,
- die Anzahl der verwalteten Bestellungen,
- die Anzahl der angelegten Produkte.

Dabei gelten folgende Anforderungen:

- Alle Mitarbeiter sollen im View enthalten sein, auch wenn sie keine Bestellungen verwaltet oder keine Produkte angelegt haben.
- Der View soll nach der Personalnummer aufsteigend sortiert sein.

Im nächsten Schritt soll ein Endpoint erstellt werden, über den der View unter `http://localhost:8080/report/mitarbeiter/uebersicht` abgerufen werden kann. Die Ausgabe soll exakt die Attribute des Beispiels enthalten:

```
[
    {
        "personalNr": 1,
        "anzahlVerwalteterBestellungen": 2,
        "anzahlAngelegterProdukte": 2
    }
]
```

(d) Erstellen Sie einen SQL-View mit dem Namen `v_mitarbeiter_bestellstatus_uebersicht`.

Der View soll für jeden Mitarbeiter und für jeden Bestellstatus eine Übersicht über die Anzahl der zugewiesenen Bestellungen liefern. Der View muss mindestens die folgenden Informationen enthalten:

- die Personalnummer des Mitarbeiters,
- den Bestellstatus,
- die Anzahl der Bestellungen in diesem Status,

Dabei gelten folgende Anforderungen:

- Alle Mitarbeiter sollen im View enthalten sein, auch wenn ihnen keine Bestellungen zugewiesen sind.
- Für jeden Mitarbeiter soll jeder definierte Bestellstatus berücksichtigt werden, auch wenn zu einem Status keine Bestellungen existieren.
- Die Anzahl der Bestellungen ist jeweils pro Mitarbeiter und Status zu ermitteln.
- Der View soll nach der Personalnummer und dem Bestellstatus aufsteigend sortiert sein.

Im nächsten Schritt soll ein Endpoint erstellt werden, über den der View unter `http://localhost:8080/report/mitarbeiter/bestellstatus-uebersicht` abgerufen werden kann. Die Ausgabe soll exakt die Attribute des Beispiels enthalten:

```
[
    {
        "personalNr": 1,
        "status": "abgeschlossen",
        "anzahlBestellungen": 0
    },
    {
        "personalNr": 1,
        "status": "bezahlt",
        "anzahlBestellungen": 0
    },
    {
        "personalNr": 1,
        "status": "neu",
        "anzahlBestellungen": 2
    },
    {
        "personalNr": 1,
        "status": "storniert",
```

```
        "anzahlBestellungen": 0
    },
    {
        "personalNr": 1,
        "status": "versendet",
        "anzahlBestellungen": 0
    }
]
```