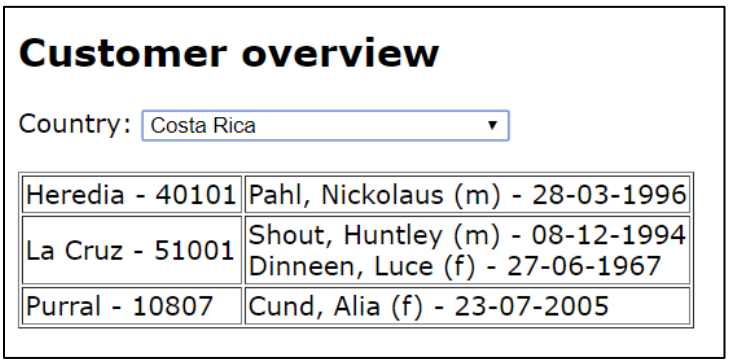
Java-Web-Applikation Kundendatenbank

Erstelle eine webbasierende Java-Applikation, die es ermöglicht eine Kunden-Datenbank zu erstellen, sie mit Daten zu befüllen, sowie Kundendaten zu suchen, zu filtern und anzuzeigen. Die postgres-Datenbank '**customerdb**' ist mit Hilfe von JPA zu erstellen.

1. Programmablauf



Die Seite der Web-Anwendung ist entsprechend der Abbildungen zu gestalten. Über die Select-Box wird vom Benutzer ein Land ausgewählt. Die Select-Box ist beim ersten Request bereits mit allen Ländern aus der Datenbank befüllt.
Nach der Auswahl eines Landes werden alle Städte dieses Landes mit allen ihren Kunden tabellarisch angezeigt.
Bei der Stadt wird neben dem Namen auch die Postleitzahl, wenn sie vorhanden ist, angezeigt. Alle Städte werden in der Tabelle alphabetisch aufsteigend sortiert angezeigt.
Bei den Kunden werden der Nachname, der Vorname, das Geschlecht und das Geburtsdatum angezeigt.
Beim ersten Deployment der Applikation werden die Kundendaten aus einer XML-Datei gelesen und in die Datenbank importiert.

2. Programmbeschreibung

Die Web-Anwendung ist nach dem MVC-Muster in folgende Dateien zu gliedern:
Im Verzeichnis **WebPages**:

- **customerView.html**
- **customerView.js**

Im package **servlets**:

- **CustomerController.java**

Im package **pojos:**

- **DataHolder.java**
- **Country.java**
- **City.java**
- **Customer.java**
- **Gender.java**

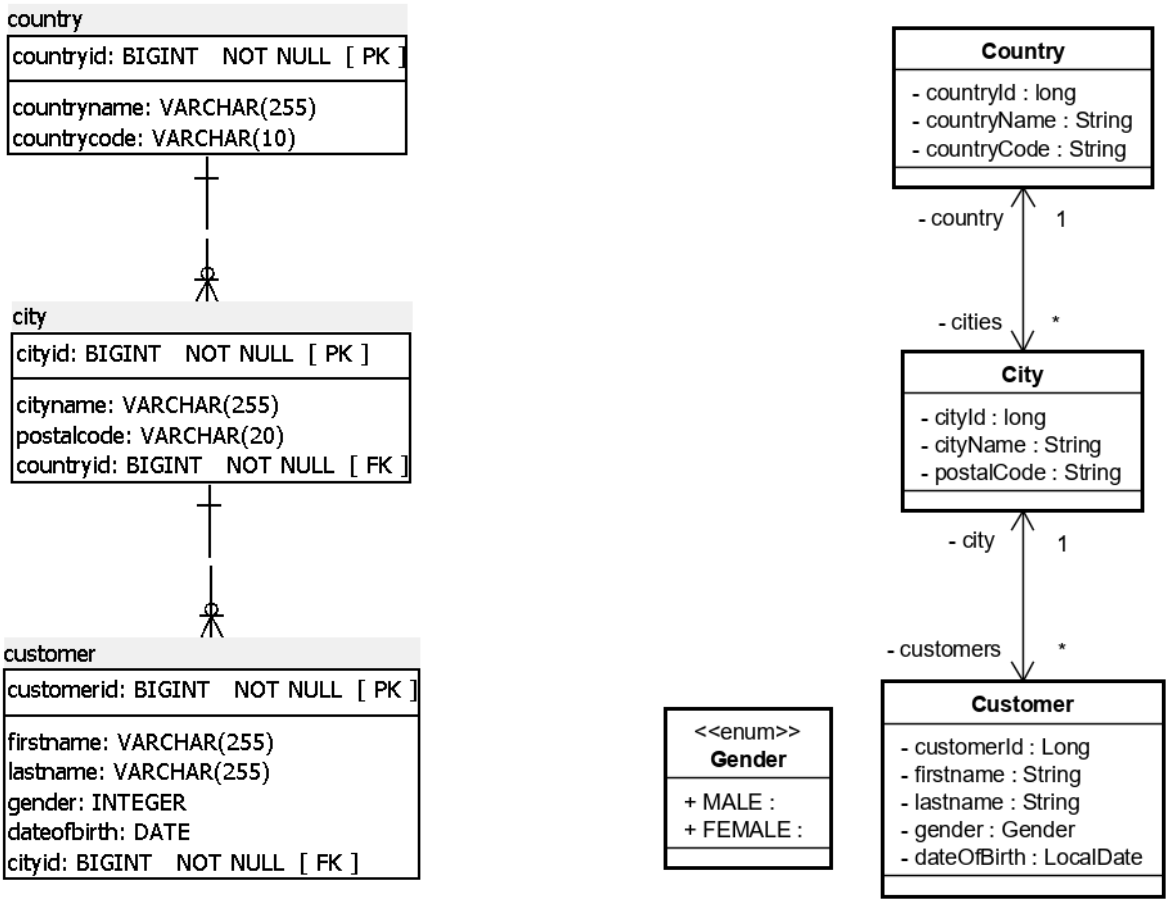
Im package **database:**

- **DB_Access.java**
- **LocalDateConverter.java**

Im package **xml:**

- **XML_Access.java**
- **LocalDateAdapter.java**

Die Java Anbindung an die postgres-Datenbank '**customerdb**' erfolgt mit Hilfe von JPA.
Erstelle das Mapping der Datenbank entsprechend dem KD und dem ERD mit Pojo-Klassen mit Hilfe von Annotationen. Alle Namen, Datentypen und Eigenschaften sind exakt einzuhalten! Die Tabellen werden über die Java-Applikation erzeugt. Der Inhalt der Datenbank wird in der Datei **customer_data.xml** zur Verfügung gestellt.



Die Struktur der XML-Datei enthält unter dem Root-Element eine Liste aller Länder, in jedem Land eine Liste aller Städte und in jeder Stadt eine Liste aller Kunden:

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<dataHolder>
  <Countries>
    <Country>
      <countryId>0</countryId>
      <countryName>Greece</countryName>
      <countryCode>GR</countryCode>
      <Cities>
        <City>
          <cityId>0</cityId>
          <cityName>Skoútari</cityName>
          <Customers>
            <Customer>
              <dateOfBirth>2004-01-23</dateOfBirth>
              <firstname>Kora</firstname>
              <gender>Female</gender>
              <lastname>Crossman</lastname>
            </Customer>
            <Customer>
              ...
            </Customer>
            ...
          </Customers>
          <postalCode></postalCode>
        </City>
        <City>
          ...
        </City>
          ...
      </Cities>
    </Country>
```

Land

Stadt

Kunde

```
    <Country>
       ...
    </Country>
      ...
  </Countries>
<dataHolder>
```

Die Pojo-Klassen sind sowohl um die JPA- als auch um die JAXB-Annotationen zu erweitern. Die Lombok-Library kann verwendet werden.

Die Klasse **pojos.Country.java**

- JPA-Mapping entsprechend dem ERD und dem KD.
- JAXB-Mapping entsprechend der XML-Datei.
- Der Primärschlüssel wird in der Datenbank erzeugt.
- Die Assoziation zur Klasse **City** ist bidirektional.
- Erstelle die **NamedQuery Country.getAll** die die Namen aller Länder, alphabetisch sortiert, aus der Datenbank holt.

Die Klasse **pojos.City.java**

- JPA-Mapping entsprechend dem ERD und dem KD.
- JAXB-Mapping entsprechend der XML-Datei
- Der Primärschlüssel wird in der Datenbank erzeugt.
- Die Assoziationen zu den Klassen **Country** und **Customer** sind bidirektional.
- Erstelle die **NamedQuery City.GetCitiesByCountryName** mit der die Namen aller Städte eines Landes, aufsteigend sortiert, aus der Datenbank geholt werden.

Die Klasse **pojos.Customer.java**

- JPA-Mapping entsprechend dem ERD und dem KD.
- JAXB-Mapping entsprechend der XML-Datei.
- Der Primärschlüssel wird in der Datenbank erzeugt.
- Die Assoziationen zur Klasse **City** ist bidirektional.

Die Klasse **pojos.Gender.java**

- Enumeration mit den Werten **MALE** und **FEMALE**.

Die Klasse **pojos.DataHolder.java**

- Wrapper-Klasse die eine Liste aller Country-Objekte enthält.
- Dient als Einstiegspunkt für das JAXB-Mapping.

Die Klasse **xml.XML_Access.java**

- Die Klasse ist als Singleton zu implementieren.
- **loadLocations()** Methode die alle Daten aus der Datei '**customer_data.xml**' mit JAXB einliest und in Form eines **DataHolder**-Objekts zurückgibt.
- Nach dem Einlesen der Daten müssen alle bidirektionalen Beziehungen zwischen den Klassen **Country** und **City**, sowie den Klassen **City** und **Customer** korrekt aufgebaut werden.

Die Klasse **xml.LocalDateAdapter.java**

- Adpater-Klasse zur Konvertierung des Datums mit JAXB.

Die Klasse **database.LocalDateConverter.java**

- Converter-Klasse zur Konvertierung des Datums für die Datenbank.

Die Klasse **database.DB_Access.java**

- Die Klasse ist als Singleton zu implementieren.
- Der Zugriff auf die Datenbank erfolgt ausschließlich mit den NamedQueries aus den Pojo Klassen.
- Öffnen und Schließen der Verbindung zur Datenbank.
- **persistDataHolder()** persistiert den gesamten Inhalt des **DataHolder**-Objekts, also alle **Country**-, **City**- und **Customer**-Objekte mit ihren Assoziationen in der Datenbank.
- **getAllCountries()**
  Verwendet die NamedQuery um alle Länder aus der DB in einer Liste zurückzugeben.
- **getCitiesOfCountry(String countryName)**
  Methode die alle Städte eines Landes mit einer NamedQuery aus der DB holt und in Form einer **List**-Collection zurückgibt.

Die Klasse **servlets.CustomerController.java**

- Das Servlet wird als Welcome-Page definiert.
- Beim ersten Start des Servlets werden die Daten aus der XML-Datei gelesen und in der Datenbank persistiert.
- Bei einem **get**-Request werden die Namen aller Länder in JSON-Form als Response zurückgeschickt.
- Bei einem **post**-Request werden die Kundendaten als JSON-Objekte im Response zurückgeschickt. Die Städte eines Landes werden aufsteigend sortiert. Gibt es in einer Stadt mehrere Kunden, so werden diese nach dem Nachnamen aufsteigend sortiert. Die Sortierungen sind in Java mit Streaming-Ausdrücken zu implementieren.
- Alle benötigten Daten werden über die Klasse **DB_Access** aus der Datenbank gelesen.

customerView.html

- Die Anzeige ist entsprechend der Abbildung zu erstellen.
- Beim Laden der Seite wird ein Ajax-post-Request an das Controller-Servlet geschickt um alle Ländernamen in der Select-Box anzuzeigen.
- Lege ein Event auf die Select-Box, die die Länder anzeigt, so dass bei jeder Änderung ein Ajax-post-Request an das Controller-Servlet erzeugt wird.
  Mit dem Request wird der Name des selektierten Landes mitgeschickt.

customerView.js

Java-Script-Datei zur Abwicklung des Ajax-fetch-Requests an das Controller-Servlet.

countryRequest()

- Erstelle einen Ajax-fetch-Request an die **doGet()**-Methode des Controller-Servlets um alle Ländernamen zu bekommen. Der Request wird beim Laden der Seite ausgeführt. Über die Callback-Methode werden JSON-Daten empfangen und in die Select-Box eingefügt.

customerRequest()

- Erstelle einen Ajax-fetch-Request an die **doPost()**-Methode des Controller-Servlets um alle Städte und deren Kunden eines Landes zu bekommen. Über die Callback-Methode werden JSON-Daten empfangen und in die Tabelle eingefügt.

Anbindung an die Datenbank

- Erstellen der Persistence-Unit.
- Einbinden aller Libraries.

Die Datei **web.xml**

- Festlegen der Startseite **CustomerController.java**