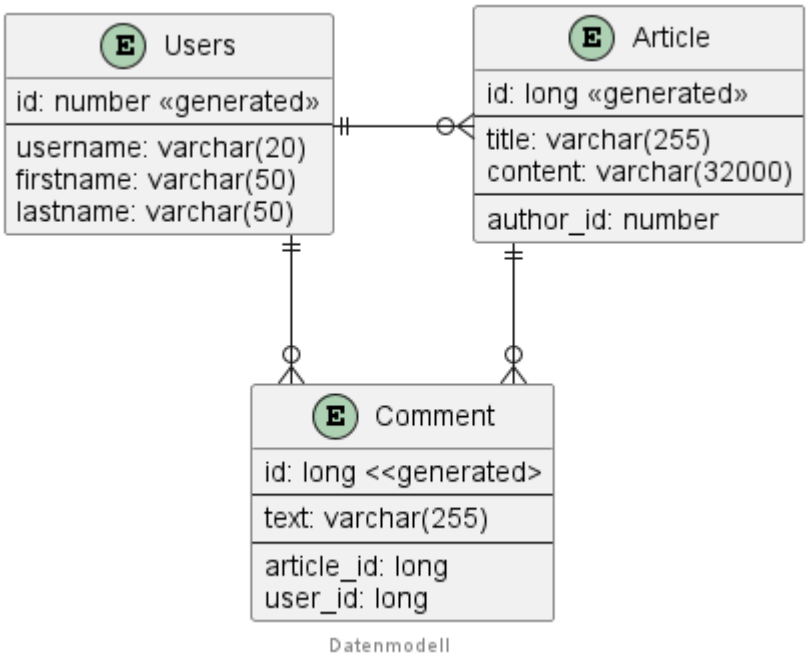
Übung Blog (JPA, REST)

Es soll ein REST-API für ein vereinfachtes BLOG-System erstellt werden.

Das Datenmodell sieht dabei wie folgt aus:



Datenmodell

Entities erstellen

Im Package `at.htl.model` sind bereits POJO-Klassen vordefiniert. Diese sind so zu erweitern, dass das vorher gezeigte Datenmodell entsteht. Primärschlüssel sind automatisch zu generieren (Surrogatschlüssel)!

Die Beziehung zwischen `User` und `Article` ist unidirektional, die zwischen `Article` und `Comment` soll bidirektional abgebildet werden.

Achten Sie auf die korrekten Datentypen bzw. Längenangaben.

Stellen Sie sicher, dass die Spalten wie im ERD ersichtlich benannt werden, da sonst das `import.sql`-Script nicht laden wird.

Die maximale Länge von VARCHAR-Spalten in Derby ist 32.672 Zeichen. Sollten größere Felder benötigt werden muss mit LOBs gearbeitet werden.

Beziehungen werden im Objektmodell nicht mit den ID´s abgebildet, sondern über Referenzen mit entsprechenden Annotationen. Um einen Fremdschlüssel anders zu benennen kann die `@JoinColumn`-Annotation verwendet werden.

REST-API erstellen

Erstellen Sie eine Resource -Klasse mit dem Namen `at.htl.boundary.ArticleResource`. Die Methoden sollen über folgenden Pfad angesprochen werden können: `/api/article`.

Für die Datenbankzugriffe ist eine Klasse mit dem Namen `at.htl.repository.ArticleRepository` anzulegen.

GET /api/article/{id}

Diese GET-Methode muss den Artikel als Json-Objekt zurückliefern. Die ID des Artikels wird in der URL mitübergeben.

Testen Sie die Funktionalität mittels `http`-Request-File.

Test:

```
GET http://localhost:8080/api/article/1

{
  "id": 1,
  "title": "TODO: Add details how to use JPA",
  "content": "First Steps with JPA",
  "user": {
    "userid": 1,
    "username": "maxi",
    "firstname": "Max",
    "lastname": "Muster"
  },
  "commentList": [
    {
      "text": "Forget to mention details about LOB columns",
      "user": {
        "userid": 1,
        "username": "maxi",
        "firstname": "Max",
        "lastname": "Muster"
      },
      "article": {
        "id": 1,
        "title": "TODO: Add details how to use JPA",
        "content": "First Steps with JPA",
        "user": 1,
        "commentList": [
          {
            "text": "Forget to mention details about LOB columns",
            "user": {
              "userid": 1,
              "username": "maxi",
              "firstname": "Max",
```

```
            "lastname": "Muster"
          },
   .....
}
```

Wir sehen hier, dass es zu einer rekursiven Auflösung kommen würde…

Anpassen der Json-Serialisierung

Nicht immer ist es wünschenswert, dass die gesamte Objekt-Hierarchie ins Json-Format serialisiert und zurückgeliefert wird.

Beispielsweise könnte gewünscht sein, dass die User-Informationen nicht mit ausgegeben werden. Natürlich könnte das über eigene DTO-Klassen (**D**ata **T**ransfer **O**bject) umgesetzt werden, was es auch ermöglicht, je nach Anwendungsfall unterschiedliche Datenstrukturen zurückzuliefern.

Soll die Information nie mitgeliefert werden könnte dies via Annotationen gesteuert werden.

JsonIgnore für Jackson

```
@JsonIgnore ①
@ManyToOne
@JoinColumn(name="author_id")
private User user;
```

① `@JsonIgnore` verhindert die Json-Serialisierung des annotierten Attributs. Für das JsonB-Framework müsste stattdessen `@JsonTransient` verwendet werden.

Durch diese Lösung geht die gesamte Benutzerinformation verloren. Als Alternative wäre es möglich, nur die User-ID mitzugeben. Dadurch kann der Client die Userdaten im Bedarfsfall abholen und diese würden nicht wiederholt mit übertragen bei einer Liste von Artikeln.

Jackson: Child-Objekte nur als Id

```
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property=
"userid") ①
@JsonIdentityReference(alwaysAsId = true) ②
@ManyToOne
@JoinColumn(name="author_id")
private User user;
```

① Hier wird konfiguriert welches Attribut anstatt des referenzierten Objekts serialisiert werden soll. In diesem Fall die `userid`.

② `@JsonIdentityReference` regelt, ob immer nur die ID serialisiert werden soll (`true`) oder ob beim allerersten Vorkommen das gesamte Objekt serialisiert werden soll und beim nächsten Vorkommen nur mehr die ID (`false`).

Eine weitere Möglichkeit wäre, die Annotation `@JsonIgnoreProperties({<attliste>})` zu verwenden. Hier kann konfiguriert werden, welche Attribute von dem Kind-Objekt nicht serialisiert werden sollen.

@JsonIgnoreProperties({"userid", "firstname", "lastname"})

Stellen Sie sicher, dass bei den Comments die Artikelinformationen nicht mitgeliefert werden, um eine Rekursion zu vermeiden! Weiters soll bei den Kommentaren nur jeweils der Username des Verfassers mitgeliefert werden und nicht die gesamten Benutzerdaten.

Test:

```
GET http://localhost:8080/api/article/1

{
  "id": 1,
  "title": "TODO: Add details how to use JPA",
  "content": "First Steps with JPA",
  "user": 1,
  "commentList": [
    {
      "text": "Forget to mention details about LOB columns",
      "user": {
        "userid": 1,
        "username": "maxi"
      },
      "article": 1
    },
    {
      "text": "Hope that´s not too complicated...",
      "user": {
        "userid": 2,
        "username": "sunny"
      },
      "article": 1
    }
  ]
}
```

GET /api/article/list/{userid}

Hier müssen alle Artikel zurückgegeben werden (Liste), welcher von der via Parameter übergebenen Userid erfasst wurden.

Hier kann die Auswirkung der Annotation `@JsonIdentityReference` gut getestet werden.

GET /api/article/comment/{commentid}

Liefert den Kommentar mit der übergebenen ID zurück.

POST /api/article/comment

Erstellen Sie eine POST-Methode, mit der neue Kommentare zu einem bestehenden Artikel angelegt werden können.

Die Daten sollen in folgender Struktur im Body übergeben werden:

```
{
  "text": "First comment",
  "user": {
    "userid": 2
  },
  "article": {
    "id": 3
  }
}
```

Rückgabe ist vom Typ Response mit dem Status CREATED (201).

Fügen Sie im Location-Header die URL ein, mit welcher auf den soeben erstellten Kommentar zugegriffen werden kann.

Die Rückgabe-Url kann einfach erstellt werden, indem eine UriInfo über den Context injected wird:
`public Response saveComment(Comment comment, @Context UriInfo context)`

`URI uri = context.getAbsolutePathBuilder().path(id).build();`

Zum Aufbau der Response nutzen Sie folgendes Kommando:
`Response.created(uri).build()`

Alternativ könnte mit `Response.entity(comment).build()` auch das neu erstellte Kommentar übertragen werden, würde hier aber nicht wirklich sinnvoll sein.