Hochschule RheinMain
H. Arega, W. Weitz

Studiengang Medieninformatik
Sommersemester 2024

Webbasierte Anwendungen

Web-Ueb-04 (15.05.2024)

Wir bauen unser System nun mit Datenbank-Support und ein paar nützlichen Features aus. Wir strukturieren die Anwendung dabei in drei Schichten. Die Zuständigkeiten sind:

- Eine `@Controller`-Klasse regelt die **Benutzer-Interaktion**,
- sie stützt sich dabei auf eine `@Service`-Klasse, welche die (in diesem Fall noch recht überschaubare) **Anwendungslogik** übernimmt und
- ein JPA-`Repository` zur persistenten **Datenhaltung** unserer Datenobjekte nutzt.

Beachten Sie bitte die Schichtengrenzen. Greifen Sie *nicht* z.B. direkt aus der obersten Schicht (Controller) unter Umgehung der mittleren (Services) direkt auf die Datenhaltung (Repository) zu. Es gibt *keinen* direkten Zugriff des Controllers auf das Repository (auch wenn der Service bei uns noch nicht viel tut).

Aufgabe 1 (Einrichtung JPA zur Datenbanknutzung)

Zur Nutzung von JPA mit der H2-Datenbank fügen Sie bitte folgende Dependencies zu Ihrer `build.gradle` hinzu. Spring Boot konfiguriert dann beim nächsten Build Ihres Projekts alles für die Nutzung dieser Komponenten (was mit Spring ohne Boot schon etwas Arbeit wäre...).

```
dependencies {
  ...
  implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
  runtimeOnly 'com.h2database:h2'
  ...
}
```

Konfigurieren Sie in der `application.properties`-Datei (im `src/main/resources`-Verzeichnis Ihres Projekts) bitte die H2-Datenbank (`http://www.h2database.com`) wie folgt:

```
# JDBC-Datenbank URI, Benutzer/Passwort setzen
spring.datasource.url=jdbc:h2:mem:mifahrdb
spring.datasource.username=h2
spring.datasource.password=h2

# Automatisch Tabellen etc. aus Entities anlegen
spring.jpa.generate-ddl=true

# Bei jedem Anwendungsstart Datenbank droppen und frisch anlegen
# Im in-memory-Betrieb weniger interessant, spaeter mit DB-Dateien aber schon.
spring.jpa.hibernate.ddl-auto=create-drop

# SQL-Logging im Spring-Boot-Log aktivieren zu Guck- und Testzwecken
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type=INFO
```

Die Logging-Einstellungen sollten im Produktionsbetrieb natürlich entfernt werden, aber zum Lernen ist es ganz interessant zu sehen, welche SQL-Anweisungen von JPA erzeugt werden.

Die gezeigte JDBC-URL lässt H2 eine "in memory"-Datenbank `mifahrdb` anlegen. Die Inhalte gehen bei Neustarts der Anwendung verloren, aber H2 ist schnell, hat einen guten SQL-Umfang und kann in diesem Betriebsmodus gut zu Entwicklungs- und Testzwecken genutzt werden. Wie man die H2-Datenbank dateibasiert (und damit ohne Datenverlust bei Neustarts) betreiben kann, wird weiter hinten erklärt.

Aufgabe 2 (Benutzer-Entity und -Repository)

Entity

Bitte legen Sie im Unterpackage `de.hsrm.mi.web.projekt.entities.benutzer` eine `Benutzer`-Klasse als JPA-Entität an.

Die Klasse soll dazu eine eindeutige `id` vom Typ `long` erhalten (von JPA generiert) und vom selben Typ ein ebenfalls von JPA verwaltetes `version`-Feld, um optimistisches Sperren zu ermöglichen. Beide neuen Attribute bekommen einen Getter.

Von Ihrem `BenutzerFormular` können Sie die Attribute `name`, `email`, `passwort`, `geburtstag` und die beiden Mag-/Mag-Nicht-Mengen samt BeanValidation-Constraints übernehmen, und da nur komplette Datensätze in der Datenbank landen sollen, muss sichergestellt werden, dass `name`, `email`, `passwort` und `geburtstag` tatsächlich ausgefüllt sind (nicht leer/`null`, das gilt auch für das `passwort`).

Hinweis: Zur Persisistierung der Mengen sollten Sie dieses Attribut zusätzlich mit der JPA-Annotation `@ElementCollection` versehen, um die korrekte Abbildung von Java `Set` zu gewährleisten. Auch die Mengen-Attribute dürfen nicht `null` sein, leere Mengen sind aber weiterhin zulässig.

Repository

Natürlich sollen die `Benutzer`-Objekte auch in der Datenbank gespeichert werden können. Legen Sie (ebenfalls im Unterpackage `...entities.benutzer`) ein Interface `BenutzerRepository` als geeignete Erweiterung von `JpaRepository` an. Es sind *keine* eigenen Abfragemethoden nötig – alles, was wir für heute brauchen, bringt das `JpaRepository` schon mit.

Formular ergänzen

Um das `BenutzerFormular` bequemer aus einem `Benutzer` befüllen zu können bzw. mit den relevanten Attribute aus dem Formular ein `Benutzer`-Objekt aktualisieren zu können, versehen Sie die `BenutzerFormular`-Klasse bitte mit den folgenden Methoden:

```
public void toBenutzer(Benutzer b)   // befuellt b mit Formularinhalt
public void fromBenutzer(Benutzer b) // fuellt Formularinhalt aus b
```

Wegen seiner speziellen Behandlung in der UI (s.u.) wird das `passwort`-Attribut *nicht* übertragen, das muss (falls gewünscht) also von Hand passieren.

Aufgabe 3 (BenutzerService und -Impl)

Bitte legen Sie nun im Paket `de.hsrm.mi.web.projekt.services.benutzer` die Schnittstelle `BenutzerService` mit folgenden Methoden an:

```
public interface BenutzerService {
    List<Benutzer> holeAlleBenutzer();
    Optional<Benutzer> holeBenutzerMitId(long id);

    Benutzer speichereBenutzer(Benutzer b);
    void loescheBenutzerMitId(long id);
}
```

... und implementieren Sie die Schnittstelle in der Klasse `BenutzerServiceImpl`, ebenfalls im Paket `...services.benutzer`. Verwenden Sie Springs Dependency-Injection-Mechanismus (kein `new`!), um sich per Constructor Injection eine Referenz auf Ihr `BenutzerRepository` im Service bereitstellen zu lassen.

speichereBenutzer(Benutzer b) speichert den übergebenen Benutzer `b` im `BenutzerRepository` ab. Bitte geben Sie das durch die Speicheraktion entstandene Entity zurück (und machen Sie sich klar, ob und worin sich dieses Rückgabeobjekt vom anfangs hereingereichten `b` unterscheiden könnte).

holeBenutzerMitId(Long id) liefert den `Benutzer` mit der gewünschten `id` in einem `Optional` zurück, falls die `id` gefunden wurde, ansonsten ein leeres `Optional`.

holeAlleBenutzer() gibt eine Liste aller `Benutzer`-Objekte, nach dem Inhalt von `name` aufsteigend sortiert, zurück.

Hinweis: Verwenden Sie hierbei die Möglichkeit, der Repository-Standardmethode `findAll()` optional ein Sortierkriterium mitgeben zu können. Die Datenbank liefert Ihnen die `Benutzer`-Objekte dann gleich in der gewünschten Reihenfolge. Sortieren Sie die Rückgabe-Liste der Repository-Abfrage *nicht* nachträglich "von Hand" in Java, eine Datenbank kann das gerade bei großen Datenmengen viel besser.

loescheBenutzerMitId(long id) löscht den Benutzer mit der ID `id` aus dem Repository.

Bitte vergessen Sie das Logging für die Service-Methoden nicht.

Aufgabe 4 (Anpassung BenutzerController)

Um unsere `BenutzerService`-Funktionalität auch nutzen zu können, ergänzen wir jetzt den `BenutzerController`. Dieser erhält wieder per Dependency Injection (kein `new`!) Zugriff auf eine `BenutzerService`-Instanz. Denken Sie daran, dass es *verboten* ist, aus dem Controller direkt auf das Benutzer-Repository zuzugreifen.

Den vom Service zur Bearbeitung aus der Datenbank geholte Bentuzer werden wir uns im Session-Scope merken. Bitte sehen Sie ein entsprechendes Sessionattribut `benutzer` vor und ändern bzw. ergänzen Sie geeignete Handler-Methoden für die folgendenden HTTP-Anfragen:

GET /benutzer/{n} – wenn für n der Wert 0 übergeben wird, soll eine neuer Benutzer erfasst werden. Bitte belegen Sie das Session-Attribut `benutzerformular` mit einem frischen (leeren) Benutzerformular und das neue `benutzer`-Session-Attribut mit einer "leeren" Instanz von `Benutzer`. Um für $n = 0$ die andere Nutzung des `benutzerbearbeiten.html`-Templates deutlicher zu machen, wird bei $n = 0$ als Titel nicht "Benutzerprofil n bearbeiten", sondern "Neues Benutzerprofil" ausgegeben (natürlich geeignet lokalisiert).

Falls $n > 0$, soll das Session-Attribut `benutzerformular` mit einem `BenutzerFormular` belegt werden, dessen Inhalt aus der Datenbank-`Benutzer` mit der übergebenen ID n bezogen wird. Dieses `Benutzer`-Objekt wird im Session-Attribut `benutzer` für später aufgehoben.

POST /benutzer/{n} – nach dem Abschicken des Benutzer wird wie bisher nach (Validierungs-)Fehlern geschaut und der Nutzer zurück in das Formular geschickt, falls es Fehler gab.

Falls keine Validierungsfehler vorliegen, soll der (relevante) Inhalt des ausgefüllten `BenutzerFormular`-Objekts mit Hilfe der oben ergänzten Hilfsmethoden in den im Sessionattribut `benutzer` zwischengespeicherten `Benutzer` übernommen und das Ergebnis mit Hilfe des Service in die Datenbank gespeichert werden. Mit dem aus der Speicherung resultierenden `Benutzer`-Objekt wird dann das Sessionattribut `benutzer` aktualisiert (bitte überlegen, warum das nötig ist – es hat mit optimistischem Sperren zu tun).

Beachten Sie dabei die Behandlung des **passwort**-Formularfelds, dessen Inhalt bei unseren Formular-Korrekturschleifen bei Verwendung des `password`-Typs im HTML `INPUT`-Feld bewusst nicht erneut von Thymeleaf vorbelegt wird. Es wäre natürlich falsch, bei erneutem Abschicken des Formulars ein leeres Passwort-Feld in die Datenbank zu übernehmen, wenn dort zuvor ein Passwort gesetzt war.

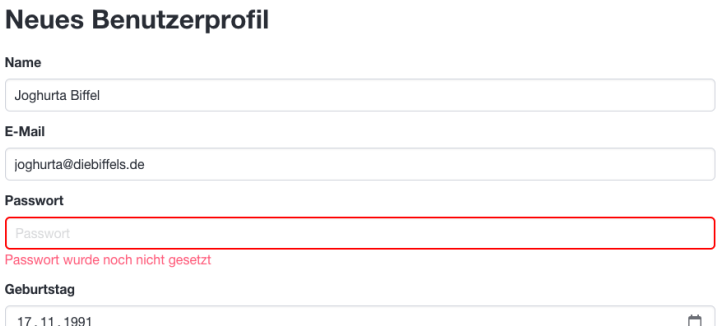
Bitte sorgen Sie dafür, dass bereits in der `Benutzer` Datenbank-Entität einmal gesetzte Passwörter nur mit neuen, nicht-leeren und regelkonformen Passwörtern, nicht jedoch mit Leereinträgen überschrieben werden. Sichern Sie dazu sowohl Ihr `passwort`-Attribut in der `Benutzer` `@Entity` geeignet ab, so dass nur in unserem Sinne (vgl. letztes Übungsblatt) "gute" und gleichzeitig nicht-leere Passwörter abgespeichert werden können.

Passsen Sie bitte zudem Ihre POST-Handlermethode in Ihrem `@Controller` an, so dass bei einem leeren `passwort`-Formulareintrag das bisher in der Datenbank gespeicherte Passwort beibehalten wird. Natürlich wollen wir dem Nutzer auch im Formular Feedback geben, wenn das Abspeichern nicht möglich ist, wenn das Passwortfeld im Formular nicht ausgefüllt wurde **und** der Benutzer noch nicht (mit Passwort) in der Datenbank steht.

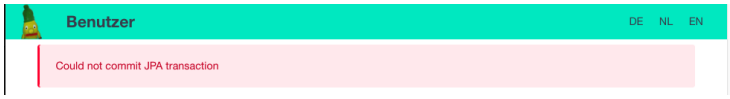
Letztere Überprüfungen sind direkt nicht mit statischen Bean-Validation-Annotationen lösbar, denn ob ein leeres Passwort-Formularfeld in Ordnung ist, hängt davon ab, ob ein Benutzer neu angelegt wird (dann ist ein Passwort nötig) oder ob ein bestehender Benutzer aus der DB später bearbeitet wird (dann bleibt das Passwort aus der Datenbank auch bei Änderung der anderen Angaben erhalten, bis es explizit im Bearbeitungsformular neu gesetzt wird).

Sie können diese Fälle aber leicht in Ihrer POST-Handlermethode unterscheiden. Um das Passwort-Formularfeld bei Bedarf als fehlerhaft zu markieren (wie es bei fehlgeschlagenen Bean-Validation-Checks wäre), können Sie in der Handlermethode auf Ihrem `BindingResult`-Objekt für Ihr Formularobjekt "von Hand" mit der `rejectValue(`*`feldname`*`,` *`Fehlermeldung-ID`*`,` *`Defaultmeldung`*`)` -Methode einzelne Formularfelder als fehlerhaft markieren (natürlich mehrsprachig). Beispiel:

```
bindingResult.rejectValue("passwort",        /* Formularfeld */
      "benutzer.passwort.ungesetzt",         /* Message-Key */
      "Passwort wurde noch nicht gesetzt"); /* Default-Meldung */
```



Beim Speichern in der Datenbank können verschiedene Exceptions auftreten. Bitte fangen Sie im Controller alle `Exception` beim Aufruf von `speichereBenutzer()` ab und sorgen Sie dafür, dass der Nutzer auf dem Eingabeformular bleibt. Damit er im Bilde ist, dass und was schiefgelaufen ist, belegen Sie das `Model`-Attribut `info` mit der Fehlermeldung aus der Exception und geben Sie den Inhalt von `info` oben auf der Bearbeitungsseite aus, sollte er nicht leer sein. Damit dieser Mechanismus künftig auf allen Ihren Seiten einheitlich funktioniert, bietet es sich an, die Anzeige des `info`-Inhalts in das vorhandene Thymeleaf-Fragment für den Seitenkopf einzubauen. Es wird der im `Model`-Attribut `info` gesetzte String ausgegeben, wie er ist (z.B. eine Exception-Message…).



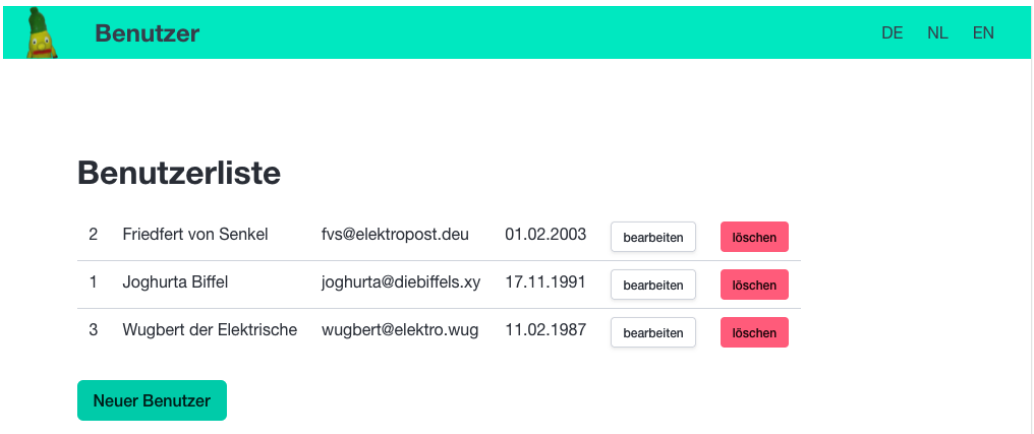
Denken Sie auch daran, dass Exceptions in aller Regel zu loggen sind.

Ein *neu* angelegter Benutzer erhält durch das Abspeichern eine (ebenfalls neue) ID, die wir vorab nicht kennen. Falls die POST-Operation also erfolgreich war und ein neu erfasster Benutzer abgespeichert wurde, leiten Sie den Nutzer bitte per Redirect auf die “korrekte” URL `/benutzer/`id weiter, um nicht auf der Neuanlegen-Pfad `/benutzer/0` zu bleiben. Falls n beim POST nicht 0 war, bleiben Sie einfach auf dem Pfad `/benutzer/`id.

Sie sollten nun in der Lage sein, per URI-Pfad `/benutzer/0` mit dem Bearbeitungsformular neue Benutzer zu erfassen, die dann in der Datenbank landen sollten.

Aufgabe 5 (Benutzer-Liste)

Nun ergänzen wir unseren `BenutzerController` um Handler-Methoden und ein Thymeleaf-Template, um einen Überblick über den Datenbank-Inhalt zu erhalten (zumindest Benutzer-ID, Name, und E-Mail). Das könnte nach Umsetzung der folgenden Schritte z.B. so aussehen:



Das `uebersetzungen.csv` im read.MI wurde um geeignete Einträge für die Seiten- und Tabellenüberschriften ergänzt, damit die Anwendung weiterhin mehrsprachig (DE/EN/NL) benutzt werden kann.

`GET /benutzer` (ohne Pfad-Parameter) benötigt eine neue Handlermethode, welche eine aufsteigend nach Name und (bei gleicher Kategorie) nach E-Mail sortierte Liste aller Benutzer mit Hilfe der entsprechenden Service-Methode aus der Datenbank kramt und mit Hilfe eines (ebenfalls neu anzulegenden) Thymeleaf-Templates `benutzerliste.html` anzeigt, wie oben zu sehen.

Wie dort zu erkennen ist hat jede Zeile einen Bearbeitungs-Link, der auf den URI-Pfad `/benutzer/`n verweist (n ist wieder die ID des Benutzers aus der betreffenden Zeile).

Um die Benutzerliste von überall einfach erreichen zu können, sehen Sie bitte eine passenden Link im Kopf-Templatefragment vor.

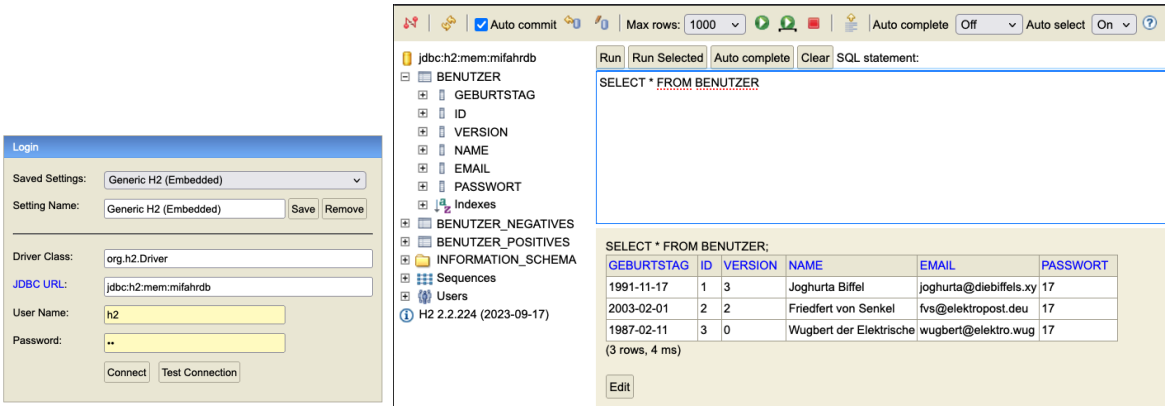
`GET /benutzer/{n}/del` – erfolgt der GET-Aufruf für einen Benutzer n mit zusätzlichem Pfad-Element `del` so soll der Benutzer mit der ID n aus der Datenbank gelöscht und die (nun kürzere) Benutzerliste per Redirect erneut angezeigt werden (der Browser ist also danach auf "`/benutzer`").

Bitte ergänzen Sie je Zeile auch hier einen Lösch-Link, um den betreffenden Eintrag bequem löschen zu können, wie im Screenshot eingangs gezeigt.

Aufgabe 6 (Ergänzender Tipp: H2 SQL-Konsole)

Erzeugen Sie einige Datenbankeinträge mit Hilfe Ihrer Webanwendung und bebrowsen Sie dann `http://localhost:8080/h2-console`. Durch Einbindung der Spring Boot DevTools wurde diese automatisch zusammen mit JPA der H2-Datenbank konfiguriert.

Loggen Sie sich mit Hilfe der Daten aus den `spring.datasource...`-Attributen in `application.properties` ein und sehen Sie sich datenbankseitig Ihre Tabelleninhalte mit Hilfe einiger SQL-Statements an.



Natürlich kann H2 auch das Filesystem zur Datenhaltung verwenden. So bleiben Ihre SQL-Tabellen auch über Neustarts Ihrer Anwendung erhalten. Zum Umschalten ist keine Codeänderung nötig – stellen Sie in der `application.properties` die URL einfach auf `jdbc:h2:~/mifahrdb`, so dass H2 künftig Datei(en) mit dem Stammnamen `mifahrdb...` in Ihrem Homedirectory (`~`) verwendet.

Damit nicht trotzdem bei jedem Anwendungsstart Ihre Datenbank gedroppt und neu angelegt wird, setzen Sie die Eigenschaft `spring.jpa.hibernate.ddl-auto` bitte zudem auf `update`. Starten Sie Ihre Anwendung bitte neu, legen Sie einige Profile an und überprüfen Sie, ob diese über mehrere Neustarts der Serveranwendung erhalten bleiben.

Sie haben nun eine Datenbank-gestützte Webanwendung, mit der Sie Datensätze erfassen, validieren, bearbeiten, auflisten und löschen können! Da JPA unabhängig von der konkreten Datenbank ist, wäre die Nutzung anderer DBMSe wie Postgres im Wesentlichen nur eine Frage der Anpassung Ihrer `build.gradle`-Dependencies und der `application.properties`.

Akzeptanzkriterien für diesen Projektstand:

- Mit URI-Pfad `/benutzer/0` lässt sich ein neuer Benutzer erfassen, die in der DB gespeichert werden. Das Formular ist beim Einstieg leer (auch wenn vorher ein anderer Benutzer bearbeitet wurde). Der Nutzer wird im Erfolgsfall auf den URI-Pfad `/benutzer/`n weitergeleitet (n = Datenbank-ID des gerade gespeicherten Benutzers) und kann die zuvor gemachten Eingaben weiter bearbeiten.
- Wenn ein neuer Benutzer ohne Passwortvergabe angelegt werden soll, erfolgt eine Fehlermeldung und es erfolgt kein neuer Datenbankeintrag.
- Wenn ein bestehender Benutzer bearbeitet wird, wird kein Passwort angezeigt. Wenn das Passwortfeld beim Abschicken des Formulars valide ausgefüllt wurde, wird (auch) das Passwort im Datenbank-Satz geändert. Wenn das Passwort-Formularfeld leer geblieben ist, werden Änderungen an anderen Feldern im Datenbank-Satz übernommen, das Passwort bleibt jedoch unverändert.
- URI-Pfad `/benutzer` zeigt eine nach Name aufsteigend sortierte Liste aller Benutzer aus der DB an.
- Von der Benutzerliste aus lassen sich einzelne Benutzer über einen danebenstehenden Link bearbeiten bzw. löschen.
- Wenn zwei Leute denselben Benutzer bearbeiten und einer seine Änderungen speichert, erhält der zweite beim Abspeichern eine Fehlermeldung und verbleibt auf der Bearbeitungsseite.
- Im für `info`-Meldungen vorgesehenen Ausgabefeld des Kopf-Fragments erscheint in diesem Fehlerfall eine entsprechende Fehlermeldung (Exception-Message genügt).
- Die Anwendung ist auch in den neuen Teilen der UI für DE/EN/NL lokalisiert, umschaltbar über die bereits vorhandenen Links in der Kopfzeile.