

# Webbasierte Anwendungen

Web-Ueb-02 (24.04.2024)

Mit dem heutigen Übungsblatt beginnt das Praktikums-Projekt dieses Semesters. Wir fügen im Laufe der folgenden Wochen je Übungsblatt weitere Bestandteile hinzu, wodurch bis zum Vorlesungsende eine größere Anwendung entsteht, in der die in der Vorlesung behandelten Konzepte und Technologien umgesetzt sind.

- Verwenden Sie bitte kontinuierlich für das gesamte Praktikumsprojekt das (serverseitig im 2024web-Git-Ordner angelegte) Git-Repository 2024webXX mit Ihrem SpringBoot-Gradle-Projekt (XX = Ihre Gruppennummer). Verwenden Sie bitte **genau dieses Repository** und halten Sie Vorgaben aus den Aufgaben ein, insb. bei URI-Pfaden, Parametern, Package-/Klassen-/Methoden-/Template-Namen etc.
- Es muss stets möglich sein, Ihr Repository zu klonen und auf der obersten Verzeichnisebene im master-Branch des resultierenden 2024webXX-Ordners mit `./gradlew build` ohne Weiteres eine lauffähige (testbare) Anwendung zu erhalten bzw. das Projekt mit `./gradlew bootRun` in der Pool-Linux-Umgebung ausführen zu können. Vermeiden Sie daher bitte Abhängigkeiten von Ihrer lokalen Installation (wie z.B. absolute Datei-/Ordnerpfade). Die gesamte Entwicklungshistorie muss anhand des Repositories nachvollziehbar sein, daher bitte das begonnene Repository immer fortführen und nicht löschen oder auf andere wechseln.
- Falls nicht anders angegeben müssen die Lösungen zu den Aufgaben **innerhalb von 12 Tagen** ab Erscheinen des Übungsblattes, also bis einschließlich Montag der übernächsten Woche, in Ihrem Git Repository wie oben beschrieben ausführbar vorliegen und mindestens die am Ende jedes Übungsblattes zusammengefassten Akzeptanzkriterien erfüllen (also wenn die Aufgaben am Mittwoch herauskommen, muss die Lösung bis spätestens zum übernächsten Montag im Git sein). Es dürfen natürlich nur selbst entwickelte (nicht selbst abgeschriebene, nicht selbst KI'te) Lösungen je Gruppe abgegeben werden. Da wir schrittweise eine Gesamtanwendung entwickeln, bauen die Aufgaben aufeinander auf. Es ist daher ratsam, kontinuierlich am Projekt zu arbeiten und Rückstand zu vermeiden.
- Die letzte Vorlesungswoche ist für Nacharbeiten freigehalten (kein neues Blatt).
- Nach größeren zusammenhängenden Projektabschnitten (wird auf den dazu abzuschließenden Übungsblättern angekündigt) wird es nach Ablauf der 12-Tage-Frist Gruppentermine geben, bei denen der Projektstand vorgeführt und besprochen wird. Jedes Gruppenmitglied muss dabei in der Lage sein, die Umsetzung der Aufgaben zu erläutern und auf Aufforderung ad-hoc eigenständig kleinere Änderungen oder Erweiterungen im Code vorzunehmen.
- Wie bereits auf dem ersten Übungsblatt erwähnt müssen eigene Beiträge im Git nachvollziehbar sein. Um das Autorenfeld der Commits korrekt auszufüllen sind dazu der eigene Klarname (Vorname Nachname) und die HSRM-Mailadresse korrekt in die Konfigurationsdatei `$HOME/.gitconfig` einzutragen und für alle eigenen Commits *einheitlich* zu verwenden. Dies gilt insbesondere auch für Commits, die auf privaten Rechnern gemacht werden.
- Von allen Teammitgliedern wird ein ausgewogener, kontinuierlicher und anhand eigener Git-Commits nachvollziehbarer Anteil an der Projektarbeit erwartet ("Pair Programming" ist keine Begründung, wenn das nicht der Fall ist).

# Projekt 2024: Die Mifahrzentrale

Wir bauen in diesem Semester eine Mifahrzentrale, um die nachhaltige Mobilität durch die Organisation gemeinsamer Fahrten zu unterstützen (am besten natürlich mit Ökostrom, Kutschen oder Ruderbooten).

Die Benutzer sollen im weiteren Verlauf sowohl Touren anbieten als auch als Mitnahmebedürftige nach passenden Fahrten recherchieren und Mitfahrten buchen können.

Da jeder Mensch so seine Eigenheiten hat, die auf längeren Touren zu Konflikten führen können, soll in Vorbereitung auf eine später zu implementierende Harmonieoptimierungskomponente jeder Benutzer in seinem Profil Eigenschaften hinterlegen können, die er mag, und solche, die er nicht leiden kann, so dass z.B. keine Nichtrauchernden mit Nikotinabhängigen oder Opersingende mit Meditationsfreunden im selben Fahrzeug landen.

Im ersten Schritt schaffen wir nun eine Möglichkeit, Benutzer-Profile für unser System (nur) zu bearbeiten.

## Aufgabe 1 (Projekt anlegen)

Erstellen Sie bitte ein (Java-/Gradle-basiertes) Spring-Boot-Projekt mit folgenden Angaben in VS Code:

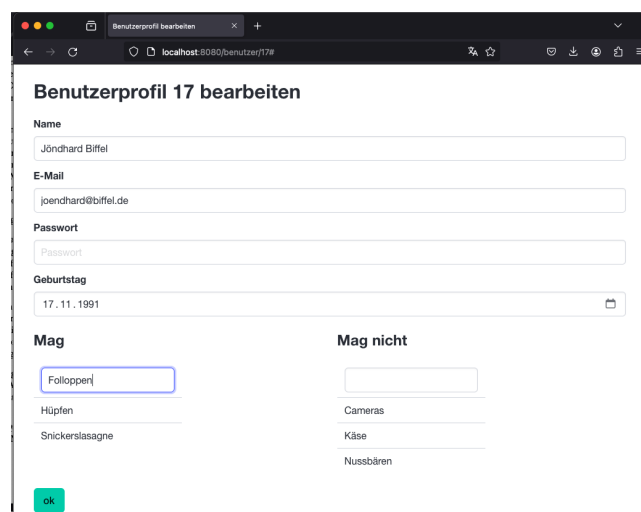
Spring Boot Version	3.2.5
Projektsprache	Java
Group ID	de.hsrn.mi.web
Artifact ID	projekt
Packaging type	Jar
Java version	21

Wählen Sie danach dabei folgende Abhängigkeiten aus: “Spring Web”, “Thymeleaf”, “Spring Boot DevTools” und “Spring Boot Actuator”.

## Aufgabe 2 (Template benutzerbearbeiten.html)

Thymeleaf-Templates werden auf Grundlage “normaler” HTML-Dateien erstellt, die sich auch *ohne Server* direkt in einem Browser darstellen lassen.

Bitte schreiben Sie eine HTML-Datei `benutzerbearbeiten.html`, welche Eingabefelder für die in der Abbildung gezeigten Angaben enthält.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/benutzer/17#'. The page title is 'Benutzerprofil 17 bearbeiten'. The form contains the following fields:

- Name: Jöndhard Biffel
- E-Mail: joendhard@biffel.de
- Passwort: (empty field)
- Geburtsdag: 17.11.1991

Below these fields are two columns of checkboxes for 'Mag' and 'Mag nicht' with associated food items:

Mag	Mag nicht
<input type="checkbox"/> Folloppen	<input type="checkbox"/>
<input type="checkbox"/> Hüpfen	<input type="checkbox"/> Camaras
<input type="checkbox"/> Snickerslasagne	<input type="checkbox"/> Käse
	<input type="checkbox"/> Nussbären

At the bottom left of the form is a green 'OK' button.

Es wird später die Möglichkeit geben, eine begrenzte Anzahl “positiver” (unter der Überschrift “Mag”) und “negativer” (unter “Mag nicht”) Eigenheiten hinzuzufügen (beides Mengen von `String`). Über das Eingabefeld

jeweils unter diesen Überschriften können weitere Eigenheiten hinzugefügt werden. Dies nur vorab zur Orientierung, damit Sie diese Elemente im HTML-Template vorsehen können (die Beispiel-Eingaben in der Abbildung dienen nur zu Demonstrationszwecken).

Verwenden Sie projektbegleitend Ihre in *Auszeichnungssprachen* erworbenen Kenntnisse, damit alles schön aussieht. Falls Sie eines der gängigen CSS-Frameworks (wie Bulma, PureCSS, ...) kennen, können Sie dieses verwenden, müssen es aber nicht (manuelles Styling wie im 2. Semester ist völlig ausreichend).

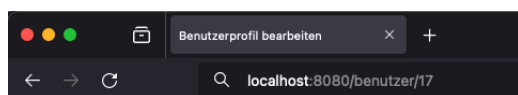
### Aufgabe 3 (BenutzerController.java - GET-Handler)

Bitte legen Sie Ihre `benutzerbearbeiten.html`-Datei vorgesehenen Projekt-Ordner für Thymeleaf-Templates ab und sorgen Sie dafür, dass die Datei als Thymeleaf-Template erkannt werden kann.

Nun soll das Template von der Spring-Anwendung gerendert werden. Legen Sie im Projekt dazu bitte ein Unterpackage `ui` und darin wiederum ein Unterpackage `benutzer` an. Der sich ergebende, komplette Package-Pfad wäre also `de.hsrn.mi.web.projekt.ui.benutzer` (bitte nachvollziehen).

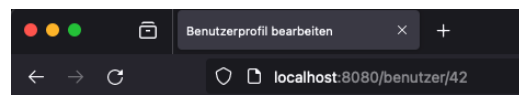
Hier fügen Sie bitte eine `BenutzerController`-Klasse hinzu und ergänzen eine Handler-Methode für HTTP-GET-Anfragen auf URI-Pfade der Form `/benutzer/n`, wobei  $n$  eine Benutzer-Nummer (Ganzzahl, long) ist. Die Methode soll also z.B. auf Anfragen der Form `http://localhost:8080/benutzer/17` oder `http://localhost:8080/benutzer/5` reagieren, indem sie das vorbereitete Thymeleaf-Template anzeigt.

Erweitern Sie das Template und die Handlermethode nun so, dass die Zahl  $n$  aus der Anfrage-URL als Benutzer-nummer in der Titelzeile der Bearbeitungsseite erscheint:



#### Benutzerprofil 17 bearbeiten

Name  
Lindhardt Riffel



#### Benutzerprofil 42 bearbeiten

Name  
Friedfert von Sankail

### Aufgabe 4 (BenutzerController.java – Benutzer bearbeiten)

Ziel ist es nun, das Formular mit einer passenden Java-Datenklasse `BenutzerFormular.java` zu hinterlegen, so dass Eingaben dort abgespeichert werden bzw. der Inhalt des Formularobjekts im angezeigten Formular wiedergegeben wird. Im ersten Schritt lassen wir die Listen “Mag ich” und “Mag ich nicht” zunächst außen vor.

Bitte ergänzen Sie Ihr HTML-Formular passend, so dass das Abschicken ein HTTP-POST mit den Formulardaten auf den ursprünglichen URI-Pfad auslöst (Hinweis: Sie können als Wert für `th:action` z.B. “@{#}” verwenden, damit bleiben Sie auf derselben URL bzw. Seite).

Fügen Sie im Projekt-Unterpaket `ui.benutzer` eine geeignete Datenklasse `BenutzerFormular` passend zu Ihren Eingabefelder im HTML-Formular hinzu, um die Eingaben geeignet abzubilden.

Sicherlich sollen die Formulardaten über mehrere Runden hinweg und auch, wenn man zwischendurch eine andere Seite geöffnet hat und dann wieder zum Formular zurückkehrt, je Benutzersitzung erhalten bleiben. Bitte speichern Sie Ihr Formularobjekt daher im Session-Kontext und sorgen Sie dafür, dass zu Beginn einer Sitzung das zugehörige Session-Attribut mit einem “leeren” Formularobjekt initialisiert wird.

Testen Sie bitte, ob Sie Ihre Seite abrufen können (HTTP GET), dann Werte in die Eingabefelder eintragen und diese nach einem Abschicken des Formulars wieder sehen (wie gesagt bis auf den Abschnitt mit den Angaben, was man mag und was nicht). Testen Sie auch, ob Sie mit dem **selben** Browsertab auf eine andere URL wechseln

(z.B. <https://spring.io>) und danach auf Ihre Seite (z.B. <http://localhost:8080/benutzer/1>) zurückkehren können und Ihre vorherigen Formularinhalte wieder vorfinden.

Ebenso sollten Sie testen, Ihre Anwendung mit zwei verschiedenen Browsern (z.B. Firefox und Chrome – also nicht nur Tabs/Browserfenster desselben Browsers) zu öffnen. Beide Formulare müssen unabhängig voneinander bearbeitbar sein (separate Sessions), wenn die Vorgänge sich gegenseitig beeinflussen, stimmt etwas nicht.

## Aufgabe 5 (Vorlieben und Antipathien hinzufügen)

Nun wollen wir es noch ermöglichen, eine gewisse Anzahl von Vorlieben (“Mag ich”) und Antipathien (“Mag ich nicht”) zu sehen und neue zu erfassen. Im Formular sind dazu unter diesen Überschriften entsprechende Eingabefelder vorgesehen.

Im BenutzerController soll eine Variable `maxwunsch` hinterlegt werden, die zunächst den Wert 5 erhält und bei der Darstellung des Formulars durchgereicht wird. Diese Zahl soll sowohl über den Aufzählungen des Vorlieben-Abschnitts erscheinen als auch deren Anzahl begrenzen, indem das jeweilige Eingabefeld ausgeblendet wird, sobald `maxwunsch`-viele Angaben erfasst sind, so dass man keine weiteren eingeben kann:

Mag	Mag	...	Mag	Mag
(max. 5)	(max. 5)		(max. 5)	(max. 5)
<input type="text" value="mag ich ..."/>	<input type="text" value="mag ich ..."/>		<input type="text" value="Hörnli"/>	<input type="text" value="Hörnli"/>
Mandli	Mandli		Mandli	Mandli
Nussi	Nussi		Nussi	Nussi
Schockchi	Rübli		Rübli	Rübli
	Schockchi		Schockchi	Schockchi

Obwohl im Formular zwei Mengen (*keine* Listen) zur Speicherung verwendet werden, soll die Ausgabe lexikographisch aufsteigend erfolgen. Sie können dies direkt im Thymeleaf-Template durch Verwendung einer geeigneten Funktion aus den `#lists`-Utility-Methoden erreichen (Hinweis: <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#lists>)

Bitte stellen Sie sicher, dass auch die beiden Mag/Mag-nicht-Sammlungen korrekt erhalten bleiben, wenn Sie mehrere Bearbeitungsschritte machen, die Webseite hin- und zurück wechseln und mit zwei Browsern verschiedene Benutzer-Sessions simulieren (wie oben schon für die anderen Felder getestet).

Benutzerprofil 17 bearbeiten

Name

E-Mail

Passwort

Geburtsdag

**Mag**  
(max. 5)  
Hörnli  
Mandli  
Nussi  
Rübli  
Schockchi

**Mag nicht**  
(max. 5)  
  
Gurken  
Käse

ok

## Hinweise:

Aussagekräftiges **Logging** ist für Server-Anwendungen unverzichtbar. Legen Sie bitte für diese und **alle** folgenden Controller-Klassen, die Sie jemals schreiben werden, **regelmäßig** eine `Logger`-Instanz an, über welche alle (wesentlichen) Aufrufe, Änderungen und Fehler protokolliert werden. Verwenden Sie **nie** wieder `“System....println()”` dazu, das bringt Unglück.

**Wichtig: Instanzvariablen** in Controller-Klassen sind **kein** Ersatz für **Session-Attribute** und können u.a. Nebenläufigkeitsprobleme (wie sie in Webanwendungen häufig sind) ergeben, die schwer zu diagnostizieren sind. Spring instanziert jede Controller-Klasse nur einmal und nutzt sie für beliebig viele Requests von beliebig vielen Benutzern.

Ein Objekt als Instanzvariable der Controller-Klasse würde von *allen* Nutzern der Webanwendung gesehen und bearbeitet – also **nicht** je Benutzersitzung getrennt, wie es gedacht ist. Das ist in einer Webanwendung kein kleiner, sondern ein schwerer Fehler.

Sie dürfen im Beispiel das Formular-Datenobjekt daher **nicht** in einer Instanzvariablen der Controllerklasse halten, Instanzvariablen in Controllerklassen sind außer in Ausnahmefällen (z.B. `Logger`) stets zumindest begründungsbedürftig. Dies ist übrigens ein typischer “aber es geht doch”-Fall, bei dem man das Problem nicht sieht, wenn man alleine vor sich hintestet, und im Echtbetrieb mit parallelen Nutzern werden die **fuuuuuurchtbaren** Auswirkungen dann sichtbar.

---

Akzeptanzkriterien für diesen Projektstand:

- `http://localhost:8080/benutzer/123` (oder andere Zahl *n*) liefert Benutzerprofil-Formular mit der übergebenen Zahl *n* in der Überschrift und den beschriebenen Eingabemöglichkeiten.
- Es können maximal so viele Vorlieben/Antipathien erfasst werden, wie in der `BenutzerController`-Klasse in der `maxwunsch`-Variablen hinterlegt ist, danach verschwindet das jeweilige Eingabefeld unter “Mag” bzw. “Mag nicht” für weitere Falschantworten.
- Session-Check: Bei Wechsel von Benutzerbearbeitungs-Seite auf eine andere Webseite und Wiederaufruf der Benutzerbearbeitungs-Seite erscheinen die vorherigen Inhalte wieder. Dies gilt auch für die `Mag(nicht)Listen`.
- Session-Check: Zwei Browser (nicht nur zwei Fenster/Tabs desselben Browsers) öffnen, beide Sitzungen müssen unabhängig sein (also nicht gegenseitig Inhalte überschreiben oder übernehmen).

## Für die gesamte Aufgabe und darüber hinaus gilt...

Bitte gehen Sie in **kleinen Schritten** vor, **testen** Sie häufig und **verwenden Sie** `Logger` (nie `System...println()`), um sich Infos aus dem Innenleben Ihrer Handler-Methoden geben zu lassen.

Nutzen Sie die Übung auch, um sich tiefer mit dem **Debugger** vertraut zu machen und die (unvermeidlichen) Exception-Tracebacks lesen zu lernen. Es lohnt sich hier, von unten anzufangen – die interessantesten sind oft die beiden letzten Zwischenüberschriften.

## Für eine ruhige Minute – Wie finden sich die Web-Grundkonzepte vom Anfang wieder?

Sie haben auf diesem Blatt gesehen, wie einfach das Halten von **Sitzungskontext** sein kann. Die zugrunde liegenden Probleme und Einschränkungen aus dem HTTP/Web-Umfeld sind aber immer noch dieselben.

Es lohnt sich daher, in einem ruhigen Moment den Abschnitt “Grundprobleme von Webanwendungen” vor diesem Hintergrund noch einmal durchzusehen und sich zu jedem Teilproblem klar zu machen, wie Spring WebMVC konkret damit umgeht. Das vertieft einerseits das Verständnis des Frameworks und hilft andererseits nachzuvollziehen, welche vielleicht als umständlich empfundenen Schritte (wie die Verwendung eigener `th:action` und `th:href`-Attribute in HTML-Templates) notwendig sind, um mit Einschränkungen von HTTP & Co umzugehen.

Auf diesem Blatt haben wir einfach ein Java-Formularobjekt im Model abgelegt, und weil wir den betreffenden Attributnamen zum “SessionAttribute” erklärt haben, bleibt der Inhalt während der ganzen Sitzung über viele Aufrufe hinweg erhalten und kann in verschiedenen Handlermethoden/Ansichten genutzt werden.

Natürlich kann Java/Spring auch nicht zaubern – auch hier wird eine Sitzungs-ID mit dem Standardnamen **jsessionId bei allen Anfragen hin- und hergereicht** – als Cookie, verstecktes Formularfeld oder angehängt an (generierte) Link-URLs. Die Möglichkeit, diese Zusatzinfos automatisch auf verschiedene Arten auch in HTML-Templates (Formular-Action, Link-HREF) hinzuzufügen ist auch der Grund, warum Sie in Templates die betreffenden `th:-Attribute` verwenden sollten (also `th:action` statt nur `action` in Formularen und `th:href` statt `href` bei Links).

Nutzen Sie die “Webentwickler-Tools” (`strg + shift + i`) Ihres Browsers, um bei offener “Netzwerkanalyse” einmal durch alle Stationen Ihrer Anwendung durchzuklicken und sich die Request-Inhalte näher anzusehen.

Suchen Sie dabei nach Auftreten von `jsessionid` im Requestpfad, unter “Cookies” und im Sourcecode der empfangenen HTML-Seite, insbesondere von der Formular-Seite zur Profil-Erfassung. Nichts Auffälliges? Cookies werden vorrangig genutzt, um die `jsessionid` zu halten, da das am einfachsten ist und kein “Umschreiben” von URLs erfordert.

Bauen Sie einmal zwei Links z.B. auf `/benutzer/1` und `/benutzer/17` in die Seite ein. Füllen Sie das Benutzer-Formular aus, schicken Sie es ab, und wechseln Sie dann mit den beiden Links hin und her. Die Formularinhalte müssten erhalten bleiben (Session).

Stellen Sie Ihren Browser nun einmal so ein, dass alle Cookies geblockt werden, und rufen Sie die Formularseite nochmals ab. Wenn Sie die `th:-Attribute` konsequent genutzt haben, wird Ihre Anwendung weiterhin funktionieren. Das Framework reagiert **automatisch** auf die Nichtverfügbarkeit von Cookies und weicht auf die Einbettung der `jsessionid` z.B. in die Link-URL aus (Sie sehen die `jsessionid` dann entsprechend nach Anklicken eines der Links oben im URL-Feld des Browsers).

Sie als Anwendungsentwickler werden damit deutlich entlastet. Sie können “normal” entwickeln und sagen einmal, welche Model-Attribute im Session-Scope gehalten werden sollen – das Framework kümmert sich darum, wie unter den aktuellen Gegebenheiten das Ping-Pong-Spiel mit der SessionID umgesetzt werden kann. Sie müssen dem Framework nur (durch das Nutzen der “richtigen” `th:-Attribute` die Möglichkeit dazu geben).

Vollziehen Sie dies bitte nach. Vergessen Sie danach bitte nicht, Ihren Browser wieder in den gewohnten Normalzustand zu versetzen.