Praktikum zu Web-Technologien/Web-Engineering, Wintersemester 2023/2024

Praktikum 10 (Node.js und Express, Bearbeitungszeit: 2 Wochen)

Im Rahmen des Praktikums entwickeln wir eine Web-Anwendung, welche wir Schritt für Schritt mit weiteren Anforderungen, Funktionen und Technologien erweitern.

Meilenstein 3: "JavaScript"
Sobald Sie Praktikum 8, 9 und 10 vollständig bearbeitet haben, haben Sie Meilenstein 3 erreicht! Sprechen Sie uns (Sven Jörges, Andreas Harrer oder Julian Feder) im Praktikum an. Wir schauen uns dann gemeinsam Ihren Stand in einer kleinen Abnahme an. Dabei muss Ihre Gruppe, in welcher Sie die Praktika bearbeitet haben, vollständig anwesend sein. Für diesen Meilenstein sind insgesamt 5 Bonuspunkte erreichbar. Dies ist der letzte Meilenstein. **Wichtig: Abnahmen erfolgen nur bis maximal 18.01.24 (Ende der Vorlesungszeit)!**

Hinweis zum entstehenden Code:

- Zur Verwaltung und zum kollaborativen Bearbeiten Ihres Quellcodes empfehlen wir Ihnen die Nutzung von Git (z.B. in Verbindung mit dem GitLab-Server des FB4, Login per FH-Account). Eine kurze Einführung in Git finden Sie hier.
- Falls Sie Git nicht verwenden, so legen Sie Ihren Quellcode pro Praktikumsaufgabe in separaten Verzeichnissen (z.B. „praktikum2“, „praktikum3“) ab, damit die einzelnen Entwicklungsschritte bei den Abnahmen ersichtlich sind.

Hintergrund: Auf dem Weg zur Client-Server-Architektur

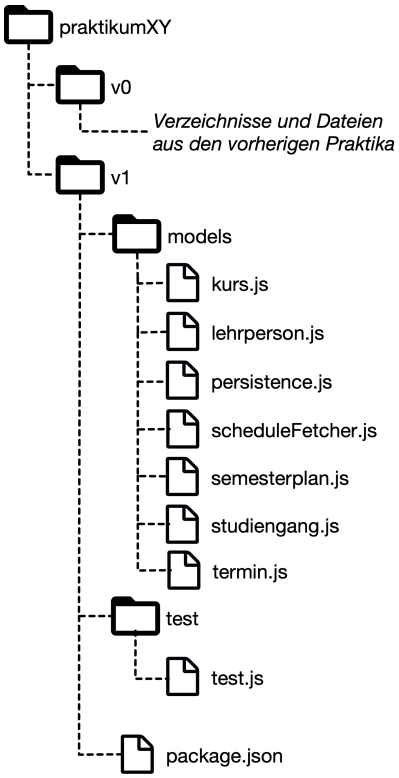
Bisher handelt es sich bei unserer App um eine rein client-seitige Web-Anwendung. In den verbleibenden Praktika bauen wir sie zu einer server-seitigen Web-Anwendung auf Basis von Node.js und Express um. Dabei werden wir zunächst dafür sorgen, dass die Anwendung ihre Daten von der Stundenplan-API des FB 4 beziehen und in ein gewünschtes Datenformat konvertieren kann (mit Hilfe von Node.js → Teil 1). Danach werden wir das in den bisherigen Praktika erarbeitete Frontend (HTML, CSS und client-seitiges JavaScript) so umbauen, dass es über Express server-seitig und dynamisch verfügbar ist (→ Teil 2).

Teil 1: Stundenplandaten mit Node.js abrufen

Vor der Bearbeitung von Teil 1 sollten Sie sich die Lernmodule G-01 - G-06 in ILIAS anschauen. Diese vermitteln alle notwendigen Grundlagen für die Lösung dieser Aufgabe.

Aufgabe 1: Vorbereitung des Projekts

Legen Sie folgende Verzeichnisstruktur für das Projekt an:



- Das Verzeichnis "models" enthält Code, der sich mit der Datenhaltung und -aufbereitung unserer App beschäftigt. Die Dateien "persistence.js" (als Vorlage, die von Ihnen fertigzustellen ist) und "scheduleFetcher.js" (bereits vollständig) finden Sie in ILIAS. Die übrigen Dateien im Verzeichnis "models" ("kurs.js", "lehrperson.js", "semesterplan.js", "studiengang.js", "termin.js") realisieren die Fachobjekte unserer App (Details siehe Aufgabe 2).
- Im Verzeichnis "test" befindet sich ein Test-Web-Server, welchen Sie in Aufgabe 3 realisieren.
- Die Datei "package.json" wird durch das Programm *npm* automatisch erzeugt und bindet eine externe Bibliothek ein, die wir für unser Projekt benötigen (Details siehe Aufgabe 1).

Gehen Sie dabei wie folgt vor:

1. Verschieben Sie die Dateien und Verzeichnisse aus dem Stand von Praktikum 9 in das Verzeichnis "v0", damit sie uns dort noch für den anstehenden Umbau zur Verfügung stehen.

2. Erstellen Sie das neue Verzeichnis "v1", und darin die neuen Verzeichnisse "models" und "test".

3. Das Verzeichnis "models" soll den Code enthalten, der sich mit der Datenhaltung und -aufbereitung unserer App beschäftigt. Laden Sie das ZIP-Archiv praktikum10-dateien.zip aus ILIAS herunter und entpacken Sie dieses. Verschieben Sie die enthaltenen Dateien "persistence.js" und "scheduleFetcher.js" in das "models"-Verzeichnis. Die ebenfalls enthaltene Datei "routes.js" benötigen wir erst für Teil 2.

4. Für das Abrufen der Daten von der Stundenplan-API benötigt das Modul "scheduleFetcher.js" die externe Bibliothek "node-fetch". Diese steht über den Paket-Manager *npm* von Node.js zur Verfügung. *npm* ist Teil der Node.js-Installation, welche Sie bereits im vorherigen Praktikum durchgeführt haben. Um die Bibliothek *node-fetch* in unserem Projekt einzubinden, öffnen Sie eine Konsole (siehe Hinweise aus Praktikum 9) im Verzeichnis "v1" und geben Sie folgende Befehle ein:

   ```
   $ npm init
   $ npm install node-fetch
   ```

   Erklärung der Befehle:

   1. `npm init` erstellt eine grundlegende Projektkonfiguration. Das Kommando stellt Ihnen dazu einige Fragen, z.B. zum Namen Ihrer App. Wenn Sie sich bei einer Frage nicht sicher sind, können Sie diese auch einfach ohne Eingabe durch Drücken von *Return*/*Enter* bestätigen, dann setzt *npm* an dieser Stelle einen Standardwert ein. Einzige wichtige Einstellung: Legen Sie bei der Frage nach dem "Entry Point" der Anwendung die Datei "app.js" fest. Nach Ausführung des Kommandos

sollte Ihr Projekt die neue Datei "package.json" enthalten.

2. `npm install node-fetch` bindet schließlich die Bibliothek *node-fetch* in unser Projekt ein. Dazu wird die Bibliothek (sowie deren abhängige Bibliotheken) heruntergeladen und in dem Verzeichnis "node_modules" gelagert. Zudem wird die Bibliothek in der "package.json"-Datei im Abschnitt `dependencies` als Abhängigkeit unseres Projekts vermerkt.

Hinweise:

- Weitere Erläuterungen zu *npm* folgen mit Lernmodul *H-01: Einführung und Installation* im Themenbereich *"H - Express"*. Die obige Erklärung reicht jedoch zum Lösen von Teil 1 dieses Praktikums völlig aus.
- Falls Sie mit Git arbeiten, so nehmen Sie das Verzeichnis "node_modules" von der Versionskontrolle aus, indem Sie dieses in der Datei ".gitignore" eintragen.

Aufgabe 2: Fachobjekte einbauen und aufbereiten

In diesem Schritt sorgen wir dafür, dass unsere Anwendung die benötigten Datenstrukturen unterstützt und die von der Stundenplan-API importierten Informationen entsprechend konvertieren kann. Gehen Sie wie folgt vor:

1. In Praktikum 8 haben Sie die Fachobjekte `Kurs`, `Lehrperson`, `Termin`, `Studiengang` und `Semesterplan` bereits implementiert. Kopieren Sie den entsprechenden Code so in das Verzeichnis `models`, dass für jedes Fachobjekt ein Modul (also eine `.js`-Datei) entsteht (siehe Abbildung in Aufgabe 1). Vergessen Sie nicht, die *Schnittstelle* jedes Moduls zu definieren.

2. Ergänzen Sie die Funktion `istValiderTyp` im Fachobjekt `Kurs`, so dass zusätzlich der Kurstyp "Organisatorische Veranstaltung" mit dem Wert `Org` unterstützt wird.

3. Ergänzen Sie das Fachobjekt `Kurs` um ein zusätzliches Attribut `gruppenbuchstabe`, so dass der Konstruktor so aussieht (Beispiel in Klassensyntax, Parameterreihenfolge beachten!):

```
class Kurs {
  constructor(modulId, name, typ, studiengang, semester, gruppenbuchstabe,
              lehrperson, termin) {
     // ...
  }

  //...
}
```

4. Ergänzen Sie die Datei "persistence.js", die wir in Aufgabe 1 in unser Projekt kopiert haben. Diese Datei importiert Daten zu Studiengängen, Modulen, Stundenplänen etc. mit Hilfe des "scheduleFetcher.js"-Moduls (siehe Funktion `initialisiereLehrangebot`). Die importierten Daten werden dann in dem Array `lehrangebot` verwaltet. In der Datei "persistence.js" ist schon teilweise Code vorgegeben, manche Stellen sind jedoch noch mit `[TODO]`-Kommentaren versehen. Vervollständigen Sie die Datei, indem Sie die mit `[TODO]` markierten Stellen implementieren:

    1. "persistence.js" verwendet zunächst nur "scheduleFetcher.js" als externes Modul. Binden Sie weitere benötigte Module geeignet ein.

    2. Ergänzen Sie "persistence.js" um folgende weitere Funktionen:

        1. `ermittleStudiengangZuId(id: string): Studiengang`
        Liefert das `Studiengang`-Objekt mit der entsprechenden ID oder `undefined`, wenn kein entsprechendes `Studiengang`-Objekt im Lehrangebot vorhanden ist.

        2. `ermittleKursZuStudiengangUndId(studiengangId: string, kursId: string): Kurs`
        Liefert das `Kurs`-Objekt mit der entsprechenden ID zum Studiengang mit der gegebenen ID, oder `undefined`, wenn kein entsprechendes `Kurs`-Objekt im Studiengang vorhanden ist.

        3. `holeAlleStudiengaenge(): Studiengang[]`
        Liefert alle im Lehrangebot enthaltenen Studiengänge.

    3. Ergänzen Sie die *Schnittstelle* des Moduls "persistence.js". Konkret sollen das `lehrangebot`-Array sowie alle Funktionen

durch andere Module genutzt werden können.

Aufgabe 3: Test-Web-Server bauen

Um die Funktionstüchtigkeit des bisherigen Standes zu überprüfen, realisieren Sie nun einen kleinen Test-Web-Server. Bei Aufruf einer bestimmten URL soll dieser die importierten Stundenplandaten in einer rudimentären HTML-Seite darstellen. Gehen Sie dabei wie folgt vor:

1. Realisieren Sie den Web-Server in der Datei "v1/test/test.js".
2. Verwenden Sie zur Realisierung das Node.js-Modul "http" sowie unser "persistence.js"-Modul aus Aufgabe 2.
3. Sorgen Sie dafür, dass der Web-Server unter der URL http://localhost:8844 erreichbar ist.
4. Beim Eintreffen einer beliebigen HTTP-Anfrage soll der Web-Server eine rudimentäre HTML-Seite liefern, welche alle Studiengänge mit den jeweils enthaltenen Modulen anzeigt.
5. Erzeugen Sie die entsprechende HTML-Seite *dynamisch mit Hilfe von Template-Literalen*. Dabei soll zu jedem `Studiengang`-Objekt der Name, die ID sowie die Gesamtzahl der zugehörigen Module angezeigt werden. Die zugehörigen Module sollen zudem jeweils in einer ungeordneten Liste dargestellt werden. Zu jedem Modul soll diese Liste die Modul-ID, den Namen, den Typ sowie den Nachnamen der Lehrperson anzeigen.

   Die resultierende HTML-Seite soll wie in folgendem Ausschnitt strukturiert sein:

   # Study Planner Test

   ## Praktische Inf. StgPO 2019 (INPBPI)

   206 Kurse enthalten:

   - 46990 BWL-Anwendungen (1. Semesterhälfte) [StgPO: PI-19, SYT/SOT-15] (V, Kunau)
   - 46990 BWL-Anwendungen (2. Semesterhälfte) [StgPO: PI-19, SYT/SOT-15] (V, Teschler-Nunkesser)
   - 42073 Mathematik für Informatik 3 (V, Kuhnt)
   - 42073 Mathematik für Informatik 3 (Kuhnt) (T, Bernstein)
   - 46990 BWL-Anwendungen (1. Semesterhälfte) [StgPO: PI-19, SYT/SOT-15] (P, Kunau)
   - 42012 Algorithmen und Datenstrukturen (V, Stark)
   - [...]

   ## Technische Inf. StgPO 2019 (INPBTI)

   202 Kurse enthalten:

   - 46990 BWL-Anwendungen (1. Semesterhälfte) [StgPO: PI-19, SYT/SOT-15] (V, Kunau)
   - 42073 Mathematik für Informatik 3 (V, Kuhnt)
   - 46990 BWL-Anwendungen (2. Semesterhälfte) [StgPO: PI-19, SYT/SOT-15] (V, Teschler-Nunkesser)
   - 42073 Mathematik für Informatik 3 (Kuhnt) (T, Bernstein)
   - 46990 BWL-Anwendungen (1. Semesterhälfte) [StgPO: PI-19, SYT/SOT-15] (P, Kunau)
   - 42012 Algorithmen und Datenstrukturen (V, Stark)
   - [...]

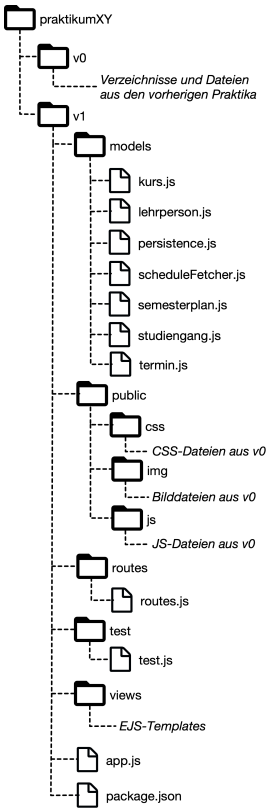
   Die `[...]` im obigen Ausschnitt dienen nur der Veranschaulichung - zeigen Sie einfach alle Daten an. Ein Styling mittels CSS ist nicht erforderlich. Nutzen Sie geeignete semantische HTML-Elemente zur Struktuierung der Seite.
6. Sorgen Sie dafür, dass beim Hochfahren des Test-Servers der Import der Daten über die Stundenplan-API durchgeführt wird (→ Funktion `initialisiereLehrangebot` in "persistence.js").

Teil 2: Frontend mit Express

Vor der Bearbeitung von Teil 2 sollten Sie sich die Lernmodule H-01 - H-05 in ILIAS anschauen. Diese vermitteln alle notwendigen Grundlagen für die Lösung dieser Aufgabe.

Aufgabe 4: Ergänzung der Projektstruktur

Um die bisherige Oberfläche unserer Web-Anwendung in ein server-seitiges und dynamisches Frontend auf der Basis von Express zu überführen, erweitern wir die im vorherigen Praktikum angelegte Projektstruktur wie folgt:



Neue Verzeichnisse und Dateien:

- Im Verzeichnis "public" befinden sich statische Ressourcen, die an Clients ausgeliefert werden sollen (CSS-Stylesheets, Bilder, client-seitige JavaScript-Dateien → Details siehe Aufgabe 6).
- Die Datei "routes/routes.js" definiert die Logik für das Routing der Web-Anwendung (Details siehe Aufgabe 6).
- Das Verzeichnis "views" enthält *EJS-Templates* zur dynamischen Erzeugung der Seiten unserer Anwendung (Details siehe Aufgabe 7).
- Die Datei "app.js" ist der Einstiegspunkt, über welchen sich unsere Web-Anwendung starten lässt.

Gehen Sie dabei wie folgt vor:

1. Kopieren Sie folgende Dateien aus dem Stand von Praktikum 9 (Verzeichnis "v0") in die Verzeichnisstruktur:
    1. Alle `.css`-Dateien in das Verzeichnis "v1/public/css",
    2. alle Bilddateien in das Verzeichnis "v1/public/img",
    3. ggf. benötigte `.js`-Dateien (Achtung, nur client-seitiges JavaScript!) in das Verzeichnis "v1/public/js".
2. Installieren Sie mit Hilfe von *npm* im Verzeichnis "v1" Express und EJS:

    ```
    $ npm install express
    $ npm install ejs
    ```

    Diese sollten anschließend in der "package.json"-Datei im Abschnitt `dependencies` zu finden sein.

Aufgabe 5: Persistenz und Funktionen für Semesterpläne

Ergänzen Sie das Modul "persistence.js" um folgende weitere Funktionen:

1. `erstelleSemesterplan(name: string, semester: string, jahr: number, studiengangId: string, kurse: Kurs[])`

Erstellt ein neues `Semesterplan`-Objekt. Verwalten Sie alle Semesterpläne in einem Array `semesterplaene`.

2. `ermittleSemesterplanZuId(id: string): Semesterplan`
   Liefert das `Semesterplan`-Objekt mit der entsprechenden ID oder `undefined`, wenn kein entsprechendes `Semesterplan`-Objekt im `semesterplaene`-Array vorhanden ist.

3. `holePlaeneGruppiertNachSemester()` und `holePlaeneGruppiertNachStudiengang()`:
   Liefert alle Semesterpläne, gruppiert nach Semester bzw. Studiengang. Verwenden Sie zur Implementierung die aus Praktikum 9 bekannte Funktion `gruppiereNach` wieder, indem Sie diese in die "persistence.js" kopieren.

Aufgabe 6: Modulare Realisierung der Anwendung

Realisieren Sie nun die server-seitige Web-Anwendung mit Hilfe von Express und EJS. Gehen Sie wie folgt vor:

1. Sorgen Sie dafür, dass die Anwendung unter der URL http://localhost:8123 erreichbar ist.

2. Realisieren Sie die Datei "v1/app.js" als Einstiegspunkt für die Anwendung. Hier können Sie auch benötigte Einstellungen vornehmen.

3. Lagern Sie die Logik für das Routing in ein eigenes Modul "v1/routes/routes.js" aus. Verschieben Sie die in praktikum10-dateien.zip enthaltene Datei "routes.js" in das neue "routes"-Verzeichnis. Das Modul bereitet benötigte URLs/Routen als Middleware-Funktionen vor. Ergänzen Sie diese, so dass folgende URLs/Routen durch die Anwendung unterstützt werden (mit `[TODO]` markierte Kommentare in "routes.js" beachten):

   1. http://localhost:8123/index?gruppierung=X → Liste der Semesterpläne (in v0: `index.html`), gruppiert nach Semesterplänen oder Studiengang. *X* kann dabei entweder den Wert "Studiengang" oder "Semesterplan" besitzen. Wird der Parameter "gruppierung" nicht angegeben, so soll standardmäßig nach Semester gruppiert werden.
   2. http://localhost:8123/plan?id=X → Detailseite zum Semesterplan (in v0: `plan.html`). *X* bezeichnet die ID des Semesterplanes, der angezeigt werden soll.
   3. http://localhost:8123/kurs?sid=X&kid=Y → Detailseite zum Kurs (in v0: `kurs.html`). *X* bezeichnet die ID des Studienganges, *Y* bezeichnet die ID des Kurses in Studiengang *X*).
   4. http://localhost:8123/neu und http://localhost:8123/waehleStudiengang → Benötigt zum Erstellen eines neuen Semesterplanes (Details siehe Aufgabe 7, Punkt 6).
   5. http://localhost:8123 → Leitet weiter auf http://localhost:8123/index.
   6. Statische Inhalte (Bilder, CSS, client-seitiges JavaScript) sollen aus dem Verzeichnis "public" (bzw. aus darin enthaltenen Unterverzeichnissen) heraus ausgeliefert werden.
   7. Für nicht existierende URLs und Ressourcen soll die Anwendung den Statuscode 404 (NOT FOUND) sowie eine Fehlerseite zurückliefern. Diese Fehlerseite soll eine Meldung anzeigen und BenutzerInnen informieren, dass die angefragte Seite/Ressource nicht gefunden wurde. Erstellen Sie die Fehlerseite als *EJS-Template* (siehe Aufgabe 7).

Aufgabe 7: Dynamische Erzeugung der Webseiten

Sorgen Sie nun dafür, dass alle Webseiten der Anwendung server-seitig und dynamisch mit Hilfe von EJS erzeugt werden:

1. *Templates:* Realisieren Sie alle HTML-Seiten Ihrer Web-Anwendung (auch die Fehlerseite!) als *EJS-Templates* und legen Sie diese im Verzeichnis "views" ab.

   Tipp: Verwenden Sie die HTML-Dateien aus dem Stand von Praktikum 9 (v0) als Vorlage.

2. *Modularisierung:* Lagern Sie die Bereiche, die sich auf allen HTML-Seiten wiederholen (d.h. Kopfbereich, Fußbereich und Bereich für aktuelle Meldungen), in eigene EJS-Templates aus. Verwenden Sie `include`, um diese Bereiche in *allen Seiten* jeweils an den richtigen Stellen einzubinden.

3. *Navigation:* Da wir nun über die Liste der Semesterpläne bzw. Detailseite eines Semesterplanes auf die entsprechenden Detailseiten navigieren können, benötigen wir die Hyperlinks "Details zum Semesterplan" und "Details zum Kurs" im Navigationsbereich nicht mehr. Entfernen Sie diese entsprechend.

4. *Dynamische Inhalte:* Sorgen Sie dafür, dass die Inhalte *aller Seiten* nun mit EJS *vollständig dynamisch* eingefügt werden. Die Seiten sollen also nicht, wie bisher, feste Beispieldaten enthalten, sondern ihre Daten aus der Datenhaltung (Modul "persistence.js") beziehen.

5. *Gruppierung der Semesterpläne:* In Praktikum 9 haben Sie die Gruppierung der Semesterpläne (Seite *Liste der Semesterpläne*, in v0: `index.html`) dynamisch mit Hilfe von clientseitigem JavaScript (DOM-Manipulation) realisiert. Lösen Sie das Umschalten der Gruppierung (nach Semester oder nach Studiengang) nun serverseitig. Gehen Sie dazu wie folgt vor:

    1. Entfernen Sie das Script `index.js` in v1.

       Hinweis: In v0 soll für die Abnahme weiterhin die Variante mit der DOM-Manipulation vorhanden sein und funktionieren.

    2. Ergänzen Sie die Seite *Liste der Semesterpläne* so, dass sich die Dropdown-Box zum Umschalten der Gruppierung (Semester oder Studiengang) nun in einem Formular befindet. Fügen Sie dem Formular zusätzlich eine Schaltfläche zum Absenden hinzu.

    3. Binden Sie das Formular an die URL http://localhost:8123/index an (siehe Aufgabe 6, Punkt 3.1). Übermitteln Sie die Einstellung der Dropdown-Box über den Anfrageparameter "gruppierung", damit Ihr Server weiß, welche Ansicht zu erzeugen ist.

6. *Erstellen eines neuen Semesterplanes:* Auch die Formulare zum Erstellen eines neuen Semesterplanes sollen nun dynamisch erzeugt und an unseren Express-Server angebunden werden. Realisieren Sie dazu folgendes Verhalten:

    1. Ein Aufruf auf die URL http://localhost:8123/neu (siehe Aufgabe 6, Punkt 3.4) soll die Seite zu Schritt 1 (Auswahl des Studienganges, in v0: `plan-neu-schritt1.html`) anzeigen. Die auswählbaren Studiengänge sollen nun von der Datenhaltung (Modul "persistence.js") bezogen werden.

    2. Nach Auswahl eines Studienganges und Absenden des Formulars soll die ID des gewählten Studienganges an den Server übertragen werden (URL http://localhost:8123/waehleStudiengang, siehe Aufgabe 6, Punkt 3.4). Auf der Serverseite soll dann das passende `Studiengang`-Objekt mitsamt den zugehörigen Kursen von der Datenhaltung (Modul "persistence.js") bezogen werden. Diese Daten sollen dann verwendet werden, um die passende Seite zu Schritt 2 (Eingabe weiterer Daten und Auswahl der Kurse, in v0: `plan-neu-schritt2.html`) anzuzeigen.

    3. In Schritt 2 gibt die Benutzerin die übrigen Daten des Semesterplanes (Name, Semester, Jahr) ein und wählt entsprechende Kurse aus. Nach Absenden des Formulars werden diese Daten wieder an die URL http://localhost:8123/neu (siehe Aufgabe 6, Punkt 3.4) gesendet. Auf Server-Seite wird aus den gesendeten Daten mit Hilfe der Funktion `erstelleSemesterplan` (siehe Aufgabe 5, Punkt 1) ein neuer Semesterplan erstellt. Danach wird die Benutzerin auf die Seite *Liste der Semesterpläne* weitergeleitet. Der neu angelegte Semesterplan soll dort direkt sichtbar sein.

7. *Layout beibehalten:* Alle Seiten sollen grundsätzlich so aussehen und funktionieren wie im Stand von Praktikum 9 (mit Ausnahme des verkürzten Navigationsber

Allgemeine Hinweise:

- Validieren Sie Ihren HTML-Code mit dem W3C Markup Validator. Der entstehende HTML-Code soll beim Check keine Fehler und Warnungen produzieren.
- Ausnahme: Warnungen bzgl. `date` - und `time` -Eingabefeldern können Sie ignorieren.
- Validieren Sie Ihren CSS-Code mit dem W3C CSS Validation Service. Der entstehende CSS-Code soll beim Check keine Fehler und Warnungen produzieren.