Praktikum zu Web-Technologien, Sommersemester 2023

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 07.07.23 (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

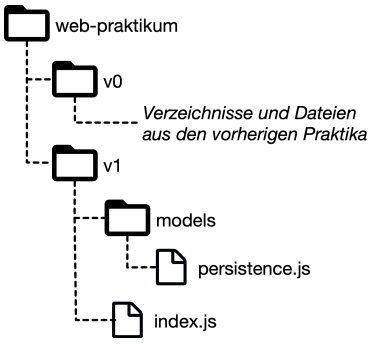
Bisher handelt es sich bei unserer Tutorial-App um eine rein client-seitige Web-Anwendung. In den verbleibenden Praktika bauen wir sie zu einer server-seitigen Web-Anwendung auf Basis von Node.js und Express um. Dabei werden wir zunächst dafür sorgen, dass die Suche nach Tutorials auf der Seite mit der *Liste der Kategorien* (`list.html`) tatsächlich funktioniert (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: Tutorial-Suche mit Node.js

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 und Persistenzmodul

Legen Sie folgende Verzeichnisstruktur für das Projekt an:



- Das Verzeichnis "models" enthält Code, der sich mit der Datenhaltung und -aufbereitung unserer App beschäftigt. Innerhalb dieses Praktikums erstellen Sie dort die Datei "persistence.js" (Details siehe Aufgabe 1.3).
- Die Datei "index.js" ist der Startpunkt unseres Servers (Details siehe Aufgabe 2).

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 das neue Verzeichnis "models".

3. Das Verzeichnis "models" soll den Code enthalten, der sich mit der Datenhaltung und -aufbereitung unserer App beschäftigt. Legen Sie dort eine neue Datei "persistence.js" an. Diese Datei stellt ein Modul für die Persistenz der App dar. Füllen Sie die Datei wie folgt:

    1. Kopieren Sie den JavaScript-Code, welchen Sie in Praktikum 8 entwickelt haben, in die Datei "persistence.js". Dies umfasst die Fachobjekte `Kategorie`, `Tutorial`, `Kapitel` und `Bild`, die nicht objektspezifischen Hilfsfunktionen `getDauerInStundenUndMinuten` und `getTutorialsZuKategorie` sowie den Code zur Erstellung der Beispieldaten (inklusive der Arrays für die `Kategorie`- und `Tutorial`-Objekte). **Kopieren Sie nicht den Teil des Codes, welcher die Beispieldaten auf der Browser-Konsole ausgibt!**

    2. Ergänzen Sie die *Schnittstelle* des Moduls "persistence.js". Konkret sollen die Arrays mit den Beispieldaten sowie die Funktionen `getDauerInStundenUndMinuten` und `getTutorialsZuKategorie` durch andere Module genutzt werden können.

Aufgabe 2: Backend für das Suchformular

Das Suchformular auf der Seite mit der *Liste der Kategorien* (`list.html`) ist bisher lediglich mit einem Test-Server verbunden. Wir realisieren nun mit Node.js einen Server, der die Suche realisiert: Zu einem im Suchformular eingegebenen Suchbegriff soll der Server Tutorials mit passenden Namen aus den Beispieldaten heraussuchen und diese in Form einer HTML-Seite mit Suchergebnissen für die Benutzer*innen darstellen. Gehen Sie dabei wie folgt vor:

1. Server

1. Realisieren Sie den Web-Server in der Datei "v1/index.js".
2. Verwenden Sie zur Realisierung das Node.js-Modul "http" sowie unser "persistence.js"-Modul aus Aufgabe 1.
3. Sorgen Sie dafür, dass der Server unter der URL http://localhost:8844 erreichbar ist.
4. Beim Eintreffen einer HTTP-Anfrage soll sich der Server folgendermaßen verhalten:
    1. Zunächst soll der Server den im Suchformular eingegebenen Suchbegriff aus der HTTP-Anfrage lesen. Da unser Suchformular die Daten per GET sendet, finden wir den Suchbegriff im Anfrageteil (*query*) der URL. Um diesen auszulesen, benötigen Sie das Node.js-Modul "url". Binden Sie dieses wie bekannt ein:

        ```
        const url = require("url");
        ```

        Nutzen Sie folgenden Code, im den Anfrageteil der URL auszulesen (`req` ist dabei die HTTP-Anfrage, welche dem *Request-Listener* übergeben wurde):

        ```
        const queryParams = url.parse(req.url, true).query;
        ```

        Hinweis: Falls Sie unsicher sind, wie Sie mit `queryParams` weiterarbeiten können, so lassen Sie sich den Wert der Variable einfach einmal auf der Konsole ausgeben, um ihn zu inspizieren.
    2. Im zweiten Schritt soll der Server die passenden Tutorials aus den Beispieldaten unseres Persistenzmoduls ("persistence.js") lesen. Konkret sollen hier alle `Tutorial` -Objekte ermittelt werden, deren `name` -Eigenschaft den Suchbegriff *enthält*. Nutzen Sie z.B. die MDN-Dokumentation zum `String` -Objekt, um passende Funktionen für diesen Zweck zu finden. Die so ermittelten `Tutorial` -Objekte sind unsere *Suchtreffer*.
    3. Falls in Schritt 4.2 einer oder mehrere Suchtreffer ermittelt werden konnten, so soll der Server eine HTML-Seite zurückliefern, die die Suchtreffer wie in der unten

stehenden Abbildung (linke Seite) darstellt.

Nutzen Sie eine *ungeordnete Liste* zur Darstellung der Suchtreffer. In der darüber stehenden Überschrift soll der Suchbegriff wiedergegeben werden (`Tutorials mit: $SUCHBEGRIFF`). Erzeugen Sie die Inhalte der HTML-Seite *dynamisch* und *server-seitig* (d.h. nicht unter Verwendung von DOM-Manipulation o.Ä.). Verwenden Sie zu diesem Zweck, wie in den Lernmaterialien gezeigt, die *Template-Literale* von JavaScript. Die resultierende HTML-Seite soll sich in die Formatierung der restlichen Seiten unserer Web-Anwendung einfügen (CSS-Styling, Kopfbereich, Fußbereich etc.).

4. Falls in Schritt 4.2 kein Suchtreffer ermittelt werden konnte, so soll die erzeugte HTML-Seite aussehen wie in der unten stehenden Abbildung (rechte Seite) dargestellt.

Darstellung des Suchergebnisses (links: Suchtreffer gefunden, rechts: keine Suchtreffer):

2. Client

Passen Sie das Suchformular auf der Seite mit der *Liste der Kategorien* (`list.html`) so an, dass es den eingegebenen Suchbegriff an http://localhost:8844 sendet.

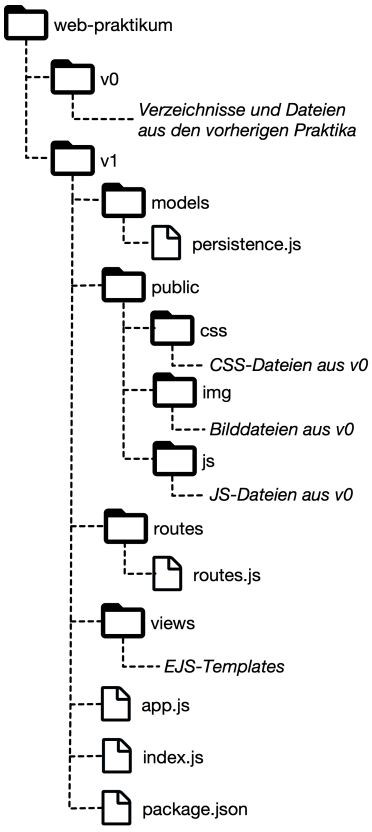
Hinweis: Zum Testen der Suche müssen Sie den neu implementierten Node.js-Server (Verzeichnis "v1") starten. Zudem sollten Sie die Webseite (Verzeichnis "v0") lokal mit Hilfe des Live-Servers von Visual Studio Code ausführen (vgl. Video "Extensions in Visual Studio Code & Live-Server" im Praktikum-Ordner in ILIAS, rechte Seite).

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 3: 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 4).
- Die Datei "routes/routes.js" definiert die Logik für das Routing der Web-Anwendung (Details siehe Aufgabe 4).
- Das Verzeichnis "views" enthält *EJS-Templates* zur dynamischen Erzeugung der Seiten unserer Anwendung (Details siehe Aufgabe 5).
- Die Datei "app.js" ist der Einstiegspunkt, über welchen sich unsere Web-Anwendung starten lässt.
- Die Datei "index.js" ist weiterhin der Startpunkt des Node-Servers für die Suche (siehe Aufgabe 2).
- 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´3).

Gehen Sie dabei wie folgt vor:

1. Kopieren Sie folgende Dateien aus dem 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. alle `.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 init
$ npm install express
$ npm install ejs
```

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

Hinweis: 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 4: 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:8020 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. Laden Sie dazu das ZIP-Archiv praktikum10-dateien.zip aus ILIAS herunter und entpacken Sie dieses. Verschieben Sie die enthaltene Datei "routes/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:8020 → Liste der Kategorien (in v0: `list.html`)
    2. http://localhost:8020/tutorials?category=X → Tutorials zu einer Kategorie (in v0: `tutorials.html`, *X* bezeichnet den Namen der Kategorie*)
    3. http://localhost:8020/tutorial?name=X → Detailseite zu einem Tutorial (in v0: `tutorial.html`, *X* bezeichnet den Namen des Tutorials*)
    4. http://localhost:8020/form → Formular zum Hinzufügen eines neuen Tutorials (in v0: `form.html`)
    5. Statische Inhalte (Bilder, CSS, client-seitiges JavaScript) sollen aus dem Verzeichnis "public" (bzw. aus darin enthaltenen Unterverzeichnissen) heraus ausgeliefert werden.
    6. 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 3).

[*] *Die durch diese Implementierung entstehenden URLs gelten eigentlich als unsicher, da im Namen z.B. Leerzeichen enthalten sein könnten. In einem Produktivsystem wäre dies kein guter Stil - wie nehmen es in diesem Fall aber als Kompromiss hin, da unsere Datenobjekte nicht über saubere IDs verfügen.*

Aufgabe 5: 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 v0 als Vorlage.

2. *Modularisierung:* Lagern Sie die Bereiche, die sich auf allen HTML-Seiten wiederholen (d.h. Kopfbereich, Fußbereich und Bereich mit Zusatzinformationen), 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 Kategorien zu den enthaltenen Tutorials und von dort aus auf die Detailseite eines Tutorials navigieren können, benötigen wir die Einträge "Tutorials der Kategorie X" und "Details zum Tutorial Y" im Navigationsbereich nicht mehr. Entfernen Sie diese entsprechend auf allen Seiten.

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 in Aufgabe 1 vorbereiteten Datenhaltung (Modul "persistence.js") beziehen.

   Hinweis: Neben den "textuellen" Daten wie z.B. Name der Kategorie, Dauer eines Tutorials etc. gilt dies insbesondere auch für die Bilder, d.h. Bild (Logo) einer Kategorie und eines Tutorials.

5. Realisieren Sie die *Detailseite zu einem Tutorial* (Route http://localhost:8020/tutorial?name=X) so, dass die zusätzlichen Informationen abhängig davon dargestellt werden, ob es sich um ein Text-Tutorial oder ein Video-Tutorial handelt:

   1. *Text-Tutorial:* Es wird ein Hyperlink zum Tutorial angezeigt.
   2. *Video-Tutorial:* Es werden folgende Informationen angezeigt:

      - Das Video: Integrieren Sie den YouTube-Embed-Code des angezeigten Tutorials hier direkt in Ihre Seite, so dass das Video eingebettet angezeigt

wird.
    - Kapitel des Videos als Tabelle

6. *Layout beibehalten:* Alle Seiten sollen grundsätzlich so aussehen und funktionieren wie in v0 (mit Ausnahme des reduzierten Navigationsbereiches). Insbesondere sollen die Layouts (Grid, Flexbox), das client-seitige JavaScript (dynamisches Formular, Burger-Menü) sowie das Backend für die Suche (Aufgabe 2) weiterhin funktionieren.

Aufgabe 6: Hinzufügen eines Tutorials

Realisieren Sie die Web-Anwendung so, dass das *Formular zum Hinzufügen eines neuen Tutorials* voll funktionstüchtig ist, d.h.:

1. Nach Ausfüllen und Absenden des Formulars sollen die Eingaben an unsere server-seitige Express-Anwendung geschickt werden.

2. Die Express-Anwendung soll aus den Eingaben ein neues `Tutorial`-Objekt erzeugen und dieses in unserer Datenhaltung (Modul "persistence.js") hinzufügen. Insbesondere soll das neue `Tutorial`-Objekt auch mit den entsprechenden Kategorien verknüpft werden.

3. Im Anschluss an das erfolgte Hinzufügen soll die BenutzerIn im Browser auf die *Liste der Kategorien* (Route http://localhost:8020) weitergeleitet werden.

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.