zirpins ...	on 29 May
..	
gta_v3	5 months ago
README.md	5 months ago

README.md

3. Aufgabe - serverseitige Anwendung erstellen

Die dritte Aufgabe beschäftigt sich mit dem **GTA-Server**. Der GTA-Server verwaltet die GeoTags und bietet dem Client Funktionen zum Abfragen und Hinzufügen dieser Daten. Dabei kann der Server das HTML-Gerüst der App dynamisch aus einem **EJS-Template** erzeugen, um darin die aktuellen GeoTags und Koordinaten einzubetten. Daneben stellt der Server **statische Dateien** zum Abruf per HTTP zur Verfügung. Dazu gehören z.B. Bilder und clientseitige Skripte.

Die Aufgabe übt die Programmierung von **Modulen**, **Klassen**, **Funktionen** und **Arrays** mit **JavaScript** ein. Zudem werden serverseitige JavaScript-Technologien - insbesondere der Aufbau von **Express Apps**, die Erstellung einfacher **Express Routen**, sowie die Verwendung von **EJS Templates** - vertieft.

3.1. Vorbereitung

Aktualisieren sie zunächst das Github Repository https://github.com/zirpins/vs1lab. Wenn Sie den git-Tipps aus Aufgabe 1 gefolgt sind, gehen sie wie folgt vor (Beispiel für Linux/Mac):

```
cd ~/git/vs1lab # wechsle in das git Verzeichnis
git checkout master # wechsle in den Hauptzweig (eigene Änderungen vorher mit 'commit
git pull # lade Aktualisierungen herunter
git checkout dev # wechsle wieder in den eigenen Branch
git merge master # übernehme mögliche Änderungen aus dem Hauptzweig (Daumen drücken,
```

Nach der Aktualisierung kann die zweite Aufgabe vorbereitet werden.

3.1.1 Vorherige Lösungen übernehmen

1. Kopieren Sie die Datei `Aufgabe1/gta_v1/public/stylesheets/style.css` aus Aufgabe 1 nach `Aufgabe3/gta_v3/public/stylesheets/style.css` .
2. Kopieren Sie die Datei `Aufgabe2/gta_v2/public/javascripts/geotagging.js` aus Aufgabe 2 nach `Aufgabe3/gta_v3/public/javascripts/geotagging.js` .

3.1.2 EJS Template aus HTML-Dokument ableiten

1. Öffnen sie `Aufgabe3/gta_v3/views/index.ejs` in ihrem **Editor**.
2. Kopieren sie den Inhalt `Aufgabe2/gta_v2/public/index.html` in die EJS-Datei und **ersetzen** sie die **Beispieleinträge der Discovery-Liste** mit den folgenden Zeilen:

```
<% if (taglist !== undefined) taglist.forEach(function(gtag) { %>
    <li><%= gtag.name %> ( <%= gtag.latitude %>,<%= gtag.longitude %>) <%= gtag.h
<% }); %>
```

3.1.3 Node-js vorbereiten

An dieser Stelle benötigen sie eine Node.js Umgebung. Falls sie die empfohlene Variante mit **Visual Studio Code** und **Remote Containers** Erweiterung nutzen, öffnen sie das `vs1lab` -Verzeichnis in einem Node.js-Container (siehe Laboranleitung). Alternativ kann Node.js lokal installiert werden.

Mit gegebener Node.js Umgebung gehen sie wie folgt vor:

1. Führen sie `npm install` im Verzeichnis `Aufgabe3/gta_v3/` aus, um die nötigen Module zu laden und zu installieren.
2. Um die App später zu starten, führen sie `npm start` im Verzeichnis `Aufgabe3/gta_v3/` aus und öffnen sie http://localhost:3000 im **Browser**. Am Anfang funktioniert das aber noch nicht, da noch einige Änderungen gemacht werden müssen.

3.2. Teilaufgaben

Die Aufgabe besteht nun in der Entwicklung der Serverskripte. Die Clientseite (Browserskripte) muss nur leicht angepasst werden.

3.2.1 Implementierung des Servers

Der Server besteht in dieser Aufgabe aus mehreren **Express.js** Serverskripten zur Verarbeitung von HTTP-Requests und einem **EJS Template** zur dynamischen Erzeugung von neuen HTML-Seitendarstellungen.

Die Serverimplementierung hat die folgende Struktur:

- `bin` (Ordner für ausführbare Skripte)
  - `www` (Startskript des Servers)
- `models` (Ordner für JavaScript-Module zur anwendungsspezifischen Datenhaltung und -verarbeitung)
  - `geotag.js` (JavaScript Modul für die GeoTag Klasse)
  - `geotag-store.js` (JavaScript Modul für die Speicherung von GeoTag Objekten)
  - `geotag-examples.js` (JavaScript Modul für die Erstellung von GeoTag-Beispieldaten)
- `public` (Ordner für statische Inhalte)
  - `images` (Ordner für Bilder der Webseite)
  - `javascripts` (Ordner für clientseitige JavaScript-Module)
    - `geotagging.js` (JavaScript Modul zum Start der clientseitigen Verarbeitung aus Aufgabe 2)
    - `location-helper.js` (JavaScript Modul zur Auslagerung der 'LocationHelper'-Klasse aus Aufgabe 2)
    - `map-manager.js` (JavaScript Modul zur Auslagerung der 'MapManager'-Klasse aus Aufgabe 2)
  - `stylesheets` (Ordner für CSS-Dateien)
  - `favicon.ico` (Icon-Datei als Symbol der Webseite)
- `routes` (Ordner für JavaScript Module zur Definition von Routen)
  - `index.js` (Zentrales JavaScript Modul mit allen Routendefinitionen des Servers)
- `views` (Ordner für EJS-Templates)
  - `index.ejs` (Zentrale Template-Datei zum Rendern der App)
  - `error.ejs` (Template-Datei zur Ausgabe von Fehlermeldungen)
- `app.js` (Zentrales JavaScript Modul zur Konfiguration von Express)
- `package.json` (Konfiguration von NPM-Paket und Abhängigkeiten)

3.2.1.a Serverskript fertigstellen

Die erste (und umfangreichste) Teilaufgabe besteht in der Vervollständigung einzelner Teile der Server-Implementierung. Die Anforderungen stehen jeweils als Kommentare im Sourcecode. Hier sind auch Hinweise auf relevante Stellen in der Express Dokumentation enthalten. Im einzelnen gibt es folgende Teilaufgaben:

- Backend Funktionen zur Verwaltung von GeoTags
  - Eine **Klasse für GeoTag Objekte** erstellen (`./model/geotag.js`)
  - Eine Klasse für die **Speicherung und Suche von GeoTag Objekten** erstellen (`./model/geotag-store.js`)
  - Die **Beispieldaten für GeoTag Objekte** einlesen (`./model/geotag-examples.js`)
- Server Konfiguration und Routen für die wichtigsten Endpunkte
  - **Statische Dateien** bereitstellen (`./app.js`)
  - Die **1. Route** `/` zur **Erzeugung der Einstiegsseite** ist vorgegeben. (`./routes/index.js`)
    - Hier sieht man, wie mit EJS eine HTML-Seite erzeugt wird.
  - Eine **2. Route** `/tagging` zur **Speicherung von GeoTags** erstellen. (`./routes/index.js`)
  - Eine **3. Route** `/discovery` zur **Abfrage von GeoTags** erstellen. (`./routes/index.js`)

3.2.1.b Server Template erweitern

Die Routen im Serverskript verwenden das EJS-Template `./views/index.ejs` , um die nächste Sicht der Browser-GUI als HTML-Dokument zu erstellen. EJS-Direktiven zur Füllung der Ergebnisliste haben wir darin schon vorbereitet (siehe 3.1.2).

Aufgabe: Ergänzen sie nun noch EJS-Direktiven, um in die entsprechenden `input` -Elemente in den Tagging- und Discovery-Formularen `value` -Attribute für die aktuellen Koordinatenwerte einzusetzen. Die Koordinatenwerte müssen sie dazu aus dem Skript an das Template übergeben (Ab der 2. Anfrage liefert der Client seine aktuellen Koordinaten immer als Formularfelder mit).

3.2.2 Anpassung des Clients

Wir wollen nun die Client-Implementierung noch etwas optimieren: Die Abfrage der GeoLocation API (jedes Mal nach dem Laden einer Seite) erzeugt eine erhebliche Latenz und stört damit bei wiederholtem Aufruf den Interaktionsfluss. Wir wollen daher die einmal abgerufenen Koordinaten wiederverwenden.

Die Lösung haben wir in der letzten Teilaufgabe schon vorbereitet: Der Server schreibt die Koordinaten, die er beim Aufruf aus den Formularfeldern ausliest, wieder zurück, so dass diese beim nächsten Seitenaufbau im Client schon eingetragen sind. Wir brauchen also in der `updateLocation` -Funktion nur noch zu testen, ob schon Koordinaten in den Formularfeldern stehen und nur wenn dies *nicht* der Fall ist die GeoLocation API aufrufen.

Aufgabe a: Ändern sie zunächst das Skript `./public/javascripts/geotagging.js` so, dass sie die `MapManager` und `LocationHelper` Klassen in eigene Skripte auslagern (diese sind im entsprechenden Ordner schon vorhanden). Laden sie dazu in der HTML Seite alle Skripte. Die ausgelagerten Klassen können dann aus `geotagging.js` entfernt werden.

Aufgabe b: Erweitern Sie nun die `updateLocation` -Funktion im Client-Skript `./public/javascripts/geotagging.js` . Lesen sie dort die geeigneten Formularfelder im DOM aus und testen sie, ob schon Koordinaten eingetragen sind. Rufen sie die Methode `LocationHelper.findLocation()` nur noch dann auf, wenn es die Situation erfordert.

3.2.3 `data-*` -Attribute und Map Marker

Die `getMapUrl` -Methode der `MapManager` -Klasse besitzt einen Parameter, dem man einen Array von GeoTag Objekten übergeben kann. Wenn dieser vorliegt, werden an den Positionen der GeoTags Marker in die Karte eingetragen, d.h. die Tags werden auf der Karte sichtbar.

Das Problem ist nun, im Client Skript das Array mit GeoTag Objekten verfügbar zu machen. Diese Information liegt im Server schon vor. Wie aber kommen die Daten zum Client? In der nächsten Aufgabe werden wir zu diesem Zweck AJAX-Abfragen einführen. An dieser Stelle wollen wir noch eine andere Variante kennenlernen:

Der Server kann das Array mit GeoTag Objekten als `data-*` -Attribut einem geeigneten Element beifügen. Erweitern sie dazu das EJS-Template derart, dass es dem `img` -Element der Karte ein `data-tags` -Attribut beifügt. In das Attribut schreiben sie das Array mit GeoTag Objekten als JSON-String. Für ein Array `taglist` erzeugt der Aufruf `JSON.stringify(taglist)` den JSON-String.

Auf der Clientseite können sie dann das Attribut aus dem DOM lesen und den string wieder in ein Array Objekt umwandeln. Für einen JSON-String `taglist_json` erzeugt der Aufruf `JSON.parse(taglist_json)` das korrespondierende JavaScript Array Objekt. Dieses Array Objekt können sie der `getMapUrl` -Methode als Parameter übergeben.

Aufgabe: Erweitern Sie den Aufruf der `getMapUrl` -Methode aus der `updateLocation` -Funktion und übergeben sie neben der aktuellen Position des Clients zusätzlich einen Array von GeoTag Objekten des aktuellen Suchergebnisses im Discovery Widget. Auf der Karte sollen dann für alle Elemente der Ergebnisliste entsprechende Marker angezeigt werden.

Checkliste

Zur Übersicht folgen noch mal alle Anforderungen in kompakter Form als Checkliste.

1. Teilaufgabe: Server-Implementierung

- [ ] **Skripte** für Backend Funktionen fertigstellen

- [ ] **Klasse für GeoTag Objekte** erstellen (`./model/geotag.js`)
- [ ] **Klasse zur GeoTag Speicherung** erstellen (`./model/geotag-store.js`)
  - [ ] Privaten Array nutzen
  - [ ] Methoden `addGeoTag` , `removeGeoTag` , `getNearbyGeoTags` , `searchNearbyGeoTags` realisieren
- [ ] **Beispieldaten für GeoTag Objekte** einlesen (`./model/geotag-examples.js`)

- [ ] **Server Konfiguration und Routen** fertigstellen
  - [ ] **Statische Dateien** bereitstellen (`./app.js`)
  - [ ] Route `/tagging` zur **GeoTags Speicherung** erstellen. (`./routes/index.js`)
  - [ ] Route `/discovery` zur **GeoTag Abfrage** erstellen. (`./routes/index.js`)

- [ ] Server **Template** erweitern (`./views/index.ejs`)
  - [ ] Koordinaten in Formulare eintragen

2. Teilaufgabe: Client-Anpassung

- [ ] Teil A: `MapManager` und `LocationHelper` **Klassen in eigene Skripte auslagern** (`./public/javascripts/geotagging.js`)
- [ ] Teil B: `updateLocation` -Funktion **optimieren**
  - [ ] Auslesen von Formularfeldern mit Koordinaten
  - [ ] Bedingte Ausführung von `LocationHelper.findLocation()`

3. Teilaufgabe: Karten-Erweiterung

- [ ] EJS-Template mit `data-*` -Attribut erweitern
  - [ ] Im `img` -Element ein `data-tags` -Attribut erstellen und **Array mit GeoTag Objekten als JSON-String** einfügen
- [ ] Aufruf von `getMapUrl` aus `updateLocation` erweitern
  - [ ] Array mit GeoTag Objekten übergeben und **Map Marker anzeigen**
