zirpins ...	on 2 Oct 2021
..	
gta_v1/public	13 months ago
README.md	13 months ago

1. Aufgabe - Web App strukturieren und gestalten

In der ersten Aufgabe soll die **Web-Oberfläche** der GT-App entstehen. Dazu gehört ein **HTML-Dokument** zur Strukturierung der Inhalte und ein **CSS-Stylesheet** für Layout und Gestaltung.

Die Aufgabe vertieft den Umgang mit **HTML** und besonders mit **HTML-Formularen** und zeigt deren Zusammenhang mit **CSS3** durch verschiedene **Selektoren** sowie die Funktionsweise von **Flexbox**.

Vorbereitung

Für die Aufgabe existieren schon Templates auf **GitHub**:

- https://github.com/zirpins/vs1lab.

Laden sie das ganze Repository von GitHub herunter. Wir zeigen hier durchgängig eine mögliche Methode zur Arbeit mit Git (Beispiel für Linux/Mac):

```
mkdir ~/git # lege Verzeichnis git im Honeverzeichnis an
cd ~/git # wechsle in das neue Verzeichnis
git clone https://github.com/zirpins/vs1lab.git # git befehl zum Herunterladen des Re
```

Wenn sie das Repository wie gezeigt 'gecloned' haben, können sie später Aktualisierungen herunterladen. Sie sollten dazu Ihre Lösungen nicht direkt in das kopierte Repository schreiben (Tipps zu Zweigen bzw. 'branches' folgen unten), Aktualisierungen laden sie dann wie folgt herunter (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
```

Die Dateien der Aufgabe befinden sich nun im Ordner `~/git/vs1lab/Aufgabe1/` .

Teil A - Struktur einer Webanwendung mit HTML5 erstellen

In Teil A soll die Struktur der Oberfläche als HTML-Seite entstehen. Dafür gibt es folgende Ziele:

- Die App soll alle Funktionen auf *einer* Seite kombinieren (**Single Page App**) und sich in **Header**-, **Main**- und **Footer-Bereich** gliedern. Im Hauptteil soll ein **Tagging Widget** und ein **Discovery Widget** enthalten sein.
- Das Tagging Widget fragt über ein **Formular** die Attribute `latitude` , `longitude` , `name` und `hashtag` eines neuen GeoTags ab.
- Das Discovery Widget zeigt in einer **Tabelle** und einer **Karte** die GeoTags der Umgebung an. Über ein **Formular** kann ein `searchterm` (Suchbegriff) eingegeben werden, um die GeoTags zu filtern. Damit auch in der Nähe der aktuellen Position gesucht werden kann, soll das Formular die Koordinaten (`latitude` , `longitude`) als **versteckte Eingaben** beinhalten.

A.1. Vorbereitung

Erstellen sie zunächst einen neuen Zweig im Repository (Beispiel für Linux/Mac):

```
cd ~/git/vs1lab # wechsle in das git Verzeichnis
git branch dev  # erzeuge Branch 'dev'
git checkout dev  # wechsle in den neuen branch
```

Nun können sie in ihrem Arbeitszweig unabhängig vom Hauptzweig arbeiten. Ein Template der Seite liegt im Ordner `Aufgabe1` des Repositories. Öffnen sie `Aufgabe1/gta_v1/public/index.html` in ihrem **Editor** und in Ihrem **Web Browser**. Z.B. so (Beispiel für Linux/Mac mit vscode und chrome):

```
cd ~/git/vs1lab/Aufgabe1/gta_v1/public # wechsle in das Arbeitsverzeichnis
code ./index.html # öffne vscode Editor mit Webseite
chromium ./index.html # öffne Chromium Web Browser mit Webseite (Linux)
open -a "Google Chrome" ./index.html # öffne Chrome Web Browser mit Webseite (Mac)
```

Wenn sie in den folgenden Aufgaben einen guten Zwischenstand erreicht haben oder fertig sind, sollten sie dies im Repository als sogenannten *Commit* festhalten, der lokal auf ihrem Rechner verwaltet wird (Beispiel für Linux/Mac):

```
git add ~/git/vs1lab/Aufgabe1  # merke Änderungen für Aufgabe 1 für Commit vor
git commit -m 'first solution for web page' # führe commit durch mit Beispielkommenta
```

Commits sind feste Zwischenstände im Repository, zu denen sie immer wieder zurückkehren können. Geht in der Folge etwas schief, können sie alle Dateien leicht wieder zurücksetzen. Nach dem Commit können sie auch gefahrlos in einen anderen Zweig (z.B. 'master') wechseln.

Pro-Tipp: *Der Commit befindet sich in Ihrem 'dev'-Branch und ändert den 'master'-Branch nicht. Der 'master'-Branch sollte auch nicht verändert werden, damit Sie später ggf. Aktualisierungen aus dem ursprünglichen Repository nachträglich übernehmen können. Bitte schicken Sie auch niemals Commits an das ursprünglichen Repository zurück (git push), sondern arbeiten Sie immer lokal auf Ihrer eigenen Kopie. Falls Sie für die Gruppenarbeit ein geteiltes Repository benötigen, erstellen Sie zunächst eine Kopie davon auf GitHub (als 'Fork') und clonen sie dann den Fork. Dabei helfen wir Ihnen gerne.*

A.2. Teilaufgaben

Formularstruktur

Im Template fehlen noch die Formulare. Diese sollen nun erstellt werden.

- Ergänzen sie geeignete `Input` Elemente im Tagging-Formular und im Discovery-Formular.
- Vergeben sie eindeutige `id` Attribute für die Felder, um sie später in JavaScript finden zu können.
- Deklarieren sie für alle Felder jeweils ein `label`.
- Verwenden sie `fieldset` und `legend` zur Begrenzung des Formulars.
- Deklarieren sie für jedes Formular ein Element zum Absenden der Daten.
- Fügen sie als Hilfe für die Benutzer *Beispiel Platzhalter* in die Felder ein.

Formularvalidierung

Für die Formulare soll nun noch eine Validierung definiert werden. Diese Validierung soll **nur HTML5-Attribute** verwenden .

Setzen sie folgende Regeln um:

- Latitude und Longitude erhalten feste Werte und können nicht verändert werden.
- Im Tagging-Formular muss ein Name angegeben werden, der Hashtag ist optional.
- Namen dürfen 10 Buchstaben lang sein.
- Hashtags müssen mit `#` beginnen und dürfen dann noch 10 Buchstaben haben.

Teil B - Webanwendung mit CSS3 gestalten

Teil B dreht sich um die Gestaltung der GT-App und dabei insbesondere um das Layout der GUI. Hierfür verwenden wir einen einfachen Ansatz ausschließlich auf Basis von CSS3.

B.1. Vorbereitung

Templates für Seite und Stylesheet liegen im Ordner `Aufgabe1` des Repositories. Öffnen sie `Aufgabe1/gta_v1/public/index.html` in ihrem **Editor** und in Ihrem **Web Browser**. Öffnen sie zusätzlich die Datei `Aufgabe1/gta_v1/public/stylesheets/style.css` in ihrem **Editor**. Dies geht z.B. so (Beispiel für Linux/Mac):

```
cd ~/git/vs1lab/Aufgabe1/gta_v1/public # wechsle in das Arbeitsverzeichnis
code ./index.html ./stylesheets/style.css # öffne Atom Editor im Wurzelverzeichnis de
chromium ./index.html # öffne Chromium Web Browser mit Webseite (Linux)
open -a "Google Chrome" ./index.html # öffne Chrome Web Browser mit Webseite (Mac)
```

B.2 Aufgabe (Responsive Grid Layout)

Die Komponenten einer Web-GUI werden oft als **Grid Layout** angeordnet. Hierbei werden mehrere Reihen und Spalten vorgegeben, die über den Screen verteilt sind und die Position für Elementen der Webseite vorgeben.

Grid Layout mit Flexbox

Eine moderne Umsetzung eines solchen Grid Layouts funktioniert mit einer **Flexbox**. Hierbei fügt man verschiedene 'Child'-Elemente innerhalb eines Containers (der Flexbox) hinzu, welche nun dynamisch und sehr spezifisch angeordnet werden können. Dies kann entweder über "normale" CSS-Eigenschaften (width, height, margin, padding, etc...) definiert werden oder mit den extra dafür vorgesehen Flex-Eigenschaften. Eine gute Übersicht dazu finden sie unter https://css-tricks.com/snippets/css/a-guide-to-flexbox/.

Wir haben auf diese Weise schon die Deklaration einfacher Grid Layouts in der Vorlesung behandelt, die sich im Sinne eines Responsive Designs an die Größe des Browserfensters anpassen. In der Aufgabe werden wir den Ansatz für verschachtelte Grids mit unterschiedlicher Spaltenbreite erweitern.

Responsives Verhalten mit Media Queries

Um die Seite auf kleineren Endgeräten zu simulieren können sie einfach ihr Browser-Fenster minimieren und entsprechend klein/groß ziehen. Wenn sie eine gewisse Breite unterschreiten, werden sie feststellen, dass die einzelnen Spalten des Grids zu schmal werden.

Auf mobilen Geräten wäre es also besser, die Spalten wären untereinander positioniert. Hier können sie - wie in der Vorlesung behandelt - **Media Queries** benutzen.

Messen sie eine geeignet Breite aus, ab wann Spalten untereinander klappen sollen (in Chrome öffnen sie dazu die Entwicklertools, markieren den html-Tag und dann sehen sie in der rechten oberen Ecke des Browsers die Viewport-Abmessungen; ein geeigneter Wert könnte z.B. 850px sein). Nun müssen sie innerhalb der Media Queries lediglich definieren, dass Spalten untereinander positioniert werden und eine Breite von 100% haben sollen. Eine Ausführliche Erläuterung zur Verwendung von media queries finden sie unter

README.md

nicht der "Simulation" entspricht. Das liegt an der Standart-Viewport Konfiguration, welche sie ebenfalls ändern müssen. Das geht aber ganz einfach mit einem HTML-Tag, den sie im HEAD unterbringen müssen: `<meta name="viewport" content="width=device-width, initial-scale=1.0">`

Aufgabenstellung

Gestalten sie das Layout der GT-App als **verschachteltes zweispaltiges Grid**: Die erste Spalte enthält das Tagging-Widget mit dem Tagging-Formular. Die zweite Spalte enthält das Discovery-Widget. Dies beginnt mit dem Filter Formular. Darunter sollen zwei weitere verschachtelte Spalten für die Ergebnisliste und für die Karte folgen.

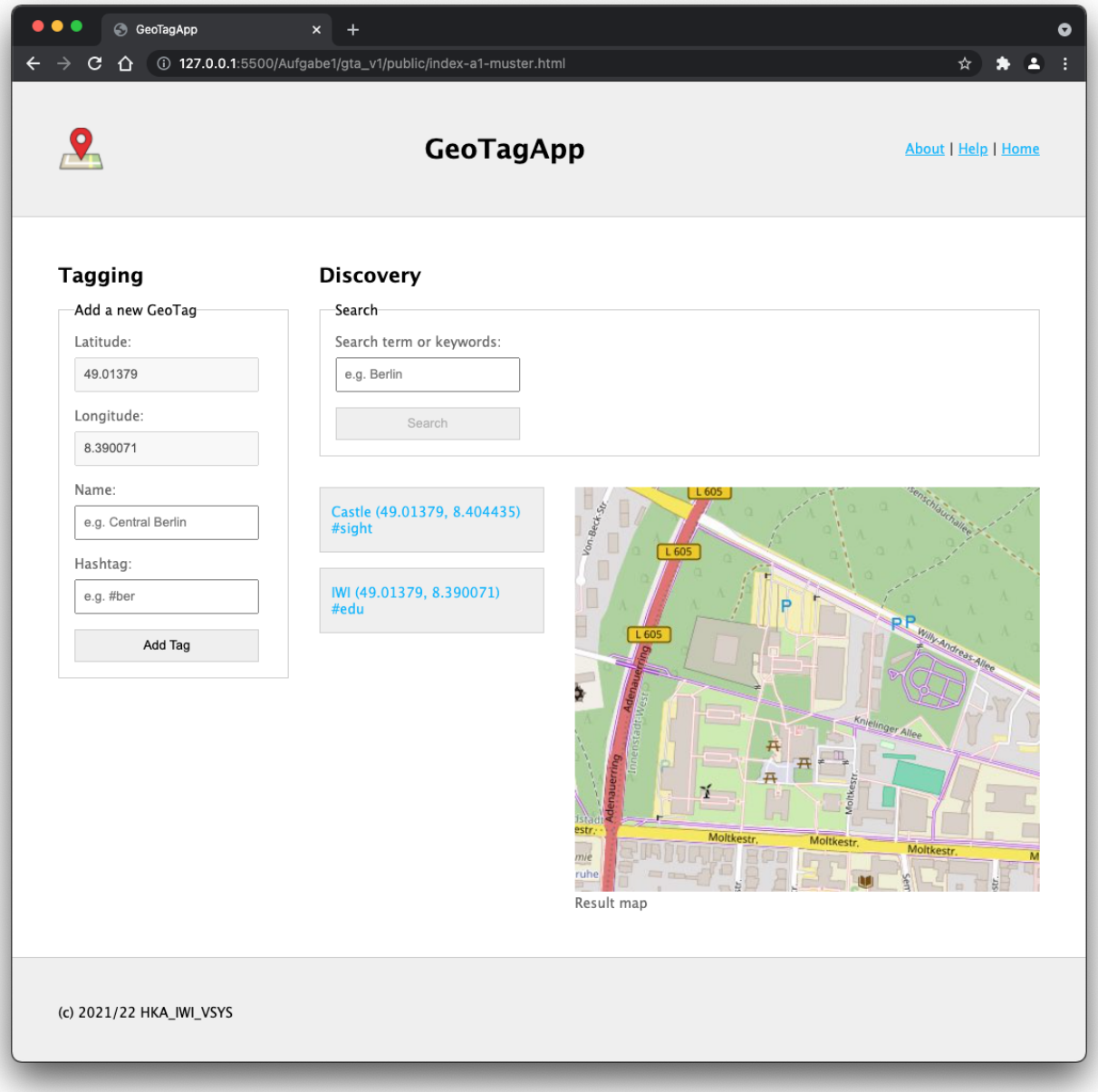
Bezüglich **Responsive Design** soll die Breite der Spalten sich relativ zur verfügbaren Bildschirmbreite bei gleichbleibendem Verhältnis der Spalten untereinander anpassen. Ab einer minimalen Bildschirmbreite sollen alle Spalten "untereinander klappen". Es gibt dann also nur noch eine Spalte mit vielen Reihen.

Setzen sie weitere **gestalterische Ziele** um:

- Farbig abgesetzte Header- und Footer-Bereiche
- Vertikale Anordnung der Eingaben im Tagging Formular
- Größere Boxen für die Eingaben aller Formulare

- Discovery Ergebnisliste mit farbigen Boxen

Das Ergebnis könnte wie folgt aussehen:



Allgemeine Hinweise

- Nutzen sie das `display` Property um die Positionierung von Elementen zu beeinflussen (in CSS `display:block` statt in HTML `<br>`)
- Verwenden sie verschiedene Arten von Selektoren (`element` , `id` , `class`).

Implementierung des Grid Layouts

- Bei Grids mit unterschiedlich breiten Spalten wird gerne eine Aufteilung der Screen-Breite in 12 gleiche Teile vorgenommen. Spalten werden dann nach der Zahl ihrer Teile ausgewählt, so dass am Ende wieder 12 Teile resultieren (z.B. 3 Spalten mit 2, 4 und 6 Teilen).

- Als Vorlage zur Implementierung so eines Layouts finden Sie im Template schon Regelmengen für verschiedene Klassen `.row` und `.col-x`. Mithilfe dieser Klassen sollen sie HTML-Elemente als Zeilen oder Spalten deklarieren, so dass sich das vorgegebene Grid Layout ergibt.
- Achten sie auf eine genaue Ausrichtung der Spalten. Die Summe der Teile gewählter Spalten sollte in jeder Reihe 12 ergeben.

Implementierung des Responsive Design

- Deklarieren sie eine **Media Query**, um bei geringer Breite Spalten untereinander zu klappen.

Checkliste

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

Teil A: Formulare fertigstellen

- [x] `Input` Elemente im Tagging- und Discovery-Formular ergänzen
  - [x] Felder im Tagging Formular: `latitude`, `longitude`, `name` und `hashtag`
  - [x] Felder im Discovery Formular: `searchterm` sowie `latitude` und `longitude` als versteckte Eingaben
  - [x] Eindeutige `id` Attribute für die Felder
  - [x] Für alle Felder jeweils ein `label`
  - [x] Platzhalter für alle Felder
- [x] `fieldset` und `legend` zur Begrenzung des Formulars
- [x] Für jedes Formular ein Element zum Absenden
- [x] Formular-Validierung
  - [x] Latitude und Longitude mit festen Werten
  - [x] Im Tagging-Formular: Name obligatorisch, Hashtag optional
  - [x] Namen: max. 10 Buchstaben lang
  - [x] Hashtags: beginnen mit `#`, max. 10 Buchstaben

Teil B: Seite mit CSS3 gestalten

- [x] Layout als verschachteltes zweispaltiges Grid realisieren
  - [x] Klassen `.row` und `.col-x` verwenden
- [x] Seitengestaltung
  - [x] Farbig abgesetzte Header- und Footer-Bereiche
  - [x] Vertikale Anordnung der Eingaben im Tagging Formular
  - [x] Größere Boxen für die Eingaben aller Formulare
  - [x] Discovery Ergebnisliste mit farbigen Boxen

- [x] Responsives Verhalten
  - [x] Spalten relativ zur Bildschirmbreite anpassen
  - [x] Ab min. Bildschirmbreite Spalten untereinander klappen
    - [x] Mit Media Query realisieren