technische Doku Haecksenkarte V2.0

Hosting

- statisch auf map.haecksen.org
  - gesourct im iFrame auf https://www.haecksen.org/lokale-gruppen/
- Content wird per Cronjob von Gitlab geholt (maintained by Drakulix)
- ssh Zugänge haben: TODO
- Server: TODO

Code

- Repository auf GitLab:
  - Gruppe: https://gitlab.com/haeckmap
  - HaeckMap: https://gitlab.com/haeckmap/HaeckMap_v2.0
  - Tickets: https://gitlab.com/groups/haeckmap/-/issues

Design

- js only, weil statische Webseite
  - Frameworks für Map:
    - Leaflet (Upgrade auf V1.9.4 für Koordinaten in Karte auswählen)
      - TODO: ggf bessere Version finden, diese scheint etwas unperformant zu sein
    - Topojson (für LK Layer)
    - geolib.js (zur exakten Bestimmung des Landkreises von einem Paar Koordinaten)
    - leaflet-fullscreen (Captain obvious)
  - ansonsten (bis jetzt) nur pure Vanilla js und css
- intensive Verwendung von template literals zum HTML bauen -> das ist hier so sinnvoll, weil perspektivisch auf eine App migriert werden soll und template literals sich leichter in echte Templates migrieren lassen

Repository Struktur

- css: captain obvious
- data:
  - external (Daten aus externen Quellen)
    - topojson (Landkreise mit Außengrenzen)
  - generated (Daten, die in der App in der DB stehen)
    - counties (Länder, Bundesstaaten und Kreise mit Schwerpunkten)
    - group_data
    - translations
  - field_mapping: Mapping von Daten auf UI
  - constants: Captain obvious
- dist: Platzhalter für minified js/css
- img: captain obvious
- js:
  - main: (View-übergreifende) Funktionen getriggert durch User und View-übergreifende Funktionen
    - functions: zB Sprachen wechseln, user-defined styles finden, subscribe RSS
    - listener: alle Funktionen, die an Buttons usw hängen
  - model: Klassen für Datenstrukturen und parsen der Daten
    - county_map: Modell der Landflächen, benötigt zum Bestimmen eines Standorts anhand von Koordinaten
    - local_group: Modell eines Eintrags in den Gruppen/Space-Daten
    - parse_data: Logik zum Einlesen/Konvertieren der Daten
  - template: Funktionen, die das HTML generieren (template literals)
    - hier gibt es für jede Fläche in der UI 1 file, und innerhalb deren 1-mehrere Funktionen, die Teile dieser UI erzeugen
  - views: Aufbau und (View-spezifische) Funktionen für Views
    - hier gibt es auch für jede Fläche in der UI 1 file, und darin sind die Funktionen dieser UI-Bestandteile hinterlegt, zB die Suchfunktion in der Liste
  - main.js: Script, das ganz am Ende geladen wird und nur die Initialisierung der Verarbeitung auslöst
- dev: Templates/snippets aus denen die Website in der Pipeline gebaut wird, und die externen js Importe für den offline Gebrauch Dev/Test
- generate_files.py: generiert in der Pipeline die index(_xx).html files, die RSS feed files, und die js Dateien mit den generierten Daten
- download_newest_artifacts.sh: Skript zum Download des fertigen package (zur Verwendung im cron Job und für Testzwecke)
- html/index_dev(_offline).html: HTML file für Dev/Test Zwecke

Details

- data/generated
  - das bedeutet, dass diese Daten aus einer “internen” im Sinne von Haecksen-eigenen Quelle kommen, aber nicht manuell erzeugt/gepflegt werden

    - der Einfachheit halber können bis V3 auch manuelle Änderungen vorgenommen werden, die müssen dann nur ebenfalls in der DB eingepflegt werden
- data/generated/counties
    - Diese enthalten nur den Schwerpunkt einer Fläche, keine Kette von Koordinaten, mit denen man eine Grenze in die Karte zeichnen könnte. Das ist sozusagen eine lightweight-Variante, um eine Detektierung einer Fläche zu einem Paar Koordinaten feststellen zu können. Ohne diese zusätzlichen Daten, würde der Algorithmus zur Bestimmung der Daten zu Land/Bundesstaat/Landkreis nur die nächstgelegene Fläche ausgeben, auch wenn diese fernab der gesuchten Koordinaten liegt. Demo: einfach mal in der fullFeatured Version über Amerika oder Afrika Add new anklicken, dann werden die Felder vermutlich mit Island bzw Zypern oder so ausgefüllt (Europa ist halbwegs abgedeckt)
- field_mapping
    - im field_mapping werden die Input-Daten mit Attributen zur internen Verarbeitung assoziiert, zB:
        - welcher Datentyp sie sind, damit im Formular das richtige Element dargestellt wird (zB DropDown, Radio Buttons, Checkboxen oder Text Input)
        - für alle Datentypen mit fixen Optionen stehen hier auch die gültigen Optionen (wenn die nirgendwo stünden, könnte man sie ja auch im Formular nicht zum Anklicken anbieten)
        - für welche “types” sie gültig sind (zB ist das Picnic Date nur für Picnics gültig)
        - wo sie angezeigt werden sollen (Anzeige in Map/Liste ja/nein, Anzeige im Formular ja/nein)
        - mandatory: Pflichtfeld ja/nein, editable: editierbar ja/nein, calculated: Wert wird programmatisch bestimmt (zB “Standort”, das aus den Einzelangaben zusammengesetzt ist)
        - privacy: das Feld kann im Formular als privat markiert werden, dann wird in der Mail unter metadata -> privacy dieses Feld mit aufgelistet, das Import-Tool berücksichtigt diese Angabe dann und setzt das Flag private in der DB auf true - diese Daten werden dann nicht exportiert / von V3 nicht angezeigt
            - Bitte beachten: momentan ist dieses Flag für alle Einträge deaktiviert
    - das field_mapping ist die wichtigste Stellschraube zur Anpassung der UI - hier werden u.a. die Reihenfolge der Darstellung der Daten für die UI, ihre Gruppierung, ein-/ausblenden, Eigenschaften von Feldern usw gesteuert
    - die allermeisten Anpassungen an der Darstellung der Attribute lassen sich allein damit lösen
- js/model/parse_data
    - da die Daten jetzt nicht nur 1x in der Map gebraucht werden sondern auch 1x in der List view und 1x im Formular Änderung melden, werden sie nicht mehrmals gelesen sondern einmal am Anfang und als Objekte in eine Liste gespeichert, siehe local_group...
- js/model/local_group
    - ...diese Objekte ("LocalGroup"s) haben alle wichtige Funktionalität direkt an sich, wie zB das Aggregieren der Addressdaten, das Ersetzen der Links, das Erzeugen eines json snippets von sich selbst, das Bestimmen von Koordinaten/Standortdaten

passend zueinander und natürlich alle Daten (“Attribute”)
    - local_group “harmoniert” die Daten von in- und output mit der UI - ein Record, der von local_group exportiert wird kann exakt so auch importiert werden und würde immer exakt gleich in der UI aussehen
    - das hilft primär beim Erzeugen des outputs für die E-Mail, hat aber zB auch das Gimmick “Preview als Draft in der Map” abgeworfen

Daten

- momentan als json gespeichert, Format:

```
{
            "type": "Feature",
            "properties": {
                "fid": 1,
                "typ": "space",
                "count_fictive": 18,
                "space_name": "name"
                // more attributes
            },
            "geometry": {
                "type": "Point",
                "coordinates": [
                    9.0,
                    52.0,
                ],
            }
        }
```

Revision #3
Created 2025-01-28 15:50:10 UTC by zombielypse
Updated 2025-01-29 05:42:38 UTC by zombielypse