**Begriffdefinitionen:**

**Opentopodata.org**: Webseite, die eine API anbietet, um Höhen abzufragen. Diese API ist open-source und kann beliebig auf einem Server reproduziert werden. Höhen werden meistens durch .tif oder .hgt Dateien bereitgestellt. Diese sind als Rasterdaten gespeichert. Soll heissen, jeder Pixel besteht aus einem Grau-Wert, welcher dann durch diese API abgefragt werden kann. Auch diese Karten kommen in Tiles und werden je nach Koordinaten-Aufruf geladen oder auch nicht. Eine sehr bekannte API ist die Google Elevation API, doch diese kostet pro Aufruf und stimmt somit nicht mit unseren Kriterien überein.

**Tiles**: Ein sehr grosses Bild wird in kleine Teile aufgeteilt. Dies ist gedacht, damit der Client nicht das ganze File runterladen muss, sondern nur genau diesen Abschnitt welcher er braucht. Dies macht ein schnelles und dynamisches Ansehen der Webseite möglich. Diese Tiles werden normalerweise direkt vom Ersteller bereitgestellt, so dass der Client der eigenen Webseite die Tiles direkt von ihm anfordert. Dies spart Speicherplatz und Bandbreite auf dem eigenen/ auch GitHub Server.

**Leafletjs**: Open-source Service welcher ein Ansehen von Karten-Tiles ermöglicht. Zudem ermöglicht er ein Overlay, bei welchem Marker gesetzt, Linien gezeichnet werden und noch vieles mehr gemacht werden kann. Unsere Tiles sind von esri (<https://www.esri.com/de-de/arcgis/about-arcgis/overview>) bereitgestellt und sind die genausten und schnellsten Kartenfragmente, die für die Öffentlichkeit verfügbar gemacht wurden. Jedoch können bei solchen Tiles keine Höhen abgefragt werden. Nicht einmal Leafletjs unterstützt diese Funktionen. Deshalb wird ein Dienst wie Opentopodata benötigt, um diese Höhenkarten auslesen zu können.

**GitHub Pages:** Dies ist ein Dienst von GitHub, welcher ein Repository zu einer Webseite macht, welche über *nutzernamen*.github.io erreichbar ist. Dieser Service ist gratis und ist speziell für Entwickler gedacht. Durch die Schnelligkeit von GitHub kann die Webseite innert Sekunden erneuert werden, obwohl man offline arbeitet. Genau aus diesem Grund habe ich GitHub Pages gewählt. Ich hätte noch eine andere Website, doch dies braucht jedes Mal Stunden, bis der neue Code online ist.

**European Petroleum Survey Group Geodesy (EPSG):** Dies war eine Arbeitsgruppe der europäischen Öl- und Gaserkundungsunternehmen. Sie definierte die EPSG-Kodierungen, welche genutzt werden, um Koordinatenreferenzsysteme zu unterscheiden. Die verschiedenen Höhenkarten sind also in verschiedenen Koordinatenreferenzsysteme verfasst. Glücklicherweise kann die opentopodata API mit verschiedenen Systemen rechnen. (<https://de.wikipedia.org/wiki/European_Petroleum_Survey_Group_Geodesy> , abgefragt am 02.03.2020)

**Same-Origin-Policy (SOP):** Die Same-Origin-Policy (SOP) blockt Anfragen von Skriptsprachen wie Javascript auf Ressourcen, die von einer anderen Webseite oder Speicherplatz stammen. Dies bietet ein Schutz des Clients. (<https://de.wikipedia.org/wiki/Same-Origin-Policy> , abgefragt am 02.03.2020)

**Cross-Origin Resource Sharing (CORS):** Cross-Origin Resource Sharing (CORS) wurde entwickelt, um die Limiten der Same-Origin-Policy (SOP) zu umgehen. So können, falls der angesprochene Server dies durch die CORS Konfiguration zulässt, Daten zum Client gelangen. Mittlerweile unterstützen alle Browser CORS. (<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> , abgefragt am 02.03.2020) (<https://de.wikipedia.org/wiki/Cross-Origin_Resource_Sharing>, abgefragt am 02.03.2020)

**NGINX**: NGINX ist eine Alternative zu Apache. Mit beiden Applikationen ist es möglich, Webserver zu hosten. NGINX für braucht hierfür jedoch weniger Ressourcen und kann noch für vieles mehr. (<https://www.nginx.com/resources/glossary/nginx/> , abgefragt am 02.03.2020)

**SRTM (von Shuttle Radar Topography Mission):** Bei der SRTM Mission im Februar 2000 wurden Fernerkennungsdaten der Erde aus dem Weltraum aufgezeichnet. Sie dienen dazu, ein hochauflösendes und einheitliches Geländemodell der Welt zu erstellen. Die frei zugängliche Genauigkeit ist 30m (Jede Bogensekunde). Die Höhenabweichung für absolute Werte liegt hier bei etwa 6.2m. (<https://www.researchgate.net/post/What_is_the_vertical_resolutionaccuracy_of_Global_SRTM_1_arc_second_30_m2> , abgefragt am 02.03.2020) (<https://de.wikipedia.org/wiki/SRTM-Daten> , abgefragt am 02.03.2020)

OpenStreetMap:

API:

JSON:

Proxy:

AJAX: Hierbei handelt es sich um eine Abfragerate von 25m. Die Genauigkeit ist offiziell (+/-) 7m. Europaweit ist das die genauste Karte, die frei erhältlich ist.

GitHub:

EU-DEM: <https://courses.neteler.org/eu-dem-new-digital-surface-model-at-25m/>

**Dokumentation:**

**[26.02.2020 – 8 Stunden]:** Ich hatte gestern die Idee einen «War Thunder Grind» Rechner mit GitHub Pages zu erstellen. War Thunder ist ein Computerspiel mit Flugzeugen, Panzer und Schiffen vom 2. WK bis hin in den Kalten Krieg. Doch die Zeit, um weiterzukommen, ist enorm. Deshalb wollte ich einen Rechner erstellen, welcher berechnet, wie lange man etwa noch braucht.

Als erstes musste ich mich Informieren, wie man Karten auf Webseiten anzeigt. Da man für Google Maps oder Google Earth zahlen muss, um sie nutzen zu können, kam dies für uns nicht in Frage. Ich stiess auf Leafletjs mit OpenStreetMap als Kartenanbieter. Durch das Tutorial von Leaflet entstand schnell eine Webseite mit einer Karte, auf der man Marker setzen und zoomen kann. Doch dies Mühsame Arbeit kam, alsbald ich begann, mich um die Höhe zu kümmern. Ich verstand nicht, dass dies nicht so einfach ist und dass in der Tat, wenige Kartendienste Höhendaten liefern. Ich erkannte, dass Leafletjs keine Höhendaten auslesen kann und dass ich einen anderen Weg finden musst. Durch recherchieren, stiess ich auf SRTM und andere Höhenkartenanbieter. Die Files waren meistens in .tif oder .hgt. Also begann ich nach Lösungen zu suchen, wie ich diese Daten auf der Webseite auswerten kann. Es waren viele open-source Lösungen vorhanden, doch entweder wurden sie schon lange beendet, brauchten einen eigenen Server oder waren einfach nicht geeignet, da sie zum Beispiel nur ein Typ von EPSG unterstützten. Durch einen Stackoverflow Artikel stiess ich auf eine Webseite, opentopodata.org, die eine API zur Höhenberechnung durch AJAX Abfragen zur Verfügung stellt. Ich schien gerettet. Doch das nächste Problem lief mir schnell über den Weg. Durch CORS Implementation im Browser und Fehlkonfiguration des API-Servers war ich nicht in der Lage, die JSON Rückgabe der API zu lesen. Da ich keinen Zugriff auf den API-Server habe, konnte ich das CORS Problem nicht an der Wurzel packen, sondern musst eine Client Lösung finden. Es gibt zwei Wege. Entweder man deaktiviert CORS im Browser oder man nutzt einen Proxy Server, welcher dieses CORS Problem umgeht. Beide Wege haben ein grosses, jedoch verschiedenes Problem. Bei der ersten Handelt es sich um eine grosse Sicherheitslücke. Zu dem muss der Client ein Plugin herunterladen, um CORS zu deaktivieren. Dies kann man nicht von einem Client verlangen. Die Lösung mit einem Proxy Server ist jedoch auch nicht geeignet, da die Webseite dann von der Funktion und Bandbreite dieses bereitgestellten Dienstes abhängig ist. Jedoch blieb mir zu diesem Zeitpunkt keine andere Wahl als über Proxies zu gehen.

**[27.02.2020 – 9 Stunden]** Bis jetzt kann die Webseite nicht viel mehr als Marker hinzufügen, die dazugehörige Höhe anfragen und in ein Popup schreiben und eine Linie zwischen beiden Punkten ziehen. Jedoch ist die Webseite noch statisch, so heissen, die Marker können nicht verschoben werden. Um eine neue Route aufzusetzen, oder eine bestehende zu ändern, muss die Webseite neu geladen werden. Schnell überrannte mich ein anderes Problem. Opentopodata.org ist mehr als perfekt, doch der Service lässt nur 100 Anfragen pro Tag zu. Da infolge von CORS alle Anfragen über ein Proxy Server müssen, summieren dich die Anfragen, egal von welcher IP Adresse man darauf zugreift. Das grosse Suchspiel begann. Ich suchte Stunden nach Lösungen, um CORS zu umgehen. Der einzig plausible weg schien mir, alle verfügbaren Proxy Server hinzuzufügen und automatisch zum nächsten wechseln, falls einer das Anfragenlimit erreicht hat.

**[28.02.2020 – 6 Stunden]** Mein nächster Schritt war es, das ziehen der Marker zu gewährleisten. Denn bis an hin konnte man keine Änderungen vornehmen. Ich versuchte alles zu generalisieren und die einzelnen Instanzen dem ganzen Code verfügbar zu machen. Nun wird die x, y und z Position in einem Array gespeichert. Hinzu werden die Marker in einem anderen Array gespeichert. So kann jede Funktion jederzeit jeden Marker oder jede Position ansprechen und auswerten. Ich wurde mit den Linien nicht fertig, doch das Herumschieben und Aktualisieren der Marker funktionierte. Durch den Tag habe ich das Limit aller drei Proxies verbraucht, weshalb ich nicht mehr die Höhe abfragen konnte.

**[29.02.2020 – 14 Stunden]** Mit dem Willen dem ganzen Abhängigkeitsproblem ein Ende zu setzten, versuchte ich mit der Anleitung von opentopodata.org einen eigenen Server aufzusetzen. Ich kramte meinen alten Windows Computer hervor und liess Linux über den Stick laufen. Vorinstalliert im GitHub Repository waren gewisse Höhenkarten als Test. Doch um genaue Daten zu bekommen, muss man hochauflösende Höhenkartenteile herunterladen. Aus vorherigen Recherchen ging hervor, dass EU-DEM, von copernicus, die genausten Karten sind. Doch eine Karte hier ist 6.5 Gigabyte gross. Ich hätte keine Chance gehabt, wenn mir die Webseite opentopo.org nicht über den Weg gelaufen wäre. Denn mein Rechner hat stolze 8GB Ram, welches nicht einmal ausreichte, um die hochauflösende Datei zu öffnen. Opentopo.org stellt kleine Kartenteile bereit, die je 25 Megabyte gross sind, mit der gleichen Auflösung. Es sind zwar beachtlich mehr Teile, doch er muss immer nur genau diese Datei öffnen, in der die Koordinaten des abgefragten Ortes liegt. So schaffte ich die API zum laufen zu bringen. Jetzt konnte ich endlich CORS richtig konfigurieren, um die Ressourcen überall verfügbar zu machen. Erfolgreiche Tests im Lan und Wan haben bestätigt, dass alles, samt Firewall, richtig läuft. Der Zeitpunkt der Implementierung in die GitHub Pages Website kam schnell – So auch den nächsten Fehler. Da ich über GitHub (SSL) eine Ressource von meinem Server (kein SSL Zertifikat) erhalten wollte, lies dies der Browser aus Sicherheitsgründen nicht zu. So musste ich schliesslich wieder auf den Proxy Service zurückgreifen, um die Browser auszutricksen. Immerhin waren die Anfragen der API nicht mehr beschränkt.

**[1.03.2020 – 2 Stunden]** Heute packte ich das Problem an der letzten Wurzel. Dem Proxy Server. Ich dachte mir, wenn ich schon die eigene API Zuhause hoste, kann ich gerade so gut die Webseite Zuhause hosten. Ich konfigurierte den vorinstallierten Nginx Server, so dass er meine API auf Port 5000 anbietet, und die HTML Webseite auf Port 80. Somit sind alle CORS, sowie SSL, Probleme gelöst und ich kann den Server selbst handhaben. Der einzige Rückschlag ist die lange IPV6 Adresse in der Adressleiste. Doch ich werde den Server wie eine Unterseite meiner beiden Webseiten (be3dart.ch und be3dart.github.io) handhaben, so muss sich niemand eine IP Adresse merken und kann bequem per Google BE3dARt suchen.

[Zeit für Dokumentation: 3 Stunden]