Projektdokumentation ARIS

Gruppenmitglieder: Selina Büchel, Karin Mattmann, Yasaman Atighi Moghadam, Isabel von Ah

Daten

https://data.sbb.ch/explore/dataset/haltestelle-perronkante-inkl-bls/table/

Es handelt sich um ein Datensatz, in welchem man verschiedene Angaben über die Perronkanten der Schweizer Bahnhöfe findet, beispielsweise Perrontyp, -länge oder -höhe oder auch über die vorhandenen Hilfstritte. Mithilfe der mitgelieferten Koordinaten lassen sich auch die genauen Standorte der Perrons ermitteln.

Idee

Der Datensatz ist in unserer Webapplikation auf drei Hauptseiten aufbereitet. Diese sind durch einen Navigationsbereich erreichbar. Auf der ersten Seite sind die Daten in einer Tabelle mit ausgewählten Spalten formatiert. Es gibt die Funktion, nach Haltestellen alphabetisch oder Perronkantenlänge numerisch zu sortieren. Zudem kann man nach Perrontyp filtern oder nach Haltestellennamen suchen. Auf der Kartenseite sind alle Perrons auf einer Karte mit roten Kreisen angezeigt. In der letzten Seite sind die Daten der Kantenhöhe und des Höhenverlauf in Kreisdiagrammen angezeigt. Die Perronlänge ist in einem Balkendiagramm angezeigt.

Implementierung

Die Umsetzung des Projektes erfolgte mittels Python und dem Webframework Flask sowie der Web-Template-Engine Jinja. Zusätzlich wurde JavaScript und unterschiedliche JavaScript-Libraries eingesetzt (leaflet für die Implementierung der Karte und apexcharts für die Diagramme). Für die Darstellung wurde das css-Framework Bootstrap genutzt.

Im Python-File *data_gathering.py* werden die gewünschten Spalten des Datensatzes, der im csv-Format vorliegt, umgewandelt und in eine sqlite-Datenbank eingefügt. Zum Schluss werden die beiden Funktionen aus *data_clean.py* aufgerufen, um die Daten auf die gewünschte Darstellung vorzubereiten.

In *app.py* wird dann wieder auf die bereits befüllte und bereinigte Datenbank zugegriffen und dann werden die html-Seiten mit der Methode *render_template* generiert. Die dabei mitgegebenen Daten und Variabeln werden anschliessend mithilfe von Jinja in die Seiten eingefügt und verarbeitet. Die drei Seiten der Applikation setzen sich aus den gleichbleibenden Header und Footer und dem variierenden Teil zusammen.

Die erste Seite wird in app.py über vier verschiedene Routs generiert, mit jeweils unterschiedlicher Sortierung. Jedes Mal wird die Funktion *table()* aufgerufen, welche auf die Datenbank zugreift und eine Suchanfrage für die gewählten Filter und/oder Suche vorbereitet und diese dann mit dem Sortierungsabschnitt ergänzt. Während die Sortierung mittels Routs funktioniert, werden die Filter und die Suchanfrage über die URL-Argumente umgesetzt, welche dann mit Python zu einer SQL-Abfrage bearbeitet werden.

Die zweite Seite ist etwas simpler aufgebaut, da nur eine einfache Datenbankabfrage gemacht werden muss für die Kartendarstellung. Die Perrons sind mit roten Kreisen markiert und beim Draufklicken wird der Name der Haltestelle, die Gleisnummer sowie die Länge des Perrons angezeigt. Jeder Kreis und das dazugehörige Popup wird einzeln generiert, indem einmal über den gesamten Datensatz iteriert wird.

Um die Diagramme auf der dritten Seite zu erstellen, werden gleich entsprechende Datenbankabfragen gemacht und die erhaltenen Objekte in Python in geeignete Listen umgewandelt. Diese lassen sich dann in die entsprechenden Arrays der apexcharts-Optionen einfügen.

Für die Versionsverwaltung haben wir mit git und für die Zusammenarbeit mit gitHub gearbeitet.

Problemstellungen und Alternativideen

Stolpersteine und Gelerntes

Am herausforderndsten, aber auch am lehrreichsten, war das erfolgreiche Verknüpfen der verschiedenen Tools, das Finden der richtigen Methods und Libraries und das Weitergeben der Daten und Variablen, vom csv-File bis zur Anzeige im html. Insbesondere die Verwendung von Jinja war für uns noch nicht so intuitiv, da wir zuerst nicht verstanden haben, dass wir die Jinja-Ausdrücke einfach dort im Template einsetzen dürfen, wo die entsprechenden Variablen schlussendlich stehen sollen, zum Beispiel genau innerhalb der Anführungszeichen, zwischen welchen die Klassen des Elements aufgelistet werden. Als Jinja im JavaScript-Teil verwendet wurde, kam noch zusätzlich hinzu, dass unsere Editoren dies nicht erkannten und den ganzen Abschnitt als Fehler markierten.

Schwierigkeiten und ihre Lösungen

Die Spalten Haltestelle und Perronkantenlänge sollten nach dem Alphabet und der Nummerngrösse sortiert werden. Ursprünglich war angedacht, dies mit zwei Pfeilen neben den entsprechenden Spaltenüberschriften zu implementieren, die Farbe und Richtung ändern sollten. Jedoch war es schwierig, die Abfrage mit dem Pfeil zu verbinden, da die

Seiten jedes Mal neu geladen werden. Die Funktionalität wurde deshalb mittels vier Buttons umgesetzt.

Noch offene Punkte und weitere Ideen

Bei jedem Auswählen einer anderen Sortierung wird die Seite neu geladen und es ist leider nicht möglich, die Suche und Filter beizubehalten. Weiter hat sich Umsetzung einer Pagination mit Flask als schwieriger herausgestellt als erwartet und diese Funktion musste deshalb weggelassen werden.

Als anderer Lösungsweg hätte auch durchgehend JavaScript (und node.js) genutzt werden können anstelle von Flask. Weiter hätte auch PostgreSQL statt SQLite verwendet werden können, was allerdings die Arbeit mit GitHub erschwert hätte.

Was ausserdem noch Verbesserungspotential hätte, ist die Ladezeit. Da jede Funktionalität eine Abfrage der gesamten Datenbank verursacht, verstreicht jeweils ein Augenblick, bis die Liste geladen wird. Dies könnte möglicherweise gelöst werden, indem nur beim ersten Laden der Seite eine Abfrage gemacht wird und die Sortierung und die Filterung stattdessen mit Python vorgenommen wird.