benzin.tech

Logo von benzin.tank

Entwickelt von:
Alexander Gohlke
Larissa Janine Süßenbach
Felix Schon

Inhaltsverzeichnis

1. Aufgabenstellung

1.1 Allgemein

In diesem Projekt wird eine Software entwickelt werden, die Tankstellen-Preise vorhersieht und die Tankstrategie für spezifische Routen optimiert.

1.2 Gegebene Daten

Dieses Vorhersehen geschieht mit Hilfe von Preisanalysen, die aus Datenbanken im Plain-Text-Format erstellt werden. Die erste Datenbank listet alle Tankstellen in Deutschland im folgenden Muster auf:

```
5;team Tankautomat Hesbüll;Raiffeisen;Raiffeisenstr.;1;25927;Hesbüll;54.9009;8.81069
6;Aral Tankstelle;ARAL;Keitumer Landstraße;"";25980;Sylt-Ost;54.9004021;8.339205
7;team Tankautomat Süderlügum;team;Hauptstr.;16;25923;Süderlügum;54.8708;8.90684
8;ELAN Suederluegum;ELAN;HAUPTSTR.;4;25923;SUEDERLUEGUM;54.8692;8.90622
```

Abb. 1: Auszug aus der Datenbank “Tankstellen.csv”
Formatierung: ID;TankstellenName;Konzern;Straße;Hausnummer;PLZ;Ort;LON;LAT

Um nun an die Preise der einzelnen Tankstellen zu gelangen brauchen wir zusätzliche Daten. Für aufgelistete Tankstelle gibt es also noch eine weitere Datenbank, die die Benzinpreise folgendermaßen enthält:

```
2017-06-27 16:37:06+02;1329
2017-06-27 17:13:06+02;1349
2017-06-27 18:13:06+02;1329
2017-06-27 18:15:06+02;1329
2017-06-27 21:13:05+02;1319
2017-06-27 23:03:05+02;1399
```

Abb. 2: Auszug aus der Datenbank “1.csv”
Formatierung: Jahr-Monat-Tag Stunden:Minuten:Sekunden+Hundertstel;Preis in ct.

Diese Datenbanken wurden von Tankerkönig.de vom 08.09.2017 extrahiert.

Weitere vorgegebene Daten waren die Großkreisformel zur Berechnung der Distanz zwischen Tankstelle a und Tankstelle b. Diese sieht folgendermaßen aus:

*dist(a,b)=6378.388*acos(sin(lat a)*sin(lat b)+cos(lat a)*cos(lat b)*cos(lon b-lon a))*

Darüber hinaus haben wir eine Tankkapazität von 3 Litern und einen Spritverbrauch von 5,6l/100km vorgegeben bekommen.

2. Planung

2.1 Allgemein

Bei der Planung zur Realisierung der Aufgabe entschieden wir uns für eine Webapplikation, in der ein Benutzer seine relevanten Daten eintragen kann und seine Tankstrategie angezeigt bekommt. Diese werden dann von einem Server berechnet.
Als relevante Daten definieren wir Routen, die für die Berechnung der Tankstrategie vonnöten sind. Die Route setzt sich aus mehreren Zeilen zusammen in denen Datum und die Tankstellen-ID stehen und somit einen Routenpunkt bilden.
In der ersten Zeile jedoch steht immer eine 3. Dies ist die Tankkapazität, die von der Aufgabenstellung vorgegeben ist.

2.2 Datenanalyse

Bei der Datenanalyse hatten wir mehrere Ansätze, wie wir den Preis berechnen könnten. Wir haben uns letztendlich für eine vereinfachte Analyse entschieden, wo wir für jede Tankstelle 365 Preise analysieren. Das heißt, dass jeder Tag einen analysierten Preis hat, der von der Tankstelle abhängig ist. Dafür haben wir für ein gewisses Datum, alle Preis-Daten aus der Vergangenheit genommen und diese zu einem Mittelwert berechnet. Dadurch kann man auch gut sehen, welche Tankstelle im Schnitt günstiger ist. Das ist der Grund dafür, dass wir auf der Website eine Funktion zur Preisabfrage von einzelnen Tankstellen implementiert haben.

2.3 Routenplaner (aus zeitlichen Gründen entfernt!)

Wie man den Daten entnehmen kann, ist nur ein Startpunkt und Zielpunkt gegeben. Wir wollen also alle Zwischenziele, d.h. Tankstellen an denen man tankt, dynamisch berechnen, um so die günstigste Strecke zu berechnen.

Letztendlich mussten wir den Routenplaner aus zeitlichen und anderen Gründen entfernen.

2.4 GUI-Planung

Beim GUI-Design haben wir uns für eine, für den Nutzer leicht bedienbaren, Benutzeroberfläche entschieden und versuchen Diese auch minimal zu halten.

Dabei standen anfangs vier Buttons zur Verfügung, die den Nutzer auf verschiedene Seiten weitergeleitet haben. Dazu gehörten “Benutzer”, “Karte”, “Impressum” und “Route”, wobei in “Route” die Endergebnisse des gesamten Programms zusammengefasst dargestellt werden.
Um nun aber die leichte Bedienbarkeit zu fördern, haben wir die Anzahl der Buttons auf zwei reduziert und somit blieben “Route” und “Karte” bestehen.
Anstatt des “Benutzer”-Buttons haben wir ein Seitenelement eingebaut, das den Benutzer, egal auf welcher der beiden Seiten man sich befindet, begleitet und seine persönlichen Angaben anzeigt und diese auch gegebenenfalls abändern lässt.
Durch das Entfernen des Routenplaners wurden jedoch die Variablen entfernt und somit implementierten wir die verlangten Routen, die mit Hilfe des Routen-Erstellers erzeugt wurden.
Es stehen nun Routen zur Verfügung, die im Seitenelement auswählbar sind. Diese ersetzen die persönlichen Angaben des Benutzers.

Desweiteren haben wir uns viele Gedanken zur Darstellung der Tankstellen gemacht und dabei an einigen Konzepten überlegt.
Zum einen entschieden wir uns für einen “Tanknadelparser” und ein Feature, das im Footer alle Tankstellen in Form von Punkten darstellt, die hinter einander und mit relativen Abstand zu der

echten Entfernung der Tankstellen stehen. Diese werden durch Linien verbunden und beim Hovern über eines der Punkte erscheint der Name der Tankstelle. Bei einem Klick auf einen dieser Punkte wird man zu der Karte weitergeleitet und die ausgewählte Tankstelle wird angezeigt. Dies sollte aber nicht die Hauptanzeige werden, sondern lediglich eine Übersicht der Route darstellen.

Die Hauptanzeige der Ergebnisse wollten wir mit einer Tabelle darstellen, in der die Informationen der Tankstellen stehen und die zu tankenden Liter und der dabei voraussichtlich entstehende Preis. Diese Tabelle wird auf der Routen-Seite angezeigt.

Zusätzlich haben wir an einen Multilingualen-Webdesign überlegt und wollten dafür sorgen, dass man sich die Sprache mit einem Klick auf eine Länderflagge aussuchen kann.

Das Seitenfeld war ebenfalls eine Überlegung um dem Bediener eine angenehme Übersicht über die Daten zu geben, welches auf beiden Seiten angezeigt wird um die Konsistenz zu bewahren.

Das Layout der Website soll konsistent sein und nicht rumspringen und den Nutzer nicht verwirren. Deswegen haben wir uns dafür entschieden, dass, mit der Ausnahme des Hauptprogramms, jedes Feature entweder auf beiden Seiten angezeigt oder, wenn dies nicht möglich ist, rausgenommen wird.

Ein weiterer Punkt in der Planung war die Implementierung einer Karte. Dies haben wir mithilfe von Leaflet[1] und OpenStreetMap[2] realisiert. Die Karte soll dem Nutzer helfen, eine grobe Orientierung seiner Route zu erhalten.

2.5 Programmiersprachen

Wir haben für die Berechnungen der Daten und Route C++ benutzt, für den Routen-Ersteller Java und für die Webapplikation hauptsächlich Python, zusammen mit JavaScript, HTML, SQL und css.

1 http://leafletjs.com/
2 https://www.openstreetmap.de/

3. Hilfsprogramme

3.1 Flask

Bild von Flask

Flask ist ein Web Framework, welches in Python geschrieben ist. Es ist darauf ausgelegt einfach Bibliotheken zu integrieren und stellt daher keine Komponenten zur Verfügung.

3.2 Git

Bild von git

Git ist ein Versionsverwaltungssystem, in dem man mit mehreren Personen an einem Projekt gleichzeitig arbeiten und jeder seine Fortschritte zu einem beliebigen Zeitpunkt mit den anderen teilen kann.

3.3 OpenStreetMap

Bild von OpenStreetMap

OpenStreetMap ist eine Datenbank mit Geoinformationen, die frei nutzbar ist und die Geoinformationen sammelt, strukturiert und in Form einer Landkarte darstellen kann.

3.4 Leaflet

Bild von leaflet

Leaflet ist eine open-source Bibliothek, die dafür da ist, interaktive Karten darzustellen.

4. GUI

4.1 Allgemein

Wie oben schon beschrieben, benutzen wir eine Webapplikation als Benutzeroberfläche. Dafür haben wir uns einen Server gemietet mit der Domain “www.benzin.tech”.

4.2 Probleme und Lösungen

Bei der Realisierung der in der Planung beschriebenen Features, sind wir auf einige Probleme gestoßen, die hier beschrieben werden.

4.2.1 Kalender

Das erste Problem, auf das wir bei der GUI gestoßen sind, war die Entwicklung des Kalenders, da wir dort keine Erfahrung mit hatten und uns viel beibringen und ausprobieren mussten. Dabei haben wir es dann geschafft den Kalender mit Python und Flask zu implementieren.

4.2.2 Übersetzung

Die Übersetzung in verschiedene Sprachen konnten wir einfach mit einer, über Flask laufenden Hilfsbibliothek namens “Babel”, implementieren. Bei der Ausführung gab es jedoch einen Bug, den wir nicht beheben konnten:
Die Variable, die die Sprache bestimmt, konnte erfolgreich geändert werden, doch das Neuladen der Seite führte dazu, dass die Variable auf den alten Wert zurück sprang.
Auch die Flaggen, die wir zur Sprachauswahl implementiert hatten, blieben unverändert.
Daher haben wir uns dafür entschieden die Übersetzung wegzulassen. Im Code gibt es noch Fragmente dieser Idee, die nicht entfernt wurden.

4.2.3 Implementierung von “select2”

Für die Implementierung des DropDown-Menü für die Anzeige der Tankstellen, wollten wir zuerst eine Vorlage aus
http://select2.github.io/select2/ benutzen. Dies hat aufgrund vieler Errors nicht funktioniert und wir haben uns für ein eigenes DropDown-Menü entschieden.
Da die Tankstellen-Datenbank relativ groß ist, verzögert sich die Suchfunktion. Diese Verzögerung betrug anfangs 2 Minuten und ist nun, nach viel Optimierung, immer noch vorhanden aber wesentlich kürzer und somit kein störender Faktor mehr.

4.2.4 Warum die Webseite nun nicht erreichbar ist

Durch den Entwicklungsprozess wurde anscheinend der Python-Interpreter anscheinend leicht beschädigt. Ein problematisches Paket auf dem Server scheint Matplotlib zu sein, der beim Ausführen einen Pointer-Error erzeugt.
Wir haben versucht dies durch viele Ansätze zu beheben, darunter Python neuinstallieren, Python virtual environment, Python dev Version. Leider haben wir am Ende keine Lösung gefunden um den Fehler zu beheben.
Jedoch wird dieses auch nach Abgabe weiter versucht in Stand zu bringen.

5. Tankpreis Analyse

5.1 Allgemein

Um einen Preis zu ermitteln braucht man verschiedene Werte die relevant für die Analyse sind. Da es über mehrere Jahre am Tag auch mehrere Preisänderungen gab waren es sehr viele Daten. Diese wollten wir vereinfachen, indem wir aus den bekannten Daten für jeden Tag im Jahr einen Preis festlegen. Dieser ist der Mittelwert aller an diesem Tag eingetragenen Preise. Da es nun aber mehrere Jahre gibt, haben wir dies für alle vorhandenen Jahre vorgenommen und dann alle Jahre zusammen zu einem Mittelwert berechnet. Es gibt also nun pro Jahrestag einen Wert für jede Tankstelle. Je größer die Datenbank wird, desto genauer wird diese Art der Analyse.
Die Berechnungen dieser Analyse läuft über unseren Server.

5.2 Probleme und Lösungen

5.2.1 Fehlende Daten

Als wir uns die Datenbanken angeguckt haben, ist uns aufgefallen, dass 304 Preis-Datensätze fehlen. Dies hat zufolge, dass es für 304 Tankstellen keine Preisdaten gibt. Eine Lösung für dieses Problem könnte sein, dass man benachbarte Tankstellen für die Analyse benutzt, da diese häufig ähnliche Preise vorweisen.

5.2.2 Rundung

Da Tankstellenpreise immer mit 0,9ct enden, sind unsere Werte nicht ganz richtig, da diese auf alle Nachkommastellen enden können. Deshalb haben wir eine Rundung implementiert, die alle Werte mit x,0ct - x,4ct abrundet auf x-1,9ct und bei x,5ct - x,8ct auf x,9ct aufgerundet, wobei x der Tankstellenpreis in Cent ist.

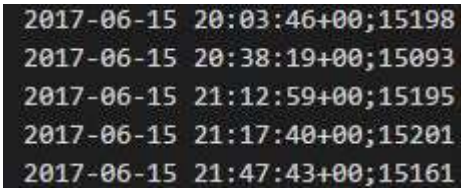
5.2.3 Testwerte

Um zu testen, wie genau unsere Preisanalyse ist, haben wir einmal unseren Algorithmus mit Tankstellen IDs und Datum, die innerhalb der gegebenen Daten sind, durchgeführt und die Ergebnisse mit den tatsächlichen Preisen verglichen. In den meisten fällen liegen die Unterschiede unterhalb von 10 cent, sehr selten aber auch höher. Dabei ist zu berücksichtigen, dass für diese Daten kein großer Datensatz vorhanden war. In der Zukunft liegende Ergebnisse werden daher wahrscheinlich genauer sein.
Die genauen Ergebnisse der Testroute stehen im Anhang.

6. Routenanalyse

6.1 Allgemein

Die Route wird in einer .csv-Datei angegeben. Diese besteht aus mehreren Zeilen, wobei jede einzelne aus der Ankunftszeit an der Tankstelle und der Tankstellen ID besteht.



Ausschnitt einer Routen.csv Datei

Für die Berechnung der Route sind wir von einem 3 Liter Tank und einem Verbrauch von 5,6 Liter pro 100km ausgegangen. Der Tank ist beim Startpunkt leer. Diese Daten wurden in der Aufgabenstellung vorgegeben. Diese Analyse wird von unserem Server ausgeführt.

6.2 Algorithmus

Um eine möglichst günstige Fahrt zu erreichen haben wir einen Algorithmus entwickelt. Bevor wir mit diesen anfangen, nehmen wir noch folgendes an:

Man tankt nur genau so viel, wie man braucht, um die nächste Tankstelle zu erreichen. Außerdem tanken wir genau so viel, wie der Rechner es uns vorgibt. Das heißt, dass die Ergebnisse nicht gerundet werden um ein möglichst genaues und einzigartiges Ergebnis zu erhalten. Da wir auch nur einen 3 Liter Tank zur Verfügung stehen haben machen kleine Änderungen schon einen Unterschied.

Der Algorithmus ist folgendermaßen aufgebaut:

-Zuerst suchen wir die Tankstelle mit dem günstigsten Preis.

-Wir tanken voll oder genau passend, um die Ziel-Tankstelle zu erreichen.

-Wir wiederholen dies für die Tankstellen bis zu dieser Tankstelle. Diese Tankstelle ist nun das Ende.
-Wir wiederholen dies noch einmal für die Tankstellen ab dieser Tankstelle bis zum Ende.
Wir haben, aus praktischen Gründen, den Algorithmus in 2 Teile aufgeteilt. Der erste Teil ist der rekursive Teil. Dieser sucht die günstigste Tankstelle und ordnet ihr das Ende zu. Der zweite Teil berechnet dann mit Hilfe dieser Zuordnung die zu tankenden Liter.

6.2.1 Algorithmus Teil 1

-Suche die günstigste Tankstelle.
-Speichere das aktuelle Ende zu dieser Tankstelle
-Führe das gleiche nun nochmal für die Tankstellen bis zu dieser Tankstelle durch. Die erste Tankstelle bleibt gleich, die End-Tankstelle ist die aktuelle Tankstelle.
-Führe nun das gleiche nochmal für die Tankstellen ab dieser Tankstelle bis zum Ziel durch. Die erste Tankstelle ist die aktuelle und das Ende ist das Ziel.

```
1   void CalculateRoute::calculate(int von, int to)
2   {
3     if(von <= to && von != route->length-1)
4         {
5           int min = von;
6           for(int x=min+1; x<=to;x++)
7           {
8           if(allStation[listStation[x]-1]->getPreisatTime(0) <
9           allStation[listStation[min]-1]->getPreisatTime(0))
10                {
11                    min = x;
12                }
13          }
14          orderList[min] = to+1;
15          calculate(von, min-1);
16          calculate(min+1, to);
17  }
18  }
```

Code zu Teil 1

6.2.2 Algorithmus Teil 2

-Durchlaufe die Tankstellen-Datenbank

-Berechne die Entfernung der Tankstelle zu seiner End-Tankstelle

(Dabei ist zu beachten, dass z.B. von Tankstelle 1 zu Tankstelle 3 die Strecke von Tankstelle 1 zu Tankstelle 2 zu Tankstelle 3 berechnet werden muss)

-Mit Hilfe der Entfernung den Liter-Verbrauch ausrechnen.

-Ist der Verbrauch größer als 3 Liter, wird er Verbrauch auf 3 Liter gesetzt, da dies die maximale Tankgröße ist.

-Wenn der Verbrauch kleiner ist, als die aktuelle Tankfüllung, dann einmal den Verbrauch von der Tankfüllung abziehen und die zu Tankende Menge ist 0 Liter. Ist der Verbrauch größer als die Tankfüllung, dann zieht man die Tankfüllung vom Verbrauch ab und tankt das Ergebnis.

```
double litterAvailabel = 0.0;
for(int x=0; x<route->length-1;x++)
{
  double litter = (distanceVonAToB(x, orderList[x])/100)*5.6;
  if(litter > 3.0)litter = 3.0;
  if(litter < litterAvailabel)
  {
    litterAvailabel -= litter;
    toRefuel[x] = 0.0;
  }
  else
  {
    toRefuel[x] = litter - litterAvailabel;
    litterAvailabel = litter -((distanceVonAToB(x,x+1)/100)*5.6);
  }
}
```

Code zu Teil 2

6.3 Probleme und Lösungen

6.3.1 Entfernungsberechnung

Der Weg von der Tankstelle 1 zur Tankstelle 3 ist nicht der direkte Weg, sondern man muss alle Zwischen-Tankstellen mit einbeziehen. Dies hat den Algorithmus ein bisschen komplizierter gemacht aber letztendlich hat es dann geklappt.

6.3.2 Großkreis-Formel

Für die Berechnung der Distanz zwischen zwei Tankstellen wurde uns eine Großkreis-Formel vorgegeben. Diese sieht folgendermaßen aus:

*dist(a,b)=6378.388*acos(sin(lat a)*sin(lat b)+cos(lat a)*cos(lat b)*cos(lon b-lon a))*

Das Problem für uns war die Überprüfung, ob wir Diese richtig implementiert haben und wie genau sie ist, da wir die mit Online-Kartenanbietern verglichen haben und dort andere Ergebnisse erhielten. Auch ist uns aufgefallen, dass die Python Bibliothek, anstatt den Wert 6378.388, den Wert 6372.795 nimmt.

Ob unsere Werte nun korrekt sind haben wir einfach folgendermaßen getestet:

Jeder von uns hat die Formel in einer unterschiedlichen Sprache programmiert und dann die Ergebnisse verglichen. Da wir alle die gleichen Ergebnisse bekommen haben gehen wir nun davon aus, dass wir die Formel richtig implementiert haben. Es ist natürlich möglich, dass wir alle falsch liegen. Eine genaue Überprüfung ist also nicht möglich.

6.3.3 Rundung

Ein weiteres Problem hier war die Rundung des Rechners.
Das erste Problem entstand schon beim Einlesen der Längen- und Breitengrade, da diese nicht konkret als Floating Point dargestellt werden. Man hat also nun schon ungenauigkeiten durch die Breiten- und Längengraden.

6.3.4 Probleme mit der kompatibilität unter Windows

Wir stellten fest, dass der Code, der unter Linux läuft, nicht eins zu eins unter Windows nutzbar ist.
Windows erfordert eine leicht andere Syntax im Umgang zu den Server-Ports. Zusätzlich sind gewisse Funktionen in anderen Bibliotheken zu finden oder gar nicht vorhanden.
Wir hatten also Probleme bei der Übertragung des Codes von Linux auf Windows in Bezug auf das Handling das Ports und den Funktionen to_string() und strptime().
Die Funktion to_string() wandelt einen Integer oder eine Gleitkommazahl in einen String um.
strptime() wandelt ein Datum in einen String um (zb. “2016-01-20 13:55:30”.
Für die Ersetzung to_string() Funktion haben wir Streams verwendet.
Für die strptime() Funktion haben wir eine eigene Funktion toTime() geschrieben.

Diese sieht folgendermaßen aus:

```
struct tm toTime(string time)
{
    string jahr = time.substr (0,4);
    string monat = time.substr (5,2);
    string tag = time.substr (8,2);
    string stunde = time.substr (11,2);
    string minute = time.substr (14,2);
    string secunde = time.substr (17,2);
    struct tm erg;
    erg.tm_year = atoi(jahr.c_str()) -1900;
    erg.tm_mon = atoi(monat.c_str())-1;
    erg.tm_mday = atoi(tag.c_str());
    erg.tm_hour = atoi(stunde.c_str());
    erg.tm_min = atoi(minute.c_str());
    erg.tm_sec = atoi(secunde.c_str());
return erg;
}
```

Bild der toTime() Funktion

Wir bieten also, als Lösung des Problems, einen separaten, abgewandelten Code an.
Da zur Zeit Webserver meist linux-basierte Systeme sind, haben wir uns hauptsächlich auf diesen Code fokussiert.
Der Code für Windows ist zwar funktionsfähig aber nicht so gut getestet wie der des Linux-Systems.

6.4 Effizienz

Wir haben unsere Strategie mit einer naiven Strategie bei einigen Testrouten verglichen. Dabei ist anzumerken, dass wir Routen nehmen sollten, die besonders gut bei unserer Strategie abscheinden. Wir haben uns dagegen entschieden und zufällige bzw. willkürliche Routen erstellt, da wir auch zeigen wollen, dass ggf. auch “worst case Routen” dabei sein können, wo die naive Strategie besser ist. Als naive Strategie haben wir “immer volltanken” benutzt. Das heißt, dass man bei jeder Tankstelle den

Tank wieder komplett auffüllt. Dabei ist uns aufgefallen, dass unsere Strategie in den meisten Fällen effizienter ist als die Naive.
Die genauen Ergebnisse sind im Anhang. Dabei sind die Routen IrgendwoImNirgendwo.csv und IrgendwoImNirgendwo2.csv die “worst case Routen”.

7. Routen-Ersteller

7.1 Allgemein

Damit man vernünftige und nachvollziehbare Routen erstellen kann, haben wir uns dafür entschieden einen Routen-Ersteller zu schreiben. Er ist in Java geschrieben und hat ein eigenes GUI.
Für die Berechnung der Zeiten gehen wir von einer Durchschnittsgeschwindigkeit von 60 km/h aus.
Dieses Programm war erst nur ein Hilfsprogramm für uns, um Routen einfach und schnell erstellen zu können. Wir haben uns aber dazu entschlossen, dass es mit in das Projekt kommt, um es zu demonstrieren.

7.2 Funktionen

Der Routen-Ersteller bietet eine Liste mit allen Tankstellen, die man über eine Suchfunktion filtern kann und die es erlaubt eine eigene Route zu erstellen, zu analysieren und als .csv-Datei zu speichern. Dies hat den Vorteil, nicht jede Tankstelle in der Datenbank zu suchen und per Hand in eine .csv-Datei zu schreiben.

7.3 Probleme und Lösungen

Da dieses Programm recht simpel ist, gab es keine größeren Probleme. Es wäre jedoch schön gewesen, das Programm in der Website implementiert zu haben. Dafür fehlte uns aber letztendlich die Zeit. Aus diesem Grund ist das Programm nun ein Stand-Alone.

7.3.1 Distanzberechnungsfehler

Bei der Erstellung von Routen ist uns aufgefallen, dass für manche Tankstellen eine falsche Distanz berechnet wird.

Beispielsweise wird eine Distanz von 46km berechnet, obwohl die Tankstellen über 200km voneinander entfernt sind. Dies passiert zwar eher selten aber kann trotzdem zu Fehlern bei der Analyse führen. Dieses Problem konnte leider nicht behoben werden, da die Ursache unklar ist. Die Formel für die Distanzberechnung stimmt.

8. Zusammenfassung

Das Projekt ist in drei Teile unterteilt:
Der Routen-Ersteller, die Website und der Server.

8.1 Der Routen-Ersteller

Der Routen-Ersteller ist dafür da, eine Route zusammenzubauen. Er unterstützt einen dabei mögliche Routen zu erstellen, indem er nach gewissen Tankstellen suchen kann, darauf hinweist, falls eine Tankstelle mit einem 3 Liter tank nicht zu erreichen ist und berechnet außerdem die Zeiten, an denen man an den einzelnen Zielen ankommt, sowie den Verbrauch und die Distanz, sowohl gesamt als auch für die einzelnen Zwischenziele. Er kann die erstellte Route dann in einer .csv-Datei speichern, die dann von der Website eingelesen werden kann.

8.2 Die Website

Auf der Website kann man die eingelesenen Routen ansehen und auswählen. Wenn man sich für eine Route entschieden hat und die Analyse starten will, wird die Route an den Server geschickt. Das Ergebnis wird dann in Form einer .json-Datei zurückgegeben und von der Website aus lesbar dargestellt.
Außerdem kann man sich eine Tankstelle raussuchen und für diese spezifisch eine Preisanalyse anfragen. Dies funktioniert wie bei der Routen-Übertragung und der Server erkennt den unterschied und schickt wiederum die Ergebnisse an die Website zurück.

8.3 Der Server

Der Server analysiert die übergebene Route, berechnet eine Tankstrategie und übergibt diese über einen Port an die Website

in Form einer -json-Datei. Desweiteren wird auch eine .csv-Datei in dem Ordner “Ausgabe” hinterlegt.

8.3.1 Die Klassen

8.3.1.1 benzintech.cpp

“benzintech.cpp” ist das Hauptprogramm. Dort findet die Kommunikation zur Website über einen Port statt und es wird entschieden, ob es sich um eine Preisabfrage oder eine Routenanalyse handelt.

8.3.1.2 “Station”-Klasse

Hier werden alle Daten zu einer Tankstelle gespeichert. Dazu gehören alle Daten, die in der Datenbank “Tankstellen.csv” zur Verfügung stehen.
Desweiteren enthält die Klasse eine Funktion, die einen Preis zu einem gewissen Datum analysiert und zurückgibt.

8.3.1.3 “GasPrice”-Klasse

Diese Klasse liest die Preisdaten einer Tankstelle ein und führt gegebenenfalls die Analyse durch.

8.3.1.4 “DriveRoute”-Klasse

In der “DriveRoute”-Klasse werden die Daten einer Route eingelesen und gespeichert. Eine Route besteht aus mehreren Zeilen in denen pro Zeile jeweils ein Datum mit Zeit und eine ID der Tankstellen steht.

8.3.1.4 “CalculateRoute”-Klasse

Die Berechnung der Tankstrategie für eine Route und die Speicherung der des Ergebnisses finden hier statt.

8.3.1.5 “toJson”-Klasse

Um der Website eine formatierte Datei zu schicken, wandelt diese Klasse die Daten einer Preisabfrage oder einer Tankstrategie in das .json-Format um.

8.4 Allgemeine Probleme

Bei dem Projekt war es sehr wichtig, dass viel Kommunikation in der Gruppe herrscht. Dafür hatten wir einen Termin in der Woche wo wir uns getroffen haben um über bestehende Probleme zu reden oder unsere Herangehensweise zu überdenken. Dennoch sind manche Probleme aufgetreten.

Bei den Treffen haben wir zu viel Zeit in die Planung des Routenplaners gesteckt und zu wenig in die Analyse. Genaue Schnittstellen zu definieren, damit wir recht schnell ein funktionierendes Grundprogramm aufgebaut hätten, welches das Testen und die Funktionen leichter und anschaulicher darstellt, wäre eine zusätzliche Entlastung gewesen.

Auch die gegenseitige Hilfe war recht schwer, weil wir untereinander nicht genau wussten, woran der andere gerade arbeitet, da wir das Projekt in kleinere Teile aufgeteilt haben, was nicht das Problem war, jedoch die Transparenz fehlte dabei.

Außerdem haben wir viel Zeit und Gedanken in den Routenplaner gesteckt (hier ist nicht der Routen-Ersteller gemeint!), der es letztendlich aber leider nicht in das Programm geschafft hat, da der Rechenaufwand zu hoch war und er mit vielen Problemen verbunden war.

Anhang

Analysierte Daten der Testroute (Tankpreis Analyse)

(Werte in cent*10)

6000 Esso Tankstelle DR.-C.-OTTO-STR. 12 0 44879 BOCHUM

uhrzeit	originaldaten	geschätzte daten	differenz
2016-10-10 12:00:00	1309	1349	40
2016-10-11 12:00:00	1289	1339	50
2016-10-12 12:00:00	1269	1339	70
2016-10-13 12:00:00	1279	1349	70
2016-10-14 12:00:00	1279	1349	70
2016-10-15 12:00:00	1379	1349	-30
2016-10-16 12:00:00	1289	1339	50
2016-10-17 12:00:00	1269	1349	80
2016-10-18 12:00:00	1279	1339	60
2016-10-19 12:00:00	1289	1339	50
2016-10-20 12:00:00	1289	1349	60
2016-10-21 12:00:00	1269	1349	80

6500 Georg Behrens Hattinger Str. 122 58332 Schwelm

uhrzeit	originaldaten	geschätzte daten	differenz
2015-10-09 12:00:00	1319	1299	-20
2015-10-10 12:00:00	1349	1299	-50
2015-10-11 12:00:00	1319	1279	-40
2015-10-12 12:00:00	1339	1319	-20
2015-10-13 12:00:00	1339	1279	-60
2015-10-14 12:00:00	1329	1279	-50
2015-10-15 12:00:00	1309	1289	-20
2015-10-16 12:00:00	1309	1299	-10
2015-10-17 12:00:00	1299	1299	0
2015-10-18 12:00:00	1319	1279	-40
2015-10-19 12:00:00	1319	1319	0
2015-10-20 12:00:00	1319	1279	-40

6300 Tucht energy - Heizöl Tel.: 02331/13081 Eilperstraße 129 58091 Hagen

uhrzeit	originaldaten	geschätzte daten	differenz
2016-09-10 12:00:00	1269	1339	70
2016-09-11 12:00:00	1329	1329	0
2016-09-12 12:00:00	1289	1339	50
2016-09-13 12:00:00	1289	1339	50

2016-09-14 12:00:00	1299	1329	30	
2016-09-15 12:00:00	1289	1329	40	
2016-09-16 12:00:00	1259	1329	70	
2016-09-17 12:00:00	1249	1339	90	
2016-09-18 12:00:00	1279	1329	50	
2016-09-19 12:00:00	1229	1339	110	+
2016-09-20 12:00:00	1289	1339	50	
2016-09-21 12:00:00	1259	1329	70	

200 Aral Tankstelle Dreilanden 7 25826 Sankt Peter Ording

uhrzeit	originaldaten	geschätzte daten	differenz
2016-10-10 12:00:00	1309	1329	20
2016-10-11 12:00:00	1329	1319	-10
2016-10-12 12:00:00	1319	1329	10
2016-10-13 12:00:00	1329	1339	10
2016-10-14 12:00:00	1329	1349	20
2016-10-15 12:00:00	1399	1379	-20
2016-10-16 12:00:00	1309	1319	10
2016-10-17 12:00:00	1339	1329	-10
2016-10-18 12:00:00	1379	1319	-60
2016-10-19 12:00:00	1329	1329	0
2016-10-20 12:00:00	1319	1339	20
2016-10-21 12:00:00	1309	1349	40

100 Walter Röpcke Schubystr. 59 24837 Schleswig

uhrzeit	originaldaten	geschätzte daten	differenz
2015-01-01 11:00:00	1259	1329	70
2015-01-02 11:00:00	1319	1329	10
2015-01-03 11:00:00	1289	1339	50
2015-01-04 11:00:00	1339	1339	0
2015-01-05 11:00:00	1329	1329	0
2015-01-06 11:00:00	1319	1329	10
2015-01-07 11:00:00	1299	1329	30
2015-01-08 11:00:00	1299	1329	30
2015-01-09 11:00:00	1309	1329	20
2015-01-10 11:00:00	1299	1339	40
2015-01-11 11:00:00	1299	1339	40
2015-01-12 11:00:00	1289	1329	40

1040 AGRAVIS Ems-Jade GmbH, Tankstelle Jever Wittmunder Str. 20 26441 Jever

uhrzeit	originaldaten	geschätzte daten	differenz
2016-04-01 13:00:00	1249	1319	70
2016-04-02 13:00:00	1249	1339	90
2016-04-03 13:00:00	1249	1319	70
2016-04-04 13:00:00	1239	1319	80

2016-04-05 13:00:00	1249	1309	60	
2016-04-06 13:00:00	1229	1339	110	+
2016-04-07 13:00:00	1229	1329	100	
2016-04-08 13:00:00	1199	1319	120	+
2016-04-09 13:00:00	1239	1339	100	
2016-04-10 13:00:00	1239	1319	80	
2016-04-11 13:00:00	1229	1319	90	
2016-04-12 13:00:00	1229	1309	80	

10030 TOTAL MAINHAUSEN INDUSTRIESTR. 2 63533 MAINHAUSEN

uhrzeit	originaldaten	geschätzte daten	differenz	
2017-01-04 11:00:00	1399	1429	30	
2017-01-05 11:00:00	1409	1429	20	
2017-01-06 11:00:00	1399	1429	30	
2017-01-07 11:00:00	1379	1429	50	
2017-01-08 11:00:00	1419	1319	-100	
2017-01-09 11:00:00	1409	1429	20	
2017-01-10 11:00:00	1399	1439	40	
2017-01-11 11:00:00	1369	1429	60	
2017-01-12 11:00:00	1369	1429	60	
2017-01-13 11:00:00	1359	1429	70	
2017-01-14 11:00:00	1389	1429	40	
2017-01-15 11:00:00	1419	1319	-100	

477 Gnoien, Tessiner Str. 2 Tessiner Str. 2 17179 Gnoien

uhrzeit	originaldaten	geschätzte daten	differenz	
2015-01-05 12:00:00	1319	1389	70	
2015-01-06 12:00:00	1319	1369	50	
2015-01-07 12:00:00	1309	1369	60	
2015-01-08 12:00:00	1269	1379	110	+
2015-01-09 12:00:00	1299	1379	80	
2015-01-10 12:00:00	1299	1379	80	
2015-01-11 12:00:00	1309	1409	100	
2015-01-12 12:00:00	1299	1389	90	
2015-01-13 12:00:00	1289	1369	80	
2015-01-14 12:00:00	1279	1369	90	
2015-01-15 12:00:00	1239	1379	140	+
2015-01-16 12:00:00	1249	1379	130	+

234 AVIA Tankstelle An der Bäderstraße 17 23777 Heringsdorf

uhrzeit	originaldaten	geschätzte daten	differenz
2016-06-01 13:00:00	1379	1429	50
2016-06-02 13:00:00	1329	1429	100
2016-06-03 13:00:00	1359	1419	60
2016-06-04 13:00:00	1369	1409	40
2016-06-05 13:00:00	1389	1419	30

2016-06-06 13:00:00	1389	1419	30
2016-06-07 13:00:00	1399	1419	20
2016-06-08 13:00:00	1389	1429	40
2016-06-09 13:00:00	1359	1429	70
2016-06-10 13:00:00	1389	1419	30
2016-06-11 13:00:00	1369	1409	40
2016-06-12 13:00:00	1409	1419	10

10 AVIA Tankstelle An der B 5 0 25923 Braderup

uhrzeit	originaldaten	geschätzte daten	differenz
2015-08-08 12:00:00	1379	1379	0
2015-08-09 12:00:00	1369	1369	0
2015-08-10 12:00:00	1369	1369	0
2015-08-11 12:00:00	1359	1359	0
2015-08-12 12:00:00	1369	1369	0
2015-08-13 12:00:00	1389	1389	0
2015-08-14 12:00:00	1399	1399	0
2015-08-15 12:00:00	1379	1379	0
2015-08-16 12:00:00	1369	1369	0
2015-08-17 12:00:00	1369	1369	0
2015-08-18 12:00:00	1359	1359	0
2015-08-19 12:00:00	1369	1369	0

6000 Esso Tankstelle DR.-C.-OTTO-STR. 12 0 44879 BOCHUM

uhrzeit	originaldaten	geschätzte daten	differenz
2016-10-10 12:00:00	1309	1349	40
2016-10-11 12:00:00	1289	1339	50
2016-10-12 12:00:00	1269	1339	70
2016-10-13 12:00:00	1279	1349	70
2016-10-14 12:00:00	1279	1349	70
2016-10-15 12:00:00	1379	1349	-30
2016-10-16 12:00:00	1289	1339	50
2016-10-17 12:00:00	1269	1349	80
2016-10-18 12:00:00	1279	1339	60
2016-10-19 12:00:00	1289	1339	50
2016-10-20 12:00:00	1289	1349	60
2016-10-21 12:00:00	1269	1349	80

6500 Georg Behrens Hattinger Str. 122 58332 Schwelm

uhrzeit	originaldaten	geschätzte daten	differenz
2015-10-09 12:00:00	1319	1299	-20
2015-10-10 12:00:00	1349	1299	-50
2015-10-11 12:00:00	1319	1279	-40
2015-10-12 12:00:00	1339	1319	-20
2015-10-13 12:00:00	1339	1279	-60
2015-10-14 12:00:00	1329	1279	-50

2015-10-15 12:00:00	1309	1289	-20
2015-10-16 12:00:00	1309	1299	-10
2015-10-17 12:00:00	1299	1299	0
2015-10-18 12:00:00	1319	1279	-40
2015-10-19 12:00:00	1319	1319	0
2015-10-20 12:00:00	1319	1279	-40

6300 Tucht energy - Heizöl Tel.: 02331/13081 Eilperstraße 129 58091 Hagen

uhrzeit	originaldaten	geschätzte daten	differenz
2016-09-10 12:00:00	1269	1339	70
2016-09-11 12:00:00	1329	1329	0
2016-09-12 12:00:00	1289	1339	50
2016-09-13 12:00:00	1289	1339	50
2016-09-14 12:00:00	1299	1329	30
2016-09-15 12:00:00	1289	1329	40
2016-09-16 12:00:00	1259	1329	70
2016-09-17 12:00:00	1249	1339	90
2016-09-18 12:00:00	1279	1329	50
2016-09-19 12:00:00	1229	1339	110 +
2016-09-20 12:00:00	1289	1339	50
2016-09-21 12:00:00	1259	1329	70

200 Aral Tankstelle Dreilanden 7 25826 Sankt Peter Ording

uhrzeit	originaldaten	geschätzte daten	differenz
2016-10-10 12:00:00	1309	1329	20
2016-10-11 12:00:00	1329	1319	-10
2016-10-12 12:00:00	1319	1329	10
2016-10-13 12:00:00	1329	1339	10
2016-10-14 12:00:00	1329	1349	20
2016-10-15 12:00:00	1399	1379	-20
2016-10-16 12:00:00	1309	1319	10
2016-10-17 12:00:00	1339	1329	-10
2016-10-18 12:00:00	1379	1319	-60
2016-10-19 12:00:00	1329	1329	0
2016-10-20 12:00:00	1319	1339	20
2016-10-21 12:00:00	1309	1349	40

Strategie-Vergleich zwischen naiver und unserer (Routenanalyse)

(Werte in €)

Bertha Benz Memorial Route.csv
kosten: **18.1814**
kosten naiv: **19.3546**

BergTour.csv
kosten: **0.570555**
kosten naiv: **3.987**

BodenseeTour.csv
kosten: **3.94874**
kosten naiv: **6.79364**

DresdenNachBerlin.csv
kosten: **8.99845**
kosten naiv: **10.4964**

IrgendwoImNirgendwo.csv
kosten: **12.7084**
kosten naiv: **11.8903**

IrgendwoImNirgendwo2.csv
kosten: **6.03631**
kosten naiv: **5.8716**

MudersbachNachSiegen.csv
kosten: **1.03585**
kosten naiv: **4.83441**

MunchenNachSiegen.csv
kosten: **39.239**
kosten naiv: **42.5599**

SiegenNachKoln.csv
kosten: **10.9981**
kosten naiv: **11.7669**

StuttgartNachHemmingen.csv
kosten: **1.96293**
kosten naiv: **5.45171**

TourAmMeer.csv
kosten: **12.6252**
kosten naiv: **15.1015**