**Projektarbeit Modul 318 – MyÖV-App**



Abbildung 1: S9 auf dem Weg nach Luzern

Inhalt

[Einleitung 1](#_Toc531703253)

[Aufgabenstellung 2](#_Toc531703254)

[Mockup / Diagramme 2](#_Toc531703255)

[Code Richtlinien 5](#_Toc531703256)

[Code 6](#_Toc531703257)

[Testfälle 10](#_Toc531703258)

[Quellenangabe 13](#_Toc531703259)

## Einleitung

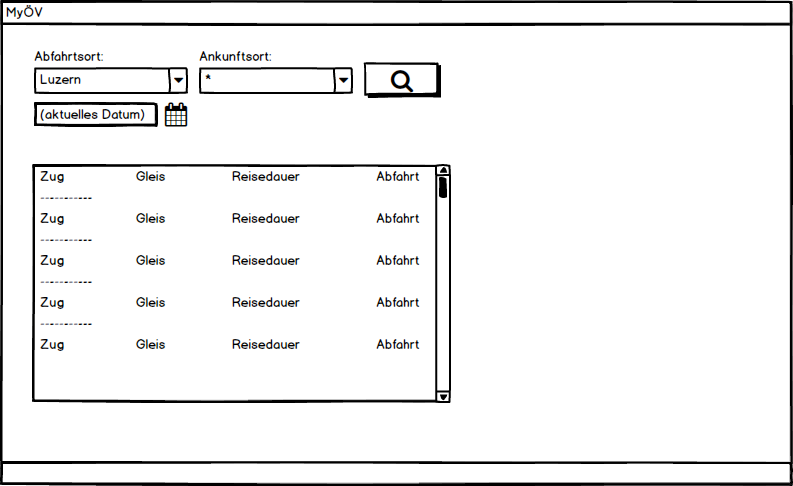
In der heutigen Zeit ist Mobilität nicht mehr weg zu denken. Alle müssen so schnell wie möglich an ihr Ziel kommen und sind wegen jeder kleiner Verspätung und Änderung verärgert. Damit man diese Umstände verhindern kann gibt es zahlreiche Apps welche versprechen genau dieses zu tun. Die meisten sind unübersichtlich und nicht Userfreundlich. Genau das wird mit der MyÖV-App behoben.

## Aufgabenstellung

Im ÜK bekamen wir die Aufgaben innerhalb von 3 Tage mit unserem Wissen eine SBB-App Replikation zu erstellen. Bei unserer App muss der Benutzer die Verbindungen anhand der Start und Endstation suchen können. Es sollen die nächsten 4 Verbindungen angezeigt werden. Ebenfalls muss er sehen können welche Verbindungen ab seiner Station verfügbar sind. Das sind die 3 Grundanforderungen. Auch gibt es noch zwei Kriterien die es erfüllen sollte. Einerseits sollten bei der Eingabe schon Suchresultate erscheinen. Der Benutzer möchte nicht nur die aktuellen Verbindungen sehen, sondern auch welche die zu einem an einem späteren Zeitpunkt fahren. Dafür muss man ein Timepicker sowie ein Datumsfeld einbauen. Nice to have Features sind eine Karte welche anzeigt wo der Zielbahnhof liegt, die nächste Station welche anhand vom Standort ermittelt wird und einen Button mit welchem man die gewünschten Verbindungen versenden kann.

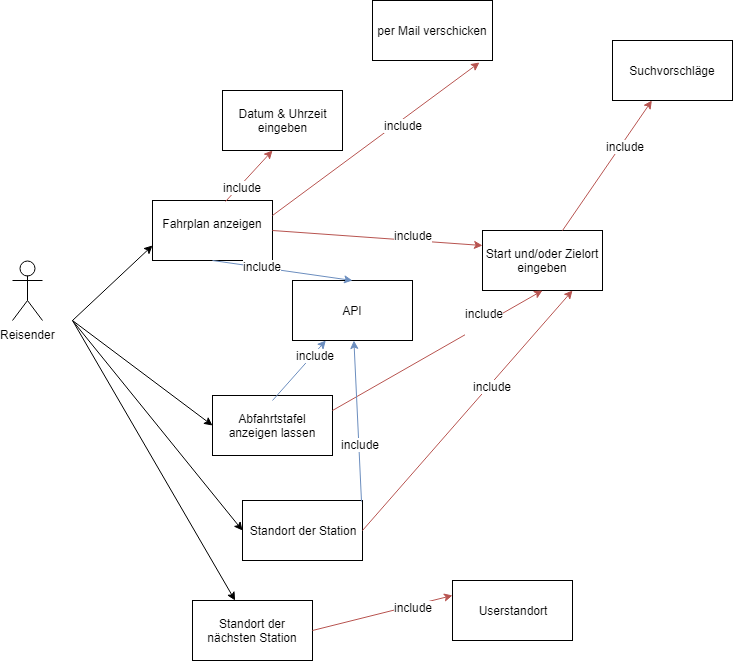
Nebst den Anforderungen an das Programm selber wird noch die Qualität des Codes bewertet. Bewertet wird die: Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Wartbarkeit und Portierbarkeit. Mit Portierbarkeit ist gemeint, ob man die Software einfach installieren und deinstallieren kann und welche Auswirkungen es auf andere Softwares hat.

## Mockup / Diagramme

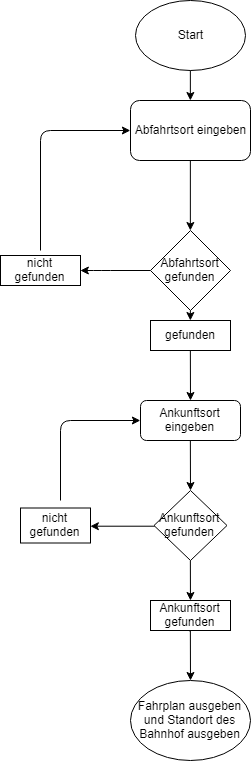
Bevor ich mit dem programmieren beginne, muss ich noch ein paar Diagramme erstellen, damit ich sehe was ich alles machen kann. Als erstes erstellte ich ein GUI-Design an welchem ich mich orientieren werde. 

Ich habe versucht, das Design schlicht und einfach zu halten, sodass man ohne gross zu überlegen weiss, was man machen muss. Im obigen Beispiel sind nur die 3 Grundanforderungen erhalten. Deshalb ist auf der rechten Seite noch Platz. Dort wird die Google-Maps Karte eingefügt welche den aktuellen Standort sowie den Ziel Bahnhof anzeigt. Das ist aber nur der Fall, wenn ich noch genügend Zeit habe. Der Browser wird dann nicht in einem externen Browserfenster geöffnet, sondern im WPF eigenen WebBrowser welchen man in seine Applikation einbauen kann. Der Button um den Fahrplan per Mail zu versenden, kommt und an die Fahrplantafel heran.

Nebst dem GUI-Design haben wir auch noch ein UseCase kreiert. Man sieht also welche Aktion was ganz genau macht.



Im UseCase Diagramm wird beschrieben, was in einem bestimmten Fall passiert oder passieren wird. Nehmen wir an, dass der User den Fahrplan anzeigen möchte. Dafür braucht man das Datum um den Fahrplan der richtigen Zeit anzeigen zu lassen. Ebenfalls muss man den Start sowie Zielort eingeben damit die API eine korrekte Ausgabe ausgibt. Logischerweise muss auch die API included werden. Wenn man den Fahrplan per Mail versenden möchte, muss die Funktion per Mail versenden included werden. Da man nebst dem Fahrplan anzeigen ebenfalls die Abfahrtstafel sehen möchte wurde das ebenfalls ins UseCase Diagramm mit aufgenommen. Die Abfahrtstafel beinhaltet den Startort eingeben sowie die API. Um den Standort der Station herauszufinden, braucht man den Start/Zielort und eine Karte wie z.B Google Maps. Damit die nächste Station angezeigt werden kann, wird der Userstandort benötigt.



Nebst dem UseCase Diagramm wurde auch noch ein Sequenzdiagramm erstellt. Hier sieht man was passiert, wenn ein User seinen Fahrplan/Abfahrtsplan abrufen möchte. Sobald er nach einem Abfahrtsort sucht, wird nachgeschaut, ob die Station vorhanden ist. Falls nicht wird der User aufgefordert eine andere Station einzugeben. Ansonsten geht man weiter zu der Endhaltestelle. Dort ist ebenfalls wieder das gleiche Prinzip welches gebraucht wird. Wenn beide Stationen gefunden wurden, wird der Fahrplan ausgegeben. Falls man an der Abfahrtstafel steht, wird eine Haltestelle verlangt.

## Code Richtlinien

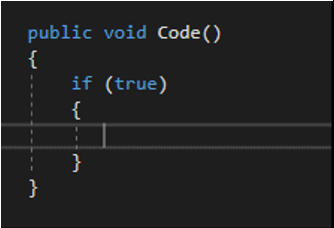
\**Das ganze wurde in einer Gruppenarbeit gelöst\**

Mitglieder:

* Noah Zemp
* Michel Leisibach

Ein Coding Standartdokument zeigt dem Entwickler, wie er seinen Code schreiben muss. Damit nicht jeder Entwickler einen eigenen Style verwendet, halten sich die Programmierer an die Richtlinien. Dadurch ist der Code für einen anderen Entwickler einfacher zu verstehen. Für unser Projekt haben wir eigene Code Richtlinien erstellt.

1. Die Kommentare des Codes werden oberhalb hineingeschrieben. Damit bleibt die Übersichtlichkeit erhalten. Funktionen werden mit der XML Dokumentation von C# kommentiert.
2. Die geschweiften Klammern werden unterhalb platziert.



1. Benennungsrichtlinien

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Object Name | Notation | Length | Plural | Underscores |
| Class Name | PascalCase | 128 | No | No |
| Constructor Name | PascalCase | 128 | No | No |
| Method Name | PascalCase | 128 | Yes | No |
| Method arguments | camelCase | 128 | Yes | No |
| Local variables | PascalCase | 50 | No | No |
| Constants name | PascalCase | 50 | Yes | No |
| Field name | camelCase | 50 | Yes | Yes |
| Properties name | PascalCase | 50 | Yes | No |
| Delegate name | PascalCase | 128 | No | No |
| Enum type name | PascalCase | 128 | Yes | No |

1. Die GUI-Controls werden folgendermassen benannt:

Buttons -> btn\_Funktion

Label -> lbl\_Funktion

TextBoxen -> txt\_Funktion

Combobox -> cmb\_Funktion

Radiobuttons -> rdo\_Funktion

Chekboxen -> chk\_Funktion

1. Die Klassennamen werden mit Grossbuchstaben begonnen.
2. Alle Public Methoden müssen ebenfalls beschrieben werden

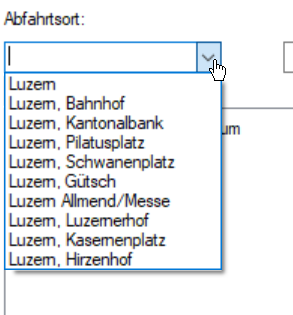
## Code

Für die Projektarbeit haben wir verschiedene Anforderungen erhalten welche umgesetzt werden müssen. Diese werden in 3 verschiedene Kategorien unterteilt. Must/should und nice to have. Die Verschiedenen Anforderungen werden nun Schritt für Schritt erklärt.

**A001**

«Als ÖV-Benutzer möchte ich Start- und Endstation mittels Textsuche suchen können, damit ich nicht alle Stationsnamen auswendig lernen muss.»

Für diese Anforderungen habe ich mit Comboboxen gearbeitet. Sobald der User einen Text eingibt und Enter drückt, werden ihm in der Combobox die verschiedenen Stationen angezeigt welche auf die Eingabe übereinstimmen.



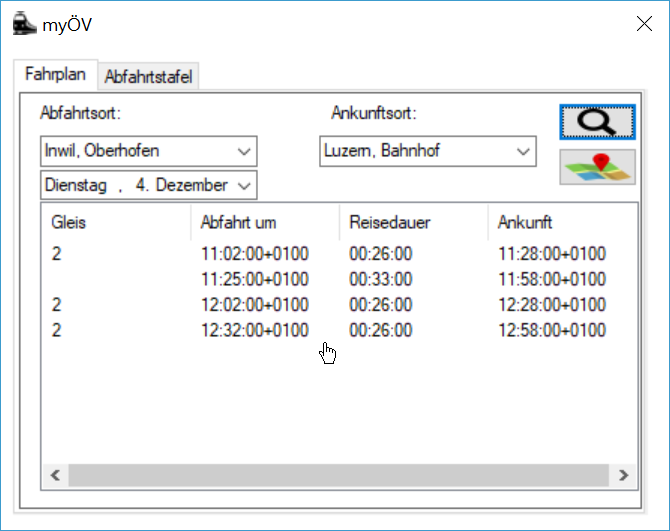
Im obersten Feld sieht man die Eingabe welche der User eingegeben hat. In den nachfolgenden Feldern, werden ihm die Stationen angezeigt welche auf seine Eingabe übereinstimmen. Man kann die Auswahl ebenfalls mit den Pfeiltasten steuern. Einfach die Pfeiltaste nach oben oder unten verwenden. Das gleiche Funktioniert beim Ankunftsort. Wenn man Enter drückt kann es sein, dass noch weitere Stationen erscheinen. Sobald man die entsprechende Station gefunden hat, kann man auf Tab drücken und man ist im nächsten Feld.



**A002**

«Als ÖV-Benutzer möchte ich die aktuellen, d.h mindestens die nächsten vier bis fünf Verbindungen zwischen den beiden gefundenen und ausgewählten Stationen sehen, damit ich weiss wann ich zur Station muss, um den für mich idealen Anschluss zu finden.

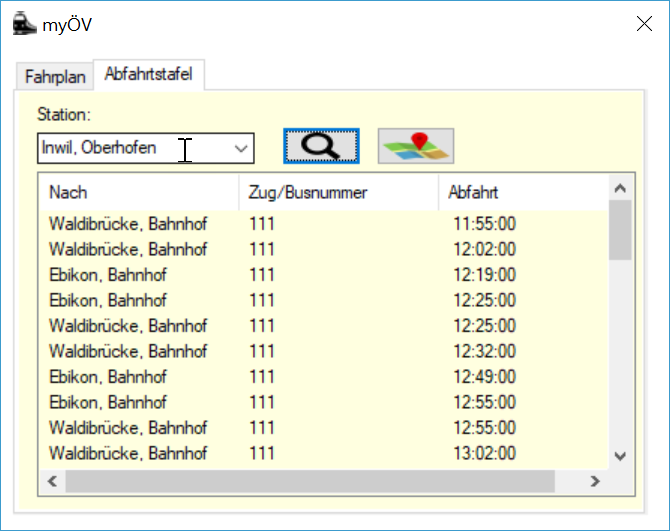
Damit der Benutzer die Verbindungen suchen kann, muss er Start sowie End Station eingegeben haben. Ansonsten erhält er eine Meldung welche ihm Mitteilt, dass die Station nicht gefunden wurde.



Sobald er die Lupe oben rechts geklickt hat, wird die Suche gestartet. Wenn er die entsprechende Verbindung gefunden hat, gibt er die nächsten vier Verbindungen aus. Aus der Anzeige kann der Benutzer das Gleis, die Abfahrtszeit, die Reisedauer sowie die Ankunftszeit entnehmen. Damit der User den entsprechenden Fahrplan angezeigt bekommt, braucht es folgenden Code.

**A003**

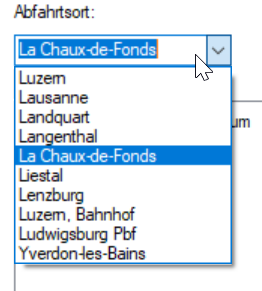
«Als ÖV-Benutzer möchte ich sehen, welche Verbindungen ab einer bestimmten Station vorhanden sind, damit ich bei mir zuhause eine Art Abfahrtstafel haben kann.»

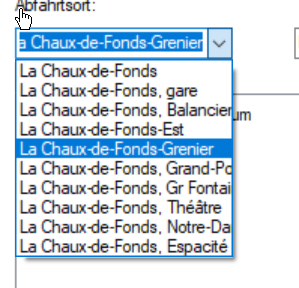


Hier sieht man die Abfahrtstafel von der Station Inwil Oberhofen. Man sieht eine Liste wann welche Linie wohin fährt. Damit man suchen kann muss man wie gewohnt auf die Lupe klicken. Das Prinzip vom Code ist genau das gleiche wie beim Fahrplan. Deshalb wird es hier nicht mehr gezeigt.

**A004**:

«Als ÖV-Benutzer möchte ich, dass schon während meine Eingabe erste Suchergebnisse erscheinen, damit ich effizienter nach Stationen suchen kann.

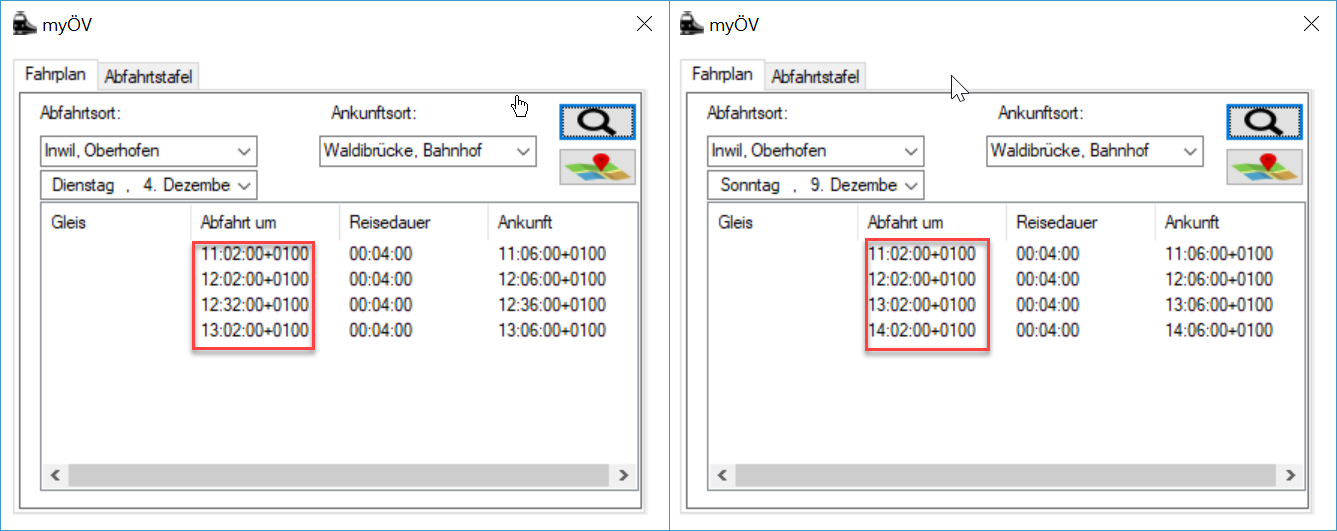




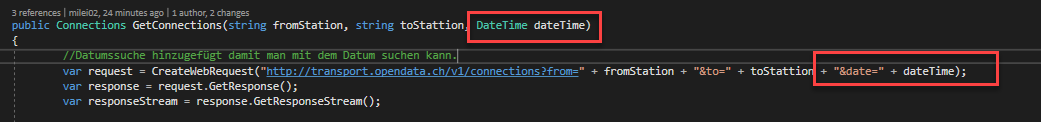
Wie haben am Anfang ein kleines L eingeben. Dann sind, nachdem man **Enter** gedrückt hat, die verschiedenen Städte erschienen. Hier habe ich mit den **Pfeiltasten** La Chaux-de-Fonds ausgewählt und anschliessend **Enter** gedrückt. Dann sind weitere Stationen vorgeschlagen worden die La Chaux-de-Fonds im Namen haben. Sobald man die Station gefunden hat, kann man mit dem **Tabulator** ins nächste Feld springen und dort weitermachen.

**A005**

«Als ÖV-Benutzer möchte ich nicht nur aktuelle Verbindungen suchen können, sondern auch solche zu einem beliebigen anderen Zeitpunkt, damit ich zukünftige Reisen planen kann.»



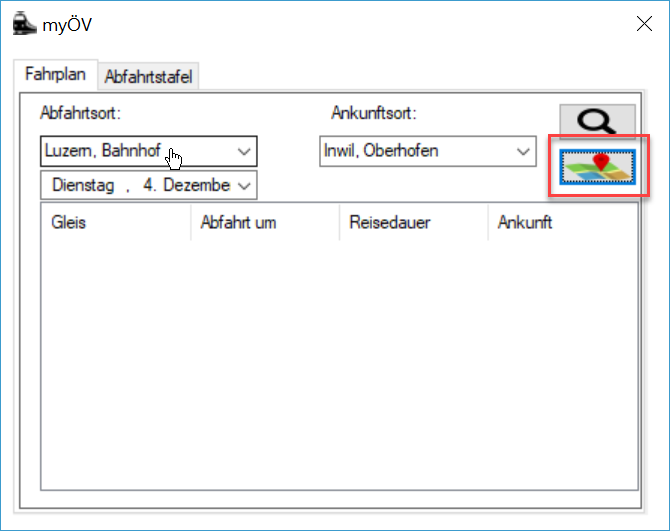
Hier sieht man den Fahrplan für die Strecke Inwil Oberhofen an einem Dienstag und an einem Sonntag. Man sieht, dass die Busse am Sonntag nur im 1 Stunden Takt verkehren. Am Dienstag fährt er über den Mittag im 30 Minuten Takt. Unter dem Abfahrtsort kann der User das Datum einstellen. Damit das alles funktioniert, musste ich der API einen weiteren Parameter mitgeben und den Webrequest erweitern.



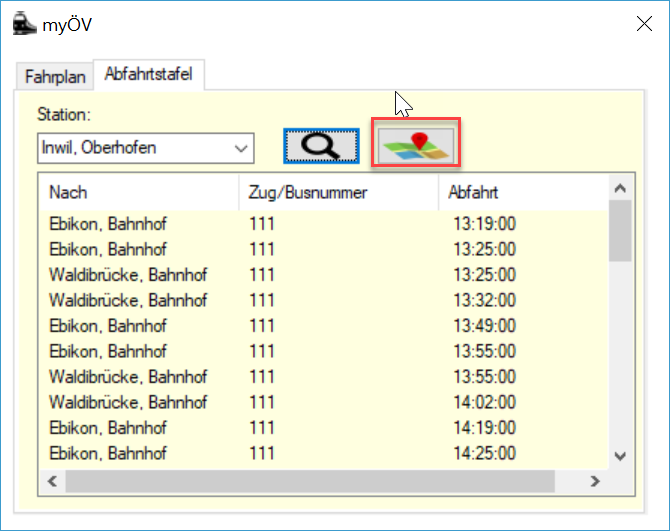
Hier sieht man, dass ich den Webrequest mit dem Datum erweitert habe. Ich möchte also nicht nur die Routen die mit den Start/Endstationen übereinstimmen, sondern nun soll auch noch das Datum stimmen.

**A006**

«Als ÖV-Benutzer möchte ich sehen, wo sich eine Station befindet, damit ich mir besser vorstellen kann, wie die Situation vor Ort aussieht.» Bei meiner Applikation habe ich es folgendermassen gelöst. Bei der Fahrplansuche wird jeweils die Endstation ausgegeben, da ich davon ausgehe, dass der User die Startstation schon kennt. Unter dem Suchbutton gibt es noch einen Knopf welcher eine Karte abbildet.



Wenn man auf die Karte klickt, wird man auf Google Maps weitergleitet und sieht wie es dort vor Ort aussieht. Beim Fahrplan wird also die Endstation ausgegeben. Bei der Abfahrtstafel habe ich es so eingerichtet, dass die Station auf Google Maps angezeigt wird, welche man gesucht hat.



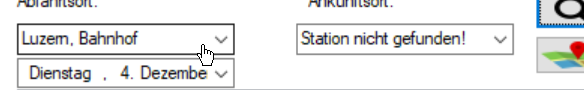
## Testfälle

Damit der User möglichst wenig Fehler hat, werden verschiedene Testfälle durchlaufen um diese zu verhindern. In der myÖV-App wurden ebenfalls verschiedene Testfälle durchgeführt.

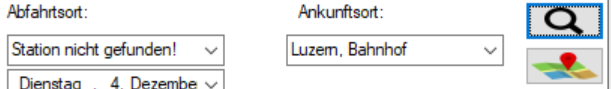
**Testfall 1:**

Es kann gut sein, dass man ausversehen ein Feld leer lässt. Bei der Stationen suche ist dies wahrscheinlicher als beim Abfahrtsplan. Dafür habe ich alle Testfälle ein Beispiel gemacht:

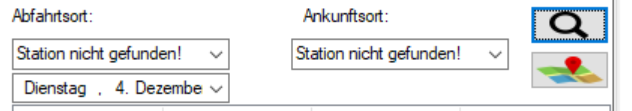
*Beispiel 1:*



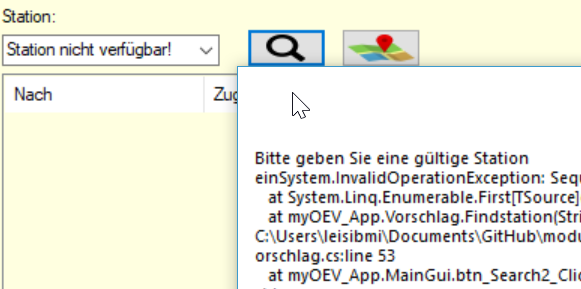
*Beispiel 2:*



*Beispiel 3:*



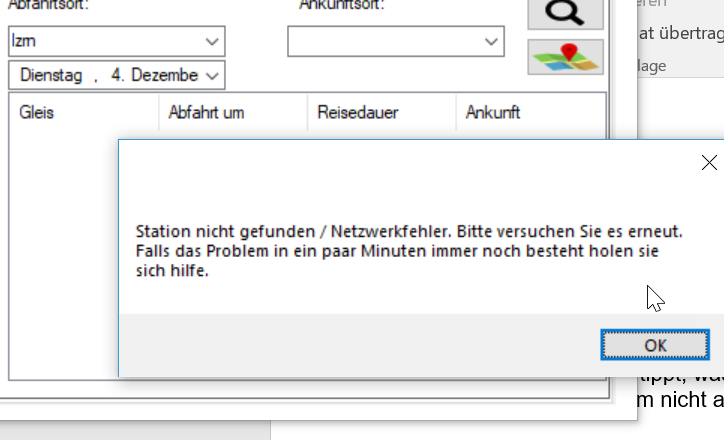
*Beispiel 4:*



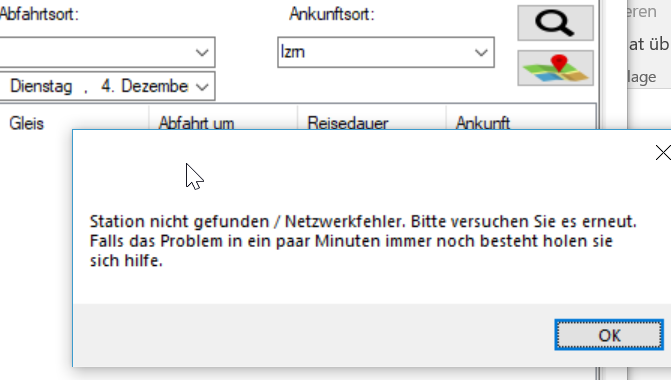
**Testfall 2:**

Wenn sich ein User vertippt, was ab und zu mal vorkommen kann, ist es ebenfalls wichtig, dass sich das Programm nicht abstürzt.

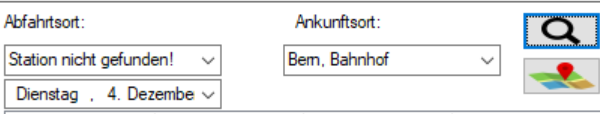
*Beispiel 1:*



*Beispiel 2:*



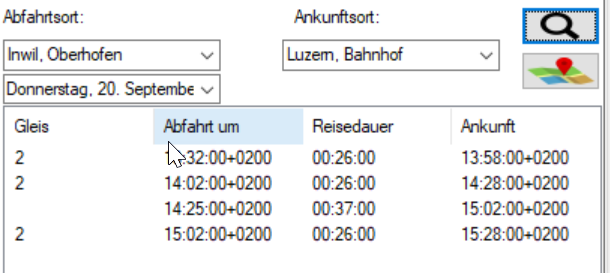
*Beispiel 3:*



Hier habe ich am Anfang einfach irgendwelche Buchstaben eingegeben.

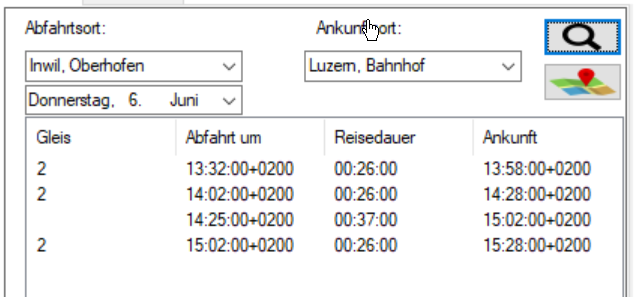
**Testfall 3:**

Im dritten Testfall habe ich ein Datum genommen welches schon ein paar Monate zurück liegt. Das konnte das Programm problemlos handhaben und es gab kein Error.



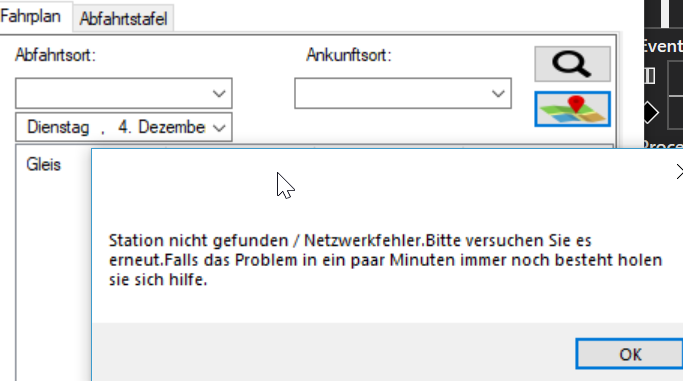
**Testfall 4:**

Da ich jetzt probiert habe, wie das Programm reagiert, wenn man ein Datum nimmt welches in der Zukunft liegt. Auch das hat ohne Probleme geklappt.



**Testfall 5:**

Wenn der User auf das MapSymbol klickt ohne, dass eine Station ausgewählt wurde kommt eine Fehlermeldung.



Das gleiche passiert auch bei der Abfahrtstafel.

## Quellenangabe

Abbildung 1:

<https://static.az-cdn.ch/__ip/JzqE8ZPaEYWbxXe-hXSMYkimduc/3495894c95101e051740ac4136f2b8b4edc453d2/n-large-16x9-far>