ük 318

Die ÖV-Applikation



Gerry Kushiv Sroy, 27.11.18

Inhalt

[1.0 Einführung 2](#_Toc531687798)

[2.0 Use Case und Aktivitätsdiagramm 2](#_Toc531687799)

[3.0 Funktionalitäten 3](#_Toc531687800)

[4.0 Mockup 4](#_Toc531687801)

[5.0 Systemtests 5](#_Toc531687802)

[6.0 Installationsanleitung 7](#_Toc531687803)

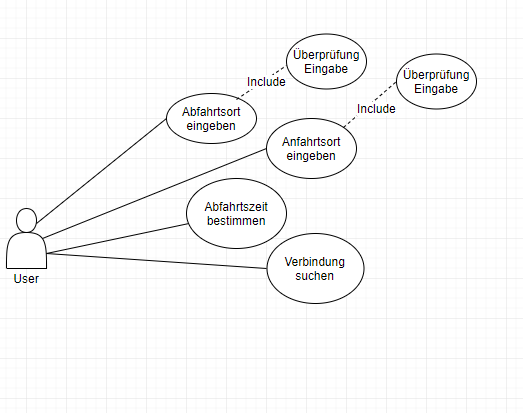
[7.0 Anhang 9](#_Toc531687804)

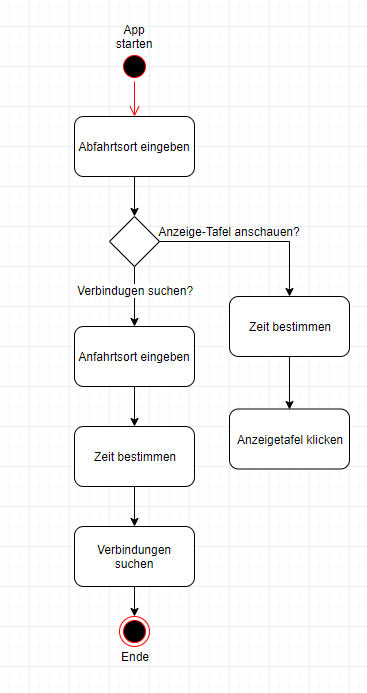
# Einführung

In dieser Dokumentation werde ich aufzeigen was ich im ÜK 318 als Projektarbeit gemacht habe. Die Aufgabestellung bestand daraus nach Vorgaben eine ÖV-Applikation zu programmieren. Diese Applikation sollte dabei benutzerfeundlich sowie die Kundenanforderungen erfüllen, hier wird es die Aufgabenstellung sein. Auch sollte die Applikation so wenig Fehler wie möglich haben um den Benutzer die Anwendung so einfach wie möglich zu machen.

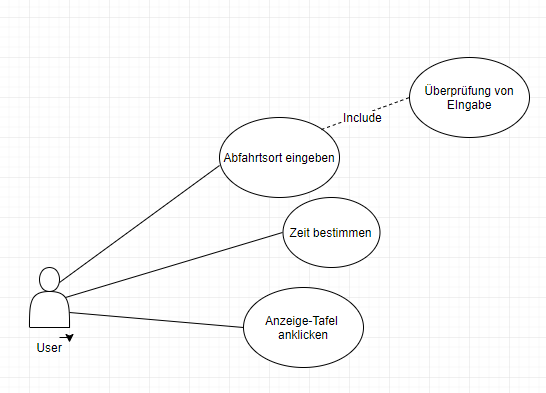
# 2.0 Use Case und Aktivitätsdiagramm

Hier habe ich meine Use Cases sowie mein Aktivitätsdiagramm für mein Projekt.

User Case für Start- und Endstation Verbindungsuche: Aktivitätsdiagramm:



User Case für Abfahrtstafel:



# Funktionalitäten

Hier habe ich alle die Anforderungen aufgelistet und beschrieben welche ich wie umgesetzt habe.

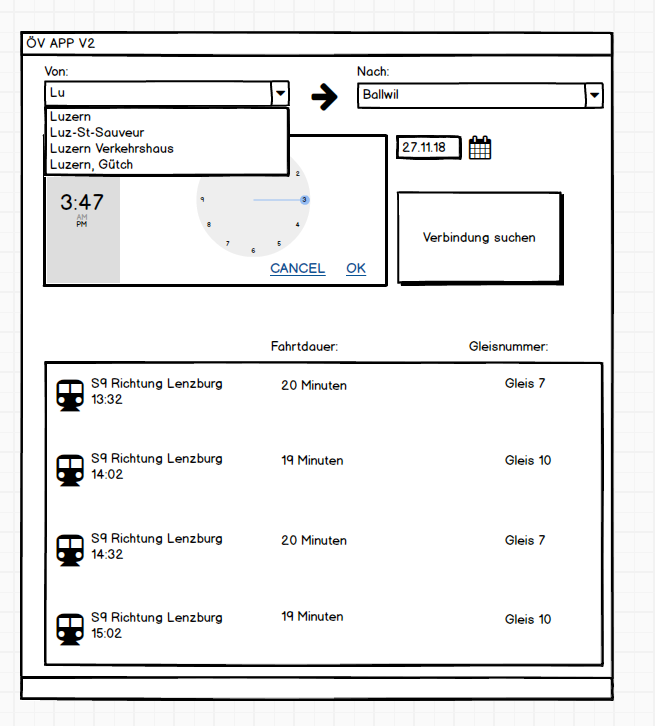
|  |  |  |  |
| --- | --- | --- | --- |
| ID | Beschreibung | Umgesetzt? | Beschreibung Umsetzung |
| A001 | Als ÖV-Benutzer möchte ich Start- und Endstation mittels Textsuche suchen können, damit ich nicht alle Stationsnamen auswendig lernen muss. | Ja | Umgesetzt mit TextBoxen,  Abfahrt und Anfahrtsort eingeben |
| 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 erwischen. | Ja | Umgesetzt mit Button,  Abfahrtsort und Anfahrtsort auswählen und danach mit Button-Klick Verbindungen suchen. |
| 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. | Ja | Umgesetzt mit Button,  Abfahrtsort und Anfahrtsort eingeben und mit Buttonklick die nächsten Verbindungen von der gesuchten Station suchen. |
| A004 | Als ÖV-Benutzer möchte ich, dass schon während meiner Eingabe erste Such-Resultate erscheinen, damit ich effizienter nach Stationen suchen kann. | Ja | Umgesetzt mit Listbox,  Bei Eingabe in TextBox wird gleichzeitig bei der Eingabe die Vorschläge angezeigt von der Listbox. |
| 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. | Teilweise | Umgesetzt mit DateTimePicker,  Zeit bestimmen und dann Verbindungen suchen, alle Verbindungen ab der gesuchten Zeit werden angezeigt.  Nur teilweise umgesetzt, weil Datum fehlt. |
| A006 | Als ÖV-Benutzer möchte ich sehen, wo sich eine Station befindet, damit ich mir besser vorstellen kann, wie die Situation vor Ort aussieht. | Nein |  |
| A007 | Als ÖV-Benutzer möchte Stationen finden, die sich ganz in der Nähe meiner aktuellen Position befinden, damit ich schnell einen Anschluss erreichen kann. | Nein |  |
| A008 | Ich möchte meine gefundenen Resultate via Mail weiterleiten können, damit auch andere von meinen Recherchen profitieren können. | Nein |  |

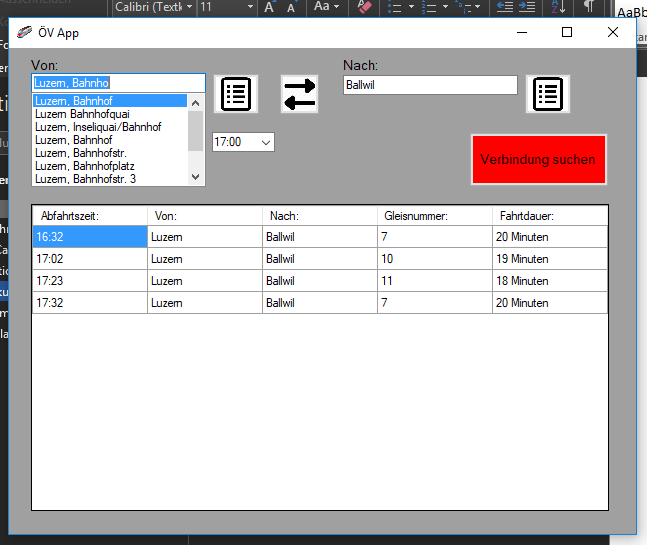
Die A005 habe ich nur teilweise umgesetzt, weil ich nur die Zeit als Eingabe brauchen wollte. A006 bis A008 setzte ich nicht um, weil mir die Zeit fehlte.

Bei A004 kann man mit den Pfeiltasten in der Liste navigieren und mit Enter sowie TAB die gesuchte Station dann selektieren, was aber nicht funktioniert ist, dass man mit der Maus die gesuchte Station nicht anklicken kann.

# Mockup

Für das GUI der ÖV-App hatte ich mir schon viele Gedanken gemacht. Die Applikation sollte schlicht, einfach und selbsterklärend sein. So habe ich ein Mockup erstellt welches etwa diesen Anforderungen entspricht. Ich habe auch das umgesetzte GUI zum Vergleich auf dieser Seite eingefügt.

**Mockup:**

**Umgesetztes GUI:**

# Systemtests

**Testfall 1**

User möchte die Stationen Luzern, Bahnhof sowie die Endstation Zürich HB suchen.

|  |  |  |
| --- | --- | --- |
| **Schritt** | **Aktivität** | **Bild & erwartetes Resultat** |
| 1 | Abfahrtsort Luzern, Bahnhof in TextBox eingeben. | Luzern, Bahnhof wird in der TextBox erkannt. |
| 2 | Anfahrtsort Zürich HB in TextBox eingeben. | Zürich HB wird in der TextBox erkannt. |

**Testfall 2**

User möchte von Root D4, Oberfeld nach Ebikon, Bahnhof ab einer bestimmten Zeit reisen.

|  |  |  |
| --- | --- | --- |
| **Schritt** | **Aktivität** | **Bild & erwartetes Resultat** |
| 1 | Abfahrtsort Root D4, Oberfeld in TextBox eingeben. | Root, Oberfeld wird in der TextBox erkannt. |
| 2 | Anfahrtsort Ebikon, Bahnhof in TextBox eingeben. | Ebikon, Bahnhof wird in der TextBox erkannt. |
| 3 | Verbindungen suchen mit Klick auf Button. | Mit dem Klick auf den Button «Verbindungen suchen» werden anschliessend die Verbindungen zu der aktuellen Zeit angezeigt. |

**Testfall 3**

User möchte von der Station Rotkreuz, Bahnhof Süd alle möglichen Verbindungen ab einer bestimmten Zeit suchen.

|  |  |  |
| --- | --- | --- |
| **Schritt** | **Aktivität** | **Bild & erwartetes Resultat** |
| 1 | Abfahrtsort Rotkreuz, Bahnhof Süd in TextBox eingeben. | Rotkreuz, Bahnhof Süd wird in der TextBox erkannt. |
| 2 | Anzeigetafel Button klicken | Klick auf Anzeigetafel führt dazu, dass alle möglichen Verbindungen von der gesuchten Haltestelle angezeigt wird. |

**Testfall 4**

User versucht einen Abfahrtsort zu suchen aber er kennt nur den Namen vom Ort und nicht die Station.

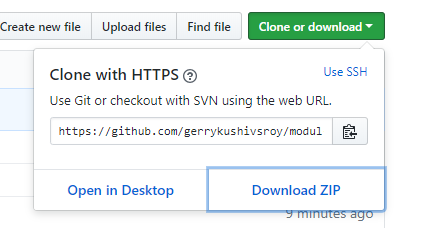
|  |  |  |
| --- | --- | --- |
| **Schritt** | **Aktivität** | **Bild und erwartetes Resultat** |
| 1 | Abfahrtsort Ballwil in TextBox eingeben | Bei der Eingabe erscheinen alle Vorschläge welche mit Ballwil anfangen. |

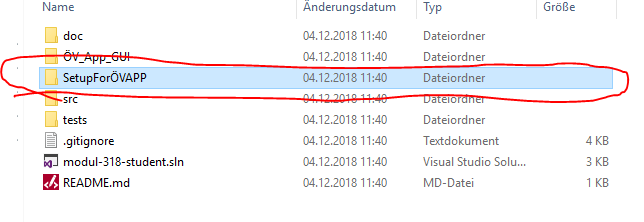
**Testfall 5**

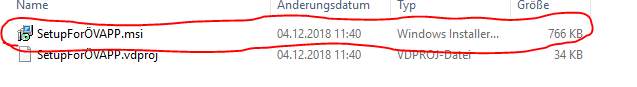
User möchte ab 12:00 von Ballwil nach Emmenbrücke reisen und sucht die möglichen Verbindungen ab dieser Zeit.

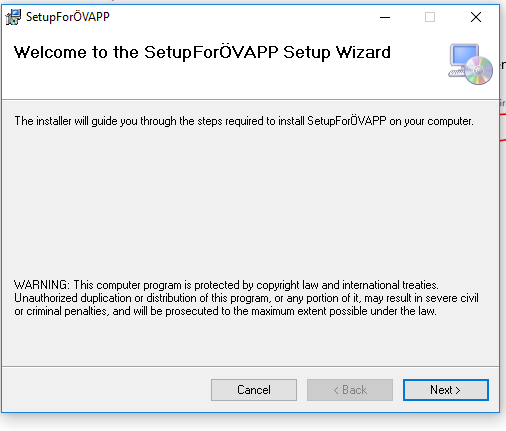
|  |  |  |
| --- | --- | --- |
| **Schritt** | **Aktivität** | **Bild und erwartetes Resultat** |
| 1 | Abfahrtsort Ballwil in TextBox eingeben | Abfahrtsort Ballwil wird erkannt |
| 2 | Anfahrtsort Emmenbrücke in TextBox eingeben. | Anfahrtsort Emmenbrücke wird erkannt |
| 3 | Zeit auf 12:00 setzen | Zeit korrekt setzten |
| 4 | Verbindungen suchen mit Buttonklick auf Verbindungen suchen | Es werden anschliessend die letzte sowie die nächsten Verbindungen angezeigt. |

# Installationsanleitung

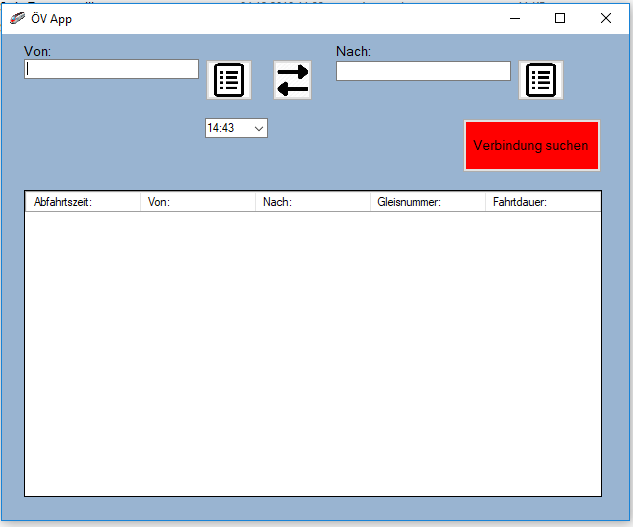
Für das Installieren meiner ÖV-App kann man mein Repository herunterladen und anschliessend entpacken.

Danach öffnet man den entpackten Ordner und anschliessend geht man zum Setup Ordner.

Beim Setup Ordner angekommen kann man einfach das Setup ausführen und anschliessend die Schritte im Setup befolgen.



Wenn man die Installation abgeschlossen hat kann das Programm starten und auch benutzen.



# Anhang

Hier habe ich noch als Anhang meine Programmier-Richtlinien die ich für dieses Projekt definiert habe.

**Programmier-Richtlinien**

Wieso diese Richtlinien?

Der Zweck der Richtlinien ist es Erleichterung der Arbeit von Teammitglieder im Projekt. Unter Erleichterung versteht man die Lesbarkeit, Verständlichkeit und Wartbarkeit vom Code. Einhaltung vom vorgängig definierten Rechtlinien für variablen, Prozeduren und Klassen Namen, kann Lesbarkeit und Wartbarkeit des Codes wesentlich verbessern. Während der Wartung ist die Einhaltung eines definierten Programmstiels noch wichtiger als während der Entwicklung. Ein Programmiersteil sollte nicht unbedingt wie eine programmier-Richtlinien ausgelegt werden. Verstösse dagegen sollten erlaubt sein, sofern diese gut begründet sind.

Unsere Richtlinien

* Immer bevor man eine Klammer nach einem Funktionsaufruf setzt, einen Abstand davor setzen z.B. / MachEineRolle (a)
* Am Anfang und am Ende einer Klammer wird *kein* Abstand gemacht. z.B. so: MachEinRad (a) und nicht so: machEinrad ( a )
* Sollte man einen generischen Typ nutzen, so wird der Datentyp ohne Abstand direkt angehängt z.B. <int>List Beispiel = new List<int>
* Eine geschweifte klammer bekommt immer eine einzelne Zeile. So fängt man immer eine neue Zeile an sobald man eine geschweift klammer platziert hat.

if (a)

{

code ();

code ();

}

if (a) {

code ();

code ();}

* Klassennamen werden immer mit einem sinnvollen Namen und mit einem Grossbuchstaben definiert.

Public Class Car -> Klasse über Autos

Public Class tastatur -> Klasse über Mäuse

* Die Funktion sollte nur das machen, wie sie auch heisst.

Private void getAuto()

{

String hallo;

Return Hallo;

}

Private void GetAuto()

{

String auto;

Return auto;

}

* Funktionen sollten so kurz wie möglich gemacht werden. (max. 20 Zeilen)

Private void getAuto()

{

String auto;

Return auto;

}

Private void getAuto()

{

String auto;

String auto;

String auto;

String auto;

String auto;

String auto;

String auto;

String auto;

…

Return auto;

}

* Methoden sowie Funktionen fangen immer mit einem Grossbuchstaben an.

Private void GetAuto()

{

String auto;

Return auto;

}

Private void GETAUTO()

{

String auto;

Return auto;

}

* Kommentare möglichst kurz und sinnvoll schreiben.

//Hier habe ich einen leeren string initialisiert

//Autocomplete wird erstellt

* Variabeln werden immer kleingeschrieben.

String Eingabe = «»;

String input = «»;

* Code darf keine Warnings enthalten.

Am Schluss dürfen auf der Fehlerlistekeine Warnings enthalten sein.

* Code wird «auf Englisch geschrieben» d.h. dass Methoden, Variabeln und Funktionen bekommen einen Englischen Namen. (#Kommentare dürfen auf Englisch verfasst werden.)

string Bnutzereingabe = ComboBox\_Input.text;

string userinput = ComboBox\_Input.text;

* Variabeln müssen sinnvoll benannt werden. So soll jeder der sie zum ersten Mal sieht direkt wissen für was diese Variabel eingesetzt werden.

int x = ID;

int userid = ID;