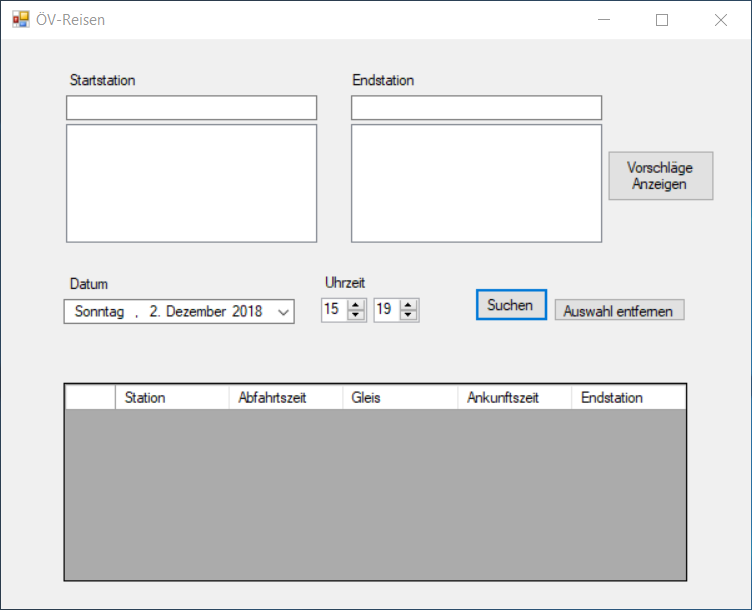
ÖV-Applikation



ÜK-318

Levin Joller

27. November 2018

Inhaltsverzeichnis

[Einleitung 3](#_Toc531600364)

[Zweck der Dokumentation 4](#_Toc531600365)

[Mockup 4](#_Toc531600366)

[Umsetzung 5](#_Toc531600367)

[Programmierrichtlinien 5](#_Toc531600368)

[Eigene Programmierrichtlinien 5](#_Toc531600369)

[Allgemein 5](#_Toc531600370)

[Variablen 5](#_Toc531600371)

[Properties 6](#_Toc531600372)

[Methoden 6](#_Toc531600373)

[Klassen 6](#_Toc531600374)

[Kommentare 7](#_Toc531600375)

[GUI-Controls 7](#_Toc531600376)

[Funktionen der Applikation 7](#_Toc531600377)

[Veranschaulichung mit Diagrammen 8](#_Toc531600378)

[Anwendungsfalldiagramm 8](#_Toc531600379)

[Aktivitätsdiagramm 9](#_Toc531600380)

[Testfälle 10](#_Toc531600381)

[Installationsanleitung 11](#_Toc531600382)

[Quellen 12](#_Toc531600383)

# Einleitung

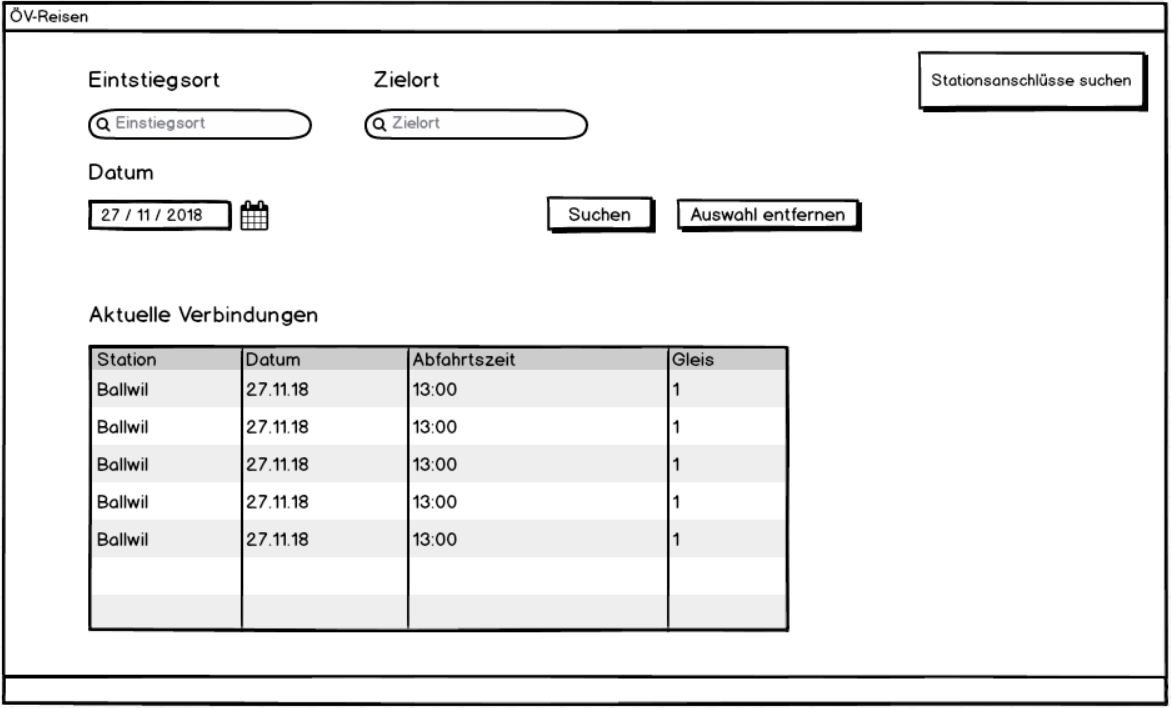
Ziel der Arbeit ist es, eine Applikation zur Anzeige von Informationen über den aktuellen Bahnverkehr nach Vorgaben zu schreiben. Dafür verwende ich die Programmiersprache C# und die dazu passende Entwicklungsumgebung namens "Visual Studio" von Microsoft.

# Zweck der Dokumentation

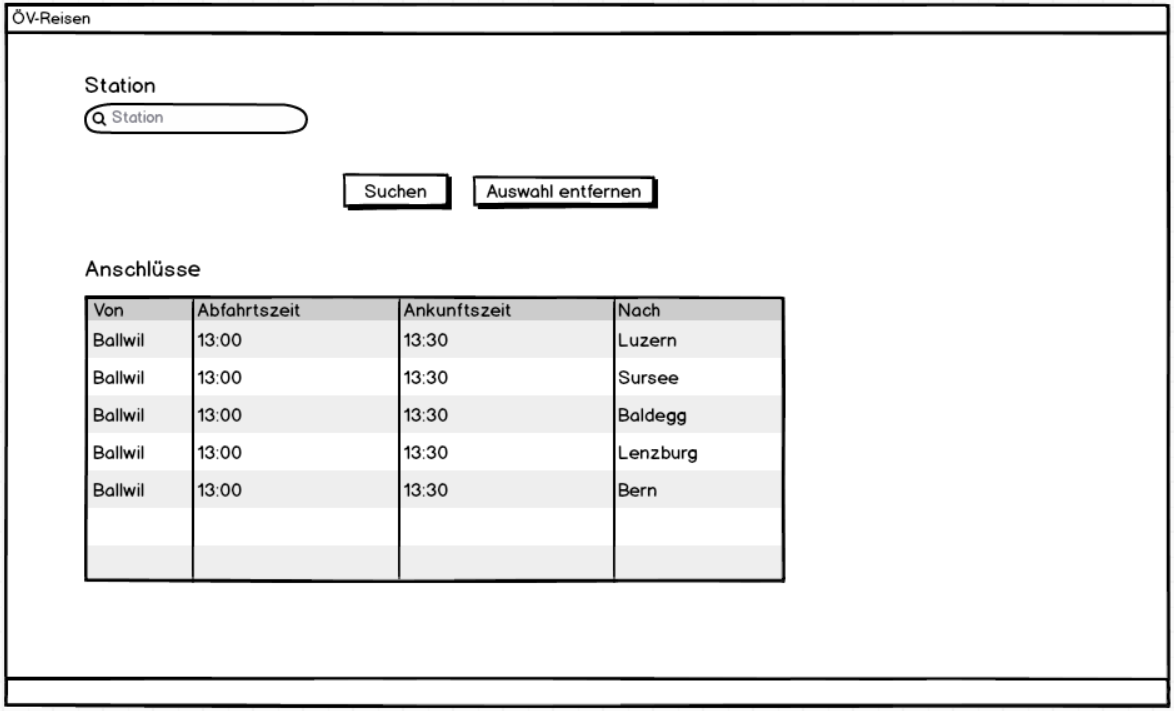
Man schreibt eine Dokumentation zu einer Applikation, damit man an einem späteren Zeitpunkt ohne grossen Zeitaufwand sich wieder einarbeiten kann. Festgelegte Regeln und Abmachungen sollte man deshalb in einer solchen Dokumentation festhalten. Es hat auch der Vorteil, dass wenn man mit mehreren Personen an einem Projekt arbeitet jeder seine Arbeit auf dieselbe Art vollbringt. Dies macht den Code und die Gestaltung einheitlich und somit sauberer. Zudem erleichtert es die Weitergabe von Projekten, da man alle zu beachtende Punkte durch die Dokumentation mitgibt und somit die Gefahr, dass man etwas vergisst verringert.

# Mockup

Dies sind meine Vorstellungen vor der Umsetzung des Projekts, damit ich einen Anhaltspunkt bei der Umsetzung habe. Ich habe versucht die grafische Benutzeroberfläche so schlicht wie möglich zu gestalten, aber immer noch selbsterklärend.



Der Benutzer sollte seinen Einstiegsort, also seinen Abfahrtsort und seinen Zielort eintragen können und die im Zeitrahmen befindende Verbindungen anzeigen lassen. Zudem sollte man den gewünschten Zeitpunkt auswählen können. Die Möglichkeit die Auswahl per Knopfdruckt zu entfernen sollte die Bedienung erleichtern. Mit dem Knopf "Stationsanschlüsse suchen" wechselt man zum zweiten Fenster.



Mit dem zweiten Interaktionsfenster kann der Benutzer Anschlussverbindungen von einer angegebenen Station sehen. Das Fenster sollte gleichzeitig und unabhängig vom ersten Fenster existieren, aber beim Schliessen des ersten Fensters auch geschlossen werden.

# Umsetzung

## Programmierrichtlinien

Der Zweck der Programmierrichtlinien ist es, dass der Programmcode Einheitlich ist und somit Änderungen auch von anderen Personen einfach vorzunehmen sind. Damit ein Caos im Code vermieden wird und die Arbeit im Team vereinfacht braucht man solche Richtlinien. Zudem sollen auch Personen, die den Code selbst nicht geschrieben haben den Code mit wenig Aufwand verstehen können. Damit wir zusätzlicher Zeitaufwand zum Verstehen des Codes vermieden.

Folgende Punkte sollten beim Code schreiben beachtet werden:

Lesbarkeit, Wartbarkeit, Übersichtlichkeit, Verständlichkeit

## Eigene Programmierrichtlinien

Autoren: Tuan Binh Tran, Dominic Pohland, Levin Joller - 28.11.2018

### Allgemein

Es werden Tabs für Einrückungen verwendet anstatt Spaces (Leertaste). Es wird alles in Englisch geschrieben, mit Ausnahme der Kommentare (Deutsch)

### Variablen

Variablen verwenden camelCase und werden gleich beim Deklarieren initialisiert. Variablen verwenden keine Abkürzungen. (usrGrp -> userGroup)

**Beispiel:**

string namingConventions = null;

int number = null;

**falsches Beispiel:**

string NamingConventions;

int Number;

### Properties

Properties sind in PascalCase geschrieben und sind plural.

**Beispiel:**

public string FirstName { get; set;}

**falsches Beispiel:**

public string firstname {get; set;}

### Methoden

Methoden werden in PascalCase geschrieben. Methoden fangen mit einem Verb an und beschreibt deren Funktion. Brackets fangen auf einer neuen Linie an.

**Beispiel:**

public string DoSomething ()

{

//code

}

**falsches Beispiel:**

public string haus

{

//code

}

### Klassen

Klassen fangen mit einem Grossbuchstaben an und sind Singular.

**Beispiel:**

public class Tree

{

//Properties

}

**falsches Beispiel:**

public class Trees

{

//Properties

//...

}

### Kommentare

Kurze und verständliche Kommentare vor Methoden, WENN Methodename nicht selbsterklärend ist. Kommentare sind in Deutsch geschrieben. Nach einem Kommentar ist KEINE Leerzeile.

**Beispiel:**

// Gibt den string in UPPERCASE zurück.

DoSomething (string text) {

// code

}

**falsches Beispiel:**

/\* Gibt den string in UPPERCASE zurück. \*/

DoSomething (string text) {

/\* code \*/

}

### GUI-Controls

Folgenden Controls starten mit folgendem Präfix und danach mit einem Grossbuchstaben.

* Label = lbl...
* Button = btn...
* ComboBox = cbo...
* TextBox = txt...
* ListBox = lst...
* PictureBox = pic...
* Timer = tmr...
* RadioButton = opt...
* CheckBox = chk...
* GroupBox = grp...
* Panel = pnl...

## Funktionen der Applikation

**Aufgabe 1:**

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

Umsetzung:

Ich habe zwei Eingabefelder und einen grossen Ausgabebereich erstellt. Damit man sich die Namen der Stationen nicht genaustens merken muss, kann man mithilfe eines Knopfes Vorschläge in den untern Felder unterhalb der Eingabe anzeigen lassen. Dafür sollte man mindesten den Anfangsbuchstaben seiner Start- und Zielstation kennen. Um die Bedienung zu erleichtern, habe ich einen Knopf hinzugefügt, der alle Eingabeelemente und Ausgabeelemente bereinigt um von neuem anzufangen.

**Aufgabe 2:**

Dfd

Umsetzung:

Die gefunden Verbindungen lasse ich in einem "DataGridView" ausgeben. Zur Übersicht habe ich die einzelnen Zeilen passend beschriftet. Um die von der API erhaltene Zeit in "UNIX-Timestamp" korrekt darzustellen, habe ich dafür eine Methode erstellt, die die Angabe in "System.DateTime" umwandelt.

**Aufgabe 3:**

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

Umsetzung:

Damit es übersichtlich bleibt, habe ich das Suchen der Verbindungen und das Anzeigen der Anschlüsse mithilfe von Tabs getrennt. Zusätzlich überprüfe ich, auf welchem Tab ich mich als Benutzer befinde, um den "AcceptButton" auf die aktuelle Anzeige zusetzen.

**Aufgabe 5:**

Umsetzung:

Damit der Benutzer das Datum und die Uhrzeit der gewünschten Abfahrt anzugeben kann, habe ich für das Datum das "DateTimPicker" Element hinzugefügt und für die Uhrzeit zwei "NummericUpDown" Elemente erstellt. Im Hintergrund habe ich dafür eine neue API-Abfrage erstellt um auf die zusätzlichen Bedingungen einzugehen.

**Allgemein:**

Im Projekt konnte ich Errors und Abstürze soweit abfangen, aber ich konnte wegen mangelnder Zeit nicht spezifische Meldungen auf die Fehler ausgeben. Leider konnte ich nur allgemeine Meldungen anzeigen lassen.

## Veranschaulichung mit Diagrammen

### Anwendungsfalldiagramm

Dieses Use-Case Diagramm veranschaulicht die Interaktionen des Benutzers mit dem zu entwickelten System.

### Aktivitätsdiagramm

Das Aktivitätsdiagramm beschreibt die Abläufe mit der Benutzung der Applikation. Entscheidungen bestimmen den Weg/Ablauf, den man geht. Das heisst, es werden alle möglichen Wege, die man bei dieser Applikation beschreiten kann aufgezeichnet.

# Testfälle

# Installationsanleitung

# Quellen

<https://coderwall.com/p/e8rzuq/how-to-convert-a-unix-timestamp-to-a-net-system-datetime-object>