Software-dokumentation



ÖV-Applikation

Inhaltsverzeichnis

[2 Vorwort 3](#_Toc531672268)

[3 Anforderungen 3](#_Toc531672269)

[4 Hauptteil 4](#_Toc531672270)

[4.1 Programmier-Umgebung und GUI-Toolkit 4](#_Toc531672271)

[4.2 Analyse 4](#_Toc531672272)

[4.2 Design 7](#_Toc531672273)

[4.3 Naming-Conventions 9](#_Toc531672274)

[4.4 Implementierung 9](#_Toc531672275)

[4.5 Testing 10](#_Toc531672276)

[4.6 Dokumentation 10](#_Toc531672277)

[4.7 Git 10](#_Toc531672278)

[4.8 Verzeichnisstruktur 10](#_Toc531672279)

[5 Rückblick 10](#_Toc531672280)

[6 Quellen 10](#_Toc531672281)

# Vorwort

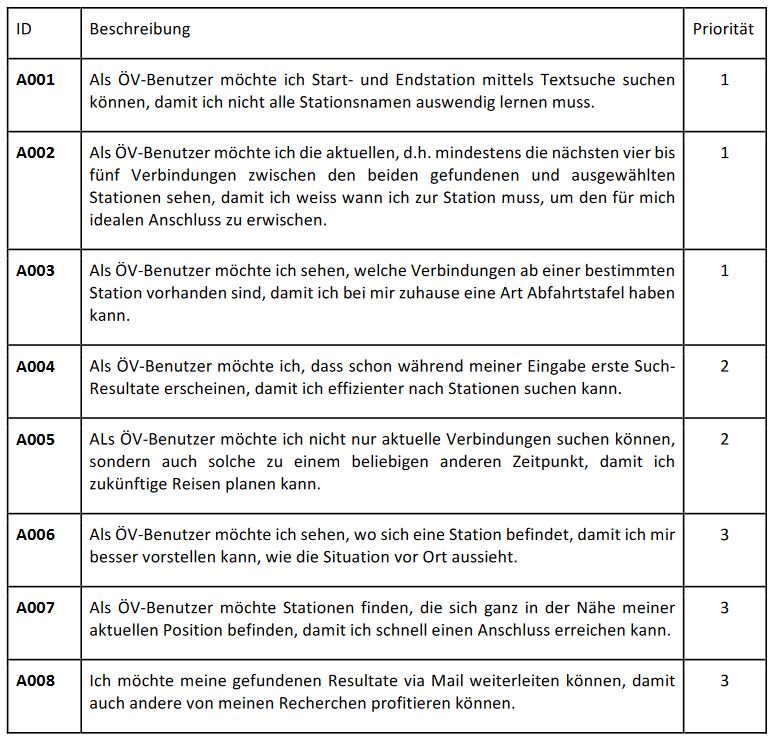
Im Überbetrieblichen Kurs Modul-318 haben wird den Auftrag erhalten, eine Windows-Applikation zu schreiben. Das Ziel des ÜKs und auch von mir war ein Programmier-Projekt von A-Z durchzuspielen. Was ein solches Projekt zu beinhalten hat, wurde uns in den ersten eineinhalb Tagen gezeigt.

Wie ich diese Schritte in meinem Projekt umgesetzt habe und alles was man zu meiner Applikation wissen muss, habe ich hier dokumentiert.

# Anforderungen

Wir hatten für unsere Applikation 8 Anforderungen. Diese waren in 3 Prioritätsstufen aufgeteilt.

1=must / 2 = should / 3 = nice to have



# Hauptteil

## Programmier-Umgebung und GUI-Toolkit

Da ein Teil dieses ÜKs die Einführung/Weiterbildung unserer Kenntnisse in VisualStudio war, stand für mich von Anfang an fest, dass ich VS für meine Projektarbeit verwenden werde.

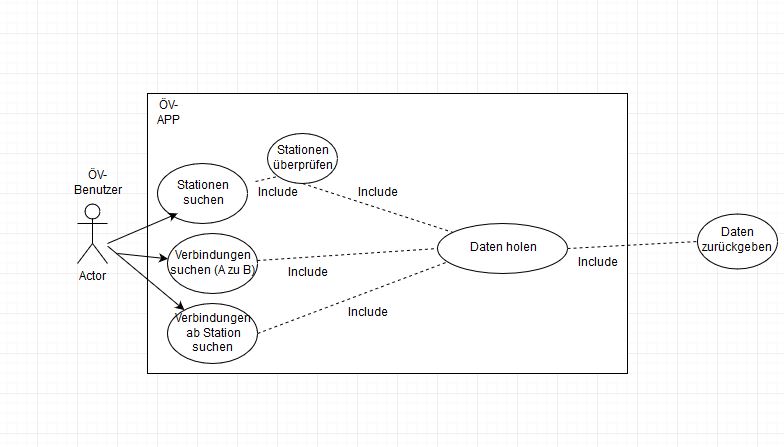
Bei der Programmiersprache hatten wir die Vorgabe C# zu verwenden. Jedoch hatten wir die Freiheit bei Framework/GUI-Toolkit. Zur Auswahl standen uns Windows Forms und WPF (Windows Presentation Foundation).   
Da ich zurzeit noch nicht in der Applikationsentwicklung arbeite und daher «nur» Erfahrung mit Windows Forms aus der Schule habe, entschied ich mich für WinForms.

Die Unterschiede der beiden Klassen-Bibliotheken sind in diesem [Artikel](https://www.yuhiro.de/winforms-versus-wpf-ein-vergleich/) sehr gut erklärt.

## Analyse

Um zu analysieren, was sich hinter der einzelnen Anforderung verbirgt und wie wir diese implementieren könnten, haben wir mehrere Diagramme erstellt. Für die Grafiken meiner Diagramme habe ich Draw.io verwendet.

Dazu haben wir zuerst ein UML Diagramm für die drei Anforderungen erstellt. Dies hat uns geholfen zu sehen, was ein Benutzer (Actor) machen kann/muss und was dazu in unserer App implementiert werden muss und was wird z.B. von der API erhalten können.



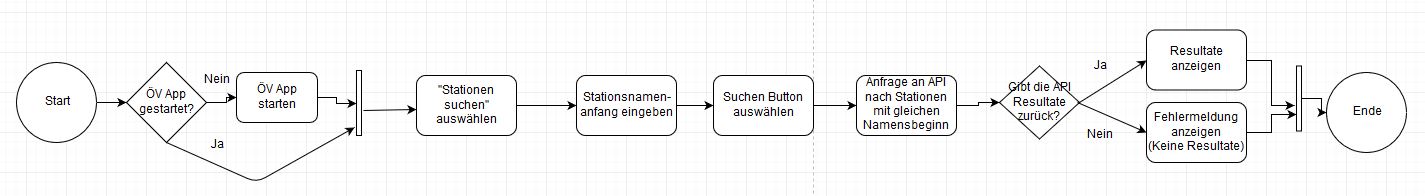
Für diese Aktionen habe ich drei UseCaseTabellen erstellt, die zeigen sollen, was die einzelnen Aktionen beinhalten:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Usecasename: | ÖV Station suchen | | Beschreibung: | Der ÖV Benutzer möchte Start- und Endstation mittels Textsuche suchen, diese werden automatisch aufgelistet. | | Vorbedingungen: | Programm installiert, Internetverbindung hergestellt. | | Auslöser: | Beginn der Eingabe in die TextBox. | | Akteure: | ÖV-Benutzer | | Ablauf: | 1. ÖV-App starten 2. «Stationen suchen» auswählen 3. Station Namen in TextBox anfangen einzugeben 4. Suchen Button auswählen 5. Einer der Vorschläge auswählen | | Varianten: | Startstation, Endstation | | Ergebnis: | Station vervollständigt in TextBox. | |

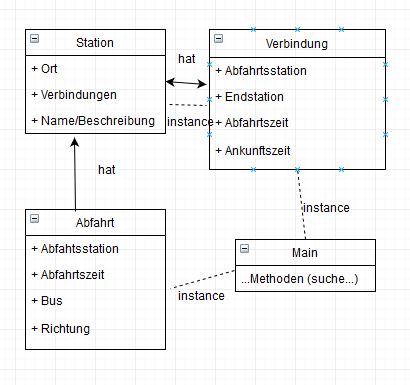
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Usecasename: | Verbindungen zwischen zwei Stationen finden | | Beschreibung: | Der ÖV Benutzer möchte mindestens die nächsten vier bis fünf Verbindungen zwischen zwei Stationen angezeigt erhalten. | | Vorbedingungen: | Programm installiert, Internetverbindung hergestellt. | | Auslöser: | User möchte die Verbindungen zwischen zwei Stationen finden. | | Akteure: | ÖV-Benutzer | | Ablauf: | 1. ÖV-App starten 2. «Verbindungen suchen» auswählen 3. Start- und Endstationsnamen eintragen 4. Suchen Button auswählen | | Varianten: | keine | | Ergebnis: | Liste der Verbindungen wird angezeigt. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Usecasename: | Abfahrten von einer Station finden | | Beschreibung: | Der ÖV Benutzer möchte die Abfahrten ab einer bestimmten Station angezeigt erhalten. | | Vorbedingungen: | Programm installiert, Internetverbindung hergestellt | | Auslöser: | User möchte die Abfahrten ab einer Station finden. | | Akteure: | ÖV-Benutzer | | Ablauf: | 1. ÖV-App starten 2. «Abfahrten suchen» auswählen 3. Stationsname eintragen 4. Suchen Button auswählen | | Varianten: | keine | | Ergebnis: | Liste der Abfahrten wird angezeigt. | |

Um den genauen Ablauf einer dieser drei Aktionen genauer aufzuzeigen und zu verstehen, habe ich ein Aktivitäts-Diagramm für die Suche nach einer Station erstellt. Dabei habe ich darauf geachtet, dass das Diagramm für jeden so verständlich wie möglich gemacht ist.



Anschliessend habe ich noch ein Klassendiagramm erstellt, bei dem ich mir genauer darüber Gedanken gemacht habe, welche Klassen meine Applikation beinhalten sollte.



# 4.2 Design

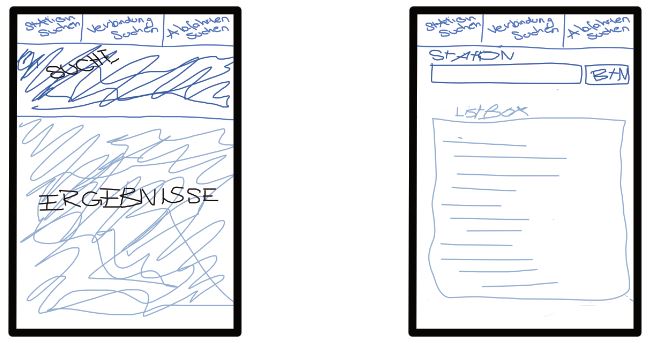
Während des ÜKs wurden uns verschiedene Aspekte aufgezeigt, die zu einem ergonomischen GUI beitragen.

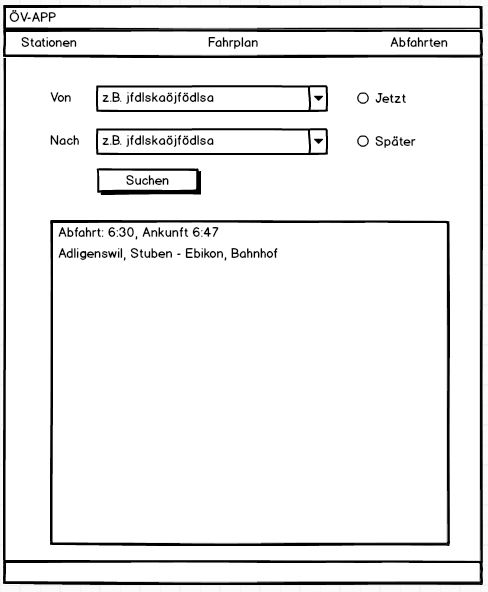
Ich habe versucht, folgende sechs Aspekte besonders in meinem GUI zu beachten:

* Selbsterklärend (nicht zuerst ein Manual lesen müssen, bevor man die App verwenden kann)
* Logisch (damit man bei der Form «SearchConnectionForm» die verschieden Verbindungen finden kann)
* Komfortabel (praktisches, modernes, schlichtes GUI)
* Gewohnt (gleiches Layout auf allen Forms, verschiedene Reiter oben)
* Übersichtlich (kompakt und nach dem Motto «weniger ist mehr», auf das Wesentliche konzentriert)
* Fehlerbewusst (auf falsche Eingaben aufmerksam gemacht werden)

Diese Punkte habe ich bereits beim Erstellen meines Mockups berücksichtigt.

Für die Mockups habe ich Affinity Designer und Balsamiq verwendet.





# 4.3 Naming-Conventions

Ich habe mich für Folgende Naming-Conventions entschieden:

Variablen:

* Lokale Variablen => PascalCase
* Globale Variablen => PascalCase
* Variablen werden «oben» definiert

Methoden:

* PascalCase
* Benamselung nach Verb-Nomen
* Parameter camelCase
* Alle Methoden Kommentar

Klassen:

* PascalCase
* Nomen als Namen

WinForms Objects:

* Typ-Namen (btnBuyObject)
* camelCase

Klammern:

* {} => Neue Linie

Forms:

* PascalCase

Die Benennungen der GUI-Objekte und der Codes habe ich in Englisch gewählt.

# 4.4 Implementierung

Bei der Implementierung habe ich zuerst die «SearchConnectinForm» mit Hilfe der WinForms-Objekte erstellt. Dazu habe ich meine bereits erstellen Mockups verwendet.

Im Vergleich zu den Mockups habe ich mich an meine Grundstruktur gehalten. Jedoch habe ich anstatt der 3 geplanten Forms nur 2 erstellt und dabei die „Stationssuch-Funktion“ und die „Abfahrtsplan-Funktion“ zusammen genommen.

# 4.5 Testing

??????

# 4.6 Dokumentation

Ich habe alles, was für mich als relevant galt in ein .txt File unformatiert eingetragen. Ab Tag 4 habe ich diese Stichwörter in diesem Dokument vervollständigt. Sobald neue Infos, die in diese Dokumentation gehörten, auftauchten, habe ich diese fortlaufend eingepflegt.

# 4.7 Git

Für die Versionierung meiner Applikation habe ich Git verwendet. Sobald ich eine Veränderung vorgenommen und abgeschlossen habe, habe ich diese commited. Bei der Benennung habe ich darauf geachtet, zuerst zu notieren, was gemacht wurde und erst dann, wo die Änderung vorgenommen wurde (z.B. «Edited Rows dgvDepartures»).

# 4.8 Verzeichnisstruktur

Um eine saubere Verzeichnisstruktur zu haben, habe ich einen src für alle Codefiles, einen tests für alle Testfiles und einen img für alle Bilder Ordner erstellt.

# Rückblick

?????

# Quellen

???