**M318 Dokumentation SBB Fahrplan**

|  |  |
| --- | --- |
| Erstelldatum | **28.11.2018** |
| Autor | **Lino Meyer** |
| Abgabedatum | **05.12.2018** |

**Inhaltsverzeichnis**

[1 Einleitung 3](#_Toc531780715)

[1.1 Zweck des Dokuments 3](#_Toc531780716)

[2 Funktionen 4](#_Toc531780717)

[2.1 Umgesetzte Funktionen 4](#_Toc531780718)

[2.2 Nicht umgesetzte Funktionen 4](#_Toc531780719)

[3 Programm installieren 5](#_Toc531780720)

[4 Systemtest 7](#_Toc531780721)

[4.1 Testfall «Textsuche» 7](#_Toc531780722)

[4.2 Testfall «Verbindungen» 7](#_Toc531780723)

[4.3 Testfall «Fahrplan» 8](#_Toc531780724)

[5 Programmierrichtlinien 9](#_Toc531780725)

[5.1 Zweck von Programmierrichtlinien 9](#_Toc531780726)

[5.2 Programmierrichtlinien 9](#_Toc531780727)

[5.2.1 Einrückung 9](#_Toc531780728)

[5.2.2 Kontrollstrukturen 9](#_Toc531780729)

[5.2.3 Funktionsaufrufe 10](#_Toc531780730)

[5.2.4 Kommentare 10](#_Toc531780731)

[5.2.5 Namenskonventionen 11](#_Toc531780732)

[6 Klassendiagramm 12](#_Toc531780733)

[7 GUI – Design 13](#_Toc531780734)

[8 Diagramme 15](#_Toc531780735)

[8.1 Use Case Diagramm 15](#_Toc531780736)

[8.2 Aktivitätsdiagramm 15](#_Toc531780737)

# Einleitung

Im ÜK M318-1 objektbasiert Programmieren geht es darum ein Projekt zu realisieren. In unserem Fall war es die Aufgabe einen eigenen SBB Fahrplan mit den Daten der SBB Schnittstelle zu realisieren. Bevor wir mit dem Projekt anfingen machten wir eine Einführung in C# und diverse Programmierübungen. Ausserdem haben wir viele Möglichkeiten zur Planung und Visualisierung von Projekten und auch Arbeitsmethoden wie Scrum angeschaut.

## Zweck des Dokuments

Dieses Dokuments soll keine ausführliche Dokumentation sein, sondern die wichtigsten Informationen zu meinem Projekt beinhalten. Darunter gehört eine eigene Programmierrichtlinie, GUI – Designs und einige Diagramme. Wie auch einige Systemtests zu den wichtigsten Anforderungen

# Funktionen

## Umgesetzte Funktionen

Ich habe alle obligatorischen Anforderungen und eine der Anforderungen mit 2. Priorität in mein Programm eingefügt. Mein Programm kann also Stationen mittels Textsuche finden. Die wichtigsten Informationen zu den Verbindungen zwischen zwei ausgewählten Stationen, zu einem beliebigen Zeitpunkt, anzeigen. Und es ist auch möglich einen Fahrplan zu erstellen, der alle verfügbaren Verbindungen einer Station anzeigt.

|  |  |  |
| --- | --- | --- |
| **ID** | **Beschreibung** | **Priorität** |
| **A001** | Als ÖV-Benutzer möchte ich Start- und Endstation mittels Textsuche suchen können, damit ich nicht alle Stationsnamen auswendig lernen muss. | 1 |
| **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. | 1 |
| **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. | 1 |
| **A004** | Als ÖV-Benutzer möchte ich, dass schon während meiner Eingabe erste Suchresultate erscheinen, damit ich effizienter nach Stationen suchen kann. | 2 |
| **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. | 2 |
| **A006** | Als ÖV-Benutzer möchte ich sehen, wo sich eine Station befindet, damit ich mir besser vorstellen kann, wie die Situation vor Ort aussieht. | 3 |
| **A007** | Als ÖV-Benutzer möchte Stationen finden, die sich ganz in der Nähe meiner aktuellen Position befinden, damit ich schnell einen Anschluss erreichen kann. | 3 |
| **A008** | Ich möchte meine gefundenen Resultate via Mail weiterleiten können, damit auch andere von meinen Recherchen profitieren können. | 3 |

Blau = umgesetzt

## Nicht umgesetzte Funktionen

Aus zeitlichen Gründen konnte ich keine der Funktionen mit 3. Priorität umsetzen. Mit der Textsuche während der Eingabe hatte ich viele Probleme mit der Performance und mit Abstürzen des Programms während der Laufzeit. Da meine Textsuche meiner Meinung nach schon sehr gut zugänglich ist

(durch TAB -> ENTER -> ENTER für erstes Resultat), wollte ich für diese Anforderung nicht noch mehr Zeit aufwenden.

# Programm installieren

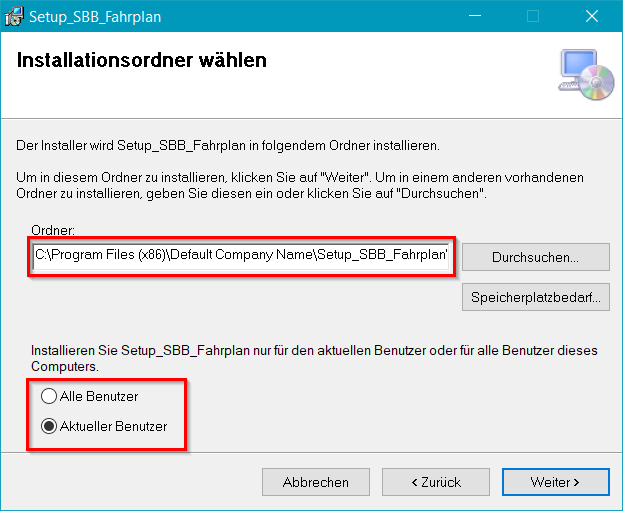
Für die Installation des Programms habe ich einen *installer* erstellt. Der Installer ist jeweils im Release und im Projekt SBB\_Fahrplan unter dem Ordner installer zu finden.

Die Installation ist sehr einfach und erfolgt folgendermassen:

1.Installer ausführen

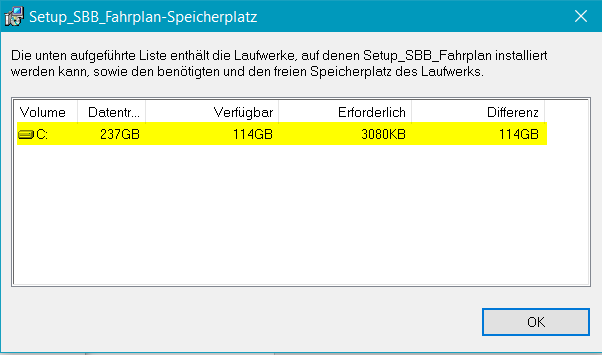


2. Speicherort und Benutzer für Installation auswählen

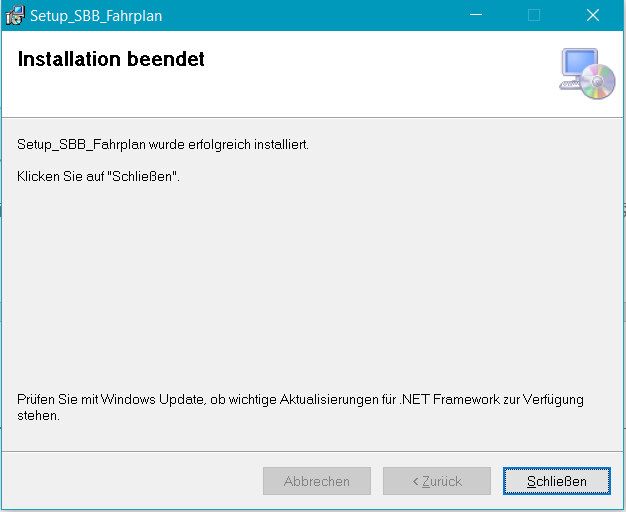


3. Benötigter Speicherplatz einsehen

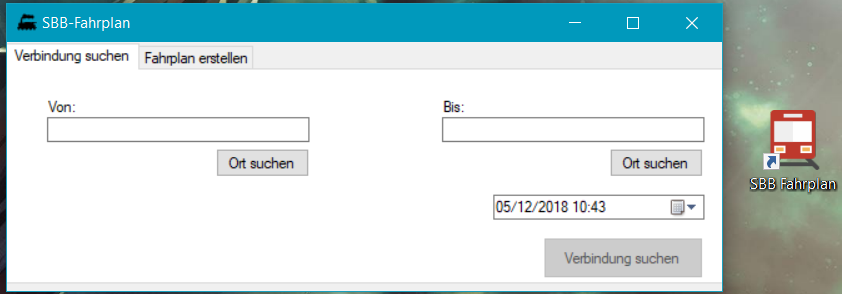




4. Installation ausführen



Es wird automatisch eine Verknüpfung zum Programm auf dem Desktop und im Startmenü, mit einem Icon, erstellt.



# Systemtest

## Testfall «Textsuche»

|  |  |  |
| --- | --- | --- |
| Schritt | Aktivität | Erwartetes Resultat |
| 1 | Benutzer startet Programm | Das Programm wird gestartet und die Benutzeroberfläche für die Verbindungssuche wird geöffnet. |
| 2 | Benutzer gibt eine Station ein | Textfeld füllt sich mit eingegebenem Text. |
| 3 | Benutzer klickt auf den zum Textfeld zugehörenden Button «Ort suchen» | Ein Fenster mit einer Liste der gefundenen Stationen öffnet sich. |
| 4 | Benutzer wählt eine Station aus | Das Fenster mit der Liste schliesst sich und das Textfeld füllt sich mit der ausgewählten Station. |

## Testfall «Verbindungen»

|  |  |  |
| --- | --- | --- |
| Schritt | Aktivität | Erwartetes Resultat |
| 1 | Benutzer startet Programm | Das Programm wird gestartet und die Benutzeroberfläche für die Verbindungssuche wird geöffnet. |
| 2 | Benutzer gibt Abfahrtsstation ein | Textfeld füllt sich mit eingegebenem Text. |
| 3 | Benutzer klickt auf den zum Textfeld zugehörenden Button «Ort suchen» | Ein Fenster mit einer Liste der gefundenen Stationen öffnet sich. |
| 4 | Benutzer wählt Abfahrtsstation aus | Das Fenster mit der Liste schliesst sich und das Textfeld füllt sich mit der ausgewählten Station. |
| 5 | Benutzer wiederholt Schritt 2 – 4 mit der Ankunftsstation |  |
| 6 | Benutzer gibt Datum und Zeit ein | Datum und Zeit wird entsprechend den Eingaben angepasst. |
| 7 | Benutzer klickt auf den Button  «Verbindung suchen» | Es werden Verbindung zwischen der Abfahrts- und Ankunftsstation, zur eingegebenen Zeit gesucht, wenn keine gefunden wird, wird ein Fehler ausgegeben, sonst wird ein Fenster mit den gefundenen Verbindungen geöffnet. |

## Testfall «Fahrplan»

|  |  |  |
| --- | --- | --- |
| Schritt | Aktivität | Erwartetes Resultat |
| 1 | Benutzer startet Programm | Das Programm wird gestartet und die Benutzeroberfläche für die Verbindungssuche wird geöffnet. |
| 2 | Benutzer wählt Reiter «Fahrplan erstellen» | Die Benutzeroberfläche unter dem Reiter «Fahrplan erstellen» wird geöffnet. |
| 3 | Benutzer gibt eine Station ein | Textfeld füllt sich mit eingegebenem Text. |
| 4 | Benutzer klickt auf den zum Textfeld zugehörigen Button «Ort suchen» | Ein Fenster mit einer Liste der gefundenen Station öffnet sich. |
| 5 | Benutzer wählt eine Station aus | Das Fenster mit der Liste schliesst sich und das Textfeld füllt sich mit der ausgewählten Station. |
| 6 | Benutzer klickt auf den Button «Fahrplan erstellen» | Die eingegebene Station wird gesucht, falls keine gefunden wird, wird ein Fehler ausgegeben, sonst wird ein Fenster mit allen möglichen Verbindungen der Station geöffnet. |

# Programmierrichtlinien

## Zweck von Programmierrichtlinien

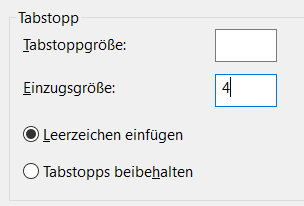
Vor allem wenn eine Gruppe an einem Projekt arbeitet, kommt es oft zu Problemen mit den verschiedenen Programmierstilen der einzelnen Mitglieder. Darum gibt es Richtlinien an die man sich halten muss, um einen gut strukturierten, konstant gleich formatierten und lesbaren Code zu ermöglichen.

## Programmierrichtlinien

Die folgenden Programmierrichtlinien wurden von Nevin Helfenstein und Lino Meyer erstellt.

### Einrückung

Einrückungen gehören zu den wichtigsten Mitteln um den Code strukturiert zu halten.

Wir haben uns für Einrückungen in Höhe von vier Leerzeichen entschieden. Das heißt, im Visual Studio muss eingestellt sein, dass ein Tab **vier** Leerzeichen entspricht. Ausserdem sollte jeder einstellen, dass die Tabs durch Leerzeichen ersetzt werden.

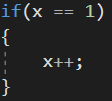
### Kontrollstrukturen

Mit Kontrollstrukturen sind folgende Elemente gemeint:

* if
* for
* while
* switch

Hier sollte darauf geachtet werden, dass die geschweiften Klammern immer auf der Höhe der Kontrollstruktur in einer separaten Zeile stehen.

Gute Beispiel:



Schlechtes Beispiel:



Bei if – Anweisungen ist die verkürzte Schreibweise erlaubt.



### Funktionsaufrufe

Ein Funktionsaufruf geschieht immer ohne Leerzeichen zwischen dem Funktionsnamen, der öffnenden Klammer und dem ersten Übergabeparameter. Die einzelnen Parameter werden mit Komma und Leerzeichen voneinander getrennt.

Gutes Beispiel:



Schlechtes Beispiel:



### Kommentare

Damit auch andere die eventuell vorhandenen Bugs in eurem Code fixen können, muss dieser nicht nur übersichtlich, sondern auch verständlich sein. Daher ist der ein oder andere Kommentar unumgänglich. Hier ist der richtige Mittelweg zu wählen. Zu viel Kommentare verschränkt den Blick auf das wesentliche, zu wenig Kommentare lässt ihn undurchdringbar erscheinen…

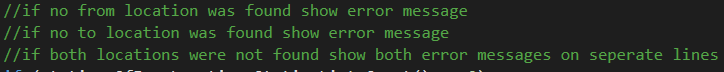
Kommentare sollten in der Regel in Englisch erfasst werden, so dass der Sourcecode auch von nicht deutschsprachigen Entwicklern gelesen werden kann.

Kommentare direkt im Sourcecode (keine Funktions- und Methodenbeschreibung) müssen eingerückt werden. So dass sie auf derselben Ebene wie der zu kommentierende Code stehen. Ein- bis zweizeilige Kommentare werden wie folgt dargestellt:

Gutes Beispiel:

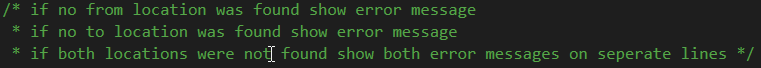


Schlechtes Beispiel:



Wie oben zu sehen ist, sollte auch in Kommentaren von Umlauten abgesehen werden. Erstreckt sich ein Kommentar über mehr als 2 Zeilen sollten Block-Kommentare eingesetzt werden:

Gutes Beispiel:



Schlechtes Beispiel:

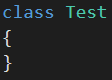


### Namenskonventionen

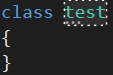
#### Klassen

Klassen sollen selbstsprechende Namen erhalten. Kryptische Abkürzungen sollte man, wenn möglich vermeiden. Klassen sollten immer mit einem Großbuchstaben beginnen.

Gutes Beispiel:



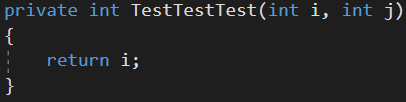
Schlechtes Beispiel:



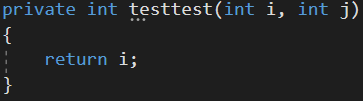
#### Funktionen

Bei Funktionen sind der erste Buchstabe und die ersten Buchstaben der nächsten Worte grossgeschrieben.

Gutes Beispiel:



Schlechtes Beispiel:



#### Variablen

Variablen werden im Camel Case Style geschrieben. Dies soll heißen, dass der erste Buchstabe klein geschrieben ist und der erste Buchstabe des nächsten Wortes gross.

Gutes Beispiel:



Schlechtes Beispiel:



#### Konstanten

Bei Konstanten werden alle Buchstaben grossgeschrieben und die einzelnen Worte innerhalb des Namens durch Unterstriche getrennt.

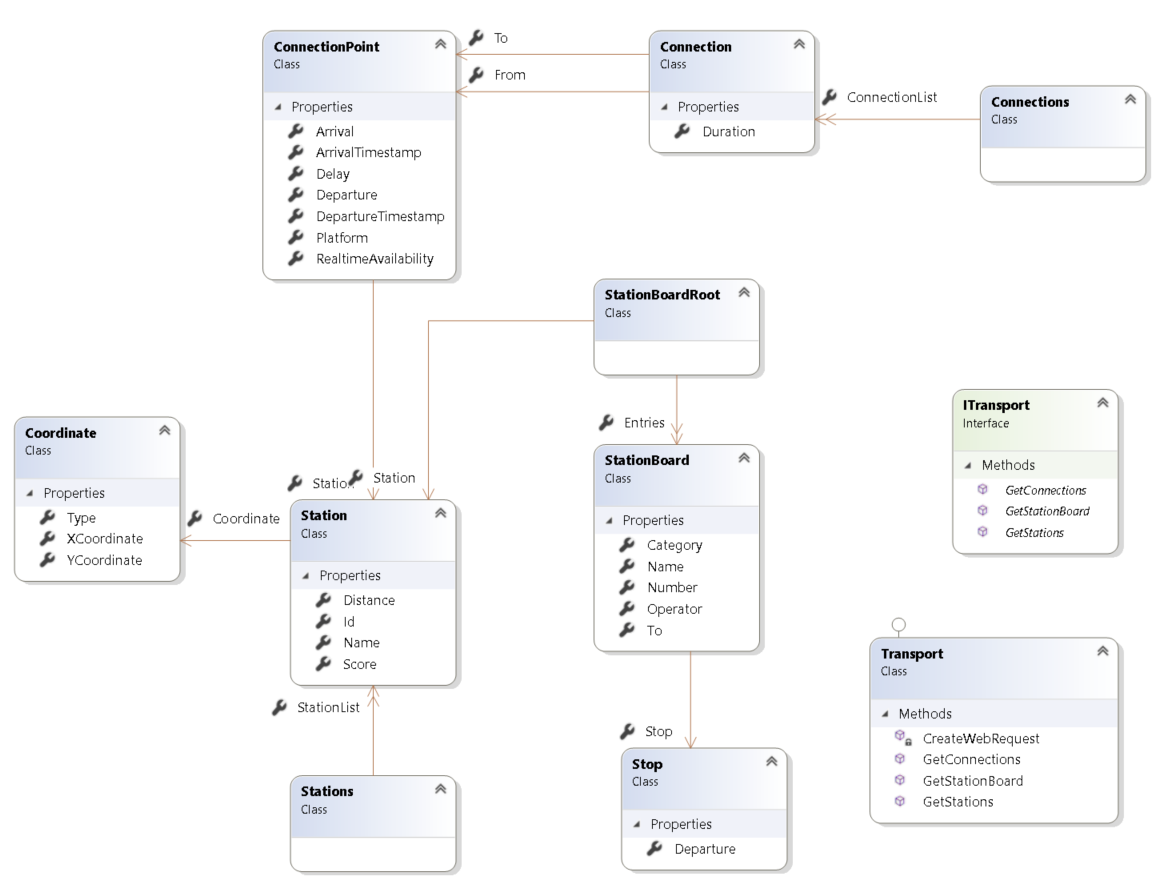
Gutes Beispiel:

****

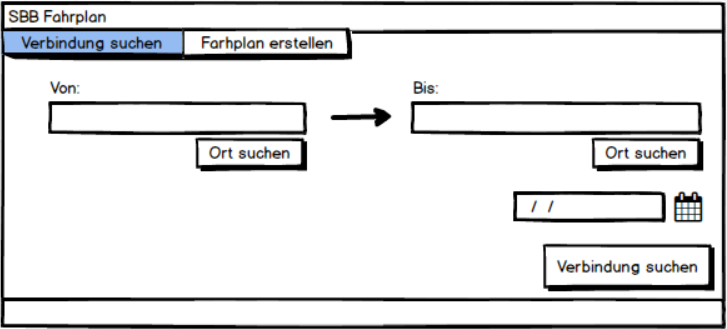
Schlechtes Beispiel:

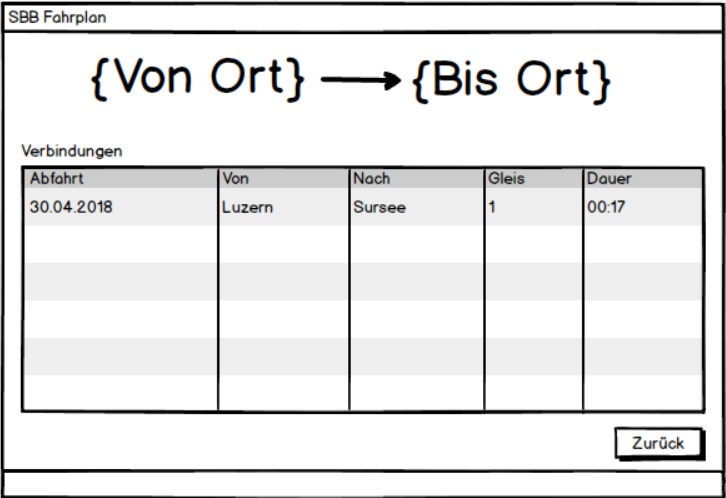


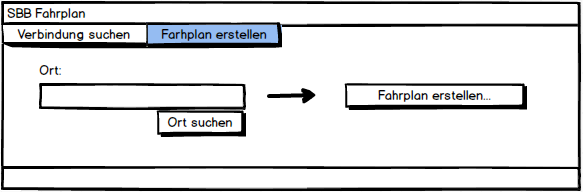
# Klassendiagramm

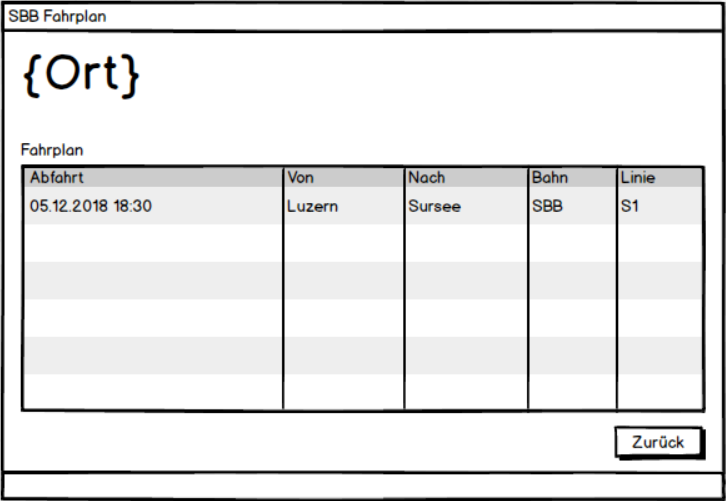


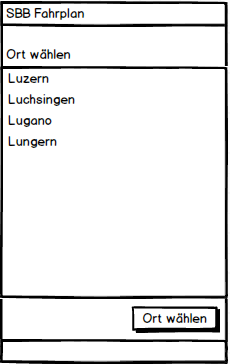
# GUI – Design





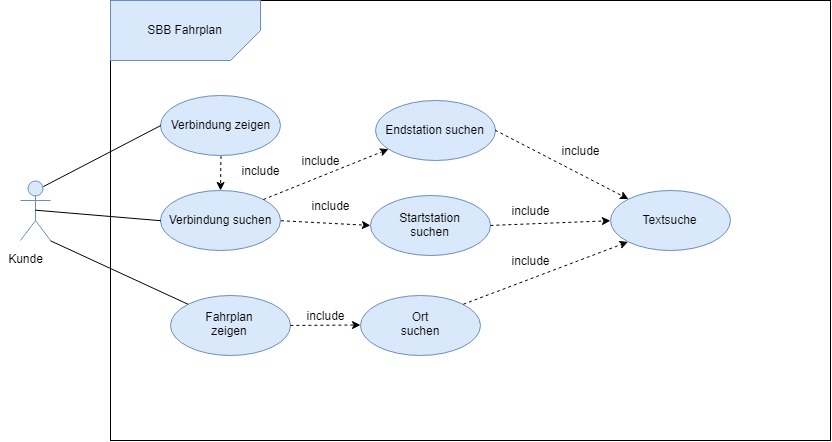
****



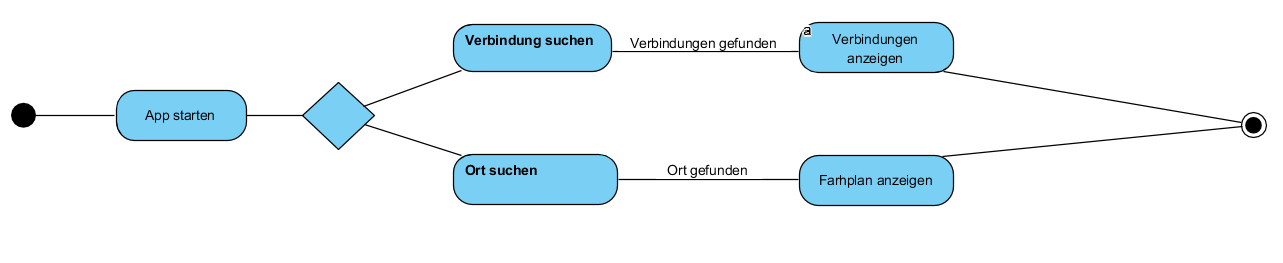


# Diagramme

## Use Case Diagramm



## Aktivitätsdiagramm

****