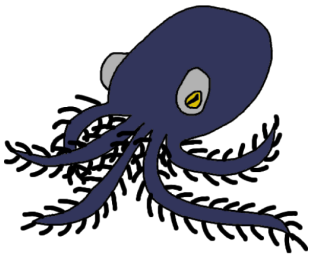
intension

DHBW
Duale Hochschule
Baden-Württemberg

# **Featherkraken**: Bestpreissuche für Flugangebote mit variablen Abflughäfen



## STUDIENARBEIT

des Studienganges Informatik
an der Dualen Hochschule Baden-Württemberg Stuttgart
von
Ingo Kuba

| | |
|---|---|
| **Matrikelnummer, Kurs** | place, holder |
| **Ausbildungsfirma** | intension GmbH |
| **Betreuer** | Place Holder |

# Erklärung zur Eigenleistung

Hiermit erkläre ich, dass ich die vorliegende Studienarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.
Die Stellen der Studienarbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Ostfildern, den 27. Mai 2020 _____

# Zusammenfassung

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Akronyme

**CDI** Context Dependency Injection. 3

**ERM** Entity Relationship Model. 1, 2, 5, 6

**IATA** IATA airport code. 8, 9

**Jakarta EE** Jakarta Enterprise Edition. 2, 3

**UML** Unified Markup Language. 2

# 1 Einleitung

## 1.1 Motivation

In der Regel möchte ein Fluggast den günstigsten Preis für eine bestimmte Route A nach B. Jede Flugsuchmaschine im Internet bietet diese Feature. Manchmal sucht ein Fluggast auch einfach nach Inspiration und möchte Angebote von A nach X, wobei X variabel ist. Einige Suchmaschinen bieten diese Suche bereits an. Worum es in dieser Studienarbeit geht, ist der umgekehrte Fall: X nach B. Also von welchem beliebigen Flughafen kommt man möglich günstig an ein festes Ziel. Gerade auf hochpreisigen Strecken kann es sich lohnen einen Umweg zu fliegen.

## 1.2 Aufgabenstellung

Bei der Suche sollen die klassischen Filterkriterien implementiert werden. Das heißt die Unterscheidung ob man nur einen Hinflug oder Hin- und Rückflug buchen möchte. Des weiteren soll man jeweils ein Datum für An- und Abreise festlegen können, welches um drei Tage flexibel sein soll. Neben der Buchungsklasse (Economy, Business, First Class) soll auch die Wahl der Airline oder Allianz eingeschränkt werden können. Außerdem soll man Passagier- und Umsteigeanzahl wählen können.
Zusätzlich soll ein Entfernungsfilter um einen möglichen Abflughafen bereitgestellt werden. Zum Beispiel wird nur nach Angeboten gesucht, bei dem sich der Startflughafen maximal 800km (Entfernungsfilter) vom Flughafen Stuttgart (möglicher Abflughafen) entfernt befindet.
Diese Flugsuchmaschine soll über ein Web-Frontend vom Nutzer bedient werden können. (?, ?)

# 2 Grundlagen

## 2.1 Entity Relationship Model

Um das Datenmodell der Anwendung darzustellen wurde eine vereinfachte Variante des Entity Relationship Model (ERM) verwendet.

## Entities

Ein Objekt oder Entity wird in einem Rechteck dargestellt und kann Attribute besitzen, wobei komplexe Attribute als Beziehungen zu anderen Objekten dargestellt werden. Der Name der Beziehung entspricht hierbei dem Attributnamen im Code. Die Anzahl der möglichen Relationen wird in UML-Notation angegeben. Zum Beispiel ist in Abbildung 1 zu sehen, dass eine Person null bis n Autos besitzen kann.

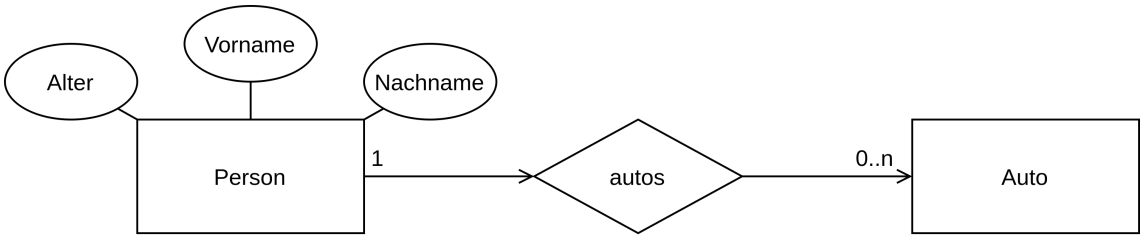



Abbildung 1: Beispiel für eine Entity mit Attributen und einer Beziehung

## Attribute

Attribute können dabei eindeutig, optional oder mehrwertig sein. Die Unterscheidung zwischen Datum, Zahl oder Zeichenkette wird in einem Entity Relationship Model nicht dargestellt.


Abbildung 2: Attribute (v.l.): eindeutig, optional und mehrwertig

## 2.2 Jakarta EE

Jakarta Enterprise Edition (Jakarta EE) ist eine Spezifikation, welche Java SE (Standard Edition) erweitert und ermöglicht damit das Entwickeln von mehrschichtigen, zuverlässigen und sicheren Netzwerkanwendungen. Zuvor war Jakarta EE als Java EE bekannt.
(*Differences between Java EE and Java SE*, 2012)

## 2.3 MicroProfile

Eclipse MicroProfile ist eine Erweiterung von Jakarta EE und eine Alternative für Spring, welches ein Framework der Firma Pivotal ist und vielgenutzte Lösung für Jakarta EE Anwendungen und Microservices.
MicroProfile bietet unter anderem Unterstützung für Context Dependency Injection (CDI), sowie eine bessere Konfigurierbarkeit der Anwendung.
(Monteiro, 2018)

## 2.4 Payara

TODO

## 2.5 Docker

Docker ist eine quelloffene Software zur Isolierung von Anwendungen in sogenannten Containern. Mit ihr können zum Beispiel Webserver oder Datenbanken schnell eingerichtet werden, ohne diese vorher aufsetzen zu müssen.

### 2.5.1 Image

*Images* sind die Vorlage zur Erstellung neuer Container. Sie können aus mehreren Schichten von anderen Images bestehen und so können komplexe Systeme leichtgewichtig und portabel erstellt werden.

### 2.5.2 Container

Ein *Container* ist die aktive Instanz eines Docker images und kann beliebig konfiguriert und bearbeitet werden. Wird ein Container beendet und ein neuer nach der Vorlage desselben Images gestartet, gehen alle Änderungen verloren. Dies lässt sich durch die Verwendung von *Volumes* verhindern, welche die Konfiguration eines Containers persistieren.

## 2.6 React.js

TODO

# 3 Entwurf

## 3.1 Auswahl der externen API

Um Flugdaten zu erhalten muss eine externe Schnittstelle benutzt werden, welche die Pflichtanforderungen erfüllt und dabei leicht zu verwenden ist. Die Wahl der Schnittstelle fiel dabei auf eine Rest-API, da diese sehr einfach zu benutzen sind. Für die Auswahl des Anbieters wurde eine Tabelle[3] erstellt, welche die Pflichtanforderungen mit den Funktionalitäten der jeweiligen Schnittstelle abgleicht. Dabei erfüllte die API von Kiwi nicht nur alle Anforderungen, sondern war auch sehr gut dokumentiert und mit Beispielen beschrieben. Des Weiteren bot Kiwi auch eine Rest-API um Flughäfen im Umkreis gegebener Koordinaten zu finden, was der Aufgabe entgegen kam.

| | **Skyscanner** | **Hipmunk** | **Kajak** | **Flight Data** | **Flight Bookings** | **Kiwi Flights** |
|---|---|---|---|---|---|---|
| **Single flight** | yes | yes | yes | yes | yes | yes |
| **Two directions** | no | no | no | yes | no | yes |
| | | | | | | |
| **Specific date** | yes | yes | yes | yes | yes | yes |
| **Flexible date** | no | limited | limited | yes | no | yes |
| | | | | | | |
| **Class** | yes | yes | yes | limited | yes | yes |
| **Passengers** | yes | yes | yes | no | yes | yes |
| **Airline** | yes | no | no | no | no | yes |
| **Stops** | no | no | no | no | no | yes |

Abbildung 3: Tabelle zum Vergleich der APIs

## 3.2 Datenmodell

Das Entity Relationship Model für das Datenmodell wurde hier aufgeteilt in Suchanfrage (`SearchRequest`) und Suchergebnis (`SearchResult`).

### 3.2.1 SearchRequest

Eingehende Anfragen an den Service (`SearchRequest`) haben verschiedene Parameter, welche die zuvor genannten Pflichtanforderungen abdecken. Komplexe Attribute wie die Zeitspanne (`Timespan`) für An- und Abreise sowie den Ursprungs- und Ziel-Flughafen (`Airport`) wurden in eigene Objekte ausgelagert, damit sie so wiederverwendet werden können.

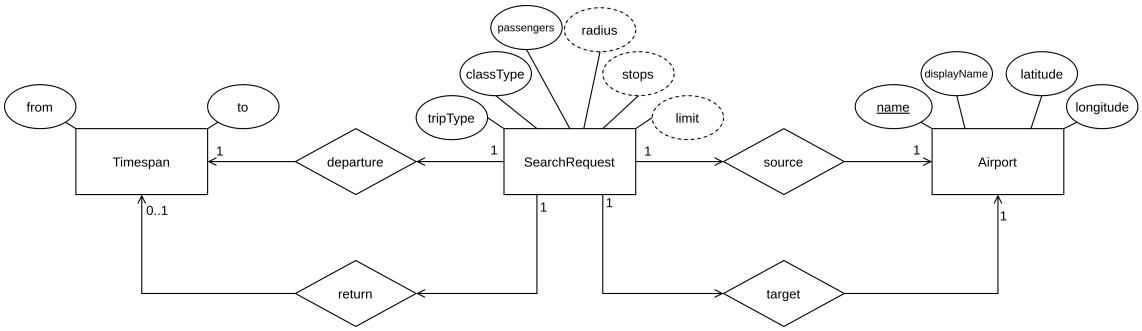



Abbildung 4: Entity Relationship Model des SearchRequest Objekts

### 3.2.2 SearchResult

Antworten des Service (`SearchResult`) sind etwas komplexer als Anfragen aufgebaut, teilen jedoch manche Attribute beziehungsweise Objekte mit diesen. So enthält ein Suchergebnis eine Ansammlung von sogenannten `Trips`, welche sich aus Flügen (`Flight`) der An- und Abreise zusammensetzen. Diese Flügen haben widerrum selbstverständlich selbst jeweils einen Start- und Zielflughafen, welche in dem `Route`-Objekt mit Informationen zu Abflug- und Ankunftzeiten sowie der Airline enthalten sind.
Bei erster Betrachtung fällt auf, dass Informationen über Airlines und Zeiten redundant sind, was jedoch für die optimale Anzeige in der Oberfläche gedacht ist.

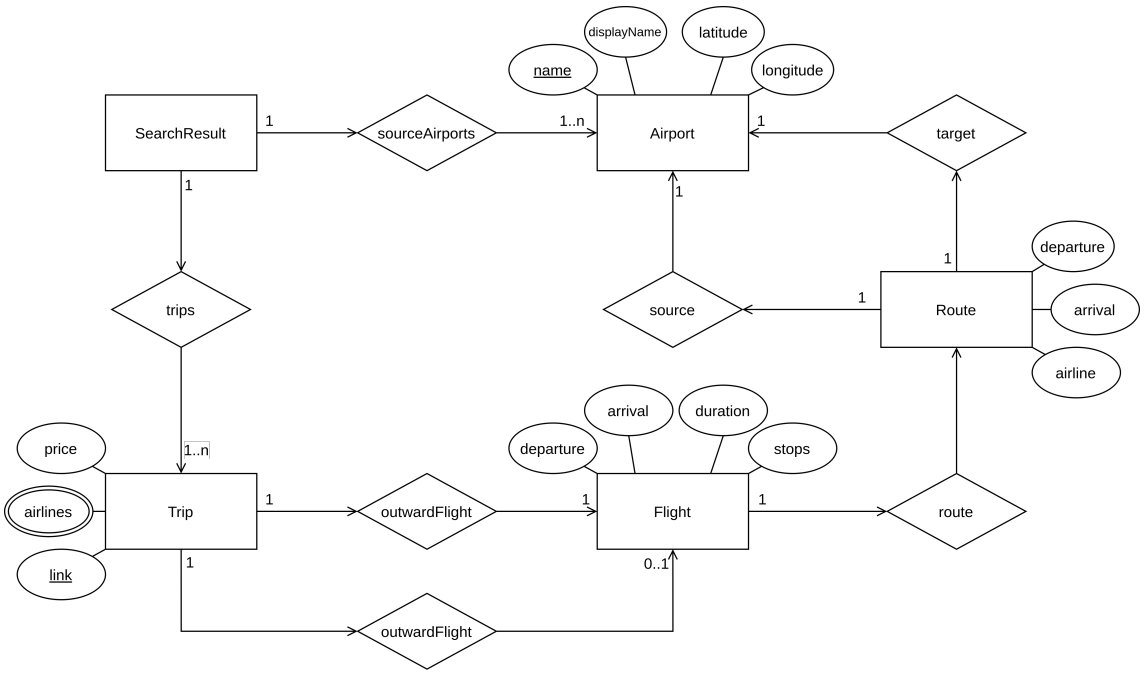



Abbildung 5: Entity Relationship Model des SearchResult Objekts

## 3.3 Framework

Die Anwendung wird in eine Server- und eine Web-Anwendung unterteilt, für welche jeweils eine Framework-Technologie gewählt werden muss. Die Serveranwendung übersetzt Suchanfragen von der Oberfläche zu einem Format, welches die externe Schnittstelle akzeptiert und leitet das Sucherergebnis daraufhin an die Oberfläche zurück. Dies bedeutet, dass die Serveranwendung als Rest-Client für die externe Schnittstelle und selbst als Rest-Schnittstelle für die Oberfläche dient. Für die Komponenten war das Thema Plattformunabhängigkeit sehr wichtig und aus eigener Erfahrung eignet sich dafür Java sehr gut, da es eine große Community hat und eine Anwendung in kurzer Zeit aufgesetzt ist. Außerdem bietet Java einfache Möglichkeiten eine externe Rest-Schnittstelle anzusprechen.
Als Bedingung für die Web-Oberfläche ist es unabdingbar, dass die Anwendung intuitiv bedienbar ist und auf allen Endgeräten gut dargestellt werden kann. Für diesen Zweck kommen eigentlich nur drei Frameworks in Frage: Angular, React.js und Vue.js. Dabei wurde React gewählt, welches sich besser als Vue.js und Angular für kleine Anwendungen eignet, wie es in diesem Fall zutrifft. Bei dem hier vorliegenden Fall wird es sogar eine sogenannte Single-Page-Webapplication, das heißt die gesamte Funktion kann auf einer Seite dargestellt werden.

# 4 Implementierung

## 4.1 Backend

Das Backend der Anwendung dient als Proxy zwischen Web-Frontend und externer API und wurde in Java geschrieben. Dabei wurde Jakarta EE[2.2] und Eclipse Microprofile[2.3] verwendet, um eine Rest-Schnittstelle für die Suche von Flügen zu bieten. Durch das Verwenden von Microprofile lässt sich die Anwendung einfach in einem Payara Application Server starten. Dieser wurde mithilfe eines Docker Images umgesetzt und dadurch lässt er sich einfach in einer Cloud Umgebung oder klassisch auf einem Windows- oder Linux-Server deployen.

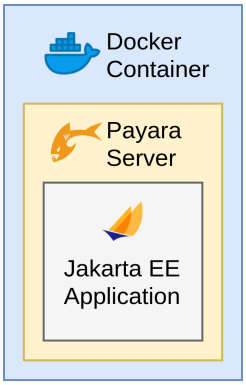



Abbildung 6: Aufbau des Backends

### 4.1.1 Flughafensuche: GET /airports

Um bestimmte Flughäfen mit einem Teil des Namens des Flughafens oder der Stadt suchen zu können, gibt es den Endpoint `/airports` der mit einem GET-Request erreicht werden kann. Die Eingabe wird über den Query-Parameter 'query' in der URL des Requests übergeben. Ein beispielhafter Request für die Suche von Flughäfen in Amsterdam kann zum Beispiel folgendermaßen aufgebaut sein:

```
GET www.featherkraken-example.com/airports?query=Ams
```

Damit werden alle Flughäfen gesucht, welche den Wortteil 'Ams' im IATA airport code oder Städtenamen enthalten. Die Antwort des Services besteht aus einer Liste aus gefundenen Flughäfen, welche jeweils IATA airport code, vollen Anzeigenamen und Koordinaten enthalten.

### 4.1.2 Flugsuche: POST /flights

Die Kernfunktion der Anwendung steckt hinter dem Endpoint `/flights`, welcher eingehende Suchanfragen in ein für die externe API verständliches Format übersetzt und das Suchergebnis dann zurückgibt. Suchparameter werden hier nicht in der URL des Requests übergeben, sondern mit einem POST im Request-Body, da die Suchanfrage so übersichtlicher und leichter zu dokumentieren ist. Bei der Suche kann wahlweise ein Radius angegeben werden, dann werden Flüge von umliegenden Abflughäfen gesucht. So kann eine Suchanfrage für Flüge von Frankfurt und Umgebung (100km) zum Beispiel so aussehen:

```
{
        "source": {
                "name": "FRA",
                "displayName": "Frankfurt International Airport",
                "latitude": 50.033056,
                "longitude": 8.570556
        },
        "target": {
                "name": "LAX",
                "displayName": "Los Angeles International",
                "latitude": 33.9425,
                "longitude": -118.40806
        },
        "radius": 100,
        "passengers": 1,
        "departure": {
                "from": "11.11.2020"
                "to": "13.11.2020"
        },
        "return": {
                "from": "22.11.2020"
        },
        "classType": "Economy",
        "tripType": "Round trip"
}
```

Für die Flughafenobjekte in `source` und `target` ist nur der eindeutige IATA airport code in `source.name` entscheidend. Der Radius wird in Kilometern angegeben. In diesem Beispiel werden Hin- und Rückflüge zwischen Frankfurt und Los Angeles in der Klasse 'Economy' gesucht. Außerdem ist zu erkennen, wie ein variables Abflugdatum (11.11. - 13.11.2020) angegeben werden kann.

# 5 Testing

Unit tests? How to test external api? Aspects of an API: speed? maybe Beständigkeit in der Speed? -> Postman

# 6 Zusammenfassung

## 6.1 Ausblick

-> mehr Schnittstellen anbinden (ist schon bereit dafür) -> jedoch nicht async -> Monitor speed of my requests!!

# Literatur

*Differences between Java EE and Java SE.* (2012). Zugriff auf https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

Monteiro, J.-L. (2018). *What is Eclipse MicroProfile?* Zugriff auf https://www.tomitribe.com/blog/what-is-eclipse-microprofile/