Software Engineering Übung

Gruppe: 7
Übungsleiter: Martin Vasko

Designmodell v.1.0

Projekttitel:easyFlight

Projekthomepage:
https://cewebs.cs.univie.ac.at/SWE/ws16/index.php?m=D&t=uebung&c=show&CEWebS_c=g050052-7t2

Gruppenmitglieder:

MatNr:	Nachname:	Vorname:	e-mail:
1148885	Bakic	David	a1148885@unet.univie.ac.at
1505013	Kirnbauer	Bernd	a1505013@unet.univie.ac.at
1505015	Plank	Stefan	a1505015@unet.univie.ac.at
1468802	Ussenbayeva	Maria	a1468802@unet.univie.ac.at

Erstellen sie ein Designmodell gemäß *Unified Process* das zumindest folgende Aspekte umfasst:

- Klassendesign
- Use-Case-Realization-Design
- Übersichtsklassendiagramm

- Architekturbeschreibungen

1 Klassendesign

- **Klasse User:**
  Die Klasse User ist eine Oberklasse für die Klasse Analyst, Fluggesellschaft und Customer. Diese Klasse hat als Attribute eine ID und ein Passwort, mit diesen Werten soll sich später ein jeder User (Analyst, Flugesellschaft, Customer) einloggen können. Methoden sind nur Setter und Getter.
- **Klasse Fluggesellschaft:**
  Diese Klasse erbt, die vorhin beschriebenen Attribute und Methoden von der Oberklasse User. Des Weiteren hat eine jede Fluggesellschaft einen einzigartigen Namen.
- **Klasse FlugGManagment:**
  Durch diese Klasse ist es möglich, Fluggesellschaften persistent mittels DAOs zu speichern, löschen und auf diese zuzugreifen.
- **Klasse Flug:**
  Die wahrscheinlich wichtigste Klasse ist der Flug, da ohne Flüge das Projekt keinen Sinn machen würde. Diese hat mehrere Attribute, die wichtigsten sind: die ID, welche einzigartig sein muss, den Ticketpreis, der von der Fluggesellschaft nach Erstellung des Fluges wieder geändert werden kann, sowie den Destinationcountry. Destinationcountry ist insofern wichtig, da bei Erstellung des Fluges es eine Abfrage geben wird, ob der eingegebene Destinationcountry ein ‚dangerouscountry' (ArrayList) ist, wenn ja muss die Fluggesellschaft noch eine zusätzliche Information über den Flug einfügen.
  Methoden gibt es in dieser Klasse keine außerordentlichen, da es sich im Prinzip nur um Getter und Setter handelt.
- **Klasse Flugmanagment:**
  Attribute hat diese Klasse nur eine Referenz der Klasse FlugDAO bzw. ein Objekt der Klasse SerializedFlugDAO.
  Mittels diesem können nun folgende Methoden realisiert werden:
  - ❖ addFlight (Flight) void
  - ❖ deleteFlight (int) void
  - ❖ listFlights () string
  - ❖ meanprice () double
  - ❖ meanutilization () double
  - ❖ getFlightDAO () FlightDAO
  - ❖ getFlightDetails (int) string
  - ❖ seeFStatistics () string
  - ❖ setParameters (int, int) void
  - ❖ changeTicketprice (int, double) void

- **Klasse Analyst:**

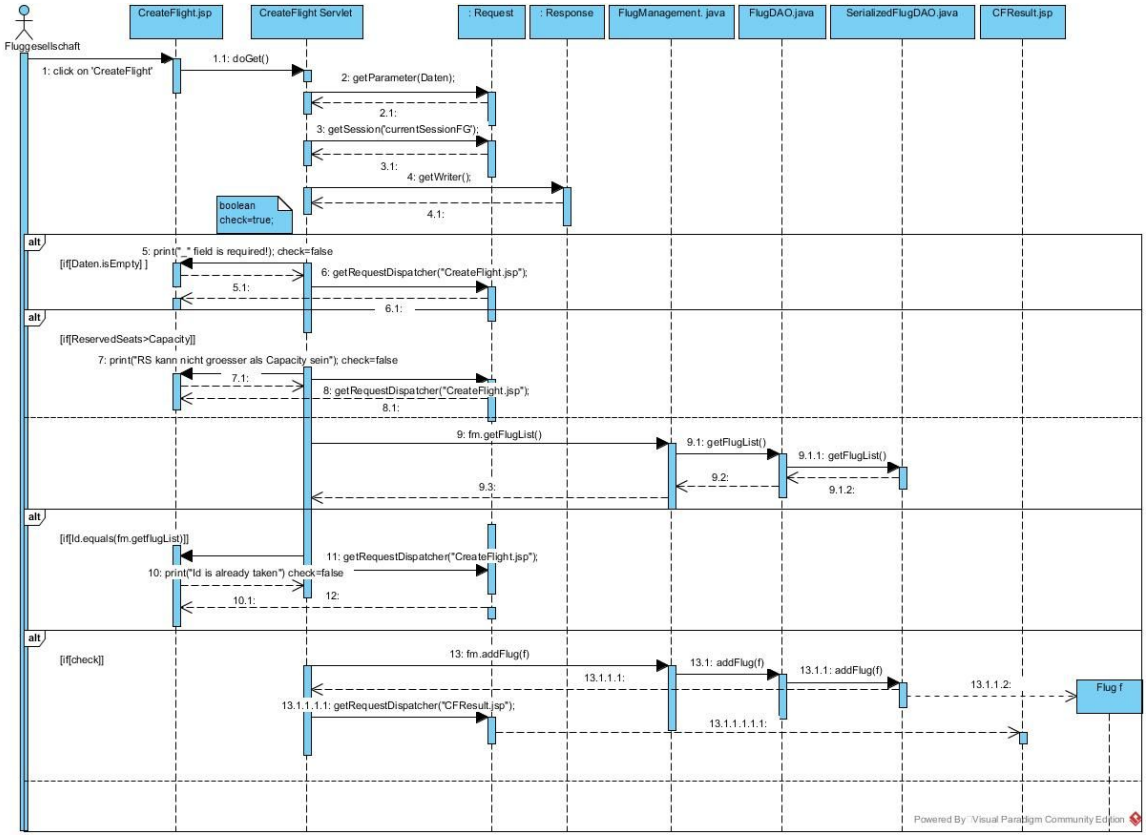
Gleich wie die Klasse Fluggesellschaft erbt auch die Klasse Analyst von der Klasse User. Des Weiteren hat ein Analyst der vollständigkeit halber einen Vor – und Nachnamen. Die Klasse Analyst hat ansonsten nur set und get Methoden.

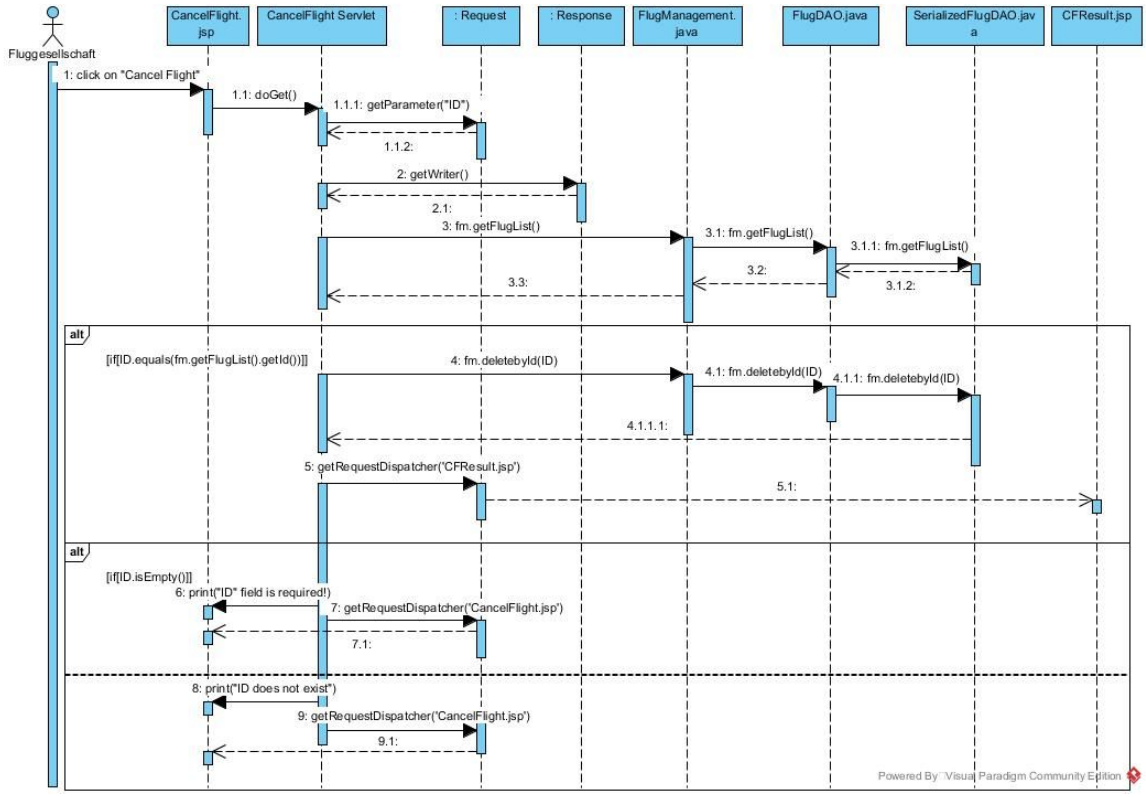
- **Klasse Customer:**
  Die Klasse Customer erbt auch von der Klasse User und hat noch zusätzlich Attribute wie Vorname, Nachname, Geburtsdatum und Adresse, welche benötigt werden um einen Flug zu buchen.
  Bei den Methoden handelt es sich um Setter und Getter.
- **Klasse Buchung:**
  Um die Buchungen persistent festzuhalten haben wir uns entschieden eine eigene Klasse Buchungen samt Serialisierungsklassen zu implementieren, da dadurch der Zugriff bei Stornierungen eines einzelnen Kunden sowie des ganzen Fluges erleichtert wird.
  Attribute: BuchungsNr - int, Booker - Customer, Flight - Flug
- **Serialisierungsklassen und Interfaces:**
  Für jede der oben beschriebenen Klassen (+ die Klasse Combi) gibt es Serialisierungsklassen und Interfaces um die Daten auch persistent festzuhalten.

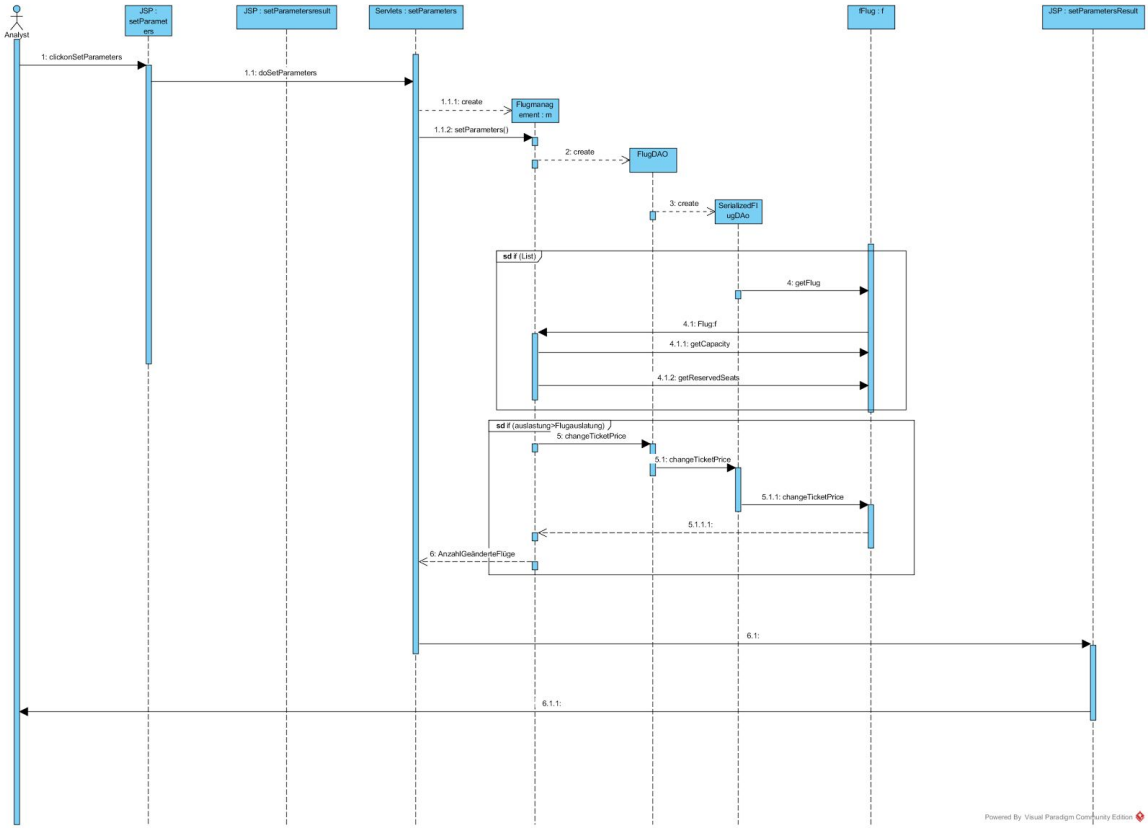
2 Use Case Realization Design

Modellieren sie, wie die zentralen Use Cases ihres Systems mittels Designklassen realisiert werden. Modellieren sie dabei die wesentlichen statischen und dynamischen Aspekte jedes Use Cases mit geeigneten Klassendiagrammen und Sequenzdiagrammen.

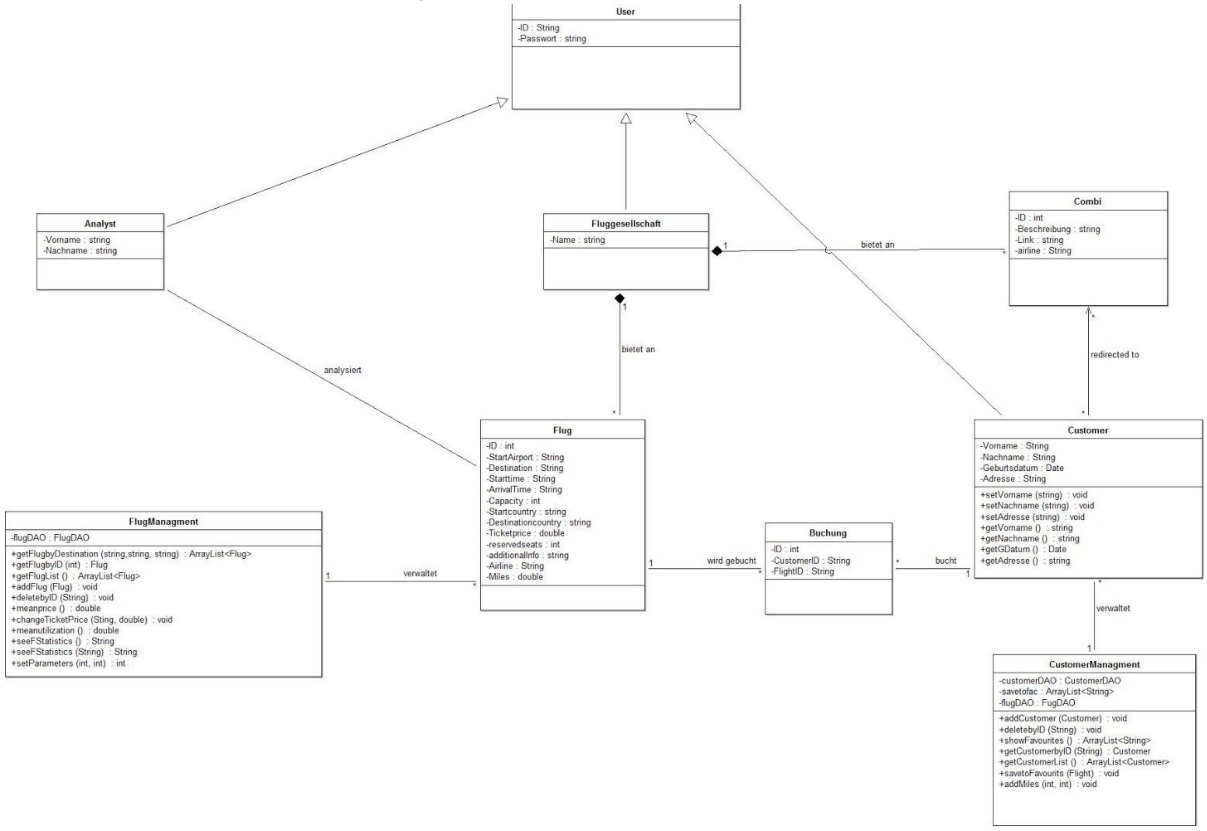
USE CASE


Fluggesellschaft
CreateFlight.jsp
CreateFlight Servlet
: Request
: Response
FlugManagement. java
FlugDAO.java
SerializedFlugDAO.java
CFResult.jsp
1: click on 'CreateFlight'
1.1: doGet()
2: getParameter(Daten);
2.1:
3: getSession('currentSessionFG');
3.1:
4: getWriter();
4.1:
boolean check=true;
alt
[if[Daten.isEmpty]]
5: print("_" field is required!); check=false
5.1:
6: getRequestDispatcher("CreateFlight.jsp");
6.1:
alt
[if[ReservedSeats>Capacity]]
7: print("RS kann nicht groesser als Capacity sein"); check=false
7.1:
8: getRequestDispatcher("CreateFlight.jsp");
8.1:
9: fm.getFlugList()
9.1: getFlugList()
9.1.1: getFlugList()
9.1.2:
9.2:
9.3:
alt
[if[Id.equals(fm.getflugList)]]
10: print("Id is already taken") check=false
10.1:
11: getRequestDispatcher("CreateFlight.jsp");
12:
alt
[if[check]]
13: fm.addFlug(f)
13.1: addFlug(f)
13.1.1: addFlug(f)
13.1.1.2:
Flug f
13.1.1.1:
13.1.1.1.1: getRequestDispatcher("CFResult.jsp");
13.1.1.1.1.1:
Powered By Visual Paradigm Community Edition


Fluggesellschaft
CancelFlight. jsp
CancelFlight Servlet
: Request
: Response
FlugManagement. java
FlugDAO.java
SerializedFlugDAO.jav a
CFResult.jsp
1: click on "Cancel Flight"
1.1: doGet()
1.1.1: getParameter("ID")
1.1.2:
2: getWriter()
2.1:
3: fm.getFlugList()
3.1: fm.getFlugList()
3.1.1: fm.getFlugList()
3.1.2:
3.2:
3.3:
alt
[if[ID.equals(fm.getFlugList().getId())]]
4: fm.deletebyId(ID)
4.1: fm.deletebyId(ID)
4.1.1: fm.deletebyId(ID)
4.1.1.1:
5: getRequestDispatcher('CFResult.jsp')
5.1:
alt
[if[ID.isEmpty()]]
6: print("ID" field is required!)
7: getRequestDispatcher('CancelFlight.jsp')
7.1:
8: print("ID does not exist")
9: getRequestDispatcher('CancelFlight.jsp')
9.1:
Powered By Visual Paradigm Community Edition

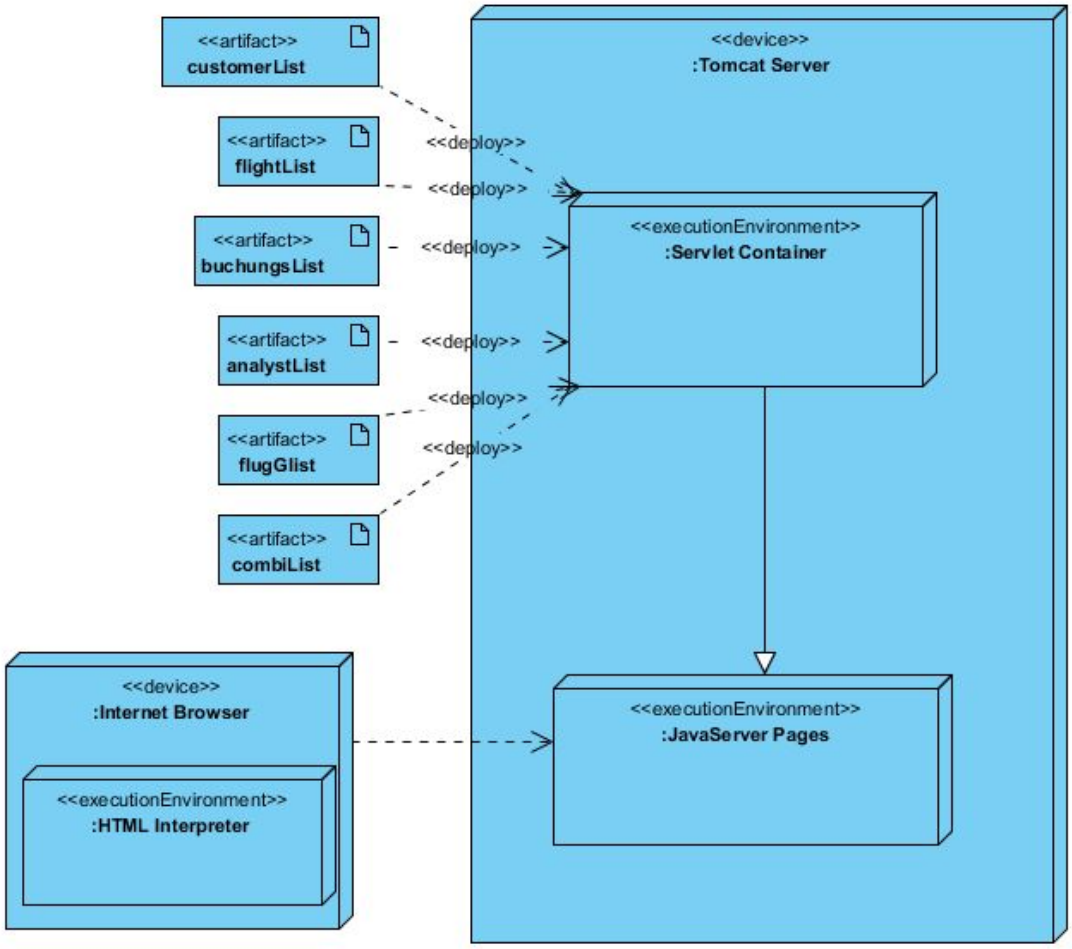

Analyst
JSP : setParameters
JSP : setParametersresult
Servlets : setParameters
fFlug : f
JSP : setParametersResult
1: clickonSetParameters
1.1: doSetParameters
1.1.1: create
Flugmanagement : m
1.1.2: setParameters()
2: create
FlugDAO
3: create
SerializedFlugDAo
sd if (List)
4: getFlug
4.1: Flug:f
4.1.1: getCapacity
4.1.2: getReservedSeats
sd if (auslastung>Flugauslatung)
5: changeTicketPrice
5.1: changeTicketPrice
5.1.1: changeTicketPrice
5.1.1.1:
6: AnzahlGeänderteFlüge
6.1:
6.1.1:
Powered By Visual Paradigm Community Edition

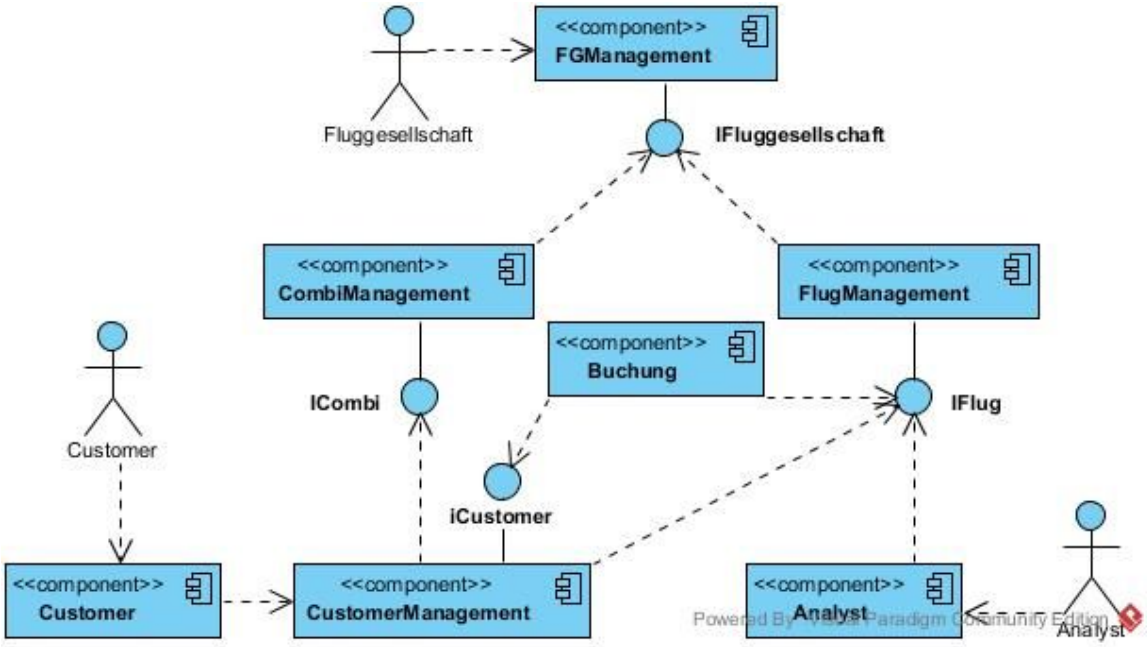
3 Übersichtsklassendiagramm



4 Architekturbeschreibung

Beschreiben sie die Architektur ihres Systems mittels Komponentendiagrammen und Deploymentdiagrammen.


<<artifact>>
customerList
<<artifact>>
flightList
<<artifact>>
buchungsList
<<artifact>>
analystList
<<artifact>>
flugGlist
<<artifact>>
combiList
<<deploy>>
<<deploy>>
<<deploy>>
<<deploy>>
<<deploy>>
<<deploy>>
<<device>>
:Tomcat Server
<<executionEnvironment>>
:Servlet Container
<<executionEnvironment>>
:JavaServer Pages
<<device>>
:Internet Browser
<<executionEnvironment>>
:HTML Interpreter


Fluggesellschaft
<<component>>
FGManagement
IFluggesellschaft
<<component>>
CombiManagement
<<component>>
FlugManagement
<<component>>
Buchung
ICombi
IFlug
Customer
iCustomer
<<component>>
Customer
<<component>>
CustomerManagement
<<component>>
Analyst
Analyst
Powered By Visual Paradigm Community Edition