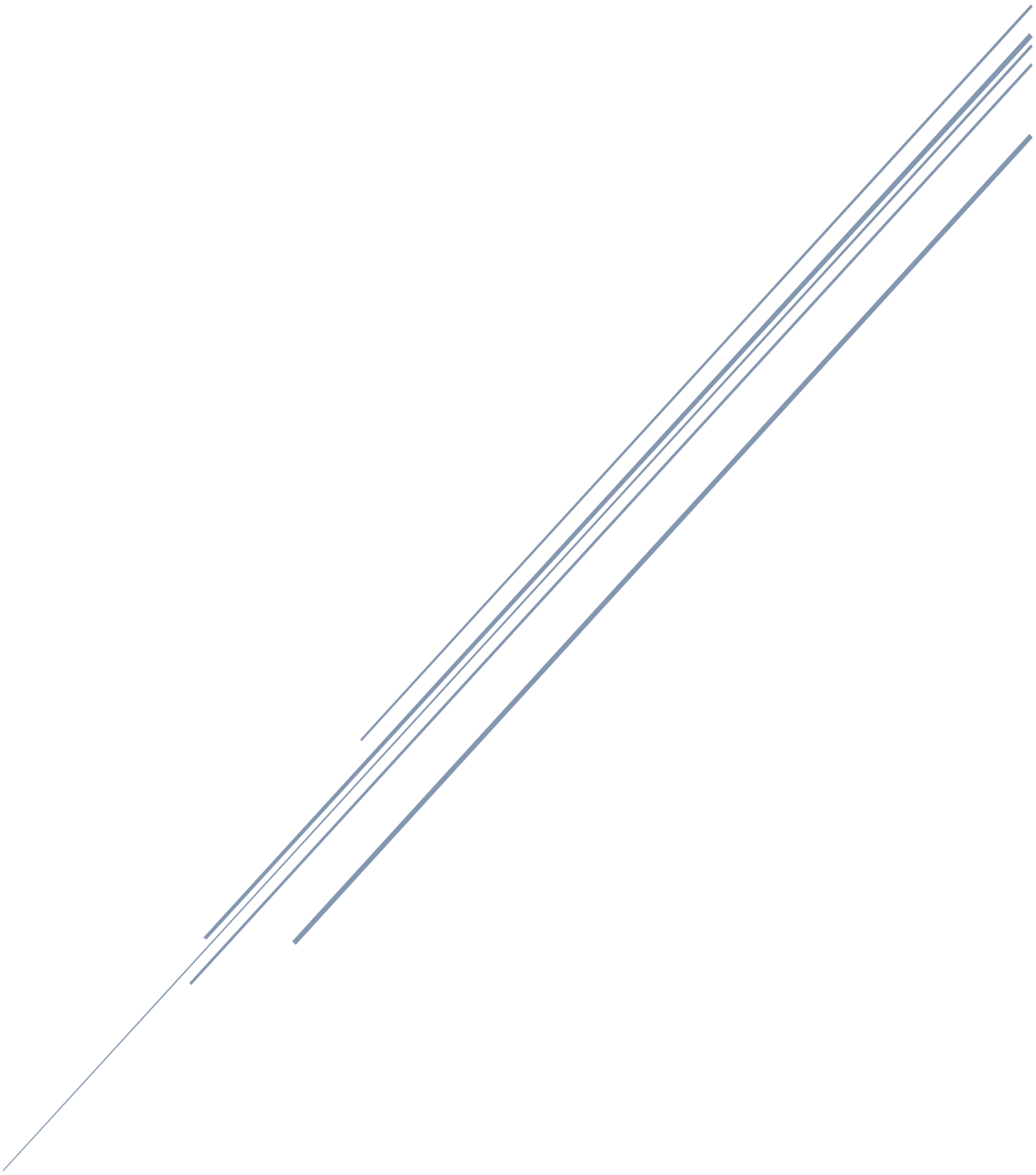
# HAW HAMBURG

## Software Engineering I



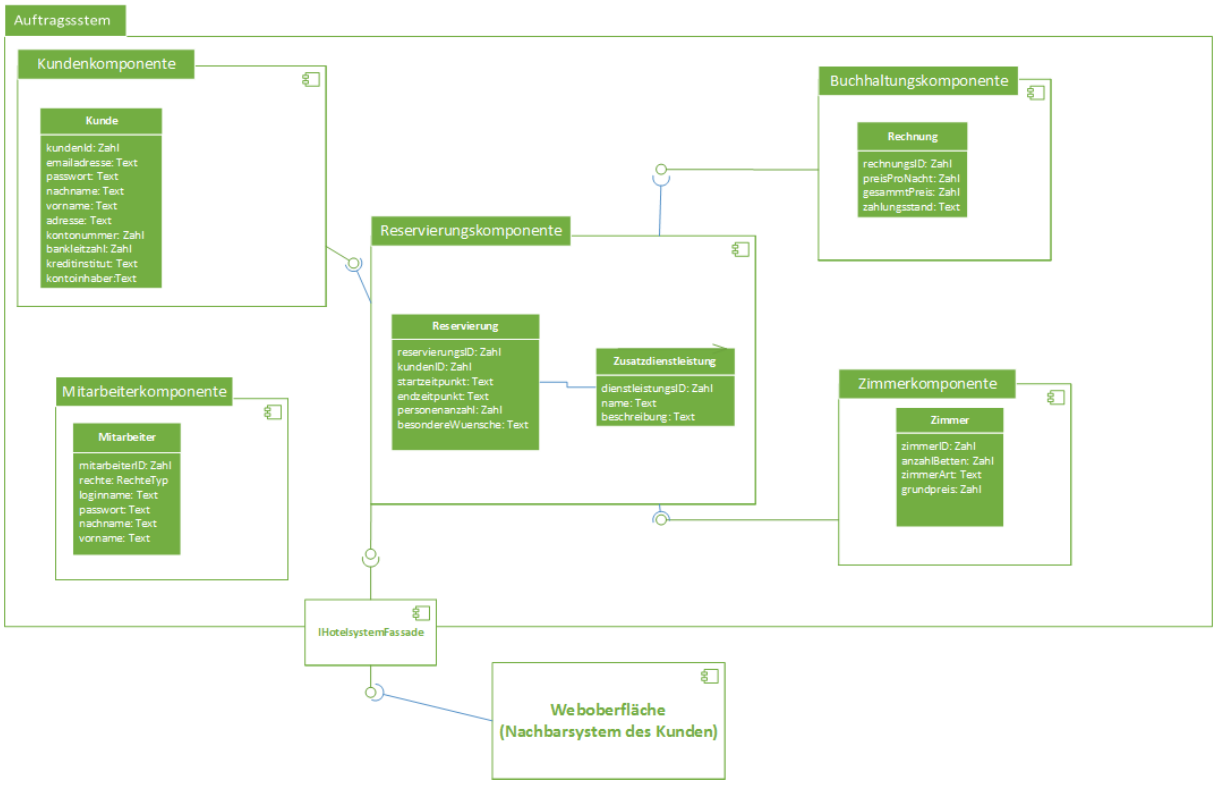
HAW Hamburg
Steffen Giersch & Maria Lüdemann

Steffen Giersch & Maria Lüdemann
HAW HAMBURG
12.12.2013

# Inhaltsverzeichnis

Steffen Giersch & Maria Lüdemann
HAW HAMBURG
12.12.2013

# Aufgabe 6 Erster Architektur Entwurf




Wir haben den Systemkern in fünf Komponenten aufgeteilt die nicht alle gegenseitig aufeinander zugreifen können. Als Herzstück dient hier die Reservierungskomponente denn sie greift auf die Interfaces der Kundenkomponente der Buchhaltungskomponente und der Zimmerkomponente zu und stellt der Fassade ein eigenes Interface zur Verfügung. Die Fassade dann stellt die Verbindung zur Weboberfläche. Es ist nicht von Nöten, dass die Komponenten sich untereinander kennen oder auf die Reservierungskomponente zugreifen können und somit haben wir dies von Anfang an unterbunden um durch Kapselung der Daten eine weitere Sicherheitsebene zu schaffen. Einzig die Reservierungskomponente ist in der Lage auf Datenzuzugreifen und auch sie erhält keine ganzen Entitäten sondern nur atomare Daten.

# Aufgabe 7 Systemoperationen

**IHotelFassadeInterface:**

```
public interface IHotelsystemFassade {

        /**Kommando:
         * Muss vor jeder anderen Funktion aufgerufen und erfolgreich beendet werden.
         * Einlogvorgang des Benutzers.  wenn falsch.
         * @pre passwort ist angemessen verschlüsselt
         * @param name Name des Kunuden != null
         * @param passwort Passwort (verschlüsselt) des Kunden
```

```
     * @throws IllegalArgumentException Exception für den Fall, dass das Passwort nicht
     zum Namen passt falsch eingegeben wurde
     */
     void login(String name, String passwort) throws IllegalArgumentException;

     /**Abfrage:
     * Kann nach dem einloggen jederzeit aufgeführt werden
     * Sucht für die Auswahloberfläche eine Liste an Zimmern die in dem gewählten
     Zeitraum nicht belegt sind.
     * @pre start < ende
     * @param start Start des Belegungszeitraums != null
     * @param ende Ende des Belegungszeitraums != null
     * @return Liste der passenden Zimmer. Wenn keine passenden Zimmer gefunden
     wurde wird die leere Liste zurück gegeben
     * @throws IllegalArgumentException Wenn start >= ende
     */
List<Integer> findeFreieZimmerMit(DatumTyp start, DatumTyp ende) throws
IllegalArgumentException;

     /**Abfrage:
     * Kann nach dem einloggen jederzeit aufgeführt werden
     * Sucht für die Auswahloberfläche eine Liste an Zimmern des passenden Typs die in
     dem gewählten Zeitraum nicht belegt sind.
     * @pre start < ende und 0 < zimmertyp < 4
     * @param start Start des Belegungszeitraums != null
     * @param ende Ende des Belegungszeitraums != null
     * @param zimmertyp Typnummer des gewählten Zimmers != null
     * @return Liste der passenden Zimmer. Wenn keine passenden Zimmer gefunden
     wurde wird die leere Liste zurück gegeben
     * @throws IllegalArgumentException Wenn start >= ende oder ein nicht existenter
     Zimmertyp gewählt wurde
     */
List<Integer> findeFreieZimmerMit(DatumTyp start, DatumTyp ende, int zimmertyp) throws
IllegalArgumentException;

     /**Abfrage:
     * Kann nach dem einloggen jederzeit aufgeführt werden
     * Sucht für die Auswahloberfläche eine Liste an Zimmern des passenden Typs und
     genügend Platz die in dem gewählten Zeitraum nicht belegt sind.
     * @pre start < ende und 0 < zimmertyp < 4 und 0 < anzahlPersonen
     * @param start Start des Belegungszeitraums != null
     * @param ende Ende des Belegungszeitraums != null
     * @param zimmertyp Typnummer des gewählten Zimmers != null
     * @param anzahlPersonen Anzahl der unterzubringenden Personen != null
     * @return Liste der passenden Zimmer. Wenn keine passenden Zimmer gefunden
     wurde wird die leere Liste zurück gegeben
     * @throws IllegalArgumentException Wenn start >= ende oder ein nicht existenter
     Zimmertyp gewählt wurde oder anzahlPersonen < 1
     */
List<Integer> findeFreieZimmerMit(DatumTyp start, DatumTyp ende, int zimmertyp, int
anzahlPersonen) throws IllegalArgumentException;
```

```
        /**Abfrage:
         * Kann jederzeit durchgeführt werden.
         * Lässt sich von einem Zimmer Informationen geben
         * @param zimmernummer ID des Zimmers != null
         * @return List<Integer> in der Form [AnzahlBetten, ZimmerArt, Grundpreis]
        * @throws IllegalArgumentException Wenn eine nicht existierende Zimmernummer
        gewählt wurde
         */
List<Integer> getZimmerInformationen(Integer zimmernummer) throws
IllegalArgumentException;

        /**Kommando:
         * Kann nach dem einloggen jederzeit, allerdings maximal ein mal pro Zimmer pro Tag
         pro Benutzer ausgeführt werden
         * Blockiert Zimmer im System um zu verhindern, dass Zimmer öfter reserviert werden ´
           können.
         * Diese Blockierung wird automatisch nach 10 Minuten aufgehoben.
         * @pre zimmernummer gehört zu einem existierenden Zimmer das noch nicht belegt
         ist
         * @param zimmernummer Die Zimmernummer und gleichzeitig die ID des Zimmers !=
         null
         * @throws IllegalArgumentException Wenn das Zimmer schon belegt oder geloggt ist
        oder nicht existiert
         */
void lockZimmer(int zimmernummer) throws IllegalArgumentException;

        /**Kommando:
         * Kann nach dem einloggen jederzeit aufgeführt werden
         * Erstellt die fertige Reservierung. Diese ist nun fix und eine Rechnung kann verschickt
        werden.
         * @pre start < ende und 0 < zimmertyp < 4 und 0 < anzahlPersonen  und 0 <
        |zimmernummerZuPersonenzahlMap|
         * @param kundenID ID des Kunden != null
         * @param start Start des Belegungszeitraumes != null
         * @param ende Ende des Belegungszeitraumes != null
        * @param zimmernummerZuPersonenzahlMap Map die von Zimmernummer auf die
        Anzahl der Personen in diesem Zimmer mapt != null
         * @param zusatzleistungen Liste der hinzugebuchten Zusatzdienstleistungen, leere
        Liste wenn keine gewählt
         * @param wuensche Spezielle Wünsche des Kunden als einfacher String, leerer String
        wenn keine Wünsche
         * @return Ist die Reservierung erfolgreich gewesen (true) oder nicht (false)
         */
        boolean newReservierung(int kundenID, DatumTyp start, DatumTyp ende,
Hash<Integer, Integer> zimmernummerZuPersonenzahlMap, List<Integer> zusatzleistungen,
String wuensche);

}
```

Steffen Giersch & Maria Lüdemann
HAW HAMBURG
12.12.2013

# Aufgabe 8 Interner Schnittstellen Entwurf

Zuweisung zu verantwortlichen Komponenten (Kurzvorm):

IKundenkomponenteServices:

- void login(String name, String passwort) throws IllegalArgumentException;

IReservierungskomponenteServices:

- List<Integer> findeFreieZimmerMit(DatumTyp start, DatumTyp ende) throws IllegalArgumentException;
- List<Integer> findeFreieZimmerMit(DatumTyp start, DatumTyp ende, int zimmertyp) throws IllegalArgumentException;
- List<Integer> findeFreieZimmerMit(DatumTyp start, DatumTyp ende, int zimmertyp, int anzahlPersonen) throws IllegalArgumentException;
- void lockZimmer(int zimmernummer) throws IllegalArgumentException;
- boolean newReservierung(int kundenID, DatumTyp start, DatumTyp ende, Hash<Integer, Integer> zimmernummerZuPersonenzahlMap, List<Zusatzleistung> zusatzleistungen, String wuensche);

IZimmerkomponenteServices:

Dies ist allerdings eine Zusatzfunktion die durch Aufgabe 8 kommt und in Aufgabe 7 noch nicht dazu gehört deswegen ausführlich mit Kommentaren hier dargestellt.

```
/**Abfrage:
 * Kann jederzeit aufgerufen werden
 * Gibt zu dem eingegebenen Zimmer die dazu passende Art des Zimmers zurück
 * @param zimmer ZimmerID des abzufragenden Zimmers
 * @return ID der Zimmerart
 * @throws IllegalArgumentException Wenn das Zimmer nicht existiert
 */
Integer getZimmerArt(Integer zimmer) throws IllegalArgumentException;
```

Wir haben uns die Systemoperation Zimmer findeFreieZimmerMit genommen um sie hier als Sequenzdiagramm darzustellen.

Steffen Giersch & Maria Lüdemann
HAW HAMBURG
12.12.2013