

Dokumentation Team A

Dominik Aigner,
Nina Hartmann,
Samuel Jäger,
Ida Mazinger,
Clara Tschamon

Inhalt

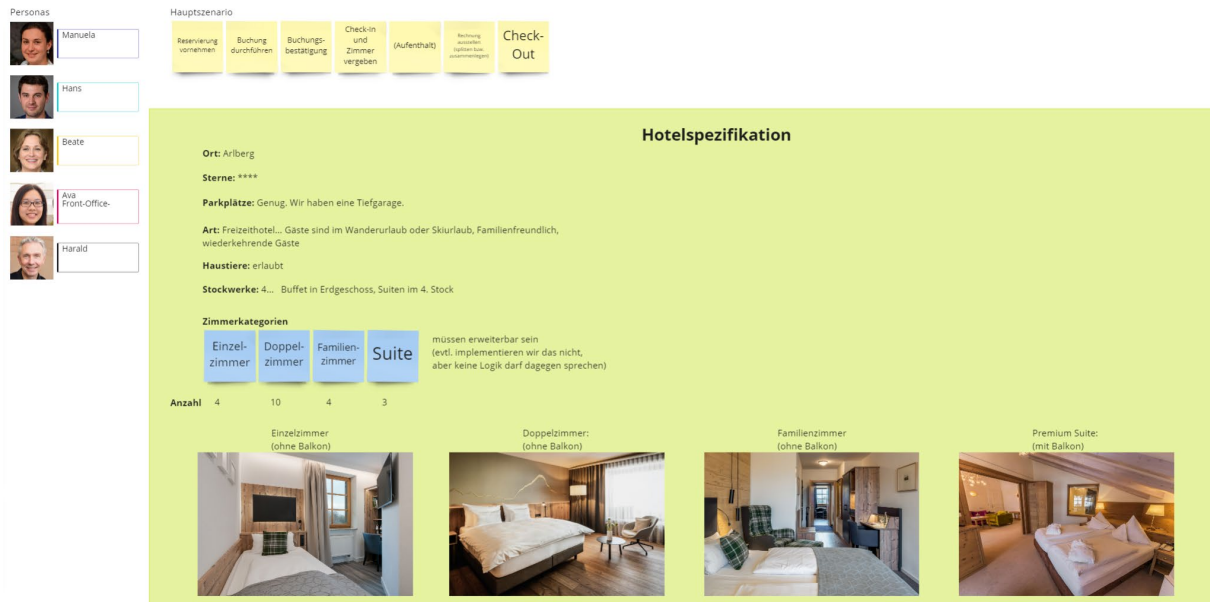
1. Vision.....	2
2. Projektablauf.....	3
3. Architektur.....	4
4. Verwendete Technologien	6
5. Fehlerbehandlung	7
1. Logging.....	9
2. Was noch fehlt & Lessons learned.....	10
3. Anleitung zur Erweiterbarkeit.....	12
4. Teamarbeit, Zeitaufwand, Schwierigkeiten.....	13

1. Vision

Wir wollten ein Hotelmanagementsystem für ein mittelgroßes Hotel entwickeln.

Das Folgende Bild ist unsere Vorstellung von dem Zielhotel.

Wichtig ist zu erwähnen, dass das Hotelmanagementsystem für jede beliebige Art von Hotels angewendet werden soll. Das heißt, dass das System erweiterbar und nicht einschränkend sein soll.



2. Projektablauf

Sprint 1	
geplant	umgesetzt
Walk-In vornehmen	nein

Sprint 2	
geplant	umgesetzt
Walk-In vornehmen	ja
Check-Out vornehmen	ja
Einfache Rechnung erstellen	ja

Sprint 3	
geplant	umgesetzt
Walk-In und Check-Out mit Searchbar ausstatten	ja
Rechnung als PDF erstellen	ja
Buchungsübersicht erstellen	ja

Sprint 4	
geplant	umgesetzt
Reservierung über Web inclusive Reservierungsübersicht	ja
Reservierung bestätigen über Buchungssystem	ja
Email Mock	ja
	Check-In: ja

3. Architektur

- wird von seinem ViewController kontrolliert:
- wird von ihm befüllt und ausgelesen
- löst z.B. onAction-Aktionen aus

- ein ViewController pro fxmI-Datei
- wenn mehr als eine fxmI-Datei zur Handhabung von einem UseCase gebraucht wird, gibt es einen, der für die anderen zuständig ist (wie bei WalkIn)
- handhabt alle @FXML-Objekte (befüllen, onAction, auswerten)
- ersetzen des aktuellen Fensters durch `MainApplication.getMainViewController.loadIntoContentArea(...)`
- hier werden Exceptions abgefangen und gehandhabt

- ein UseCaseController pro UseCase (z.B. WalkIn, CheckOut)
- enthält DTOs welche vom ViewController befüllt werden
- kennt seinen ViewController nicht

- alle Transaktionen zwischen View- und Domain-Schicht laufen über ihn
- übernimmt Aufgaben wie get, getAll, save

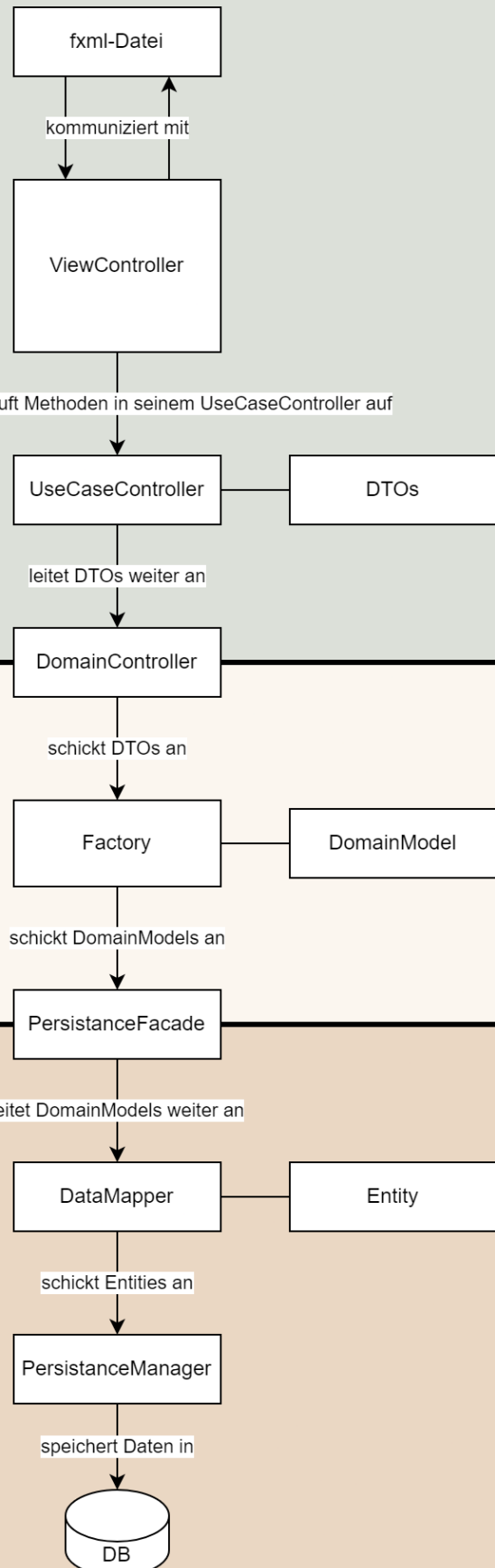
- eine Factory pro DomainModel-Typ
- übersetzt DTO in DomainModel und umgekehrt
- hat Methoden wie get, getAll, save (Factory entscheidet, ob auf der PersistenceFacade insert oder store aufgerufen wird)

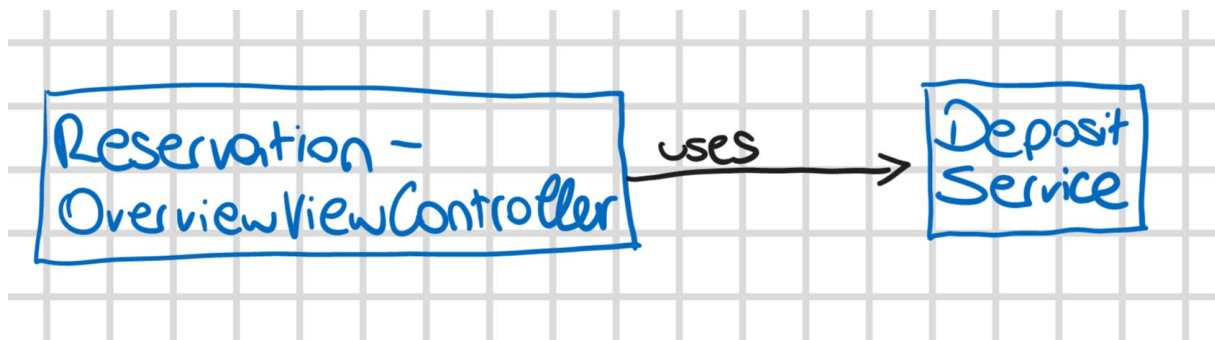
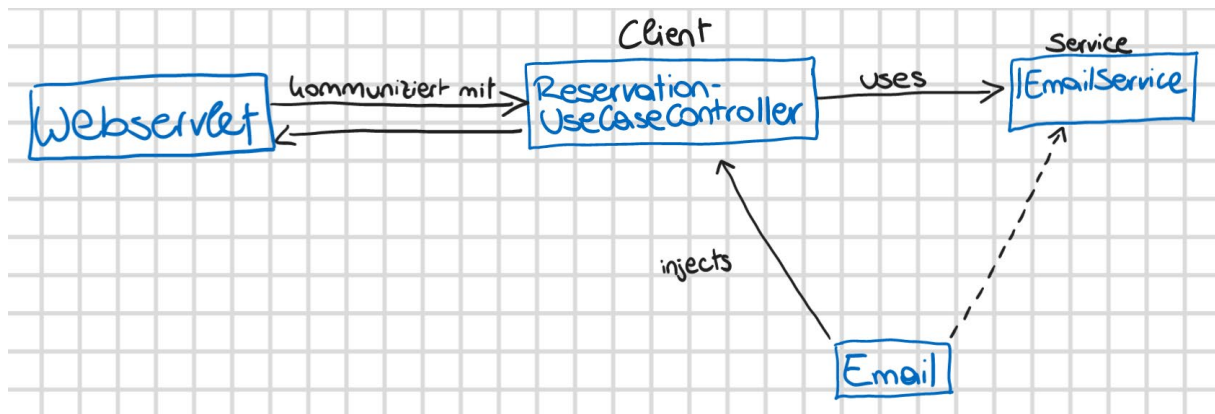
- alle Transaktionen zwischen Domain- und Persistence-Schicht laufen über ihn
- übernimmt Aufgaben wie get, getAll, insert und store

- ein DataMapper pro Entity-Typ
- übersetzt DomainModel in Entity und umgekehrt
- hat Methoden wie get, getAll, insert, insert und store

- Speichern, Lesen und Updaten der Daten

- momentan H2
- speichert alle Daten
- löschen darf man nie





4. Verwendete Technologien

- Gradle,
- JavaFX,
- ControlsFX für CheckComboBox und SearchableComboBox
- Hibernate,
- H2 Database,
- Log4j (only for debugging),
- Apache PDF Box,
- Jakarta Servlets,
- Tomcat 10.0.27

5. Fehlerbehandlung

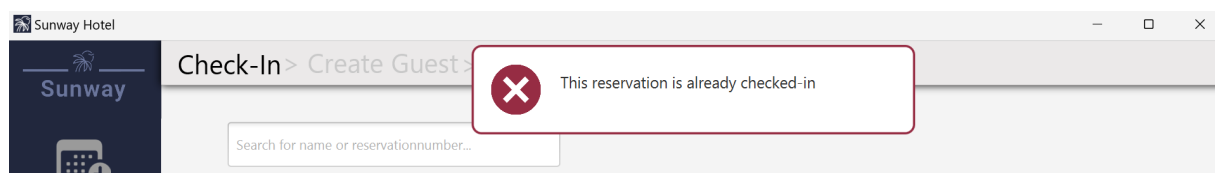
In Klasse MainController gibt es die Methode `alert()`

```
50 @ public void alert(String message, WarningType warningType) {
51     String color = "#21273d";
52     String iconName = "";
53     switch (warningType) {
54         case ERROR:
55             color = "#962c43";
56             iconName = "error";
57             break;
58         case WARNING:
59             color = "#D3AB1B";
60             iconName = "warning";
61             break;
62         case INFORMATION:
63             color = "#21273d";
64             iconName = "information";
65             break;
66         case CONFIRMATION:
67             color = "#30A41E";
68             iconName = "confirmation";
69             break;
70     }
71
72     AnchorPane alertPane = new AnchorPane();
73     alertPane.setStyle("-fx-border-color: " + color + ";" +
74         "-fx-border-width: 2px;" +
75         "-fx-background-color: white;" +
76         "-fx-min-height: 80; -fx-max-height: 120;" +
77         "-fx-min-width: 400; -fx-max-width: 400;" +
78         "-fx-border-radius: 8px;" +
79         "-fx-background-radius: 8px;");
```



Kann dann so aufgerufen werden...



```
private void searchedReservationsListViewChanged(){
    ReservationDTO selectedReservation = searchedReservationsListView.getSelectionModel().getSelectedItem();
    if(selectedReservation != null){
        searching = false;
        if(selectedReservation.getBooking() != null && selectedReservation.getBooking().getCheckInDatetime() != null){
            MainApplication.getMainController().alert(message: "This reservation is already checked-in", WarningType.ERROR);
        }
    }
}
```

So können dem User schön und sinnvoll Rückmeldungen gegeben werden.



Hier sieht man wo wir Exceptions abgefangen haben.

Find in Files 22 matches in 11 files ☐ File mask: *.java  

try{   | Cc W *

In Project	Module	Directory	Scope
try {			CheckIn2ViewController.java 159
try {			CheckIn2ViewController.java 172
try {			CheckIn3ViewController.java 150
try {			CheckIn3ViewController.java 168
try {			CheckInViewController.java 69
try {			CheckOut1ViewController.java 231
try {			CheckOut1ViewController.java 242
try {			CheckOut2ViewController.java 73
try {			CheckOut2ViewController.java 83
try {			CheckOut2ViewController.java 104
try {			HomeController.java 109
try {			HomeController.java 120
try {			ReservationOverviewViewController.java 301
try {			ReservationOverviewViewController.java 305

1. Logging

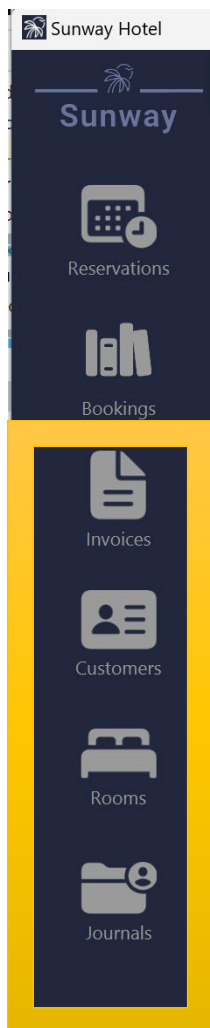
Immer wenn eine Buchung/Reservierung erstellt wird, verändert wird usw. wird diese Information in die Konsole und in eine Datei geloggt. Der Logger wird in der Persistence Facade aufgerufen.

```
PersistenceFacade.java x
1 usage ClaraTschanon
59 public static Long insertReservation(Reservation reservation) {
60     long number = ReservationDataMapper.instance().insert(reservation);
61     logger.info("Reservation with number " + number + " was inserted.");
62     return number;
63 }
64
1 usage ClaraTschanon
65 @ public static void storeReservation(Reservation reservation){
66     logger.info("Reservation with number " + reservation.getNumber() + " was stored.");
67     ReservationDataMapper.instance().store(reservation);
68 }
69
1 usage ClaraTschanon
70 public static void deleteReservation(Reservation reservation){ReservationDataMapper.instance().delete(reserva
71
ClaraTschanon +1
72 @SuppressWarnings("rawtypes")
73 public static Optional<Customer> getCustomer(Long id) { return CustomerDataMapper.instance().get(id); }
76
1 usage ClaraTschanon
77 @ public static ArrayList<Customer> getSavedCustomers(){ return CustomerDataMapper.getSavedCustomers();}
78
1 usage ClaraTschanon +1
79 public static Long insertCustomer(Customer customer) {
80     long number = CustomerDataMapper.instance().insert(customer);
81     logger.info("Customer with number " + number + " was inserted.");
82     return number;
83 }
84
```

2. Was noch fehlt & Lessons learned...

Was noch fehlt

- Reservierung bearbeiten (Bei einchecken können Daten geändert werden aber einfach so kann die Reservierung nicht bearbeitet werden).
- Es wäre nicht schwer gewesen umzusetzen, man hätte einfach den Walk-In wiederverwenden können.
- Buchung bearbeiten... auch nicht schwer umzusetzen eigentlich.
- Rechnung bearbeiten
- Login / Logout



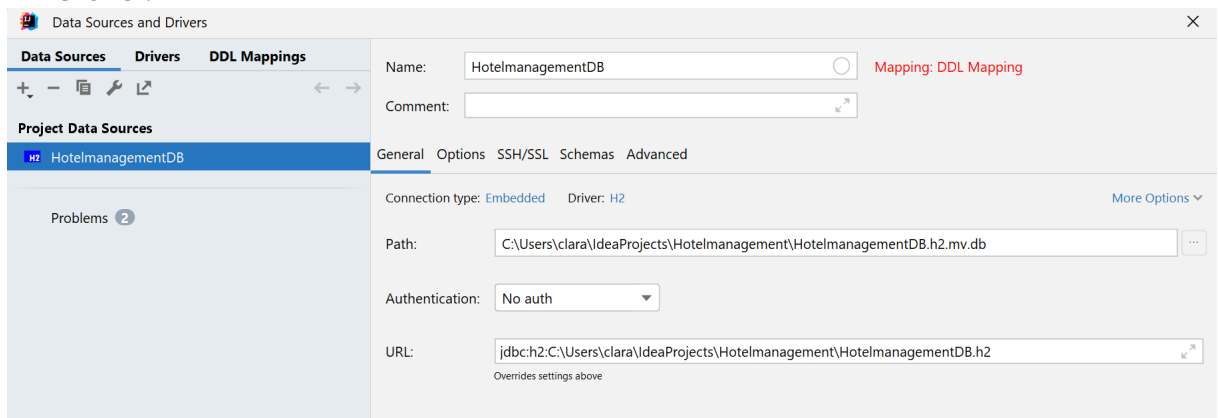
- diese 4 Punkte wurden nicht implementiert.
- Zu Codeverschönerung:
 - a. Codereduzierung durch Anwendung von Generik z.B. in den DataMappern usw.
 - b. In Datei loggen immer wenn neue Buchung erstellt wird / neu Reservierung / neuer Kunde, ... das könnte man in der Persistence Facade machen.

Lessons learned

- Umgang mit den oben genannten Technologien,
- Architekturmuster,
- Teamwork und Scrum

3. Anleitung zur Erweiterbarkeit

- 1) In der Datenbank Configuration den eigenen absoluten Pfad angeben.
➔ Absoluter Pfad damit Tomcat die Datenbank findet ansonst sucht Tomcat im Apache Ordner.



- 2) Im Persistence.xml den eigenen absoluten Pfad zur Datenbank angeben damit Tomcat die Datenbank findet.

```
<!--  
    <property name="jakarta.persistence.jdbc.driver" value="org.h2.Driver" />  
    <property name="jakarta.persistence.jdbc.url" value="jdbc:h2:C:\Users\samuel\Documents\GitHub\Hotelmanagement\HotelmanagementDB.h2" />  
    <!--<property name="jakarta.persistence.jdbc.url" value="jdbc:h2:D:\FHV_Informatik\Semester_3\Software_Engineering\Projekt\HotelmanagementDB.h2" />  
    <property name="jakarta.persistence.jdbc.url" value="jdbc:h2:C:\Users\clara\IdeaProjects\Hotelmanagement\HotelmanagementDB.h2" />  
    <!--  
    <property name="jakarta.persistence.jdbc.url" value="jdbc:h2:C:/Users/ninah/Desktop/FH_Dornbirn/Hotelmanagement/HotelmanagementDB.h2" />  
-->
```

- 3) In E-Mail-Klasse für den DataOutputStream den eigenen absoluten Pfad angeben aus denselben Gründen

```
dataOutputStream = new DataOutputStream(new FileOutputStream( name: "C:\\Users\\clara\\IdeaProjects\\Hotelmanagement\\Emails.txt", append: true));  
dataOutputStream = new DataOutputStream(new FileOutputStream( name: "C:\\Users\\samuel\\Documents\\GitHub\\Hotelmanagement\\Emails.txt", append: true));
```

4. Teamarbeit, Zeitaufwand, Schwierigkeiten

- Ausfälle wegen krank sein
- Es hat lange gebraucht, bis wir klargekommen sind mit GitHub und bei jemandem gab es bis zum Schluss mindestens 1-mal in der Woche schwierige Mergekonflikte, was sehr viel Zeit in Anspruch nahm...
- Daily Stand-Up wurde manchmal vergessen
- Unfaire Verteilung von Zeitaufwand.
- Geschätzter Zeitaufwand: ø39h pro Sprint pro Person