

Dokumentation Team A

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

Inhalt

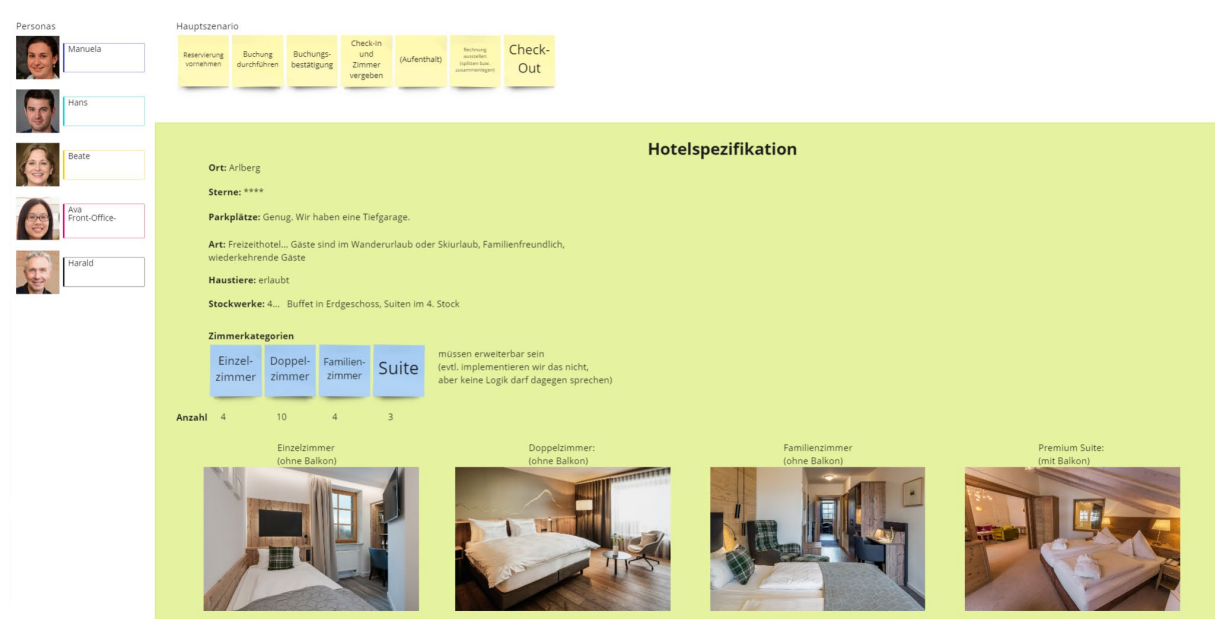
1. Vision.....	2
2. Projektablauf.....	4
3. Architektur.....	5
4. Verwendete Technologien	7
5. Fehlerbehandlung	8
6. Logging.....	11
7. Was noch fehlt & Lessons learned.....	12
8. Anleitung zur Erweiterbarkeit.....	14
9. Teamarbeit, Zeitaufwand, Schwierigkeiten.....	15

1. Vision

Die Aufgabe war, ein Hotelmanagementsystem für ein mittelgroßes (ca. 25 Zimmer) Hotel zu entwickeln.

Das folgende Bild ist unsere Vorstellung von dem Zielhotel.

Wichtig ist zu erwähnen, dass das Hotelmanagementsystem auch für kleinere und größere Hotels angewendet können werden soll. Das heißt, dass das System erweiterbar und nicht einschränkend sein soll.



The task was to develop a hotel management system with the following functionalities:

- Room reservation
- Processing of guest arrivals
- Automatic billing
- Processing of guest departures
- Contingent management for travel agencies
- Contract partner management (travel agencies, companies)
- Possibility of control of all performed actions in the computer system
- Creation of various evaluations and statistics

It was clear from the beginning that we would not be able to implement all of them, therefore we defined the so-called "Minimal Viable Product".

A minimal viable product (MVP) is a product that has the minimum set of features necessary.

Our MVP consisted of these features:

- Walk-In
- Check-In
- Check-Out
- Make a reservation through a webform
- Reservation Overview Table
- Booking Overview Table
- Create a simple invoice
- Simulate sending an email after making a reservation
- Import payment information from accounting system

Team A

In total we had 10 weeks for planning and deploying.

At the beginning we were 5 people (Dominik Aigner, Samuel Jäger, Nina Hartmann, Ida Mazinger and Clara Tschamon).

After 6 weeks Ida could not continue working with us, so we were half a teammember less. (Even though she was on sick leaf, she managed to help us sometimes.)

The purpose of the project was to learn the agile working method scrum and to expand our programming skills.

2. Projektablauf

Sprint 1	
geplant	umgesetzt
Walk-In vornehmen	nein

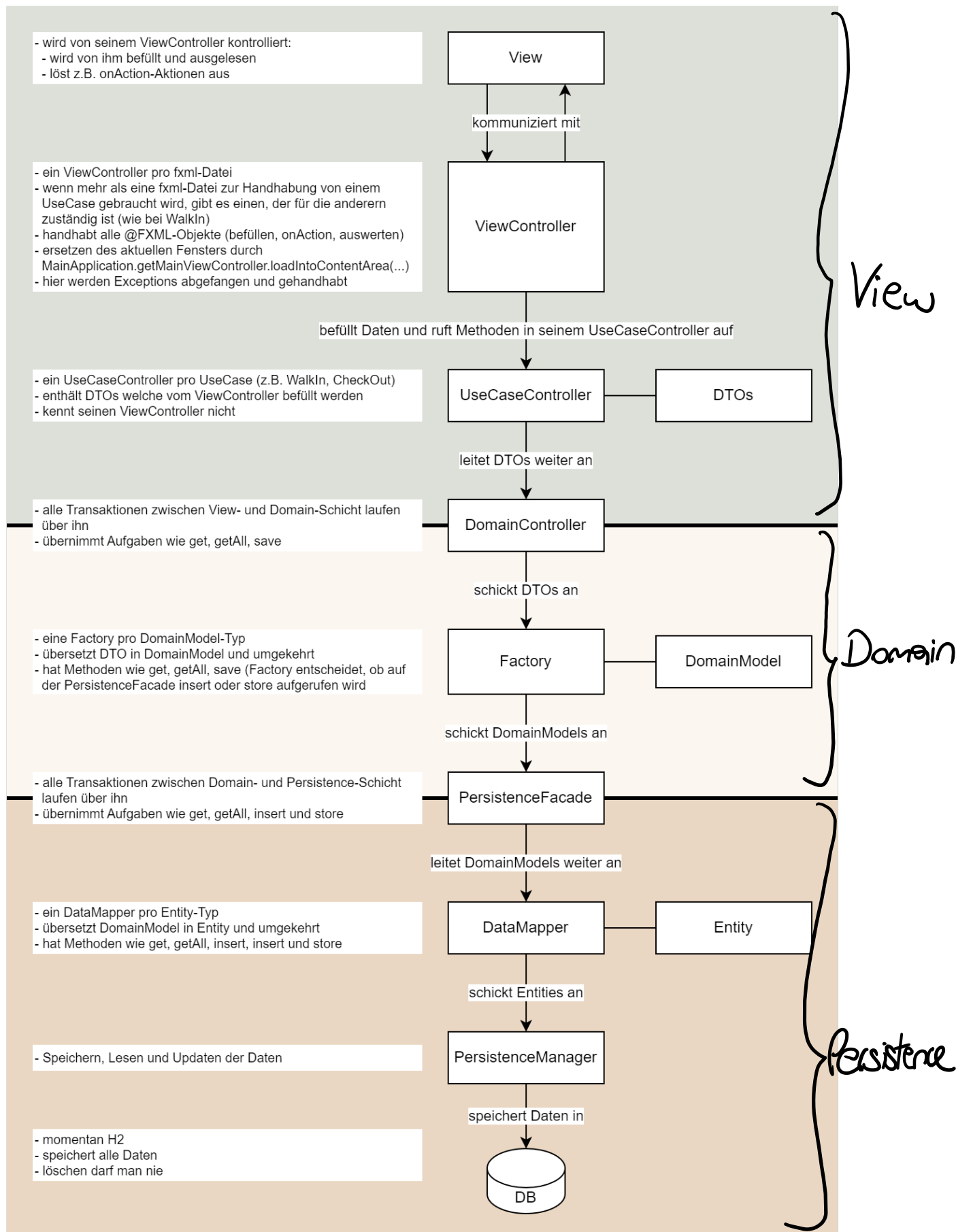
Sprint 2	
geplant	umgesetzt
Walk-In vornehmen	ja
Check-Out vornehmen	ja
Einfache Rechnung erstellen	ja

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 Buchhaltungssystem	ja
Email Mock	ja
	Check-In: ja

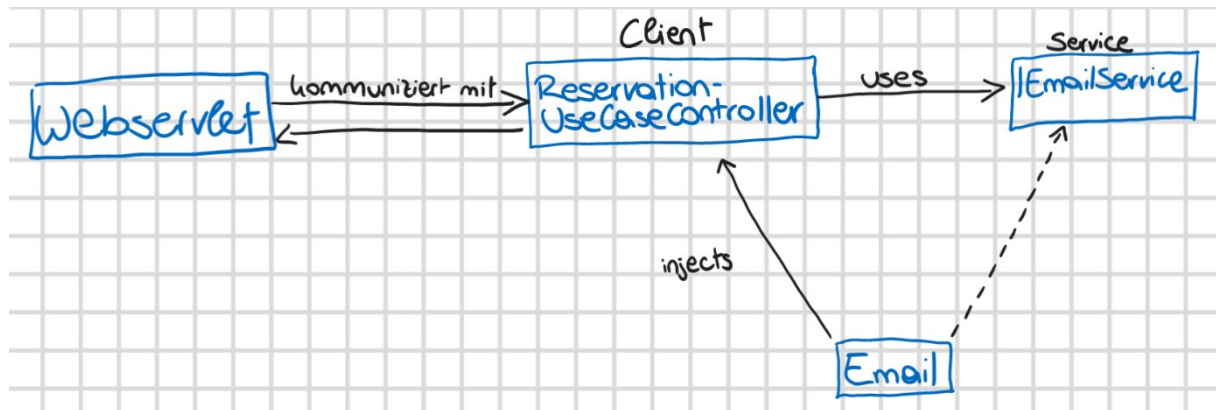
Der Check-In war nicht gefordert, wurde nicht geplant aber dann spontan noch umgesetzt.

3. Architektur



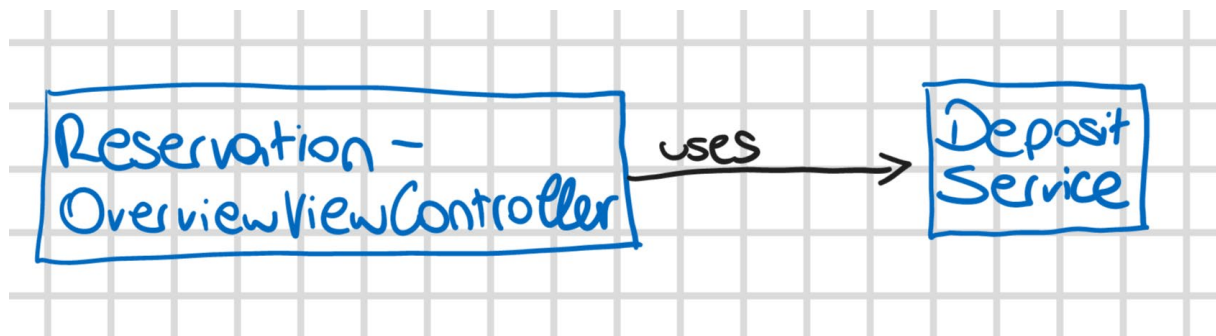
Der Email Mock wurde dann mit Hilfe von Dependency Injection eingebunden.

Dependency Injection beruht auf dem "Inversion of Control" Prinzip. Das heißt, dass in unserem Fall der ReservationUseCaseController nicht Abhängig ist von der Implementierung der Email Klasse, sondern vom IEmailService Interface. Die Email Klasse kann dann beliebig ausgetauscht werden. Z.B. könnte zum Versenden von Emails Klassen Gmail, GMXMail, Hotmail,... und alle implementieren das IEmailService interface und somit die `sendMail()` Methode. Das senden des Emails kann dann von Service zu Service unterschiedlich aussehen und der Client bekommt das nicht mit und muss nicht darüber entscheiden.



Der Deposit Service ist im package viewServices abgelegt.

Im ReservationOverviewViewController kann dann einfach eine Instanz erstellt werden und somit die funktion `parseData()` aufgerufen werden.



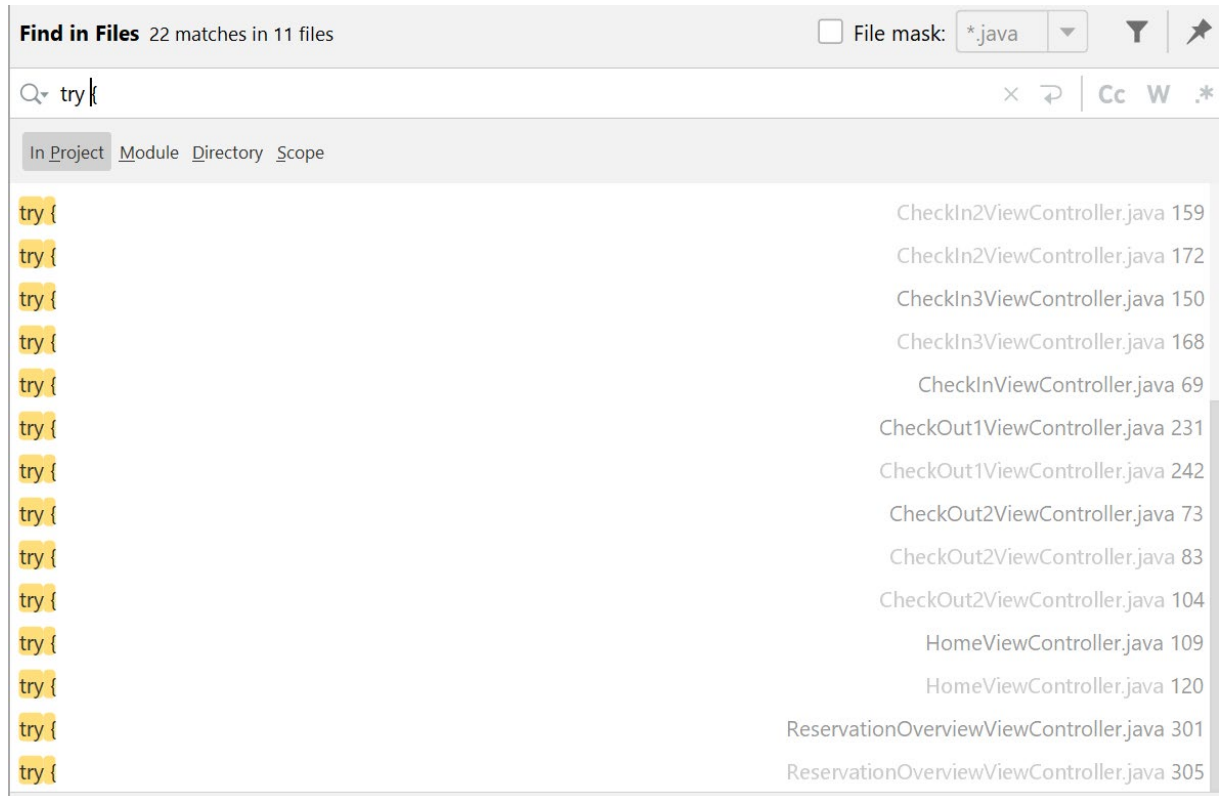
4. Verwendete Technologien

- Gradle *Version 7.5*,
- JavaFX *Version 19*,
- ControlsFX für CheckCombobox und SearchableCombobox *Version 11.1.2*,
- Hibernate *Version 5.6.12*,
- H2 Database *Version 2.1.214*,
- Log4j *Version 2.19.0*,
- Apache PDF Box *Version 2.0.27*,
- jakarta.servlet-api *Version 5.0.0*,
- jakarta.servlet.jsp.jstl *Version 2.0.0*,
- Tomcat *Version 10.0.27*

5. Fehlerbehandlung

Exceptions:

Exceptions werden in der View-Schicht abgefangen.



Alerts:

Der Design Experte Walter hat an unserem System bemängelt, dass der User zu wenig Rückmeldung bekommt. Nach dem Walk-In war dann die Frage woher der User weiß, dass der Walk-In erfolgreich war. Wir haben uns dann entschieden dies mit Alerts zu handhaben.

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


```

MainController.java
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;");

```

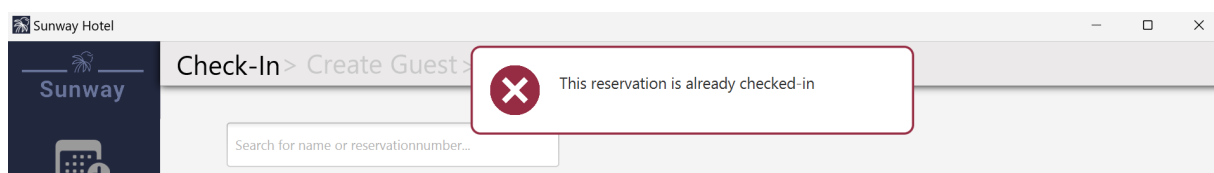
Kann dann so aufgerufen werden...

```

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);
        }
    }
}

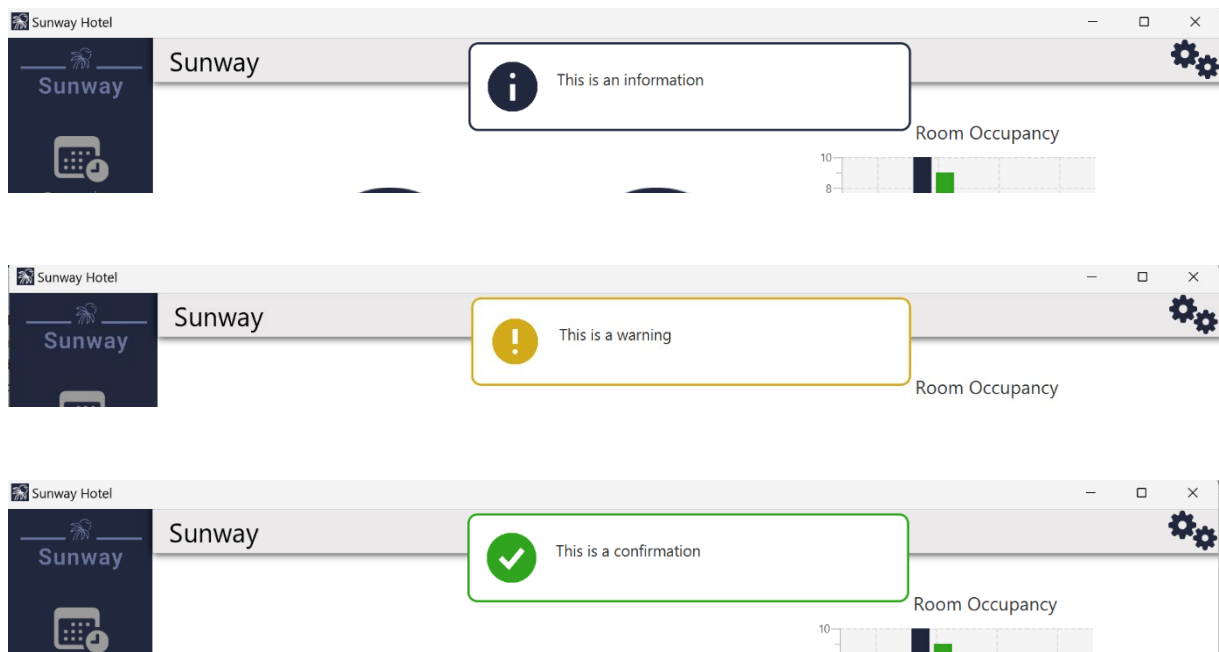
```

So können dem User schön und sinnvoll Rückmeldungen gegeben werden.



Team A

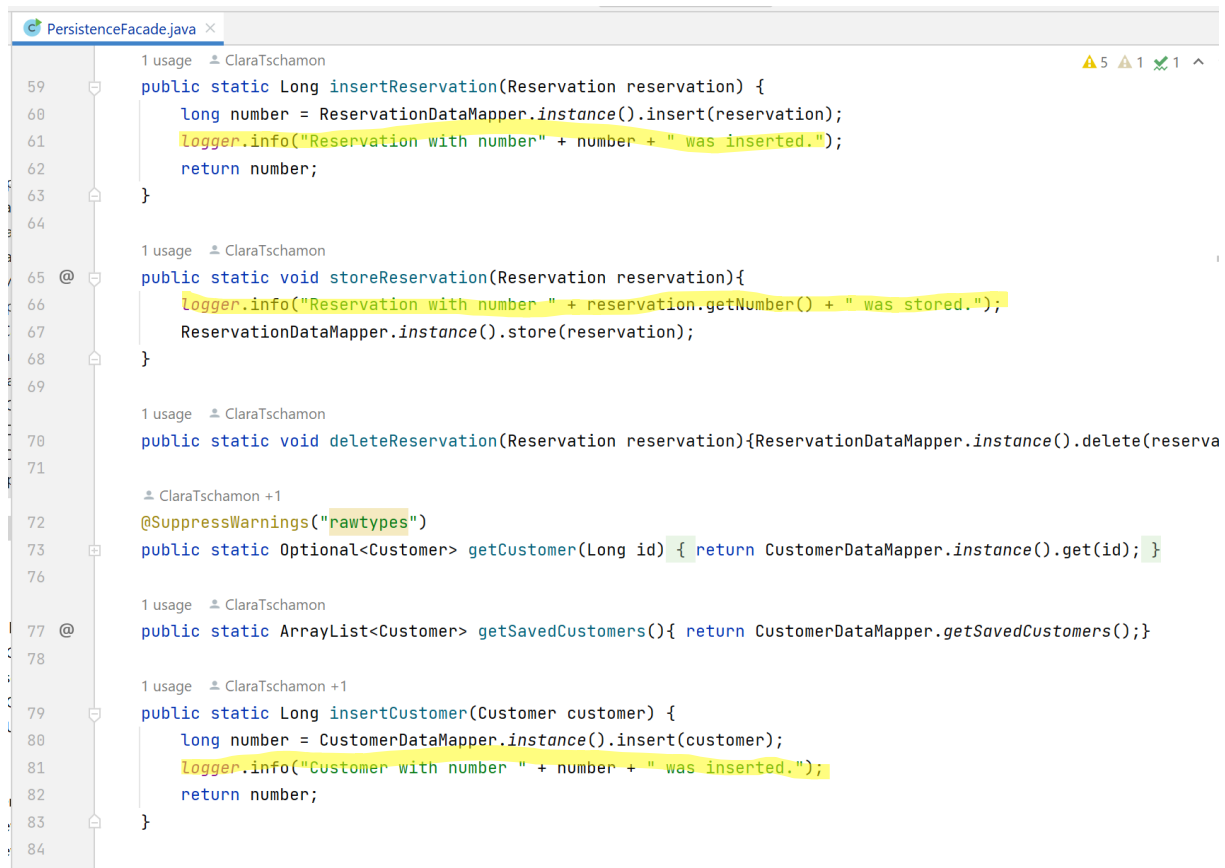
So sehen die anderen Alerts aus:



6. Logging

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

Log4j wurde auch zu Debugging Zwecken verwendet.

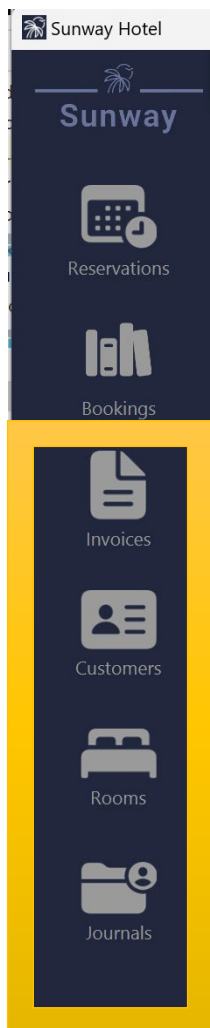


```
PersistenceFacade.java x
1 usage ClaraTschamon
59 public static Long insertReservation(Reservation reservation) {
60     long number = ReservationDataMapper.instance().insert(reservation);
61     logger.info("Reservation with number " + number + " was inserted.");
62     return number;
63 }
64
1 usage ClaraTschamon
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 ClaraTschamon
70 public static void deleteReservation(Reservation reservation){ReservationDataMapper.instance().delete(reserva
71
+ ClaraTschamon +1
72 @SuppressWarnings("rawtypes")
73 public static Optional<Customer> getCustomer(Long id) { return CustomerDataMapper.instance().get(id); }
76
1 usage ClaraTschamon
77 @ public static ArrayList<Customer> getSavedCustomers(){ return CustomerDataMapper.getSavedCustomers();}
78
1 usage ClaraTschamon +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
```

7. 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

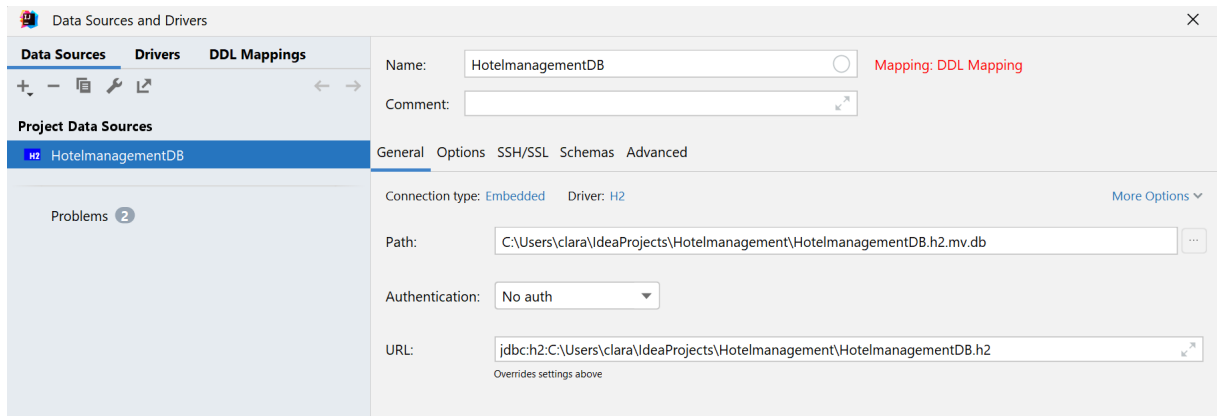
- Umgang mit den oben genannten Technologien,
- Architekturmuster,
- Teamwork und Scrum

Nächstes mal würden wir folgendes anders machen:

- Der Domain Controller sollte eine Facade sein, ist es aber nicht wirklich. Man hätte es so machen müssen, wie mit der Persistence Facade und nicht einfach alle Methoden static machen.
- Architektur folgt bei uns der Layered Architecture aber wir haben auch Dependency Injection verwendet. Nächstes Mal würden wir gleich die ganze Architektur als Hexagonale Architektur machen.
- Factories, welche die DTOs erstellen, folgen nicht dem Factory Pattern. Sollten sie aber.

8. 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\Hotelmanagement\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));
```

9. 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