Hotel Reservation System
Architecture Notebook

Revision History

Date	Author	Description	Version
03/05/2021	Irina Erofeeva Ömer Denizoğlu Durali Alagöz M.Mert Dervişoğulları	First version	v 1.0
09/05/2021	Irina Erofeeva Ömer Denizoğlu Durali Alagöz M.Mert Dervişoğulları	Updated according to Iteration2 reviews.	v 1.1
30/05/2021	Irina Erofeeva Ömer Denizoğlu Durali Alagöz M.Mert Dervişoğulları	Updated Use Case model and architecturally significant requirements section.	v 1.2
20/06/2021	Irina Erofeeva Ömer Denizoğlu Durali Alagöz M.Mert Dervişoğulları	Small grammar fixes	v 1.3

Table of Contents

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation. This document is aiming to guide software developers already working on this project and newcomers who will work for this project.

2. Architectural goals and philosophy

1. The unified process will be followed during the project to follow the architecture-centric method.
2. The architecture document must guide to simplicity, high performance and maintainability.
3. All architecturally significant requirements must be considered while developing the architecture notebook.
4. Reusing the existing components is a mandatory goal of architecture.
5. System should be extensible without modifying or rewriting existing components.

3. Assumptions and dependencies

1. Object Oriented Design and Analysis must be followed.
2. Java programming language will be used as an object-oriented language.
3. The system will run on any machine that has Java Virtual Machine.
4. The system will work with SQLite but it can also work with other Databases.

4. Architecturally significant requirements

1. SWRS section 4. Hotel guests must interact with the system through kiosks in the lobby, touch screens in their rooms -> System should have a Graphical User Interface.
2. SWRS section 2. 1. In the later stage of the development the system must be adapted to mobile devices for the remote control.
3. SWRS section 4.2.1. 1. System must provide an interface in order to provide an access to the system for other services of the hotel.
4. SWRS section 4.2.1. 2. HRS should use different external systems for payment.
5. Use Case Specification Use Case1- Manage Accounts, Use Case2- Manage Reservation of Room, Use Case4- Manage Reservation of Other services, Use Case5- Make Payment in cash, Use Case7- Monitor Reservations are significant requirements.

5. Decisions, constraints, and justifications

1. There must be a low representational gap between Domain model and Design Model. **Justification:** this should be done in order to achieve FURPS+ principles.
2. Using magic numbers is not allowed. **Justification:** this constraint is introduced in order to improve maintainability of the software by improving understandability of the code.

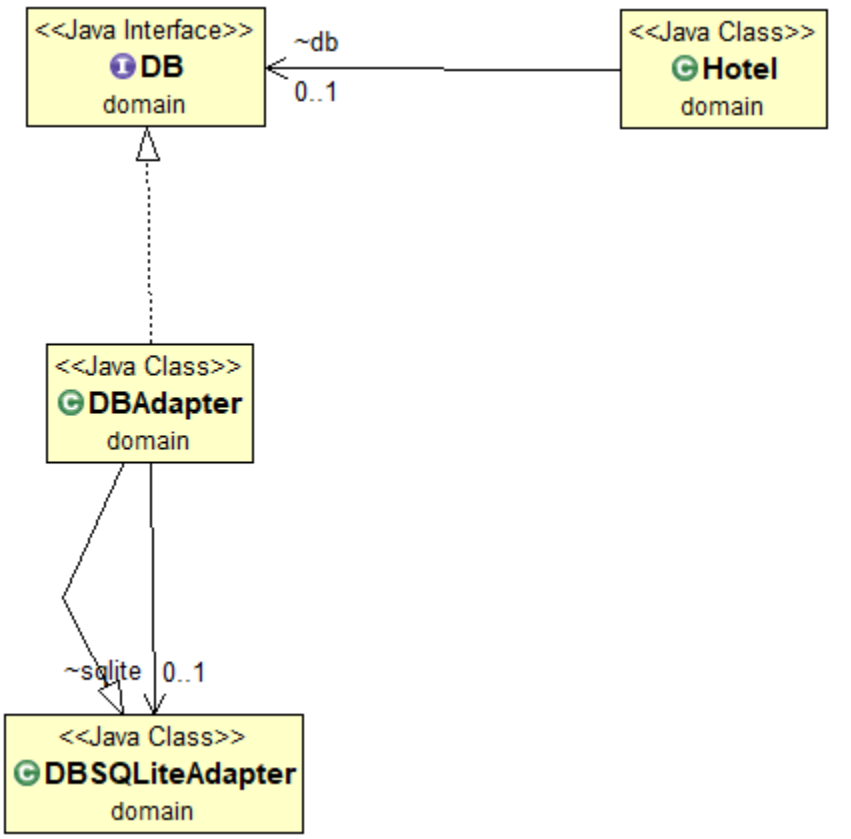
6. Architectural Mechanisms

6.1 Architectural Mechanism 1

Technical Memo

Issue: The significant external parts of the system can be changed. (Payment, Database…)
Solution Summary: If the external parts of the system are changed, the system must continue to work properly.
Factors
Robust recovery from remote database failure
Robust recovery from payment system failure

Solution
The system can continue to work with different databases or payment systems using adapter design patterns. There is a DBAdapter class that extends DBSQLiteAdapter in this implementation and implements an interface. If there is a need to connect another DB system, the new Adaptee can be switched to the new Adaptee class.



Motivation
The hotel can work with different payment systems as time goes.
The database used in the system may be insufficient as time goes.

6.2 Architectural Mechanism 2

Issue: the graphical interface of the system may be changed according to the customers' vision, however, if the changes will be introduced, they should not affect the behavior of the system
Solution Summary: model view separation.
Solution: There should not be any logical operation in the UI layer. All methods must be placed in the domain package and can only be called by UI layer.

6.3 Architectural Mechanism 3

Issue: In the course of the development, especially in early stages, changes are often. However, those changes may introduce errors, if there are complex dependencies in the code.
Solution Summary: single responsibility principle.
Solution: First solution to this problem is - Each class must have only one responsibility. So that the impact from changes in one class will be reduced.

6.4 Architectural Mechanism 4

Issue: In the course of the development, especially in early stages, changes are often. However, those changes may introduce errors, if there are complex dependencies in the code.
Solution Summary: low coupling, information hiding.
Solution: The second solution to this problem is - Using interfaces. Interfaces allow the use of methods of the classes without knowing their internal details, which makes the class using those methods to be independent from those internal details.

7. Key abstractions

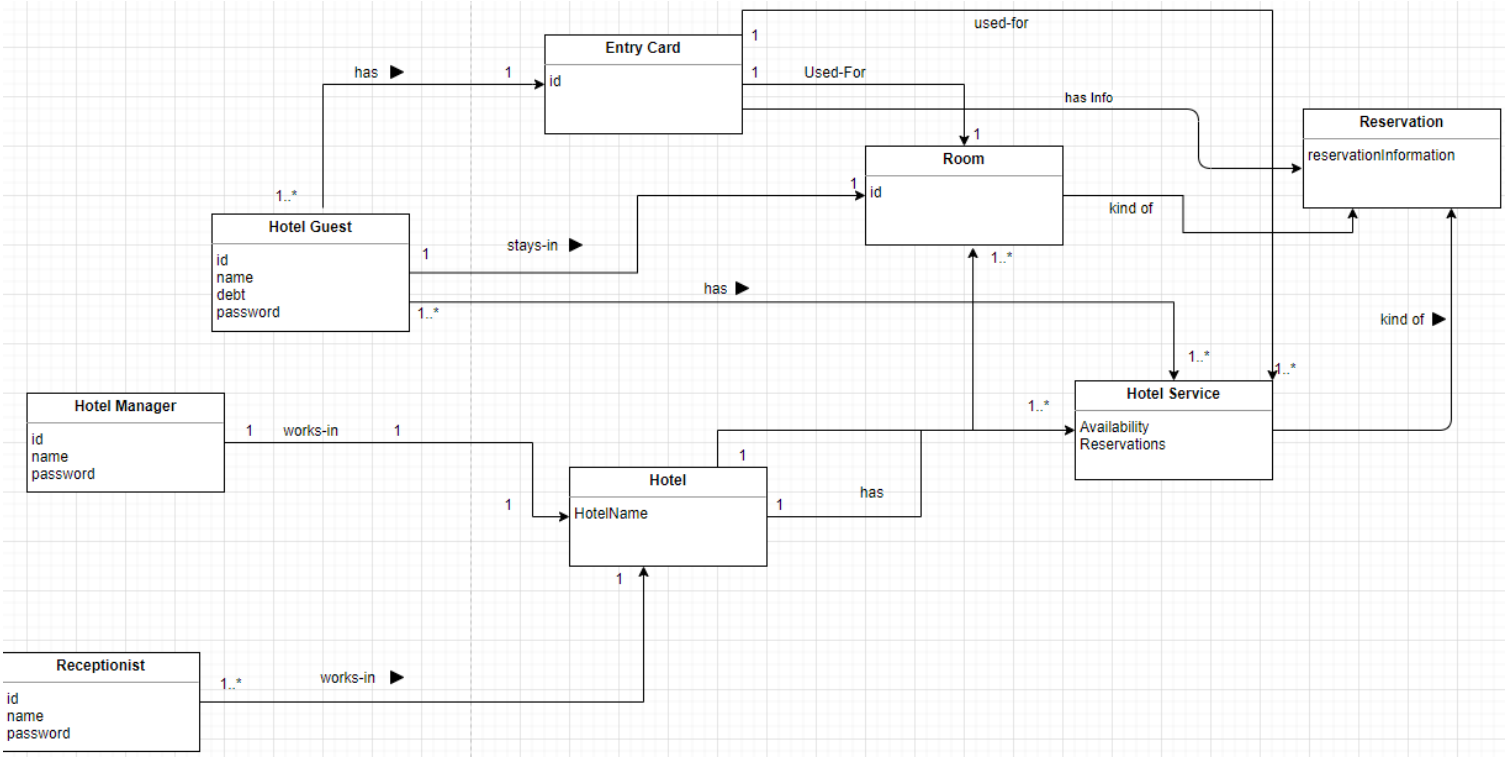


Figure 1 Domain Model

8. Layers or architectural framework

Layered Architecture will be used to achieve separation of concerns principle. By layered architecture, we created a logical encapsulation of each layer like all the code, with respect to UI stays in the view layer, all code regarding business logic stays in the domain layer. Each layer communicates with its adjacent layer, but never communicates with another layer, which is not adjacent.

Two rules for this architecture been implemented:

- All the dependencies go in one direction, from presentation to infrastructure (the infrastructure layer saves domain objects directly, so it actually knows about the classes in the domain)
- No logic related to one layer's concern should be placed in another layer. For instance, no domain logic or database queries should be done in the UI.

1. UI Layer

The user will interact with this layer. There won't be any logic operation in this layer. These layer elements will be in the HRS.UI namespace.

2. Application Logic Layer

All the logic operations about the requirement will be handled in this layer. These layer elements will be in the HRS.APPLOGIC namespace.

3. Technical Services Layer

All the technical services will be handled in this layer. These layer elements will be in the HRS.SERVICES namespace.

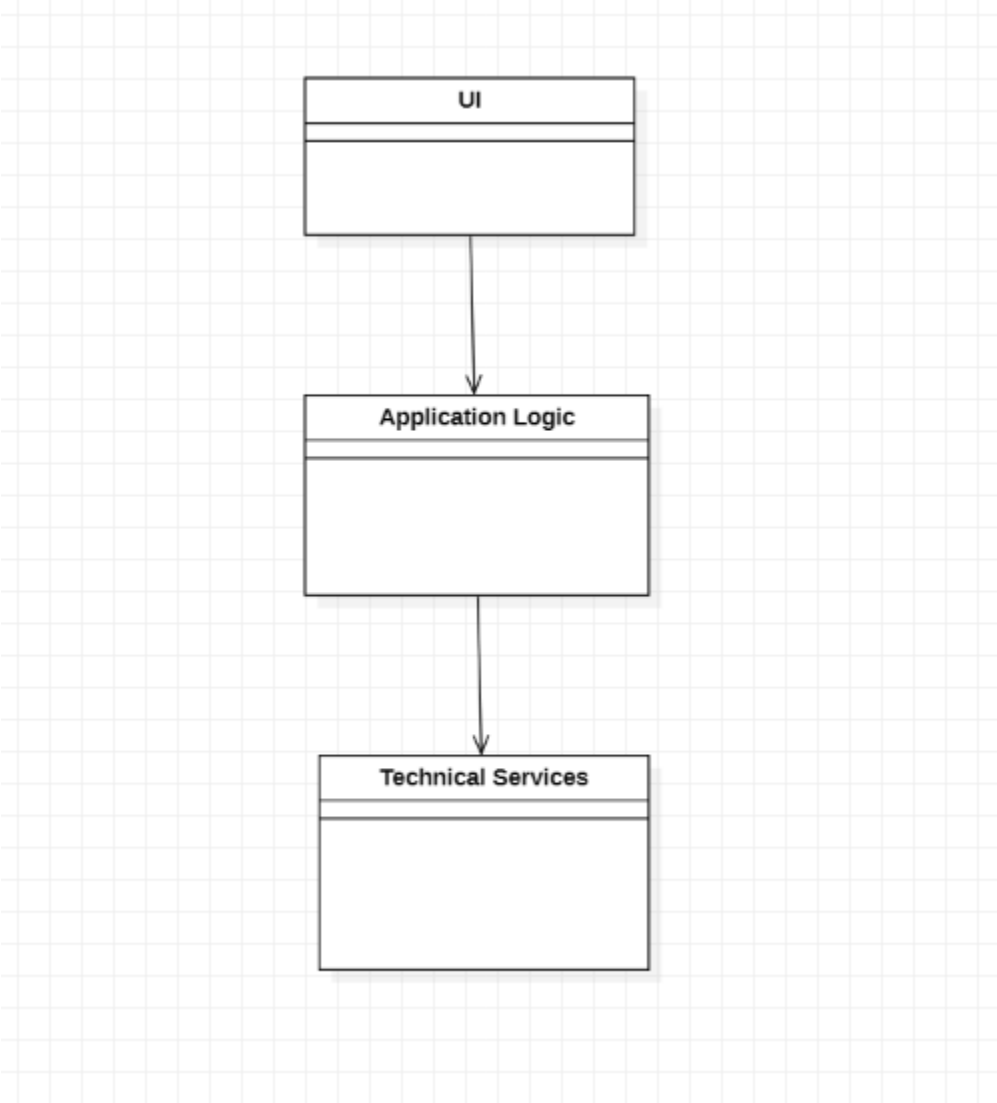


Figure 2 Layers

9. Architectural views

9.1 Logical:

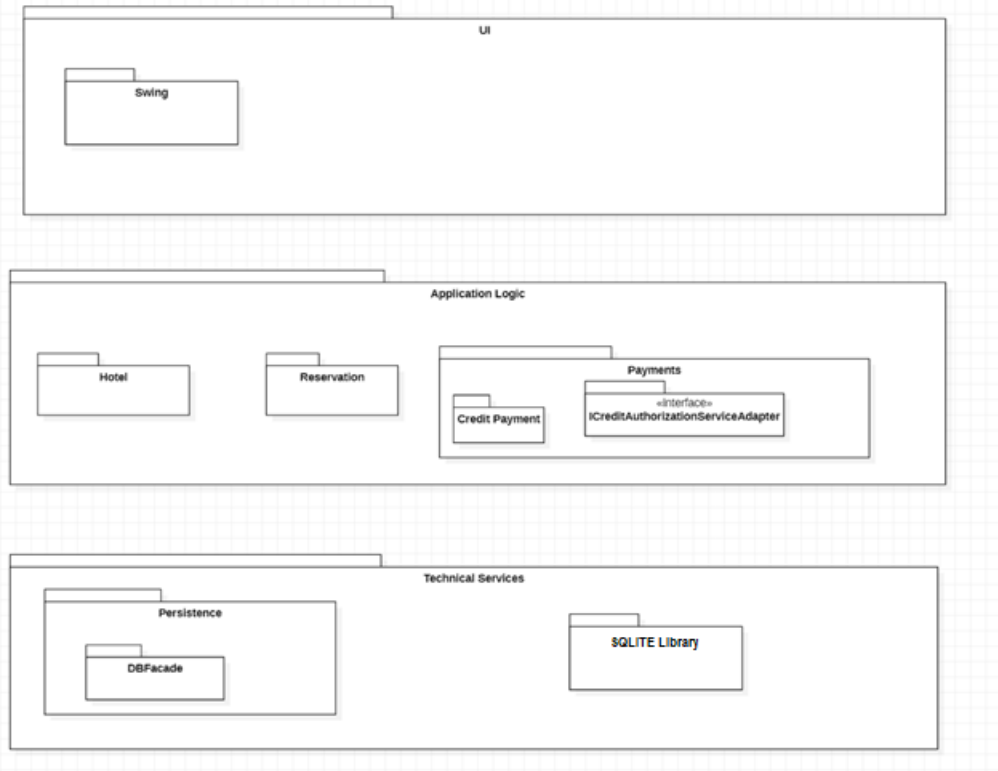


Figure 3 Logical View

9.2 Deployment:

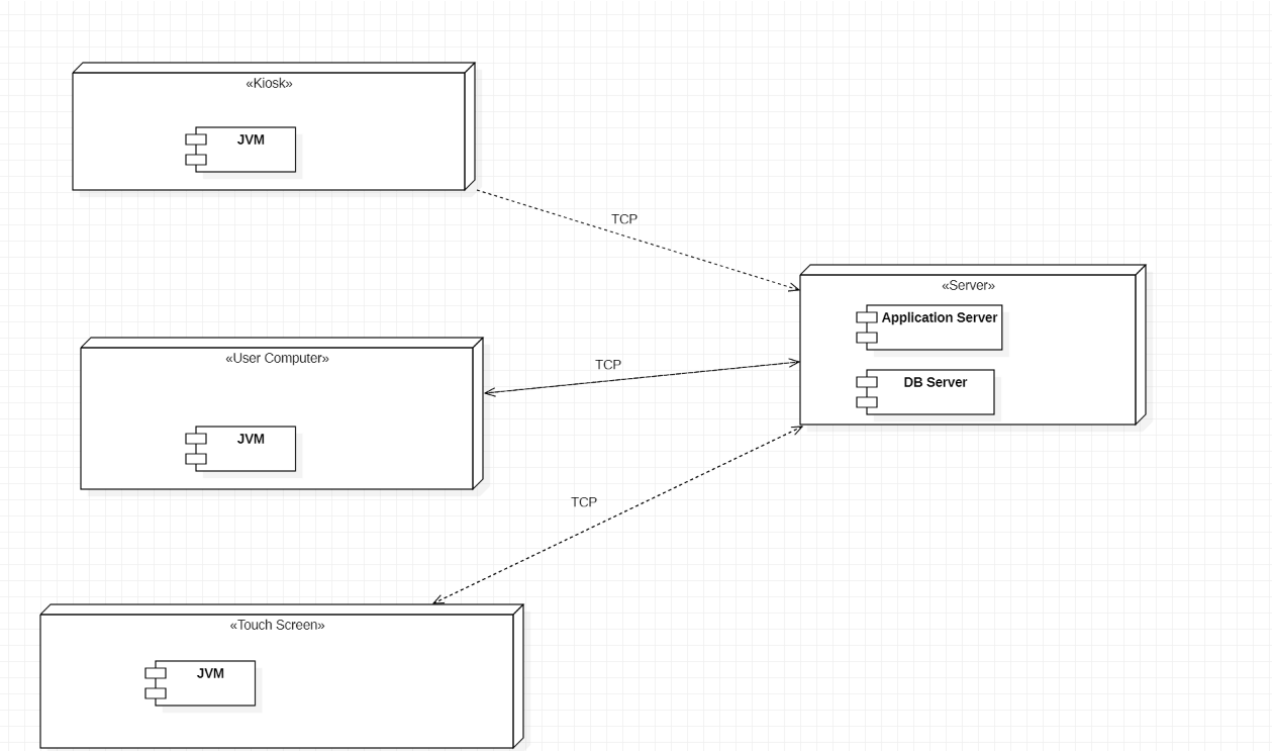


Figure 4 Deployment View

9.3 Data

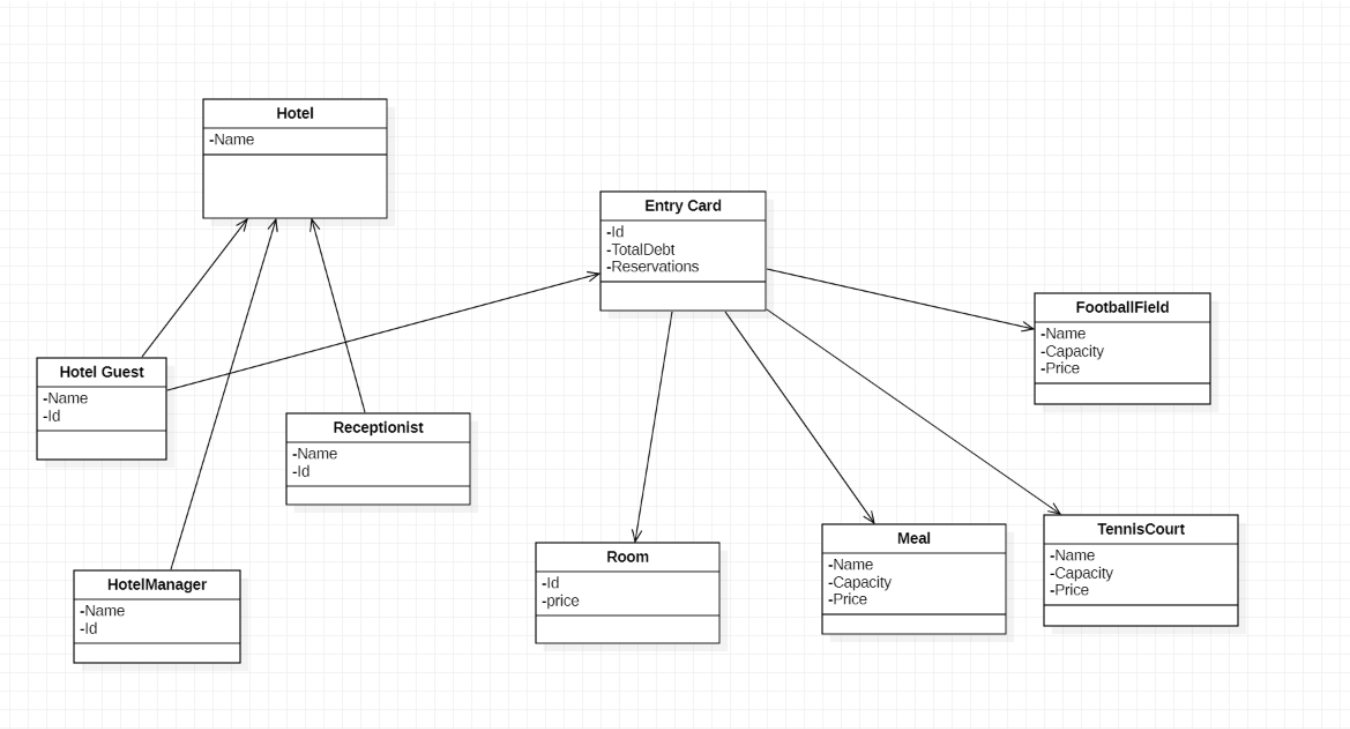


Figure 5 Data View

9.4 Use Case

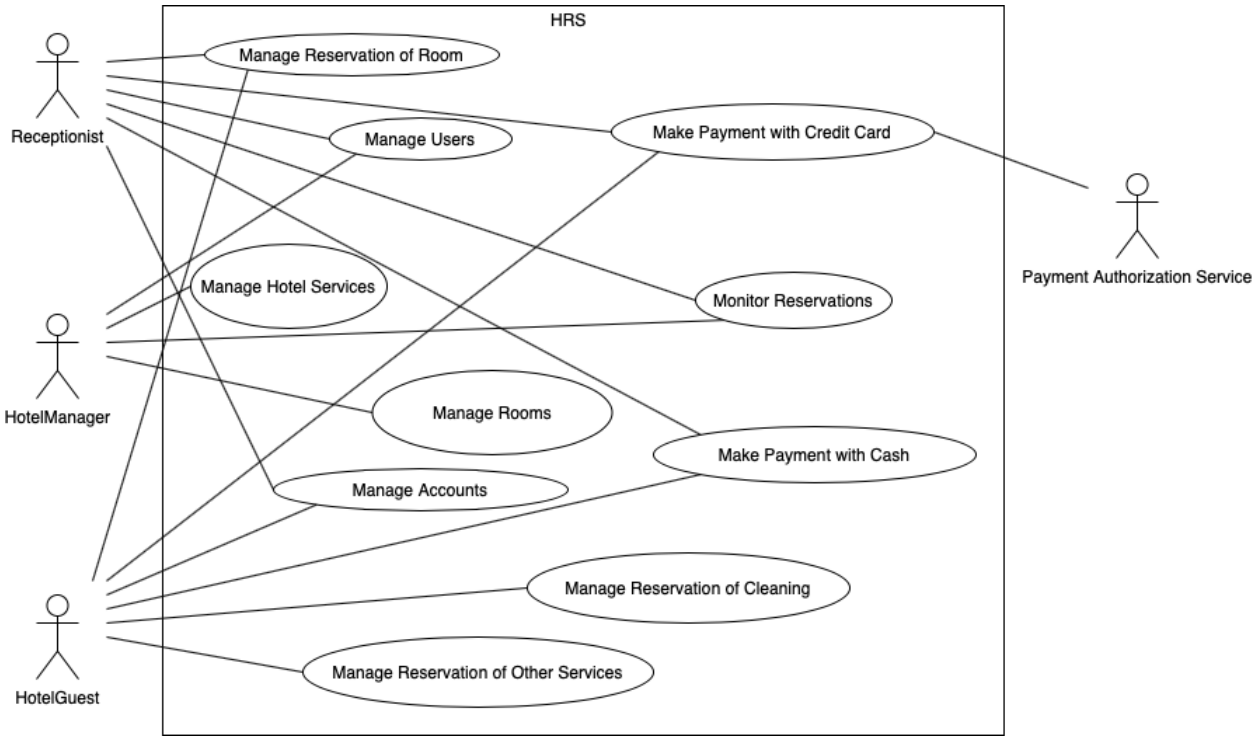


Figure 6 Use Case View