California State University Fullerton

CPSC 462



Object Oriented Software Design

SW Architecture Document (SAD)

for the



Hotel Reservation

System

|  |  |  |
| --- | --- | --- |
|  | **Josh Ibad** |  |
|  | [Chief Software Architect](mailto:joshcibad@csu.fullerton.edu?subject=Regarding the HotelBuch System) |  |
|  | [joshcibad@csu.fullerton.edu](mailto:joshcibad@csu.fullerton.edu?subject=Regarding the HotelBuch System) |  |

Revision History:

| Version | Date | Summary of Changes | Author |
| --- | --- | --- | --- |
| 1.0 | 2021-11-15 | * Initial Release | Josh Ibad |

**Table of Contents**

[1 Architectural Representation 2](#__RefHeading___Toc801_3221632959)

[2 Architectural Decisions 3](#__RefHeading___Toc803_3221632959)

[2.1 Controller GRASP Decision 3](#__RefHeading___Toc805_3221632959)

[2.1.1 Decision to be made 3](#__RefHeading___Toc807_3221632959)

[2.1.2 Options Considered 3](#__RefHeading___Toc809_3221632959)

[2.1.3 Selection and Rationale 4](#__RefHeading___Toc811_3221632959)

[3 Logical View 5](#__RefHeading___Toc813_3221632959)

[3.1 Package Diagrams 5](#__RefHeading___Toc815_3221632959)

[3.1.1 Presentation (UI) Layer Components 5](#__RefHeading___Toc817_3221632959)

[3.1.2 Domain (Application) Layer Components 5](#__RefHeading___Toc819_3221632959)

[3.1.2.1 *Hotel* 5](#__RefHeading___Toc821_3221632959)

[3.1.2.2 *Reservation* 5](#__RefHeading___Toc823_3221632959)

[3.1.2.3 *Session* 6](#__RefHeading___Toc825_3221632959)

[3.1.3 Technical Services Layer Components 6](#__RefHeading___Toc827_3221632959)

[3.1.3.1 *Persistence* 6](#__RefHeading___Toc829_3221632959)

[3.1.3.2 *Logging* 6](#__RefHeading___Toc831_3221632959)

[3.2 Interface Diagrams 6](#__RefHeading___Toc833_3221632959)

[3.2.1 Presentation (UI) Layer Interface Diagrams 6](#__RefHeading___Toc835_3221632959)

[3.2.2 Domain Layer Interface Diagrams 6](#__RefHeading___Toc837_3221632959)

[3.2.3 Technical Services Interface Diagrams 7](#__RefHeading___Toc839_3221632959)

# Architectural Representation

…

<Summarize key architectural decision in format call technical memoa short one-page description of a descision and its motivation>

# Architectural Decisions

## Controller GRASP Decision

### Decision to be made

Who should be responsible for handling an input system event whenever a user attempts to manage hotel rooms?

A good decision should provide good abstraction and minimize coupling by making sure that system level events are only concerened with system level details and have implementation details hidden below.

A poor decision would have too many Classes or interfaces that must be interfaced with for a simple procedure, and would be unscalable as classes and interfaces are added when the system grows.

### Options Considered

| Controller | Static View | Dynamic View |
| --- | --- | --- |
| Option 1: Facade (System) Controller  (Rejected) |  |  |
| Option 2:  Session (Use Case) Controller  (Selected) |  |  |
| Design Model Reference | In document 09 - Design Model, at section '1 Static View', under subsection '1.1 Hotel Reservation system - Room and Reservation Management'. (Page 2). | In document 09 - Design Model, at section '2 Dynamic View', under subsection '2.2 addHotelRooms Sequence of Execution' (Page 4), and subsection '2.6 makePayment Sequence of Exeuction' (Page 8). |

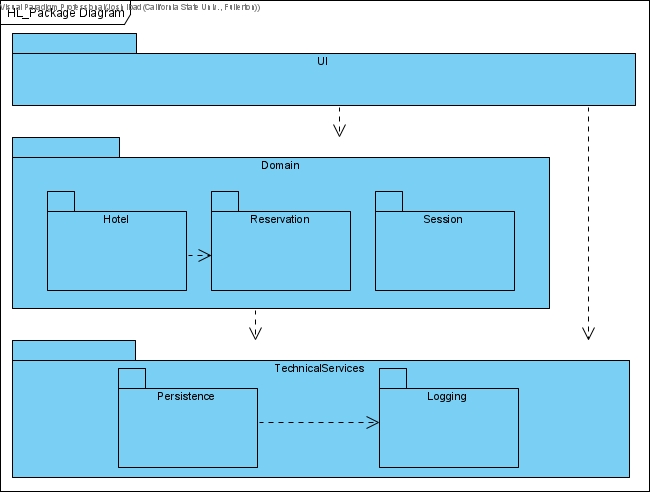
### Selection and Rationale

Option 1 has been discarded because, although it centralizes all interaction into a single system-wide Facade controller, it also means that the singular controller will be highly coupled with all other Domain-layer components and classes. In the long-run this means higher coupling and it also means lack of scalability as more functions and classes are added in the Domain-layer components.

Option 2 has been selected because it elegantly permits the subdivision of the Domain-layer into a subset of low-coupling components or sub-packages that scales well when more functionalities and classes are added. By making each system level message a single interface method relayed to a single component, rather than use case messages that then send system-level messages scattered all over the domain layer, we have have a better design with the Session controllers (use-case level controllers with handlers for each component).

# Logical View

## Package Diagrams



### Presentation (UI) Layer Components

N/A

### Domain (Application) Layer Components

#### *Hotel*

The *Hotel* domain component is the component responsible for the Hotel concept, as a container of Rooms. The primary corresponding use case that the *Hotel* component is responsible for is the *Manage Hotel Rooms* use case.The *Hotel* component holds in it, classes such as Hotel and Room as well as the interface HotelHandler. The *Hotel* component's responsibility is to keep track of a Hotel and it's Rooms, along with the description of said Rooms. These Rooms have a price, descriptions, bed count, along with a BedType and RoomType. Any functionality, class, enumeration, or interface to support the *Manage Hotel Rooms* use case are packaged into the *Hotel* component.

#### *Reservation*

The *Reservation* domain component is the component responsible for the concept of Reservations and payment related concepts such as BillingMethods, Transactions, and Accounts. The primary corresponding use case that the *Reservation* component is responsible for is the *Manage Reservations* use case. The *Reservation* holds in it, classes such as Reservation, BillingMethod, Transaction, Account as well as the interface ReservationHandler. The *Reservation* component's responsibility is to keep track of reservations and payments for these Reservations, along with the times of reservation, time of payment, billing information such as card number, expiration, cvv codes, card holder name, address and contact, card types, payment dates, etc. Any functionality, class, enumeration or interface to support the *Manage Reservation* use case that deals explicitly with the Reservation rather than the Room, is packaged into the *Reservation* component.

#### *Session*

The *Session* domain component is the component responsible for serving as an actor's user Session in the system. The *Session* component facilitates the authentication functionality, which is not an explicit use case mentioned earlier, but is an essential supporting functionality that allows all other use cases to occur. The *Session* component takes care of user roles and permissions, making sure that they are both able to reach the functionalities that they are intended to have, and ONLY the functionalities and interfaces that they are intended to have. The Session draws greatly from the TechnicalServices layer's *Persistence* component to store user credentials and roles.

### Technical Services Layer Components

#### *Persistence*

The *Persistence* technical services component is the component responsible for managing persistent data, that is, data that persists throughout system usage and system shutdowns. Ideally, a *Persistence* component would perform all CRUD operations to store and manage information, but temporarily, it solely performs Reads from regular text files in order to read the configuration of the UI and Logging.

#### *Logging*

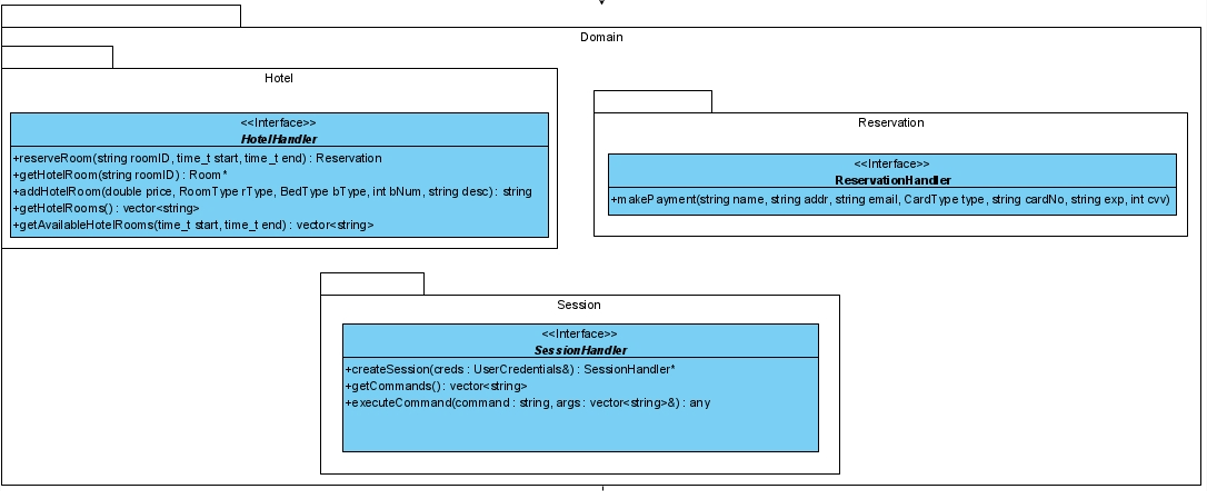
The *Logging* technical services component is the component responsible for logging system events and interactions. It takes care of printing messages into the command line in a well formatted manner, printing out the date and time of an event, followed by messages from the corresponding event. Logging provides insight to the functionality of technical services components such as the Logging and Persistence along with the UI layer. It also has the potential use of logging system events occuring the Domain layer if found necessary. With the first iterations of the Hotel Reservation system being a console application, the *Logging* component is essential for giving the user information on the system.

## Interface Diagrams

### Presentation (UI) Layer Interface Diagrams

N/A

### Domain Layer Interface Diagrams



### Technical Services Interface Diagrams

