FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Hotel System

Realization

Project documentation for the purpose of BIE-SWI course.

Authors:

Table of Contents

1. Realization Model

In this chapter, the realization of some of the use cases of the Library Information System is described, showing the communication of classes of the Wep application.

1.1 Create room

This section describes the realization of the Use Case to create a new room in the system. This action is typically restricted to administrative users.

1.1.1 Class model

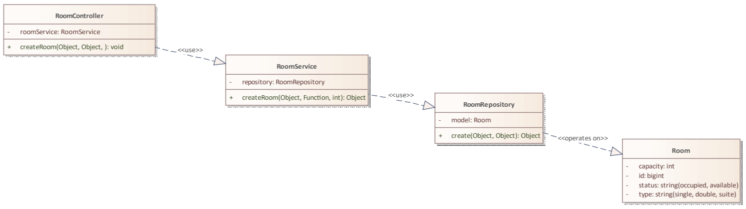
RoomController
- roomService: RoomService
+ createRoom(Object, Object,): void
<<use>>
RoomService
- repository: RoomRepository
+ createRoom(Object, Function, int): Object
<<use>>
RoomRepository
- model: Room
+ create(Object, Object): Object
<<operates on>>
Room
- capacity: int
- id: bigint
- status: string(occupied, available)
- type: string(single, double, suite)

Figure 1 - Class model

1.1.2 Communication model

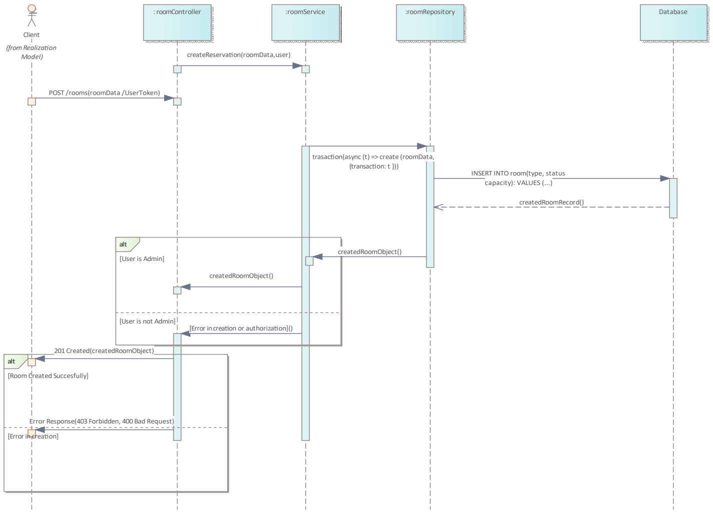
Client
(from Realization Model)
: roomController
:roomService
:roomRepository
Database
createReservation(roomData,user)
POST /rooms(roomData /UserToken)
trasaction(async (t) => create (roomData, {transaction: t }))
INSERT INTO room(type, status capacity): VALUES (...)
createdRoomRecord()
alt
[User is Admin]
createdRoomObject()
createdRoomObject()
[User is not Admin]
[Error in creation or authorization]()
201 Created(createdRoomObject)
alt
[Room Created Succesfully]
Error Response(403 Forbidden, 400 Bad Request)
[Error in creation]

Figure 2 - Communication model

This diagram illustrates an administrator creating a new room.

- The Client (Admin) sends a POST request with room details to the RoomController.
- The RoomController passes the request to RoomService after implicit authentication/authorization.
- RoomService verifies admin privileges. If authorized, it instructs RoomRepository to create the room.
- RoomRepository executes an INSERT command in the Database.
- The Database confirms creation, and the new room data is returned through RoomRepository and RoomService to RoomController.
- RoomController sends a 201 Created response with the new room to the Client. Unauthorized attempts result in an error.

1.2 Get bills

This section describes the realization of the Use Case to retrieve a list of all bills from the system. This represents the default scenario without specific filtering criteria.

1.2.1 Class model

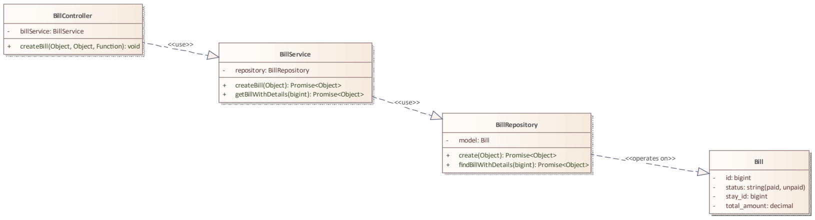
BillController
- billService: BillService
+ createBill(Object, Object, Function): void
<<use>>
BillService
- repository: BillRepository
+ createBill(Object): Promise<Object>
+ getBillWithDetails(bigint): Promise<Object>
<<use>>
BillRepository
- model: Bill
+ create(Object): Promise<Object>
+ findBillWithDetails(bigint): Promise<Object>
<<operates on>>
Bill
- id: bigint
- status: string(paid, unpaid)
- stay_id: bigint
- total_amount: decimal

Figure 3 - Class model

1.2.2 Communication Model

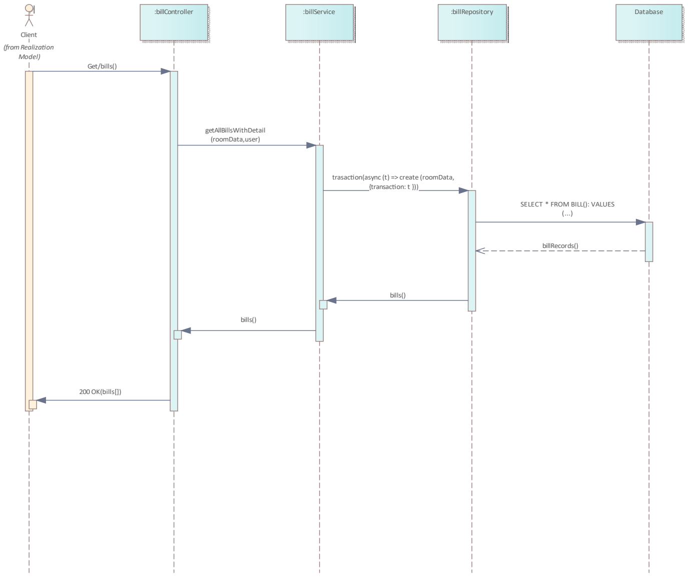

Figure 4 - Communication model

This diagram shows how a list of all bills is retrieved.

- The Client sends a GET request to the BillController to fetch all bills.
- BillController calls getAllBillsWithDetails on BillService.
- BillService requests findBillsWithDetails from BillRepository.
- BillRepository queries the Database to select all bill records, including any necessary details.

- The Database returns the bill records, which are then passed back through BillRepository and BillService to BillController.
- BillController sends a 200 OK response with the list of bills to the Client.

1.3 Reservations

1.3.1 Class model

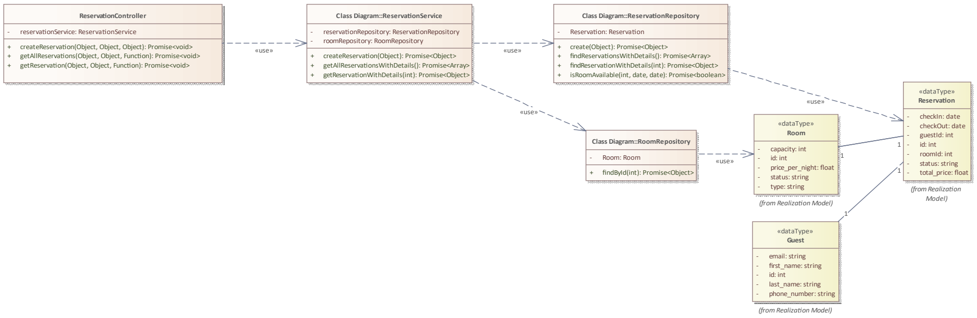
ReservationController
- reservationService: ReservationService
+ createReservation(Object, Object, Object): Promise<void>
+ getAllReservations(Object, Object, Function): Promise<void>
+ getReservation(Object, Object, Function): Promise<void>
«use»
Class Diagram::ReservationService
- reservationRepository: ReservationRepository
- roomRepository: RoomRepository
+ createReservation(Object): Promise<Object>
+ getAllReservationsWithDetails(): Promise<Array>
+ getReservationWithDetails(int): Promise<Object>
«use»
Class Diagram::ReservationRepository
- Reservation: Reservation
+ create(Object): Promise<Object>
+ findReservationsWithDetails(): Promise<Array>
+ findReservationWithDetails(int): Promise<Object>
+ isRoomAvailable(int, date, date): Promise<boolean>
«use»
«use»
Class Diagram::RoomRepository
- Room: Room
+ findById(int): Promise<Object>
«use»
«dataType»
Room
- capacity: int
- id: int
- price_per_night: float
- status: string
- type: string
(from Realization Model)
«dataType»
Reservation
- checkIn: date
- checkOut: date
- guestId: int
- id: int
- roomId: int
- status: string
- total_price: float
(from Realization Model)
«dataType»
Guest
- email: string
- first_name: string
- id: int
- last_name: string
- phone_number: string
(from Realization Model)
1
1
1
1

Figure 5 - Class Diagram

1.3.2 Create reservation

The client sends an HTTP POST request to /reservations with reservationData (e.g., room_id, guest_id, check_in_date, check_out_date).

The ReservationController passes the request body to ReservationService.createReservation().

The ReservationService:

Validates required fields and dates.

Checks room availability by calling ReservationRepository.isRoomAvailable(roomId, checkIn, checkOut), which queries the database for conflicting confirmed reservations.

If available, retrieves room details via RoomRepository.findById(roomId) to get price_per_night.

Calculates the total price based on the number of nights.

Creates the reservation with ReservationRepository.create(), including the calculated total_price.

Fetches the full reservation details with findReservationWithDetails(id).

The created reservation is returned through the layers, and the controller sends a 201 JSON response.

Errors (e.g., unavailable room, invalid dates) result in a 400 response.

1.3.2.1 Communication model

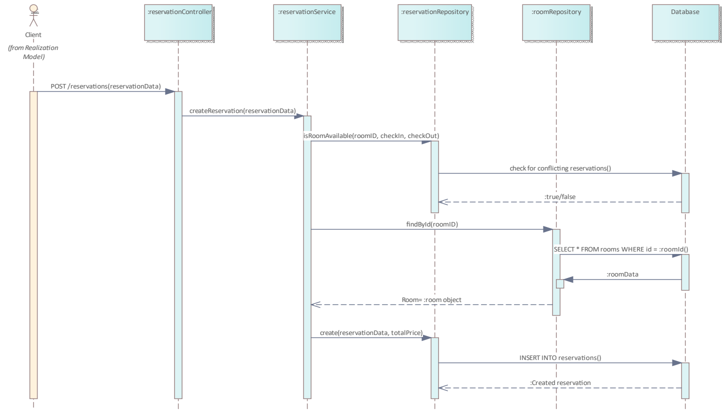
Client
(from Realization Model)
:reservationController
:reservationService
:reservationRepository
:roomRepository
Database
POST /reservations(reservationData)
createReservation(reservationData)
isRoomAvailable(roomID, checkIn, checkOut)
check for conflicting reservations()
:true/false
findById(roomID)
SELECT * FROM rooms WHERE id = :roomId()
:roomData
Room= :room object
create(reservationData, totalPrice)
INSERT INTO reservations()
:Created reservation

Figure 6 - Communication model

1.3.3 List of reservations

The client sends an HTTP GET request to the /reservations endpoint.
The ReservationController invokes getAllReservationsWithDetails() on the ReservationService.
The ReservationService calls findReservationsWithDetails() on the ReservationRepository.
The ReservationRepository queries the database to fetch all reservations, including related data (e.g., Room and Guest details).
The data flows back through the layers, and the controller returns a JSON response with the list of reservations.

1.3.3.1 Communication model

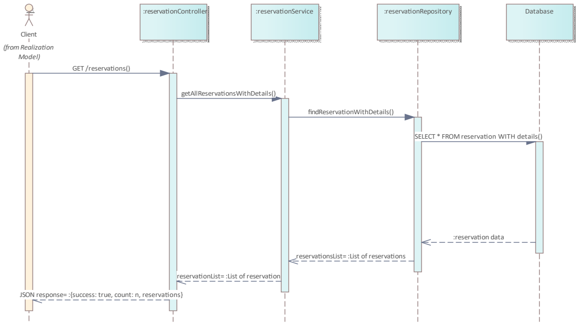
Client
(from Realization Model)
:reservationController
:reservationService
:reservationRepository
Database
GET /reservations()
getAllReservationsWithDetails()
findReservationWithDetails()
SELECT * FROM reservation WITH details()
:reservation data
reservationsList= :List of reservations
reservationList= :List of reservation
JSON response= :{success: true, count: n, reservations}

Figure 7 - Communication model

1.3.4 Reservation Detail

The client sends an HTTP GET request to /reservations/:id with the reservation ID.
The ReservationController calls getReservationWithDetails(id) on the ReservationService, passing the ID from the request parameters.
The ReservationService invokes findReservationWithDetails(id) on the ReservationRepository.
The ReservationRepository queries the database for the specific reservation, including associated details.
The data is returned through the layers, and the controller sends a JSON response with the reservation details.
If the reservation is not found, an error is thrown, and a 404 response is returned.

1.3.4.1 Communication model

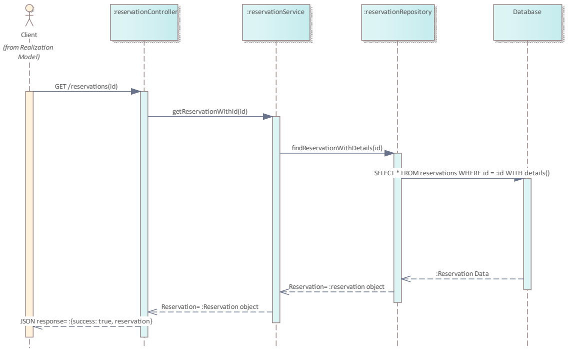
Client
(from Realization Model)
:reservationController
:reservationService
:reservationRepository
Database
GET /reservations(id)
getReservationWithId(id)
findReservationWithDetails(id)
SELECT * FROM reservations WHERE id = :id WITH details()
:Reservation Data
Reservation= :reservation object
Reservation= :Reservation object
JSON response= :{success: true, reservation}

Figure 8 - Communication model