Status

Current implementation status: Complete

No roadblocks

Screenshots

We have all of the use cases implemented, but these are the ones relevant specifically for this deliverable, along with screenshots. Some screenshots cover multiple use cases.

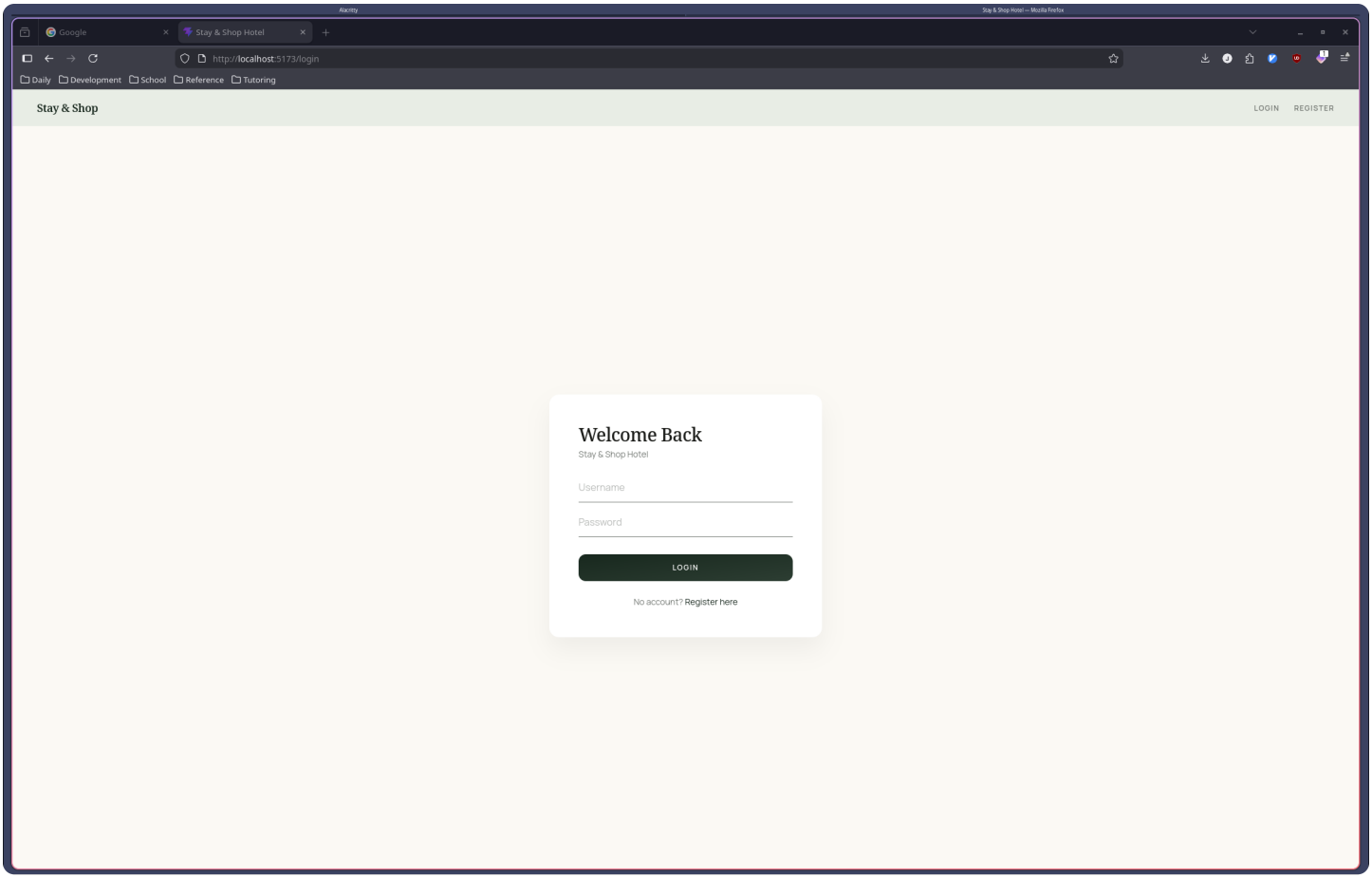
Use Cases: Authentication

Use Case: Hotel Clerk Login

- Author: Jonathan Deiss
- Connected to Backend: YES
- Implemented By: Team effort prompt engineering

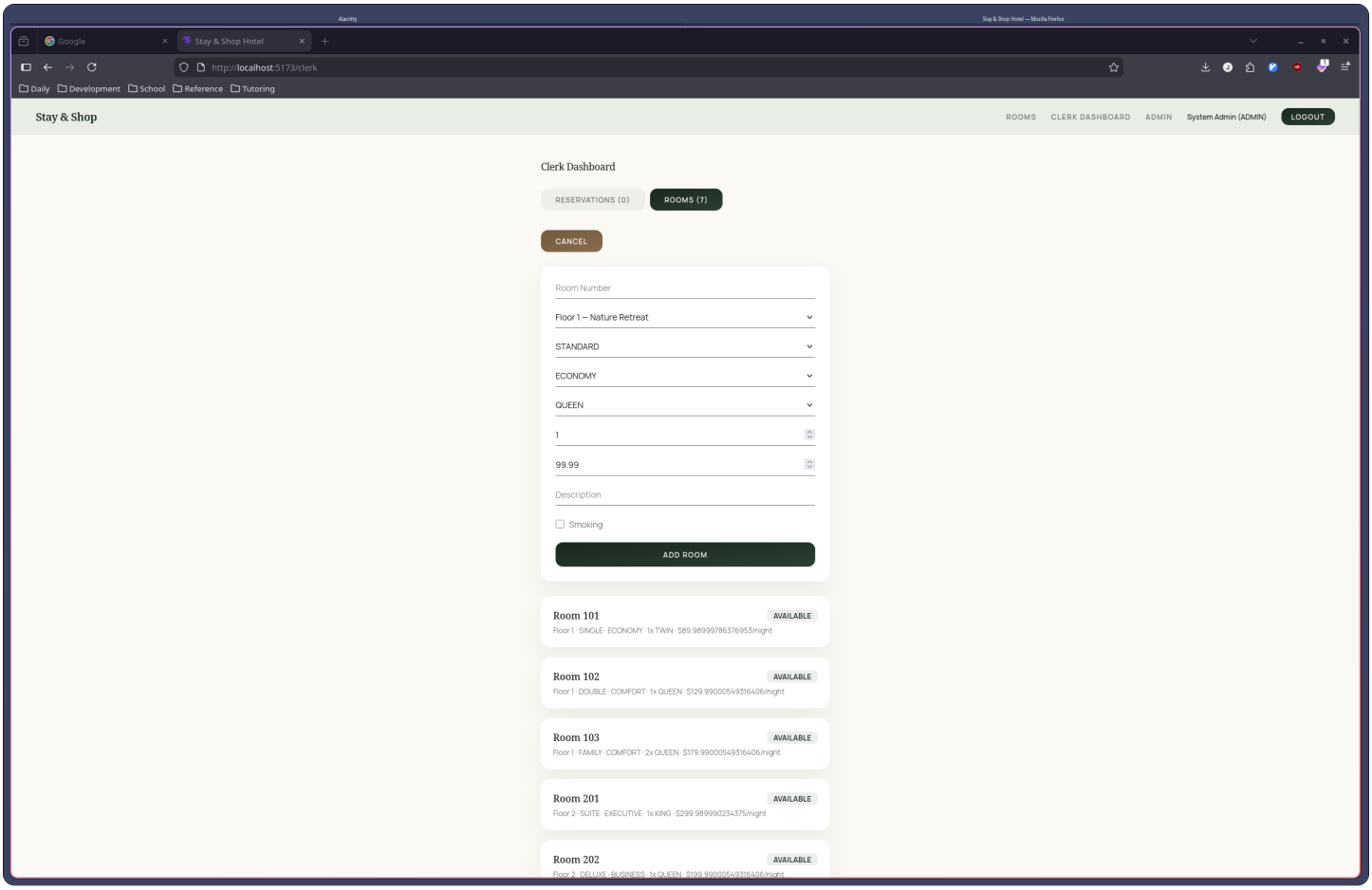
Use Case: Admin Login

- Author: Jace Yarborough
- Connected to Backend: YES
- Implemented By: Team effort prompt engineering



Use Case: Hotel Clerk Add Room

- Author: Jace Yarborough
- Connected to Backend: YES
- Implemented By: Team effort prompt engineering



Use cases: Reservations

Use Case: Search Available Room

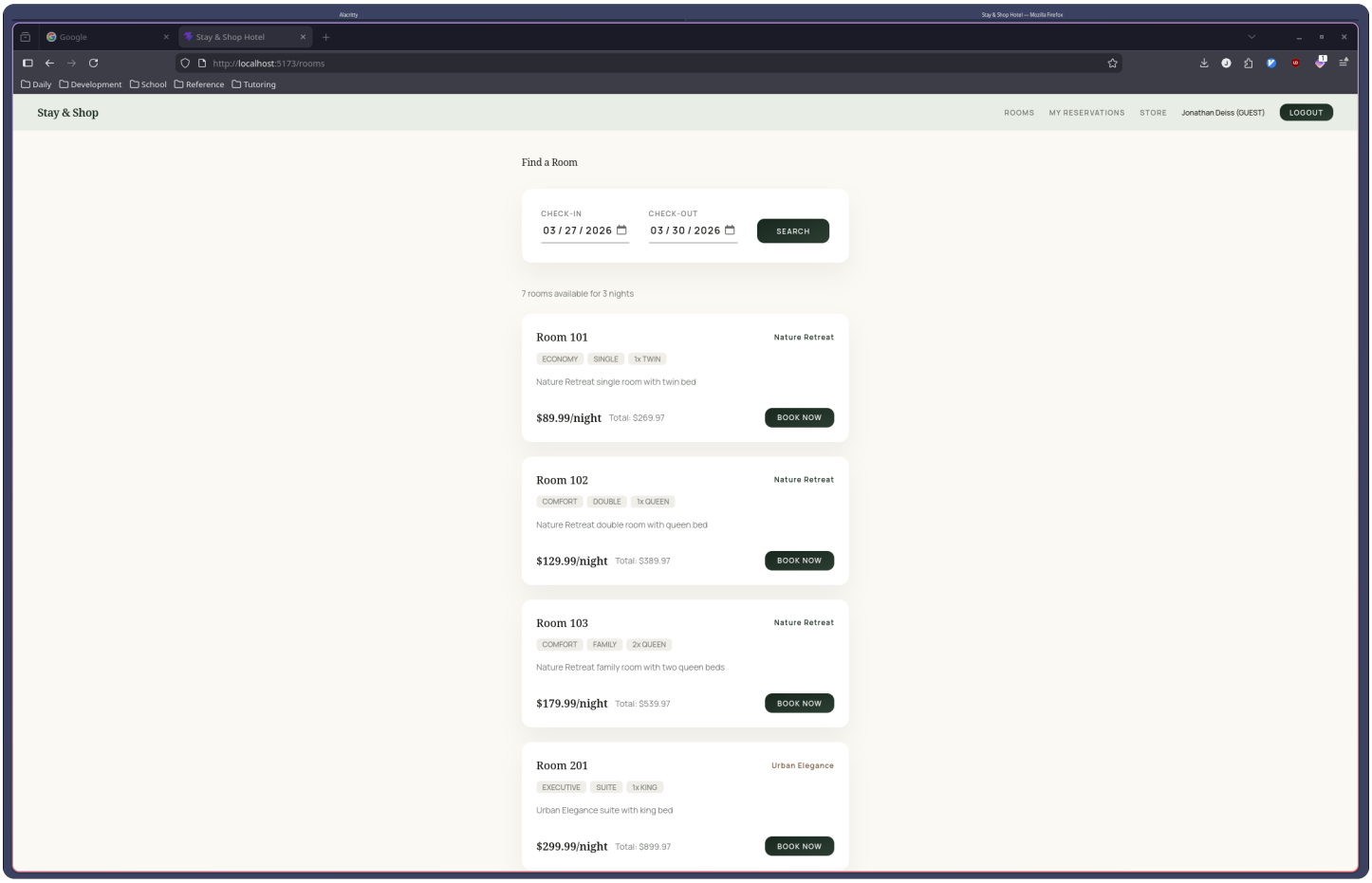
- Author: James Bagwell
- Connected to Backend: YES
- Implemented By: Team effort prompt engineering

Use Case: Make Reservation

- Author: Erick Martinez
- Connected to Backend: YES
- Implemented By: Team effort prompt engineering

Use Case: Cancel Reservation

- Author: Zain Altaf
- Connected to Backend: YES
- Implemented By: Team effort prompt engineering



Design Class Diagram

Prompt Engineering Summary

Since we have kept our documentation all in-repo using markdown files and built our diagrams with mermaid (textual form that an LLM can understand), we could easily prompt engineer. First we created a claude code instance to create an initial "rough draft:"

```
Read through our documentation in @Documentation and begin implementing. From a
technology standpoint, use:
- jOOQ
- Spring
- Frontend with react and bun as the bundler
```

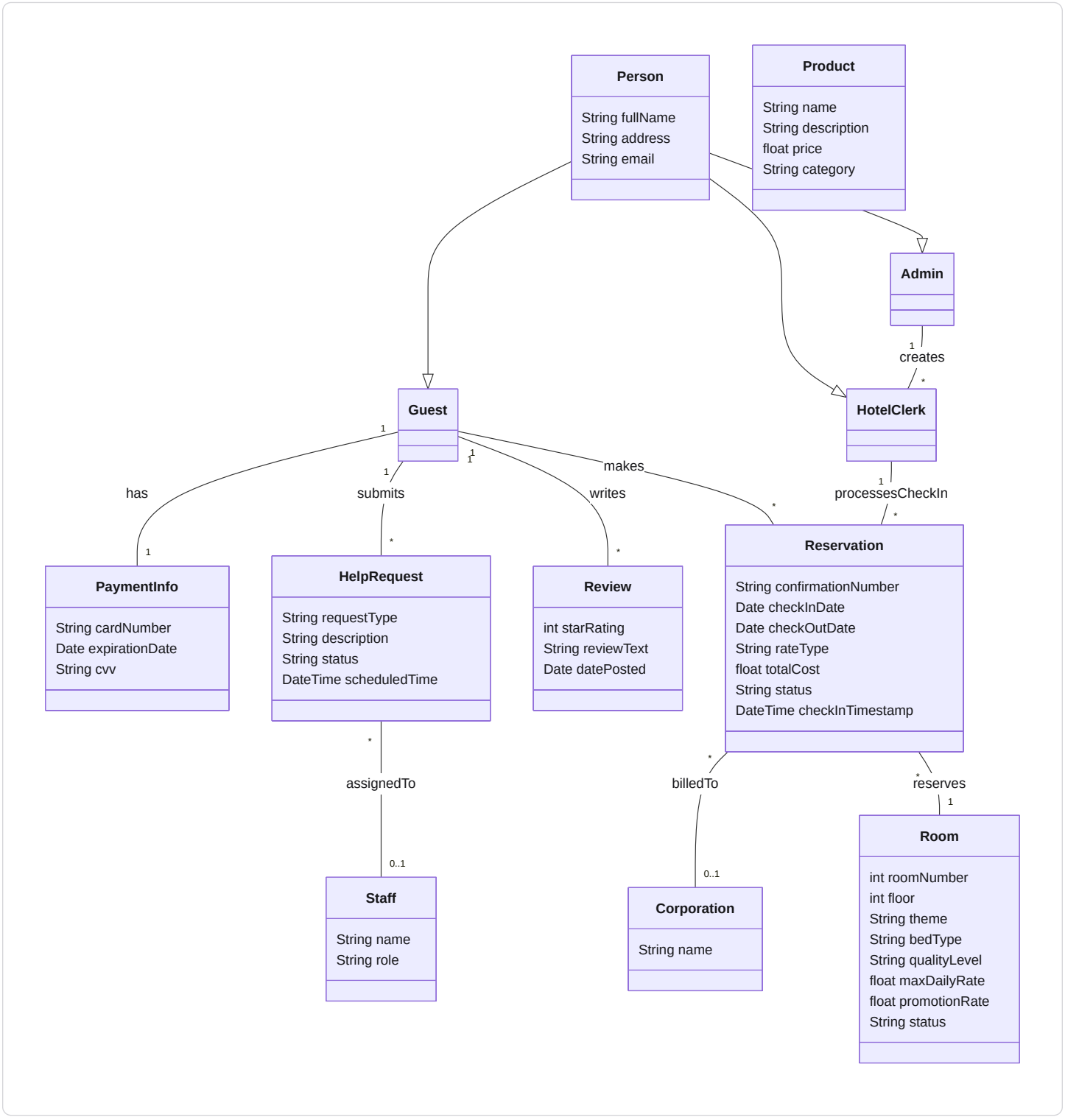
This looks simple and naive but the @Documentation directive gave the agent all documentation we had made up to this point, so it had the context needed to implement everything properly. This allowed the agent to create a working MVP, but it did not do good software design practices.

We provided it with a large document on software design, which is currently stored in our repo at `prompts/software-design.md`. This file was made by giving a prompt to another agent with most of your slides and asking it to summarize good software design in the context of our technologies. We told the agent to read this guide and create a plan for refactoring and testing. This plan is stored in our repo at `prompts/software-engineering-plan.md`.

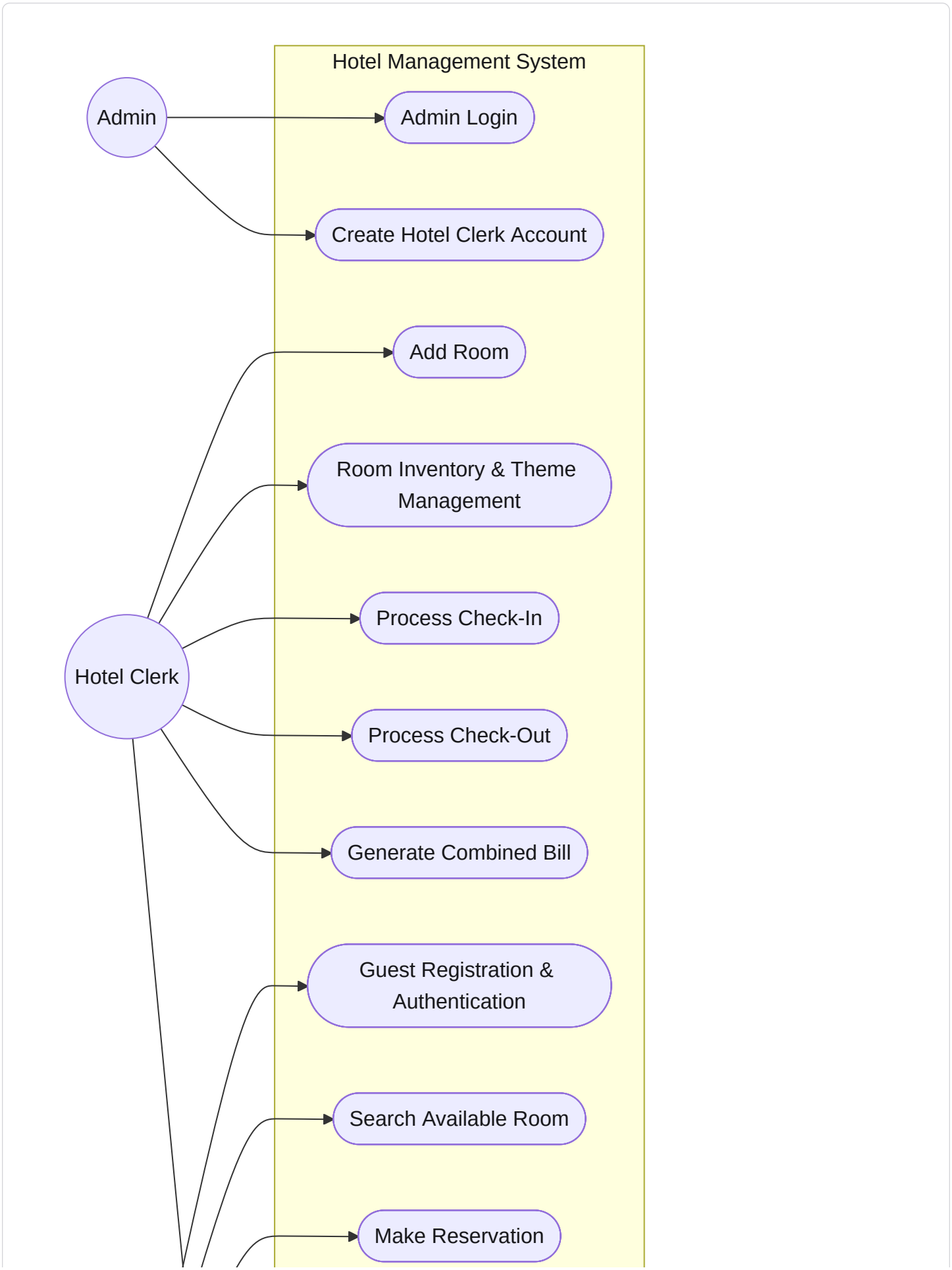
After that, we needed to improve the frontend code. We used a tool called stitch to create custom context that the agent could use to improve our frontend. We gave this context (stored in

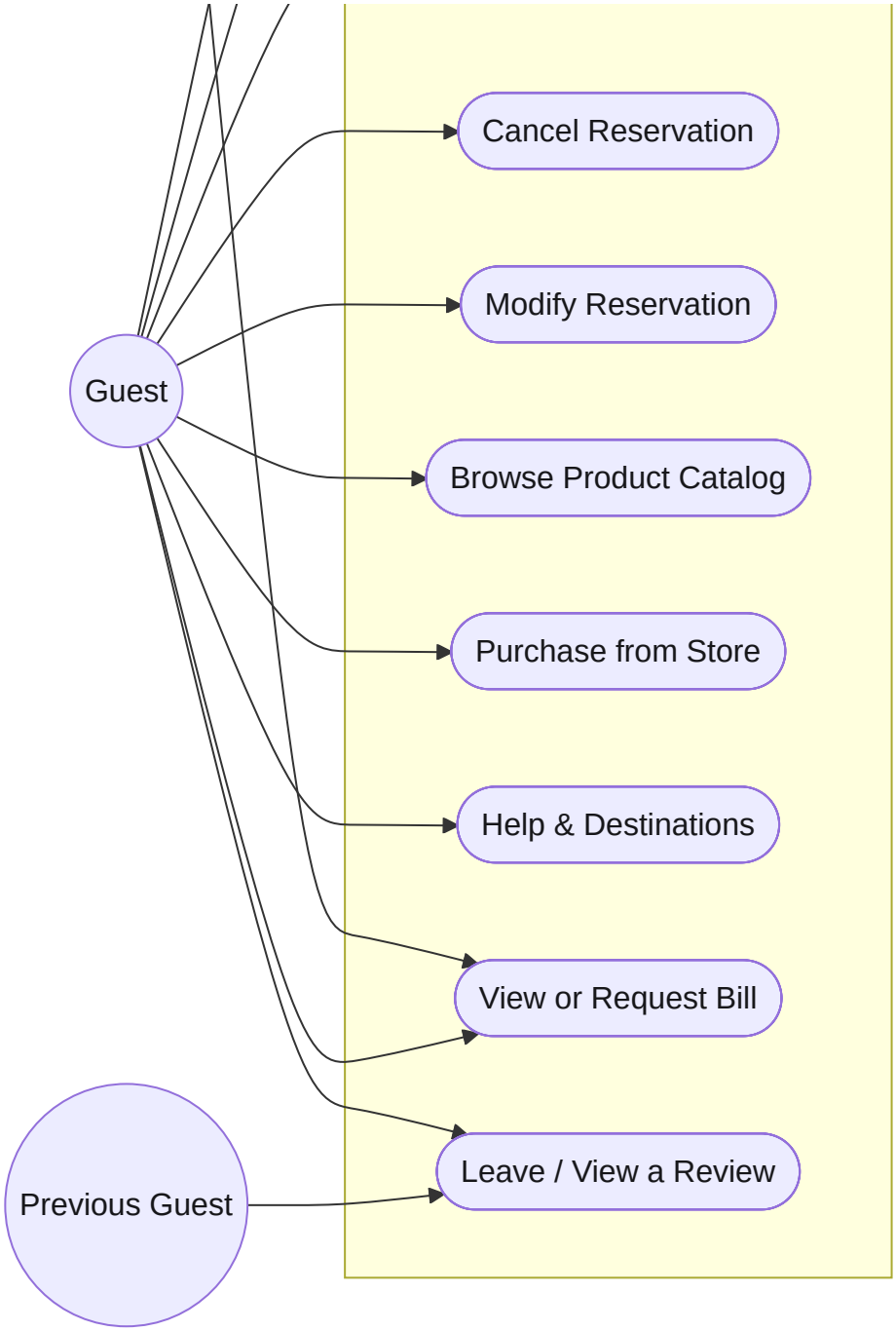
`Documentation/stitch_project_requirements_document`) to the agent and it correctly implemented our designs.

Domain Model



Use Case Diagram


Hotel Management System
Admin
Admin Login
Create Hotel Clerk Account
Add Room
Room Inventory & Theme Management
Process Check-In
Hotel Clerk
Process Check-Out
Generate Combined Bill
Guest Registration & Authentication
Search Available Room
Make Reservation

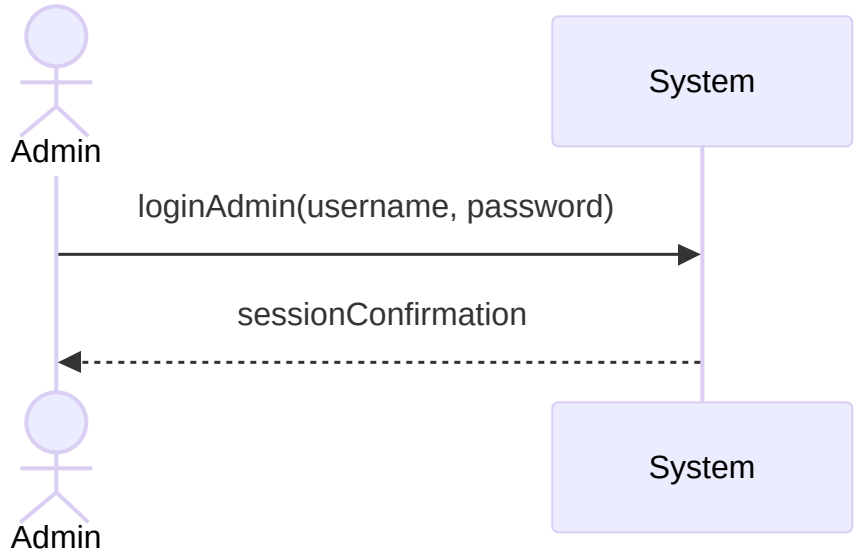


Use Cases

Admin Login

Use Case Name	**Admin Login**
Actor	Admin
Author	Jace Yarborough
Preconditions	1. System operational 2. User has a valid admin account with username and password
Postconditions	1. Admin is successfully logged in 2. Admin is redirected to admin dashboard/panel
Main Success Scenario	1. Admin navigates to login page 2. Admin enters username 3. Admin enters password 4. Admin submits credentials 5. System validates input 6. System verifies credentials 7. System displays success message 8. Admin is brought to admin dashboard
Extensions	[4]a. **Invalid username format** [4]a1 System detects username doesn't meet format requirements [4]a2 System displays error message "Invalid username or password" [4]a3 System prompts user to re-enter credentials [6]a. **Invalid credentials** [6]a1 System detects username or password is incorrect [6]a2 System increments failed login attempt counter [6]a3 System displays error message "Invalid username or password" [6]a4 Return to step 2 [6]b. **Account locked** [6]b1 System detects account has been locked due to multiple failed attempts [6]b2 System displays error message "Account locked. Contact system administrator" [6]b3 Use case ends [6]c. **Password expired**

Use Case Name	Admin Login
	[6]c1 System detects password has expired [6]c2 System prompts admin to reset password [6]c3 Redirect to password reset use case
Special Reqs	• Password must be hashed in database • Log all login attempts

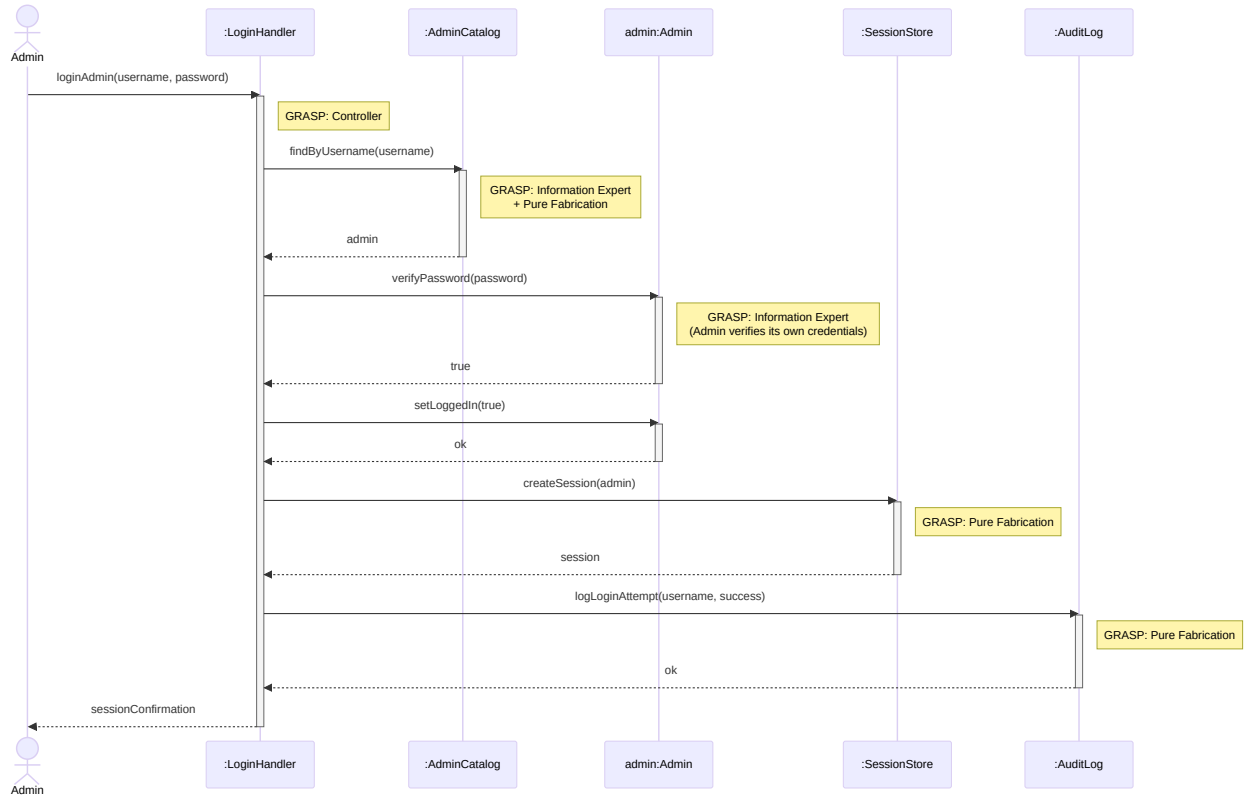


Operation Contract

Operation	`loginAdmin(username: String, password: String)`
Cross References	Use Case: Admin Login
Preconditions	1. System is operational 2. An admin account with the given username exists in the system
Postconditions	1. An admin session was created 2. Admin.isLoggedIn was set to true 3. The login attempt was logged

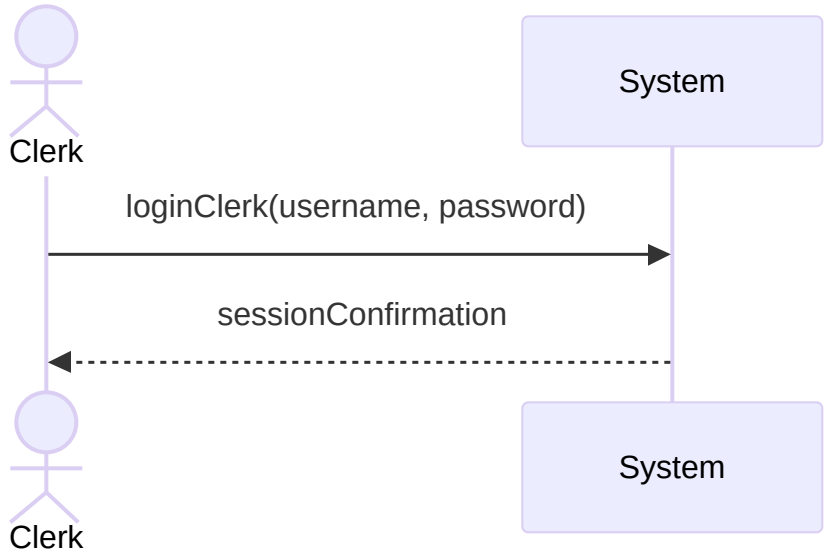
Design Sequence Diagram

Pattern	Applied To	Rationale
Controller	`:LoginHandler`	Use-case controller; receives the `loginAdmin` system operation
Information Expert + Pure Fabrication	`:AdminCatalog`	Holds all Admin accounts; finds by username and verifies credentials
Information Expert	`admin:Admin`	Manages its own `isLoggedIn` flag
Pure Fabrication	`:SessionStore`	Creates and stores the authenticated session
Pure Fabrication	`:AuditLog`	Logs all login attempts for auditing



Hotel Clerk Login

Use Case Name	**Hotel Clerk Login**
Actor	Hotel Clerk
Author	Jonathan Deiss
Preconditions	1. System is operational 2. User has a valid hotel clerk account with a username and password
Postconditions	1. Hotel clerk is successfully logged in 2. Clerk is redirected to the clerk dashboard
Main Success Scenario	1. The clerk navigates to the login page 2. The clerk enters their username 3. The clerk enters their password 4. The clerk submits the credentials 5. The system validates the input format 6. The system verifies the credentials against the database 7. The system creates a clerk session 8. The system redirects the clerk to the clerk dashboard
Extensions	[6]a. **Invalid credentials** [6]a1 The system displays "Invalid username or password" [6]a2 Return to step 2 [6]b. **Account not found** [6]b1 The system displays "Invalid username or password" (generic, for security) [6]b2 Use case ends
Special Reqs	● Passwords must be stored hashed in the database ● All login attempts (successful and failed) must be logged

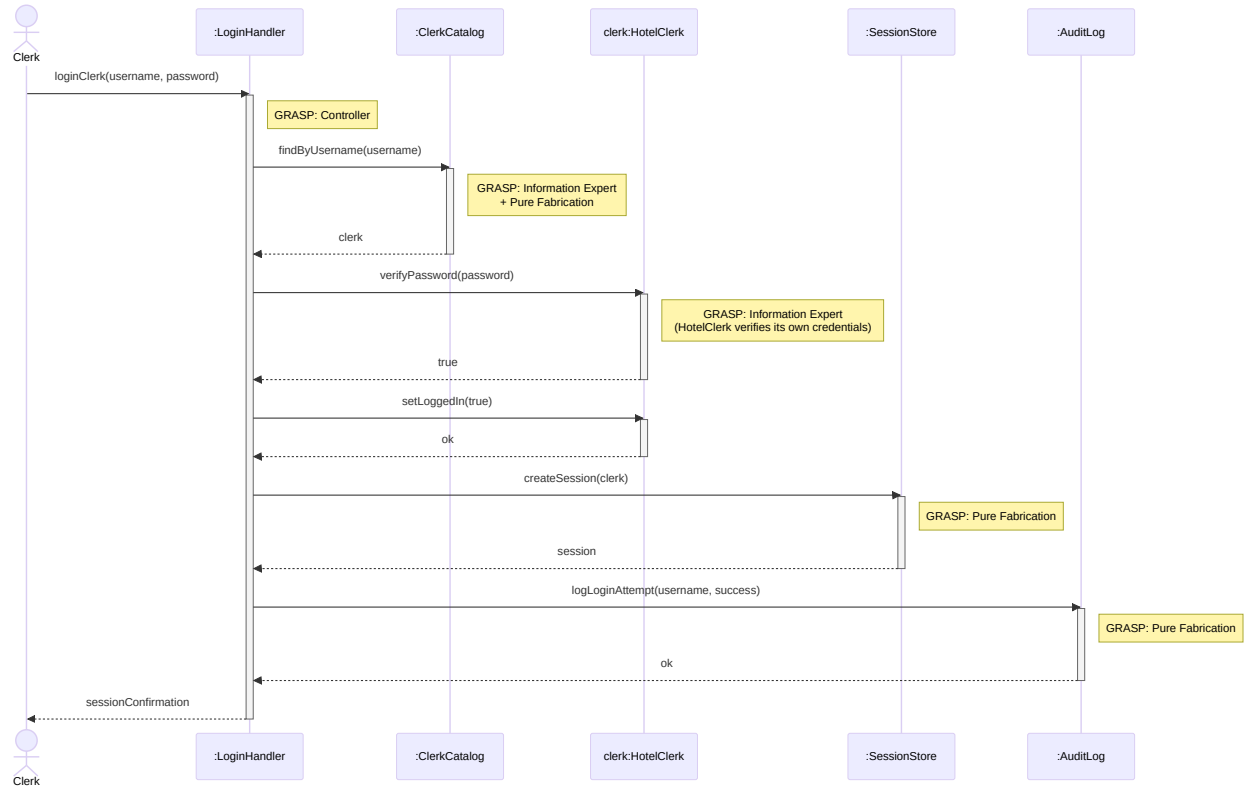


Operation Contract

Operation	`loginClerk(username: String, password: String)`
Cross References	Use Case: Hotel Clerk Login
Preconditions	1. System is operational 2. A hotel clerk account with the given username exists in the system
Postconditions	1. A clerk session was created 2. HotelClerk.isLoggedIn was set to true 3. The login attempt was logged

Design Sequence Diagram

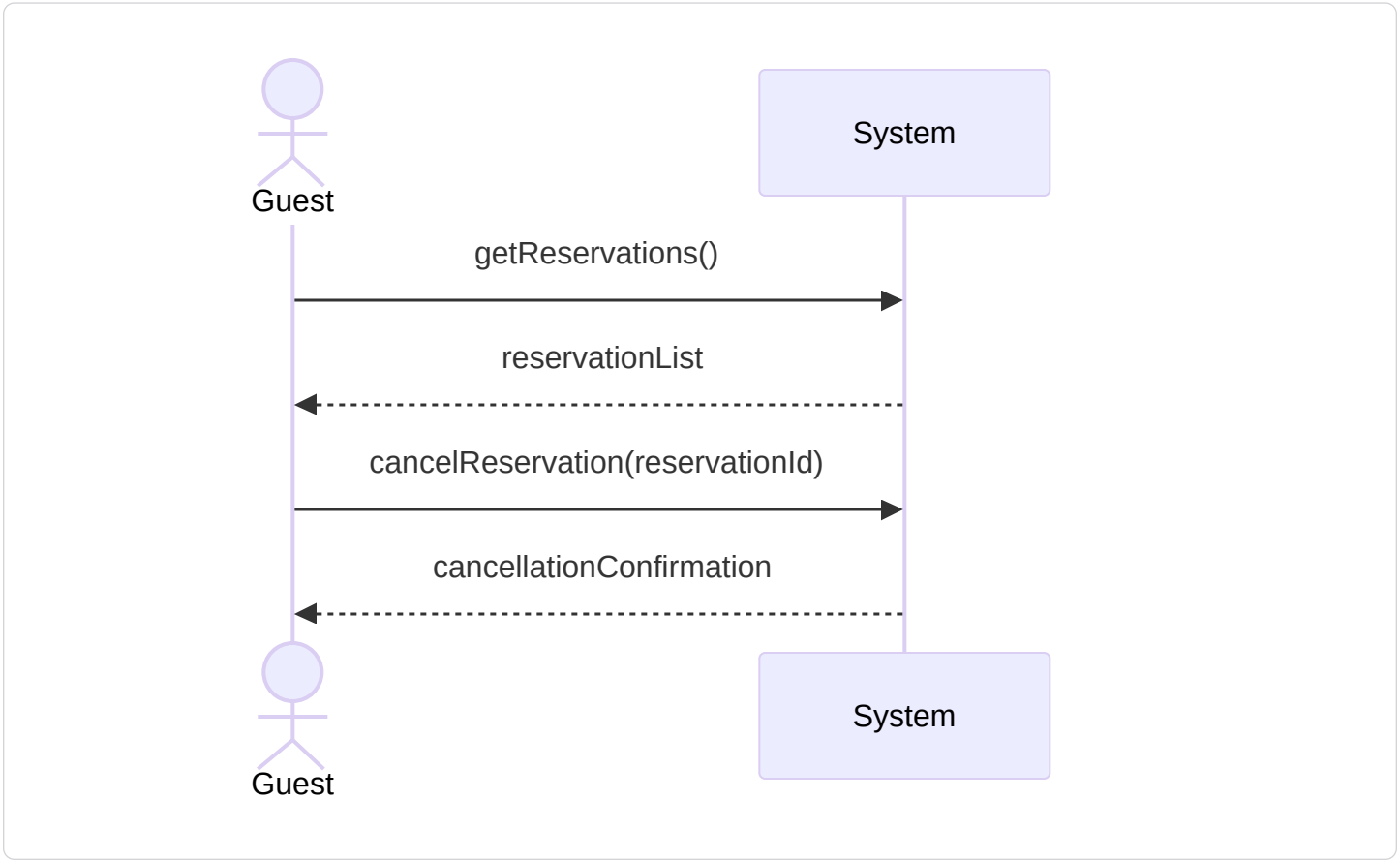
Pattern	Applied To	Rationale
Controller	`:LoginHandler`	Use-case controller; receives the `loginClerk` system operation
Information Expert + Pure Fabrication	`:ClerkCatalog`	Holds all HotelClerk accounts; finds by username and verifies credentials
Information Expert	`clerk:HotelClerk`	Manages its own `isLoggedIn` flag and verifies its own password
Pure Fabrication	`:SessionStore`	Creates and stores the authenticated clerk session
Pure Fabrication	`:AuditLog`	Logs all login attempts for auditing



Cancel Reservation

Use Case Name	**Cancel Reservation**
Actor	Hotel Guest
Author	Zain Altaf
Preconditions	1. The hotel guest is logged into the system. 2. The guest has an existing reservation.
Postconditions	1. The reservation is canceled only if cancellation is permitted. 2. If cancellation is permitted, any applicable cancellation penalty is recorded. 3.If cancellation is not permitted, the reservation remains unchanged.
Main Success Scenario	1. The guest selects the option to view reservations. 2. The system displays the guest's reservations. 3. The guest selects a reservation to cancel. 4. The system checks the time remaining until the reservation's check-in date. 5. The system determines that the cancellation request is more than the required time. 6. The system displays the applicable cancellation policy and any penalty(if required). 7. The guest confirms the cancellation. 8.The system cancels the reservation. 9. The system displays a cancellation confirmation message.
Extensions	[4]a. **Cancellation not allowed (within a specific time frame)** [4]a1 The system determines that the cancellation request is within x hours of the check-in time. [4]a2 The system displays a message explaining that cancellation is not permitted according to the policy. [4]a3 The reservation remains unchanged.
Special Reqs	● The system must enforce the X-hour cancellation policy exactly. ● Time comparisons must use the hotel's local time zone. ● All cancellation attempts must be logged for auditing and billing

Use Case Name	Cancel Reservation
	purposes.

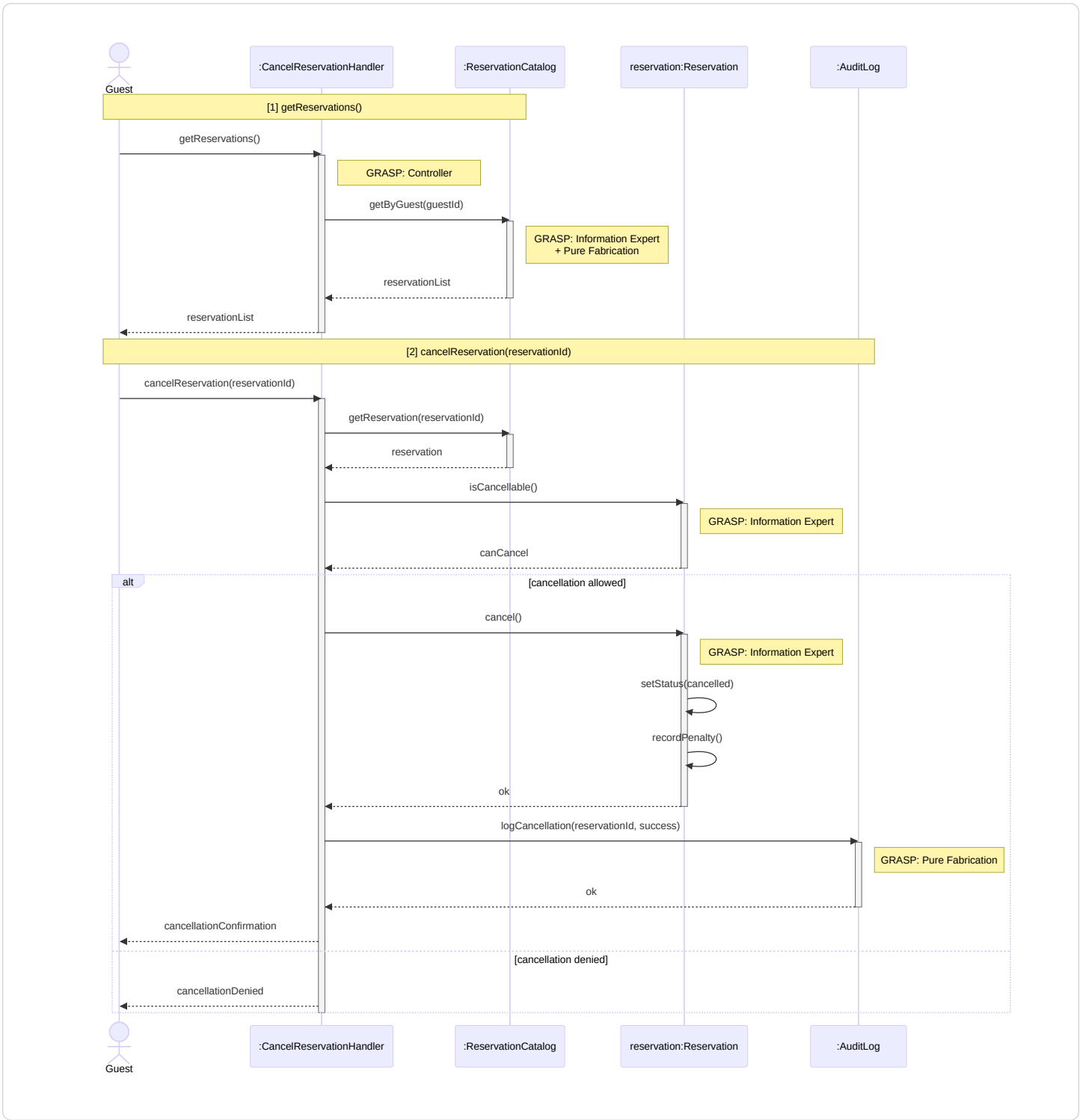


Operation Contract

Operation	`cancelReservation(reservationId: String)`
Cross References	Use Case: Cancel Reservation
Preconditions	1. Guest is logged in 2. Reservation exists and is associated with the guest 3. The cancellation request is more than X hours before the check-in time
Postconditions	1. Reservation.status was set to 'cancelled' 2. Any applicable cancellation penalty was recorded and associated with the reservation 3. The cancellation attempt was logged for auditing

Design Sequence Diagram

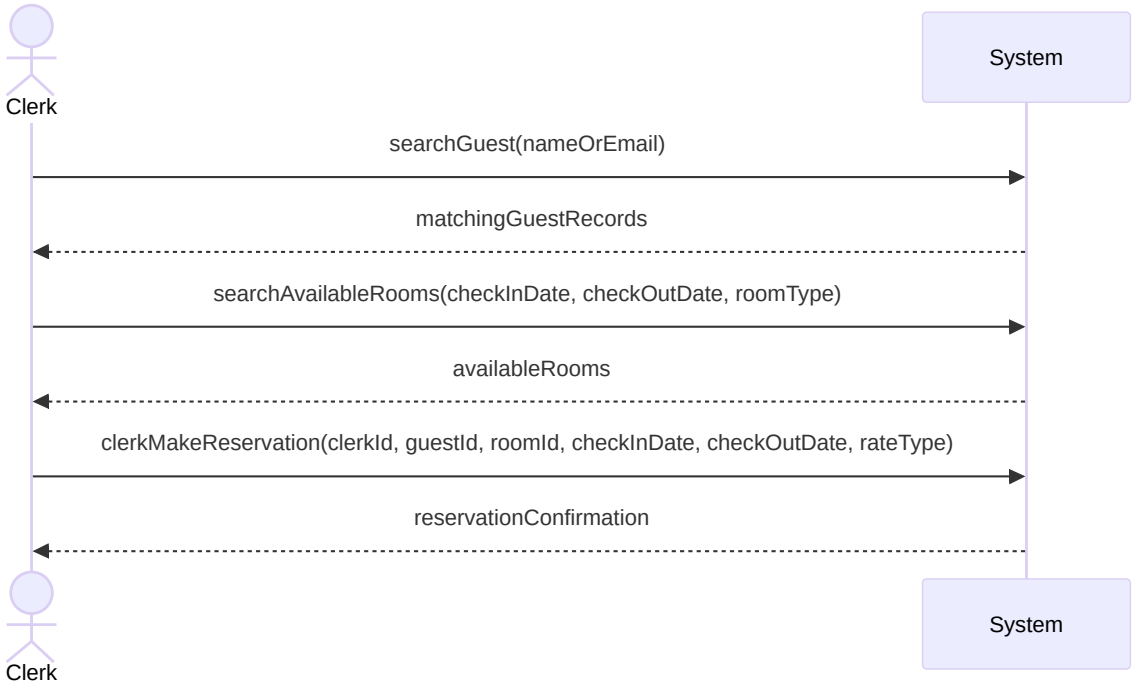
Pattern	**Applied To**	**Rationale**
Controller	`:CancelReservationHandler`	Use-case controller; handles both system operations for this use case session
Information Expert + Pure Fabrication	`:ReservationCatalog`	Holds all Reservation data; retrieves reservations by guest and by ID
Information Expert	`reservation:Reservation`	Has `checkInDate` — enforces the X-hour cancellation policy; sets its own status and records the penalty
Pure Fabrication	`:AuditLog`	Logs all cancellation attempts for auditing



Clerk Makes Reservation for Guest

Use Case Name	**Clerk Makes Reservation for Guest**
Actor	Hotel Clerk
Author	Jonathan Deiss
Preconditions	1. Hotel clerk is logged into the system 2. Room and reservation data exists in the database 3. The guest has an existing account or the clerk can look one up
Postconditions	1. A new reservation is created in the system and associated with the guest 2. The selected room is marked as reserved for the specified dates 3. Guest information is recorded and linked to the reservation
Main Success Scenario	1. The clerk selects "Make Reservation" from the clerk dashboard 2. The clerk searches for the guest by name or email 3. The system displays matching guest records 4. The clerk selects the guest 5. The clerk enters the check-in date, check-out date, and desired room type 6. The system displays available rooms matching the criteria 7. The clerk selects a room 8. The clerk selects a rate type (standard, promotion, group, or non-refundable) 9. The system calculates the total cost based on the room's quality level and rate type 10. The system creates the reservation and associates it with the guest 11. The system displays the reservation confirmation details
Extensions	[3]a. **Guest not found** [3]a1 The clerk creates a new guest record [3]a2 Continue from step 5 [6]a. **No rooms available for requested criteria** [6]a1 The system notifies the clerk that no rooms match the criteria [6]a2 The clerk adjusts dates or room type and returns to step 5 [8]a. **Corporate guest** [8]a1 The clerk selects the guest's corporation

Use Case Name	Clerk Makes Reservation for Guest
	[8]a2 The system marks the reservation for deferred corporate billing [8]a3 Continue from step 9
Special Reqs	● Reservations created by a clerk must be flagged as clerk-initiated for auditing purposes ● Corporate guest reservations must be marked for deferred billing and not charged at time of booking

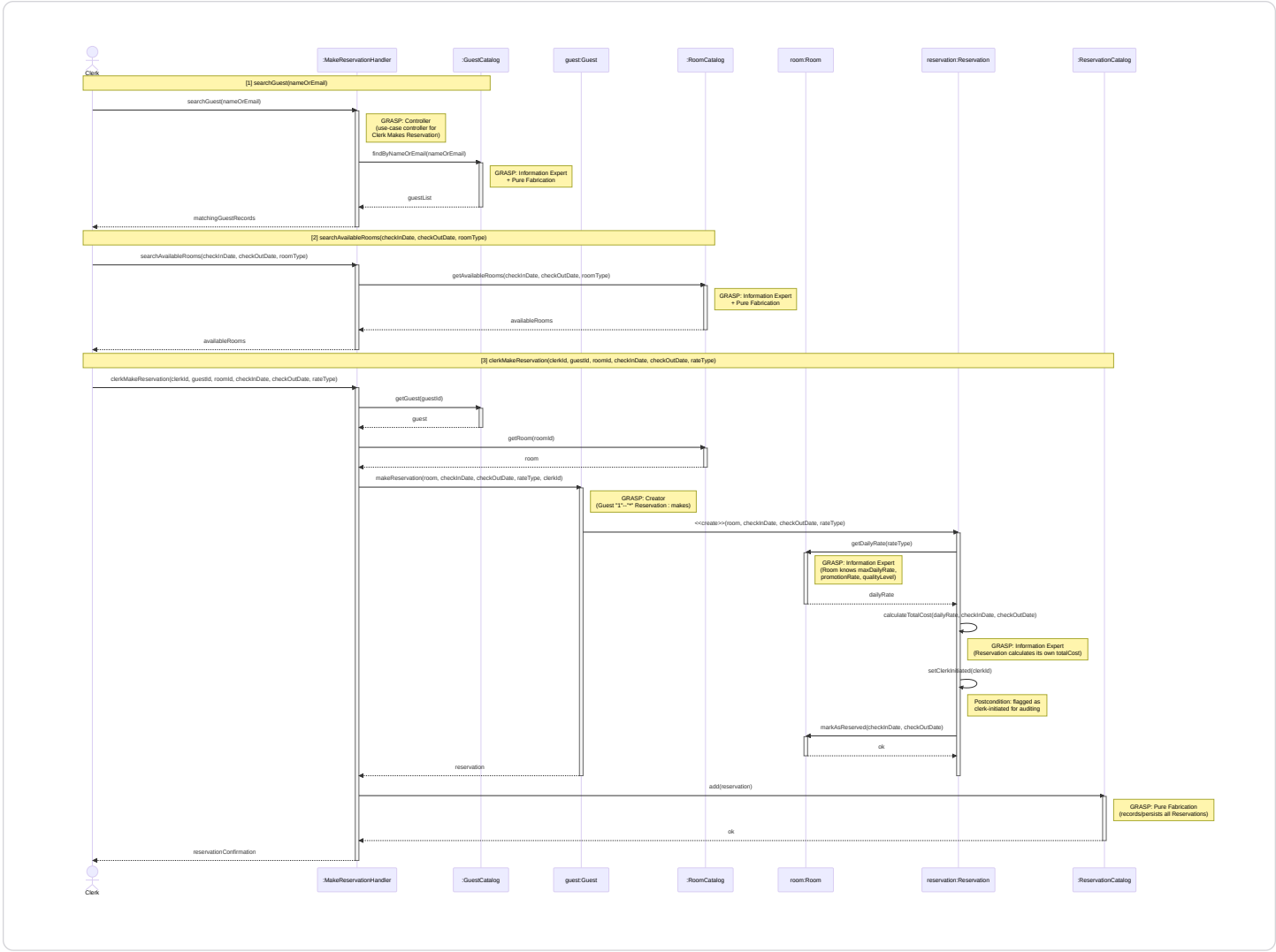


Operation Contract

Operation	`clerkMakeReservation(clerkId: String, guestId: String, roomId: String, checkInDate: Date, checkOutDate: Date, rateType: String)`
Cross References	Use Case: Clerk Makes Reservation for Guest
Preconditions	1. Hotel clerk is logged in 2. The specified guest account exists in the system 3. The selected room is available for the requested dates
Postconditions	1. A new Reservation was created and associated with the guest 2. Selected Room was marked as reserved for the specified dates 3. Reservation.totalCost was calculated based on quality level and rate type 4. Reservation was flagged as clerk-initiated and linked to the creating clerk's account

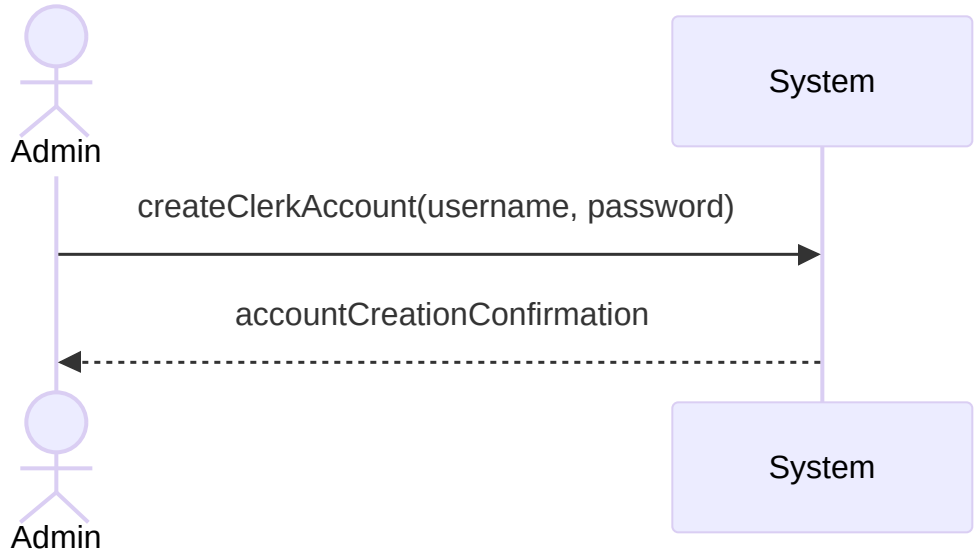
Design Sequence Diagram

Pattern	Applied To	Rationale
Controller	`:MakeReservationHandler`	Use-case controller; handles all three system operations for this use case session
Information Expert + Pure Fabrication	`:GuestCatalog`	Holds all Guest data; no direct domain counterpart
Information Expert + Pure Fabrication	`:RoomCatalog`	Holds all Room data; knows which rooms are available
Creator	`guest:Guest`	Domain model shows `Guest "1"--"*" Reservation : makes` ; Guest aggregates Reservations
Information Expert	`room:Room`	Has `maxDailyRate` , `promotionRate` , `qualityLevel` — expert on rate data
Information Expert	`reservation:Reservation`	Has `rateType` , `checkInDate` , `checkOutDate` — calculates its own `totalCost`
Pure Fabrication	`:ReservationCatalog`	Records and persists all Reservations without burdening domain objects



Create Hotel Clerk Account

Use Case Name	**Create Hotel Clerk Account**
Actor	Admin
Author	Jace Yarborough
Preconditions	1. Hotel system online and operational 2. User is logged in as an Admin
Postconditions	1. A new hotel clerk account is created 2. Clerk account has given username and default password (or custom password)
Main Success Scenario	1. Admin selects option to create hotel clerk account 2. System prompts admin to enter desired username and shows prefilled password for account. 3. Admin enters username and optional different password 4. System validates input 5. System creates clerk account 6. System displays success message for created account
Extensions	[4]a. **Username already in use** [4]a1 System detects username already in use(Ex: John_Smith) [4]a2 System displays error message and potential username replacement (EX: John_Smith1) [5]a. **Failure to create account** [5]a1 Display error message of account creation failure [5]a2 Reprompt user to try creating account again.
Special Reqs	● Create account in timely manner ● Keep log of created accounts ● Keep log of which admin created account

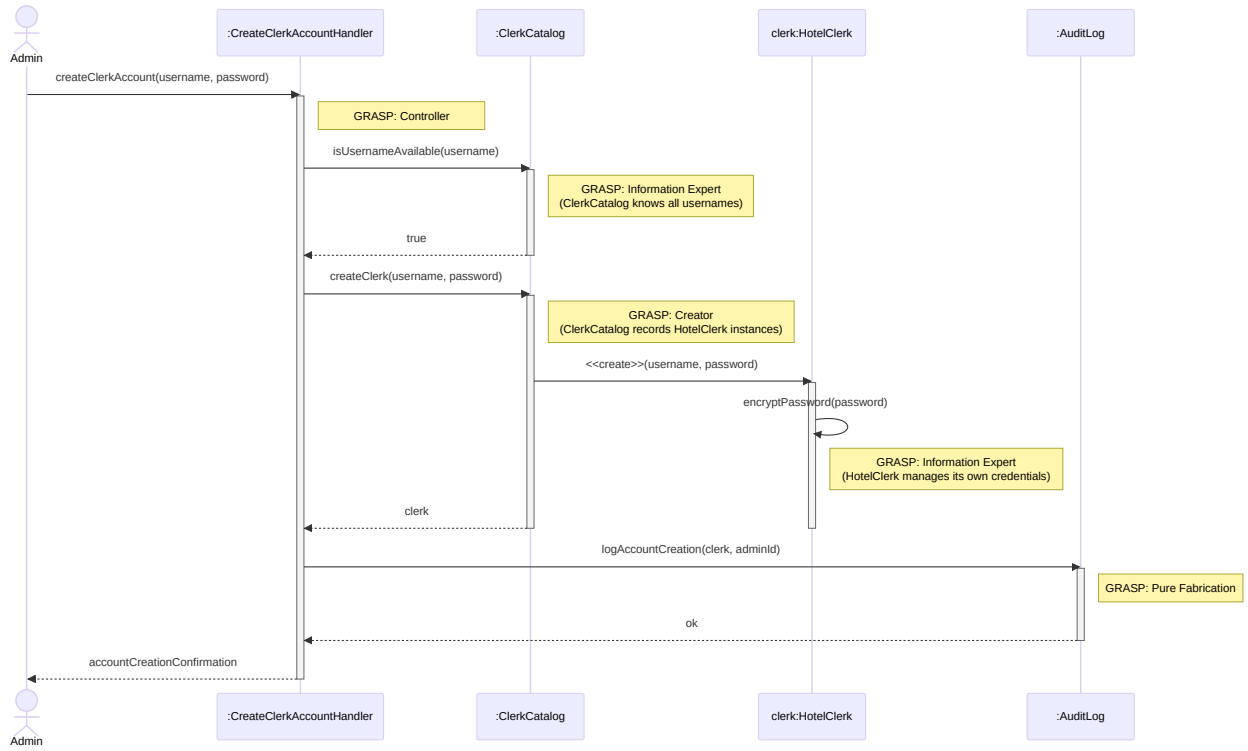


Operation Contract

Operation	`createClerkAccount(username: String, password: String)`
Cross References	Use Case: Create Hotel Clerk Account
Preconditions	1. Admin is logged in 2. The given username does not already exist in the system
Postconditions	1. A new HotelClerk account was created 2. HotelClerk.username was set 3. HotelClerk.password was encrypted and stored 4. Account creation was logged with the creating admin's identity

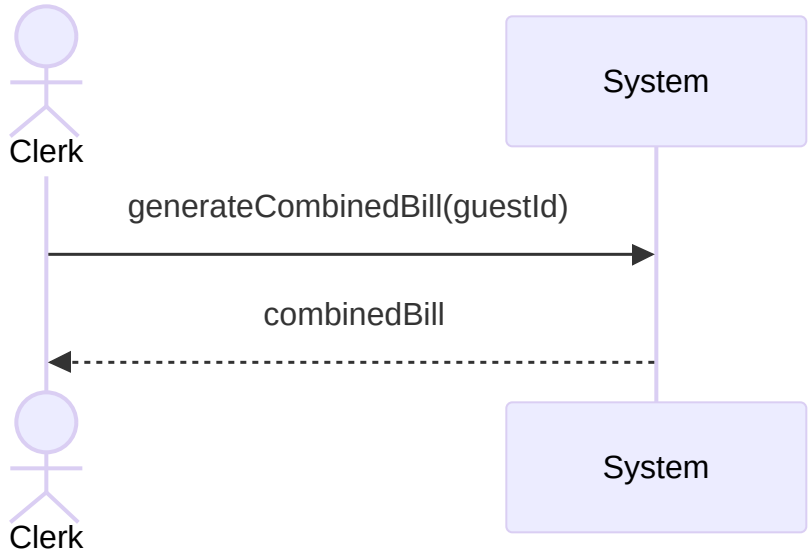
Design Sequence Diagram

Pattern	Applied To	Rationale
Controller	`:CreateClerkAccountHandler`	Use-case controller; receives the `createClerkAccount` system operation
Information Expert + Pure Fabrication	`:ClerkCatalog`	Knows all existing usernames; checks uniqueness before creation
Creator	`:ClerkCatalog`	Records HotelClerk instances (GRASP Creator: B records A → B creates A)
Information Expert	`clerk:HotelClerk`	Manages its own password encryption
Pure Fabrication	`:AuditLog`	Logs account creation with the admin's identity



Generate Combined Bill

Use Case Name	**Generate Combined Bill**
Actor	Hotel Clerk
Author	Zain Altaf
Preconditions	1. The hotel clerk is logged into the system. 2. The guest has completed check-out. 3. The guest has at least one reservation recorded in the system.
Postconditions	1. A combined bill is generated for the guest. 2. The bill includes all room charges and store purchases. 3. The finalized bill is stored in the system.
Main Success Scenario	1. The clerk selects a checked-out guest. 2. The system retrieves the guest's reservation details. 3. The system retrieves all store purchases made during the guest's stay. 4. The system calculates the total room charges. 5. The system calculates the total store charges. 6. The system applies any taxes or additional fees. 7. The system combines all charges into a single bill. 8. The system displays the bill summary. 9. The clerk reviews and confirms the bill. 10. The system finalizes and stores the bill.
Extensions	[3]a. **No store purchases recorded** [3]a1 The system generates a bill including only room charges. [2]b. **Corporate guest billing** [2]b1 The system marks the bill as corporate billing. [2]b2 The payment status is set to pending.
Special Reqs	● Bill calculations must be accurate and consistent with reservation and purchase records. ● Tax calculations must follow applicable hotel policies. ● The generated bill must be stored for auditing and reporting purposes.

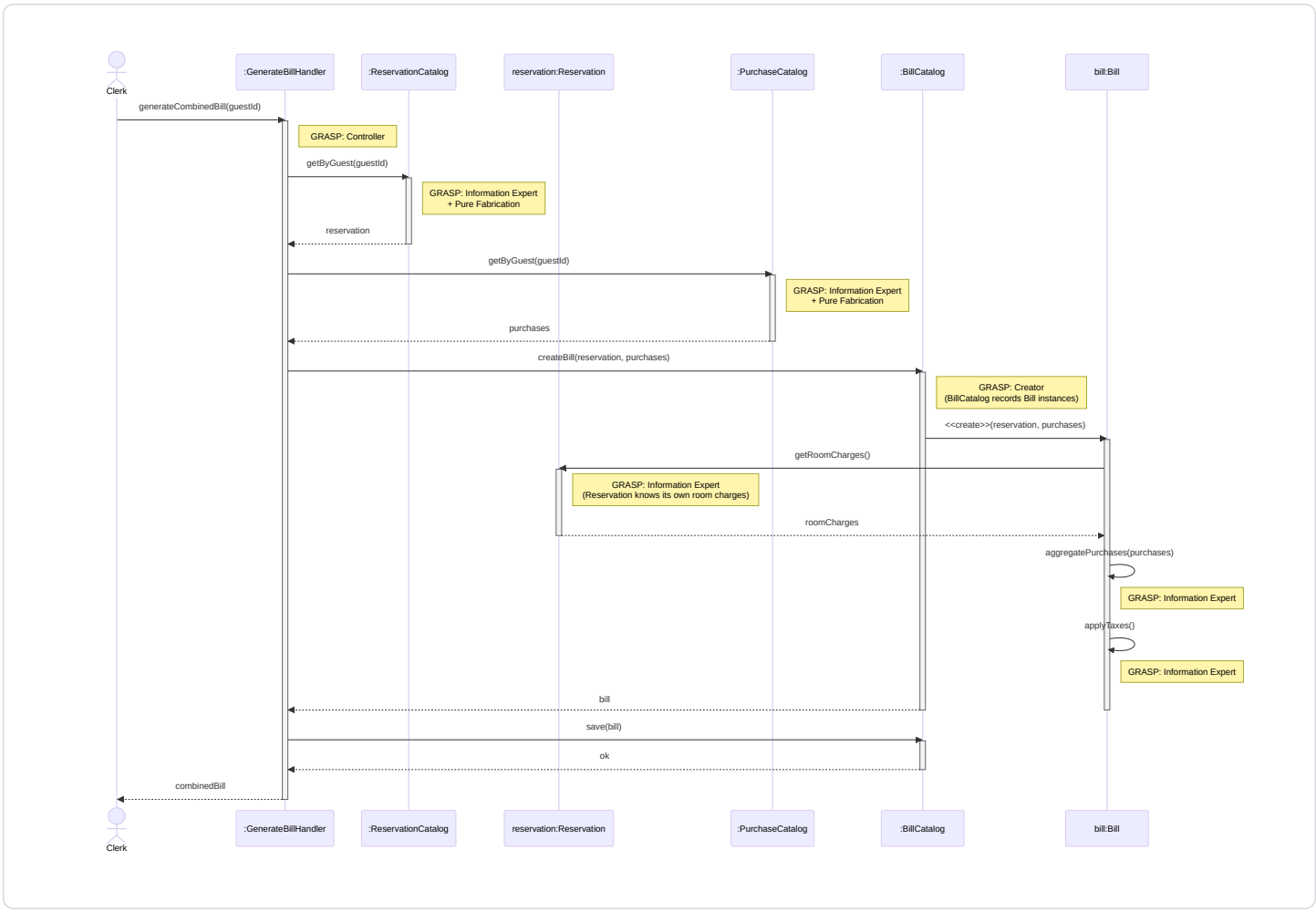


Operation Contract

Operation	`generateCombinedBill(guestId: String)`
Cross References	Use Case: Generate Combined Bill
Preconditions	1. Hotel clerk is logged in 2. Guest has completed check-out 3. At least one reservation is recorded for the guest
Postconditions	1. A combined Bill was created and associated with the guest 2. Bill included all room charges from the guest's stay 3. Bill included all store purchase charges from the guest's stay 4. Applicable taxes and fees were applied to the total 5. Finalized bill was stored in the system for auditing

Design Sequence Diagram

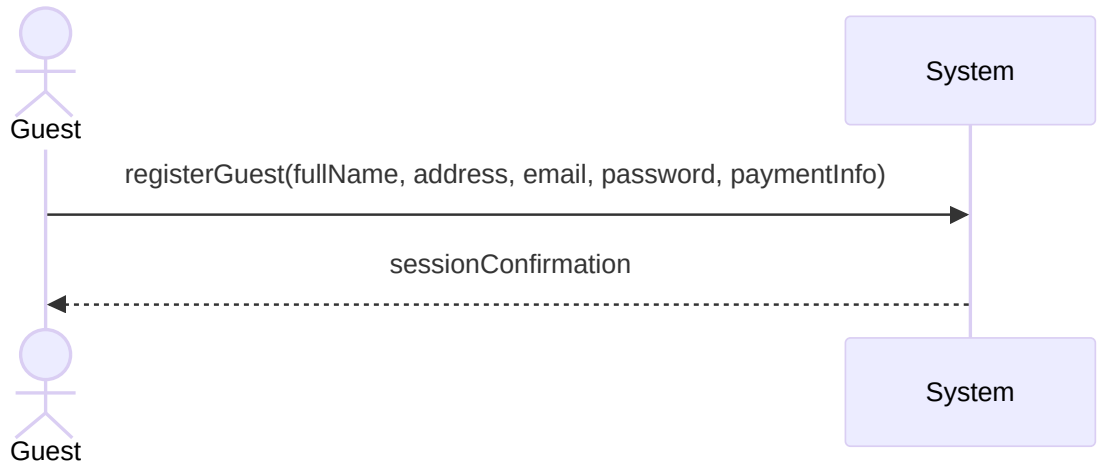
Pattern	Applied To	Rationale
Controller	`:GenerateBillHandler`	Use-case controller; receives the `generateCombinedBill` system operation
Information Expert + Pure Fabrication	`:ReservationCatalog`	Holds all Reservation data; retrieves the guest's reservation
Information Expert + Pure Fabrication	`:PurchaseCatalog`	Holds all store purchase records for a guest's stay
Information Expert	`reservation:Reservation`	Knows its own room charges (rate, dates, totalCost)
Creator + Pure Fabrication	`:BillCatalog`	Records Bill instances → creates Bill; stores finalized bill
Information Expert	`bill:Bill`	Aggregates all charges and applies taxes to its own total



Guest Registration & Authentication

Use Case Name	**Guest Registration & Authentication**
Actor	Guest
Author	Erick Martinez
Preconditions	1. The guest has access to the hotel system portal 2. The guest is not currently logged into an existing account
Postconditions	1. A new guest profile is created in the database 2. Payment information is securely tokenized/stored 3. The guest is automatically logged in and redirected to the dashboard 4. A "Welcome [Name]" message is displayed
Main Success Scenario	1. The guest selects the "Register" or "Create Account" option 2. The guest enters personal details: Full Name, Address, Email, and Password 3. The guest enters payment details: Credit Card Number, Expiration Date, and CVV 4. The system validates the format of all fields (e.g., email syntax, credit card) 5. The system checks if the email address is already registered 6. The system encrypts the password and stores the guest profile 7. The system authenticates the new session 8. The system displays a "Welcome [Guest Name]" message on the homepage/dashboard
Extensions	[4]a. **Invalid Data Format** [4]a1 The system highlights the specific field (e.g., "Invalid Credit Card Format") [4]a2 The guest corrects the data [4]a3 Continue from step 4 [5]a. **Email Already Exists** [5]a1 The system notifies the guest that an account already exists with that email [5]a2 The system offers a "Forgot Password" or "Login" link [5]a3 Use case ends

Use Case Name	Guest Registration & Authentication
	[7]a. **Authentication Failure** [7]a1 The system creates the account but fails the initial login [7]a2 The system redirects the guest to the manual Login page
Special Reqs	● PCI Compliance: Credit card data must be handled according to security standards (e.g., masking numbers in the UI) ● Data Integrity: The "Welcome" message must dynamically pull the FirstName attribute from the database ● Persistence: Guest information must remain accessible for future "Store" purchases without re-entry

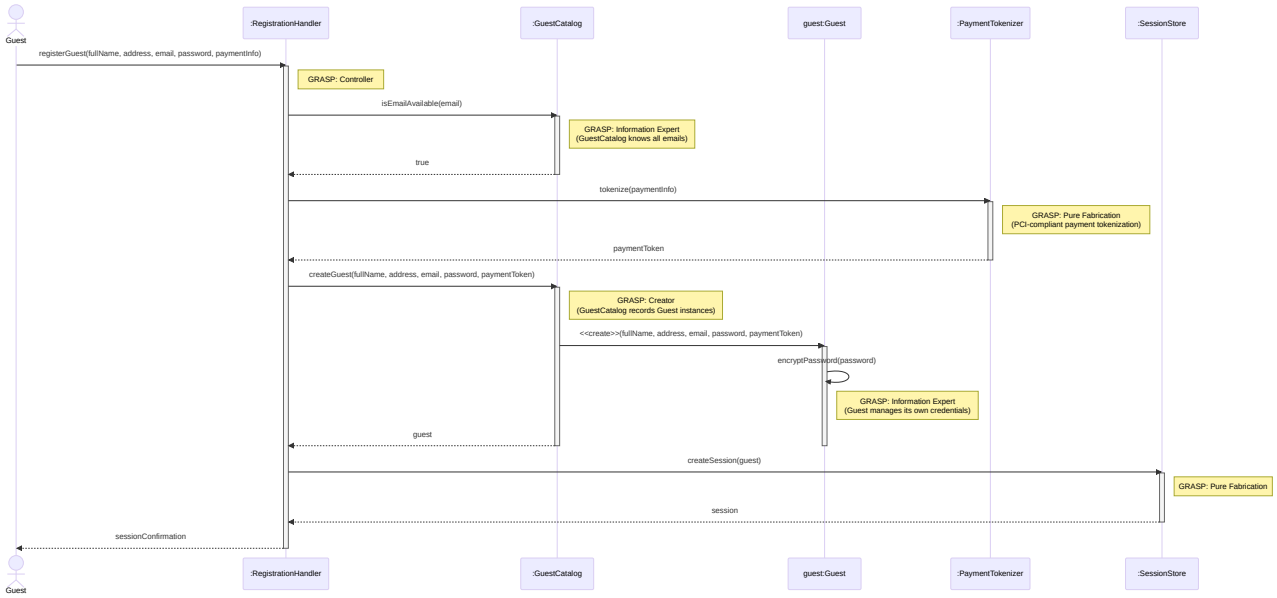


Operation Contract

Operation	`registerGuest(fullName: String, address: String, email: String, password: String, paymentInfo: PaymentInfo)`
Cross References	Use Case: Guest Registration & Authentication
Preconditions	1. Guest has access to the hotel system portal 2. Guest is not currently logged in 3. The given email address is not already registered
Postconditions	1. A new Guest profile was created in the database 2. Guest.password was encrypted and stored 3. Payment information was securely tokenized and stored 4. A new authenticated session was created and associated with the guest

Design Sequence Diagram

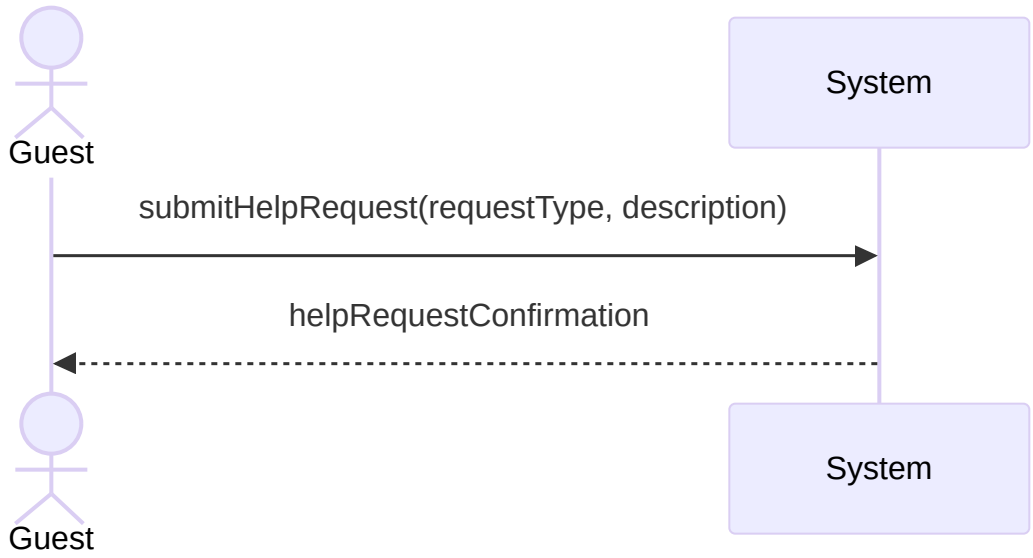
Pattern	Applied To	Rationale
Controller	`:RegistrationHandler`	Use-case controller; receives the `registerGuest` system operation
Information Expert + Pure Fabrication	`:GuestCatalog`	Knows all registered emails; checks uniqueness before creation
Creator	`:GuestCatalog`	Records Guest instances (GRASP Creator: B records A → B creates A)
Information Expert	`guest:Guest`	Manages its own password encryption
Pure Fabrication	`:PaymentTokenizer`	Tokenizes payment info for PCI compliance; no domain counterpart
Pure Fabrication	`:SessionStore`	Creates and stores the authenticated session


Guest
:RegistrationHandler
:GuestCatalog
guest:Guest
:PaymentTokenizer
:SessionStore
registerGuest(fullName, address, email, password, paymentInfo)
GRASP: Controller
isEmailAvailable(email)
GRASP: Information Expert
(GuestCatalog knows all emails)
true
tokenize(paymentInfo)
GRASP: Pure Fabrication
(PCI-compliant payment tokenization)
paymentToken
createGuest(fullName, address, email, password, paymentToken)
GRASP: Creator
(GuestCatalog records Guest instances)
<<create>>(fullName, address, email, password, paymentToken)
encryptPassword(password)
GRASP: Information Expert
(Guest manages its own credentials)
guest
createSession(guest)
GRASP: Pure Fabrication
session
sessionConfirmation
Guest
:RegistrationHandler
:GuestCatalog
guest:Guest
:PaymentTokenizer
:SessionStore

HelpDesk

Use Case Name	**HelpDesk**
Actor	Guest
Author	James Bagwell
Preconditions	1. There is a staff member on standby to help over the computer / phone 2. There is a staff member ready to complete the scheduled service at the time it was scheduled
Postconditions	1. Guest received help they needed 2. Guest schedules a service and reason for scheduling
Main Success Scenario	1. User logs into the hotel website and navigates to the HelpDesk menu 2. If they need technical help (such as Wi-Fi not working, how to use the phone, etc.), they select that option. If they need a technician or a house-keeper (for example, if the air-conditioning isn't working or the toilet is clogged), they choose the other option. 3. If the user selects the first option, they are able to chat on the computer with someone who can help. If they choose the second option, they can request a technician or house-keeper to fix the situation, and the system will assign and schedule someone to come help as soon as possible. 4. The situation is fixed.
Extensions	2a. **No Virtual Technician Available** 2a1. No staff member is available for technical support. 2a2. The system displays a message notifying the yuser of a delay and it and allows the guest to submit a request to get a call back later. 3b. **No Technician or House-keeper Available** 3b1. No technician or house-keeper is available at the requested time. 3b2. The system prompts the guest to select an alternate time for the service request.

Use Case Name	HelpDesk
Special Reqs	● The HelpDesk system must always be accessible through the hotel website. ● Live chat must occur quickly. ● All help requests and service schedules must be logged and associated with the guest's room number and account / phone number.

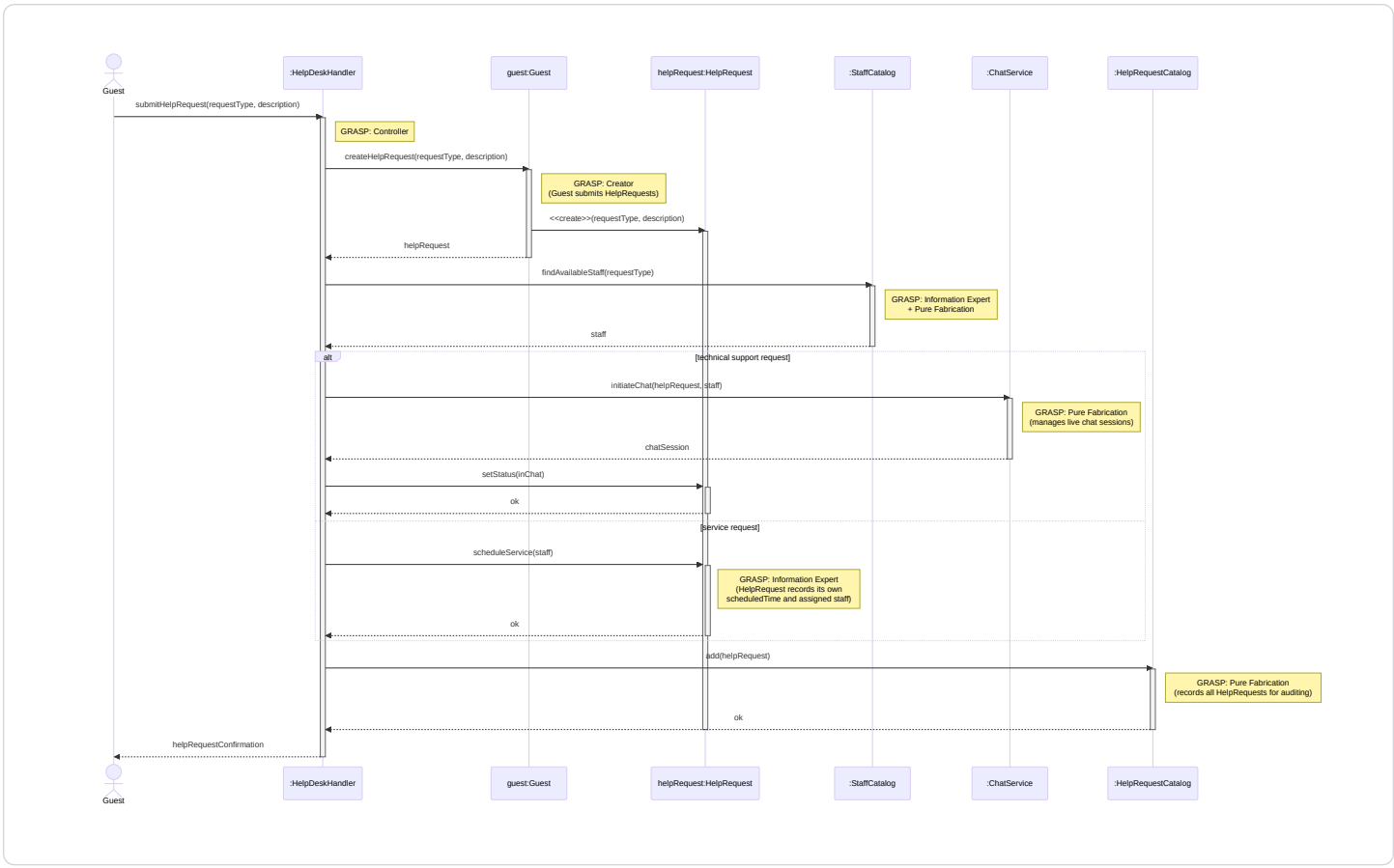


Operation Contract

Operation	`submitHelpRequest(requestType: String, description: String)`
Cross References	Use Case: HelpDesk
Preconditions	1. Guest is logged in 2. A staff member is on standby 3. Hotel system is accessible
Postconditions	1. A new HelpRequest was created and associated with the guest's account and room number 2. If technical help: a live chat session was initiated between the guest and an available staff member 3. If service request: a ServiceRequest was created, a staff member was assigned, and a service time was scheduled 4. Help request was logged with the guest's room number and account

Design Sequence Diagram

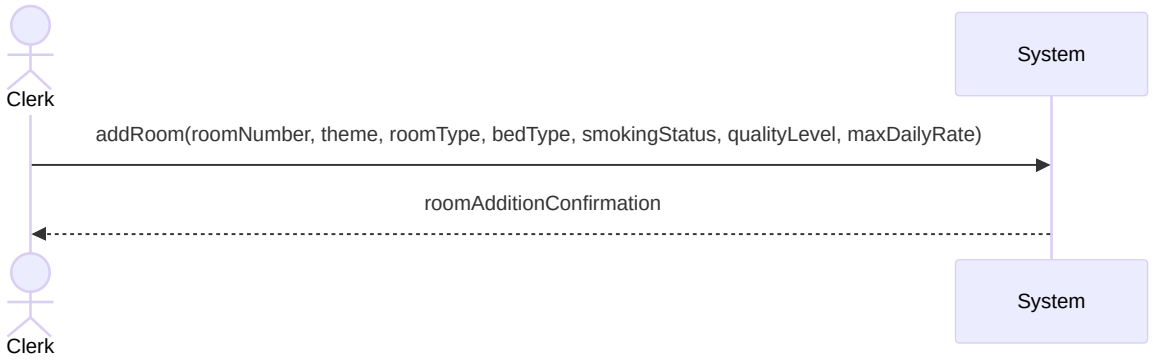
Pattern	Applied To	Rationale
Controller	`:HelpDeskHandler`	Use-case controller; receives the `submitHelpRequest` system operation
Creator	`guest:Guest`	Domain model shows `Guest "1"--"*" HelpRequest : submits` ; Guest aggregates HelpRequests
Information Expert + Pure Fabrication	`:StaffCatalog`	Holds all Staff data and knows who is currently available; no direct domain class
Information Expert	`helpRequest:HelpRequest`	Manages its own `status` and `scheduledTime` attributes
Pure Fabrication	`:ChatService`	Handles live chat session orchestration; no domain counterpart
Pure Fabrication	`:HelpRequestCatalog`	Records and persists all HelpRequests for auditing and lookup


Guest
:HelpDeskHandler
guest:Guest
helpRequest:HelpRequest
:StaffCatalog
:ChatService
:HelpRequestCatalog
submitHelpRequest(requestType, description)
GRASP: Controller
createHelpRequest(requestType, description)
GRASP: Creator
(Guest submits HelpRequests)
<<create>>(requestType, description)
helpRequest
findAvailableStaff(requestType)
GRASP: Information Expert
+ Pure Fabrication
staff
alt
[technical support request]
initiateChat(helpRequest, staff)
GRASP: Pure Fabrication
(manages live chat sessions)
chatSession
setStatus(inChat)
ok
[service request]
scheduleService(staff)
GRASP: Information Expert
(HelpRequest records its own
scheduledTime and assigned staff)
ok
add(helpRequest)
GRASP: Pure Fabrication
(records all HelpRequests for auditing)
ok
helpRequestConfirmation
Guest
:HelpDeskHandler
guest:Guest
helpRequest:HelpRequest
:StaffCatalog
:ChatService
:HelpRequestCatalog

Add Room

Use Case Name	**Add Room**
Actor	Hotel Clerk
Author	Jace Yarborough
Preconditions	1. System operational 2. Hotel clerk is logged in
Postconditions	1. New room is added to hotel inventory 2. Room is available for reservations
Main Success Scenario	1. Hotel clerk navigates to room management page 2. Hotel clerk selects "Add New Room" 3. System displays room entry form 4. Hotel clerk enters room details: - Room number - Floor/theme (Nature Retreat, Urban Elegance, Vintage Charm) - Room type (single, double, family, suite, deluxe, standard) - Bed type and quantity (twin, full, queen, king) - Smoking/non-smoking status - Quality level (executive, business, comfort, economy) - Maximum daily rate 5. Hotel clerk submits form 6. System validates all fields 7. System verifies room number is unique 8. System saves room to database 9. System displays success message 10. Hotel clerk returns to room management page
Extensions	[6]a. **Required fields missing** [6]a1 System highlights missing fields [6]a2 System displays error "Please fill in all required fields" [6]a3 Return to step 4 [6]b. **Invalid data format** [6]b1 System displays error "Invalid format for [field name]" [6]b2 Return to step 4

Use Case Name	Add Room
	[7]a. **Duplicate room number** [7]a1 System displays error "Room number already exists" [7]a2 Return to step 4 [8]a. **Database error** [8]a1 System displays error "Unable to add room. Try again" [8]a2 Use case ends
Special Reqs	• Room numbers must follow hotel numbering convention • Maximum daily rate must be positive value • All room additions must be logged

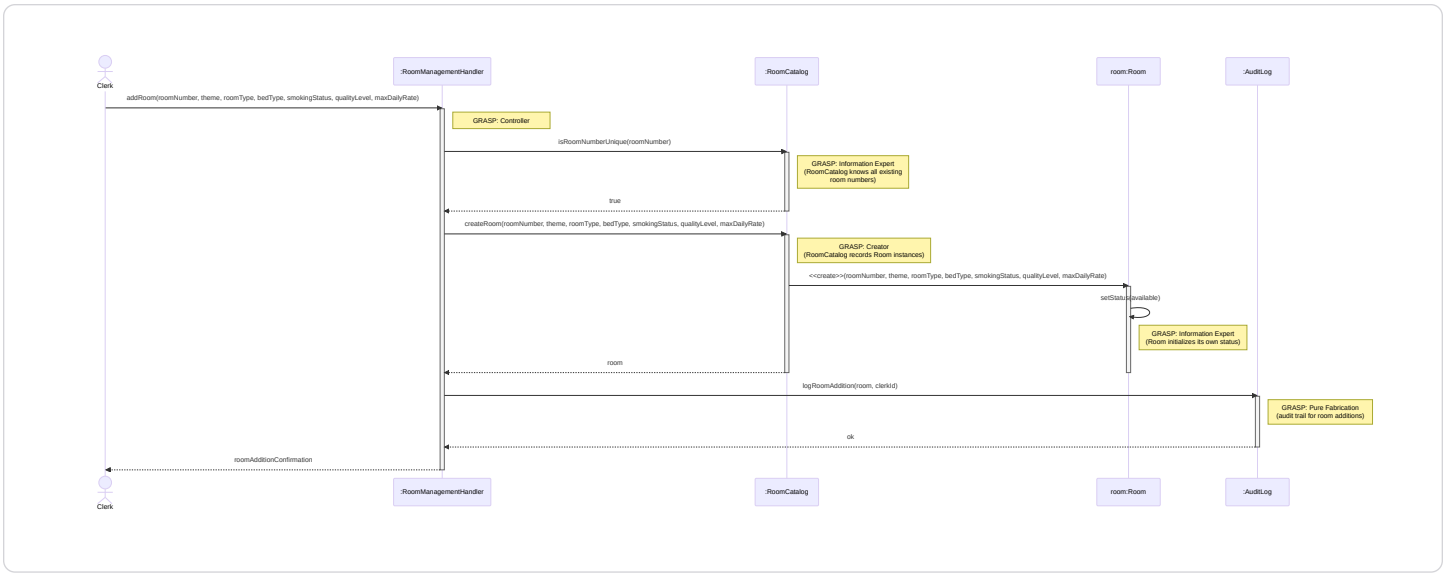


Operation Contract

Operation	`addRoom(roomNumber: String, theme: String, roomType: String, bedType: String, smokingStatus: Boolean, qualityLevel: String, maxDailyRate: Decimal)`
Cross References	Use Case: Add Room
Preconditions	1. Hotel clerk is logged in 2. System is operational
Postconditions	1. A new Room instance was created and saved to the database 2. Room was associated with the hotel inventory 3. Room.status was set to 'available' 4. The room addition was logged

Design Sequence Diagram

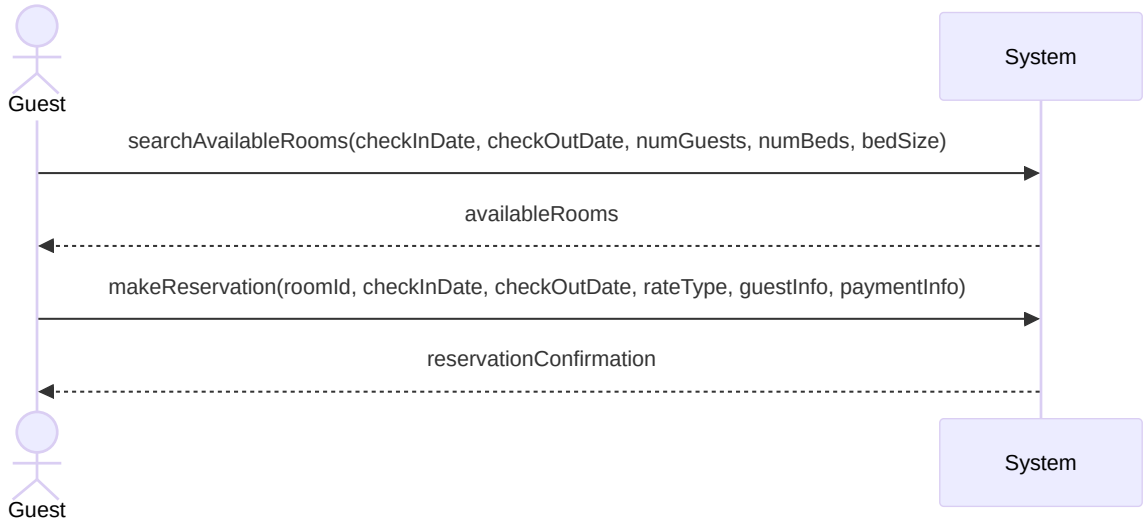
Pattern	Applied To	Rationale
Controller	`:RoomManagementHandler`	Use-case controller; receives the `addRoom` system operation
Information Expert	`:RoomCatalog`	Knows all existing room numbers; can check uniqueness before creation
Creator	`:RoomCatalog`	Records Room instances (GRASP Creator: B records A → B creates A); creates the new Room
Information Expert	`room:Room`	Initializes and manages its own `status` attribute upon creation
Pure Fabrication	`:AuditLog`	Records all room additions for the audit trail; no direct domain counterpart



Make Reservation

Use Case Name	**Make Reservation**
Actor	Hotel Guest
Author	Erick Martinez
Preconditions	1. The hotel system is functional and online 2. The user is logged in to the system 3. Room and reservation data exists in the database 4. The user has searched for available rooms
Postconditions	1. A new reservation is created in the system 2. The selected room is marked as reserved for the specified dates 3. Guest information is recorded (name, address, credit card number, expiration date) 4. Confirmation is displayed to the user
Main Success Scenario	1. The user selects a room from the list of available rooms 2. The user enters the check-in and check-out dates 3. The user selects the rate type (standard, promotion, group, or non-refundable) 4. The user enters or confirms their personal information (name, address) 5. The user enters payment information (credit card number, expiration date) 6. The system validates all input data 7. The system verifies room availability for the selected dates 8. The system calculates the total cost based on quality level and rate type 9. The system creates the reservation and stores it in the database 10. The system displays reservation confirmation with details
Extensions	[3]a. **Corporate guest selected** [3]a1 The user selects their corporation from the list [3]a2 The system records the corporation for billing purposes [3]a3 Continue from step 4 [6]a. **Invalid input data** [6]a1 The system displays an error message indicating the invalid fields

Use Case Name	Make Reservation
	[6]a2 The user corrects the input [6]a3 Continue from step 6 [7]a. **Room is no longer available** [7]a1 The system notifies the user that the room has been booked [7]a2 The system redirects the user to search for available rooms [7]a3 Use case ends
Special Reqs	• Credit card information must be securely stored • Reservation must be atomic (all or nothing)

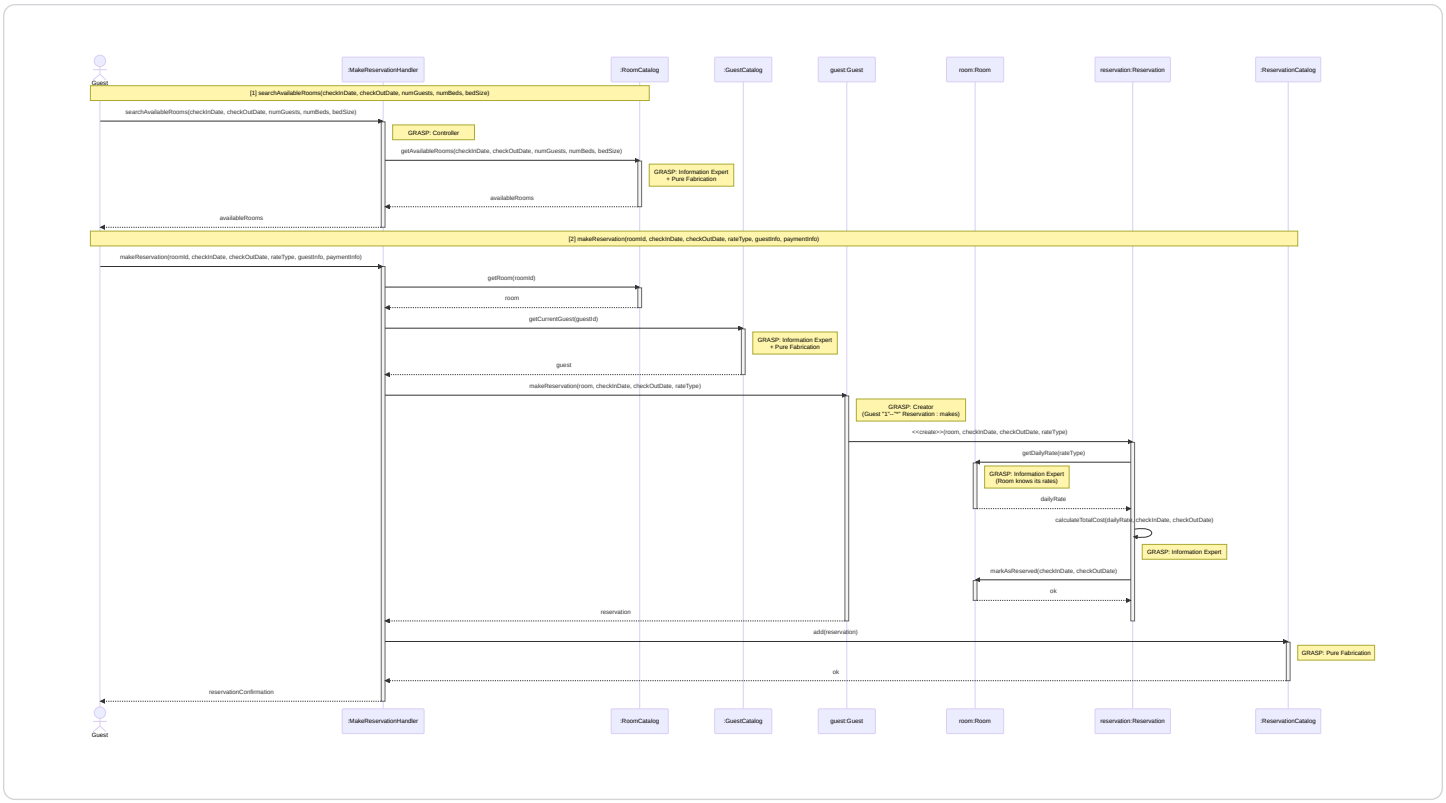


Operation Contract

Operation	`makeReservation(roomId: String, checkInDate: Date, checkOutDate: Date, rateType: String, guestInfo: GuestInfo, paymentInfo: PaymentInfo)`
Cross References	Use Case: Make Reservation
Preconditions	1. Guest is logged in 2. The selected room is available for the requested dates 3. Room and reservation data exist in the database
Postconditions	1. A new Reservation was created in the database 2. Selected Room was marked as reserved for the specified dates 3. Guest information (name, address, credit card number, expiration date) was recorded 4. Reservation.totalCost was calculated based on quality level and rate type

Design Sequence Diagram

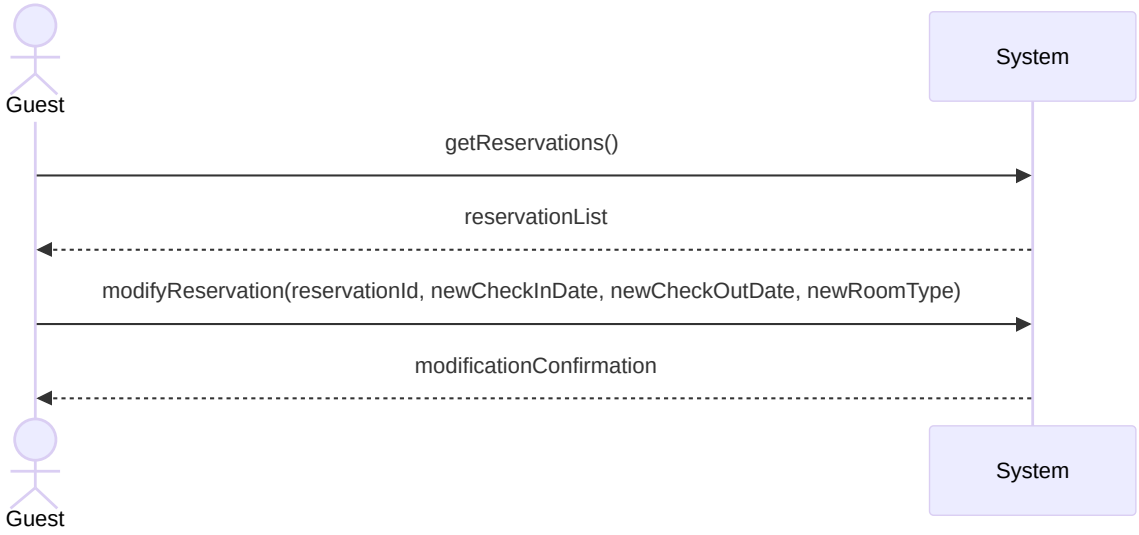
Pattern	Applied To	Rationale
Controller	`:MakeReservationHandler`	Use-case controller; handles both system operations for this use case session
Information Expert + Pure Fabrication	`:RoomCatalog`	Holds all Room data; finds available rooms and retrieves a specific room by ID
Information Expert + Pure Fabrication	`:GuestCatalog`	Retrieves the current guest from the active session
Creator	`guest:Guest`	Domain model shows `Guest "1"--"*" Reservation : makes` ; Guest aggregates Reservations
Information Expert	`room:Room`	Has `maxDailyRate` , `promotionRate` — expert on rate data
Information Expert	`reservation:Reservation`	Calculates its own `totalCost` from the room rate and stay dates
Pure Fabrication	`:ReservationCatalog`	Records and persists all Reservations



Modify Reservation

Use Case Name	**Modify Reservation**
Actor	Hotel Guest
Author	Zain Altaf
Preconditions	1. The hotel guest is logged into the system. 2.The guest has an existing reservation. 3. The reservation has not yet started.
Postconditions	1. The reservation is updated only if modification is permitted. 2. If modification is not permitted, the reservation remains unchanged. 3. Any change in price is recalculated and recorded.
Main Success Scenario	1. The guest selects the option to view their reservations. 2. The system displays the guest's reservations. 3. The guest selects a reservation to modify. 4. The system displays the current reservation details 5.The guest enters the requested changes (e.g., dates or room type). 6. The system checks whether the modification request is more than X hours before the check-in time. 7. The system checks room availability for the requested changes. 8. The system recalculates the reservation cost, if applicable. 9. The system displays the updated reservation details. 10. The guest confirms the modification. 11. The system updates the reservation. 12. The system displays a modification confirmation message.
Extensions	[6]a. **Modification not allowed (within X hours of check-in)** [6]a1 The system determines that the modification request is within X hours of the check-in time. [6]a2 The system displays a message explaining that modifications are not permitted according to the policy.

Use Case Name	Modify Reservation
Special Reqs	• The system must enforce the X-hour modification policy exactly. • Availability checks must be consistent with current reservations. • Price recalculation must follow hotel pricing rules.

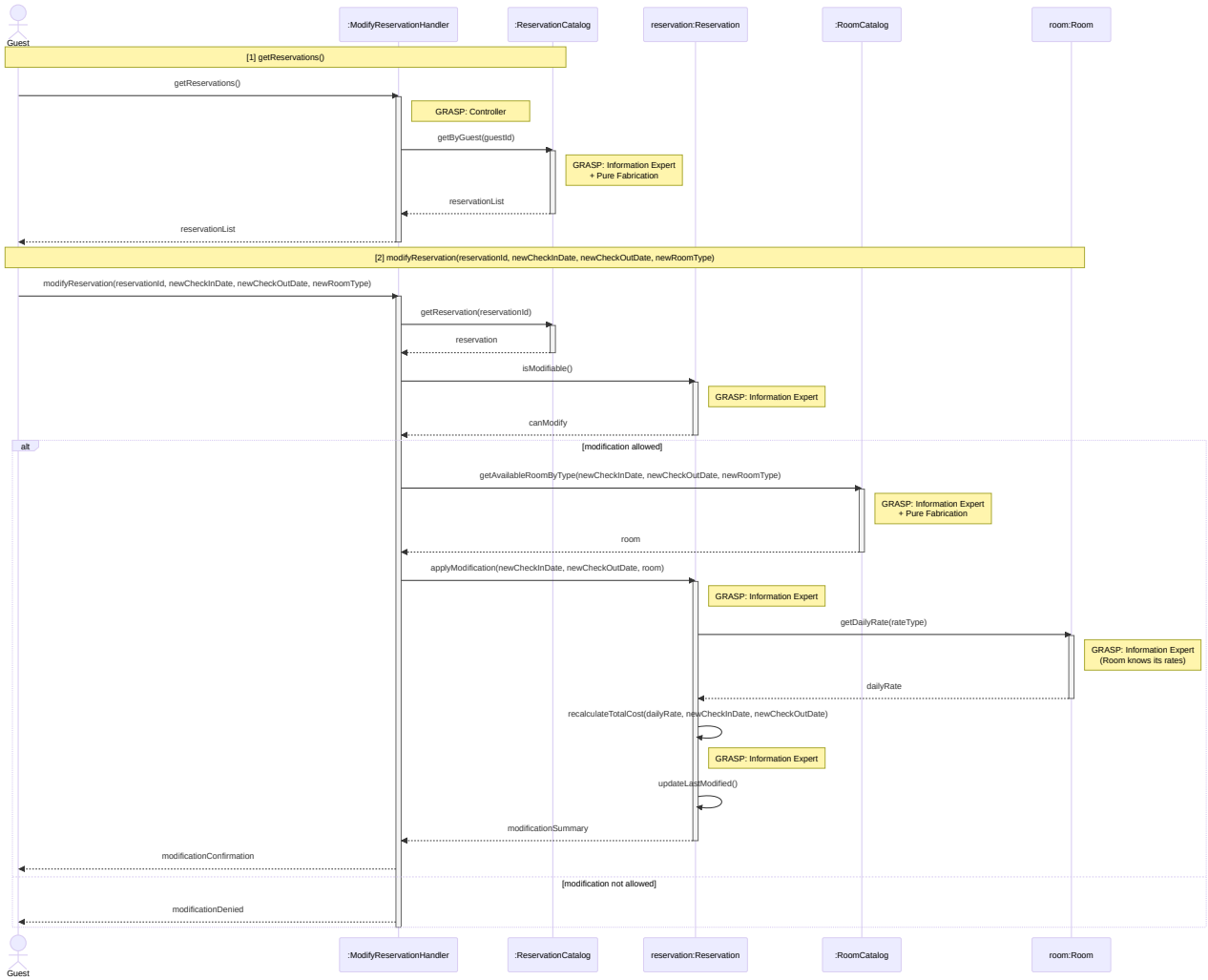


Operation Contract

Operation	`modifyReservation(reservationId: String, newCheckInDate: Date, newCheckOutDate: Date, newRoomType: String)`
Cross References	Use Case: Modify Reservation
Preconditions	1. Guest is logged in 2. Reservation exists and is associated with the guest 3. The modification is requested more than X hours before check-in 4. The reservation has not yet started
Postconditions	1. Reservation.checkInDate and/or Reservation.checkOutDate were updated (if changed) 2. Reservation was associated with the new room type (if changed) 3. Reservation.totalCost was recalculated and updated 4. Reservation.lastModified timestamp was updated

Design Sequence Diagram

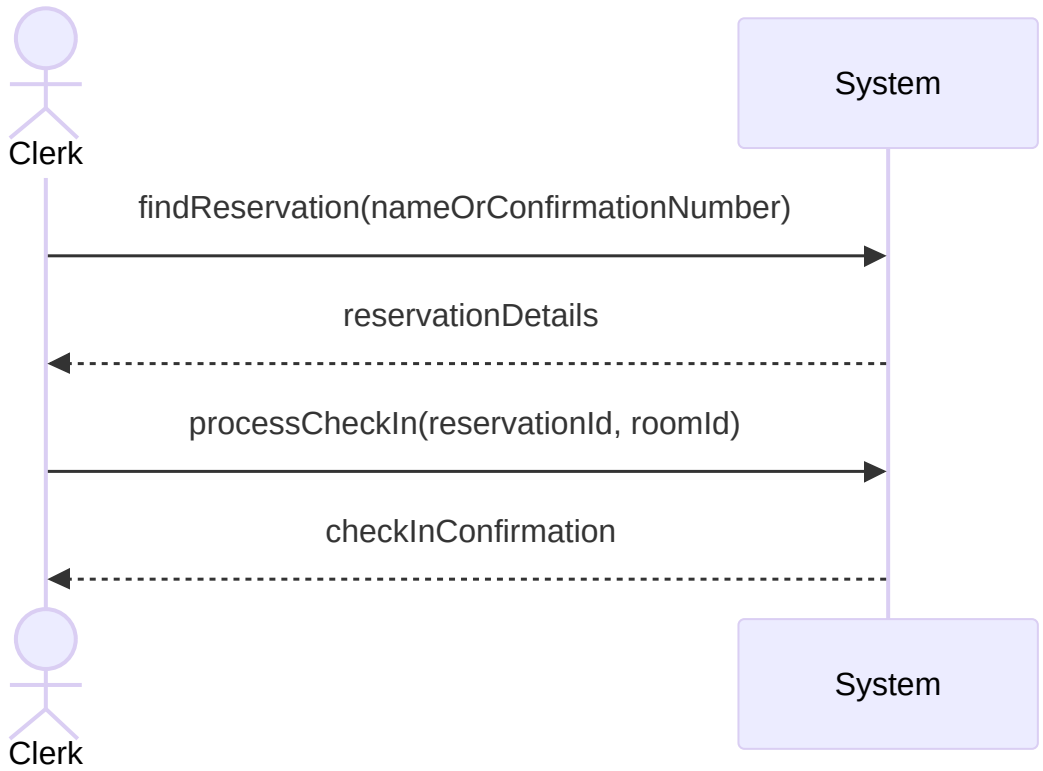
Pattern	Applied To	Rationale
Controller	`:ModifyReservationHandler`	Use-case controller; handles both system operations for this use case session
Information Expert + Pure Fabrication	`:ReservationCatalog`	Holds all Reservation data; retrieves reservations by guest and by ID
Information Expert	`reservation:Reservation`	Has `checkInDate` — enforces the X-hour modification policy; applies its own date/room changes and recalculates `totalCost`
Information Expert	`room:Room`	Has `maxDailyRate`, `promotionRate` — provides rate data for cost recalculation
Information Expert + Pure Fabrication	`:RoomCatalog`	Checks availability for the requested room type and date range



Process Check-In

Use Case Name	**Process Check-In**
Actor	Hotel Clerk
Author	Erick Martinez
Preconditions	1. The hotel system is functional and online 2. The clerk is logged in to the system 3. The guest has an existing reservation for the current date 4. At least one room matching the reservation criteria is available
Postconditions	1. The guest is checked in and assigned to a specific room 2. The room status is updated to occupied 3. The check-in date and time are recorded 4. The guest can access hotel services (including the store)
Main Success Scenario	1. The clerk searches for the guest's reservation by name or confirmation number 2. The system displays the reservation details 3. The clerk verifies the guest's identity 4. The clerk confirms the reservation details with the guest 5. The system displays available rooms matching the reservation 6. The clerk selects a room to assign to the guest 7. The system allocates the room to the guest 8. The system updates the room status to occupied 9. The system records the check-in timestamp 10. The clerk provides the room key/access information to the guest 11. The system displays check-in confirmation
Extensions	[1]a. **Reservation not found** [1]a1 The clerk verifies guest information [1]a2 The clerk offers to create a new reservation (see Make Reservation use case) [1]a3 Use case ends or continues with new reservation [4]a. **Guest requests different room type** [4]a1 The clerk searches for alternative available rooms [4]a2 The system displays available alternatives with price differences

Use Case Name	Process Check-In
	[4]a3 The guest selects a new room type [4]a4 The system updates the reservation with new rate if applicable [4]a5 Continue from step 5 [6]a. **No rooms available matching reservation** [6]a1 The system notifies the clerk of the situation [6]a2 The clerk offers an upgrade or alternative room [6]a3 The guest accepts or declines the alternative [6]a4 If declined, the clerk processes a cancellation with no penalty [6]a5 Use case ends or continues with alternative room
Special Reqs	• Check-in must update room availability in real-time • Guest must have an active reservation to access store purchasing

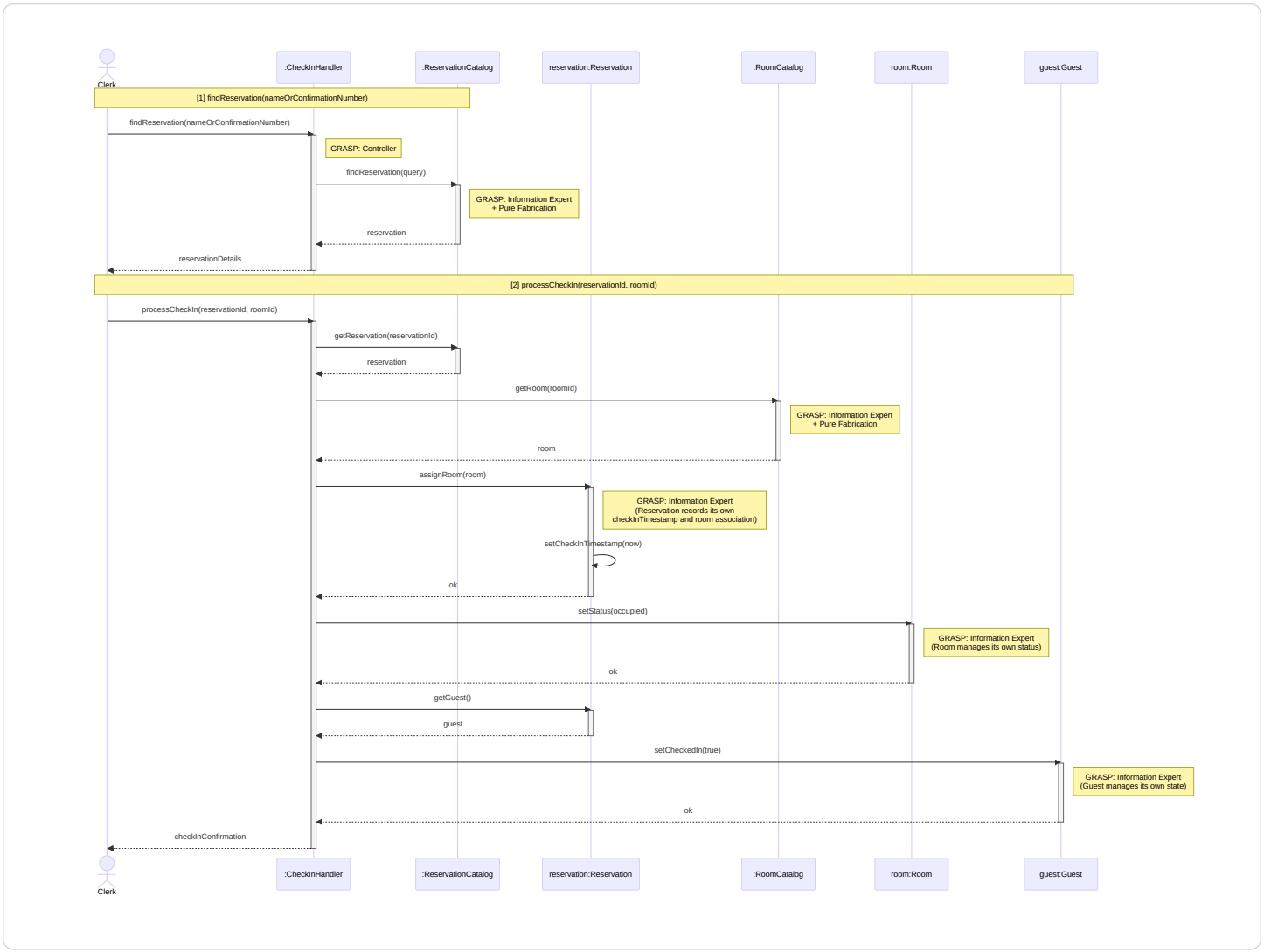


Operation Contract

Operation	`processCheckIn(reservationId: String, roomId: String)`
Cross References	Use Case: Process Check-In
Preconditions	1. Hotel clerk is logged in 2. Guest has a reservation for the current date 3. The specified room is available and matches the reservation criteria
Postconditions	1. Room.status was changed to 'occupied' 2. Reservation.checkInTimestamp was recorded 3. Reservation was associated with the specific assigned Room 4. Guest.checkedIn was set to true

Design Sequence Diagram

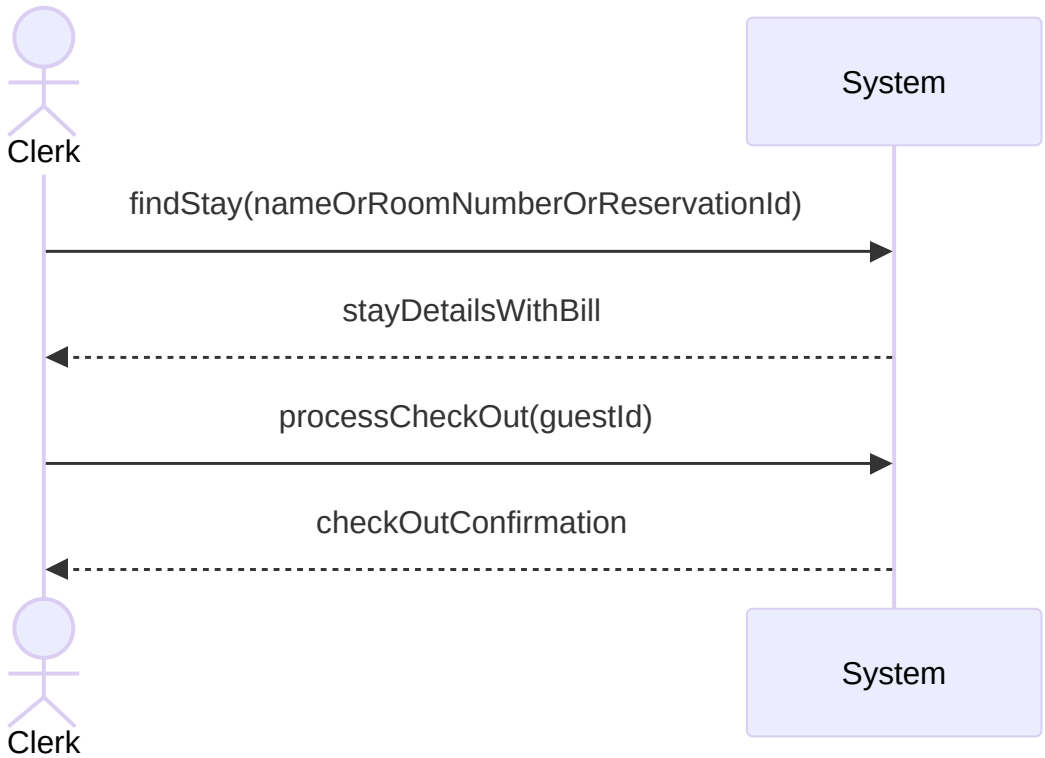
Pattern	**Applied To**	**Rationale**
Controller	`:CheckInHandler`	Use-case controller; handles both system operations for this use case session
Information Expert + Pure Fabrication	`:ReservationCatalog`	Holds all Reservation data; finds reservations by name or confirmation number
Information Expert + Pure Fabrication	`:RoomCatalog`	Holds all Room data; looks up a specific room by ID
Information Expert	`reservation:Reservation`	Records its own `checkInTimestamp` and updates its room association
Information Expert	`room:Room`	Manages its own `status` attribute
Information Expert	`guest:Guest`	Manages its own `checkedIn` flag



Process Check-Out

Use Case Name	**Process Check-Out**
Actor	Hotel Clerk
Author	[Aaron]
Preconditions	1. The hotel system is functional and online 2. The clerk is logged in to the system 3. The guest has been checked in and is currently occupying a room 4. The guest's room and stay details exist in the database
Postconditions	1. The guest is checked out and the room is released 2. The room status is updated to available (or cleaning/maintenance as configured) 3. The check-out date and time are recorded 4. The final bill is calculated and recorded 5. Any outstanding balance or payment confirmation is documented
Main Success Scenario	1. The clerk searches for the guest by name, room number, or reservation ID 2. The system displays the guest's current stay and room assignment 3. The clerk confirms the guest's identity and intent to check out 4. The system calculates the final bill (room charges, minibar, store purchases, incidentals) 5. The system displays the itemized bill to the clerk and guest 6. The guest pays any outstanding balance (or confirms prior payment) 7. The clerk confirms check-out in the system 8. The system updates the room status to available 9. The system records the check-out timestamp 10. The system displays a check-out confirmation and receipt (if requested) 11. The clerk provides the receipt or invoice to the guest
Extensions	[1]a. **Guest or room not found** [1]a1 The system displays a message that no matching stay was found [1]a2 The clerk verifies room number or guest name [1]a3 Return to step 1 or use case ends [6]a. **Payment declined or insufficient**

Use Case Name	Process Check-Out
	[6]a1 The system displays payment failure message [6]a2 The clerk requests alternative payment or arranges follow-up [6]a3 Return to step 6 or use case ends with balance documented [8]a. **System cannot update room status** [8]a1 The system displays an error and logs the failure [8]a2 The clerk retries or escalates; check-out may be completed manually and room status updated later
Special Reqs	● Check-out must update room availability in real-time for Search Available Room ● Final bill must include all room charges and any store or incidental charges linked to the stay ● Check-out time and payment status must be logged for auditing

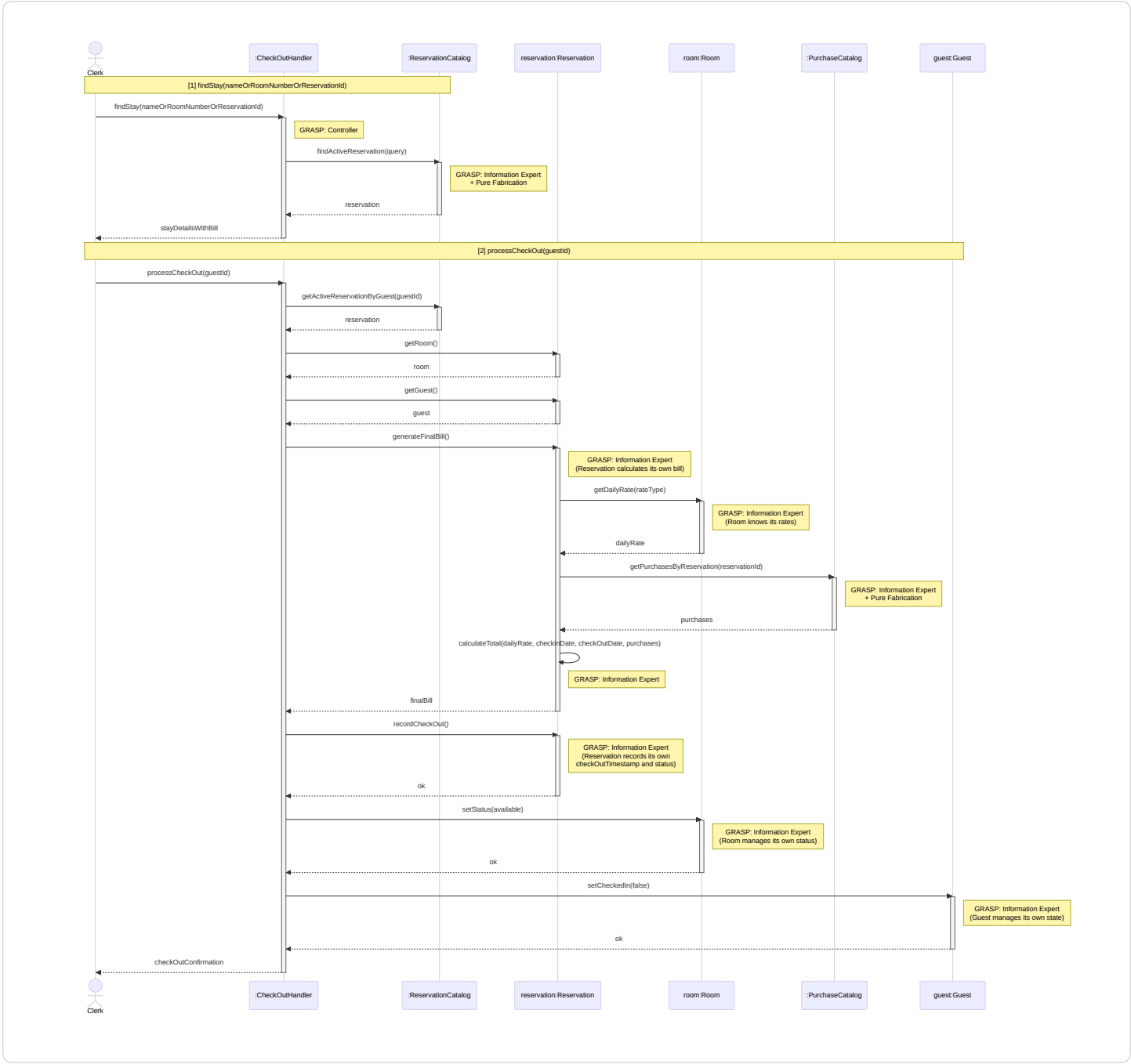


Operation Contract

Operation	`processCheckOut(guestId: String)`
Cross References	Use Case: Process Check-Out
Preconditions	1. Hotel clerk is logged in 2. Guest is currently checked in and occupying a room 3. Guest's stay details exist in the database
Postconditions	1. Room.status was changed to 'available' 2. Stay.checkOutTimestamp was recorded 3. Final bill was calculated and recorded (room charges, store purchases, and incidentals) 4. Guest.checkedIn was set to false 5. Payment status was documented and logged

Design Sequence Diagram

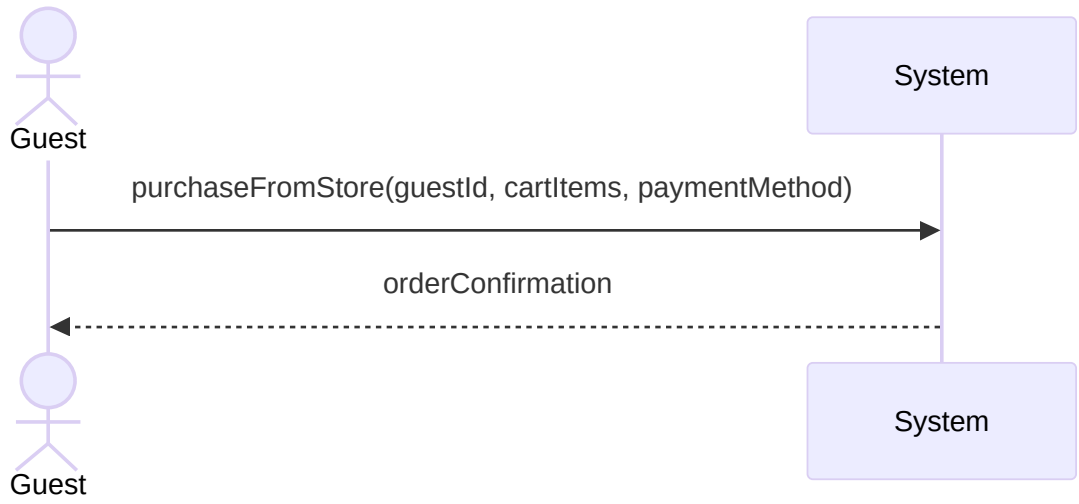
Pattern	Applied To	Rationale
Controller	`:CheckOutHandler`	Use-case controller; receives both system operations for this use case session
Information Expert + Pure Fabrication	`:ReservationCatalog`	Holds all Reservation data; finds active stays by guest/room/ID
Information Expert	`reservation:Reservation`	Has `rateType`, `checkInDate`, `checkOutDate`, `totalCost` — calculates its own final bill and records its own check-out state
Information Expert	`room:Room`	Has `maxDailyRate`, `promotionRate` — expert on rate data used in billing
Information Expert + Pure Fabrication	`:PurchaseCatalog`	Records all store/incidental purchases linked to a reservation; no direct domain class
Information Expert	`guest:Guest`	Manages its own `checkedIn` flag



Purchase from Store

Use Case Name	**Purchase from Store**
Actor	Guest
Author	[Aaron]
Preconditions	1. The guest is logged into the hotel system 2. The guest has browsed the product catalog and identified items to purchase 3. The guest is checked in (or the system allows store purchases for registered guests as per policy) 4. Products are available in inventory
Postconditions	1. The selected products are recorded as purchased and associated with the guest (and room/stay if checked in) 2. Inventory for purchased items is updated 3. Payment is recorded and the guest receives confirmation 4. Charges are applied to the room bill (if checked in) or paid at time of purchase
Main Success Scenario	1. The guest navigates to the Store from the main dashboard 2. The guest adds one or more products to the cart (product, quantity, size/variant if applicable) 3. The guest views the cart and adjusts quantities or removes items if desired 4. The guest proceeds to checkout 5. The system displays order summary (items, quantities, prices, total) and confirms guest/room for billing 6. The guest confirms payment method (charge to room or enter card) 7. The system validates payment and inventory availability 8. The system records the sale and updates inventory 9. The system applies charges to the room bill or completes the payment transaction 10. The system displays order confirmation and, if applicable, delivery or pickup details 11. The guest acknowledges the confirmation

Use Case Name	Purchase from Store
Extensions	[2]a. **Product no longer available** [2]a1 The system notifies the guest that the item is out of stock [2]a2 The guest removes the item or selects an alternative [2]a3 Continue from step 3 [7]a. **Payment failed** [7]a1 The system displays payment error message [7]a2 The guest corrects payment details or chooses another method [7]a3 Return to step 6 [7]b. **Guest not checked in and no payment method** [7]b1 The system prompts for a valid payment method to complete purchase [7]b2 Use case ends if guest cannot provide payment
Special Reqs	● Store purchases for checked-in guests must be chargeable to the room and visible on the final bill (Process Check-Out) ● Inventory must be decremented atomically with the sale ● Payment and order details must be stored securely and logged for auditing

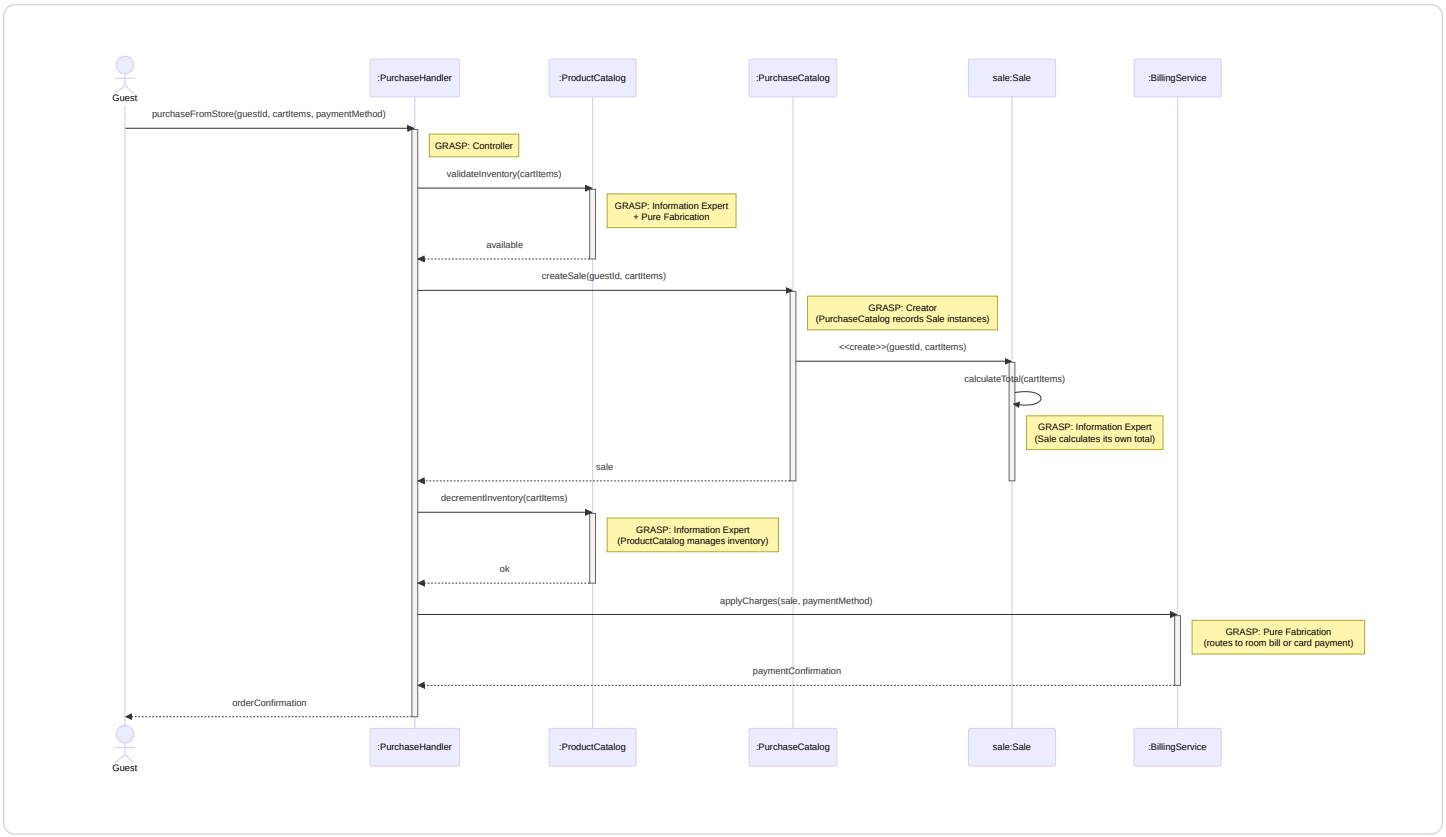


Operation Contract

Operation	`purchaseFromStore(guestId: String, cartItems: List<CartItem>, paymentMethod: PaymentMethod)`
Cross References	Use Case: Purchase from Store
Preconditions	1. Guest is logged in 2. All items in the cart are available in inventory 3. Guest is checked in or has a valid payment method on file
Postconditions	1. A new Sale was created and associated with the guest and current stay 2. Inventory quantity was decremented for each purchased item 3. Charges were applied to the guest's room bill (if checked in) or a payment transaction was completed 4. Order confirmation was generated and associated with the sale

Design Sequence Diagram

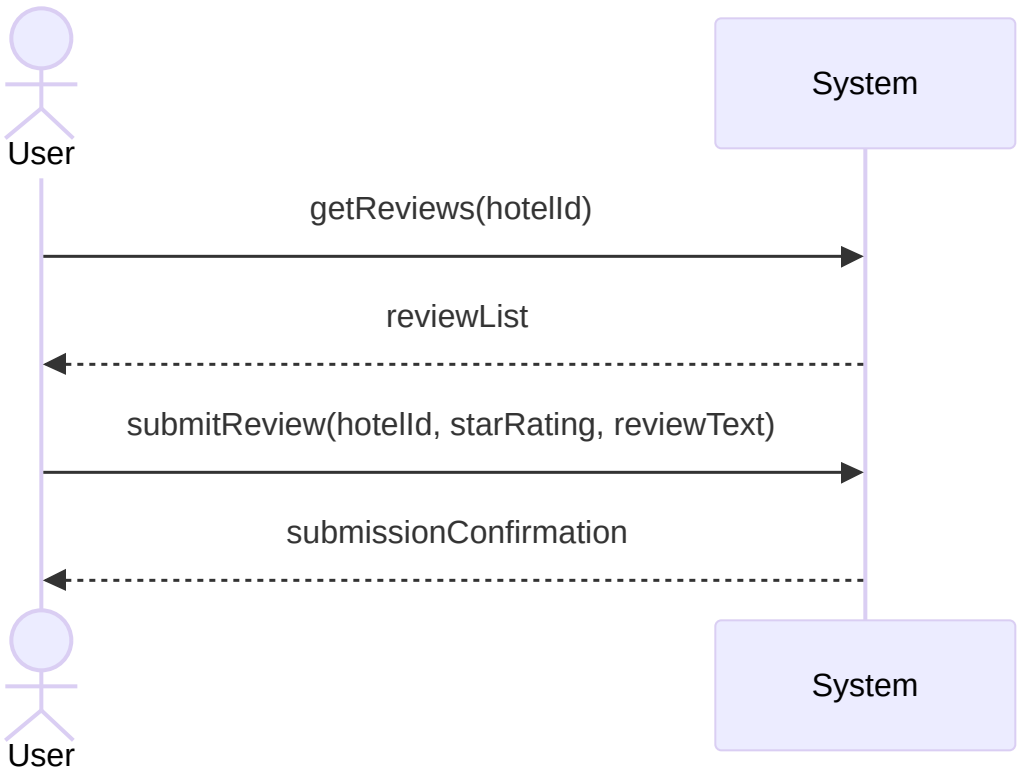
Pattern	Applied To	Rationale
Controller	`:PurchaseHandler`	Use-case controller; receives the `purchaseFromStore` system operation
Information Expert + Pure Fabrication	`:ProductCatalog`	Holds all Product and inventory data; validates stock and decrements quantities
Creator + Pure Fabrication	`:PurchaseCatalog`	Records Sale instances (GRASP Creator: B records A → B creates A)
Information Expert	`sale:Sale`	Calculates its own total from the cart items
Pure Fabrication	`:BillingService`	Routes charges to the room bill or processes a card payment; no domain counterpart



Leaving and / or Viewing a Review

Use Case Name	**Leaving and / or Viewing a Review**
Actor	Previous Hotel Guest / Potential Guest
Author	James Bagwell
Preconditions	1. The user is on the review / details page. 2. To leave a review, the user must be logged into their account and theyu must have reserved AND checked-in to a room previously.
Postconditions	1. The new review is saved to the database and displayed on the hotel page. 2. The hotel's average star rating is updated.
Main Success Scenario	1. The User selects the "Reviews" tab on the hotel detail page. 2. The System displays a list of existing reviews and the current average rating. 3. The User clicks the button to leave a review. 4. The User leaves a star rating (1–5) and writes their review in the text field. 5. The User clicks the submits the review. 6. The System validates the review and indicates that the review was successfully left.
Extensions	[3]a. **User is not logged in** [3]a1. The System prompts the user to log in or sign up. [3]a2. Upon successful login, the system redirects the user back to the review form. [5]b. **Incomplete Review Form** [5]b1. The System highlights the missing fields (for example, if the star rating is left blank). [5]b2. The System prevents submission until all required fields are filled.

Use Case Name	Leaving and / or Viewing a Review
Special Reqs	● The system must filter for profanity or restricted content before publishing. ● The user must be able to filter how many reviews they want to see (For example, show 10 reviews). ● Users should be able to sort reviews by "Most Recent" or "Highest Rated."

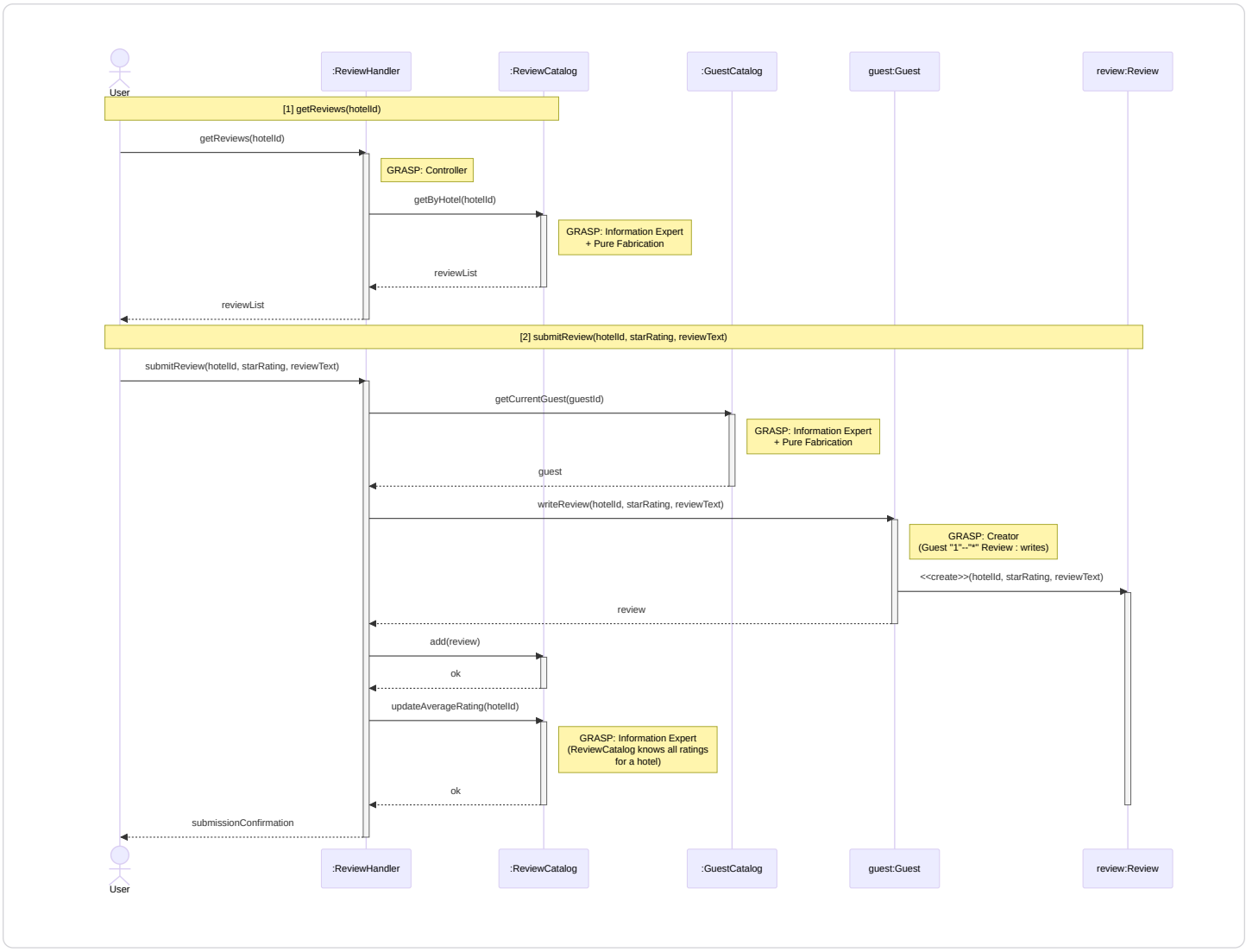


Operation Contract

Operation	`submitReview(hotelId: String, starRating: Integer, reviewText: String)`
Cross References	Use Case: Leaving and / or Viewing a Review
Preconditions	1. User is logged in 2. User has a prior completed stay (checked in) at the hotel
Postconditions	1. A new Review was created and associated with the hotel 2. Review was associated with the Guest account 3. Hotel.averageStarRating was recalculated and updated 4. Review was stored in the database and made visible on the hotel page

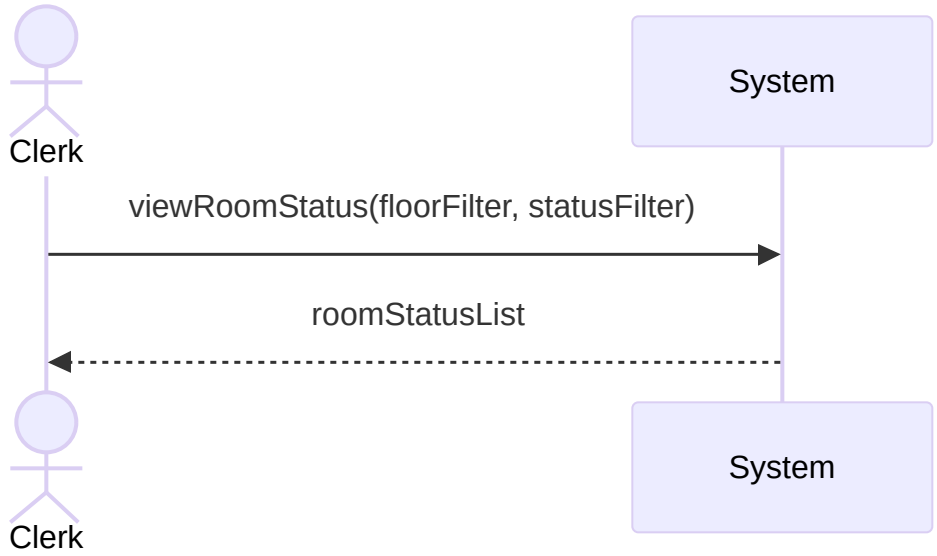
Design Sequence Diagram

Pattern	Applied To	Rationale
Controller	`:ReviewHandler`	Use-case controller; handles both system operations for this use case session
Information Expert + Pure Fabrication	`:ReviewCatalog`	Holds all Review data; retrieves reviews by hotel and recalculates average rating
Information Expert + Pure Fabrication	`:GuestCatalog`	Retrieves the current guest from the active session
Creator	`guest:Guest`	Domain model shows `Guest "1"--"*" Review : writes` ; Guest aggregates Reviews



View Room Status

Use Case Name	**View Room Status**
Actor	Hotel Clerk
Author	Jonathan Deiss
Preconditions	1. Hotel clerk is logged into the system 2. Room data exists in the database
Postconditions	1. The clerk has viewed the current status of all rooms 2. No data is modified
Main Success Scenario	1. The clerk navigates to the room status dashboard 2. The system retrieves all rooms from the database 3. The system displays each room with its room number, floor/theme, and current status (available, reserved, or occupied) 4. The clerk optionally filters rooms by floor or status 5. The system updates the displayed list based on the applied filter 6. The system displays a summary count of rooms by status
Extensions	[2]a. **No rooms in system** [2]a1 The system displays a message indicating no rooms have been added yet [2]a2 Use case ends [5]a. **No rooms match filter** [5]a1 The system displays a message indicating no rooms match the selected criteria
Special Reqs	● Room statuses must reflect real-time reservation and check-in data ● The dashboard must display a summary count of rooms by status

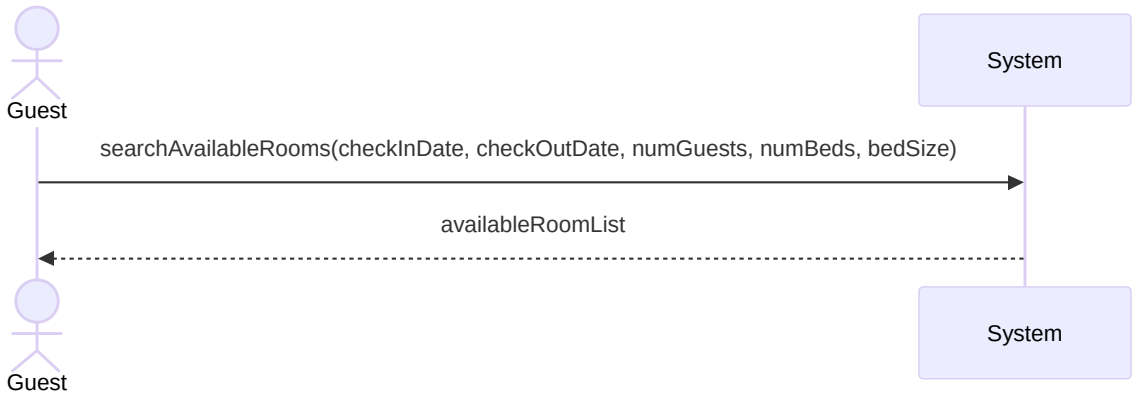


Operation Contract

Operation	`viewRoomStatus(floorFilter: String, statusFilter: String)`
Cross References	Use Case: View Room Status
Preconditions	1. Hotel clerk is logged in 2. Room data exists in the database
Postconditions	1. No domain model state was changed (read-only operation) 2. A list of rooms with their current status was retrieved and displayed, filtered by the given criteria if provided

Search Available Room

Use Case Name	**Search Available Room**
Actor	Hotel Guest
Author	James Bagwell
Preconditions	1. The hotel system is functional and online 2. Room and reservation data exists in the database
Postconditions	1. Available rooms are displayed to the user 2. Data is not modified
Main Success Scenario	1. The user selects the search option 2. The user enters their search criteria such as check in / out date, number of guests, number of beds, bed size, etc. 3. System validates input 4. System searches for rooms that match user criteria, if available 5. System displays list of available rooms that match user criteria, if available
Extensions	
Special Reqs	

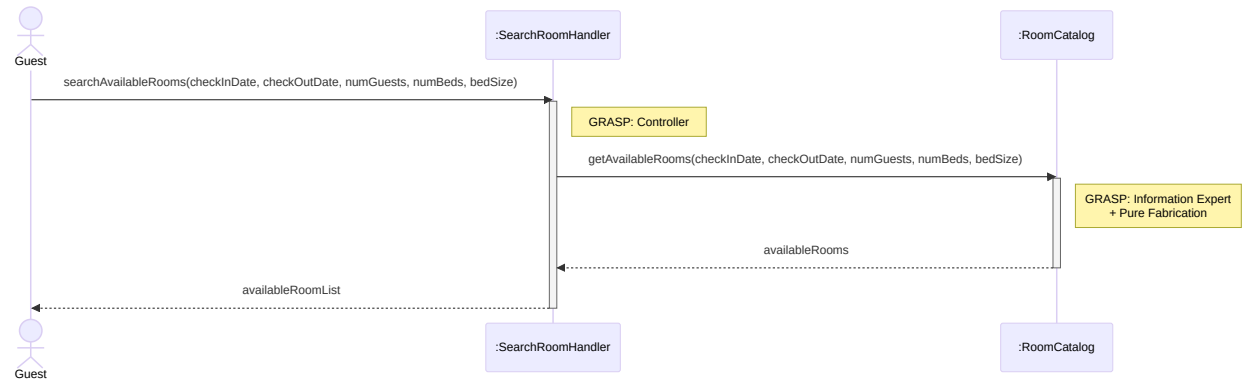


Operation Contract

Operation	`searchAvailableRooms(checkInDate: Date, checkOutDate: Date, numGuests: Integer, numBeds: Integer, bedSize: String)`
Cross References	Use Case: Search Available Room
Preconditions	1. Hotel system is functional and online 2. Room and reservation data exist in the database
Postconditions	1. No domain model state was changed (read-only operation) 2. A list of rooms matching the search criteria was retrieved and displayed

Design Sequence Diagram

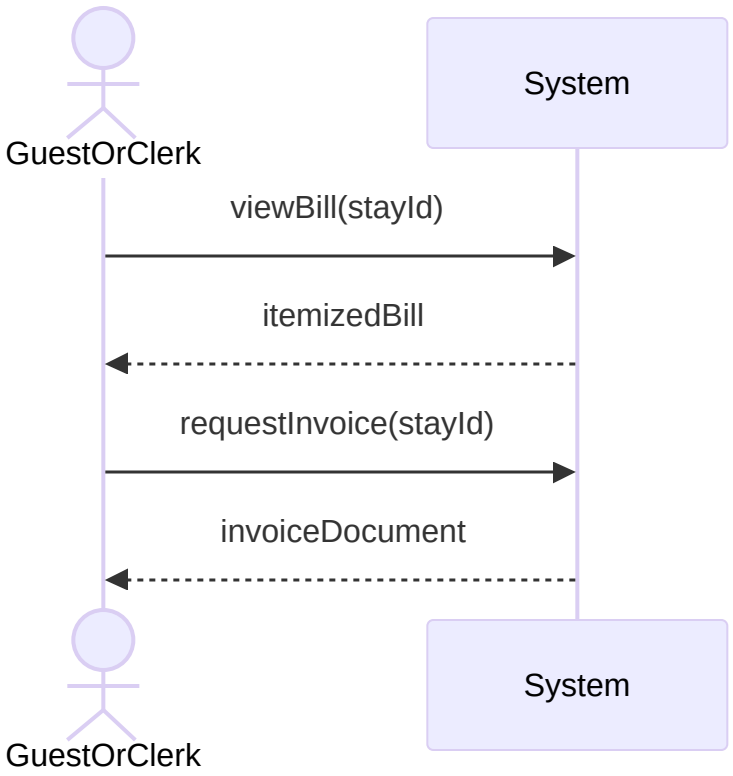
Pattern	Applied To	Rationale
Controller	`:SearchRoomHandler`	Use-case controller; receives the `searchAvailableRooms` system operation
Information Expert + Pure Fabrication	`:RoomCatalog`	Holds all Room and Reservation data; filters rooms by availability and all search criteria



View or Request Bill

Use Case Name	**View or Request Bill**
Actor	Hotel Guest or Hotel Clerk
Author	[Aaron]
Preconditions	1. The hotel system is functional and online 2. The actor is logged in to the system 3. There exists a stay, reservation, or set of charges associated with the guest (current or past)
Postconditions	1. The actor has viewed the current bill or a historical bill for the specified stay 2. If requested, an invoice or receipt is generated and made available (e.g., download or email) 3. The request is logged for auditing where applicable
Main Success Scenario	1. The guest or clerk navigates to billing, "My Stay," or reservation details 2. The guest selects their stay (or the clerk selects the guest and stay) 3. The system retrieves all charges for that stay (room, rate, taxes, minibar, store purchases, incidentals) 4. The system displays an itemized bill with line items, dates, and totals 5. The actor reviews the bill 6. If the actor requests an invoice or receipt, they select "Request Invoice" or "Download Receipt" 7. The system generates the document (PDF or formatted print) with hotel branding and bill details 8. The system makes the document available for download or sends it to the guest's email 9. The system displays confirmation that the bill was viewed and, if applicable, that the invoice was sent
Extensions	[2]a. **No stay or reservation found** [2]a1 The system displays a message that no billable stay was found for this guest [2]a2 Use case ends [6]a. **Invoice request for past stay**

Use Case Name	View or Request Bill
	[6]a1 The system allows invoice generation for completed stays within the retention period [6]a2 Continue from step 7 [6]b. **Invoice request not allowed (e.g., stay in progress and policy requires check-out first)** [6]b1 The system displays "Final invoice available at check-out" or similar [6]b2 Use case ends with bill view only
Special Reqs	● Bill must reflect all charges from room, store (Purchase from Store), and incidentals in real time ● Invoice/receipt must include all legally required information (hotel name, stay dates, tax breakdown, etc.) ● Guest may only view or request bills for their own stays; clerks may view bills for any guest as authorized

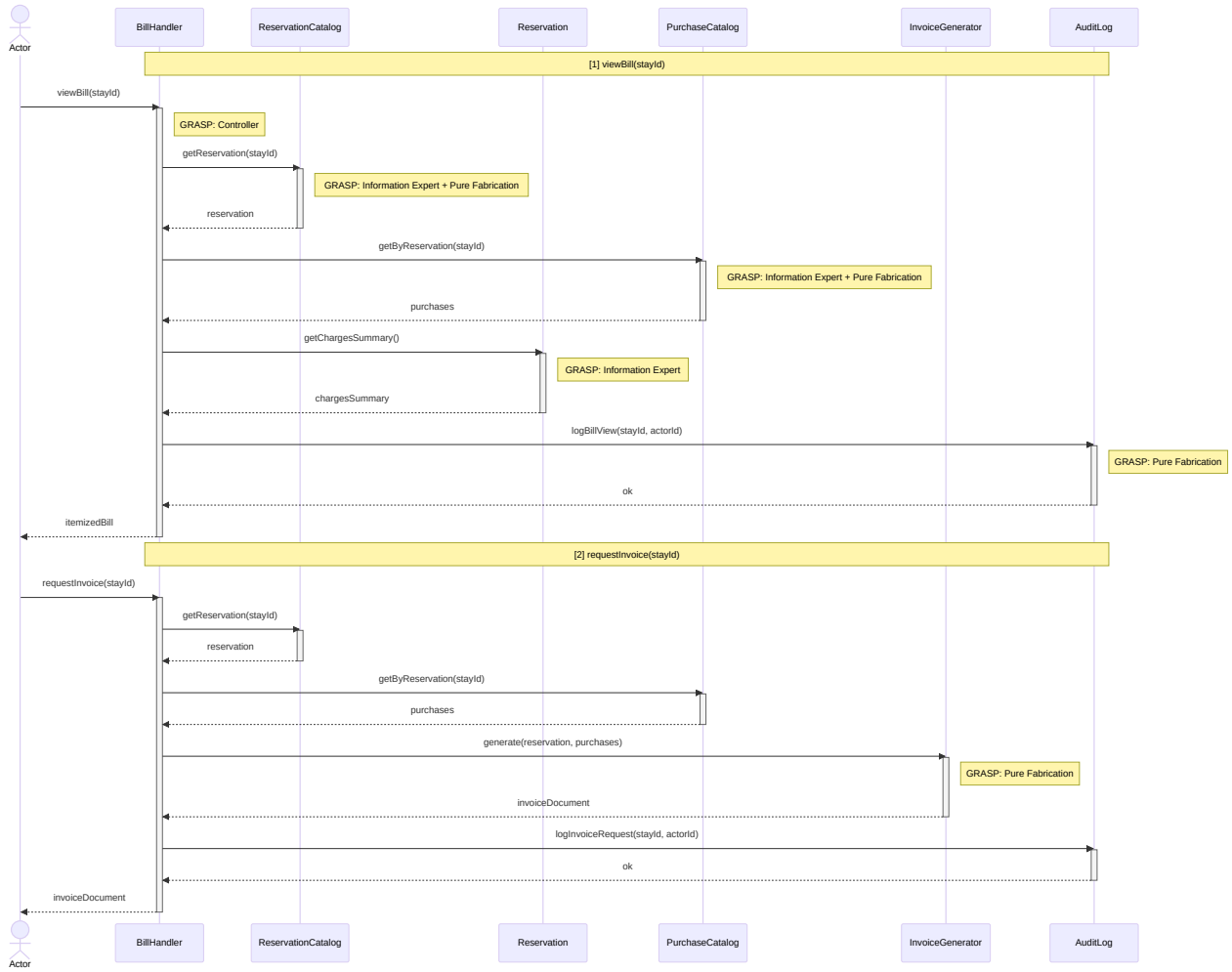


Operation Contract

Operation	`viewBill(stayId: String)` / `requestInvoice(stayId: String)`
Cross References	Use Case: View or Request Bill
Preconditions	1. Actor is logged in 2. A stay, reservation, or set of charges exists for the guest in the system
Postconditions	1. Bill view event was logged for the stay 2. An invoice document was generated containing all line items, dates, and totals (if requested) 3. Invoice was made available for download or sent to the guest's email (if requested) 4. Invoice request was logged (if applicable)

Design Sequence Diagram

Pattern	Applied To	Rationale
Controller	`:BillHandler`	Use-case controller; handles both system operations for this use case session
Information Expert + Pure Fabrication	`:ReservationCatalog`	Holds all Reservation data; retrieves the stay by ID
Information Expert	`reservation:Reservation`	Knows its own charges (room rate, dates, totalCost)
Information Expert + Pure Fabrication	`:PurchaseCatalog`	Holds all store purchase records linked to a stay
Pure Fabrication	`:InvoiceGenerator`	Generates the formatted invoice document; no domain counterpart
Pure Fabrication	`:AuditLog`	Logs bill views and invoice requests


Actor
BillHandler
ReservationCatalog
Reservation
PurchaseCatalog
InvoiceGenerator
AuditLog
[1] viewBill(stayId)
viewBill(stayId)
GRASP: Controller
getReservation(stayId)
GRASP: Information Expert + Pure Fabrication
reservation
getByReservation(stayId)
GRASP: Information Expert + Pure Fabrication
purchases
getChargesSummary()
GRASP: Information Expert
chargesSummary
logBillView(stayId, actorId)
GRASP: Pure Fabrication
ok
itemizedBill
[2] requestInvoice(stayId)
requestInvoice(stayId)
getReservation(stayId)
reservation
getByReservation(stayId)
purchases
generate(reservation, purchases)
GRASP: Pure Fabrication
invoiceDocument
logInvoiceRequest(stayId, actorId)
ok
invoiceDocument
Actor
BillHandler
ReservationCatalog
Reservation
PurchaseCatalog
InvoiceGenerator
AuditLog