Service Booking & Consulting Platform Phase 1

Salik Hassan -

Yasamin KheirKhahan -

Ali Sheikh Rabiei – 218475095

GitHubLink

(DRAFT!)

1. System overview

This system is a Service Booking & Consulting Platform that connects Clients with Consultants who offer consulting services. Clients can browse services, request bookings, cancel bookings, and view bookings and payment history. Consultants manage their availability and accept/reject booking requests, and they can complete bookings after the session is delivered. An Admin approves consultant registrations and configures system wide policies such as cancellation rules, pricing strategies, notification settings, and refund policies. Payment processing is simulated for now and is used to demonstrate correct workflow and design patterns.

2. Main domain classes (entities)

User (abstract)

Purpose: Base type for all system users.
Fields: id:String, name:String, email:String
Methods: getId(), getName(), getEmail()
Notes: Shared identity information is centralized here to avoid duplication.

Client (extends User)

Purpose: A user who browses services and requests bookings (UC1–UC7).
Fields/Methods: No extra fields are required in Phase 1. Client behavior is mainly expressed through service-layer operations (BookingService and PaymentService).

Consultant (extends User)

Purpose: A service provider who offers services, manages availability, and decides on booking requests (UC8–UC10).
Fields:

bio:String — short profile information

registrationStatus:RegistrationStatus — used for admin approval (UC11)
Methods:

listServices(): List<Service> — returns services offered by the consultant

decideBooking(b:Booking, decision:Decision): void — consultant's decision action (the actual state change is coordinated in BookingService)

Admin (extends User)

Purpose: System administrator who approves consultant registrations and configures policies (UC11–UC12).
Methods:

approveConsultantRegistration(consultantId:String, approve:boolean): void — sets consultant registration to APPROVED/REJECTED

setCancellationPolicy(p:CancellationPolicy): void

setRefundPolicy(p:RefundPolicy): void

setNotificationPolicy(p:NotificationPolicy): void

setPricingStrategy(p:PricingStrategy): void
Notes: These policy methods delegate to PolicyManager to ensure system-wide consistency.

RegistrationStatus (enum)

Purpose: Tracks consultant registration review state.
Values: PENDING, APPROVED, REJECTED

Service

Purpose: Represents a consulting service that can be booked (UC1, UC2).
Fields: serviceId:String, title:String, description:String, durationMin:int, price:double, consultant:Consultant
Methods:

getPrice(): double — base price used for payment/pricing

getDurationMin(): int — displayed in service browsing

getConsultant(): Consultant — identifies service owner

TimeSlot

Purpose: Represents a consultant's available time interval for a service.
Fields: start:LocalDateTime, end:LocalDateTime, isAvailable:boolean
Methods:

reserve(): void — marks the slot as no longer available when booked release(): void — makes the slot available again if a booking is rejected/cancelled

Booking

Purpose: Represents a booking request and its lifecycle states (UC2, UC3, UC9, UC10).
Fields:

bookingId:String

client:Client

service:Service

slot:TimeSlot

status:BookingStatus — for display/history

createdAt:LocalDateTime

state:BookingState — State pattern implementation (enforces valid transitions)

Methods (public booking actions):

confirm(): void — applied when consultant accepts; moves booking forward in lifecycle

reject(reason:String): void — applied when consultant rejects; ends booking

cancel(reason:String): void — applied by client/consultant; ends booking

paymentSuccessful(tx:PaymentTransaction): void — applied after simulated payment succeeds; transitions to PAID

complete(): void — consultant marks session completed (only allowed after PAID)

request(): void — optional helper; booking is typically created in REQUESTED state

Internal methods (state maintenance):

setState(s:BookingState): void

setStatus(st:BookingStatus): void

Notes: BookingStatus is used for history and UI output (UC4). BookingState is used to enforce rules such as "cannot complete before paid".

BookingStatus (enum)

Purpose: Required lifecycle states (spec).
Values: REQUESTED, CONFIRMED, PENDING_PAYMENT, PAID, REJECTED, CANCELLED, COMPLETED

3. Service layer classes (backend logic)

ServiceCatalog

Purpose: Provides browsing and lookup of services (UC1).
Fields: services:List<Service>
Methods:

listAllServices(): List<Service> — returns all services

findById(serviceId:String): Service — lookup for booking/payment flows

AvailabilityService

Purpose: Consultant availability management and validation (UC8 and supports UC2).
Methods:

addTimeSlot(consultant:Consultant, slot:TimeSlot): void — adds a new availability slot

removeTimeSlot(consultant:Consultant, slotId:String): void — removes a slot

listAvailableSlots(service:Service): List<TimeSlot> — returns available slots for booking

BookingRepository

Purpose: In-memory storage and retrieval of bookings (supports UC4 + system persistence).
Fields: bookings:List<Booking>
Methods: save(b:Booking), findById(id), findByClient(c), findByConsultant(con)

BookingService

Purpose: Core booking workflow coordinator and rule enforcer (UC2–UC4, UC9–UC10).
Methods:

createBooking(client:Client, service:Service, slot:TimeSlot): Booking — checks availability and creates booking in REQUESTED

confirmBooking(bookingId:String): void — confirms after consultant accept (transitions state)

cancelBooking(bookingId:String, reason:String): boolean — checks CancellationPolicy and cancels if allowed

completeBooking(bookingId:String): void — enforces "must be PAID before COMPLETED"

getBooking(bookingId:String): Booking — returns one booking

getBookingsForClient(clientId:String): List<Booking> — booking history (UC4)

Notes: BookingService coordinates repository access, availability rules, policies, and notifications rather than embedding those concerns inside domain entities.

4. Payment subsystem classes (simulation + history)

PaymentMethod (abstract)

Purpose: Stores payment method details and validates them (UC5, UC6).
Fields: owner:Client, methodId:String
Method: validate(): boolean

Concrete types:

CreditCardMethod, DebitCardMethod, PayPalMethod, BankTransferMethod
Each concrete type supports method-specific validation rules defined in the spec.

PaymentMethodService

Purpose: Saves and manages client payment methods (UC6).
Methods:

addMethod(client, method)

removeMethod(client, methodId)

listMethods(client)

updateMethod(...)

PaymentTransaction

Purpose: Represents a simulated payment attempt and supports payment history (UC7).
Fields: transactionId:String, booking:Booking, amount:double, status:PaymentStatus, methodType:String, timestamp:LocalDateTime

PaymentStatus (enum)

Values: PENDING, SUCCESS, FAILED, REFUNDED

Receipt

Purpose: Confirmation of a successful payment (optional record).
Fields: receiptId:String, amount:double, method:String, timestamp:LocalDateTime
Method: getReceiptId(): String
Relationship: A transaction may have 0..1 receipt; receipt exists only on success.

PaymentService

Purpose: Simulates payment processing and provides payment history (UC5, UC7).
Fields:

handler:PaymentHandler — head of the payment processing chain

transactions:List<PaymentTransaction> — stored history

Methods:

processPayment(booking:Booking, method:PaymentMethod): PaymentTransaction — validates method, simulates delay, generates transaction id, updates booking to PAID when successful, and creates receipt/notification

getPaymentHistory(client:Client): List<PaymentTransaction> — returns transaction history for a client

setMethod(method:PaymentMethod): void — optional helper used by CLI flow

5. Policy and notification subsystem

NotificationService

Purpose: Simulated messaging to users.
Method: notify(user:User, message:String): void

PolicyManager (Singleton)

Purpose: Stores system-wide policy configuration (UC12).
Fields: instance, cancellationPolicy, refundPolicy, notificationPolicy, pricingStrategy
Methods: getInstance() plus getters/setters for each policy.

CancellationPolicy (Strategy interface)

Methods:

canCancel(b:Booking, now:LocalDateTime): boolean

cancellationFee(b:Booking, now:LocalDateTime): double

RefundPolicy (interface)

Method: calculateRefund(tx:PaymentTransaction, now:LocalDateTime): double

NotificationPolicy (interface)

Method: shouldNotify(eventType:String): boolean

PricingStrategy (interface)

Method: calculatePrice(service:Service, booking:Booking): double

6. Design patterns used and where

Singleton: PolicyManager ensures one shared policy configuration across the whole system.

Strategy: CancellationPolicy (and optional interfaces RefundPolicy/NotificationPolicy/PricingStrategy) allow policy swapping without changing service logic.

State: Booking lifecycle is enforced using BookingState + concrete states, preventing invalid transitions (e.g., completion before payment).

Chain of Responsibility: Payment processing uses PaymentHandler and its concrete handlers (validate → process → confirm) to implement the required payment workflow in small, testable steps.