**Hotel Reservation & Review System**

**Mission Specifications**

You are a backend developer at **StaySmart Technologies**, a company developing hotel management platforms.  
You have been assigned to design a **Hotel Reservation, Service & Review Management Web API** using **.NET Core**.  
The API allows guests to book rooms, request extra services (like Spa or Airport Pickup), and leave reviews for hotels they stayed at.

The system must strictly follow **Clean Architecture** principles, using:

* Dependency Injection (DI)
* Generic Repository Pattern
* DTOs (Data Transfer Objects)
* Data Annotations for Validation
* Fluent API for Relationships
* LINQ-based analytical endpoints

**1. Set Up Environment**

Install required NuGet packages:

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Tools
* Swashbuckle.AspNetCore

Configure the database connection in appsettings.json, register DbContext and repositories in Program.cs, and enable Swagger.

**2. Create Models**

**Relationships**

* Hotel ↔ Room → One-to-Many
* Guest ↔ Booking → One-to-Many
* Room ↔ Booking → One-to-Many
* Guest ↔ GuestProfile → One-to-One ✅
* Booking ↔ Service → Many-to-Many ✅
* Guest ↔ Review → One-to-Many
* Hotel ↔ Review → One-to-Many

Each booking can include **multiple services** (like Laundry, Spa, Breakfast), and each service can belong to **multiple bookings**.

**Models**

**Hotel**

* Id: int (auto)
* Name: string (required, unique)
* Location: string (required)
* Description: string (max 500)
* Rating: decimal (computed average of reviews, read-only)

**Room**

* Id: int (auto)
* RoomNumber: string (required)
* HotelId: int (required, FK)
* PricePerNight: decimal (> 0)
* Capacity: int (> 0)
* IsAvailable: bool (default true)

**Guest**

* Id: int (auto)
* Name: string (required, max 100)
* Email: string (required, valid email, unique)
* Phone: string (optional, valid phone)
* Navigation:
  + GuestProfile (1–1)
  + List of Bookings
  + List of Reviews

**GuestProfile**

* Id: int (auto)
* GuestId: int (required, unique, FK)
* NationalId: string (required, unique)
* LoyaltyPoints: int (default 0)
* Address: string (max 300)
* DateOfBirth: DateTime (optional)

**Booking**

* Id: int (auto)
* GuestId: int (required, FK)
* RoomId: int (required, FK)
* CheckIn: DateTime (required)
* CheckOut: DateTime (required)
* TotalCost: decimal (computed)
* Status: string (default: “Active”)
* Navigation:
  + Guest
  + Room
  + List of Services (many-to-many)

**Service**

* Id: int (auto)
* Name: string (required, unique)
* Description: string
* Price: decimal (> 0)
* Navigation:
  + List of Bookings (many-to-many)

**Review**

* Id: int (auto)
* GuestId: int (required, FK)
* HotelId: int (required, FK)
* Rating: int (1–5)
* Comment: string (max 300)
* Date: DateTime (required)

**3. Requirements**

**Controllers:**

* HotelController
* RoomController
* GuestController
* GuestProfileController
* BookingController
* ServiceController
* ReviewController

**Requirements:**

* Full CRUD operations
* Use DTOs for all requests/responses
* Enforce validation via Data Annotations
* Configure all relationships via Fluent API
* Add LINQ-based analytical endpoints

**4. Controller Specifications (Endpoints)**

**1. Guest Controller**

**GET /api/guests**

* Retrieve all guests with their profile, bookings, and reviews.
* Show total amount spent and loyalty points.

**GET /api/guests/spending-summary**

* Group by guest → total amount spent and total stays.

**POST /api/guests**

* Add a new guest (validate unique email).

**PUT /api/guests/{id}**

* Update name, phone, or email.

**DELETE /api/guests/{id}**

* Only if no active bookings.

**2. GuestProfile Controller (One-to-One)**

**GET /api/guestprofiles**

* Retrieve all profiles with related guest names and loyalty points.

**GET /api/guestprofiles/{id}**

* Retrieve a single profile with guest info.

**POST /api/guestprofiles**

* Create a profile for an existing guest.
* Validate:
  + Guest must exist.
  + Guest cannot already have a profile.
  + NationalId must be unique.

**PUT /api/guestprofiles/{id}**

* Update profile info (address, loyalty points, etc.).

**DELETE /api/guestprofiles/{id}**

* Delete only if the guest still exists (cascade or optional).

**3. Hotel Controller**

**GET /api/hotels**

* Retrieve all hotels with rooms and average rating (calculated from reviews).
* Filter by location, minRating, maxRating.
* Order descending by average rating.

**GET /api/hotels/top-rated**

* Return top 5 hotels by average rating (include name, location, rating).

**POST /api/hotels**

* Add a new hotel.
* Validate unique name.

**PUT /api/hotels/{id}**

* Update hotel name, location, or description.
* Prevent duplicates.

**DELETE /api/hotels/{id}**

* Delete only if no active rooms or bookings exist.

**4. Room Controller**

**GET /api/rooms**

* Retrieve all rooms with hotel name.
* Filter by hotelId, isAvailable, or price range.
* Order by price ascending.

**GET /api/rooms/available**

* Return all available rooms grouped by hotel.
* Include average room price per hotel.

**POST /api/rooms**

* Add a room under a hotel.
* Validate: PricePerNight > 0, Capacity > 0.

**PUT /api/rooms/{id}**

* Update price, capacity, or availability.

**DELETE /api/rooms/{id}**

* Remove only if no bookings exist.

**5. Booking Controller**

**POST /api/bookings**

* Add new booking for existing guest and room.
* Validate:
  + CheckOut > CheckIn
  + No overlapping bookings for same room
  + Room is available
* Calculate:
* TotalCost = (CheckOut - CheckIn).Days \* Room.PricePerNight
* Mark room as unavailable.

**POST /api/bookings/{id}/services**

* Add one or more services to an existing booking.
* Each service adds to total cost:  
  TotalCost += Sum(selectedServices.Price)

**GET /api/bookings**

* Retrieve all bookings with room, guest, and hotel info.
* Filter by guestId, status, or date range.

**GET /api/bookings/active**

* List all ongoing bookings (CheckOut >= DateTime.Now).

**GET /api/bookings/revenue-summary**

* Group bookings by hotel and show:
  + Total revenue per hotel
  + Total number of bookings
  + Average booking cost

**PUT /api/bookings/{id}**

* Update dates, recalculate total cost, and revalidate overlaps.

**DELETE /api/bookings/{id}**

* Cancel booking and mark room available again.

**6. Service Controller**

**GET /api/services**

* Retrieve all services with how many bookings used them.
* Order by usage count descending.

**POST /api/services**

* Add new service (name must be unique, price > 0).

**PUT /api/services/{id}**

* Update service details.

**DELETE /api/services/{id}**

* Delete only if not attached to any booking.

**7. Review Controller**

**POST /api/reviews**

* Add review by guest for a hotel they stayed at.
* Validate:
  + Guest and hotel exist
  + Rating 1–5
  + One review per hotel per guest

**GET /api/reviews**

* Retrieve all reviews with guest and hotel info.

**GET /api/reviews/hotel/{hotelId}**

* Show reviews for a specific hotel with average rating and total review count.

**GET /api/reviews/summary**

* Group by hotel name showing:
  + Hotel
  + Average rating
  + Total reviews
  + Top 3 latest comments

**5. DTOs**

Create and use:

* HotelCreateDto, HotelReadDto, HotelUpdateDto
* RoomCreateDto, RoomReadDto, RoomUpdateDto
* GuestCreateDto, GuestReadDto, GuestUpdateDto
* **GuestProfileCreateDto, GuestProfileReadDto, GuestProfileUpdateDto**
* BookingCreateDto, BookingReadDto, BookingUpdateDto
* ServiceCreateDto, ServiceReadDto, ServiceUpdateDto
* ReviewCreateDto, ReviewReadDto

**Notes:**

* Use validation attributes: [Required], [EmailAddress], [Range], [StringLength]
* BookingReadDto includes list of services used and total cost
* HotelReadDto includes computed average rating

**6. Repository Pattern & Dependency Injection**

**IGenericRepository<T>**

Task<IEnumerable<T>> GetAllAsync();

Task<T?> GetByIdAsync(int id);

Task AddAsync(T entity);

void Update(T entity);

void Delete(T entity);

Task SaveChangesAsync();

**Specialized Repositories**

* **IBookingRepository**
  + Task<bool> HasOverlappingBooking(int roomId, DateTime checkIn, DateTime checkOut);
  + Task<decimal> CalculateBookingTotal(int bookingId);
* **IServiceRepository**
  + Task<IEnumerable<ServiceUsageDto>> GetServiceUsageSummary();
* **IReviewRepository**
  + Task<decimal> GetAverageRating(int hotelId);
* **IGuestProfileRepository**
  + Task<GuestProfile?> GetByGuestIdAsync(int guestId);
  + Task<bool> ExistsForGuestAsync(int guestId);

Register all repositories and context with **Scoped** lifetime.

**7. Testing & Validations**

**Testing**

* Use Swagger or Postman to test all endpoints.
* Validate data seeding (sample hotels, services).
* Test analytical LINQ endpoints (grouping, filtering, summaries).

**Validations**

* Hotel name, guest email, and service name must be unique.
* Price fields must be positive.
* CheckOut > CheckIn.
* Prevent deletion of hotels/rooms with active bookings.
* Prevent multiple reviews per guest-hotel pair.
* Guest can have only one GuestProfile.

**8. Source Control**

* Initialize Git repository.
* Commit after each controller or model change.
* Create branches for major features (Bookings, Services, Reviews).
* Push final tested version to remote repository.