TED UNIVERSITY

SPRING 2025

CMPE 313/SENG 214

Software Engineering

Prestige Hotel Reservation

Software Design Document (SDD)

SECTION 2 - TEAM 3

Submission Date:

Team Members:

- Harun Emre Erten
- Umut Uygur
- Haktan Mekik
- Ulkar Ahmadli
- Mert Temür
- Talha Saraç

Naming convention for this document: [sect no]-[team no]-[projectname]-SDD-[submission date dd//mm//yy] e.g. Sect1-Team1-HospitalManagement-SDD-150523.

Revision History

Name	Date	Reason For Changes	Version

TABLE OF CONTENTS

1. Introduction
    1. Purpose
    2. Scope
    3. Overview
    4. Reference Material
    5. Definitions and Acronyms
2. System Overview

3. System Architecture
    1. Architectural Design
    2. Decomposition Description
    3. Design Rationale
4. Data Design
    1. Data Description
    2. Data Dictionary
5. Component Design
6. Human Interface Design
    1. Overview of User Interface
    2. Screen Images
    3. Screen Objects and Actions
7. Requirements Matrix
8. Appendices

1. Introduction

1.1 Purpose

This Software Design Document (SDD) describes the system architecture and design of the Prestige Hotel Reservation system in accordance with the IEEE Std 1016-2009 standard. It serves as a comprehensive guide for developers, testers, system architects, and stakeholders to understand the technical implementation of the system requirements specified in the Software Requirements Specification (SRS) document. The document provides detailed design specifications for implementing the hotel reservation platform that enables users to search, book, and manage hotel reservations. This document is intended for use in both forward design and potential reverse-engineering contexts and is applicable regardless of system size and complexity.

1.2 Scope

The Prestige Hotel Reservation system is a web-based application that provides a complete solution for hotel booking and management. The system includes user authentication and authorization for three user types (customers, hotel administrators, and system administrators), as well as hotel search functionality with various filtering options for room booking and payment processing. It also contains hotel management tools for administrators and administrative tools for system maintenance and user management. Additionally, it involves rating and feedback system for customer reviews.

1.3 Overview

This document includes sections that cover system overview, system architecture following the MVC pattern, data design, component-level pseudocode for each MVC component, user interface details,

requirement traceability and supplementary appendices.

1.4 Reference Material

- Project Proposal Document
- SRS Document
- ASP.NET MVC Framework Documentation

1.5 Definitions and Acronyms

- SDD – Software Design Document
- SRS – Software Requirements Specifications
- API – Application Programming Interface
- Auth0 – Third-party authentication and authorization service
- DTO – Data Transfer Object
- GUI – Graphical User Interface
- MVC – Model-View-Controller architectural pattern
- MSSQL – Microsoft SQL Server
- OAuth – Open Authorization protocol
- Redis - In-memory data structure store used for caching
- REST – Representational State Transfer
- TLS – Transport Layer Security
- UI/UX - User Interface/User Experience

2. System Overview

The Prestige Hotel Reservation system is a web platform designed for hotel bookings and management. The system includes 3 types of user access:

1. **Customers**: Can search hotels in 2 separate searching options, make reservations, provide ratings and feedback, and manage their bookings.
2. **Hotel Administrators**: Can manage hotel information which they own, room availability, pricing, handle bookings, and view customer feedback.
3. **System Administrators**: Can approve hotel requests, manage users, handle complaints, and maintain system integrity.

Features Include:

- Auth0 for authentication and authorization
- Google Maps for location services

- Iyzico payment gateway for secure transactions
- Hotel search and filtering with frequent updates
- Booking process with real-time room availability and payment handling
- Role-based dashboards for hotel admins and system admins
- User rating and feedback functionality
- Complaint/help desk system

The system is designed with ASP.NET MVC architecture supporting secure, scalable, maintainable software design, also integrating external services like Auth0, Google Maps, and Iyzico.

3. System Architecture

3.1 Architectural Design

The Prestige Hotel Reservation system follows the Model-View-Controller (MVC) architectural pattern. This pattern organizes the application into three interconnected components that separate the internal representations of information from the ways information is presented to and accepted from the user.

MVC Components:

1. Model Layer:

- Represents the application's data and business logic
- Includes domain entities (User, Hotel, Room, Booking, etc.)
- Handles data validation and business rules
- Communicates with the database via repositories
- Business services encapsulate complex operations

2. View Layer:

- Responsible for rendering the user interface
- Implemented using Razor views with HTML/CSS/JavaScript
- Role-specific views for different user types
- Responsive design for desktop and mobile support

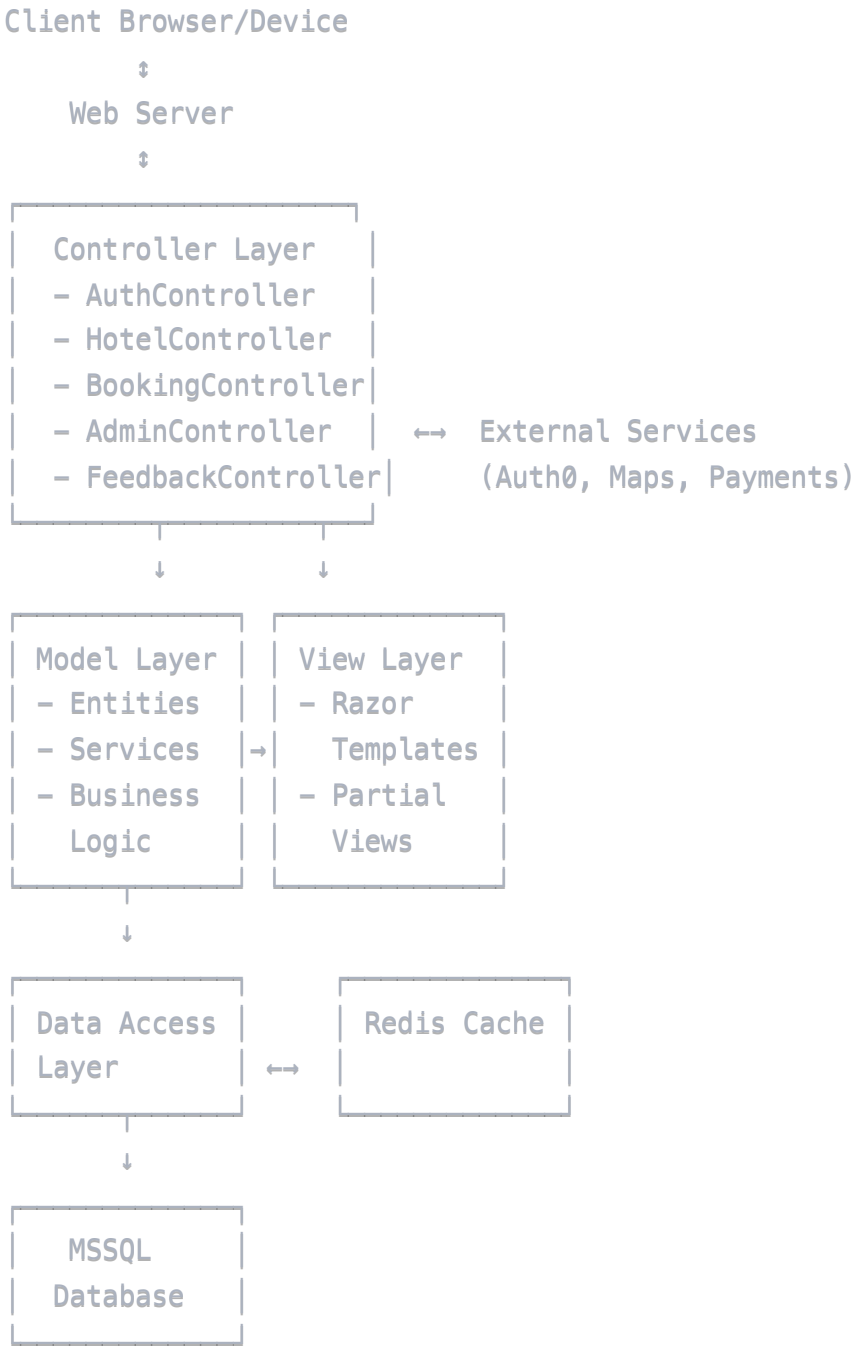
3. Controller Layer:

- Processes incoming HTTP requests
- Coordinates between Models and Views
- Handles routing and navigation flow
- Implements actions for CRUD operations
- Contains logic for input validation and error handling

Supporting Components:

- **Data Access Layer:**
  - Entity Framework Core as ORM for database interaction
  - Repository pattern for data access abstraction
  - Unit of Work pattern for transaction management
- **Authentication and Authorization:**
  - Auth0 integration for secure identity management
  - Role-based access control for different user types
  - JWT token validation for API security
- **External Service Integration:**
  - Google Maps API for location services
  - Iyzico for payment processing
  - Email service for notifications
- **Caching Layer:**
  - Redis for distributed caching
  - Cached hotel search results and frequently accessed data

High-Level Diagram:



3.2 Decomposition Description

3.2.1 Controllers

3.2.1.1 AuthController

Identification: AuthController

Type: Controller

Purpose: Manages user authentication and account management

Functions: Handles user registration, login, and password management

Actions:

- Login
- Register
- ChangePassword
- ForgotPassword
- ResetPassword
- Logout

3.2.1.2 HotelController

Identification: HotelController

Type: Controller

Purpose: Handles hotel listing and details

Functions: Displays hotel information, search results, and details pages

Actions:

- Index (list all hotels)
- Details (view specific hotel)
- Search (search hotels by criteria)
- Filter (filter search results)

3.2.1.3 ReservationController

Identification: ReservationController

Type: Controller

Purpose: Manages the booking process

Functions: Handles room reservation, modification, and cancellation

Actions:

- Create (create new reservation)
- Edit (modify existing reservation)

- Cancel (cancel reservation)
- Confirm (confirm reservation details)
- Payment (process payment)

3.2.1.4 HotelManagementController

Identification: HotelManagementController

Type: Controller

Purpose: Provides hotel management functionality for hotel administrators

Functions: Allows hotel admins to manage rooms, availability, and pricing

Actions:

- Dashboard (hotel admin dashboard)
- AddRoom (add new room)
- EditRoom (modify room details)
- DeleteRoom (remove a room)
- UpdateAvailability (update room availability)
- UpdatePricing (update room pricing)
- ManagePhotos (manage hotel and room photos)

3.2.1.5 FeedbackController

Identification: FeedbackController

Type: Controller

Purpose: Manages customer feedback and ratings

Functions: Handles submission and display of customer reviews

Actions:

- Submit (submit new feedback)
- View (view feedbacks for a hotel)
- Delete (remove inappropriate feedback - admin only)

3.2.1.6 AdminController

Identification: AdminController

Type: Controller

Purpose: Provides system administration capabilities

Functions: Manages users, approves hotels, and handles system settings

Actions:

- Dashboard (system admin dashboard)
- ManageUsers (user management)
- ApproveHotels (approve hotel listings)
- HandleComplaints (manage customer complaints)
- SystemSettings (adjust system configurations)

3.2.1.7 SupportController

Identification: SupportController

Type: Controller

Purpose: Provides customer support functionalities

Functions: Handles help requests and FAQ

Actions:

- ContactSupport (submit support request)
- FAQ (view frequently asked questions)
- ViewTickets (view support tickets - admin only)
- ReplyTicket (respond to support ticket - admin only)

3.2.2 Models

3.2.2.1 User Model

Identification: User

Type: Model

Purpose: Represents user information and authentication details

Properties:

- UserId (PK)
- FirstName
- LastName
- Email
- PhoneNumber
- UserType (Customer, HotelAdmin, SystemAdmin)
- CreatedDate
- LastLoginDate

3.2.2.2 Hotel Model

Identification: Hotel

Type: Model

Purpose: Represents hotel information

Properties:

- HotelId (PK)
- Name
- Address
- City
- Country
- Rating
- Description
- Amenities
- Photos
- AdminId (FK to User)

3.2.2.3 Room Model

Identification: Room

Type: Model

Purpose: Represents room information within a hotel

Properties:

- RoomId (PK)
- HotelId (FK)
- RoomNumber
- RoomType
- Capacity
- Price
- Description
- Photos
- IsAvailable

3.2.2.4 Reservation Model

Identification: Reservation

Type: Model

Purpose: Represents booking information

Properties:

- ReservationId (PK)
- UserId (FK)
- RoomId (FK)
- CheckInDate
- CheckOutDate
- TotalPrice
- Status (Reserved, Confirmed, Canceled, Completed)
- CreatedDate
- PaymentStatus

3.2.2.5 Payment Model

Identification: Payment

Type: Model

Purpose: Represents payment information for reservations

Properties:

- PaymentId (PK)
- ReservationId (FK)
- Amount
- PaymentDate
- PaymentMethod
- TransactionId
- Status

3.2.2.6 Feedback Model

Identification: Feedback

Type: Model

Purpose: Represents customer reviews for hotels

Properties:

- FeedbackId (PK)
- UserId (FK)
- HotelId (FK)
- Rating (1-5)
- Comment
- SubmissionDate
- Status (Pending, Approved, Rejected)

3.2.2.7 Support Model

Identification: Support

Type: Model

Purpose: Represents customer support tickets

Properties:

- TicketId (PK)
- UserId (FK)
- Subject
- Message
- CreatedDate
- Status (Open, InProgress, Resolved, Closed)
- Priority

3.2.3 Views

3.2.3.1 Home Views

Identification: Home Views

Type: View

Purpose: Displays main landing page and general information

Views:

- Index.cshtml (main landing page)
- About.cshtml (about the service)
- Contact.cshtml (contact information)
- Privacy.cshtml (privacy policy)
- Terms.cshtml (terms of service)

3.2.3.2 Auth Views

Identification: Auth Views

Type: View

Purpose: Handles user authentication interfaces

Views:

- Login.cshtml (login form)
- Register.cshtml (registration form)
- ForgotPassword.cshtml (password recovery)
- ResetPassword.cshtml (set new password)

3.2.3.3 Hotel Views

Identification: Hotel Views

Type: View

Purpose: Displays hotel listings and details

Views:

- Index.cshtml (list of hotels)
- Details.cshtml (hotel details)
- Search.cshtml (search results)
- _HotelCard.cshtml (partial view for hotel card)
- _SearchFilter.cshtml (partial view for search filters)

3.2.3.4 Reservation Views

Identification: Reservation Views

Type: View

Purpose: Handles reservation process interfaces

Views:

- Create.cshtml (booking form)
- Edit.cshtml (modify reservation)
- Confirm.cshtml (confirmation page)
- Payment.cshtml (payment processing)
- Success.cshtml (successful booking)
- Cancel.cshtml (cancellation confirmation)

3.2.3.5 User Dashboard Views

Identification: User Dashboard Views

Type: View

Purpose: Customer account management interfaces

Views:

- Index.cshtml (dashboard home)
- Profile.cshtml (user profile)
- Reservations.cshtml (booking history)
- Feedback.cshtml (submitted reviews)
- SavedHotels.cshtml (favorite hotels)

3.2.3.6 Hotel Management Views

Identification: Hotel Management Views

Type: View

Purpose: Hotel administration interfaces

Views:

- Dashboard.cshtml (admin dashboard)
- Rooms.cshtml (room management)
- Bookings.cshtml (reservation management)
- AddRoom.cshtml (add new room)
- EditRoom.cshtml (edit room details)
- HotelProfile.cshtml (hotel information)

3.2.3.7 Admin Views

Identification: Admin Views

Type: View

Purpose: System administration interfaces

Views:

- Dashboard.cshtml (admin dashboard)
- Users.cshtml (user management)
- Hotels.cshtml (hotel approval)
- Complaints.cshtml (complaint management)

- Settings.cshtml (system settings)

3.2.3.8 Support Views

Identification: Support Views

Type: View

Purpose: Customer support interfaces

Views:

- Contact.cshtml (contact form)
- FAQ.cshtml (frequently asked questions)
- Tickets.cshtml (support tickets - admin only)
- TicketDetails.cshtml (ticket details and responses)

3.2.4 Services

3.2.4.1 UserService

Identification: UserService

Type: Service

Purpose: Handles user-related business logic

Functions:

- CreateUser
- UpdateUser
- GetUserById
- GetUserByEmail
- ValidateCredentials
- ChangePassword
- GeneratePasswordResetToken

3.2.4.2 HotelService

Identification: HotelService

Type: Service

Purpose: Manages hotel-related business logic

Functions:

- CreateHotel
- UpdateHotel
- DeleteHotel
- GetHotelById
- GetHotelsByOwner
- ApproveHotel
- SearchHotels
- FilterHotels

3.2.4.3 RoomService

Identification: RoomService

Type: Service

Purpose: Manages room-related business logic

Functions:

- AddRoom
- UpdateRoom
- DeleteRoom
- CheckAvailability
- UpdateAvailability
- UpdatePricing
- GetRoomsByHotel
- GetAvailableRooms

3.2.4.4 ReservationService

Identification: ReservationService

Type: Service

Purpose: Handles booking-related business logic

Functions:

- CreateReservation
- UpdateReservation
- CancelReservation
- GetReservationById
- GetReservationsByUser
- GetReservationsByHotel
- CalculatePrice
- CheckReservationStatus

3.2.4.5 PaymentService

Identification: PaymentService

Type: Service

Purpose: Manages payment processing

Functions:

- ProcessPayment
- ValidatePayment
- GetPaymentStatus
- RefundPayment
- GetPaymentHistory
- CalculateTotalAmount

3.2.4.6 FeedbackService

Identification: FeedbackService

Type: Service

Purpose: Handles review and rating functionality

Functions:

- SubmitFeedback

- GetFeedbackByHotel
- GetFeedbackByUser
- ModerateFeedback
- CalculateAverageRating
- DeleteFeedback

3.2.4.7 SupportService

Identification: SupportService

Type: Service

Purpose: Manages customer support functionality

Functions:

- CreateSupportTicket
- UpdateTicketStatus
- GetTicketById
- GetTicketsByUser
- GetAllTickets
- RespondToTicket
- CloseTicket

3.2.5 Data Access

3.2.5.1 UserRepository

Identification: UserRepository

Type: Repository

Purpose: Data access for User entities

Methods:

- Add
- Update
- Delete
- GetById

- GetByEmail
- GetAll
- GetByRole

3.2.5.2 HotelRepository

Identification: HotelRepository

Type: Repository

Purpose: Data access for Hotel entities

Methods:

- Add
- Update
- Delete
- GetById
- GetByAdminId
- GetAll
- Search
- GetWithRooms

3.2.5.3 RoomRepository

Identification: RoomRepository

Type: Repository

Purpose: Data access for Room entities

Methods:

- Add
- Update
- Delete
- GetById
- GetByHotelId
- GetAvailable
- CheckAvailability

3.2.5.4 ReservationRepository

Identification: ReservationRepository

Type: Repository

Purpose: Data access for Reservation entities

Methods:

- Add
- Update
- Delete
- GetById
- GetByUserId
- GetByHotelId
- GetByStatus
- GetActive

3.2.5.5 PaymentRepository

Identification: PaymentRepository

Type: Repository

Purpose: Data access for Payment entities

Methods:

- Add
- Update
- GetById
- GetByReservationId
- GetByTransactionId
- GetByStatus

3.2.5.6 FeedbackRepository

Identification: FeedbackRepository

Type: Repository

Purpose: Data access for Feedback entities

Methods:

- Add
- Update
- Delete
- GetById
- GetByHotelId
- GetByUserId
- GetPending

3.2.5.7 SupportRepository

Identification: SupportRepository

Type: Repository

Purpose: Data access for Support entities

Methods:

- Add
- Update
- GetById
- GetByUserId
- GetByStatus
- GetAll

3.3 Design Rationale

The Model-View-Controller (MVC) architectural pattern was chosen for the Prestige Hotel Reservation system for the following reasons:

1. **Separation of Concerns**: MVC clearly separates the application into data (Model), user interface (View), and control logic (Controller), making the system more maintainable and testable.
2. **Framework Support**: ASP.NET Core MVC provides built-in support for this pattern with a robust framework that handles routing, model binding, validation, and view rendering.

3. **Testability**: The separation of components makes unit testing more straightforward, particularly for controllers and models.
4. **Parallel Development**: Teams can work simultaneously on different aspects (UI, business logic, data access) with minimal conflicts.
5. **Code Reusability**: Models can be reused across multiple views, and controller actions can be reused for different request types.
6. **Scalability**: The clear separation enables easier horizontal scaling of components based on load.
7. **Industry Standard**: MVC is widely adopted in web development, making it easier to onboard new developers familiar with the pattern.

Integration Choices:

- **Auth0**: Chosen for authentication to avoid the complexity and security risks of developing a custom authentication system. It provides secure, standards-based authentication with features like social login and multi-factor authentication.
- **Entity Framework Core**: Selected as the ORM for data access due to its seamless integration with ASP.NET Core and its ability to simplify database operations.
- **Redis Caching**: Implemented to improve performance for frequently accessed data, particularly for hotel search results and availability information.
- **Iyzico Payment Gateway**: Selected for its robust API, security features, and regional support for payment processing.

Alternatives Considered:

- **Microservices Architecture**: Considered but deemed too complex for the initial scope of the project. The system is designed with clear boundaries between components to potentially migrate to microservices in the future if needed.
- **SPA with Web API**: A Single Page Application with separate Web API was considered but rejected in favor of the more straightforward MVC approach for the initial phase.
- **Custom Authentication System**: Rejected in favor of Auth0 to reduce security risks and development time.

4. Data Design

4.1 Data Description

The Prestige Hotel Reservation System uses Microsoft SQL Server, a relational database management system, for persistent data storage. The data model is organized according to the domain entities in the MVC architecture, with clear relationships between tables. Additionally, Redis is used for caching frequently accessed data to improve performance.

The primary entities in the system include:

- **User Information**: Stores authentication details and profile information for all users (customers, hotel admins, system admins).
- **Hotel Data**: Contains hotel details, amenities, and policies.
- **Room Information**: Manages room types, availability, pricing, and features.
- **Reservation Data**: Tracks bookings, check-in/out dates, and status.
- **Payment Information**: Records payment transactions and status.
- **Feedback Data**: Stores ratings and reviews from customers.
- **Support Tickets**: Manages customer inquiries and admin responses.

4.2 Data Dictionary

Hotel Table

Field Name	Data Type	Data Description
HotelId	INT (PK)	Unique identifier for hotel
HotelName	VARCHAR(100)	Name of the Hotel
Address	VARCHAR(200)	Physical address
Rating	INT	Hotel's star rating
City	VARCHAR(50)	City location
Country	VARCHAR(50)	Country location
Description	TEXT	Detailed description
AdminId	INT (FK)	Hotel administrator reference

User Table

Field Name	Data Type	Data Description
UserId	INT (PK)	Unique identifier for user
FirstName	VARCHAR(50)	First name of the user
LastName	VARCHAR(50)	Last name of the user
Email	VARCHAR(200)	Email of the user for site
PhoneNumber	VARCHAR(20)	Phone number of user
UserType	VARCHAR(20)	Type of user (Customer, HotelAdmin, SystemAdmin)
CreatedDate	DATETIME	Account creation date
LastLoginDate	DATETIME	Most recent login date

Room Table

Field Name	Data Type	Data Description
RoomId	INT (PK)	Unique identifier for Room
HotelId	INT (FK)	Reference to the Hotel's PK
RoomType	VARCHAR(50)	Type of room
PictureUrl	VARCHAR(255)	URL to room picture
Capacity	INT	Capacity of the Room
Price	DECIMAL(10,2)	Price per night
RoomNumber	VARCHAR(20)	Room identifier within hotel
Description	TEXT	Room description
IsAvailable	BOOLEAN	Current availability status

Reservation Table

Field Name	Data Type	Data Description
ReservationId	INT (PK)	Unique identifier for reservation
UserId	INT (FK)	Reference to User table
RoomId	INT (FK)	Reference to Room table
CheckInDate	DATE	Arrival date
CheckOutDate	DATE	Departure date
TotalPrice	DECIMAL(10,2)	Total cost of stay
Status	VARCHAR(20)	Reservation status
CreatedDate	DATETIME	When reservation was made
SpecialRequests	TEXT	Customer's special requests

Payment Table

Field Name	Data Type	Data Description
PaymentId	INT (PK)	Unique identifier for payment
ReservationId	INT (FK)	Reference to Reservation table
Amount	DECIMAL(10,2)	Payment amount
PaymentDate	DATETIME	When payment was processed
PaymentMethod	VARCHAR(50)	Method of payment
TransactionId	VARCHAR(100)	Payment gateway transaction ID
Status	VARCHAR(20)	Payment status

Feedback Table

Field Name	Data Type	Data Description
FeedbackId	INT (PK)	Unique identifier for feedback
UserId	INT (FK)	Reference to User table
HotelId	INT (FK)	Reference to Hotel table
Rating	INT	Star rating (1-5)
Comment	TEXT	Written review
SubmissionDate	DATETIME	When feedback was submitted
Status	VARCHAR(20)	Feedback status (approved, pending, rejected)

Support Table

Field Name	Data Type	Data Description
TicketId	INT (PK)	Unique identifier for support ticket
UserId	INT (FK)	Reference to User table
Subject	VARCHAR(100)	Ticket subject
Message	TEXT	Customer's message
CreatedDate	DATETIME	When ticket was created
Status	VARCHAR(20)	Ticket status
Priority	VARCHAR(20)	Priority level

5. Component Design

5.1 Controllers

5.1.1 AuthController

Type: Controller

Purpose: Provides functionality for user registration, login, and password management.

Component Interface:

- Input: User credentials, registration details
- Output: Success/failure messages, redirect to appropriate pages
- Dependencies: UserService, Auth0Service

5.1.1.1 Login Action

Purpose: Authenticate users

Type: Controller Action

Pseudocode:

```
FUNCTION Login(email, password):
    IF Request.Method == "GET" THEN
        RETURN View()
    ENDIF

    result ← UserService.AuthenticateUser(email, password)

    IF result.Success THEN
        CreateAuthCookie(result.UserInfo)
        RETURN RedirectToAction("Index", "Home")
    ELSE
        ModelState.AddError("Authentication failed: " + result.ErrorMessage)
        RETURN View()
    ENDIF
END FUNCTION
```

5.1.1.2 Register Action

Purpose: Register a new user

Type: Controller Action

Pseudocode:

```
FUNCTION Register(registerViewModel):
    IF Request.Method == "GET" THEN
        RETURN View()
    ENDIF

    IF NOT ModelState.IsValid THEN
        RETURN View(registerViewModel)
    ENDIF

    result ← UserService.RegisterUser(registerViewModel)

    IF result.Success THEN
        RETURN RedirectToAction("Login", "Auth", { message = "Registration
successful" })
    ELSE
        ModelState.AddError("Registration failed: " + result.ErrorMessage)
        RETURN View(registerViewModel)
    ENDIF
END FUNCTION
```

5.1.1.3 ChangePassword Action

Purpose: Allow users to change their password

Type: Controller Action

Pseudocode:

```
FUNCTION ChangePassword(changePasswordViewModel):
    IF Request.Method == "GET" THEN
        RETURN View()
    ENDIF

    IF NOT ModelState.IsValid THEN
        RETURN View(changePasswordViewModel)
    ENDIF

    userId ← GetCurrentUserId()
    result ← UserService.ChangePassword(userId,
changePasswordViewModel.CurrentPassword, changePasswordViewModel.NewPassword)

    IF result.Success THEN
        RETURN RedirectToAction("Index", "UserDashboard", { message = "Password
changed successfully" })
    ELSE
        ModelState.AddError("Password change failed: " + result.ErrorMessage)
        RETURN View(changePasswordViewModel)
    ENDIF
END FUNCTION
```

5.1.2 HotelController

Type: Controller

Purpose: Manages hotel listings, search, and details.

Component Interface:

- Input: Search criteria, filter options
- Output: Hotel listings, detailed hotel information
- Dependencies: HotelService, RoomService, FeedbackService

5.1.2.1 Index Action

Purpose: Display list of hotels

Type: Controller Action

Pseudocode:

```
FUNCTION Index(page = 1, pageSize = 10):
    hotelListViewModel ← NEW HotelListViewModel

    totalHotels ← HotelService.GetTotalHotelCount()
    hotels ← HotelService.GetHotels(page, pageSize)

    hotelListViewModel.Hotels ← hotels
    hotelListViewModel.Pagination ← NEW PaginationInfo(page, pageSize, totalHotels)

    RETURN View(hotelListViewModel)
END FUNCTION
```

5.1.2.2 Search Action

Purpose: Search for hotels based on criteria

Type: Controller Action

Pseudocode:

```
FUNCTION Search(searchViewModel):
    IF NOT ModelState.IsValid THEN
        RETURN RedirectToAction("Index")
    ENDIF

    searchResults ← HotelService.SearchHotels(
        searchViewModel.Location,
        searchViewModel.CheckInDate,
        searchViewModel.CheckOutDate,
        searchViewModel.Guests,
        searchViewModel.Page,
        searchViewModel.PageSize
    )

    resultViewModel ← NEW SearchResultViewModel
    resultViewModel.Hotels ← searchResults.Hotels
    resultViewModel.TotalResults ← searchResults.TotalCount
    resultViewModel.SearchCriteria ← searchViewModel
    resultViewModel.Pagination ← NEW PaginationInfo(searchViewModel.Page,
searchViewModel.PageSize, searchResults.TotalCount)

    RETURN View(resultViewModel)
END FUNCTION
```

5.1.2.3 Details Action

Purpose: Display detailed hotel information

Type: Controller Action

Pseudocode:

```
FUNCTION Details(id):
    hotel ← HotelService.GetHotelById(id)

    IF hotel IS NULL THEN
        RETURN NotFound()
    ENDIF

    rooms ← RoomService.GetRoomsByHotel(id)
    reviews ← FeedbackService.GetFeedbackByHotel(id, 1, 5)  // First 5 reviews

    detailsViewModel ← NEW HotelDetailsViewModel
    detailsViewModel.Hotel ← hotel
    detailsViewModel.Rooms ← rooms
    detailsViewModel.Reviews ← reviews
    detailsViewModel.AverageRating ← FeedbackService.GetAverageRating(id)

    RETURN View(detailsViewModel)
END FUNCTION
```

5.1.2.4 Filter Action

Purpose: Filter hotel search results

Type: Controller Action

Pseudocode:

```
FUNCTION Filter(filterViewModel):
    IF NOT ModelState.IsValid THEN
        RETURN RedirectToAction("Search", filterViewModel.SearchCriteria)
    ENDIF

    filteredResults ← HotelService.FilterHotels(
        filterViewModel.SearchCriteria,
        filterViewModel.PriceMin,
        filterViewModel.PriceMax,
        filterViewModel.StarRating,
        filterViewModel.Amenities,
        filterViewModel.SortBy,
        filterViewModel.Page,
        filterViewModel.PageSize
    )

    resultViewModel ← NEW SearchResultViewModel
    resultViewModel.Hotels ← filteredResults.Hotels
    resultViewModel.TotalResults ← filteredResults.TotalCount
    resultViewModel.SearchCriteria ← filterViewModel.SearchCriteria
    resultViewModel.FilterCriteria ← filterViewModel
    resultViewModel.Pagination ← NEW PaginationInfo(filterViewModel.Page,
filterViewModel.PageSize, filteredResults.TotalCount)

    RETURN View("Search", resultViewModel)
END FUNCTION
```

5.1.3 ReservationController

Type: Controller

Purpose: Manages the booking process and reservation management.

Component Interface:

- Input: Reservation details, room selection, payment information
- Output: Booking confirmation, reservation status updates
- Dependencies: ReservationService, RoomService, PaymentService, EmailService

5.1.3.1 Create Action

Purpose: Create a new reservation

Type: Controller Action

Pseudocode:

```
FUNCTION Create(roomId, checkInDate, checkOutDate, guests):
    IF NOT IsAuthenticated() THEN
        RETURN RedirectToAction("Login", "Auth", { returnUrl = CurrentUrl })
    ENDIF

    room ← RoomService.GetRoomById(roomId)

    IF room IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF NOT RoomService.IsRoomAvailable(roomId, checkInDate, checkOutDate) THEN
        RETURN RedirectToAction("Details", "Hotel", { id = room.HotelId, error =
"Room no longer available" })
    ENDIF

    totalPrice ← ReservationService.CalculatePrice(roomId, checkInDate,
checkOutDate)

    bookingViewModel ← NEW BookingViewModel
    bookingViewModel.Room ← room
    bookingViewModel.CheckInDate ← checkInDate
    bookingViewModel.CheckOutDate ← checkOutDate
    bookingViewModel.Guests ← guests
    bookingViewModel.TotalPrice ← totalPrice

    RETURN View(bookingViewModel)
END FUNCTION
```

5.1.3.2 Confirm Action

Purpose: Confirm reservation details before payment

Type: Controller Action

Pseudocode:

```
FUNCTION Confirm(bookingViewModel):
    IF NOT ModelState.IsValid THEN
        RETURN View("Create", bookingViewModel)
    ENDIF

    // Re-validate availability
    IF NOT RoomService.IsRoomAvailable(bookingViewModel.RoomId,
bookingViewModel.CheckInDate, bookingViewModel.CheckOutDate) THEN
        ModelState.AddError("Room is no longer available for selected dates")
        RETURN View("Create", bookingViewModel)
    ENDIF

    // Re-calculate price to ensure accuracy
    bookingViewModel.TotalPrice ← ReservationService.CalculatePrice(
        bookingViewModel.RoomId,
        bookingViewModel.CheckInDate,
        bookingViewModel.CheckOutDate
    )

    RETURN View(bookingViewModel)
END FUNCTION
```

5.1.3.3 Payment Action

Purpose: Process payment for reservation

Type: Controller Action

Pseudocode:

```
FUNCTION Payment(bookingViewModel):
    IF NOT ModelState.IsValid THEN
        RETURN View("Confirm", bookingViewModel)
    ENDIF

    // Create temporary reservation
    reservationId ← ReservationService.CreateReservation(
        GetCurrentUserId(),
        bookingViewModel.RoomId,
        bookingViewModel.CheckInDate,
        bookingViewModel.CheckOutDate,
        bookingViewModel.Guests,
        bookingViewModel.TotalPrice,
        "Pending"
    )

    // Prepare payment model
    paymentViewModel ← NEW PaymentViewModel
    paymentViewModel.ReservationId ← reservationId
    paymentViewModel.Amount ← bookingViewModel.TotalPrice
    paymentViewModel.Currency ← "USD"
    paymentViewModel.ReturnUrl ← GeneratePaymentReturnUrl(reservationId)

    RETURN View(paymentViewModel)
END FUNCTION
```

5.1.3.4 PaymentComplete Action

Purpose: Handle payment completion

Type: Controller Action

Pseudocode:

```
FUNCTION PaymentComplete(reservationId, status, transactionId):
    reservation ← ReservationService.GetReservationById(reservationId)

    IF reservation IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF status == "success" THEN
        // Update reservation status
        ReservationService.UpdateReservationStatus(reservationId, "Confirmed")

        // Record payment
        PaymentService.RecordPayment(reservationId, reservation.TotalPrice, "Credit
Card", transactionId, "Completed")

        // Send confirmation email
        EmailService.SendReservationConfirmation(reservation)

        RETURN RedirectToAction("Success", new { id = reservationId })
    ELSE
        // Handle failed payment
        ReservationService.UpdateReservationStatus(reservationId, "PaymentFailed")

        RETURN RedirectToAction("Failed", new { id = reservationId, error = "Payment
processing failed" })
    ENDIF
END FUNCTION
```

5.1.3.5 Cancel Action

Purpose: Cancel an existing reservation

Type: Controller Action

Pseudocode:

```
FUNCTION Cancel(id):
    reservation ← ReservationService.GetReservationById(id)

    IF reservation IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF reservation.UserId != GetCurrentUserId() AND NOT IsAdmin() THEN
        RETURN Forbidden()
    ENDIF

    IF Request.Method == "GET" THEN
        RETURN View(reservation)
    ENDIF

    result ← ReservationService.CancelReservation(id)

    IF result.Success THEN
        // Send cancellation email
        EmailService.SendCancellationConfirmation(reservation)

        RETURN RedirectToAction("Index", "UserDashboard", { message = "Reservation
cancelled successfully" })
    ELSE
        RETURN View(reservation, { error = result.ErrorMessage })
    ENDIF
END FUNCTION
```

5.1.4 HotelManagementController

Type: Controller

Purpose: Provides hotel administration functionality.

Component Interface:

- Input: Hotel and room details, availability updates
- Output: Management dashboards, confirmation messages
- Dependencies: HotelService, RoomService, ReservationService

5.1.4.1 Dashboard Action

Purpose: Display hotel admin dashboard

Type: Controller Action

Pseudocode:

```
FUNCTION Dashboard():
    IF NOT IsHotelAdmin() THEN
        RETURN RedirectToAction("Login", "Auth")
    ENDIF

    adminId ← GetCurrentUserId()
    hotels ← HotelService.GetHotelsByAdminId(adminId)

    dashboardViewModel ← NEW HotelAdminDashboardViewModel
    dashboardViewModel.Hotels ← hotels

    IF hotels.Count == 1 THEN
        // If admin has only one hotel, load detailed stats
        hotel ← hotels[0]
        dashboardViewModel.CurrentBookings ←
ReservationService.GetActiveReservationsByHotel(hotel.Id)
        dashboardViewModel.RecentReviews ←
FeedbackService.GetRecentFeedbackByHotel(hotel.Id, 5)
        dashboardViewModel.OccupancyRate ← RoomService.GetOccupancyRate(hotel.Id)
        dashboardViewModel.RevenueStats ←
ReservationService.GetRevenueStats(hotel.Id)
    ENDIF

    RETURN View(dashboardViewModel)
END FUNCTION
```

5.1.4.2 AddRoom Action

Purpose: Add a new room to a hotel

Type: Controller Action

Pseudocode:

```
FUNCTION AddRoom(hotelId):
    IF NOT CanManageHotel(hotelId) THEN
        RETURN Forbidden()
    ENDIF

    hotel ← HotelService.GetHotelById(hotelId)

    IF hotel IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF Request.Method == "GET" THEN
        roomViewModel ← NEW RoomViewModel
        roomViewModel.HotelId ← hotelId
        RETURN View(roomViewModel)
    ENDIF

    IF NOT ModelState.IsValid THEN
        RETURN View(roomViewModel)
    ENDIF

    roomId ← RoomService.AddRoom(
        roomViewModel.HotelId,
        roomViewModel.RoomNumber,
        roomViewModel.RoomType,
        roomViewModel.Capacity,
        roomViewModel.Price,
        roomViewModel.Description,
        roomViewModel.Amenities,
        roomViewModel.IsAvailable
    )

    // Handle image uploads if any
    IF roomViewModel.Images IS NOT NULL AND roomViewModel.Images.Count > 0 THEN
        FOREACH image IN roomViewModel.Images DO
            RoomService.AddRoomImage(roomId, image)
        END FOREACH
    ENDIF

    RETURN RedirectToAction("Rooms", new { hotelId = hotelId, message = "Room added
successfully" })
END FUNCTION
```

5.1.4.3 EditRoom Action

Purpose: Modify room details

Type: Controller Action

Pseudocode:

```
FUNCTION EditRoom(id):
    room ← RoomService.GetRoomById(id)

    IF room IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF NOT CanManageHotel(room.HotelId) THEN
        RETURN Forbidden()
    ENDIF

    IF Request.Method == "GET" THEN
        roomViewModel ← MapRoomToViewModel(room)
        RETURN View(roomViewModel)
    ENDIF

    IF NOT ModelState.IsValid THEN
        RETURN View(roomViewModel)
    ENDIF

    result ← RoomService.UpdateRoom(
        id,
        roomViewModel.RoomNumber,
        roomViewModel.RoomType,
        roomViewModel.Capacity,
        roomViewModel.Price,
        roomViewModel.Description,
        roomViewModel.Amenities,
        roomViewModel.IsAvailable
    )

    // Handle image uploads if any
    IF roomViewModel.Images IS NOT NULL AND roomViewModel.Images.Count > 0 THEN
        FOREACH image IN roomViewModel.Images DO
            RoomService.AddRoomImage(id, image)
        END FOREACH
    ENDIF

    RETURN RedirectToAction("Rooms", new { hotelId = room.HotelId, message = "Room
updated successfully" })
END FUNCTION
```

5.1.4.4 DeleteRoom Action

Purpose: Remove a room from a hotel

Type: Controller Action

Pseudocode:

```
FUNCTION DeleteRoom(id):
    room ← RoomService.GetRoomById(id)

    IF room IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF NOT CanManageHotel(room.HotelId) THEN
        RETURN Forbidden()
    ENDIF

    // Check if room has active reservations
    hasActiveReservations ← ReservationService.HasActiveReservations(id)

    IF hasActiveReservations THEN
        RETURN RedirectToAction("Rooms", new { hotelId = room.HotelId, error =
"Cannot delete room with active reservations" })
    ENDIF

    hotelId ← room.HotelId  // Save before deletion

    RoomService.DeleteRoom(id)

    RETURN RedirectToAction("Rooms", new { hotelId = hotelId, message = "Room
deleted successfully" })
END FUNCTION
```

5.1.5 FeedbackController

Type: Controller

Purpose: Manages user reviews and ratings.

Component Interface:

- Input: User ratings and reviews
- Output: Confirmation messages, feedback listings
- Dependencies: FeedbackService, HotelService, ReservationService

5.1.5.1 Submit Action

Purpose: Submit a new review for a hotel

Type: Controller Action

Pseudocode:

```
FUNCTION Submit(hotelId):
    IF NOT IsAuthenticated() THEN
        RETURN RedirectToAction("Login", "Auth", { returnUrl = CurrentUrl })
    ENDIF

    userId ← GetCurrentUserId()

    // Check if user has stayed at this hotel
    hasStayed ← ReservationService.HasUserStayedAtHotel(userId, hotelId)

    IF NOT hasStayed AND NOT IsAdmin() THEN
        RETURN RedirectToAction("Details", "Hotel", { id = hotelId, error = "You can
only review hotels you've stayed at" })
    ENDIF

    hotel ← HotelService.GetHotelById(hotelId)

    IF hotel IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF Request.Method == "GET" THEN
        feedbackViewModel ← NEW FeedbackViewModel
        feedbackViewModel.HotelId ← hotelId
        feedbackViewModel.HotelName ← hotel.Name

        RETURN View(feedbackViewModel)
    ENDIF

    IF NOT ModelState.IsValid THEN
        feedbackViewModel.HotelName ← hotel.Name
        RETURN View(feedbackViewModel)
    ENDIF

    result ← FeedbackService.SubmitFeedback(
        userId,
        hotelId,
        feedbackViewModel.Rating,
        feedbackViewModel.Comment
    )

    IF result.Success THEN
        RETURN RedirectToAction("Details", "Hotel", { id = hotelId, message = "Thank
you for your feedback" })
    ELSE
        ModelState.AddError(result.ErrorMessage)
```

```
        feedbackViewModel.HotelName ← hotel.Name
        RETURN View(feedbackViewModel)
    ENDIF
END FUNCTION
```

5.1.6 AdminController

Type: Controller

Purpose: Provides system administration functions.

Component Interface:

- Input: Admin commands, approval decisions
- Output: Admin dashboards, user and hotel management interfaces
- Dependencies: UserService, HotelService, ReservationService, FeedbackService, SupportService

5.1.6.1 Dashboard Action

Purpose: Display system admin dashboard

Type: Controller Action

Pseudocode:

```
FUNCTION Dashboard():
    IF NOT IsSystemAdmin() THEN
        RETURN RedirectToAction("Login", "Auth")
    ENDIF

    dashboardViewModel ← NEW AdminDashboardViewModel
    dashboardViewModel.UserCount ← UserService.GetTotalUserCount()
    dashboardViewModel.HotelCount ← HotelService.GetTotalHotelCount()
    dashboardViewModel.ReservationCount ←
ReservationService.GetTotalReservationCount()
    dashboardViewModel.PendingHotels ← HotelService.GetPendingHotels()
    dashboardViewModel.RecentUsers ← UserService.GetRecentUsers(5)
    dashboardViewModel.OpenTickets ← SupportService.GetOpenTickets(5)

    RETURN View(dashboardViewModel)
END FUNCTION
```

5.1.6.2 ApproveHotel Action

Purpose: Approve or reject a hotel listing

Type: Controller Action

Pseudocode:

```
FUNCTION ApproveHotel(id, approved):
    IF NOT IsSystemAdmin() THEN
        RETURN RedirectToAction("Login", "Auth")
    ENDIF

    hotel ← HotelService.GetHotelById(id)

    IF hotel IS NULL THEN
        RETURN NotFound()
    ENDIF

    IF approved THEN
        result ← HotelService.ApproveHotel(id)
        message ← "Hotel approved successfully"
    ELSE
        result ← HotelService.RejectHotel(id)
        message ← "Hotel rejected"
    ENDIF

    IF result.Success THEN
        // Notify hotel admin via email
        adminUser ← UserService.GetUserById(hotel.AdminId)
        EmailService.SendHotelApprovalNotification(adminUser.Email, hotel.Name,
approved)

        RETURN RedirectToAction("PendingHotels", new { message = message })
    ELSE
        RETURN RedirectToAction("PendingHotels", new { error = result.ErrorMessage
})
    ENDIF
END FUNCTION
```

5.2 Models

5.2.1 User Model

Type: Model

Purpose: Represents user information in the system.

Properties:

- UserId: int
- FirstName: string
- LastName: string
- Email: string
- PhoneNumber: string
- UserType: enum (Customer, HotelAdmin, SystemAdmin)
- CreatedDate: DateTime
- LastLoginDate: DateTime?

Methods:

```
FUNCTION IsInRole(role):
    RETURN UserType == role
END FUNCTION

FUNCTION GetFullName():
    RETURN FirstName + " " + LastName
END FUNCTION
```

5.2.2 Hotel Model

Type: Model

Purpose: Represents hotel information.

Properties:

- HotelId: int
- Name: string
- Address: string
- City: string
- Country: string
- Rating: int
- Description: string
- Amenities: List<string>
- Photos: List<string>
- AdminId: int

- Status: enum (Pending, Approved, Rejected)
- Rooms: List<Room>
- Reviews: List<Feedback>

Methods:

```
FUNCTION GetAverageRating():
    IF Reviews IS NULL OR Reviews.Count == 0 THEN
        RETURN 0
    ENDIF

    totalRating ← 0
    FOREACH review IN Reviews DO
        totalRating ← totalRating + review.Rating
    END FOREACH

    RETURN totalRating / Reviews.Count
END FUNCTION

FUNCTION GetLowestRoomPrice():
    IF Rooms IS NULL OR Rooms.Count == 0 THEN
        RETURN 0
    ENDIF

    lowestPrice ← Rooms[0].Price
    FOREACH room IN Rooms DO
        IF room.Price < lowestPrice THEN
            lowestPrice ← room.Price
        ENDIF
    END FOREACH

    RETURN lowestPrice
END FUNCTION
```

5.2.3 Room Model

Type: Model

Purpose: Represents room information within a hotel.

Properties:

- RoomId: int
- HotelId: int
- RoomNumber: string

- RoomType: string
- Capacity: int
- Price: decimal
- Description: string
- Amenities: List<string>
- Photos: List<string>
- IsAvailable: bool
- Hotel: Hotel (navigation property)
- Reservations: List<Reservation> (navigation property)

Methods:

```
FUNCTION IsAvailableForDates(checkInDate, checkOutDate):
    IF NOT IsAvailable THEN
        RETURN FALSE
    ENDIF

    FOREACH reservation IN Reservations DO
        // Check if there's overlap between requested dates and existing reservation
        IF (checkInDate <= reservation.CheckOutDate AND checkOutDate >=
reservation.CheckInDate) THEN
            RETURN FALSE
        ENDIF
    END FOREACH

    RETURN TRUE
END FUNCTION
```

5.3 Services

5.3.1 UserService

Type: Service

Purpose: Handles user-related business logic.

5.3.1.1 AuthenticateUser Method

Purpose: Validate user credentials

Type: Business Logic

Pseudocode:

```
FUNCTION AuthenticateUser(email, password):
    user ← UserRepository.GetByEmail(email)

    IF user IS NULL THEN
        RETURN new AuthResult { Success = FALSE, ErrorMessage = "Invalid email or
password" }
    ENDIF

    passwordValid ← VerifyPassword(password, user.PasswordHash)

    IF NOT passwordValid THEN
        RETURN new AuthResult { Success = FALSE, ErrorMessage = "Invalid email or
password" }
    ENDIF

    // Update last login date
    user.LastLoginDate ← CurrentDateTime()
    UserRepository.Update(user)

    RETURN new AuthResult {
        Success = TRUE,
        UserInfo = new UserInfo {
            UserId = user.UserId,
            Email = user.Email,
            FirstName = user.FirstName,
            UserType = user.UserType
        }
    }
END FUNCTION
```

5.3.1.2 RegisterUser Method

Purpose: Create a new user account

Type: Business Logic

Pseudocode:

```
FUNCTION RegisterUser(registerModel):
    // Check if email already exists
    existingUser ← UserRepository.GetByEmail(registerModel.Email)

    IF existingUser IS NOT NULL THEN
        RETURN new RegistrationResult { Success = FALSE, ErrorMessage = "Email
already in use" }
    ENDIF

    // Create new user
    passwordHash ← HashPassword(registerModel.Password)

    newUser ← new User {
        FirstName = registerModel.FirstName,
        LastName = registerModel.LastName,
        Email = registerModel.Email,
        PhoneNumber = registerModel.PhoneNumber,
        PasswordHash = passwordHash,
        UserType = "Customer",  // Default role
        CreatedDate = CurrentDateTime(),
        LastLoginDate = NULL
    }

    userId ← UserRepository.Add(newUser)

    RETURN new RegistrationResult { Success = TRUE, UserId = userId }
END FUNCTION
```

5.3.2 HotelService

Type: Service

Purpose: Manages hotel-related business logic.

5.3.2.1 SearchHotels Method

Purpose: Search for hotels based on criteria

Type: Business Logic

Pseudocode:

```
FUNCTION SearchHotels(location, checkInDate, checkOutDate, guests, page, pageSize):
    // Validate parameters
    IF location IS NULL OR location == "" THEN
        RETURN new SearchResult {
            Success = FALSE,
            ErrorMessage = "Location is required"
        }
    ENDIF

    IF checkInDate <= CurrentDate() THEN
        RETURN new SearchResult {
            Success = FALSE,
            ErrorMessage = "Check-in date must be in the future"
        }
    ENDIF

    IF checkOutDate <= checkInDate THEN
        RETURN new SearchResult {
            Success = FALSE,
            ErrorMessage = "Check-out date must be after check-in date"
        }
    ENDIF

    // Try to get from cache first
    cacheKey ← GenerateCacheKey(location, checkInDate, checkOutDate, guests, page,
pageSize)
    cachedResult ← CacheService.Get(cacheKey)

    IF cachedResult IS NOT NULL THEN
        RETURN cachedResult
    ENDIF

    // Search database
    hotels ← HotelRepository.SearchHotels(location, page, pageSize)
    totalCount ← HotelRepository.GetSearchResultCount(location)

    // Filter by room availability
    availableHotels ← []

    FOREACH hotel IN hotels DO
        availableRooms ← RoomRepository.GetAvailableRooms(
            hotel.HotelId,
            checkInDate,
            checkOutDate,
            guests
        )
```

```
        IF availableRooms.Count > 0 THEN
            hotel.AvailableRooms ← availableRooms
            ADD hotel TO availableHotels
        ENDIF
    END FOREACH

    result ← new SearchResult {
        Success = TRUE,
        Hotels = availableHotels,
        TotalCount = totalCount
    }

    // Cache the result
    CacheService.Set(cacheKey, result, TimeSpan.FromMinutes(5))

    RETURN result
END FUNCTION
```

5.3.3 ReservationService

Type: Service

Purpose: Handles booking-related business logic.

5.3.3.1 CreateReservation Method

Purpose: Create a new booking

Type: Business Logic

Pseudocode:

```
FUNCTION CreateReservation(userId, roomId, checkInDate, checkOutDate, guests,
totalPrice, status):
    // Validate room availability
    isAvailable ← RoomRepository.CheckAvailability(roomId, checkInDate,
checkOutDate)

    IF NOT isAvailable THEN
        RETURN new ReservationResult {
            Success = FALSE,
            ErrorMessage = "Room is no longer available for the selected dates"
        }
    ENDIF

    // Create reservation
    newReservation ← new Reservation {
        UserId = userId,
        RoomId = roomId,
        CheckInDate = checkInDate,
        CheckOutDate = checkOutDate,
        Guests = guests,
        TotalPrice = totalPrice,
        Status = status,
        CreatedDate = CurrentDateTime()
    }

    reservationId ← ReservationRepository.Add(newReservation)

    // Mark room as unavailable for these dates
    RoomRepository.UpdateAvailabilityForDates(roomId, checkInDate, checkOutDate,
FALSE)

    RETURN new ReservationResult {
        Success = TRUE,
        ReservationId = reservationId
    }
END FUNCTION
```

6. Human Interface Design

6.1 Overview of User Interface

The Prestige Hotel Reservation system provides a responsive web interface that adapts to different screen sizes and devices. The interface follows modern design principles with a clean, intuitive layout that guides users through the booking process. It allows each type of user (Customer, Hotel Admin, System Admin) to interact with the system efficiently.

The user interface is organized into several main sections:

1. **Public Pages**
   - Home page with search functionality and recommended hotels
   - Hotel search results with filtering options
   - Hotel details with room information and booking tools
   - Login and registration forms
2. **Customer Dashboard**
   - Booking history and active reservations
   - Profile management
   - Reviews and ratings submitted
   - Saved/favorite hotels
3. **Hotel Admin Panel**
   - Hotel management dashboard
   - Room management (add, edit, delete rooms)
   - Reservation management
   - Review monitoring
4. **System Admin Panel**
   - User management
   - Hotel approval queue
   - System statistics
   - Support ticket management

The interface uses a consistent navigation structure, with a main navigation bar at the top and context-specific navigation within each section. The design emphasizes clear call-to-action buttons and visual feedback for user interactions.

6.2 Screen Images

Home Page: Features a main section with search functionality, popular destinations, and featured hotels

Search Hotel Page: Displays hotels with filtering options, sorting capabilities, and detailed information cards

Login Page: Simple forms for user authentication

Create Account Page: Simple forms for account creation

Support Page: FAQ section and contact form for customer support

6.3 Screen Objects and Actions

Home Page Objects and Actions:

- Search Form: Destination input field; Check-in/check-out date pickers; Guest count selector; Search button (triggers hotel search)
- Featured Hotels Carousel: Hotel cards with images, ratings, and prices; "View Details" buttons (navigate to hotel details)
- Navigation Menu: Home, Hotels, Destinations, Special Offers, Support links; Login/Register buttons (open authentication forms)

Search Results Page Objects and Actions:

- Filter Panel: Price range slider; Star rating checkboxes; Facilities checkboxes; "Apply Filters" button (updates search results)
- Sort Dropdown: Options: Relevance, Price (low to high), Price (high to low), Rating; Changes result ordering immediately
- Hotel Cards: Hotel image, name, rating, price; "View Details" button (navigates to hotel details); Basic information display

Hotel Details Page Objects and Actions:

- Image Gallery: Main image display; Thumbnail navigation; Fullscreen view option
- Booking Widget: Room type selector; Date selection; Guest count; "Book Now" button (initiates booking process)
- Information Tabs: Overview, Rooms, Facilities, Reviews, Location; Tab switching updates displayed content

Booking Process Objects and Actions:

- Room Selection: Room type cards with details; "Select" buttons for each room type
- Guest Information Form: Name, email, phone fields; Special requests textarea
- Payment Form: Credit card fields; Billing address inputs; "Complete Booking" button (processes payment)

7. Requirements Matrix

UC-1: Login System Requirements

Functional Requirements:

Requirement ID	Requirement Description	Design Component	Implementation Details
UC1-REQ-1	Authenticate users using registered email and password	AuthController, UserService, UserRepository	Auth0 integration with JWT token generation
UC1-REQ-2	Display error message if		