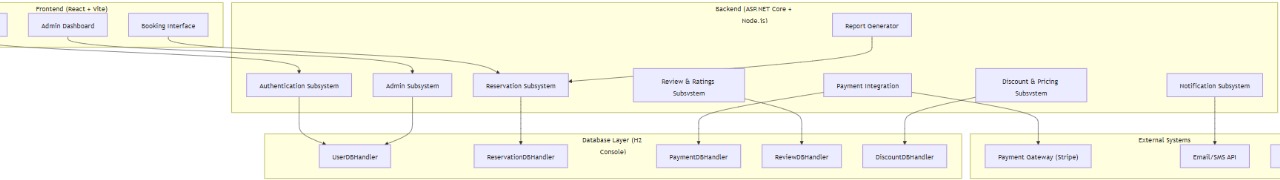
System Architecture

**Identifying Subsystems**

The Easy Park system is divided into the following key subsystems for modularity and maintainability:

1. **Authentication Subsystem**  
   Handles user registration, login, and role-based access control (user/admin).
2. **User Subsystem**  
   Manages user-facing functionality including:
   * Browsing available parking slots
   * Booking and cancelling slots
   * Viewing history and notifications
   * Submitting reviews
3. **Admin Subsystem**  
   Provides features for:
   * Parking slot management
   * Viewing/managing bookings
   * Pricing and discount configuration
   * Complaint handling and report generation
4. **Reservation and Session Subsystem**  
   Handles logic related to:
   * Creating and managing reservations
   * Tracking active parking sessions
   * Triggering expiry-based alerts
5. **Payment Subsystem**  
   Computes dynamic pricing and handles mock payment processing.
6. **Notification Subsystem**  
   Sends alerts for session expiry, confirmations, etc. (Console-based, expandable to email/SMS).
7. **Frontend UI Subsystem**  
   Built using React and Vite for all user/admin views.
8. **API Subsystem**  
   ASP.NET Web API backend exposing endpoints for all core operations.
9. **Database Subsystem**  
   Stores all data: Users, Slots, Sessions, Payments, Feedback, and Reports (H2 for development).

**UML Package Diagram**



Each of these subsystems is shown as a UML package with arrows indicating dependencies:

* UI → API → Business Logic → Database
* Notifications, Payment, and Reservation interact with API layer

**Architecture Styles**

**Key Architecture Patterns Used:**

* **Layered Architecture**
  + Presentation Layer: React frontend
  + Business Logic Layer: ASP.NET Web API services
  + Data Access Layer: Repositories and Entity Framework
  + Database Layer: H2 In-Memory
* **Client-Server Architecture**
  + React frontend (client) communicates via HTTP with ASP.NET backend (server)
* **Repository Pattern**
  + Used to abstract database logic, improving testability and separation of concerns
* **RESTful API**
  + Backend exposes REST endpoints for all actions: authentication, booking, payment, etc.

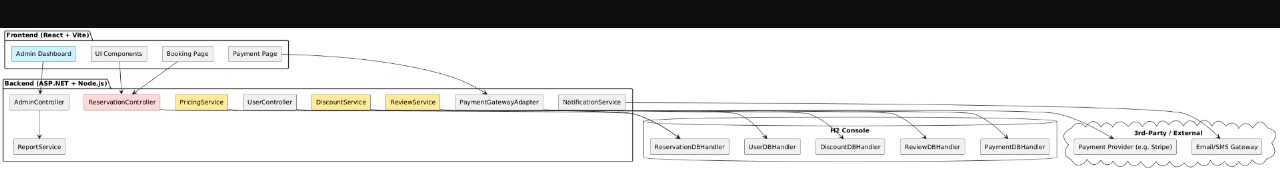
**Deployment Diagram**

A diagram of a server

AI-generated content may be incorrect.

* Client uses a browser to interact with the React frontend
* React app makes API calls to the .NET backend
* Backend accesses H2 DB for persistent storage

**Component Diagram**



**Components:**

- UserAuthService

- ReservationService

- PaymentService

- NotificationService

- AdminDashboard

- BookingManager

- PricingEngine

- ReportGenerator

- React Components (Login, Dashboard, Booking Page, Admin Panel)

These components will be presented as nodes in a UML component diagram, with interfaces like IBookingService, IAuthService and connections showing dependencies between frontend, services, and data layers.