Restaurant Reservation System API - Complete Proposal & Documentation

Executive Summary

This document outlines the design and implementation plan for a comprehensive Restaurant Reservation System Web API built using .NET C#. The system will enable restaurants to manage reservations, table availability, menu items, and customer data efficiently while providing a robust backend for web and mobile applications.

Table of Contents

---

1. Project Overview

1.1 Purpose

The Restaurant Reservation System API provides a centralized platform for managing all aspects of restaurant operations including table reservations, menu management, customer relationship management, and availability tracking.

1.2 Scope

- Online reservation booking and management
- Real-time table availability checking
- Menu item management with categories and pricing
- Customer profile management
- Reservation history and analytics
- Email/SMS notifications for confirmations and reminders
- Admin dashboard support

- Multi-restaurant support (optional phase 2)

1.3 Target Users

- Restaurant managers and staff
- Customers making reservations
- System administrators
- Third-party integration services

1.4 Technology Stack

- **Framework**: .NET 8.0 (ASP.NET Core Web API)
- **Database**: SQL Server / PostgreSQL
- **ORM**: Entity Framework Core
- **Authentication**: JWT Bearer Tokens
- **Documentation**: Swagger/OpenAPI
- **Caching**: Redis (optional)
- **Logging**: Serilog
- **Testing**: xUnit, Moq
- **Containerization**: Docker

---

2. Business Requirements

2.1 Core Business Goals

1. Reduce no-show rates through automated reminders
2. Optimize table utilization and seating capacity
3. Improve customer experience with easy booking
4. Provide data-driven insights for restaurant operations
5. Reduce manual reservation management overhead

2.2 Key Performance Indicators (KPIs)

- Reservation completion rate
- Table occupancy rate
- Average reservation duration
- Customer retention rate
- No-show percentage

- API response time (< 200ms for 95% of requests)

2.3 Business Rules

1. Reservations can be made up to 90 days in advance
2. Minimum reservation time: 30 minutes
3. Maximum party size: 12 people
4. Cancellation allowed up to 2 hours before reservation time
5. Tables can be combined for larger parties
6. Peak hours may require minimum spending or time limits
7. VIP customers get priority booking

---

3. Functional Requirements

3.1 Reservation Management

- Create, read, update, and delete reservations
- Check real-time table availability
- Automatic table assignment based on party size
- Handle special requests (allergies, occasions, seating preferences)
- Reservation status tracking (Pending, Confirmed, Seated, Completed, Cancelled, No-Show)
- Waitlist management for fully booked time slots
- Modification history tracking

3.2 Table Management

- Define tables with capacity, location, and attributes
- Mark tables as available, reserved, or occupied
- Combine tables for larger parties
- Table rotation and turnover time management
- Maintenance mode for tables under repair

3.3 Customer Management

- Customer profile creation and management
- Reservation history tracking
- Dietary restrictions and preferences
- Contact information (email, phone)

- VIP/loyalty tier management
- Blacklist functionality for problematic customers

3.4 Menu Management

- Menu item CRUD operations
- Categories (Appetizers, Main Course, Desserts, Beverages)
- Pricing and availability
- Dietary tags (Vegetarian, Vegan, Gluten-Free, etc.)
- Seasonal menus
- Image URLs for menu items

3.5 Notification System

- Email confirmations for reservations
- SMS reminders 24 hours before reservation
- Cancellation notifications
- Waitlist availability alerts
- Special occasion reminders for staff

3.6 Reporting & Analytics

- Daily reservation summary
- Table utilization reports
- Popular time slots analysis
- Customer visit frequency
- Revenue forecasting based on reservations
- No-show rate tracking

---

4. Technical Architecture

4.1 Architecture Pattern

Clean Architecture / Onion Architecture

```
┌─────────────────────────────────────┐
│   Presentation Layer (API)         │
│ - Controllers                      │
│ - Middleware                       │
│ - Filters                          │
└─────────────────────────────────────┘
        ↓
┌─────────────────────────────────────┐
│   Application Layer                │
│ - Services                         │
│ - DTOs                             │
│ - Validators                       │
│ - Mappers                          │
└─────────────────────────────────────┘
        ↓
┌─────────────────────────────────────┐
│   Domain Layer                     │
│ - Entities                         │
│ - Business Logic                   │
│ - Domain Events                    │
│ - Interfaces                       │
└─────────────────────────────────────┘
        ↓
┌─────────────────────────────────────┐
│   Infrastructure Layer             │
│ - Data Access (EF Core)            │
│ - External Services                │
│ - Email/SMS Services               │
│ - Caching                          │
└─────────────────────────────────────┘
```

4.2 Design Patterns

- **Repository Pattern**: Data access abstraction
- **Unit of Work**: Transaction management
- **Dependency Injection**: Loose coupling
- **CQRS (Optional)**: Separate read/write operations
- **Factory Pattern**: Object creation
- **Strategy Pattern**: Multiple reservation algorithms

4.3 Project Structure

```
RestaurantReservationSystem/
```

```
├── src/
│   ├── RestaurantReservation.API/
│   │   ├── Controllers/
│   │   ├── Middleware/
│   │   ├── Filters/
│   │   └── Program.cs
│   ├── RestaurantReservation.Application/
│   │   ├── Services/
│   │   ├── DTOs/
│   │   ├── Interfaces/
│   │   ├── Validators/
│   │   └── Mappings/
│   ├── RestaurantReservation.Domain/
│   │   ├── Entities/
│   │   ├── Enums/
│   │   ├── Interfaces/
│   │   └── Exceptions/
│   └── RestaurantReservation.Infrastructure/
│       ├── Data/
│       ├── Repositories/
│       ├── Services/
│       └── Migrations/
└── tests/
    ├── RestaurantReservation.UnitTests/
    └── RestaurantReservation.IntegrationTests/
```

---

5. Database Design

5.1 Entity Relationship Diagram

```
┌─────────────────┐    ┌─────────────────┐
│   Customers     │    │  Reservations   │
├─────────────────┤    ├─────────────────┤
│ CustomerId (PK) │───┐    │ ReservationId    │
│ FirstName       │   │  │ CustomerId (FK) │
│ LastName        │   └──→│ TableId (FK)    │
│ Email           │    │ ReservationDate │
│ Phone           │    │ PartySize       │
│ DateOfBirth     │    │ Status          │
│ IsVIP           │    │ SpecialRequests │
│ CreatedDate     │    │ CreatedDate     │
└─────────────────┘    │ ModifiedDate    │
                └─────────────────┘
                   │
```

```
                    │
┌─────────────────┐           │
│  Tables     │      │
├─────────────────┤           │
│ TableId (PK)   │←───────────────┘
│ TableNumber     │
│ Capacity      │
│ Location      │
│ IsAvailable     │
│ IsActive      │
└─────────────────┘


┌─────────────────┐   ┌─────────────────┐
│  MenuItems    │   │ MenuCategories │
├─────────────────┤   ├─────────────────┤
│ MenuItemId (PK) │   │ CategoryId (PK) │
│ CategoryId (FK) │───────│ Name        │
│ Name        │   │ Description   │
│ Description    │   │ DisplayOrder   │
│ Price       │   └─────────────────┘
│ IsAvailable     │
│ DietaryTags     │
│ ImageUrl      │
└─────────────────┘


┌─────────────────┐
│  Users      │
├─────────────────┤
│ UserId (PK)     │
│ Username      │
│ PasswordHash    │
│ Role        │
│ Email        │
│ IsActive      │
│ CreatedDate     │
└─────────────────┘


┌──────────────────┐
│ TimeSlots     │
├──────────────────┤
│ TimeSlotId (PK) │
│ DayOfWeek     │
│ StartTime     │
│ EndTime       │
│ MaxReservations │
│ IsActive      │
└──────────────────┘
```

```
┌──────────────────┐
│ ReservationLog   │
├──────────────────┤
│ LogId (PK)       │
│ ReservationId    │
│ Action           │
│ OldStatus        │
│ NewStatus        │
│ ChangedBy        │
│ ChangedDate      │
│ Notes            │
└──────────────────┘
```

5.2 Database Schema

Customers Table

sql

```sql
CREATE TABLE Customers (
    CustomerId INT PRIMARY KEY IDENTITY(1,1),
    FirstName NVARCHAR(100) NOT NULL,
    LastName NVARCHAR(100) NOT NULL,
    Email NVARCHAR(255) UNIQUE NOT NULL,
    Phone NVARCHAR(20) NOT NULL,
    DateOfBirth DATE,
    DietaryRestrictions NVARCHAR(500),
    IsVIP BIT DEFAULT 0,
    IsBlacklisted BIT DEFAULT 0,
    CreatedDate DATETIME2 DEFAULT GETUTCDATE(),
    ModifiedDate DATETIME2 DEFAULT GETUTCDATE()
);
```

Tables Table

sql

```sql
CREATE TABLE Tables (
    TableId INT PRIMARY KEY IDENTITY(1,1),
    TableNumber NVARCHAR(10) NOT NULL UNIQUE,
    Capacity INT NOT NULL,
    Location NVARCHAR(50),
    IsActive BIT DEFAULT 1,
    MinimumCapacity INT,
    MaximumCapacity INT,
    CreatedDate DATETIME2 DEFAULT GETUTCDATE()
);
```

Reservations Table

sql

```sql
CREATE TABLE Reservations (
    ReservationId INT PRIMARY KEY IDENTITY(1,1),
    CustomerId INT FOREIGN KEY REFERENCES Customers(CustomerId),
    TableId INT FOREIGN KEY REFERENCES Tables(TableId),
    ReservationDate DATETIME2 NOT NULL,
    PartySize INT NOT NULL,
    Status NVARCHAR(20) NOT NULL,
    SpecialRequests NVARCHAR(1000),
    Duration INT DEFAULT 120,
    IsConfirmed BIT DEFAULT 0,
    ConfirmationCode NVARCHAR(50) UNIQUE,
    CreatedDate DATETIME2 DEFAULT GETUTCDATE(),
    ModifiedDate DATETIME2 DEFAULT GETUTCDATE(),
    CancelledDate DATETIME2,
    CancellationReason NVARCHAR(500)
);
```

MenuCategories Table

sql

```sql
CREATE TABLE MenuCategories (
    CategoryId INT PRIMARY KEY IDENTITY(1,1),
    Name NVARCHAR(100) NOT NULL,
    Description NVARCHAR(500),
    DisplayOrder INT,
    IsActive BIT DEFAULT 1
);
```

MenuItems Table

```sql
CREATE TABLE MenuItems (
    MenuItemId INT PRIMARY KEY IDENTITY(1,1),
    CategoryId INT FOREIGN KEY REFERENCES MenuCategories(CategoryId),
    Name NVARCHAR(200) NOT NULL,
    Description NVARCHAR(1000),
    Price DECIMAL(10,2) NOT NULL,
    IsAvailable BIT DEFAULT 1,
    DietaryTags NVARCHAR(200),
    ImageUrl NVARCHAR(500),
    PreparationTime INT,
    CreatedDate DATETIME2 DEFAULT GETUTCDATE(),
    ModifiedDate DATETIME2 DEFAULT GETUTCDATE()
);
```

Users Table

```sql
CREATE TABLE Users (
    UserId INT PRIMARY KEY IDENTITY(1,1),
    Username NVARCHAR(100) UNIQUE NOT NULL,
    PasswordHash NVARCHAR(255) NOT NULL,
    Role NVARCHAR(50) NOT NULL,
    Email NVARCHAR(255) UNIQUE NOT NULL,
    IsActive BIT DEFAULT 1,
    CreatedDate DATETIME2 DEFAULT GETUTCDATE(),
    LastLoginDate DATETIME2
);
```

TimeSlots Table

```sql
CREATE TABLE TimeSlots (
    TimeSlotId INT PRIMARY KEY IDENTITY(1,1),
    DayOfWeek INT NOT NULL,
    StartTime TIME NOT NULL,
    EndTime TIME NOT NULL,
    MaxReservations INT,
    IsActive BIT DEFAULT 1
);
```

ReservationLog Table

```sql
CREATE TABLE ReservationLog (
    LogId INT PRIMARY KEY IDENTITY(1,1),
    ReservationId INT FOREIGN KEY REFERENCES Reservations(ReservationId),
    Action NVARCHAR(50) NOT NULL,
    OldStatus NVARCHAR(20),
    NewStatus NVARCHAR(20),
    ChangedBy INT FOREIGN KEY REFERENCES Users(UserId),
    ChangedDate DATETIME2 DEFAULT GETUTCDATE(),
    Notes NVARCHAR(1000)
);
```

---

6. API Endpoints Specification

6.1 Authentication Endpoints

POST /api/auth/register

Register a new user account.

Request Body:

```json
{
  "username": "string",
  "email": "string",
  "password": "string",
  "role": "Customer|Staff|Admin"
}
```

Response: 201 Created

```json
{
  "userId": 1,
  "username": "string",
  "email": "string",
  "token": "jwt_token"
}
```

POST /api/auth/login

Authenticate user and receive JWT token.

Request Body:

```json
{
  "username": "string",
  "password": "string"
}
```

Response: 200 OK

```json
{
  "userId": 1,
  "username": "string",
  "token": "jwt_token",
  "expiresAt": "2024-12-31T23:59:59Z"
}
```

6.2 Customer Endpoints

GET /api/customers

Get all customers (Admin only).

Query Parameters:

- pageNumber (int, default: 1)
- pageSize (int, default: 10)
- searchTerm (string, optional)

Response: 200 OK

```json
```

```json
{
  "data": [
    {
      "customerId": 1,
      "firstName": "John",
      "lastName": "Doe",
      "email": "john@example.com",
      "phone": "+1234567890",
      "isVIP": false
    }
  ],
  "pageNumber": 1,
  "pageSize": 10,
  "totalRecords": 50
}
```

GET /api/customers/{id}

Get customer by ID.

Response: 200 OK

json

```json
{
  "customerId": 1,
  "firstName": "John",
  "lastName": "Doe",
  "email": "john@example.com",
  "phone": "+1234567890",
  "dateOfBirth": "1990-05-15",
  "dietaryRestrictions": "Vegetarian",
  "isVIP": false,
  "totalReservations": 15,
  "lastReservationDate": "2024-01-10T19:00:00Z"
}
```

POST /api/customers

Create a new customer.

Request Body:

json

```
{
  "firstName": "John",
  "lastName": "Doe",
  "email": "john@example.com",
  "phone": "+1234567890",
  "dateOfBirth": "1990-05-15",
  "dietaryRestrictions": "Vegetarian"
}
```

Response: 201 Created

PUT /api/customers/{id}

Update customer information.

Request Body: (same as POST)

Response: 200 OK

DELETE /api/customers/{id}

Delete customer (soft delete).

Response: 204 No Content

6.3 Reservation Endpoints

GET /api/reservations

Get all reservations with filtering.

Query Parameters:

- date (date, optional)
- status (string, optional)
- customerId (int, optional)
- pageNumber (int, default: 1)
- pageSize (int, default: 10)

Response: 200 OK

```
json
```

```
{
  "data": [
    {
      "reservationId": 1,
      "customer": {
        "customerId": 1,
        "fullName": "John Doe",
        "phone": "+1234567890"
      },
      "table": {
        "tableId": 5,
        "tableNumber": "A5"
      },
      "reservationDate": "2024-12-25T19:00:00Z",
      "partySize": 4,
      "status": "Confirmed",
      "specialRequests": "Window seat preferred",
      "confirmationCode": "ABC123"
    }
  ],
  "pageNumber": 1,
  "pageSize": 10,
  "totalRecords": 25
}
```

GET /api/reservations/{id}

Get reservation by ID.

Response: 200 OK

POST /api/reservations

Create a new reservation.

Request Body:

```json
{
  "customerId": 1,
  "reservationDate": "2024-12-25T19:00:00Z",
  "partySize": 4,
  "duration": 120,
  "specialRequests": "Anniversary celebration",
  "preferredTableLocation": "Window"
}
```

Response: 201 Created

```json
{
  "reservationId": 1,
  "confirmationCode": "ABC123",
  "tableNumber": "A5",
  "message": "Reservation confirmed"
}
```

PUT /api/reservations/{id}

Update reservation.

Response: 200 OK

DELETE /api/reservations/{id}

Cancel reservation.

Query Parameters:

- reason (string, optional)

Response: 200 OK

```json
{
  "message": "Reservation cancelled successfully",
  "refundEligible": true
}
```

GET /api/reservations/availability

Check table availability.

Query Parameters:

- date (datetime, required)
- partySize (int, required)
- duration (int, default: 120)

Response: 200 OK

```json
```

```json
{
  "date": "2024-12-25T19:00:00Z",
  "availableSlots": [
    {
      "time": "18:00:00",
      "availableTables": 5
    },
    {
      "time": "18:30:00",
      "availableTables": 3
    },
    {
      "time": "19:00:00",
      "availableTables": 0
    }
  ]
}
```

POST /api/reservations/{id}/confirm

Confirm a pending reservation.

Response: 200 OK

POST /api/reservations/{id}/seat

Mark reservation as seated.

Response: 200 OK

POST /api/reservations/{id}/complete

Mark reservation as completed.

Response: 200 OK

POST /api/reservations/{id}/no-show

Mark reservation as no-show.

Response: 200 OK

6.4 Table Endpoints

GET /api/tables

Get all tables.

Query Parameters:

- isAvailable (bool, optional)
- minCapacity (int, optional)
- location (string, optional)

Response: 200 OK

```json
{
  "data": [
    {
      "tableId": 1,
      "tableNumber": "A1",
      "capacity": 4,
      "location": "Window",
      "isActive": true,
      "currentStatus": "Available"
    }
  ]
}
```

GET /api/tables/{id}

Get table by ID.

Response: 200 OK

POST /api/tables

Create a new table.

Request Body:

```json
{
  "tableNumber": "A1",
  "capacity": 4,
  "location": "Window",
  "minCapacity": 2,
  "maxCapacity": 6
}
```

Response: 201 Created

PUT /api/tables/{id}

Update table information.

Response: 200 OK

DELETE /api/tables/{id}

Delete table (soft delete).

Response: 204 No Content

GET /api/tables/{id}/schedule

Get table reservation schedule.

Query Parameters:

- date (date, required)

Response: 200 OK

```json
{
  "tableNumber": "A5",
  "date": "2024-12-25",
  "schedule": [
    {
      "startTime": "18:00:00",
      "endTime": "20:00:00",
      "reservationId": 123,
      "customerName": "John Doe",
      "partySize": 4
    }
  ]
}
```

6.5 Menu Endpoints

GET /api/menu/categories

Get all menu categories.

Response: 200 OK

```json
```

```
{
  "data": [
    {
      "categoryId": 1,
      "name": "Appetizers",
      "description": "Start your meal right",
      "displayOrder": 1,
      "itemCount": 12
    }
  ]
}
```

GET /api/menu/items

Get all menu items.

Query Parameters:

- categoryId (int, optional)
- isAvailable (bool, optional)
- dietaryTag (string, optional)

Response: 200 OK

```json
{
  "data": [
    {
      "menuItemId": 1,
      "name": "Caesar Salad",
      "description": "Fresh romaine lettuce with parmesan",
      "price": 12.99,
      "category": "Appetizers",
      "isAvailable": true,
      "dietaryTags": ["Vegetarian"],
      "imageUrl": "https://example.com/images/caesar-salad.jpg"
    }
  ]
}
```

GET /api/menu/items/{id}

Get menu item by ID.

Response: 200 OK

POST /api/menu/items

Create a new menu item (Admin only).

Request Body:

```json
{
  "categoryId": 1,
  "name": "Caesar Salad",
  "description": "Fresh romaine lettuce with parmesan",
  "price": 12.99,
  "dietaryTags": ["Vegetarian"],
  "preparationTime": 15,
  "imageUrl": "https://example.com/images/caesar-salad.jpg"
}
```

Response: 201 Created

PUT /api/menu/items/{id}

Update menu item.

Response: 200 OK

DELETE /api/menu/items/{id}

Delete menu item.

Response: 204 No Content

6.6 Reports Endpoints

GET /api/reports/daily-summary

Get daily reservation summary.

Query Parameters:

- date (date, required)

Response: 200 OK

```json
```

```json
{
  "date": "2024-12-25",
  "totalReservations": 45,
  "confirmedReservations": 40,
  "cancelledReservations": 3,
  "noShows": 2,
  "totalGuests": 180,
  "averagePartySize": 4,
  "tableUtilization": 85.5,
  "peakHour": "19:00"
}
```

GET /api/reports/customer-history/{customerId}

Get customer reservation history.

Response: 200 OK

GET /api/reports/revenue-forecast

Get revenue forecast based on reservations.

Query Parameters:

- startDate (date, required)
- endDate (date, required)

Response: 200 OK

---

7. Security & Authentication

7.1 Authentication Strategy

- **JWT (JSON Web Tokens)** for stateless authentication
- Token expiration: 24 hours
- Refresh token mechanism for extended sessions
- HTTPS required for all API calls

7.2 Authorization Levels

1. **Public**: No authentication required (menu viewing)
2. **Customer**: Authenticated customers (make/view own reservations)
3. **Staff**: Restaurant staff (manage reservations, update status)

4. **Admin**: Full system access

7.3 Security Measures

- Password hashing using BCrypt
- Rate limiting (100 requests per minute per IP)
- SQL injection prevention via parameterized queries
- XSS protection
- CORS configuration
- API key for third-party integrations
- Input validation using FluentValidation
- Audit logging for sensitive operations

7.4 Data Privacy

- GDPR compliance
- Customer data encryption at rest
- PII (Personally Identifiable Information) masking in logs
- Right to be forgotten implementation
- Data retention policy (7 years for reservations)

---

8. Implementation Plan

8.1 Phase 1: Foundation (Weeks 1-2)

Deliverables:

- Project structure setup
- Database schema implementation
- Entity models and DbContext
- Repository pattern implementation
- Basic authentication (JWT)

Tasks:

- Initialize .NET 8 Web API project
- Set up Entity Framework Core
- Create database migrations
- Implement generic repository

- Create user authentication endpoints

8.2 Phase 2: Core Features (Weeks 3-5)

Deliverables:

- Customer management
- Table management
- Basic reservation CRUD
- Availability checking logic

Tasks:

- Implement customer endpoints
- Implement table endpoints
- Create reservation service
- Build availability algorithm
- Add validation rules

8.3 Phase 3: Advanced Features (Weeks 6-8)

Deliverables:

- Menu management
- Reservation status workflow
- Automated table assignment
- Search and filtering

Tasks:

- Implement menu endpoints
- Create status transition logic
- Build table assignment algorithm
- Add pagination and filtering
- Implement search functionality

8.4 Phase 4: Notifications & Integrations (Weeks 9-10)

Deliverables:

- Email notifications
- SMS notifications (optional)

- Logging system
- Background jobs

Tasks:

- Integrate email service (SendGrid/SMTP)
- Set up Hangfire for background jobs
- Implement notification templates
- Create scheduled reminder jobs
- Add comprehensive logging

8.5 Phase 5: Reporting & Analytics (Week 11)

Deliverables:

- Reporting endpoints
- Dashboard data APIs
- Analytics calculations

Tasks:

- Implement report generation
- Create aggregation queries
- Build analytics endpoints
- Add caching for reports

8.6 Phase 6: Testing & Documentation (Week 12)

Deliverables:

- Unit tests
- Integration tests
- API documentation
- Deployment scripts

Tasks:

- Write unit tests (80% coverage)
- Create integration tests
- Complete Swagger documentation
- Create Docker configuration
- Write deployment guide

9. Testing Strategy

9.1 Unit Testing

Tools: xUnit, Moq, FluentAssertions

Coverage Areas:

- Service layer business logic
- Validation rules
- Domain entities
- Utility functions

Example Test:

```csharp
[Fact]
public async Task CreateReservation_ValidData_ReturnsSuccess()
{
    // Arrange
    var mockRepo = new Mock<IReservationRepository>();
    var service = new ReservationService(mockRepo.Object);

    // Act
    var result = await service.CreateReservationAsync(validReservation);

    // Assert
    result.Should().NotBeNull();
    result.IsSuccess.Should().BeTrue();
}
```

9.2 Integration Testing

Tools: WebApplicationFactory, TestContainers

Coverage Areas:

- API endpoint responses
- Database operations
- Authentication flows
- End-to-end scenarios

9.3 Performance Testing

Tools: JMeter, k6

Metrics:

- Response time under load
- Concurrent user capacity
- Database query performance
- API throughput

Targets:

- 95th percentile response time < 200ms
- Support 1000 concurrent users
- 99.9% uptime

9.4 Test Data Management

- Seed data for development environment
- Faker library for generating test data
- Separate test database
- Automated database cleanup

---

10. Deployment & Maintenance

10.1 Deployment Strategy

Environment Pipeline:

1. **Development**: Local developer machines
2. **Testing**: Automated test environment
3. **Staging**: Pre-production mirror
4. **Production**: Live environment

CI/CD Pipeline:

- GitHub Actions / Azure DevOps
- Automated testing on commit
- Docker container builds
- Blue-green deployment

10.2 Hosting Options

Recommended:

- **Azure App Service** (PaaS)
- **Azure SQL Database**
- **Azure Redis Cache**
- **Azure Key Vault** for secrets

Alternative:

- AWS Elastic Beanstalk
- Google Cloud Run
- Self-hosted Kubernetes

10.3 Monitoring & Logging

Tools:

- Application Insights / Datadog
- Serilog for structured logging
- Health check endpoints
- Performance counters

Metrics to Track:

- API response times
- Error rates
- Database performance
- User activity
- Resource utilization

10.4 Backup & Disaster Recovery

- Daily automated database backups
- Point-in-time recovery capability
- Backup retention: 30 days
- Geographic redundancy
- Recovery Time Objective (RTO): 1 hour
- Recovery Point Objective (RPO): 15 minutes

10.5 Maintenance Plan

Regular Tasks:

- Weekly security updates
- Monthly dependency updates
- Quarterly performance reviews
- Annual security audits

Version Management:

- Semantic versioning (MAJOR.MINOR.PATCH)
- API versioning in URL (/api/v1/...)
- Deprecation notices (6 months before removal)
- Changelog maintenance

---

11. API Response Standards

11.1 Success Responses

json

```json
{
  "success": true,
  "data": { },
  "message": "Operation completed successfully",
  "timestamp": "2024-01-15T10:30:00Z"
}
```

11.2 Error Responses

json

```json
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invali
```