HIV Clinic Management System

Software Design Specification

Version 1.0

– Ho Chi Minh City, January 2025 –

Record of Changes

Date	A*M, D	In charge	Change Description
2025-01-06	A	Do Dang Khoa	Initial creation of SDS document
2025-01-06	A,M	Pham Phuoc An	Added comprehensive system architecture
2025-01-06	A,M	Tao Minh Tuan	Added detailed class diagrams and specifications
2025-01-06	A,M	Nguyen Thanh Dat	Added database design and API documentation

*A - Added, M - Modified, D - Deleted

Contents

1 Overview

1.1 Code Packages

The HIV Clinic Management System follows a layered architecture pattern with clear separation of concerns. The backend is built using Spring Boot framework with Java 17, while the frontend uses React with modern JavaScript (ES6+). The system implements a comprehensive package structure that promotes maintainability and scalability.

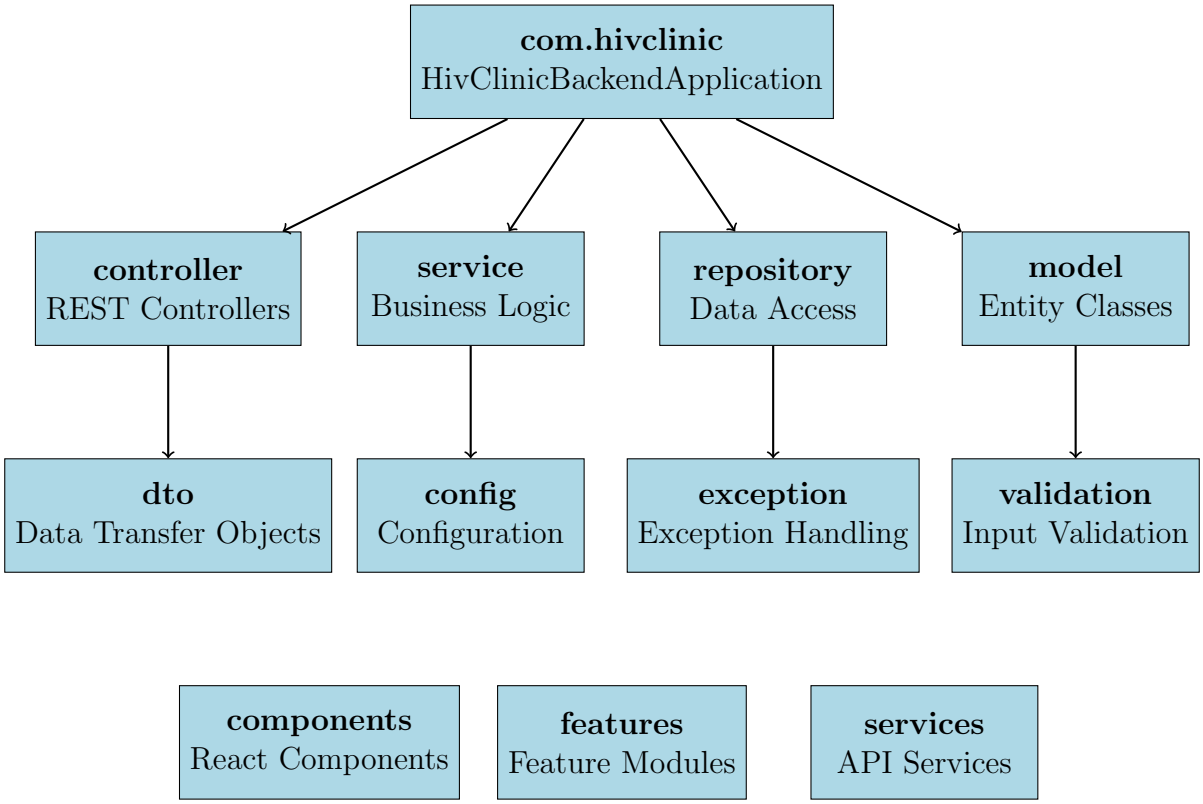


Figure 1: Package Structure Overview

Table 2: Package Descriptions

No	Package	Description
01	com.hivclinic.controller	REST API endpoints handling HTTP requests and responses. Contains controllers for authentication, appointments, user management, and ARV treatments.
02	com.hivclinic.service	Business logic layer implementing core application functionality. Handles appointment scheduling, notification management, and user authentication.
03	com.hivclinic.repository	Data access layer using Spring Data JPA. Provides database operations for all entity classes with custom queries for complex operations.
04	com.hivclinic.model	JPA entity classes representing database tables. Includes User, Appointment, ARVTreatment, Notification, and related entities.
05	com.hivclinic.dto	Data Transfer Objects for API communication. Separate request and response DTOs for clean API contracts.
06	com.hivclinic.config	Spring configuration classes including security, JWT, CORS, and database configuration.
07	com.hivclinic.exception	Custom exception classes and global exception handling for consistent error responses.
08	com.hivclinic.validation	Custom validation annotations and validators for input validation.
09	components	React components organized by functionality: layout, notifications, scheduling, and ARV treatment management.
10	features	Feature-based React modules for Admin, Doctor, Patient, and Manager dashboards.
11	services	Frontend API service layer for HTTP communication with the backend REST API.

1.2 Database Design

1.2.1 Database Schema

The HIV Clinic Management System uses Microsoft SQL Server as the primary database. The schema follows 3rd Normal Form (3NF) design principles with proper referential integrity constraints.

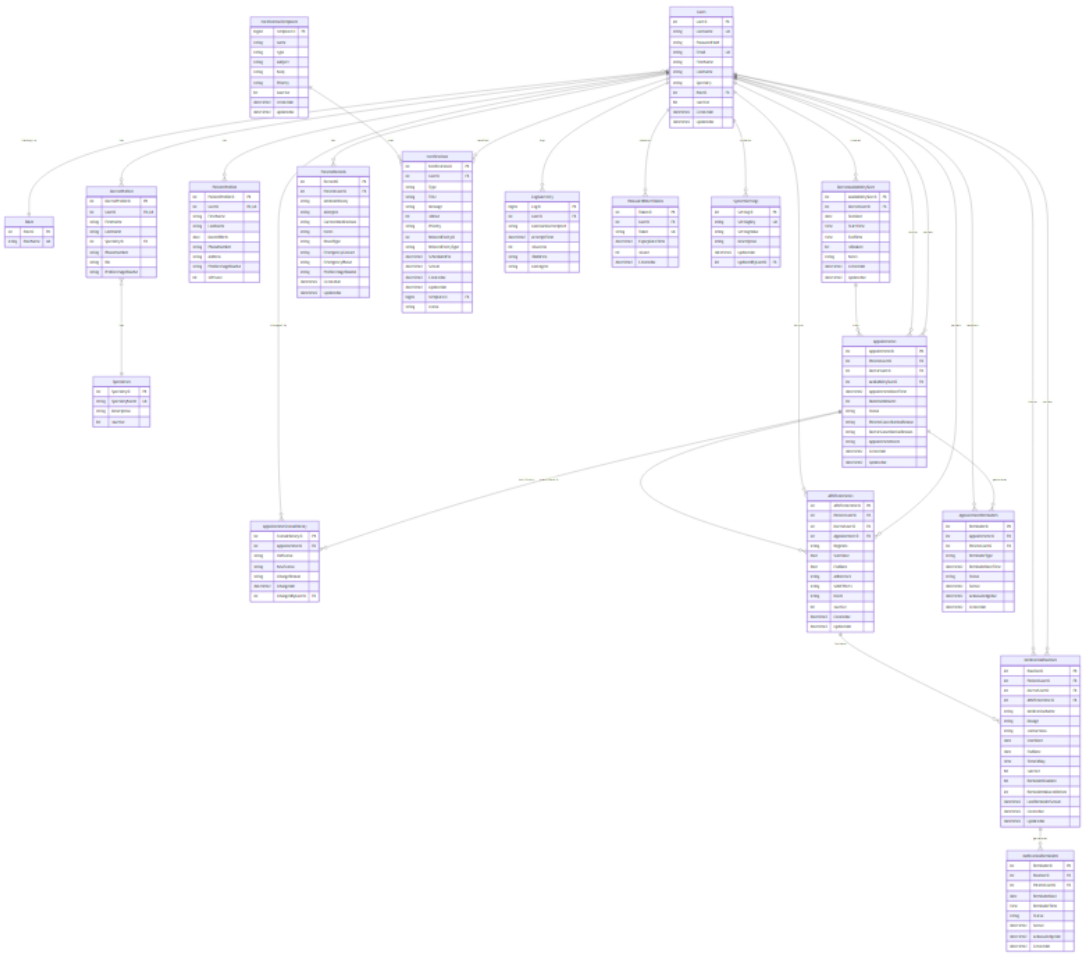


Figure 2: Database Entity Relationship Diagram

1.2.2 Table Description

No	Table	Description
01	Users	Core user table storing authentication credentials and basic user information for all user types (Admin, Manager, Doctor, Patient)
02	Roles	Role-based access control definitions with hierarchical permissions (Admin ¿ Manager ¿ Doctor ¿ Patient)
03	DoctorProfiles	Extended profile information for doctor users including specialties, bio, and profile images
04	PatientProfiles	Extended profile information for patient users including demographics and privacy settings
05	Specialties	Medical specialties reference table for categorizing doctors
06	DoctorAvailabilitySlots	Doctor's available time slots for appointment booking with date, time, and booking status
07	Appointments	Appointment records linking patients and doctors with scheduling and status information
08	PatientRecords	Comprehensive medical records including history, allergies, medications, and emergency contacts
09	ARVTreatments	Antiretroviral treatment records with regimen details, adherence tracking, and side effects

10	Notifications	System notifications for appointment reminders, medication alerts, and general communications
11	NotificationTemplates	Reusable notification templates for different notification types with customizable content
12	MedicationRoutines	Daily medication schedules for patients with reminder settings and timing
13	MedicationReminders	Individual medication reminder instances with status tracking
14	AppointmentReminders	Appointment reminder instances with multiple reminder types (24h, 1h, 30min)
15	AppointmentStatusHistory	Audit trail for appointment status changes with timestamps and reasons
16	LoginActivity	Security audit log for user authentication attempts and session management
17	PasswordResetTokens	Secure password reset token management with expiration and usage tracking
18	SystemSettings	Configurable system parameters for application behavior and feature flags

2 Code Designs

2.1 User Authentication and Authorization

This section details the comprehensive authentication and authorization system implemented using Spring Security with JWT tokens.

2.1.1 Class Diagram

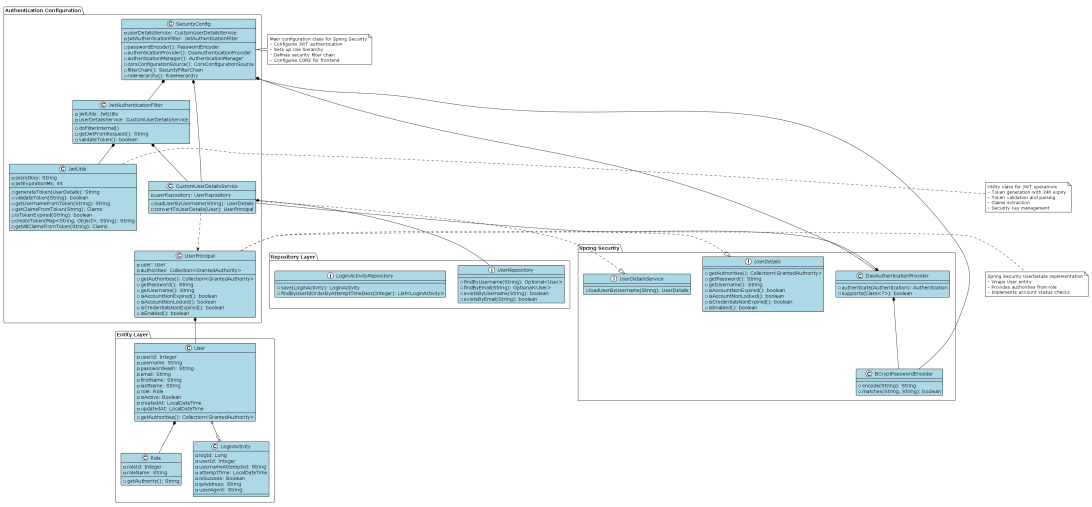


Figure 3: Authentication System Class Diagram

2.1.2 Class Specifications

SecurityConfig Class

No	Method	Description
01	passwordEncoder()	Returns BCryptPasswordEncoder instance for secure password hashing with salt

02	authenticationProvider()	Configures DaoAuthenticationProvider with custom UserDetailsService and password encoder
03	authenticationManager()	Provides AuthenticationManager bean for handling authentication requests
04	corsConfigurationSource()	Configures CORS settings to allow cross-origin requests from frontend applications
05	filterChain()	Defines security filter chain with JWT authentication, role-based access control, and endpoint security
06	roleHierarchy()	Establishes role hierarchy: ADMIN ¿ MANAGER ¿ DOCTOR ¿ PATIENT

JwtUtils Class

No	Method	Description
01	generateToken()	Generates JWT token with user details, roles, and expiration time (24 hours)
02	validateToken()	Validates JWT token signature, expiration, and claims integrity
03	getUsernameFromToken()	Extracts username from JWT token claims for authentication
04	getClaimsFromToken()	Extracts all claims from JWT token for authorization decisions
05	isTokenExpired()	Checks if JWT token has expired based on expiration claim

CustomUserDetailsService Class

No	Method	Description
01	loadUserByUsername()	Loads user details from database by username for Spring Security authentication
02	UserPrincipal()	Inner class implementing UserDetails interface with user information and authorities
03	getAuthorities()	Returns granted authorities based on user role for authorization
04	isAccountNonExpired()	Returns account expiration status based on user active status
05	isAccountNonLocked()	Returns account lock status for security purposes
06	isCredentialsNonExpired()	Returns credential expiration status
07	isEnabled()	Returns user enabled status based on isActive flag

2.1.3 Sequence Diagram

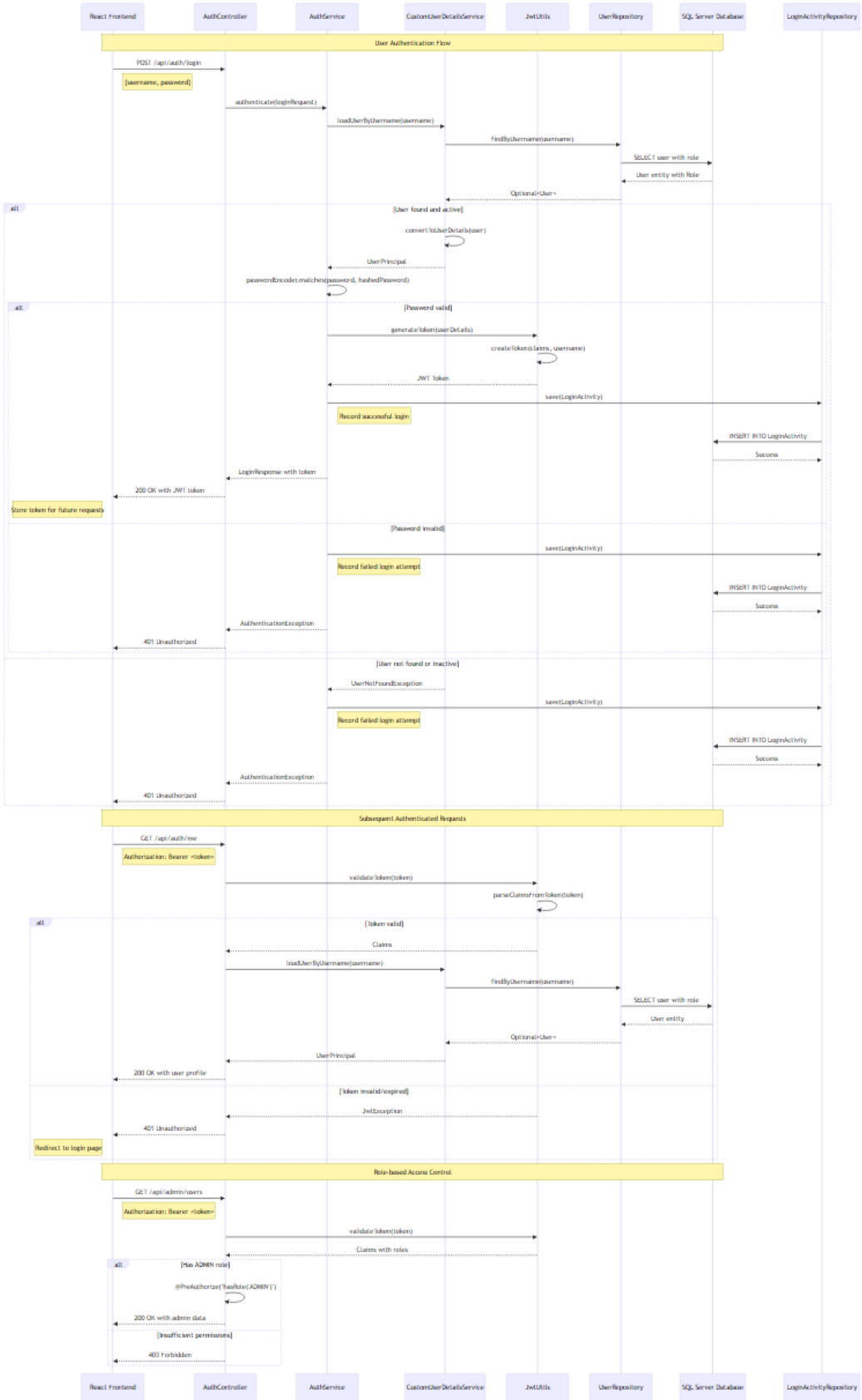


Figure 4: Authentication Sequence Diagram

2.1.4 Database Queries

Listing 1: User Authentication Queries

```sql
-- User login validation
SELECT u.UserID, u.Username, u.PasswordHash, u.Email, u.FirstName
    , u.LastName,
        u.IsActive, r.RoleName
FROM Users u
INNER JOIN Roles r ON u.RoleID = r.RoleID
WHERE u.Username = ? AND u.IsActive = 1;

-- Record login activity
INSERT INTO LoginActivity (UserID, UsernameAttempted, IsSuccess,
    IPAddress, UserAgent)
VALUES (?, ?, ?, ?, ?);

-- Update last login timestamp
UPDATE Users SET UpdatedAt = GETDATE() WHERE UserID = ?;

-- Password reset token generation
INSERT INTO PasswordResetTokens (UserID, Token, ExpiryDateTime)
VALUES (?, ?, DATEADD(hour, 24, GETDATE()));
```

2.2 Appointment Management System

This section details the comprehensive appointment booking and management system.

2.2.1 Class Diagram

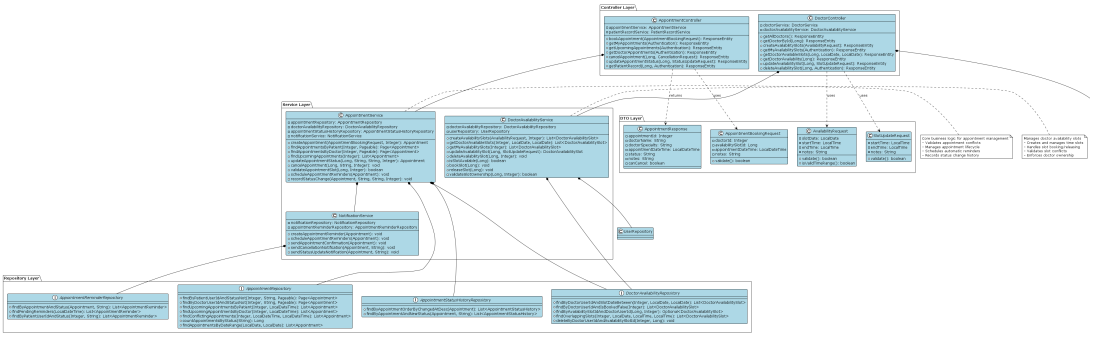


Figure 5: Appointment Management Class Diagram

2.2.2 Class Specifications

AppointmentController Class

No	Method	Description
01	bookAppointment()	POST /api/appointments/book - Creates new appointment with patient, doctor, and time slot validation

02	getMyAppointments()	GET /api/appointments/patient/my-appointments - Retrieves all appointments for authenticated patient
03	getUpcomingAppointments()	GET /api/appointments/patient/upcoming - Retrieves upcoming appointments for patient dashboard
04	getDoctorAppointments()	GET /api/appointments/doctor/my-appointments - Retrieves appointments for authenticated doctor
05	cancelAppointment()	PUT /api/appointments/{id}/cancel - Cancels appointment with reason and status update
06	updateAppointmentStatus()	PUT /api/appointments/{id}/status - Updates appointment status (Completed, No-show, etc.)
07	getPatientRecord()	GET /api/appointments/{id}/patient-record - Retrieves patient record for appointment context

AppointmentService Class

No	Method	Description
01	createAppointment()	Creates new appointment with slot validation, conflict checking, and notification scheduling
02	findAppointmentsByPatient()	Retrieves appointments for specific patient with filtering and pagination
03	findAppointmentsByDoctor()	Retrieves appointments for specific doctor with date range filtering
04	updateAppointmentStatus()	Updates appointment status with history tracking and notification triggers
05	cancelAppointment()	Cancels appointment, frees time slot, and sends cancellation notifications
06	validateAppointmentSlot()	Validates time slot availability and conflicts before booking
07	scheduleAppointmentReminders()	Schedules automatic reminders (24h, 1h, 30min before appointment)

Appointment Entity Class

No	Method/Field	Description
01	appointmentId	Primary key with auto-increment identity
02	patientUserID	Foreign key reference to Users table for patient
03	doctorUserID	Foreign key reference to Users table for doctor
04	availabilitySlotID	Foreign key reference to DoctorAvailabilitySlots table
05	appointmentDateTime	Scheduled date and time for the appointment
06	status	Current appointment status (Scheduled, Completed, Cancelled, No-show)
07	appointmentNotes	Notes and comments from doctor or patient
08	prePersist()	Sets creation timestamp before entity persistence
09	preUpdate()	Updates modification timestamp before entity update

2.2.3 Sequence Diagram

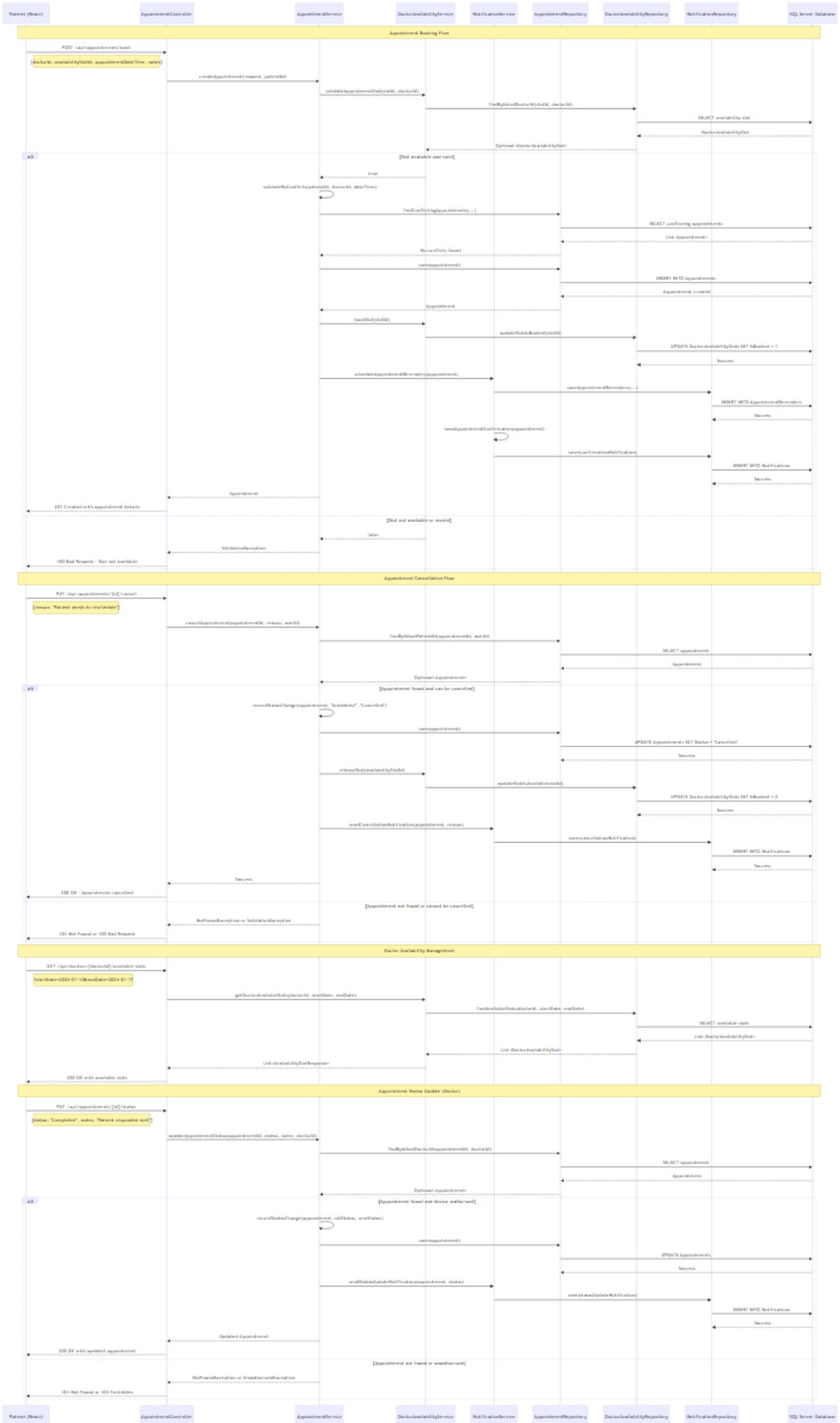


Figure 6: Appointment Booking Sequence Diagram

2.2.4 Database Queries

Listing 2: Appointment Management Queries

```
-- Create new appointment
INSERT INTO Appointments (PatientUserID, DoctorUserID,
    AvailabilitySlotID,
                          AppointmentDateTime, Status,
                             AppointmentNotes)
VALUES (?, ?, ?, ?, 'Scheduled', ?);

-- Update availability slot as booked
UPDATE DoctorAvailabilitySlots
SET IsBooked = 1, UpdatedAt = GETDATE()
WHERE AvailabilitySlotID = ?;

-- Get patient appointments with doctor details
SELECT a.*, d.FirstName as DoctorFirstName, d.LastName as
    DoctorLastName,
        s.SpecialtyName, das.SlotDate, das.StartTime, das.EndTime
FROM Appointments a
INNER JOIN Users du ON a.DoctorUserID = du.UserID
INNER JOIN DoctorProfiles d ON du.UserID = d.UserID
LEFT JOIN Specialties s ON d.SpecialtyID = s.SpecialtyID
LEFT JOIN DoctorAvailabilitySlots das ON a.AvailabilitySlotID =
    das.AvailabilitySlotID
WHERE a.PatientUserID = ? AND a.Status != 'Cancelled'
ORDER BY a.AppointmentDateTime DESC;

-- Cancel appointment and free slot
UPDATE Appointments
SET Status = 'Cancelled', PatientCancellationReason = ?,
    UpdatedAt = GETDATE()
WHERE AppointmentID = ?;

UPDATE DoctorAvailabilitySlots
SET IsBooked = 0, UpdatedAt = GETDATE()
WHERE AvailabilitySlotID = (SELECT AvailabilitySlotID FROM
    Appointments WHERE AppointmentID = ?);

-- Record status change history
INSERT INTO AppointmentStatusHistory (AppointmentID, OldStatus,
    NewStatus, ChangeReason, ChangedByUserID)
VALUES (?, ?, ?, ?, ?);
```

2.3 Notification Management System

This section details the comprehensive notification system for appointment reminders, medication alerts, and general communications.

2.3.1 Class Diagram

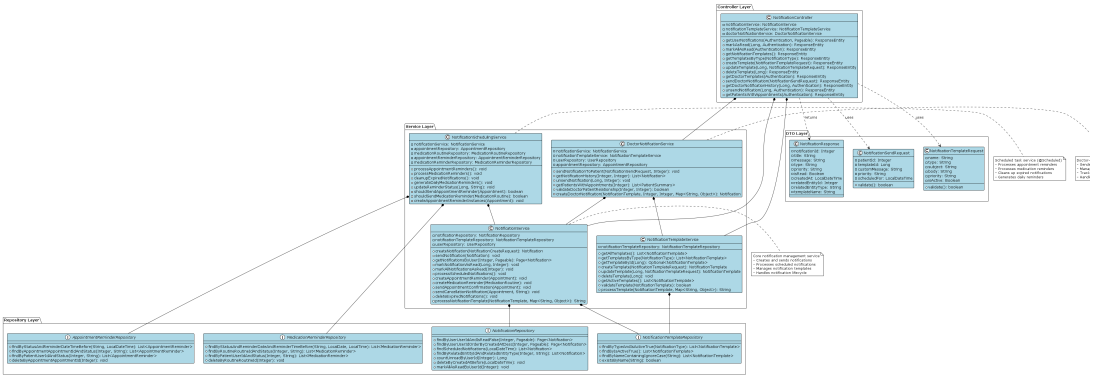


Figure 7: Notification System Class Diagram

2.3.2 Class Specifications

NotificationController Class

No	Method	Description
01	getUserNotifications()	GET /api/notifications - Retrieves paginated notifications for authenticated user
02	markAsRead()	POST /api/notifications/{id}/read - Marks specific notification as read
03	markAllAsRead()	POST /api/notifications/read-all - Marks all user notifications as read
04	getNotificationTemplates()	GET /api/notifications/templates - Retrieves available notification templates
05	getTemplatesByType()	GET /api/notifications/templates/{type} - Retrieves templates by notification type
06	createTemplate()	POST /api/notifications/templates - Creates new notification template
07	updateTemplate()	PUT /api/notifications/templates/{id} - Updates existing notification template
08	deleteTemplate()	DELETE /api/notifications/templates/{id} - Deletes notification template
09	sendDoctorNotification()	POST /api/notifications/doctor/send - Sends notification from doctor to patient
10	getDoctorNotificationHistory()	GET /api/notifications/doctor/history/{patientId} - Retrieves notification history

NotificationService Class

No	Method	Description
01	createNotification()	Creates new notification with template processing and scheduling
02	sendNotification()	Sends notification to user with delivery tracking
03	scheduleAppointmentReminders()	Schedules automatic appointment reminders (24h, 1h, 30min)
04	scheduleMedicationReminders()	Schedules daily medication reminders based on routines
05	processScheduledNotifications()	Processes pending notifications scheduled for delivery
06	markNotificationAsRead()	Marks notification as read and updates timestamp

07	getNotificationsByUser()	Retrieves paginated notifications for specific user
08	deleteExpiredNotifications()	Cleanup job for removing old notifications

NotificationSchedulingService Class

No	Method	Description
01	processAppointmentReminders()	Scheduled task (@Scheduled) for processing appointment reminders
02	processMedicationReminders()	Scheduled task for processing medication reminders
03	cleanupExpiredNotifications()	Scheduled cleanup of old and expired notifications
04	generateDailyMedicationReminders()	Generates daily medication reminders for active routines
05	updateReminderStatus()	Updates reminder status after successful delivery

2.3.3 Sequence Diagram

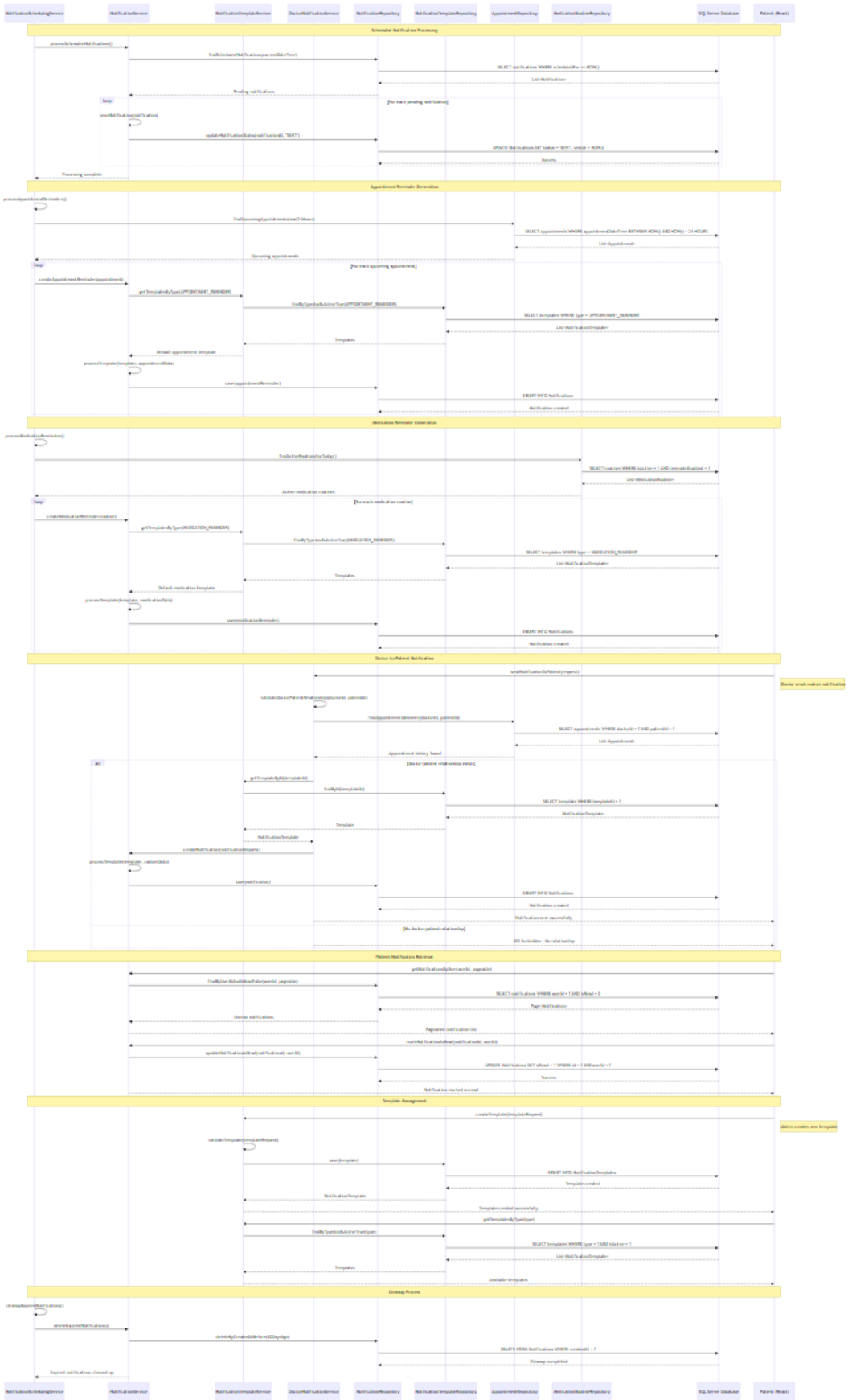


Figure 8: Notification Processing Sequence Diagram

2.3.4 Database Queries

Listing 3: Notification Management Queries

```sql
-- Create new notification
INSERT INTO Notifications (UserID, Type, Title, Message, Priority,
                           RelatedEntityID, RelatedEntityType,
                              ScheduledFor, templateId)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);

-- Get user notifications with pagination
SELECT n.*, nt.name as TemplateName
FROM Notifications n
LEFT JOIN NotificationTemplates nt ON n.templateId = nt.
   templateId
WHERE n.UserID = ? AND n.IsRead = 0
ORDER BY n.Priority DESC, n.CreatedAt DESC
OFFSET ? ROWS FETCH NEXT ? ROWS ONLY;

-- Schedule appointment reminders
INSERT INTO AppointmentReminders (AppointmentID, PatientUserID,
   ReminderType, ReminderDateTime)
SELECT AppointmentID, PatientUserID, '24_HOUR', DATEADD(hour,
   -24, AppointmentDateTime)
FROM Appointments
WHERE AppointmentDateTime > GETDATE() AND Status = 'Scheduled';

-- Process scheduled notifications
UPDATE Notifications
SET SentAt = GETDATE(), status = 'SENT'
WHERE ScheduledFor <= GETDATE() AND status = 'PENDING';

-- Mark notification as read
UPDATE Notifications
SET IsRead = 1, UpdatedAt = GETDATE()
WHERE NotificationID = ? AND UserID = ?;

-- Get notification templates by type
SELECT * FROM NotificationTemplates
WHERE type = ? AND isActive = 1
ORDER BY name;

-- Create medication reminder instances
INSERT INTO MedicationReminders (RoutineID, PatientUserID,
   ReminderDate, ReminderTime)
SELECT RoutineID, PatientUserID, ?, TimeOfDay
FROM MedicationRoutines
WHERE IsActive = 1 AND ReminderEnabled = 1;
```

2.4 ARV Treatment Management

This section details the Antiretroviral (ARV) treatment management system for HIV patients.

2.4.1 Class Diagram

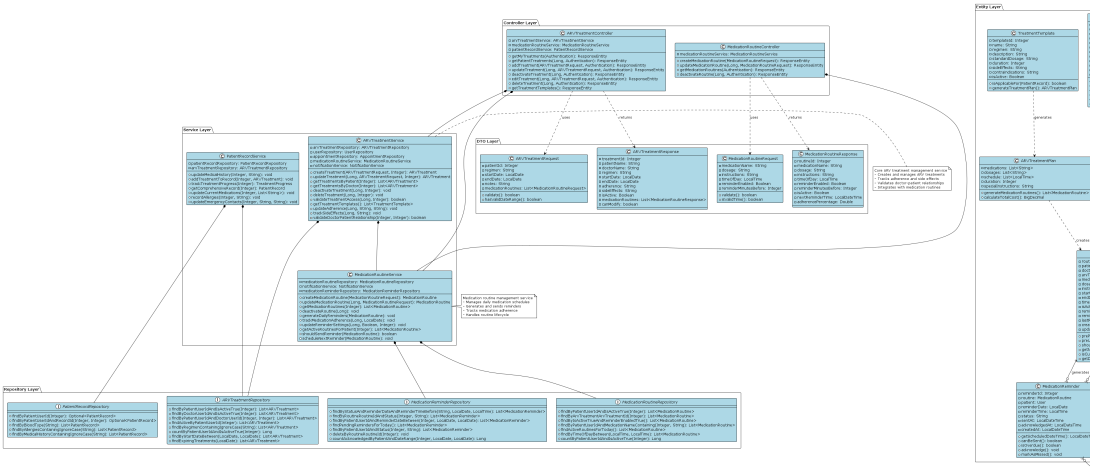


Figure 9: ARV Treatment Management Class Diagram

2.4.2 Class Specifications

ARVTreatmentController Class

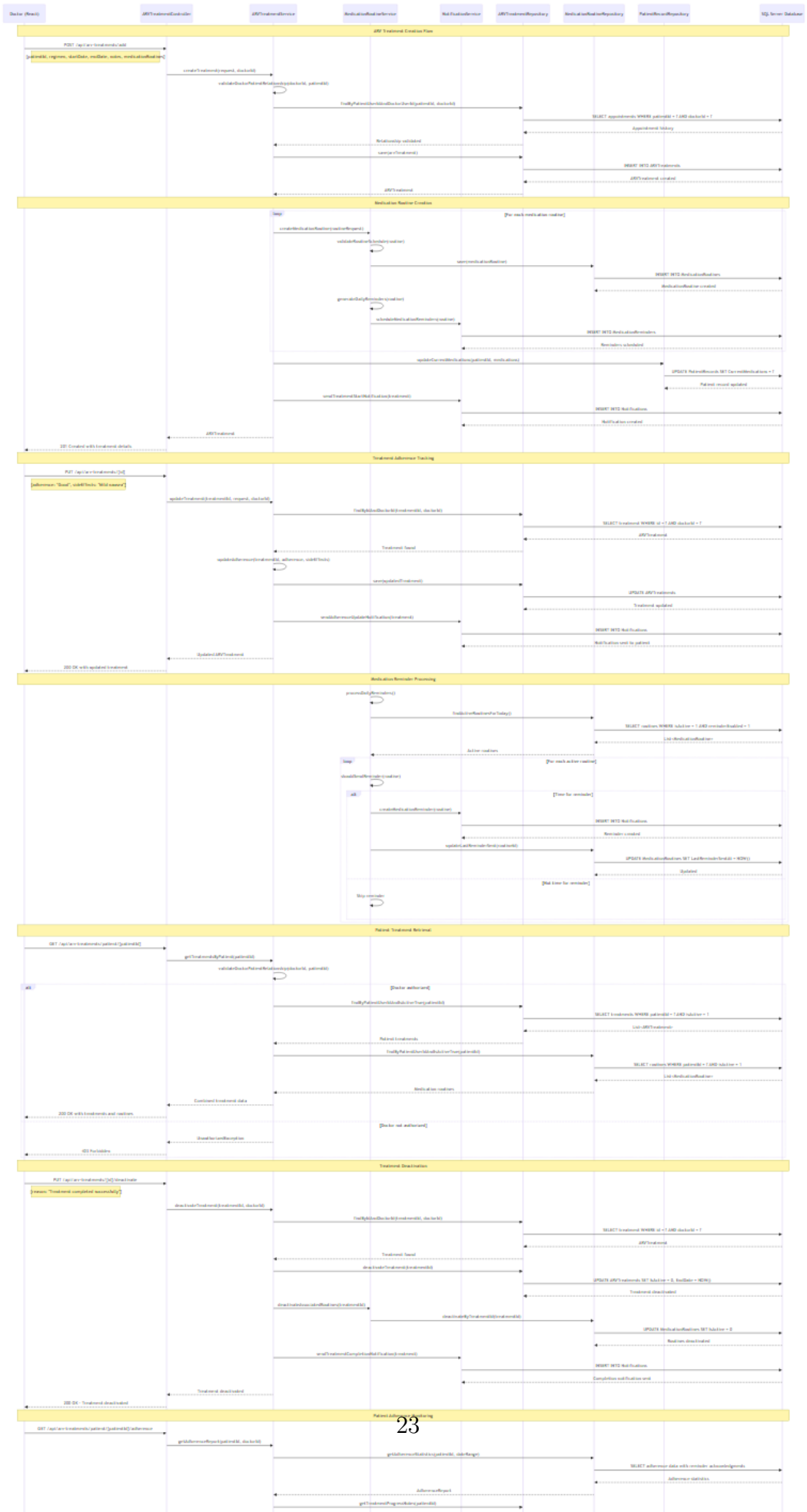
No	Method	Description
01	getMyTreatments()	GET /api/arv-treatments/my-treatments - Retrieves ARV treatments for authenticated patient
02	getPatientTreatments()	GET /api/arv-treatments/patient/{patientId} - Retrieves treatments for specific patient (doctor access)
03	addTreatment()	POST /api/arv-treatments/add - Creates new ARV treatment record
04	updateTreatment()	PUT /api/arv-treatments/{treatmentId} - Updates existing ARV treatment
05	deactivateTreatment()	PUT /api/arv-treatments/{treatmentId}/deactivate - Deactivates treatment (end of regimen)
06	editTreatment()	PUT /api/arv-treatments/{treatmentId}/edit - Edits treatment details and notes
07	deleteTreatment()	DELETE /api/arv-treatments/{treatmentId} - Deletes treatment record
08	getTreatmentTemplates()	GET /api/arv-treatments/templates - Retrieves standard treatment regimen templates

MedicationRoutineService Class

No	Method	Description
01	createMedicationRoutine()	Creates new daily medication routine with reminder scheduling
02	updateMedicationRoutine()	Updates existing medication routine and reschedules reminders
03	getMedicationRoutines()	Retrieves active medication routines for patient
04	deactivateRoutine()	Deactivates medication routine when treatment ends

05	generateDailyReminders()	Generates daily medication reminders based on routine schedule
06	trackMedicationAdherence()	Tracks medication adherence based on reminder acknowledgments

2.4.3 Sequence Diagram



2.4.4 Database Queries

Listing 4: ARV Treatment Management Queries

```
-- Create ARV treatment
INSERT INTO ARVTreatments (PatientUserID, DoctorUserID, AppointmentID,
                           Regimen, StartDate, EndDate, Notes, IsActive)
VALUES (?, ?, ?, ?, ?, ?, ?, 1);

-- Get patient treatments with doctor details
SELECT at.*, d.FirstName as DoctorFirstName, d.LastName as DoctorLastName,
       p.FirstName as PatientFirstName, p.LastName as PatientLastName
FROM ARVTreatments at
INNER JOIN Users du ON at.DoctorUserID = du.UserID
INNER JOIN DoctorProfiles d ON du.UserID = d.UserID
INNER JOIN Users pu ON at.PatientUserID = pu.UserID
INNER JOIN PatientProfiles p ON pu.UserID = p.UserID
WHERE at.PatientUserID = ? AND at.IsActive = 1
ORDER BY at.CreatedAt DESC;

-- Create medication routine
INSERT INTO MedicationRoutines (PatientUserID, DoctorUserID, ARVTreatmentID,
                                MedicationName, Dosage, Instructions, StartDate,
                                EndDate, TimeOfDay, ReminderEnabled,
                                ReminderMinutesBefore)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, 1, 30);

-- Update treatment adherence
UPDATE ARVTreatments
SET Adherence = ?, SideEffects = ?, Notes = ?, UpdatedAt = GETDATE()
WHERE ARVTreatmentID = ?;

-- Get active medication routines for reminders
SELECT mr.*, p.FirstName, p.LastName, u.Email
FROM MedicationRoutines mr
INNER JOIN Users u ON mr.PatientUserID = u.UserID
INNER JOIN PatientProfiles p ON u.UserID = p.UserID
WHERE mr.IsActive = 1 AND mr.ReminderEnabled = 1
  AND mr.StartDate <= GETDATE()
  AND (mr.EndDate IS NULL OR mr.EndDate >= GETDATE());
```

2.5 System Architecture Components

This section provides an overview of the complete system architecture including frontend-backend integration.

2.5.1 System Architecture Diagram

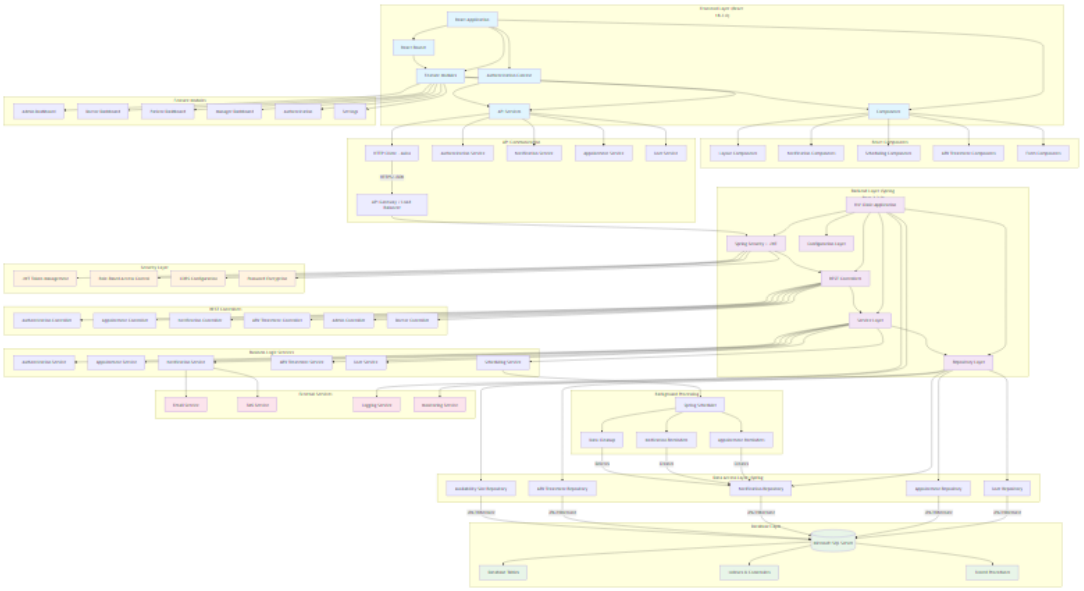


Figure 11: System Architecture Overview

2.5.2 Technology Stack

Table 15: Technology Stack Components

Layer	**Technology**	**Description**
Frontend	React 18.2.0	Modern JavaScript library for building user interfaces with hooks and functional components
Frontend Routing	React Router 6.x	Declarative routing for React applications with nested routes and authentication guards
State Management	React Context API	Built-in state management for authentication and global application state
HTTP Client	Axios	Promise-based HTTP client for API communication with interceptors for authentication
Backend Framework	Spring Boot 3.2.0	Java-based framework for building enterprise applications with auto-configuration
Security	Spring Security 6.x	Comprehensive security framework with JWT authentication and role-based access control
Data Access	Spring Data JPA	Abstraction layer for database operations with Hibernate ORM
Database	Microsoft SQL Server	Enterprise-grade relational database with T-SQL support
Build Tool	Maven 3.9.x	Project management and build automation tool for Java applications
Java Version	OpenJDK 17	Long-term support version of Java with modern language features

2.5.3 Security Architecture

Table 16: Security Architecture Components

Component	Description
JWT Authentication	JSON Web Token-based stateless authentication with 24-hour expiration
Role-Based Access Control	Hierarchical role system: Admin ¿ Manager ¿ Doctor ¿ Patient
Password Security	BCrypt hashing with salt for secure password storage
CORS Configuration	Cross-Origin Resource Sharing configuration for frontend-backend communication
Input Validation	Comprehensive input validation using Bean Validation (JSR-303) annotations
SQL Injection Prevention	Parameterized queries and JPA/Hibernate ORM protection
Session Management	Stateless session management with JWT tokens
API Endpoint Security	Method-level security with @PreAuthorize annotations

2.6 Component Architecture

2.6.1 Component Diagram

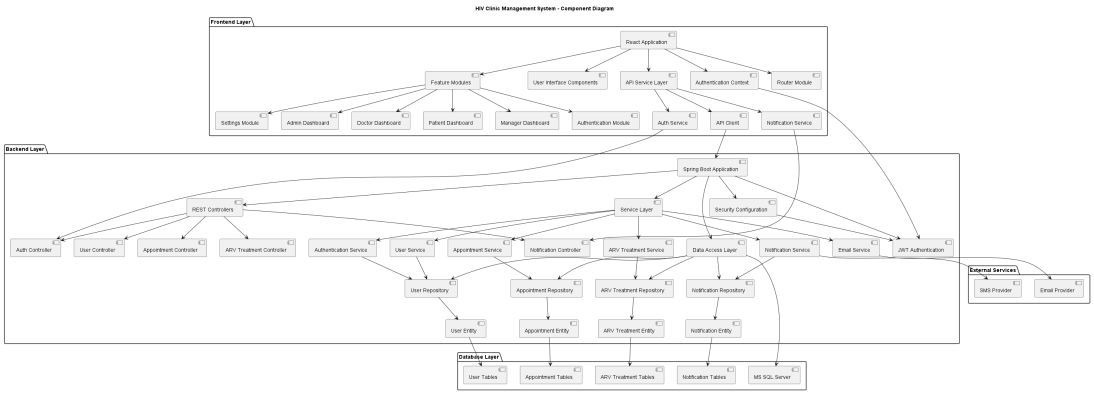


Figure 12: System Component Architecture

2.6.2 Deployment Architecture

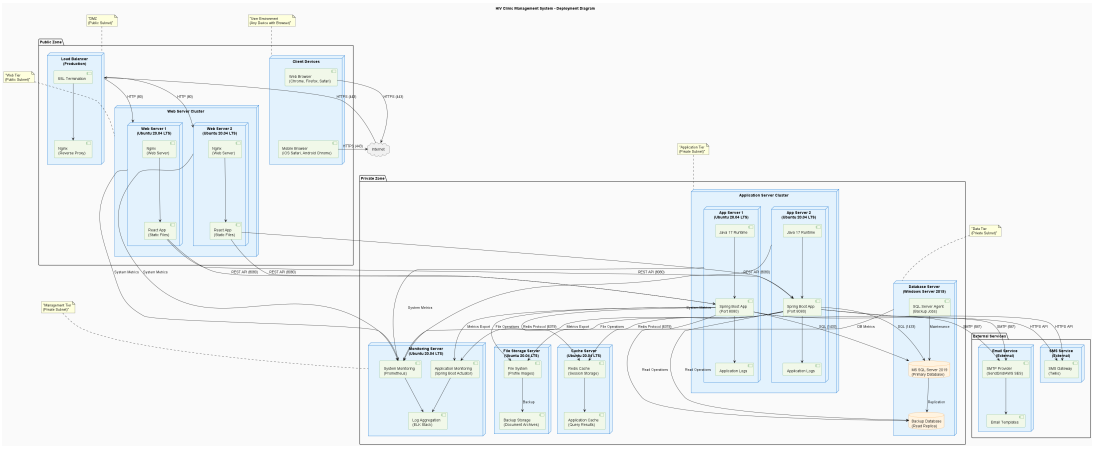


Figure 13: System Deployment Architecture

3 API Documentation

3.1 REST API Endpoints

Method	Endpoint	Role	Description
POST	/api/auth/register	Public	User registration with role assignment
POST	/api/auth/login	Public	User authentication and JWT token generation
GET	/api/auth/me	Authenticated	Get current user profile information
PUT	/api/auth/profile	Authenticated	Update user profile information
POST	/api/appointments/book	Patient	Book new appointment with doctor
GET	/api/appointments/patient/my-appointments	Patient	Get patient's appointments
GET	/api/appointments/doctor/my-appointments	Doctor	Get doctor's appointments
PUT	/api/appointments/{id}/cancel	Patient/Doctor	Cancel appointment
POST	/api/doctors/availability	Doctor	Create availability slots
GET	/api/doctors/{id}/available-slots	Patient	Get doctor's available slots
POST	/api/arv-treatments/add	Doctor	Create ARV treatment record
GET	/api/arv-treatments/my-treatments	Patient	Get patient's ARV treatments
GET	/api/notifications	Authenticated	Get user notifications
POST	/api/notifications/{id}/read	Authenticated	Mark notification as read
GET	/api/admin/users	Admin	Get all users (admin only)
POST	/api/admin/doctors	Admin	Create doctor profile
GET	/api/manager/stats	Manager	Get system statistics

4 System Behavior and Network Architecture

4.1 State Transition Diagrams

Figure 14: System State Transition Diagrams

4.2 Network Architecture

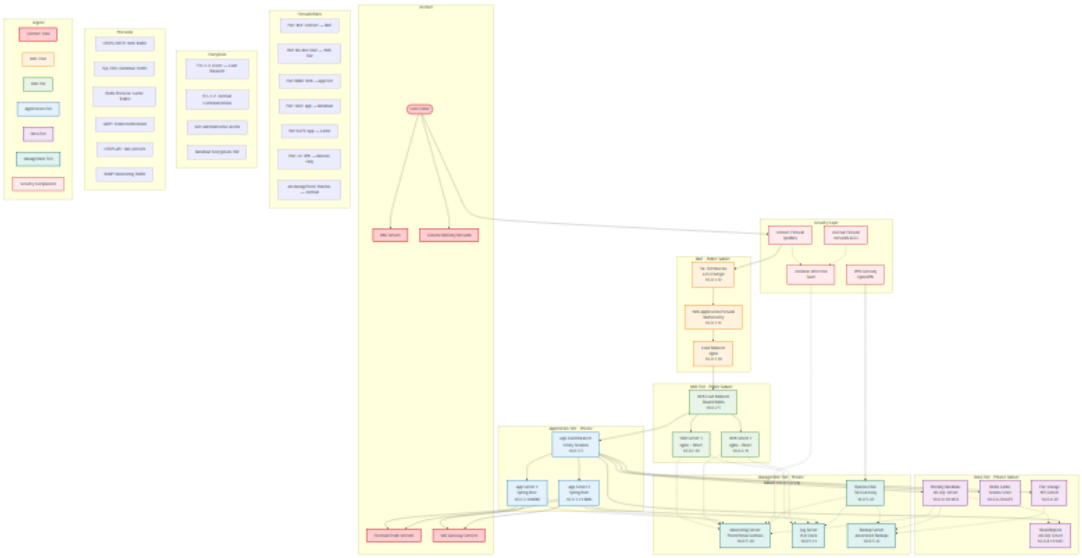


Figure 15: Network Architectureand Security Zones

5 Deployment Architecture

5.1 Development Environment

- **Frontend Development Server**: Vite development server on port 3000
- **Backend Development Server**: Spring Boot embedded Tomcat on port 8080
- **Database**: Microsoft SQL Server LocalDB or SQL Server Express
- **Build Tools**: Maven for backend, npm/yarn for frontend

5.2 Production Deployment

- **Frontend**: Static files served by Nginx or Apache HTTP Server
- **Backend**: Java JAR file deployed on application server (Tomcat, Jetty)
- **Database**: Microsoft SQL Server Standard/Enterprise Edition
- **Security**: HTTPS with SSL/TLS certificates
- **Monitoring**: Application logging with logback and system monitoring

6 Conclusion

The HIV Clinic Management System implements a comprehensive, secure, and scalable solution for managing HIV patient care. The system architecture follows modern software engineering principles with clear separation of concerns, robust security measures, and comprehensive data management capabilities.

Key architectural strengths include:

- **Layered Architecture**: Clean separation between presentation, business logic, and data access layers
- **Security-First Design**: Comprehensive security implementation with JWT authentication and role-based access control
- **Scalable Database Design**: Normalized database schema with proper indexing and referential integrity

- **RESTful API Design**: Well-designed REST API with consistent naming conventions and proper HTTP methods
- **Comprehensive Notification System**: Automated notification system for appointments and medication reminders
- **Audit Trail**: Complete audit trail for all critical operations and data changes

The system is designed to be maintainable, extensible, and capable of handling future enhancements while maintaining data integrity and security standards required for healthcare applications.