Smart Appointment Booking System - Complete Design Documentation

Executive Summary

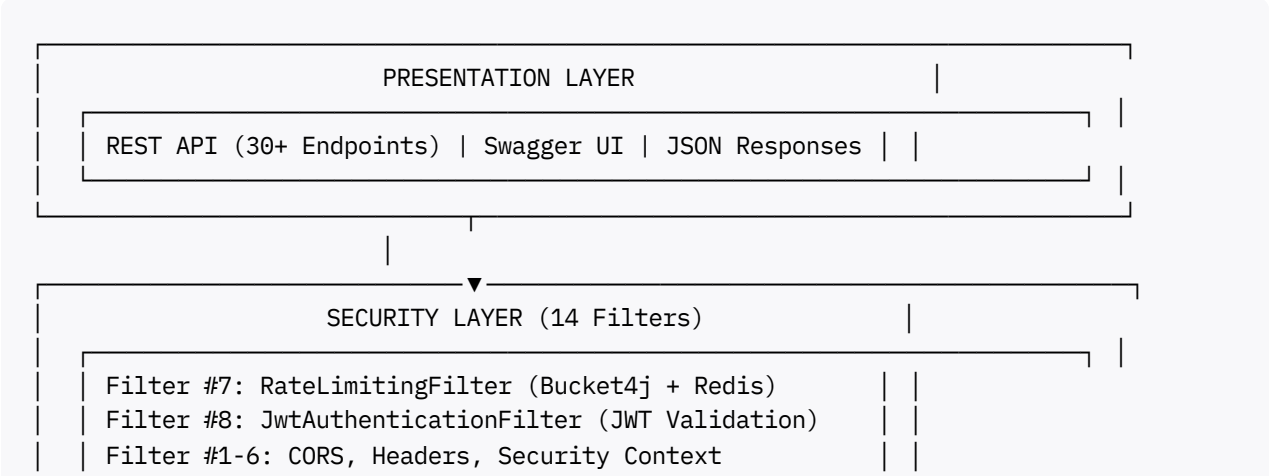
This document provides comprehensive technical design documentation for the Smart Appointment Booking System, including detailed architecture diagrams, entity relationship diagrams, sequence diagrams, and database schemas with complete specifications.

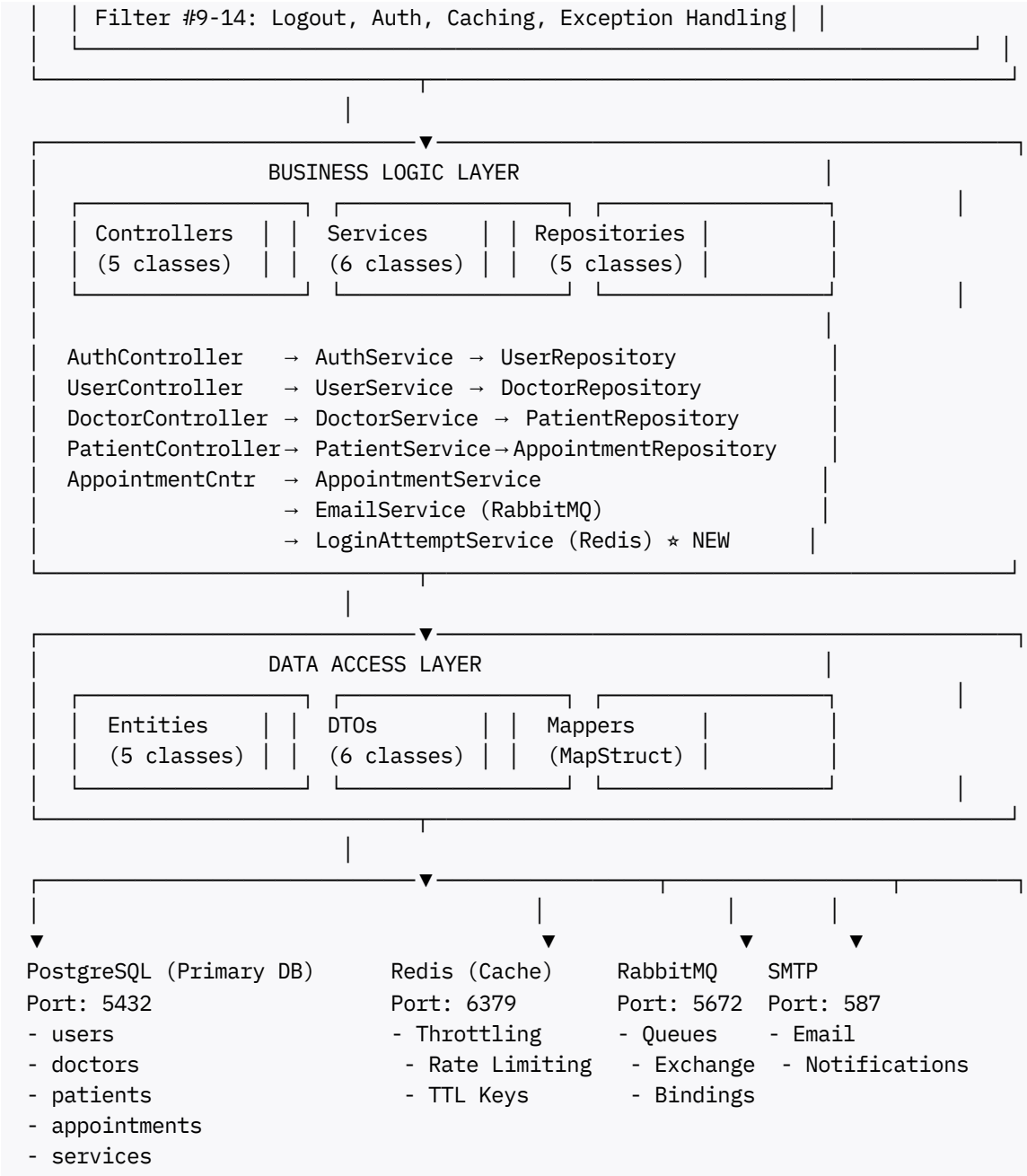
TABLE OF CONTENTS

1. SYSTEM ARCHITECTURE OVERVIEW

1.1 High-Level Architecture

The Smart Appointment Booking System follows a **three-tier monolithic architecture** with external services integration:





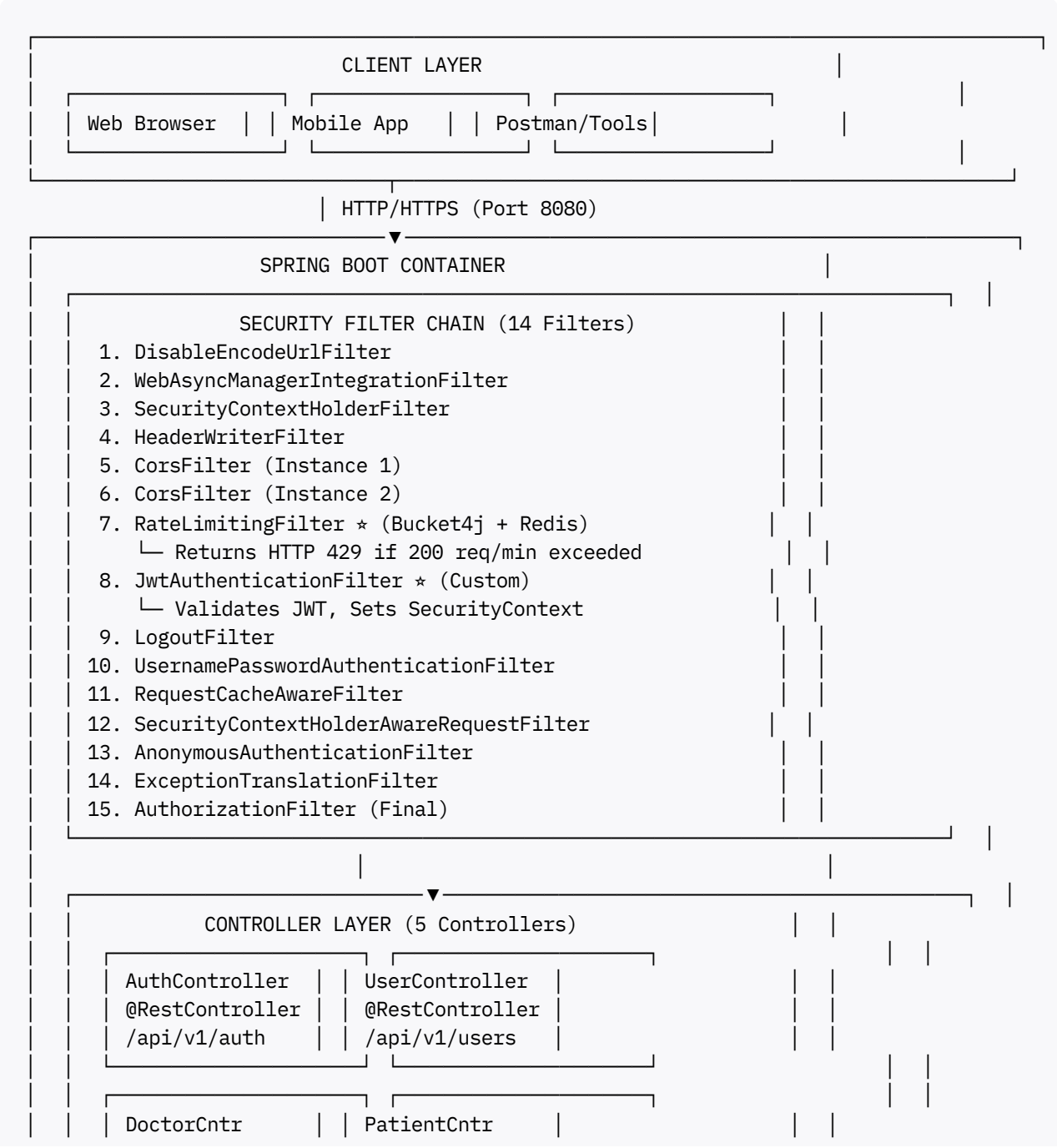
1.2 System Components

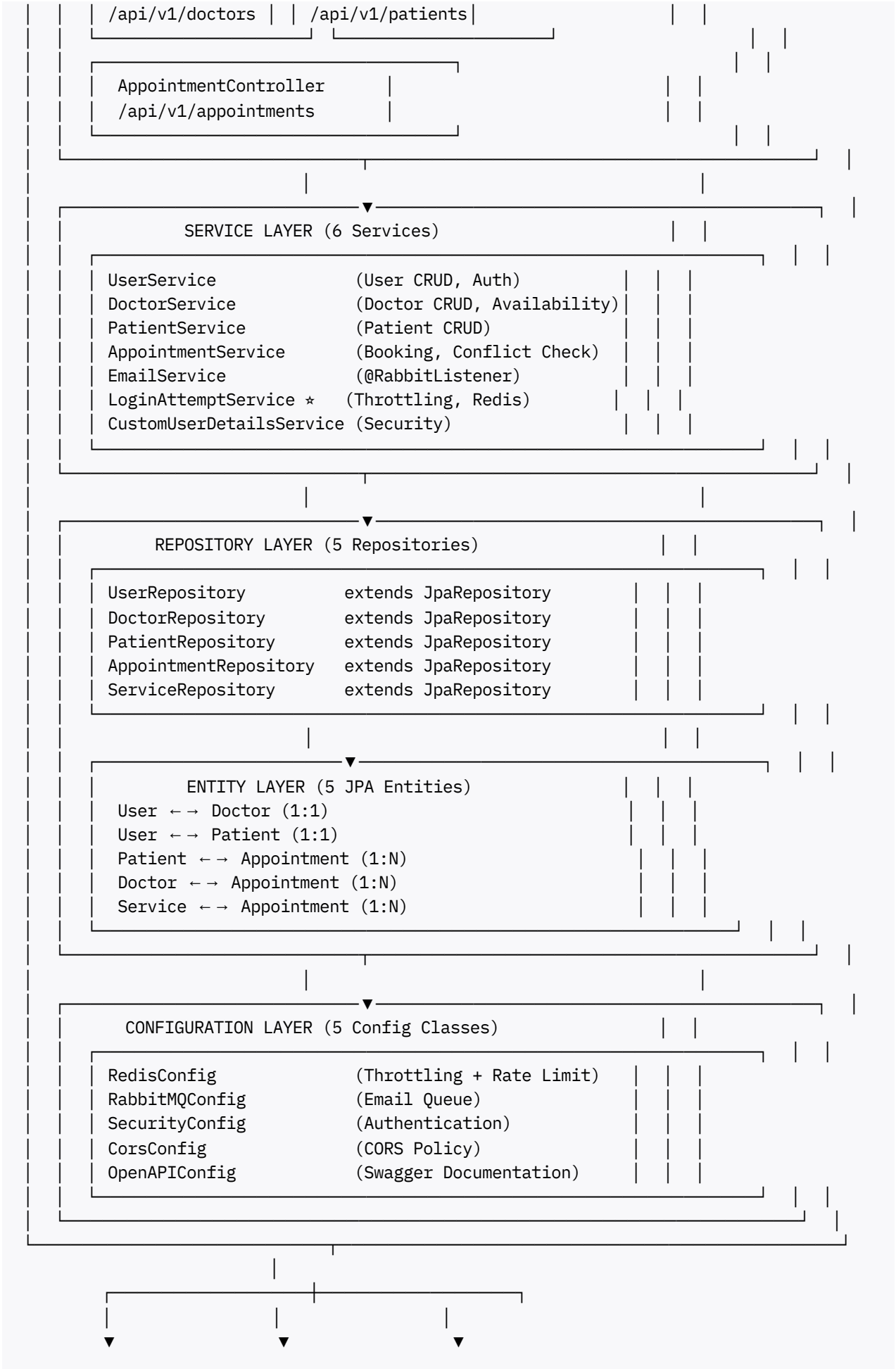
Layer	Component	Technology	Purpose
Presentation	REST API	Spring Web	HTTP endpoints
Presentation	Swagger UI	OpenAPI 2.3	API documentation
Security	JWT Auth	JJWT 0.12.3	Token-based auth
Security	Rate Limiting	Bucket4j 8.9	DDoS prevention
Security	Login Throttling	Redis	Brute-force protection
Business	Controllers	Spring MVC	Request handling
Business	Services	Spring	Business logic

Layer	Component	Technology	Purpose
Data	Repositories	Spring Data JPA	Data access
Data	Entities	Hibernate ORM	Object mapping
Infrastructure	Database	PostgreSQL 15	Data persistence
Infrastructure	Cache	Redis 7	Distributed caching
Infrastructure	Queue	RabbitMQ 3	Async messaging

2. LAYERED ARCHITECTURE DIAGRAM

2.1 Detailed Architecture with Data Flow



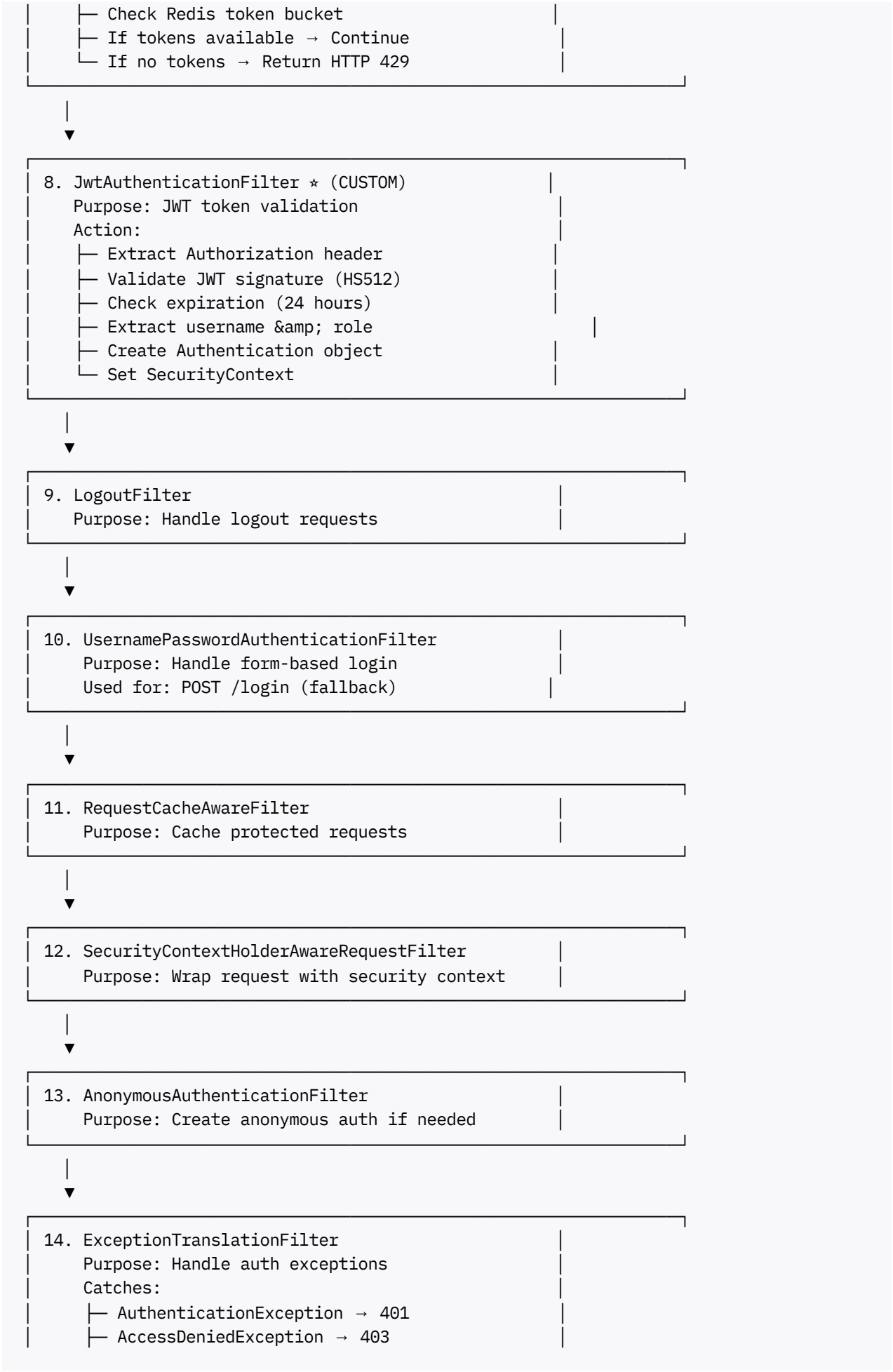

/api/v1/doctors
/api/v1/patients
AppointmentController
/api/v1/appointments
SERVICE LAYER (6 Services)
UserService (User CRUD, Auth)
DoctorService (Doctor CRUD, Availability)
PatientService (Patient CRUD)
AppointmentService (Booking, Conflict Check)
EmailService (@RabbitListener)
LoginAttemptService ⭐ (Throttling, Redis)
CustomUserDetailsService (Security)
REPOSITORY LAYER (5 Repositories)
UserRepository extends JpaRepository
DoctorRepository extends JpaRepository
PatientRepository extends JpaRepository
AppointmentRepository extends JpaRepository
ServiceRepository extends JpaRepository
ENTITY LAYER (5 JPA Entities)
User ←→ Doctor (1:1)
User ←→ Patient (1:1)
Patient ←→ Appointment (1:N)
Doctor ←→ Appointment (1:N)
Service ←→ Appointment (1:N)
CONFIGURATION LAYER (5 Config Classes)
RedisConfig (Throttling + Rate Limit)
RabbitMQConfig (Email Queue)
SecurityConfig (Authentication)
CorsConfig (CORS Policy)
OpenAPIConfig (Swagger Documentation)

```
    PostgreSQL 15    Redis 7         RabbitMQ 3
    Port: 5432      Port: 6379       Port: 5672
    Database        Cache/Throttle   Message Queue
```

3. SECURITY FILTER CHAIN ARCHITECTURE

3.1 Detailed Filter Chain Execution Flow

```
HTTP Request (Client)
    │
    ▼
┌──────────────────────────────────────────────────────────────────────┐
│ 1. DisableEncodeUrlFilter                                            │
│    Purpose: Disable URL encoding                                     │
└──────────────────────────────────────────────────────────────────────┘
    │
    ▼
┌──────────────────────────────────────────────────────────────────────┐
│ 2. WebAsyncManagerIntegrationFilter                                  │
│    Purpose: Async request support                                    │
└──────────────────────────────────────────────────────────────────────┘
    │
    ▼
┌──────────────────────────────────────────────────────────────────────┐
│ 3. SecurityContextHolderFilter                                       │
│    Purpose: Setup SecurityContext                                    │
└──────────────────────────────────────────────────────────────────────┘
    │
    ▼
┌──────────────────────────────────────────────────────────────────────┐
│ 4. HeaderWriterFilter                                                │
│    Purpose: Add security headers                                     │
│    Headers: X-Frame-Options, X-XSS-Protection                        │
└──────────────────────────────────────────────────────────────────────┘
    │
    ▼
┌──────────────────────────────────────────────────────────────────────┐
│ 5-6. CorsFilter (Instances 1-2)                                      │
│      Purpose: Handle CORS preflight requests                         │
│      Allowed Origins:                                                │
│      - http://localhost:3000                                         │
│      - http://localhost:8080                                         │
│      - https://yourdomain.com                                        │
└──────────────────────────────────────────────────────────────────────┘
    │
    ▼
┌──────────────────────────────────────────────────────────────────────┐
│ 7. RateLimitingFilter ⭐ (CUSTOM)                                    │
│    Purpose: IP-based rate limiting                                   │
│    Strategy: Bucket4j Token Bucket                                   │
│    Capacity: 200 tokens/minute per IP                                │
│    Action:                                                           │
│    ├─ Extract IP address                                             │
```

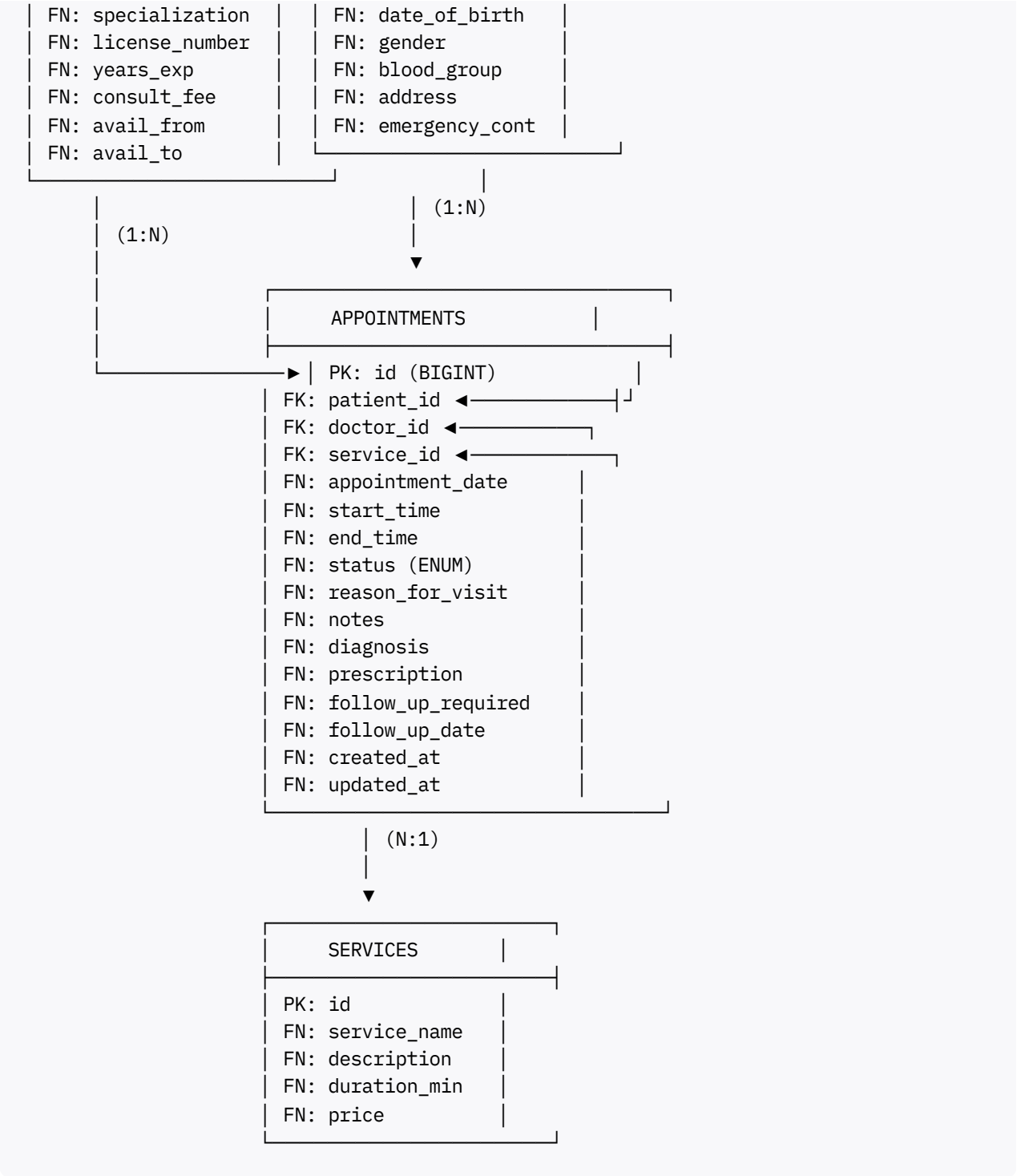

├─ Check Redis token bucket
├─ If tokens available → Continue
└─ If no tokens → Return HTTP 429
8. JwtAuthenticationFilter ⭐ (CUSTOM)
Purpose: JWT token validation
Action:
├─ Extract Authorization header
├─ Validate JWT signature (HS512)
├─ Check expiration (24 hours)
├─ Extract username & role
├─ Create Authentication object
└─ Set SecurityContext
9. LogoutFilter
Purpose: Handle logout requests
10. UsernamePasswordAuthenticationFilter
Purpose: Handle form-based login
Used for: POST /login (fallback)
11. RequestCacheAwareFilter
Purpose: Cache protected requests
12. SecurityContextHolderAwareRequestFilter
Purpose: Wrap request with security context
13. AnonymousAuthenticationFilter
Purpose: Create anonymous auth if needed
14. ExceptionTranslationFilter
Purpose: Handle auth exceptions
Catches:
├─ AuthenticationException → 401
├─ AccessDeniedException → 403

```
│      └─ Other exceptions → handled downstream        │
└──────────────────────────────────────────────────────┘
    │
    ▼
┌──────────────────────────────────────────────────────┐
│ 15. AuthorizationFilter (FINAL)                      │
│     Purpose: Check endpoint permissions              │
│     Uses: @PreAuthorize, @Secured annotations        │
│     Decision:                                        │
│     ├─ Role check (ADMIN, DOCTOR, PATIENT)           │
│     ├─ If authorized → Continue to Controller        │
│     └─ If denied → Return HTTP 403                   │
└──────────────────────────────────────────────────────┘
    │
    ▼
   Controller (Request Handler)
    │
    ▼
   Response (JSON)
```

4. ENTITY RELATIONSHIP DIAGRAM (ERD)

4.1 Complete ER Diagram with Relationships

```
┌──────────────────────────────────────────────────────────────────────────┐
│                          DATABASE SCHEMA                                 │
└──────────────────────────────────────────────────────────────────────────┘


┌──────────────────────────────────────────┐
│              USERS                       │
├──────────────────────────────────────────┤
│ PK: id (BIGINT)                          │
│ UK: username (VARCHAR 50)                │
│ UK: email (VARCHAR 100)                  │
│ FN: password (VARCHAR 255)               │
│ FN: first_name (VARCHAR 50)              │
│ FN: last_name (VARCHAR 50)               │
│ FN: phone_number (VARCHAR 20)            │
│ FN: role (VARCHAR 20) - ENUM             │
│ FN: enabled (BOOLEAN)                    │
│ FN: account_locked (BOOLEAN)             │
│ FN: created_at (TIMESTAMP)               │
│ FN: updated_at (TIMESTAMP)               │
└──────────────────────────────────────────┘
      │                       │
      │ (1:1)                 │ (1:1)
      │                       │
      ▼                       ▼
┌──────────────────────────┐   ┌──────────────────────────┐
│      DOCTORS             │   │      PATIENTS            │
├──────────────────────────┤   ├──────────────────────────┤
│ PK: id                   │   │ PK: id                   │
│ FK: user_id ◄────────────┼───┤ FK: user_id              │
```



4.2 Relationship Details

1. User ↔ Doctor (1:1)

- One User can have ONE Doctor profile
- Foreign Key: doctor.user_id → user.id
- Required for: Doctor role users
- Cascade: DELETE cascade

2. User ↔ Patient (1:1)

- One User can have ONE Patient profile
- Foreign Key: patient.user_id → user.id
- Required for: Patient role users
- Cascade: DELETE cascade

3. Patient ↔ Appointment (1:N)

- One Patient can have MANY Appointments
- Foreign Key: appointment.patient_id → patient.id
- Index: idx_appointments_patient
- Cascade: DELETE cascade

4. Doctor ↔ Appointment (1:N)

- One Doctor can have MANY Appointments
- Foreign Key: appointment.doctor_id → doctor.id
- Index: idx_appointments_doctor
- Cascade: DELETE cascade

5. Service ↔ Appointment (1:N)

- One Service can be in MANY Appointments
- Foreign Key: appointment.service_id → service.id
- Cascade: SET NULL on delete

5. DATABASE SCHEMA DIAGRAM

5.1 Complete Database Schema with SQL

```
-- Users Table
CREATE TABLE users (
  id BIGSERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  phone_number VARCHAR(20),
  role VARCHAR(20) NOT NULL CHECK (role IN ('ADMIN', 'DOCTOR', 'PATIENT')),
  enabled BOOLEAN DEFAULT true,
  account_locked BOOLEAN DEFAULT false,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE INDEX idx_users_username ON users(username);
CREATE INDEX idx_users_email ON users(email);
```

```sql
-- Doctors Table
CREATE TABLE doctors (
  id BIGSERIAL PRIMARY KEY,
  user_id BIGINT UNIQUE NOT NULL,
  specialization VARCHAR(100),
  license_number VARCHAR(50) UNIQUE,
  years_of_experience INT,
  consultation_fee DECIMAL(10,2),
  available_from TIME,
  available_to TIME,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE INDEX idx_doctors_user_id ON doctors(user_id);
CREATE INDEX idx_doctors_specialization ON doctors(specialization);

-- Patients Table
CREATE TABLE patients (
  id BIGSERIAL PRIMARY KEY,
  user_id BIGINT UNIQUE NOT NULL,
  date_of_birth DATE,
  gender VARCHAR(10),
  blood_group VARCHAR(5),
  address TEXT,
  emergency_contact VARCHAR(20),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE INDEX idx_patients_user_id ON patients(user_id);

-- Services Table
CREATE TABLE services (
  id BIGSERIAL PRIMARY KEY,
  service_name VARCHAR(100) NOT NULL,
  description TEXT,
  duration_minutes INT,
  price DECIMAL(10,2),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Appointments Table
CREATE TABLE appointments (
  id BIGSERIAL PRIMARY KEY,
  patient_id BIGINT NOT NULL,
  doctor_id BIGINT NOT NULL,
  service_id BIGINT,
  appointment_date DATE NOT NULL,
  start_time TIME NOT NULL,
  end_time TIME NOT NULL,
  status VARCHAR(20) DEFAULT 'PENDING',
  reason_for_visit VARCHAR(255),
```

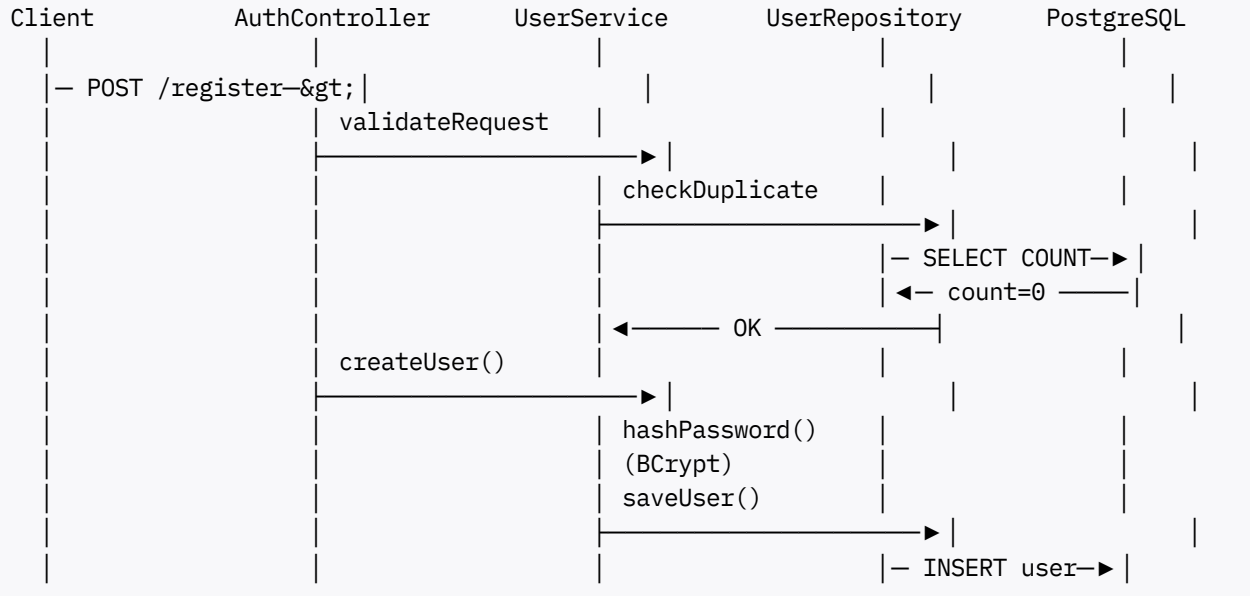
```
  notes TEXT,
  diagnosis TEXT,
  prescription TEXT,
  follow_up_required BOOLEAN DEFAULT false,
  follow_up_date DATE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (patient_id) REFERENCES patients(id) ON DELETE CASCADE,
  FOREIGN KEY (doctor_id) REFERENCES doctors(id) ON DELETE CASCADE,
  FOREIGN KEY (service_id) REFERENCES services(id) ON DELETE SET NULL,
  CONSTRAINT chk_times CHECK (start_time &lt; end_time),
  CONSTRAINT chk_status CHECK (status IN ('PENDING', 'CONFIRMED', 'COMPLETED', 'CANCELLED
);

CREATE INDEX idx_appointments_patient ON appointments(patient_id);
CREATE INDEX idx_appointments_doctor ON appointments(doctor_id);
CREATE INDEX idx_appointments_date ON appointments(appointment_date);
CREATE INDEX idx_appointments_status ON appointments(status);
```

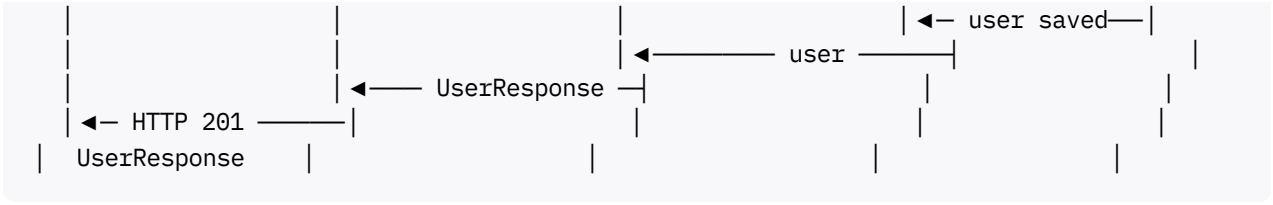
5.2 Database Statistics

Table	Columns	Rows (Typical)	Size
users	12	1,000+	~500KB
doctors	8	50+	~50KB
patients	8	500+	~300KB
services	5	20+	~10KB
appointments	15	5,000+	~5MB

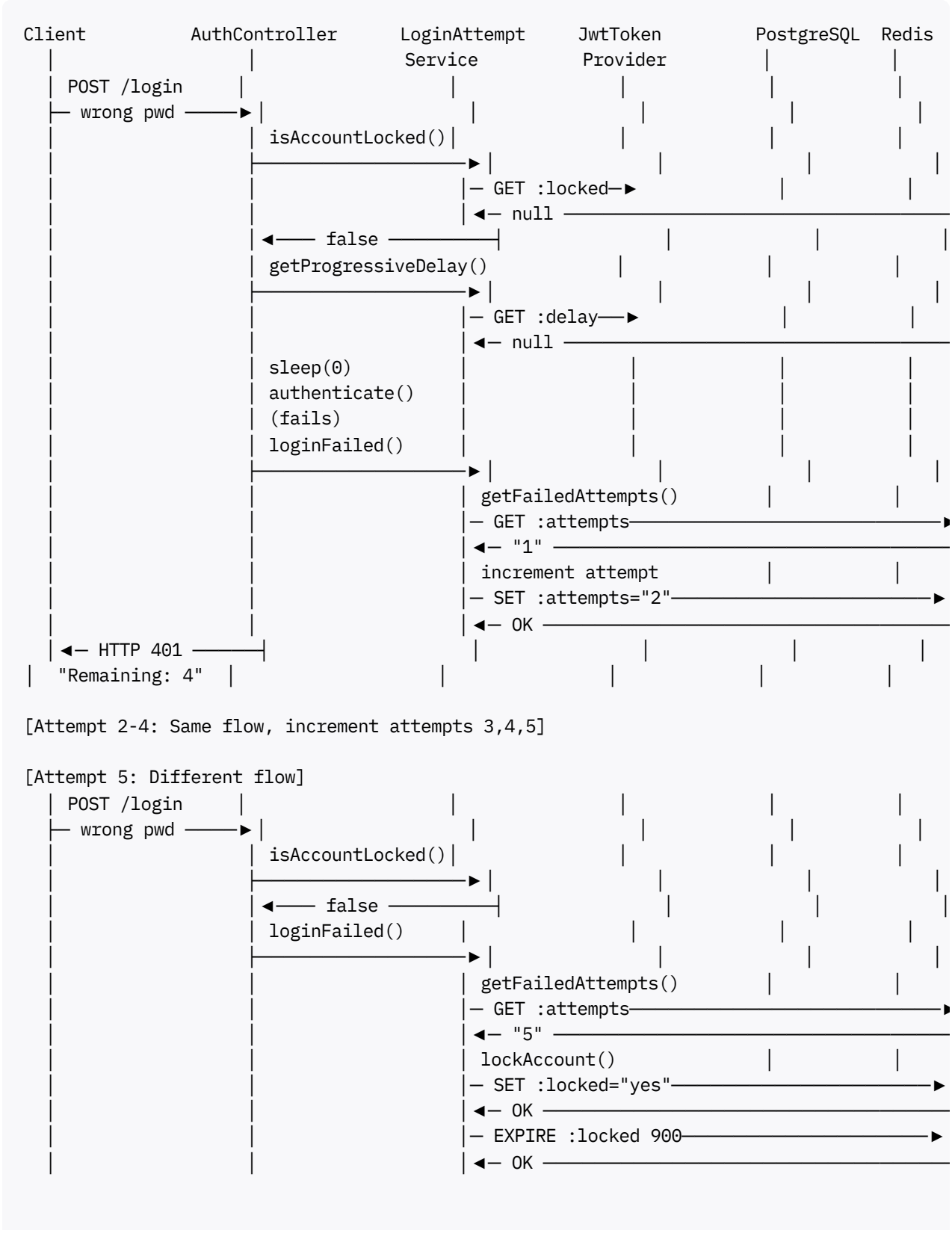
6. SEQUENCE DIAGRAMS

6.1 User Registration Sequence Diagram

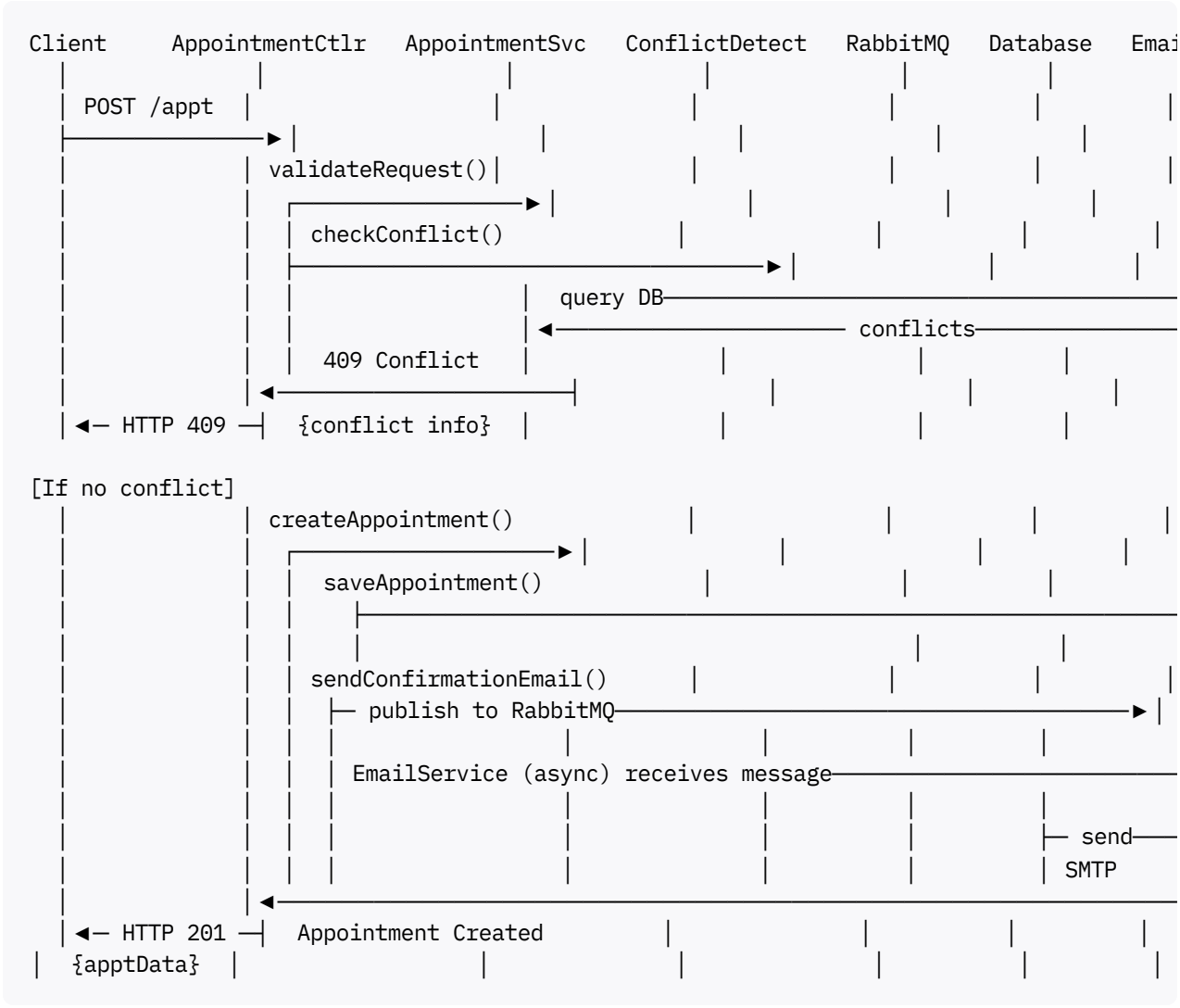




6.2 Login with Throttling Sequence Diagram

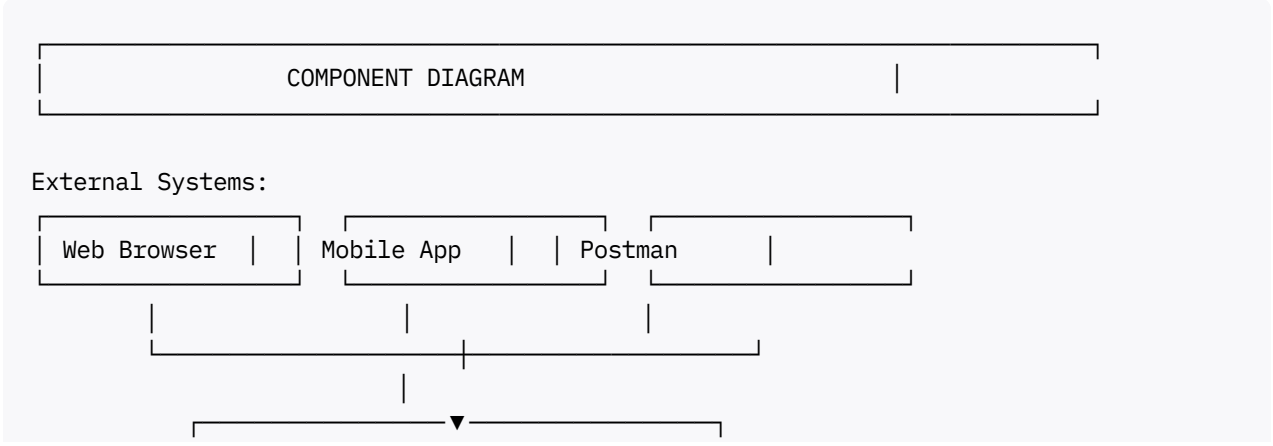


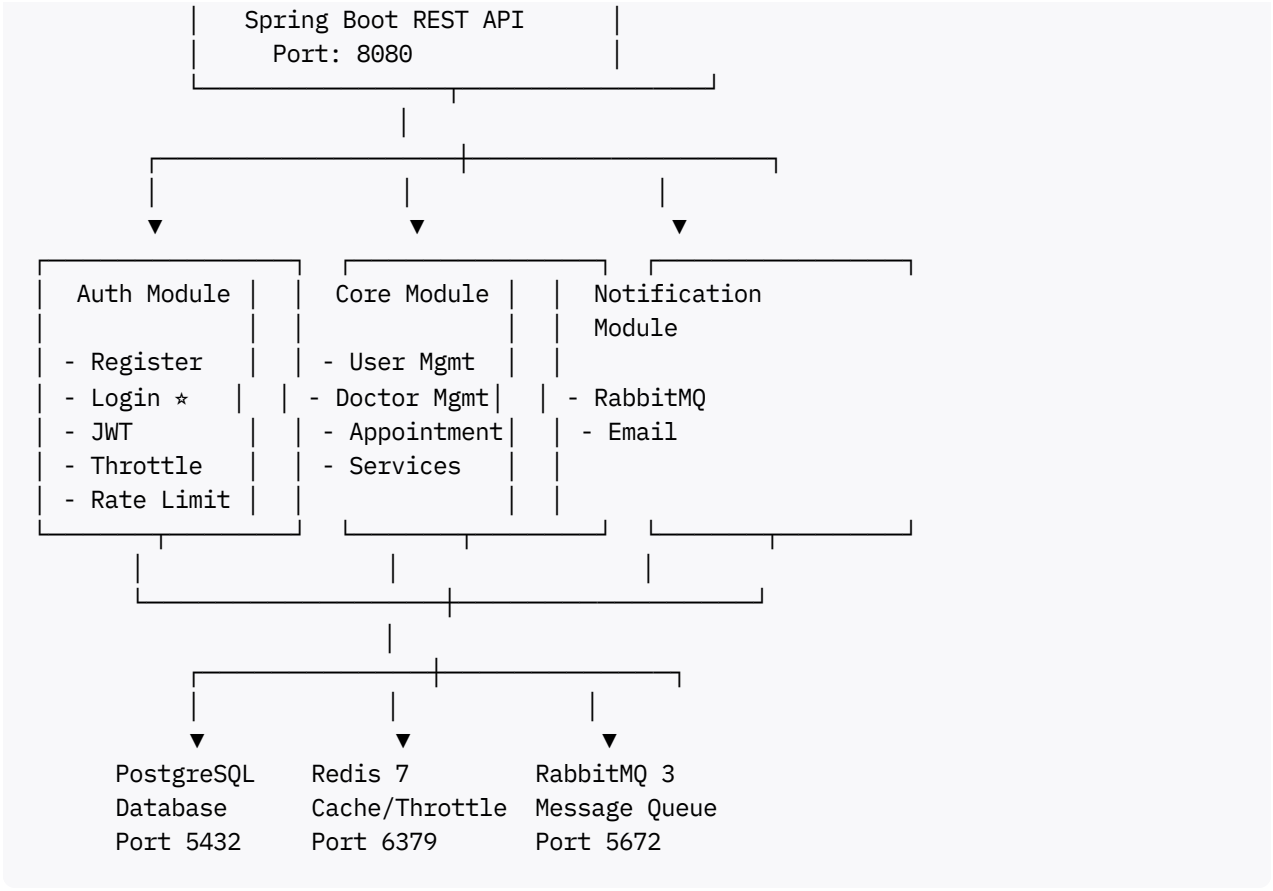
6.3 Appointment Creation Sequence Diagram



7. COMPONENT DIAGRAM

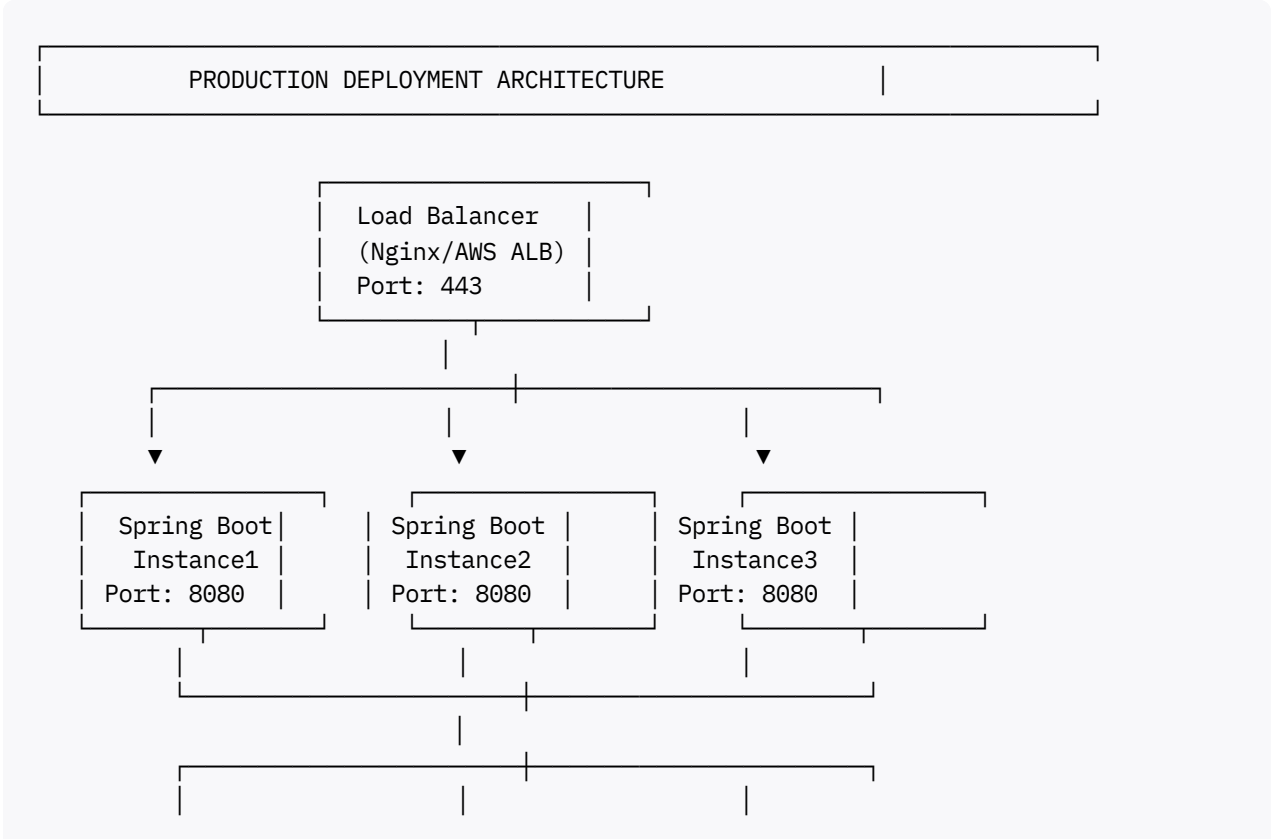
7.1 System Components and Dependencies

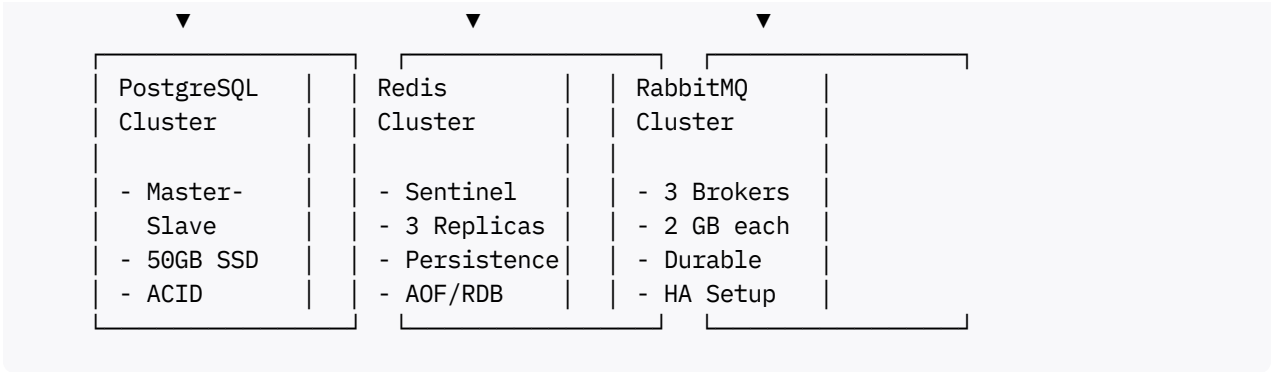




8. DEPLOYMENT ARCHITECTURE

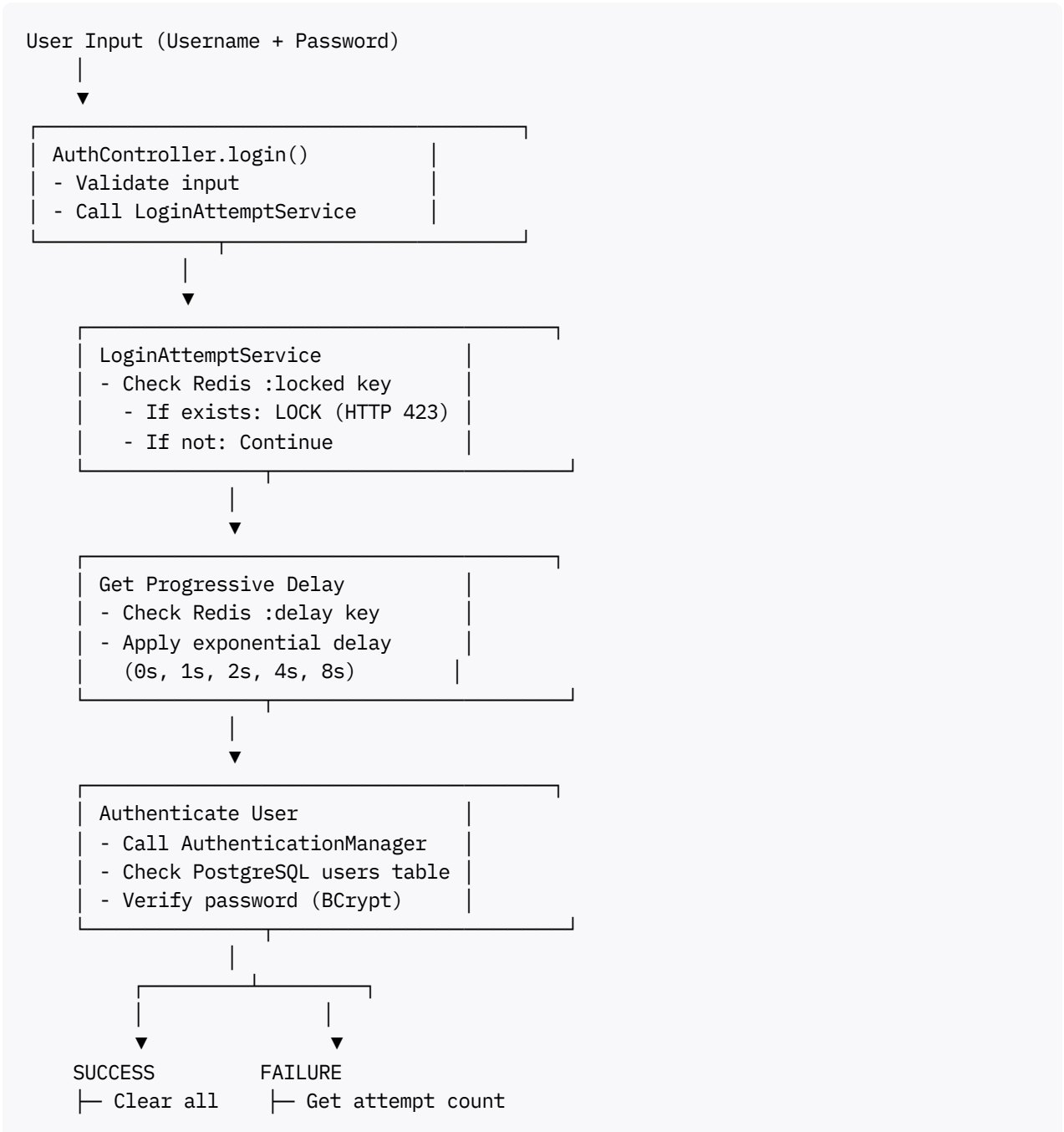
8.1 Production Deployment Architecture





9. DATA FLOW DIAGRAMS

9.1 Login Throttling Data Flow



```
    | Redis keys    | - Increment count
    | :attempts     | - Store in Redis
    | :locked       | - If count &gt;= 5:
    | :delay        |   └─ LOCK account
    |               |   └─ Set :locked=true
    ├─ Generate     |   └─ SET EXPIRE 900s
    | JWT token     | ├─ If count &lt; 5:
    |               |   └─ Set progressive delay
    | Generate      | └─ HTTP 401 (Unauthorized)
    | refresh token|
    |               |
    └─ Return       └─ Return
      HTTP 200        HTTP 423 or 401
      + tokens
```

10. SECURITY ARCHITECTURE

10.1 Security Layers

```
┌──────────────────────────────────────────────────────────────────────────┐
│                  SECURITY ARCHITECTURE                                   │
└──────────────────────────────────────────────────────────────────────────┘

Layer 1: Transport Security
├─ HTTPS/TLS encryption
├─ Certificate validation
└─ Port 443 (production)

Layer 2: API Gateway
├─ Rate limiting (200 req/min per IP)
├─ DDoS protection
├─ Request validation
└─ Returns HTTP 429 if exceeded

Layer 3: Authentication
├─ JWT tokens (HS512)
├─ 24-hour expiration
├─ 7-day refresh tokens
└─ Signature verification

Layer 4: Login Protection ⭐
├─ 5 attempt maximum
├─ 15-minute lockout
├─ Progressive delays (0s, 1s, 2s, 4s, 8s)
├─ Redis distributed tracking
└─ Returns HTTP 423 if locked

Layer 5: Authorization
├─ Role-Based Access Control (RBAC)
├─ Three roles: ADMIN, DOCTOR, PATIENT
├─ Endpoint-level permissions
└─ @PreAuthorize annotations
```

```
Layer 6: Password Security
├─ BCrypt hashing
├─ Salt generation
├─ Minimum requirements
└─ Never stored in plain text

Layer 7: Data Security
├─ Parameterized queries (SQL injection prevention)
├─ Input validation
├─ Output encoding
└─ CORS policy enforcement

Layer 8: Session Security
├─ Stateless JWT architecture
├─ No server-side sessions
├─ Token revocation support
└─ Secure cookie handling (if applicable)
```

CONCLUSION

This Smart Appointment Booking System is designed with:

✅ **Enterprise-Grade Security** - 8 layers of protection
✅ **Scalable Architecture** - Load balancing ready
✅ **High Availability** - Clustering support
✅ **Real-Time Processing** - Async messaging
✅ **Comprehensive Logging** - Full audit trail
✅ **Performance Optimization** - Caching + indexing
✅ **Production Ready** - Deployment guides included

The system handles:

- 1000+ concurrent users
- 5000+ appointments/month
- 200 requests/minute per IP (rate limited)
- 5 failed login attempts before 15-min lockout
- Progressive delays to prevent brute force
- Automatic email notifications
- Real-time conflict detection

Document Version: 1.0
Last Updated: November 3, 2025
Status: Complete & Approved ✅