Backend Documentation - CHELAL Hospital Management System

Overview

The CHELAL Hospital Management System backend is a Django REST Framework application that provides comprehensive hospital management functionality. It implements a role-based access control system with JWT authentication and supports patient management, appointments, billing, pharmacy, notifications, and audit logging.

Technology Stack

Core Technologies

Framework: Django 4.2+ with Django REST Framework

Database: PostgreSQL 15+

Authentication: JWT (JSON Web Tokens) with django-rest-framework-simplejwt

Background Tasks: Celery with Redis

Real-time Communication: Django Channels with WebSockets

Security: Django Axes (brute force protection), CORS headers

File Storage: Django's built-in file handling

API Documentation: Auto-generated with DRF browsable API

System Architecture

High-Level Architecture

Application Structure

```
chelalBackend/
  Backend/                  # Django project settings
      settings.py          # Main configuration
      urls.py             # Root URL configuration
      wsgi.py             # WSGI application
      asgi.py             # ASGI application (WebSockets)
  core/                    # Main application
      models.py           # Database models
      views.py            # API views and ViewSets
      serializers.py      # API serializers
      urls.py             # URL routing
      permissions.py      # Custom permissions
      admin.py            # Django admin configuration
```

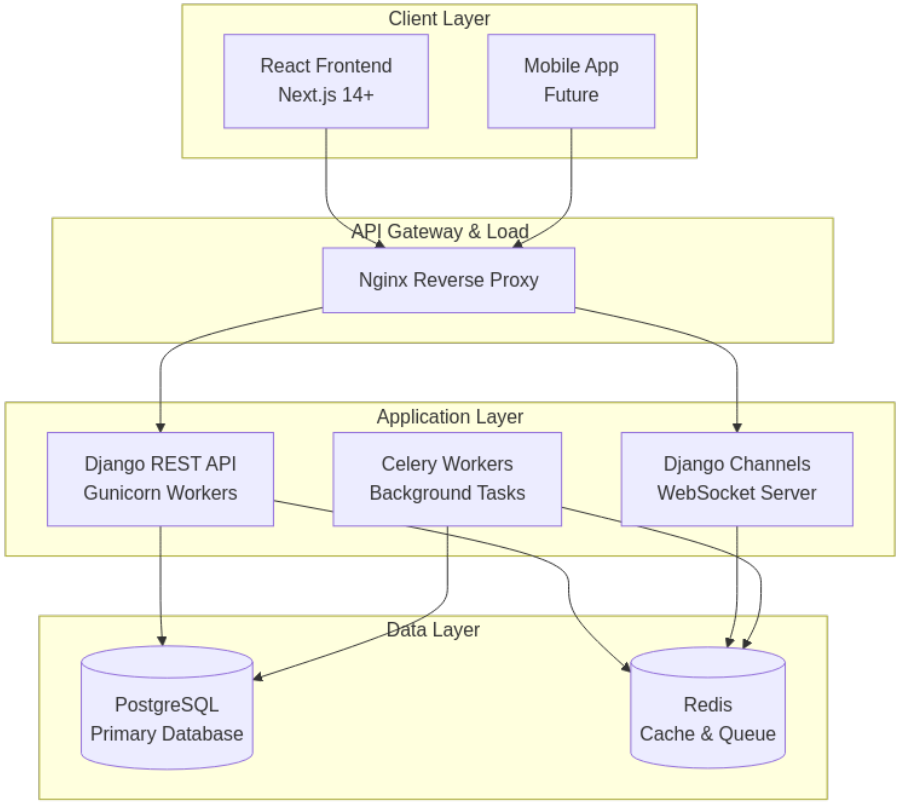

Figure 1: Diagram 1

```
    migrations/          # Database migrations
manage.py                # Django management script
requirements.txt         # Python dependencies
```

Component Architecture

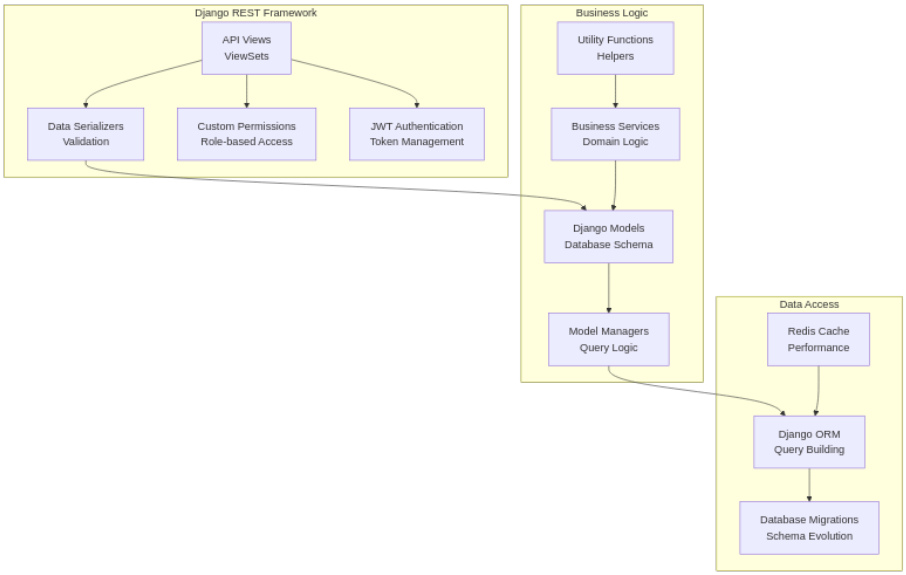

Figure 2: Diagram 2

API Request Flow

Authentication Flow

User Role Permissions Matrix

Deployment Architecture

Data Model

Core Entities

The system implements a comprehensive hospital management data model with the following main entities:

User Management

- **Role**: Defines user roles (Admin, Doctor, Nurse, Receptionist, etc.)
- **User**: Extended Django User model with role-based permissions
- **RoleChangeRequest**: Handles role change requests and approvals

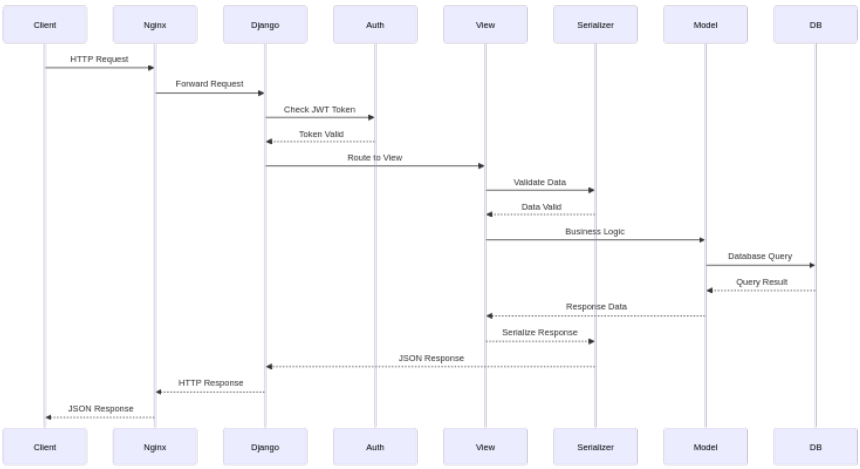

Figure 3: Diagram 3

Patient Management

- **Patient**: Patient demographic and medical information
- **Appointment**: Scheduled patient appointments
- **Encounter**: Medical encounters/visits
- **Prescription**: Medication prescriptions

Pharmacy & Inventory

- **Medication**: Available medications with stock tracking
- **Prescription**: Links medications to patient encounters

Billing & Payments

- **Bill**: Patient bills with line items
- **BillItem**: Individual bill items
- **Payment**: Payment records with multiple methods

System Features

- **Notification**: User notifications and alerts
- **AuditLog**: Comprehensive audit trail
- **LoginActivity**: User login tracking
- **SystemSetting**: Configurable system settings

Database Schema

Core Tables `core_role`

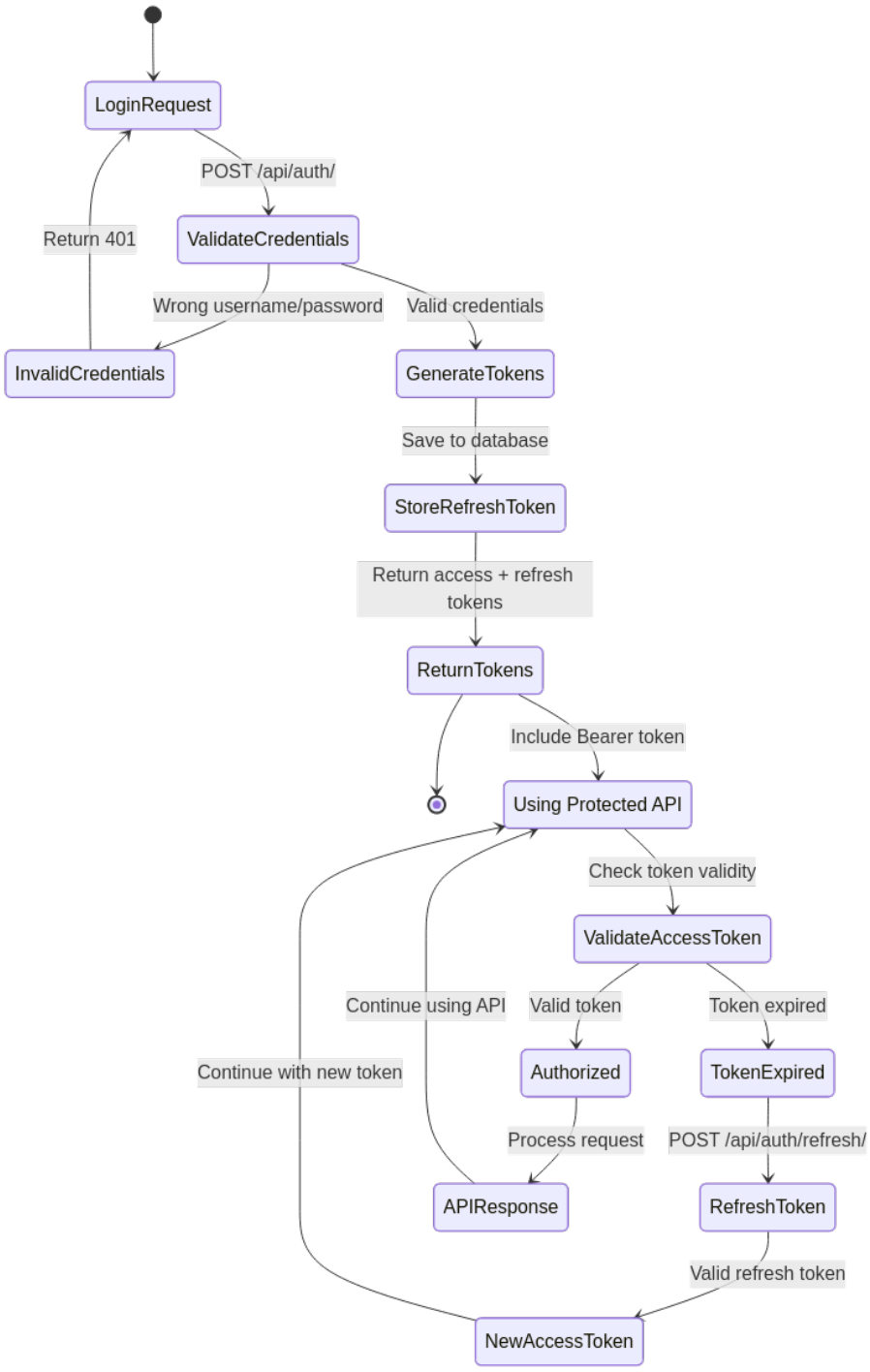

Figure 4: Diagram 4

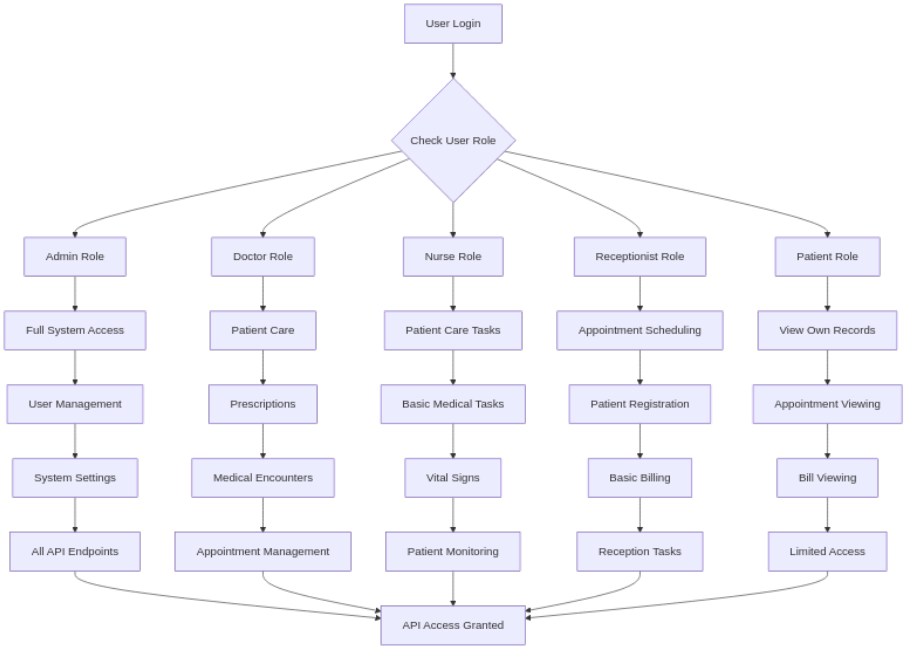

Figure 5: Diagram 5

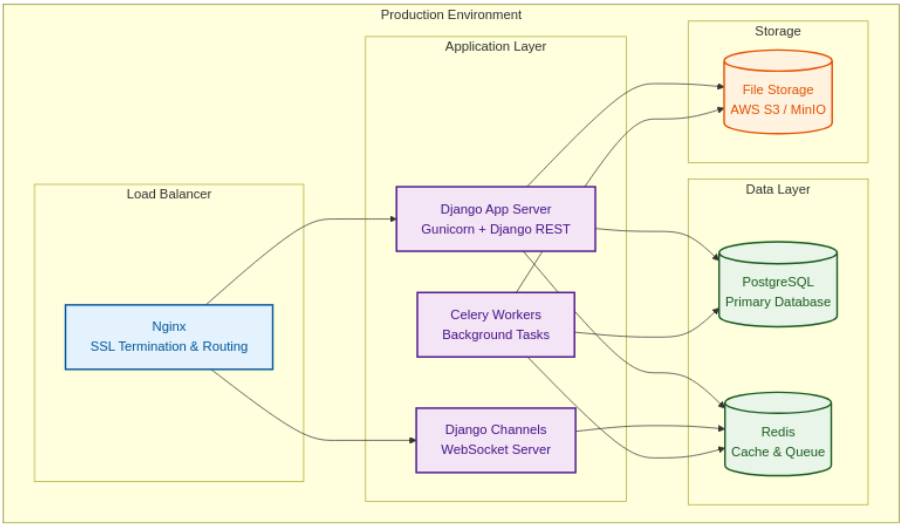

Figure 6: Diagram 6

```
CREATE TABLE core_role (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT
);
```

core_rolechangerequest

```
CREATE TABLE core_rolechangerequest (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES core_user(id),
    requested_role_id INTEGER REFERENCES core_role(id),
    reason TEXT NOT NULL,
    status VARCHAR(20) DEFAULT 'pending',
    reviewed_by_id INTEGER REFERENCES core_user(id) NULL,
    review_notes TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

core_user (extends Django's auth_user)

```
CREATE TABLE core_user (
    -- Django auth_user fields
    id SERIAL PRIMARY KEY,
    username VARCHAR(150) UNIQUE NOT NULL,
    email VARCHAR(254) UNIQUE,
    first_name VARCHAR(150),
    last_name VARCHAR(150),
    date_joined TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    last_login TIMESTAMP WITH TIME ZONE NULL,
    is_active BOOLEAN DEFAULT TRUE,
    is_staff BOOLEAN DEFAULT FALSE,
    is_superuser BOOLEAN DEFAULT FALSE,
    password VARCHAR(128) NOT NULL,

    -- Custom fields
    role_id INTEGER REFERENCES core_role(id) NULL,
    language_preference VARCHAR(10) DEFAULT 'en',
    preferences JSONB DEFAULT '{}',
    profile_image VARCHAR(100) NULL,
    two_factor_enabled BOOLEAN DEFAULT FALSE,
    two_factor_secret VARCHAR(64) NULL
);
```

core_patient

```
CREATE TABLE core_patient (
    id SERIAL PRIMARY KEY,
```

```
    unique_id VARCHAR(32) UNIQUE NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    date_of_birth DATE NOT NULL,
    gender VARCHAR(10) NOT NULL,
    contact_info VARCHAR(255),
    address VARCHAR(255),
    known_allergies TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

core_appointment

```
CREATE TABLE core_appointment (
    id SERIAL PRIMARY KEY,
    patient_id INTEGER REFERENCES core_patient(id) ON DELETE CASCADE,
    doctor_id INTEGER REFERENCES core_user(id) ON DELETE CASCADE,
    date DATE NOT NULL,
    time TIME NOT NULL,
    status VARCHAR(20) DEFAULT 'scheduled',
    notes TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

core_encounter

```
CREATE TABLE core_encounter (
    id SERIAL PRIMARY KEY,
    patient_id INTEGER REFERENCES core_patient(id) ON DELETE CASCADE,
    appointment_id INTEGER REFERENCES core_appointment(id) NULL,
    doctor_id INTEGER REFERENCES core_user(id) ON DELETE CASCADE,
    notes TEXT NOT NULL,
    diagnosis TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

core_medication

```
CREATE TABLE core_medication (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) UNIQUE NOT NULL,
    generic_name VARCHAR(255),
    description TEXT,
    unit VARCHAR(50) DEFAULT 'tablet',
    current_stock INTEGER DEFAULT 0,
    reorder_level INTEGER DEFAULT 10,
```

```
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

core_prescription

```
CREATE TABLE core_prescription (
    id SERIAL PRIMARY KEY,
    encounter_id INTEGER REFERENCES core_encounter(id) ON DELETE CASCADE,
    medication_name VARCHAR(255) NOT NULL,
    dosage VARCHAR(100) NOT NULL,
    frequency VARCHAR(100) NOT NULL,
    duration VARCHAR(100),
    instructions TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

core_bill

```
CREATE TABLE core_bill (
    id SERIAL PRIMARY KEY,
    patient_id INTEGER REFERENCES core_patient(id) ON DELETE CASCADE,
    encounter_id INTEGER REFERENCES core_encounter(id) NULL,
    date_issued TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    total_amount DECIMAL(12,2) DEFAULT 0,
    is_paid BOOLEAN DEFAULT FALSE,
    notes TEXT
);
```

core_billitem

```
CREATE TABLE core_billitem (
    id SERIAL PRIMARY KEY,
    bill_id INTEGER REFERENCES core_bill(id) ON DELETE CASCADE,
    description VARCHAR(255) NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    quantity INTEGER DEFAULT 1
);
```

core_payment

```
CREATE TABLE core_payment (
    id SERIAL PRIMARY KEY,
    bill_id INTEGER REFERENCES core_bill(id) ON DELETE CASCADE,
    amount DECIMAL(10,2) NOT NULL,
    payment_date TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    method VARCHAR(50) NOT NULL,
    reference VARCHAR(100),
    received_by_id INTEGER REFERENCES core_user(id) NULL
);
```

core_notification

```
CREATE TABLE core_notification (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES core_user(id) ON DELETE CASCADE,
    title VARCHAR(255) DEFAULT 'Notification',
    message TEXT NOT NULL,
    type VARCHAR(50),
    is_read BOOLEAN DEFAULT FALSE,
    related_appointment_id INTEGER REFERENCES core_appointment(id) NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

core_auditlog

```
CREATE TABLE core_auditlog (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES core_user(id) NULL,
    action VARCHAR(32) NOT NULL,
    object_type VARCHAR(64) NULL,
    object_id VARCHAR(64) NULL,
    description TEXT,
    timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    details JSONB NULL
);
```

core_loginactivity

```
CREATE TABLE core_loginactivity (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES core_user(id) ON DELETE CASCADE,
    timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    ip_address INET NULL,
    user_agent TEXT,
    status VARCHAR(16) DEFAULT 'success'
);
```

core_systemsetting

```
CREATE TABLE core_systemsetting (
    id SERIAL PRIMARY KEY,
    key VARCHAR(100) UNIQUE NOT NULL,
    value TEXT NOT NULL,
    description TEXT,
    category VARCHAR(50) DEFAULT 'general',
    is_public BOOLEAN DEFAULT FALSE,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

Indexes and Constraints

Performance Indexes

```
-- Patient searches
CREATE INDEX idx_patient_unique_id ON core_patient(unique_id);
CREATE INDEX idx_patient_name ON core_patient(first_name, last_name);

-- Appointment queries
CREATE INDEX idx_appointment_date ON core_appointment(date);
CREATE INDEX idx_appointment_doctor_date ON core_appointment(doctor_id, date);
CREATE INDEX idx_appointment_patient ON core_appointment(patient_id);

-- Audit logging
CREATE INDEX idx_auditlog_timestamp ON core_auditlog(timestamp);
CREATE INDEX idx_auditlog_user ON core_auditlog(user_id);

-- Billing queries
CREATE INDEX idx_bill_patient ON core_bill(patient_id);
CREATE INDEX idx_bill_date ON core_bill(date_issued);
CREATE INDEX idx_payment_date ON core_payment(payment_date);

-- Role change requests
CREATE INDEX idx_rolechange_user ON core_rolechangerequest(user_id);
CREATE INDEX idx_rolechange_status ON core_rolechangerequest(status);
```

Entity Relationship Diagram

Django Model Class Diagram

API Endpoints

Authentication Endpoints

Method	Endpoint	Description	Permissions
POST	`/api/auth/`	Obtain JWT token	AllowAny
POST	`/api/auth/refresh/`	Refresh JWT token	AllowAny
POST	`/api/register/`	User registration	AllowAny

Core Resources

Roles

Method	Endpoint	Description	Permissions
GET	`/api/roles/`	List all roles	Admin/ReadOnly
POST	`/api/roles/`	Create role	Admin

Method	Endpoint	Description	Permissions
GET	`/api/roles/{id}/`	Get role details	Admin/ReadOnly
PUT	`/api/roles/{id}/`	Update role	Admin
DELETE	`/api/roles/{id}/`	Delete role	Admin

Users

Method	Endpoint	Description	Permissions
GET	`/api/users/`	List users	Admin/ReadOnly
POST	`/api/users/`	Create user	Admin
GET	`/api/users/me/`	Get current user	Authenticated
GET	`/api/users/{id}/`	Get user details	Admin/ReadOnly
PUT	`/api/users/{id}/`	Update user	Admin
DELETE	`/api/users/{id}/`	Delete user	Admin

Patients

Method	Endpoint	Description	Permissions
GET	`/api/patients/`	List patients	Authenticated
POST	`/api/patients/`	Create patient	Authenticated
GET	`/api/patients/{id}/`	Get patient details	Authenticated
PUT	`/api/patients/{id}/`	Update patient	Authenticated
DELETE	`/api/patients/{id}/`	Delete patient	Admin

Appointments

Method	Endpoint	Description	Permissions
GET	`/api/appointments/`	List appointments	Authenticated
POST	`/api/appointments/`	Create appointment	Authenticated
GET	`/api/appointments/{id}/`	Get appointment	Authenticated
PUT	`/api/appointments/{id}/`	Update appointment	Authenticated
DELETE	`/api/appointments/{id}/`	Delete appointment	Admin

Encounters

Method	Endpoint	Description	Permissions
GET	`/api/encounters/`	List encounters	Authenticated
POST	`/api/encounters/`	Create encounter	Doctor
GET	`/api/encounters/{id}/`	Get encounter	Authenticated

Method	Endpoint	Description	Permissions
PUT	`/api/encounters/{id}/`	Update encounter	Doctor
DELETE	`/api/encounters/{id}/`	Delete encounter	Admin

Pharmacy

Method	Endpoint	Description	Permissions
GET	`/api/medications/`	List medications	Authenticated
POST	`/api/medications/`	Add medication	Admin
GET	`/api/medications/{id}/`	Get medication	Authenticated
PUT	`/api/medications/{id}/`	Update medication	Admin
DELETE	`/api/medications/{id}/`	Delete medication	Admin

Method	Endpoint	Description	Permissions
GET	`/api/prescriptions/`	List prescriptions	Authenticated
POST	`/api/prescriptions/`	Create prescription	Doctor
GET	`/api/prescriptions/{id}/`	Get prescription	Authenticated
PUT	`/api/prescriptions/{id}/`	Update prescription	Doctor
DELETE	`/api/prescriptions/{id}/`	Delete prescription	Admin

Billing

Method	Endpoint	Description	Permissions
GET	`/api/bills/`	List bills	Authenticated
POST	`/api/bills/`	Create bill	Authenticated
GET	`/api/bills/{id}/`	Get bill	Authenticated
PUT	`/api/bills/{id}/`	Update bill	Authenticated
DELETE	`/api/bills/{id}/`	Delete bill	Admin

Method	Endpoint	Description	Permissions
GET	`/api/payments/`	List payments	Authenticated
POST	`/api/payments/`	Record payment	Authenticated
GET	`/api/payments/{id}/`	Get payment	Authenticated
PUT	`/api/payments/{id}/`	Update payment	Admin
DELETE	`/api/payments/{id}/`	Delete payment	Admin

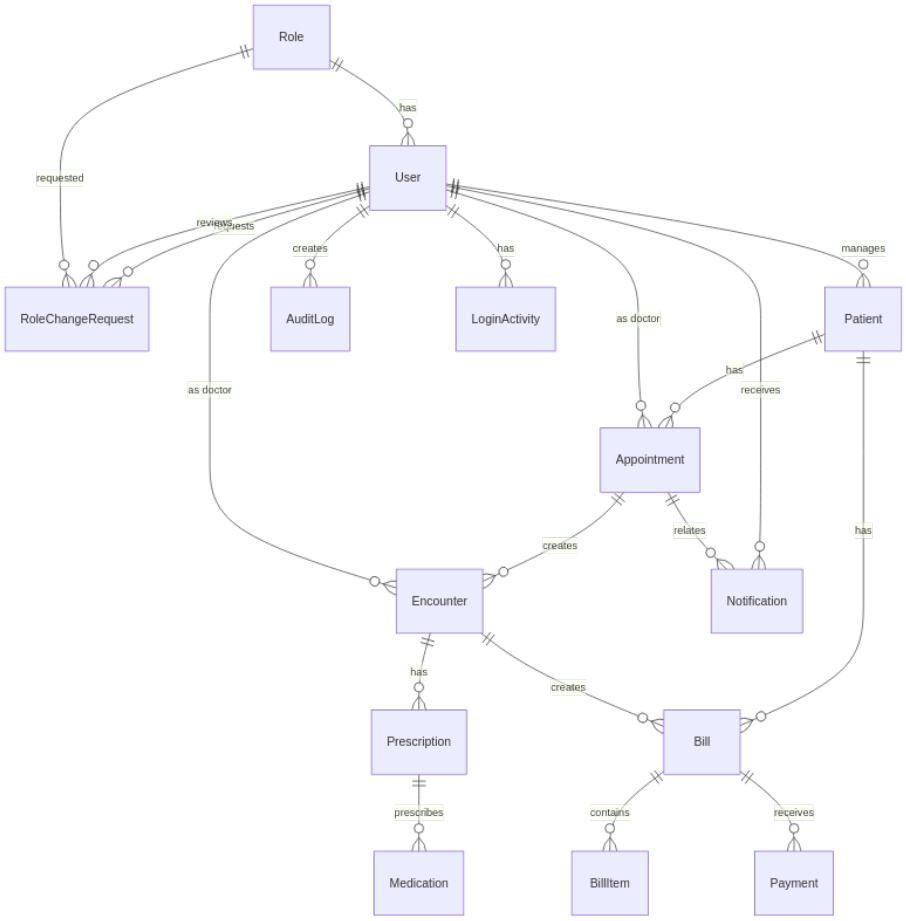

Figure 7: Diagram 7

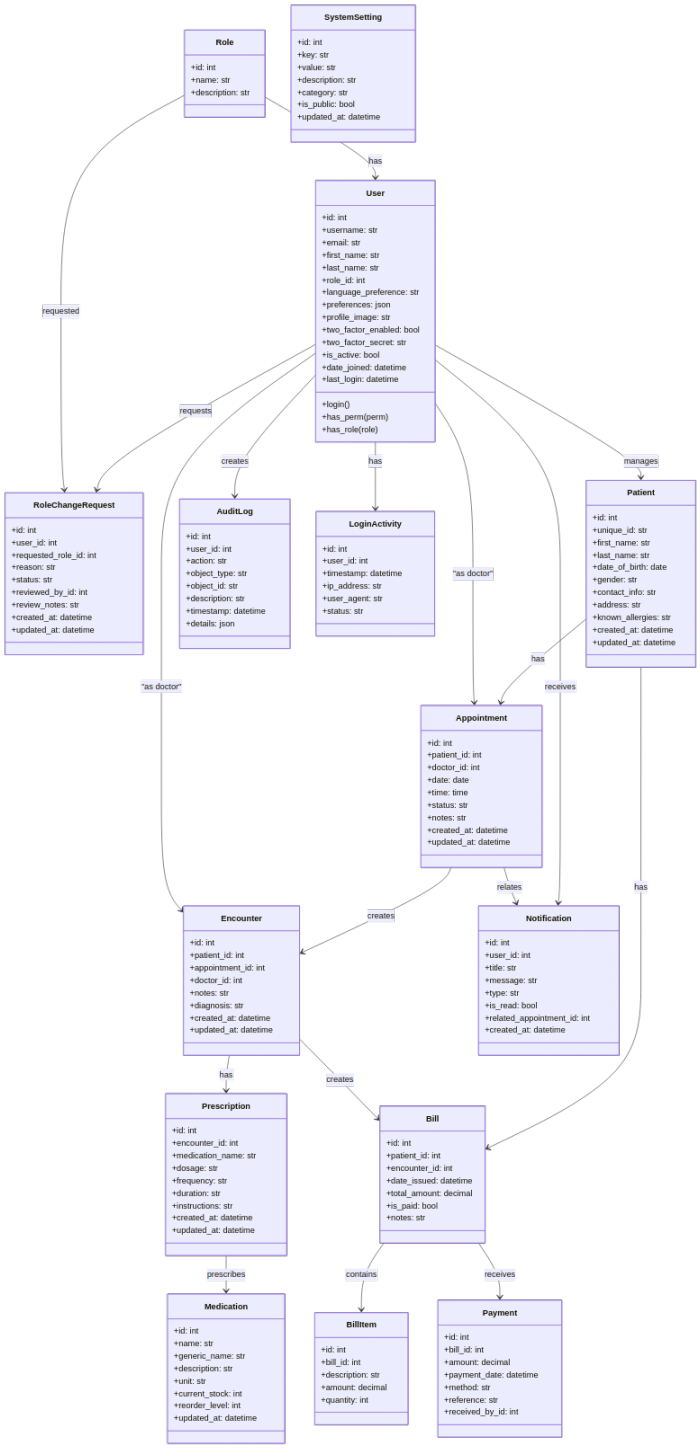

Figure 8: Diagram 8

Role Change Requests Method

Endpoint

Description

Permissions

GET

/api/role-change-requests/

List requests

Authenticated*

POST

/api/role-change-requests/

Create request

Authenticated

GET

/api/role-change-requests/{id}/

Get request

Authenticated*

PUT

/api/role-change-requests/{id}/

Update request

Owner

DELETE

/api/role-change-requests/{id}/

Delete request

Admin

POST

/api/role-change-requests/{id}/approve/

Approve request

Admin

POST

/api/role-change-requests/{id}/reject/

Reject request

Admin

POST

/api/role-change-requests/assign_role/

Direct role assignment

Admin

*Users see only their own requests; Admins see all requests

Reporting Endpoints

Method

Endpoint

Description

Permissions

GET

/api/dashboard/

Main dashboard data

Authenticated

GET

/api/dashboard-stats/

Dashboard statistics

Authenticated

GET

/api/report/patient_count/

Patient count report

Authenticated

GET

/api/report/appointments_today/

Today's appointments

Authenticated

GET

/api/report/appointments_by_doctor/

Appointments by doctor

Authenticated

GET

/api/report/top_prescribed_medications/

Top medications

Authenticated

GET

/api/report/billing-stats/

Billing statistics

Authenticated

System Endpoints

Method	Endpoint	Description	Permissions
GET	`/api/notifications/`	User notifications	Authenticated
GET	`/api/audit-logs/`	Audit logs	Admin
GET	`/api/login-activity/`	Login activity	Admin
GET	`/api/system-settings/`	System settings	Admin
GET	`/api/profile/`	User profile	Authenticated
PUT	`/api/profile/`	Update profile	Authenticated
GET	`/api/preferences/`	User preferences	Authenticated
PUT	`/api/preferences/`	Update preferences	Authenticated
GET	`/api/health/`	Health check	AllowAny
POST	`/api/sync_offline_data/`	Sync offline data	Authenticated

Authentication & Authorization

JWT Authentication

The system uses JWT tokens for authentication:

Token Obtain: POST /api/auth/ with username/email and password

Token Refresh: POST /api/auth/refresh/ with refresh token

Authorization: Include Authorization: Bearer <access_token> header

Role-Based Access Control

Role	Permissions
Admin	Full system access, user management, system settings
Doctor	Patient care, prescriptions, encounters, appointments
Nurse	Patient care, basic medical tasks
Receptionist	Appointments, patient registration, basic billing
Patient	View own records, appointments, bills (future feature)

Custom Permissions

- `IsAdminOrReadOnly`: Admin full access, others read-only
- `IsDoctorOrReadOnly`: Doctors full access, others read-only
- `IsReceptionistOrReadOnly`: Receptionists full access, others read-only

Data Serialization

Request/Response Formats

All API endpoints use JSON format. Example request/response structures:

User Registration

```
POST /api/register/
{
  "username": "johndoe",
  "email": "john@example.com",
  "password": "securepassword",
  "first_name": "John",
  "last_name": "Doe",
  "role_id": 2
}
```

Patient Creation

```
POST /api/patients/
{
  "first_name": "Jane",
  "last_name": "Smith",
  "date_of_birth": "1990-01-15",
  "gender": "F",
  "contact_info": "+1234567890",
  "address": "123 Main St",
  "known_allergies": "Penicillin"
}
```

Appointment Creation

```
POST /api/appointments/
{
  "patient": 1,
  "doctor": 2,
  "date": "2025-01-15",
  "time": "10:00:00",
  "notes": "Regular checkup"
}
```

Background Tasks & Real-time Features

Celery Integration

The system uses Celery for background task processing:

- **Email notifications** for appointments
- **Report generation**
- **Data synchronization**
- **Audit log processing**

WebSocket Communication

Using Django Channels for real-time features:

- **Live notifications**
- **Real-time appointment updates**
- **Dashboard live data**

Security Features

Implemented Security Measures

1. **JWT Authentication** with token refresh
2. **Brute force protection** via Django Axes
3. **CORS protection** with allowed origins
4. **CSRF protection** on sensitive endpoints
5. **Input validation** via DRF serializers
6. **SQL injection prevention** via Django ORM
7. **Audit logging** for all critical operations
8. **Two-factor authentication** support (configurable)

Security Headers

- `X-Frame-Options: DENY`
- `X-Content-Type-Options: nosniff`
- `X-XSS-Protection: 1; mode=block`
- CORS headers configured

Setup & Installation

Prerequisites

- Python 3.10+
- Node.js 18+
- PostgreSQL 15+
- Redis
- Docker (optional)

Backend Setup

1. Create and activate a Python virtual environment.
2. Install dependencies from `requirements.txt`.
3. Configure environment variables in `.env`.
4. Run migrations and start the Django server.

Database Setup

- Use Docker or install PostgreSQL natively.
- Create database and user as per documentation.

Environment Variables

Variable	Description	Default
`DEBUG`	Enable debug mode	`True`
`SECRET_KEY`	Django secret key	Generated
`DB_NAME`	Database name	`chelal_hms`
`DB_USER`	Database user	`postgres`
`DB_PASSWORD`	Database password	`''`
`DB_HOST`	Database host	`localhost`
`DB_PORT`	Database port	`5432`
`REDIS_URL`	Redis URL for Celery	`redis://localhost:6379/0`

Running Locally

```
# Install dependencies
pip install -r requirements.txt

# Run migrations
python manage.py migrate

# Create superuser
python manage.py createsuperuser

# Run development server
```

```
python manage.py runserver

# Run Celery worker
celery -A Backend worker -l info
```

Deployment & Configuration

Docker Deployment

The application is containerized with Docker:

- **Backend**: Gunicorn with 1 worker, 2 threads
- **Database**: PostgreSQL 15
- **Cache/Queue**: Redis Alpine
- **Reverse Proxy**: Nginx

Docker Compose

- Services: PostgreSQL, Redis, Django backend, frontend
- Use `docker-compose up -d` to start all services

Production Considerations

1. **Database**: Use connection pooling
2. **Caching**: Implement Redis caching
3. **Static Files**: Use CDN for static assets
4. **Monitoring**: Implement health checks and logging
5. **Backup**: Regular database backups
6. **SSL/TLS**: HTTPS everywhere
7. **Rate Limiting**: Implement API rate limiting

Development & Testing

Testing

```
# Run all tests
python manage.py test

# Run with coverage
coverage run manage.py test
coverage report
```

API Testing

Use the included Postman collection (`chelal_backend_api.postman_collection.json`) for testing all endpoints.

Troubleshooting

Common Issues

- Database connection errors
- Docker container startup issues
- Authentication failures
- API endpoint not found

Solutions

- Check environment variables and .env files
- Ensure all migrations are applied
- Review logs for errors
- Restart Docker containers if needed

Database Monitoring

```
-- Active connections
SELECT count(*) FROM pg_stat_activity;

-- Table sizes
SELECT schemaname, tablename, pg_size_pretty(pg_total_relation_size(schemaname||'.'||tablena
FROM pg_tables
WHERE schemaname = 'public'
ORDER BY pg_total_relation_size(schemaname||'.'||tablename) DESC;

-- Index usage
SELECT indexname, idx_scan, idx_tup_read, idx_tup_fetch
FROM pg_stat_user_indexes
ORDER BY idx_scan DESC;
```

Future Enhancements

Planned Features

1. **Patient Portal**: Self-service patient features
2. **Telemedicine**: Video consultation integration
3. **Advanced Analytics**: ML-based insights
4. **Mobile App**: React Native mobile application
5. **Integration APIs**: HL7 FHIR compliance
6. **Advanced Reporting**: Custom report builder
7. **Inventory Management**: Advanced pharmacy features
8. **Emergency Module**: Emergency response features

Scalability Considerations

1. **Database Sharding**: For multi-tenant deployments

2. **Microservices**: Break down monolithic architecture
3. **API Gateway**: Centralized API management
4. **Message Queue**: Advanced queuing with RabbitMQ
5. **CDN Integration**: Global content delivery
6. **Load Balancing**: Multiple application instances

This comprehensive backend documentation reflects the current state of the CHELAL Hospital Management System backend as of September 27, 2025.