EXPERIMENT 5

CODE :

routers

auth.py

```python
from fastapi import APIRouter, Depends, HTTPException, status, Request
from sqlalchemy.orm import Session
from slowapi import Limiter
from slowapi.util import get_remote_address

from ..db import get_db
from .. import schemas, models
from ..security import hash_password, verify_password, create_access_token
from ..models import Role, User

router = APIRouter(prefix="/auth", tags=["auth"])
limiter = Limiter(key_func=get_remote_address)

@router.post("/register", response_model=schemas.UserOut)
@limiter.limit("5/minute")
def register(request: Request, payload: schemas.UserCreate, db: Session = Depends(get_db)):
    if db.query(models.User).filter(models.User.username == payload.username).first():
        raise HTTPException(status_code=400, detail="Username already exists")
    user = models.User(username=payload.username, password_hash=hash_password(payload.password), role=payload.role)
    db.add(user)
    db.commit()
    db.refresh(user)
    return user

@router.post("/login", response_model=schemas.Token)
@limiter.limit("10/minute")
def login(request: Request, payload: schemas.Login, db: Session = Depends(get_db)):
    user = db.query(models.User).filter(models.User.username == payload.username).first()
    if not user or not verify_password(payload.password, user.password_hash):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid credentials")
    token = create_access_token(sub=user.username, role=user.role.value)
    return {"access_token": token}

```

Code 1.1

patients.py

```python
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from ..db import get_db
from .. import models, schemas
from ..deps import require_role
from ..models import Role
from ..security import encrypt_field, decrypt_field

router = APIRouter(prefix="/patients", tags=["patients"])

@router.post("/profile", response_model=schemas.PatientProfileOut)
def create_or_update_profile(payload: schemas.PatientProfileCreate, db: Session = Depends(get_db), user=Depends(require_role(Role.PATIENT))):
    profile = db.query(models.PatientProfile).filter(models.PatientProfile.user_id == user.id).first()
    if not profile:
        profile = models.PatientProfile(user_id=user.id, full_name=payload.full_name, dob=payload.dob,
                                        medical_history_enc=encrypt_field(payload.medical_history) if payload.medical_history else None)
        db.add(profile)
    else:
        profile.full_name = payload.full_name
        profile.dob = payload.dob
        profile.medical_history_enc = encrypt_field(payload.medical_history) if payload.medical_history else None
    db.commit()
    db.refresh(profile)
    out = schemas.PatientProfileOut(id=profile.id, full_name=profile.full_name, dob=profile.dob,
                                    medical_history=decrypt_field(profile.medical_history_enc) if profile.medical_history_enc else None)
    return out

@router.get("/profile/me", response_model=schemas.PatientProfileOut)
def get_my_profile(db: Session = Depends(get_db), user=Depends(require_role(Role.PATIENT))):
    profile = db.query(models.PatientProfile).filter(models.PatientProfile.user_id == user.id).first()
    if not profile:
        raise HTTPException(status_code=404, detail="Profile not found")
    return schemas.PatientProfileOut(id=profile.id, full_name=profile.full_name, dob=profile.dob,
                                     medical_history=decrypt_field(profile.medical_history_enc) if profile.medical_history_enc else None)
```

Code 2.1

pharmacy.py

```python
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from ..db import get_db
from .. import models
from ..deps import require_role
from ..models import Role, PrescriptionStatus

router = APIRouter(prefix="/pharmacy", tags=["pharmacy"])

@router.get("/pending")
def pending(db: Session = Depends(get_db), pharmacist=Depends(require_role(Role.PHARMACIST))):
    items = db.query(models.Prescription).filter(models.Prescription.status == PrescriptionStatus.ACTIVE).all()
    return [{"id": p.id, "patient_id": p.patient_id, "doctor_id": p.doctor_id, "status": p.status} for p in items]

@router.post("/{prescription_id}/validate")
def validate(prescription_id: int, db: Session = Depends(get_db), pharmacist=Depends(require_role(Role.PHARMACIST))):
    p = db.query(models.Prescription).filter(models.Prescription.id == prescription_id).first()
    if not p:
        raise HTTPException(status_code=404, detail="Prescription not found")
    # In a real system you'd record checks; here we just acknowledge
    return {"ok": True, "validated_by": pharmacist.username}

@router.post("/{prescription_id}/dispense")
def dispense(prescription_id: int, db: Session = Depends(get_db), pharmacist=Depends(require_role(Role.PHARMACIST))):
    p = db.query(models.Prescription).filter(models.Prescription.id == prescription_id).first()
    if not p:
        raise HTTPException(status_code=404, detail="Prescription not found")
    p.status = PrescriptionStatus.FULFILLED
    log = models.DispenseLog(pharmacist_id=pharmacist.id, prescription_id=p.id)
    db.add(log)
    db.commit()
    return {"ok": True}
```

Code 3.1

prescription.py

```python
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from ..db import get_db
from .. import models, schemas
from ..deps import require_role, get_current_user
from ..models import Role, PrescriptionStatus
from ..security import encrypt_field, decrypt_field

router = APIRouter(prefix="/prescriptions", tags=["prescriptions"])

@router.post("", response_model=schemas.PrescriptionOut)
def create_prescription(payload: schemas.PrescriptionCreate, db: Session = Depends(get_db), doctor=Depends(require_role(Role.DOCTOR))):
    patient = db.query(models.User).filter(models.User.id == payload.patient_id, models.User.role == Role.PATIENT).first()
    if not patient:
        raise HTTPException(status_code=404, detail="Patient not found")
    pres = models.Prescription(
        doctor_id=doctor.id,
        patient_id=payload.patient_id,
        drug_name_enc=encrypt_field(payload.drug_name),
        dosage_enc=encrypt_field(payload.dosage),
        frequency=payload.frequency,
        start_date=payload.start_date,
        end_date=payload.end_date,
    )
    db.add(pres)
    db.commit()
    db.refresh(pres)
    return schemas.PrescriptionOut(
        id=pres.id,
        doctor_id=pres.doctor_id,
        patient_id=pres.patient_id,
        drug_name=payload.drug_name,
        dosage=payload.dosage,
        frequency=pres.frequency,
        start_date=pres.start_date,
        end_date=pres.end_date,
        status=pres.status,
```

Code 4.1

```python
@router.get("/my", response_model=list[schemas.PrescriptionOut])
def my_prescriptions(db: Session = Depends(get_db), user=Depends(get_current_user)):
    q = db.query(models.Prescription)
    if user.role == Role.PATIENT:
        q = q.filter(models.Prescription.patient_id == user.id)
    elif user.role == Role.DOCTOR:
        q = q.filter(models.Prescription.doctor_id == user.id)
    else:
        raise HTTPException(status_code=403, detail="Forbidden")
    res = []
    for p in q.all():
        res.append(schemas.PrescriptionOut(
            id=p.id,
            doctor_id=p.doctor_id,
            patient_id=p.patient_id,
            drug_name=decrypt_field(p.drug_name_enc),
            dosage=decrypt_field(p.dosage_enc),
            frequency=p.frequency,
            start_date=p.start_date,
            end_date=p.end_date,
            status=p.status,
            created_at=p.created_at,
        ))
    return res

@router.post("/{prescription_id}/cancel")
def cancel_prescription(prescription_id: int, db: Session = Depends(get_db), doctor=Depends(require_role(Role.DOCTOR))):
    p = db.query(models.Prescription).filter(models.Prescription.id == prescription_id, models.Prescription.doctor_id == doctor.id).first()
    if not p:
        raise HTTPException(status_code=404, detail="Prescription not found")
    p.status = PrescriptionStatus.CANCELED
    db.commit()
    return {"ok": True}
```

Code 4.2

middleware.py

```python
from fastapi import Request
from fastapi.responses import JSONResponse
from starlette.middleware.cors import CORSMiddleware
from starlette.middleware.base import BaseHTTPMiddleware
from slowapi import Limiter
from slowapi.util import get_remote_address

from .config import settings

limiter = Limiter(key_func=get_remote_address, default_limits=["200/minute"])  # global rate limit

class SecurityHeadersMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        resp = await call_next(request)
        resp.headers["X-Content-Type-Options"] = "nosniff"
        resp.headers["X-Frame-Options"] = "DENY"
        resp.headers["Referrer-Policy"] = "no-referrer"
        resp.headers["Permissions-Policy"] = "geolocation=()"
        return resp


def add_middlewares(app):
    app.add_middleware(CORSMiddleware, allow_origins=[o.strip() for o in settings.CORS_ORIGINS.split(",")],
                       allow_credentials=True, allow_methods=["*"], allow_headers=["*"])
    app.add_middleware(SecurityHeadersMiddleware)

```

Code 5.1

models.py

```python
from sqlalchemy import Column, Integer, String, Date, DateTime, Enum, ForeignKey, Text, Boolean
from sqlalchemy.orm import relationship
from enum import Enum as PyEnum
from datetime import datetime
from .db import Base

class Role(str, PyEnum):
    DOCTOR = "doctor"
    PATIENT = "patient"
    PHARMACIST = "pharmacist"
    ADMIN = "admin"

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    username = Column(String(64), unique=True, index=True, nullable=False)
    password_hash = Column(String(256), nullable=False)
    role = Column(Enum(Role), nullable=False, index=True)
    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)

    patient_profile = relationship("PatientProfile", back_populates="user", uselist=False)

class PatientProfile(Base):
    __tablename__ = "patient_profiles"
    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE"), unique=True)
    full_name = Column(String(128), nullable=False)
    dob = Column(Date, nullable=True)
    medical_history_enc = Column(Text, nullable=True)  # AES-GCM encrypted, base64

    user = relationship("User", back_populates="patient_profile")

class PrescriptionStatus(str, PyEnum):
    ACTIVE = "active"
    CANCELED = "canceled"
    FULFILLED = "fulfilled"
```

Code 6.1

```python
class Prescription(Base):
    __tablename__ = "prescriptions"
    id = Column(Integer, primary_key=True)
    doctor_id = Column(Integer, ForeignKey("users.id", ondelete="SET NULL"), index=True)
    patient_id = Column(Integer, ForeignKey("users.id", ondelete="CASCADE"), index=True)
    drug_name_enc = Column(Text, nullable=False)
    dosage_enc = Column(Text, nullable=False)
    frequency = Column(String(64), nullable=False)  # e.g., "2x/day"
    start_date = Column(Date, nullable=False)
    end_date = Column(Date, nullable=True)
    status = Column(Enum(PrescriptionStatus), default=PrescriptionStatus.ACTIVE, index=True)
    created_at = Column(DateTime, default=datetime.~~utcnow~~, nullable=False)

class DispenseLog(Base):
    __tablename__ = "dispense_logs"
    id = Column(Integer, primary_key=True)
    pharmacist_id = Column(Integer, ForeignKey("users.id", ondelete="SET NULL"))
    prescription_id = Column(Integer, ForeignKey("prescriptions.id", ondelete="CASCADE"), index=True)
    timestamp = Column(DateTime, default=datetime.~~utcnow~~, nullable=False)
    status = Column(String(32), default="dispensed")

class AuditLog(Base):
    __tablename__ = "audit_logs"
    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("users.id", ondelete="SET NULL"), nullable=True)
    action = Column(String(256), nullable=False)
    ip = Column(String(64), nullable=True)
    at = Column(DateTime, default=datetime.~~utcnow~~, index=True)
    success = Column(Boolean, default=True)

```

Code 6.2

schemas.py

```python
from pydantic import BaseModel, Field
from typing import Optional, List, Annotated
from datetime import date, datetime
from .models import Role, PrescriptionStatus

class Token(BaseModel):
    access_token: str
    token_type: str = "bearer"

class UserCreate(BaseModel):
    username: Annotated[str, Field(min_length=3, max_length=64)]
    password: Annotated[str, Field(min_length=8)]
    role: Role

class UserOut(BaseModel):
    id: int
    username: str
    role: Role
    model_config = {"from_attributes": True}

class Login(BaseModel):
    username: str
    password: str

class PatientProfileCreate(BaseModel):
    full_name: str
    dob: Optional[date] = None
    medical_history: Optional[str] = None

class PatientProfileOut(BaseModel):
    id: int
    full_name: str
    dob: Optional[date]
    medical_history: Optional[str] = None
    model_config = {"from_attributes": True}

```

Code 7.1

```python
class PrescriptionCreate(BaseModel):
    patient_id: int
    drug_name: str = Field(..., examples=["Amoxicillin 500mg"])
    dosage: str = Field(..., examples=["1 capsule twice daily after food"])
    frequency: str = Field(..., examples=["BID"])
    start_date: date
    end_date: Optional[date]

class PrescriptionOut(BaseModel):
    id: int
    doctor_id: int
    patient_id: int
    drug_name: str
    dosage: str
    frequency: str
    start_date: date
    end_date: Optional[date]
    status: PrescriptionStatus
    created_at: datetime
    model_config = {"from_attributes": True}

```

Code 7.2

security.py

```python
import base64
import os
from datetime import datetime, timedelta
from typing import Optional

import jwt
from argon2 import PasswordHasher
from argon2.low_level import Type
from cryptography.hazmat.primitives.ciphers.aead import AESGCM

from .config import settings

ph = PasswordHasher(time_cost=settings.ARGON2_TIME_COST,memory_cost=settings.ARGON2_MEMORY_COST,parallelism=settings.ARGON2_PARALLELISM,

def hash_password(password: str) -> str:
    return ph.hash(password)

def verify_password(password: str, password_hash: str) -> bool:
    try:
        ph.verify(password_hash, password)
        return True
    except Exception:
        return False

# JWT

def create_access_token(sub: str, role: str, expires_minutes: int = settings.JWT_EXPIRE_MINUTES) -> str:
    expire = datetime.utcnow() + timedelta(minutes=expires_minutes)
    to_encode = {"sub": sub, "role": role, "exp": expire}
    return jwt.encode(to_encode, settings.JWT_SECRET, algorithm=settings.JWT_ALG)

def decode_token(token: str) -> Optional[dict]:
    try:
        return jwt.decode(token, settings.JWT_SECRET, algorithms=[settings.JWT_ALG])
    except jwt.PyJWTError:
        return None
```

Code 8.1

```python
# AES-GCM 256 helpers for field encryption
_key = base64.b64decode(settings.AES256_KEY_B64)

def encrypt_field(plaintext: str) -> str:
    if plaintext is None:
        return None
    aesgcm = AESGCM(_key)
    nonce = os.urandom(12)
    ct = aesgcm.encrypt(nonce, plaintext.encode(), None)
    return base64.b64encode(nonce + ct).decode()

def decrypt_field(ciphertext_b64: str) -> Optional[str]:
    if ciphertext_b64 is None:
        return None
    raw = base64.b64decode(ciphertext_b64)
    nonce, ct = raw[:12], raw[12:]
    aesgcm = AESGCM(_key)
    pt = aesgcm.decrypt(nonce, ct, None)
    return pt.decode()
```

Code 8.2

SCREENSHOTS :

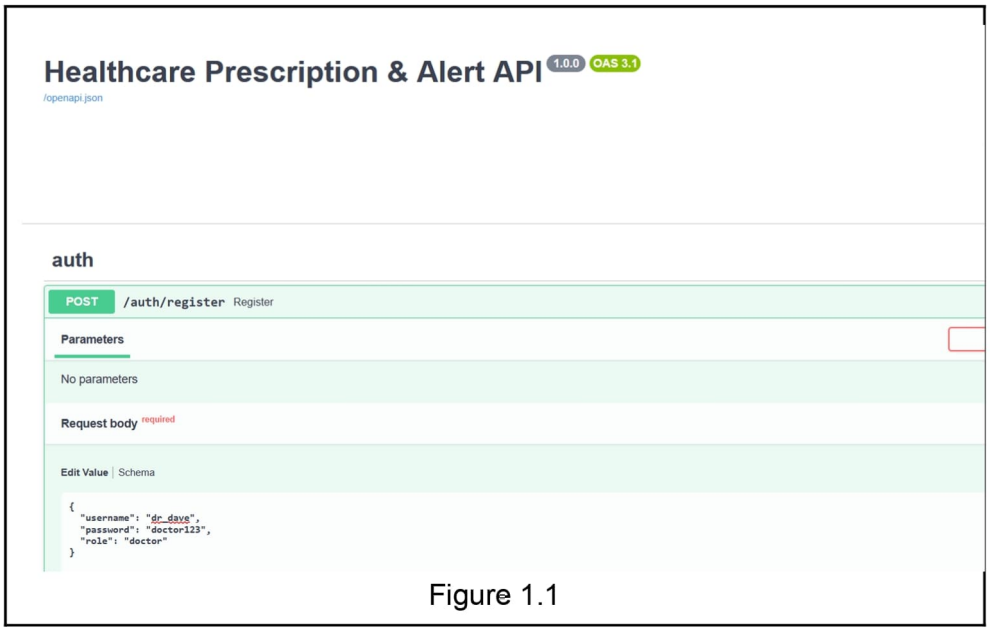

Figure 1.1

Figure 1.2

Server response

Code	Details
200	Response body

```
{
  "id": 2,
  "username": "dr_dave",
  "role": "doctor"
}
```

Response headers

```
access-control-allow-credentials: true
content-length: 45
content-type: application/json
date: Sun,24 Aug 2025 17:28:54 GMT
permissions-policy: geolocation=()
referrer-policy: no-referrer
server: uvicorn
x-content-type-options: nosniff
x-frame-options: DENY
```

Figure 1.3

POST /auth/login Login

Parameters

No parameters

Request body required

Edit Value | Schema

```
{
  "username": "dr_dave",
  "password": "doctor123"
}
```

Figure 1.4

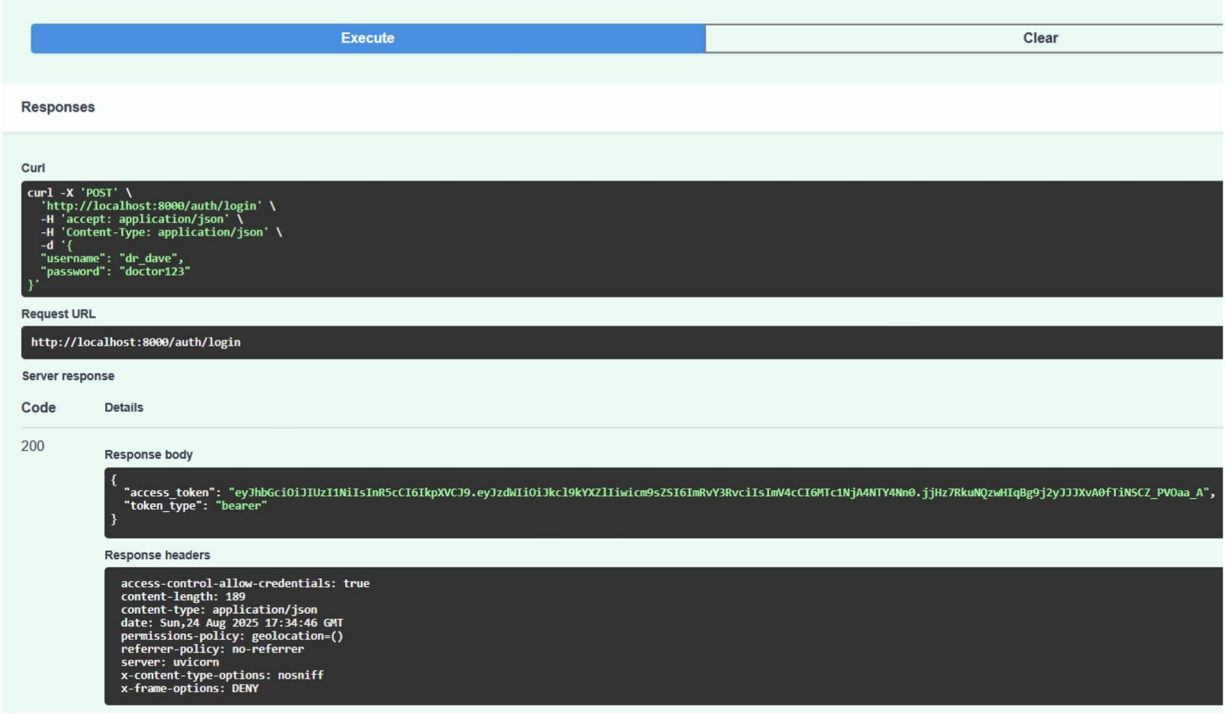

Figure 1.5

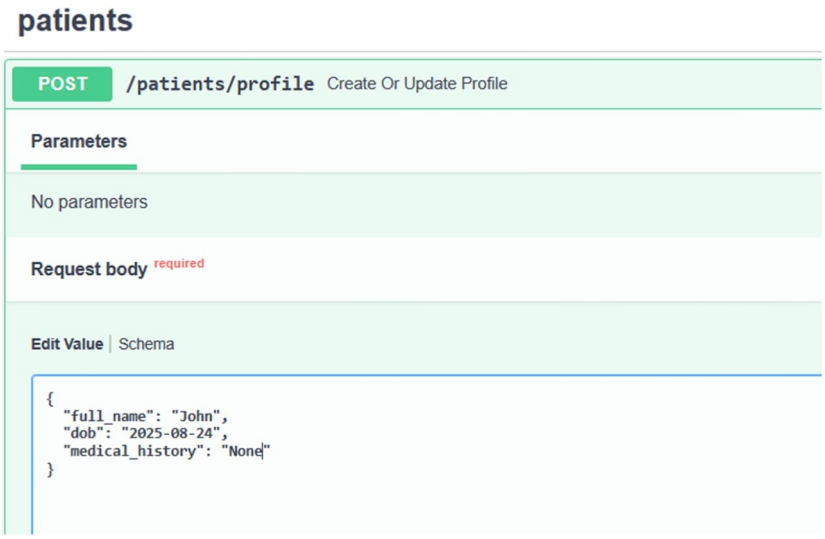

Figure 1.6

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/patients/profile' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJqb2huIiwicm9sZSI6InBhdGllbnQiLCJleHAiOjE3NTYwODgzODh9.CDFYpROGW_fgBbmtOZv_Gkp30qTFj-L6ANfmpyMSJFg' \
  -H 'Content-Type: application/json' \
  -d '{
  "full_name": "John Doe",
  "dob": "2025-08-24",
  "medical_history": "None"
}'
```

Request URL

```
http://localhost:8000/patients/profile
```

Server response

Code	Details
200	Response body

```
{
  "id": 1,
  "full_name": "John Doe",
  "dob": "2025-08-24",
  "medical_history": "None"
}
```

Response headers

```
access-control-allow-credentials: true
content-length: 75
content-type: application/json
date: Sun,24 Aug 2025 18:25:31 GMT
permissions-policy: geolocation=()
referrer-policy: no-referrer
server: uvicorn
x-content-type-options: nosniff
x-frame-options: DENY
```

Figure 1.7

GET /patients/profile/me Get My Profile

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8000/patients/profile/me' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJqb2huIiwicm9sZSI6InBhdGllbnQiLCJleHAiOjE3NTYwODgzODh9.CDFYpROGW_fgBbmtOZv_Gkp30qTFj-L6ANfmpyMSJFg'
```

Request URL

```
http://localhost:8000/patients/profile/me
```

Server response

Code	Details
200	Response body

```
{
  "id": 1,
  "full_name": "John Doe",
  "dob": "2025-08-24",
  "medical_history": "None"
}
```

Response headers

```
content-length: 75
content-type: application/json
date: Sun,24 Aug 2025 18:26:29 GMT
permissions-policy: geolocation=()
referrer-policy: no-referrer
server: uvicorn
```

Figure 1.8

prescriptions

POST `/prescriptions` Create Prescription

Parameters

No parameters

Request body required

Edit Value | Schema

```
{
  "patient_id": 4,
  "drug_name": "Amoxicillin 500mg",
  "dosage": "1 capsule twice daily after food",
  "frequency": "BID",
  "start_date": "2025-08-24",
  "end_date": "2025-08-24"
}
```

Figure 1.9

Curl

```
curl -X 'POST' \
  'http://localhost:8000/prescriptions' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkcl9kYXZlIiwicm9sZSI6ImRvY3RvciIsImV4cCI6MTc1NjA4ODkzMX0.YuHzTjBi4A1aCctGJYn9gpf-w6_BexGIb9-_yaR3bIU' \
  -H 'Content-Type: application/json' \
  -d '{
  "patient_id": 4,
  "drug_name": "Amoxicillin 500mg",
  "dosage": "1 capsule twice daily after food",
  "frequency": "BID",
  "start_date": "2025-08-24",
  "end_date": "2025-08-24"
}'
```

Request URL

```
http://localhost:8000/prescriptions
```

Server response

Code	Details
200	**Response body**

```
{
  "id": 1,
  "doctor_id": 2,
  "patient_id": 4,
  "drug_name": "Amoxicillin 500mg",
  "dosage": "1 capsule twice daily after food",
  "frequency": "BID",
  "start_date": "2025-08-24",
  "end_date": "2025-08-24",
  "status": "active",
  "created_at": "2025-08-24T18:29:23.869069"
}
```

Response headers

```
access-control-allow-credentials: true
content-length: 241
content-type: application/json
date: Sun,24 Aug 2025 18:29:22 GMT
permissions-policy: geolocation=()
referrer-policy: no-referrer
server: uvicorn
x-content-type-options: nosniff
x-frame-options: DENY
```

Figure 1.10

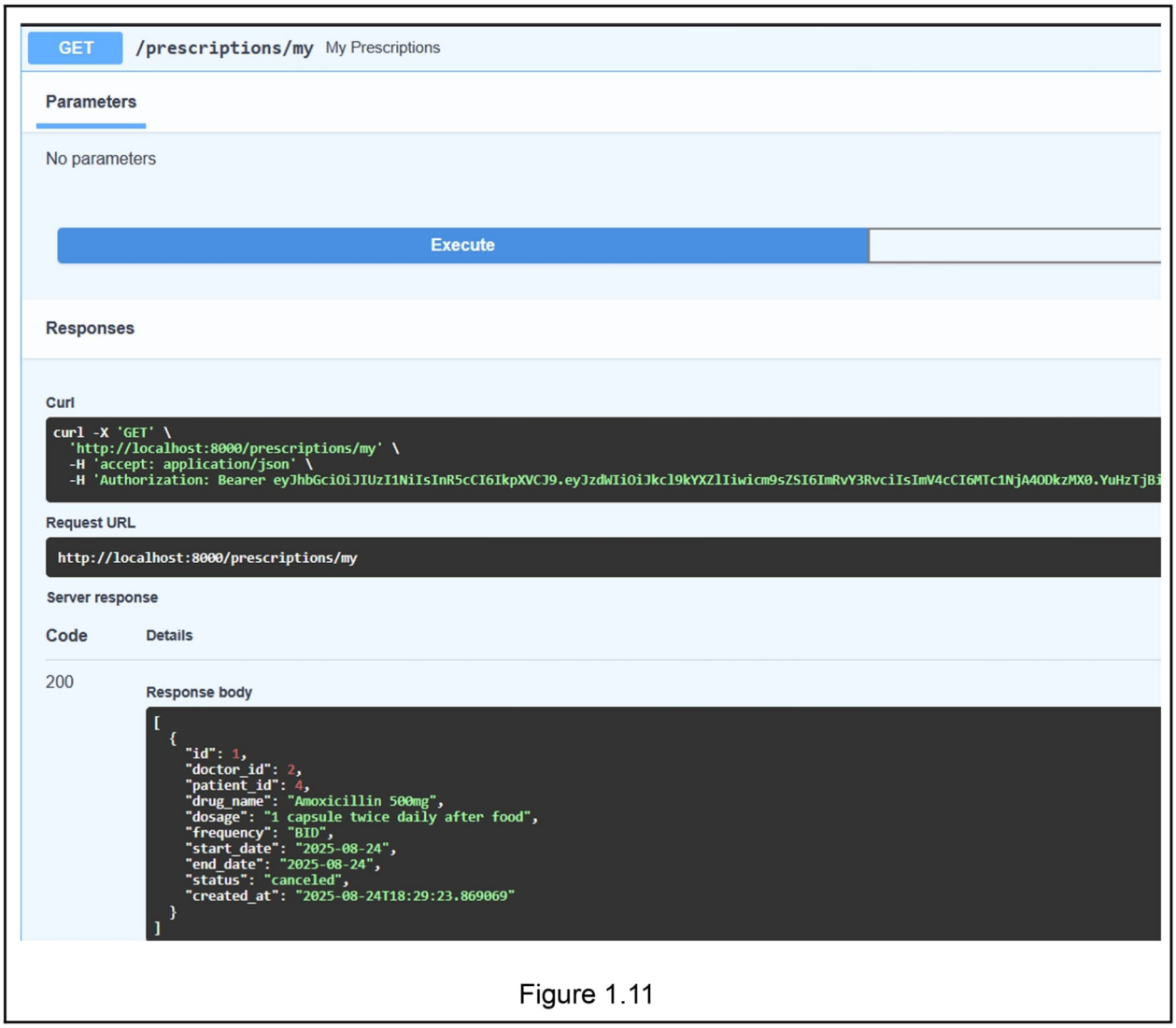
GET
/prescriptions/my
My Prescriptions
Parameters
No parameters
Execute
Responses
Curl
curl -X 'GET' \
'http://localhost:8000/prescriptions/my' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkcl9kYXZlIiwicm9sZSI6ImRvY3RvciIsImV4cCI6MTc1NjA4ODkzMX0.YuHzTjB
Request URL
http://localhost:8000/prescriptions/my
Server response
Code
Details
200
Response body
[
{
"id": 1,
"doctor_id": 2,
"patient_id": 4,
"drug_name": "Amoxicillin 500mg",
"dosage": "1 capsule twice daily after food",
"frequency": "BID",
"start_date": "2025-08-24",
"end_date": "2025-08-24",
"status": "canceled",
"created_at": "2025-08-24T18:29:23.869069"
}
]

Figure 1.11

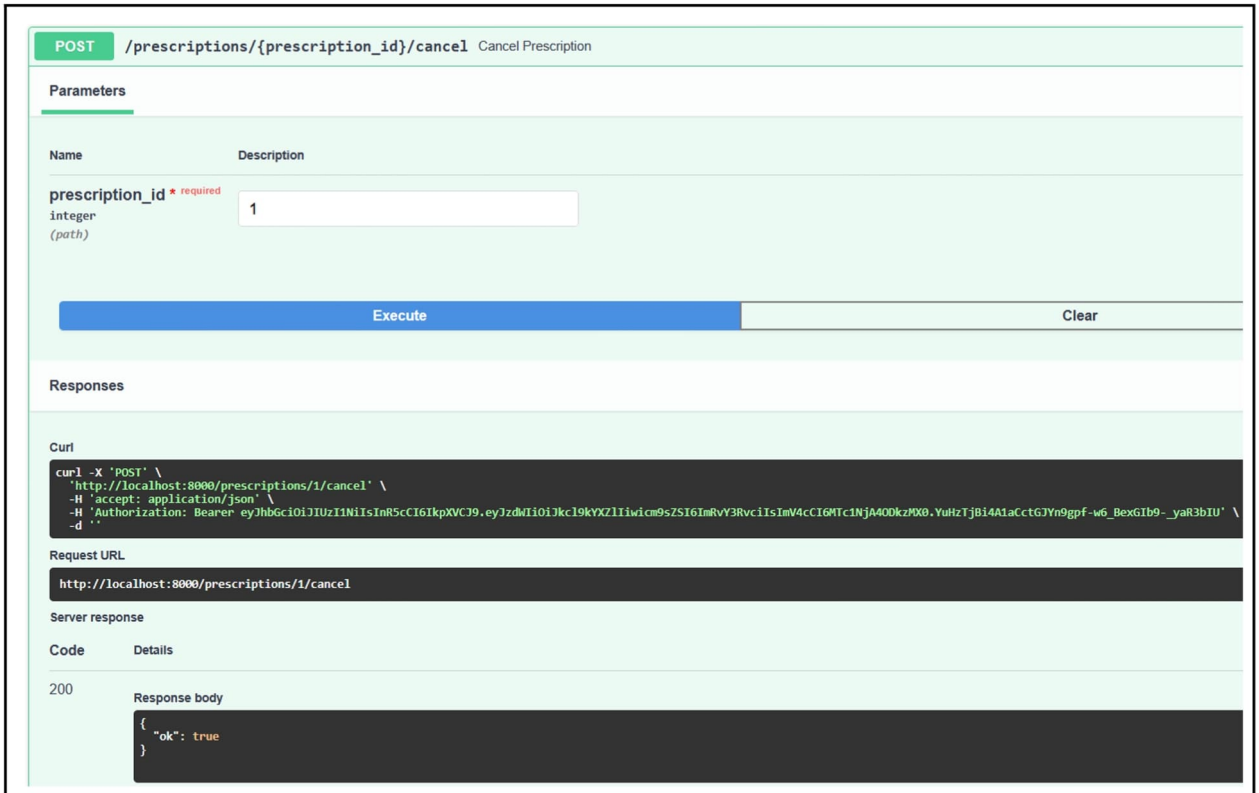

Figure 1.12

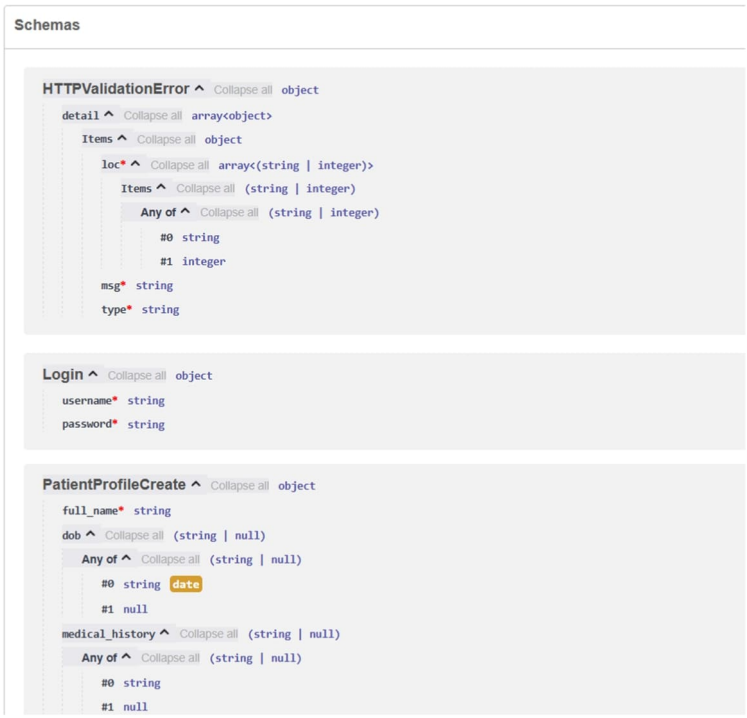

Figure 1.13

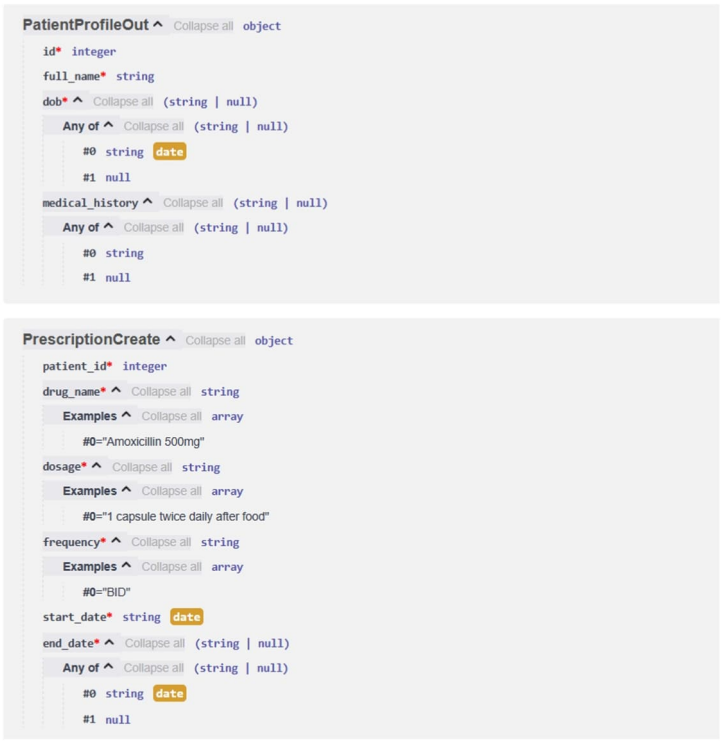

Figure 1.14

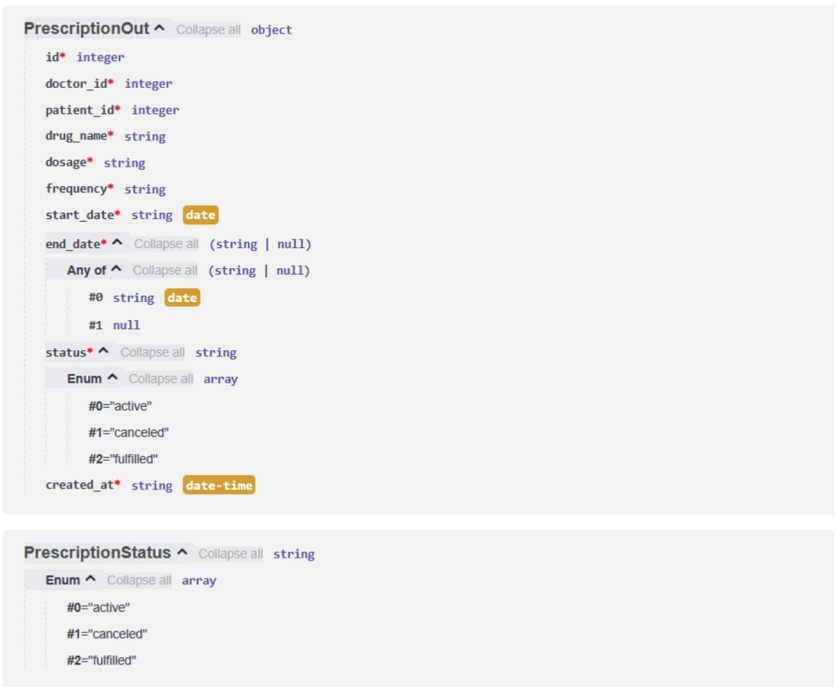

Figure 1.15

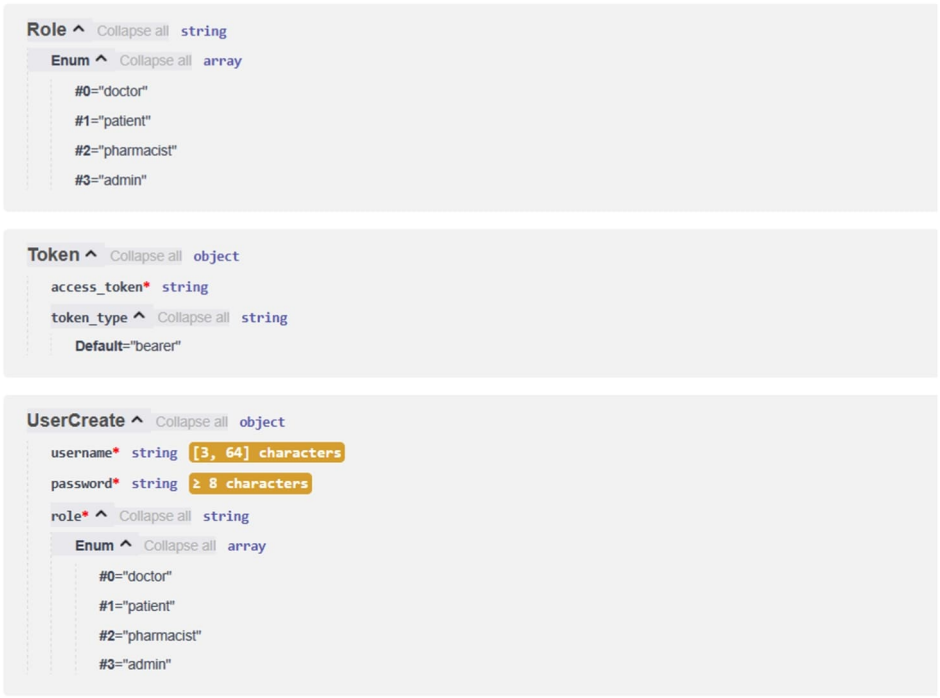

Figure 1.16

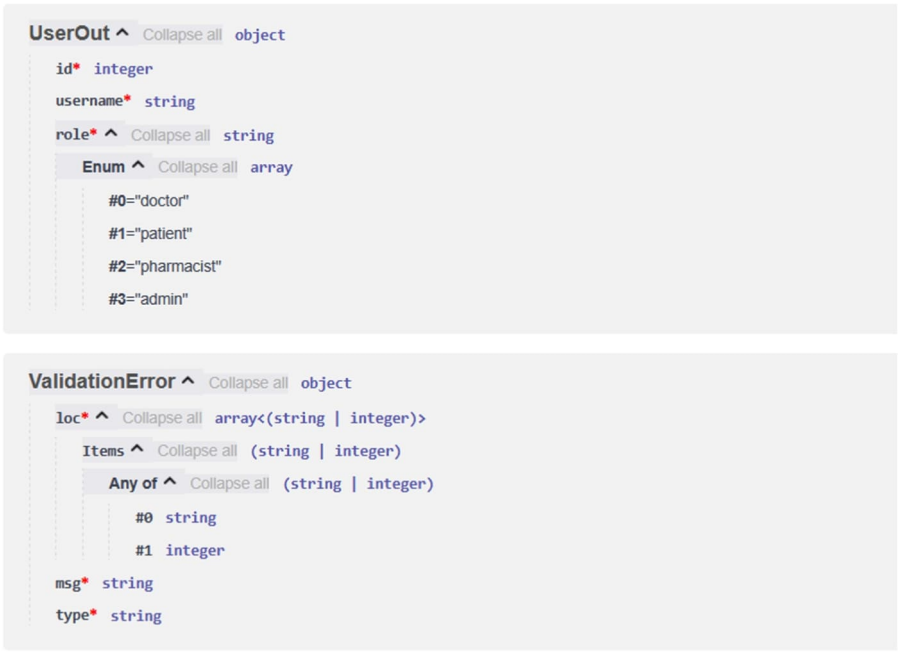

Figure 1.17

30% EXTRA CONTRIBUTION

1. Role-Based Access Control (RBAC)

- Unlike a generic API, this system introduces **multiple user roles** such as **patients, doctors, and administrators**.
- Each role has **specific permissions**:
    - Patients can view their health records and book appointments.
    - Doctors can access patient details, update diagnoses, and manage appointments.
    - Administrators can oversee the entire system and manage users.
- This ensures **fine-grained security** and prevents unauthorized access.

2. Appointment Scheduling System

- Beyond basic CRUD APIs, the project implements a **smart scheduling system**.
- Patients can book appointments with doctors, while doctors can accept, decline, or reschedule.
- The system prevents **double bookings** and ensures that appointments are managed in real-time.
- This feature adds practical value for **hospital or clinic management systems**.

3. Electronic Health Records (EHR) Management

- The API supports storing and managing **medical records digitally**.
- Patients can securely access their reports, prescriptions, and treatment history.
- Doctors can update patient health records dynamically.
- Sensitive data is **encrypted** to comply with privacy standards like **HIPAA**.

4. Audit Logging & Monitoring

- Every API call is tracked with **audit logs** (who accessed what, and when).
- This ensures **transparency and accountability**, which are essential in healthcare systems.
- Logs can also help in detecting unusual activities or security breaches.

5. Scalability & Production-Readiness

- The API is not just functional but also designed to **scale in real-world usage**.
- Features like **rate limiting**, **error handling**, and **input validation** ensure stability.
- Containerization (e.g., Docker) and deployment strategies are used for **production-grade readiness**.

6. Compliance with Healthcare Security Standards

- Unlike a generic secure API, this project emphasizes **healthcare compliance**.
- Data encryption, secure authentication (JWT/OAuth2), and restricted access align with **HIPAA/GDPR standards**.
- This makes the project realistic for **sensitive environments** like hospitals or insurance systems.