**Vehicle Rental System**

**1. Introduction**

The Vehicle Rental System is a digital platform designed to simplify the process of renting vehicles such as cars, bikes, and vans. It enables customers to search, book, and pay for rentals seamlessly while providing administrators with tools to manage vehicles, users, reservations, and payments efficiently.

**2. Objectives**

 To provide an easy-to-use platform for customers to rent vehicles online.

 To reduce manual paperwork and streamline rental operations.

 To ensure secure user authentication and safe online payments.

 To provide admins with real-time vehicle availability, booking management, and reports.

**3. Key Features**

**(a)** **Customer Features**

 User Registration & Login (with email/phone verification).

 Search & Browse Vehicles (by location, type, price, availability).

 Online Booking & Payment (secure gateway, wallet, UPI, cards).

 Booking History & Invoices.

 Cancellation & Refund Management.

 Notifications & Alerts (SMS/Email reminders).

**(b)** **Admin Features**

 Dashboard to view active bookings and revenues.

 Vehicle Management (add, update, remove vehicles).

 Customer Management (view user details, feedback).

 Booking Management (approve, cancel, reschedule)

payment & Refund Tracking.

 Reports & Analytics.

**4. System Architecture**

 Frontend: React.js / Angular (for user interface).

 Backend: FastAPI / Node.js (for business logic & APIs).

 Database: MongoDB / MySQL (for storing users, vehicles, bookings).

 Authentication: JWT-based secure login.

 Hosting & Deployment: Docker, AWS/Azure/GCP.

**5. Workflow**

 a. Customer registers or logs in.

 b. Customer searches available vehicles by date & location.

 c. Customer books a vehicle and makes payment.

 d. Admin gets booking details and confirms availability.

 e. Customer receives confirmation via app/email.

 f. After usage, the customer returns the vehicle, and billing is finalized.

 g. System updates availability for the next user.

**6. Benefits**

 For Customers: Quick booking, multiple payment options, transparency.

 For Admins: Reduced manual work, real-time insights, improved efficiency.

 For Business: Scalability, digital transformation, higher customer satisfaction.

**7. Future Enhancements**

 AI-based pricing suggestions.

 GPS tracking of rented vehicles.

 Integration with insurance services.

 Mobile App (Android/iOS) with push notifications

**8.System Implementation-Code Examples**

**8.1 Register**

@router.post("/register", response\_model=UserOut, status\_code=status.HTTP\_201\_CREATED)

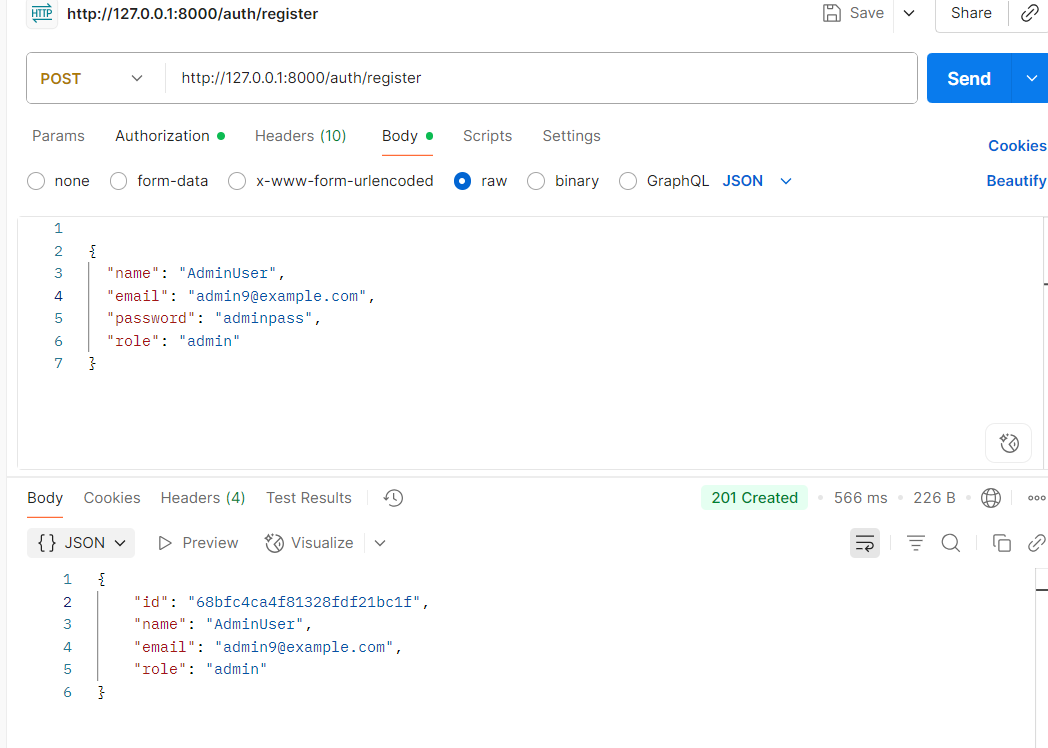
async def register(user: UserCreate):

    existing = await db.users.find\_one({"email": user.email})

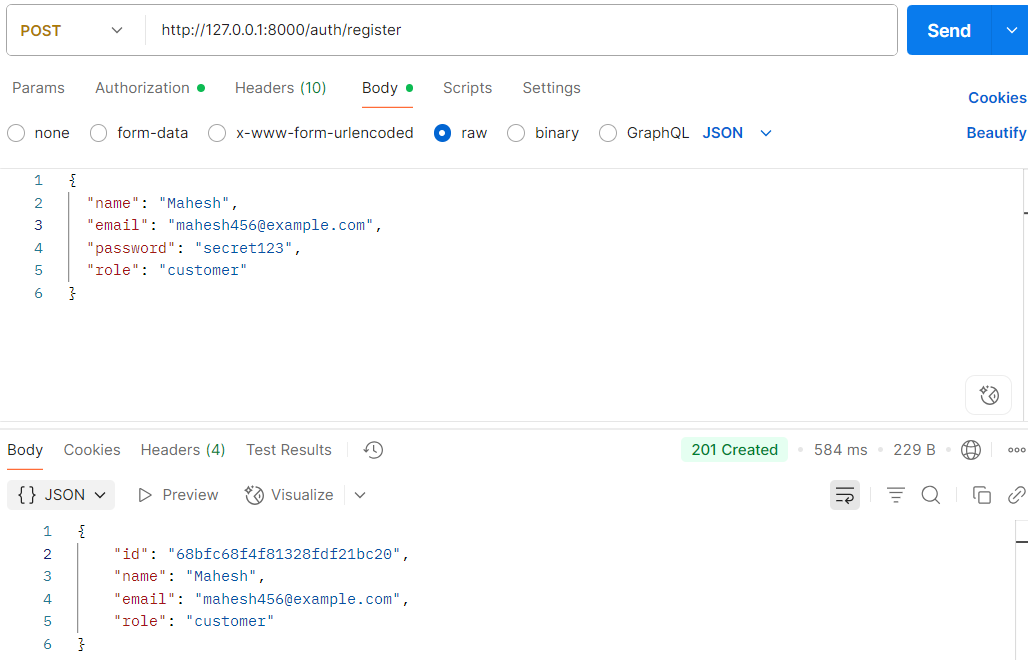
    if existing:

        raise HTTPException(status\_code=400, detail="Email already registered")

**8.1(a) Admin Register**

****

**8.2(b) User Register**

****

**8.2 Login**

@router.post("/login", response\_model=Token)

async def login(form\_data: OAuth2PasswordRequestForm = Depends()):

    user\_doc = await db.users.find\_one({"email": form\_data.username})

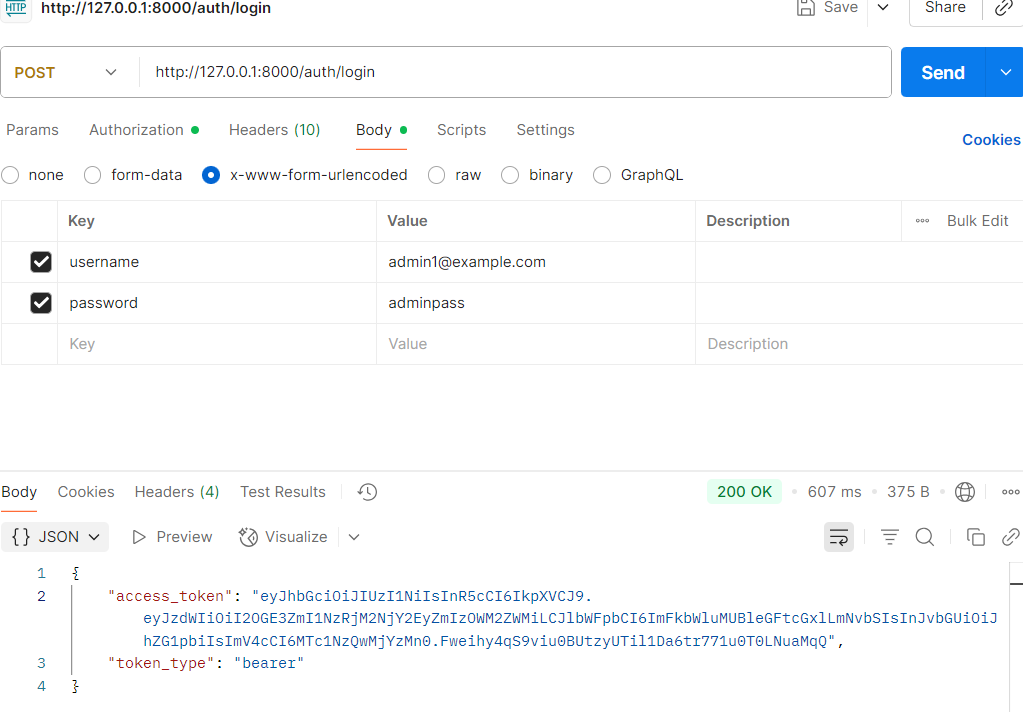
    if not user\_doc:

        raise HTTPException(status\_code=400, detail="Incorrect email or password")

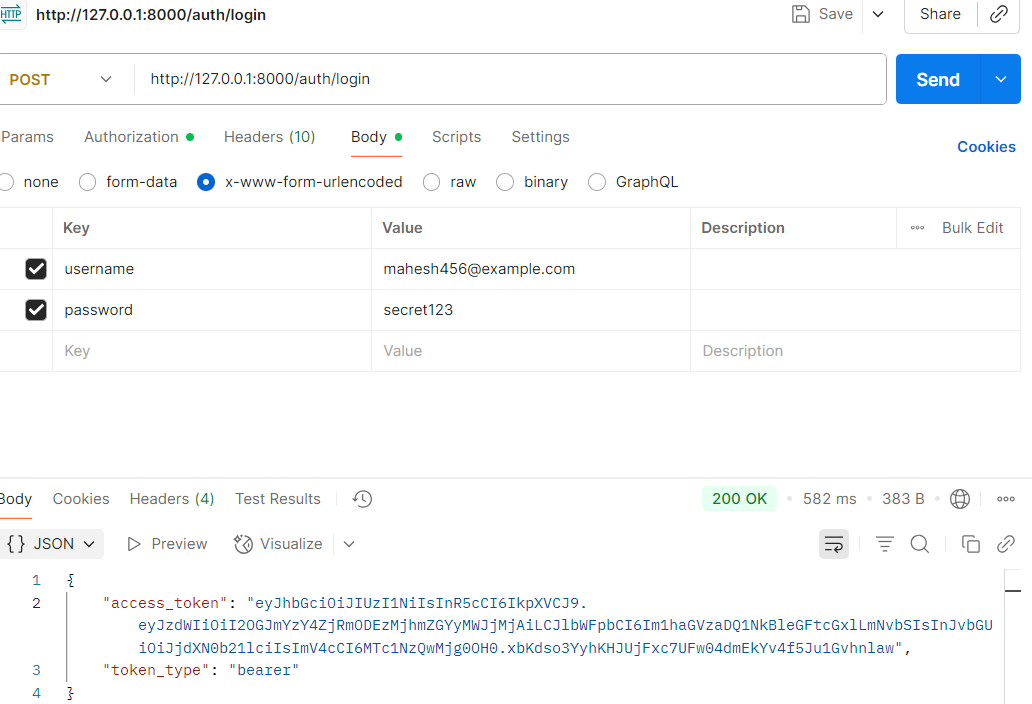
    if not verify\_password(form\_data.password, user\_doc["hashed\_password"]):

        raise HTTPException(status\_code=400, detail="Incorrect email or password")

**8.2(a)Admin login**

****

**8.2(b)User Login**

****

**8.3 Vehicles Upload**

@router.post("/", response\_model=VehicleOut, status\_code=status.HTTP\_201\_CREATED)

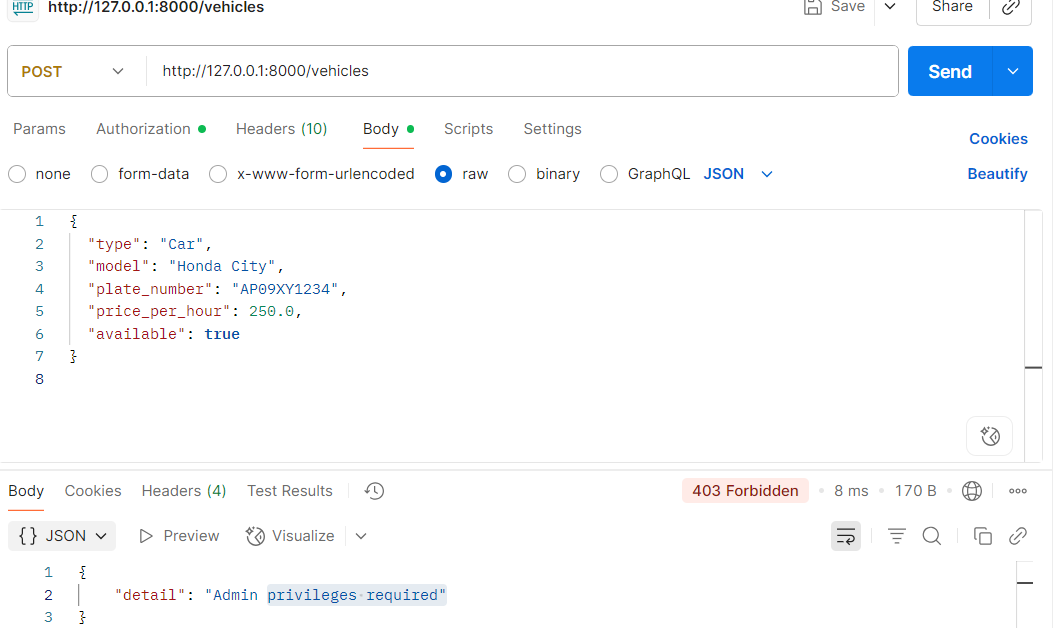
async def add\_vehicle(vehicle: VehicleCreate, admin=Depends(require\_admin)):

    doc = vehicle.dict()

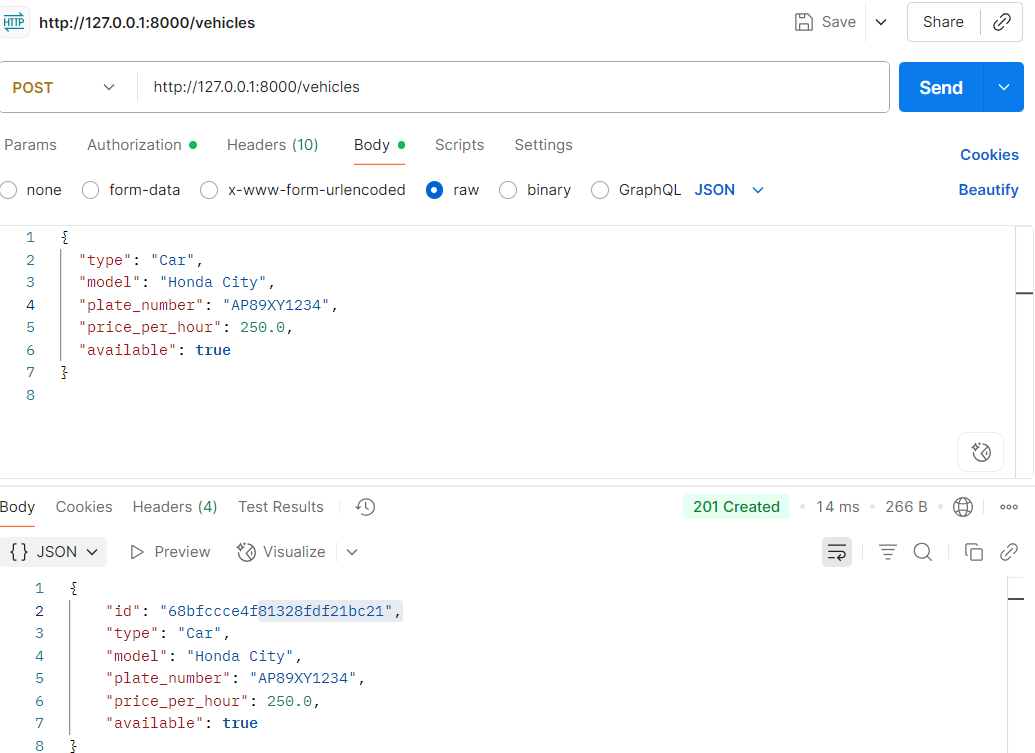
    res = await db.vehicles.insert\_one(doc)

    created = await db.vehicles.find\_one({"\_id": res.inserted\_id})

If you try to upload vehicles with login of user it shows admin privileges required

****

Login with admin you can upload a vehicle

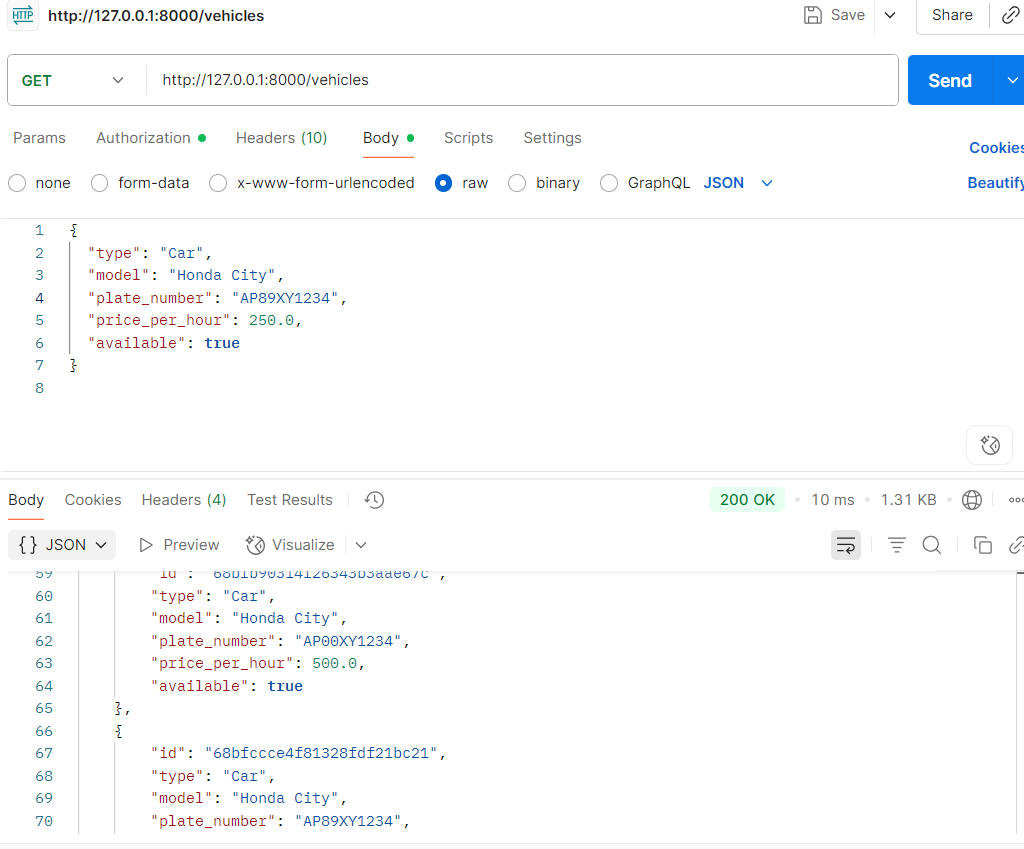


**8.4 Available Vehicles**

@router.get("/", response\_model=List[VehicleOut])

async def list\_vehicles(current\_user=Depends(get\_current\_user)):

    vehicles = await db.vehicles.find({"available": True}).to\_list(100)

****

**8.5 Booking Vehicle**

@router.post("/", response\_model=BookingOut, status\_code=status.HTTP\_201\_CREATED)

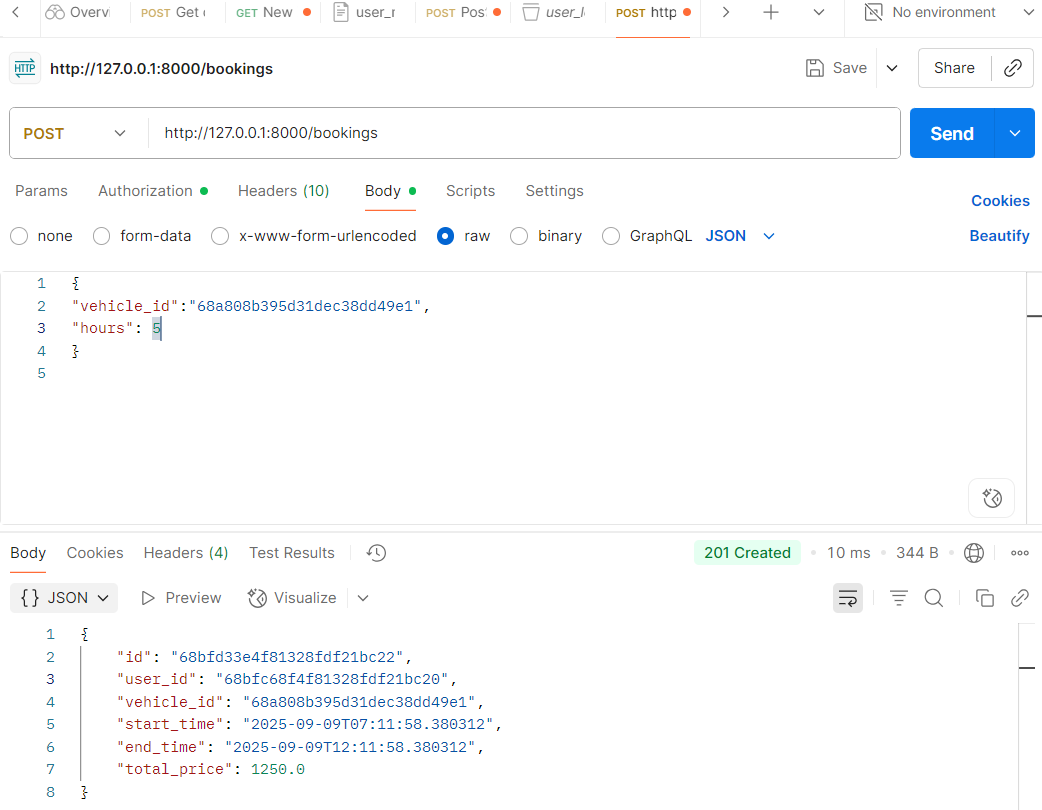
async def book\_vehicle(booking: BookingCreate, current\_user=Depends(get\_current\_user)):

    try:

        vehicle\_obj\_id = ObjectId(booking.vehicle\_id)

    except InvalidId:

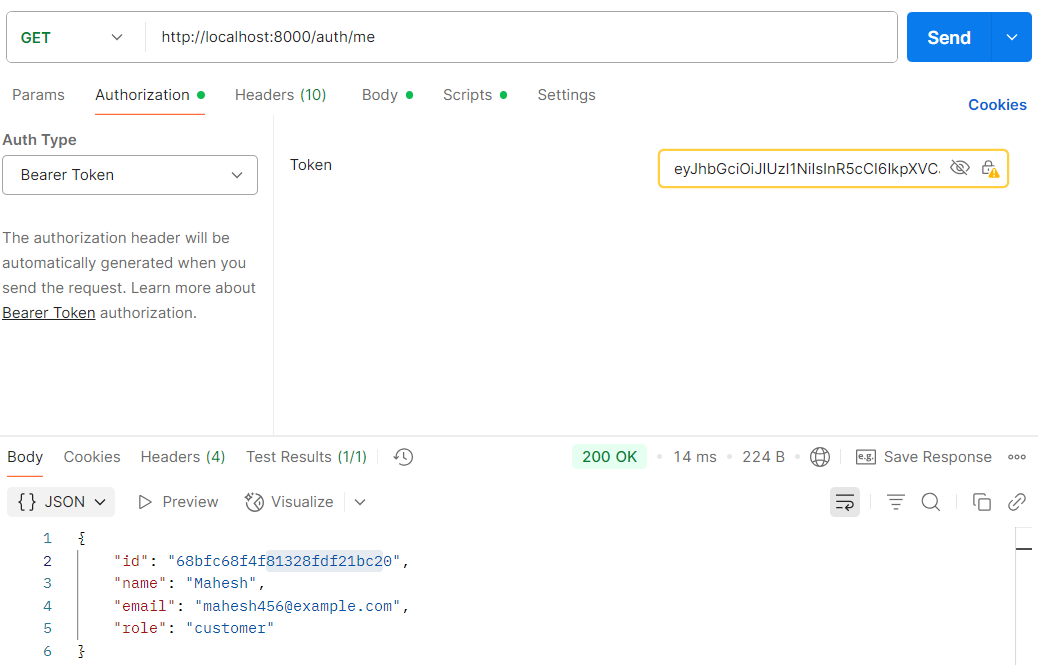
        raise HTTPException(status\_code=400, detail="Invalid vehicle id")



**8.6 Get Current User**

@router.get("/me")

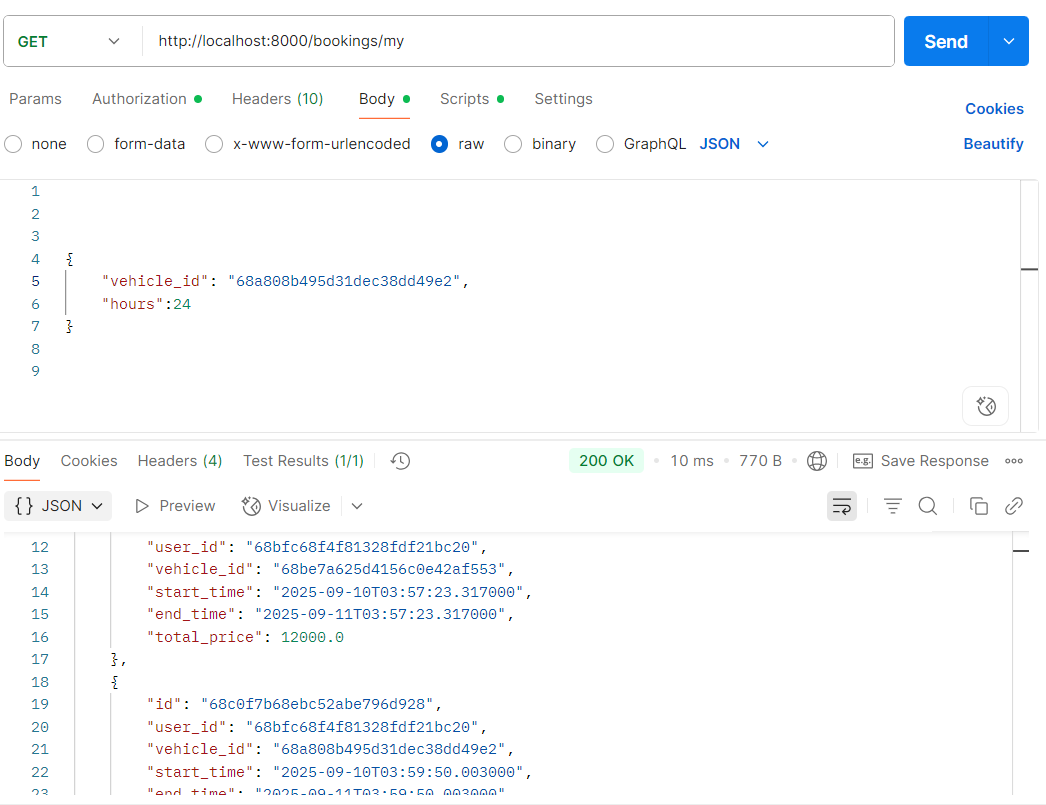
async def get\_me(current\_user=Depends(get\_current\_user)):



**8.7 All Bookings of Current User**

@router.get("/all", response\_model=List[BookingWithVehicleOut])

async def all\_bookings(admin=Depends(require\_admin)):

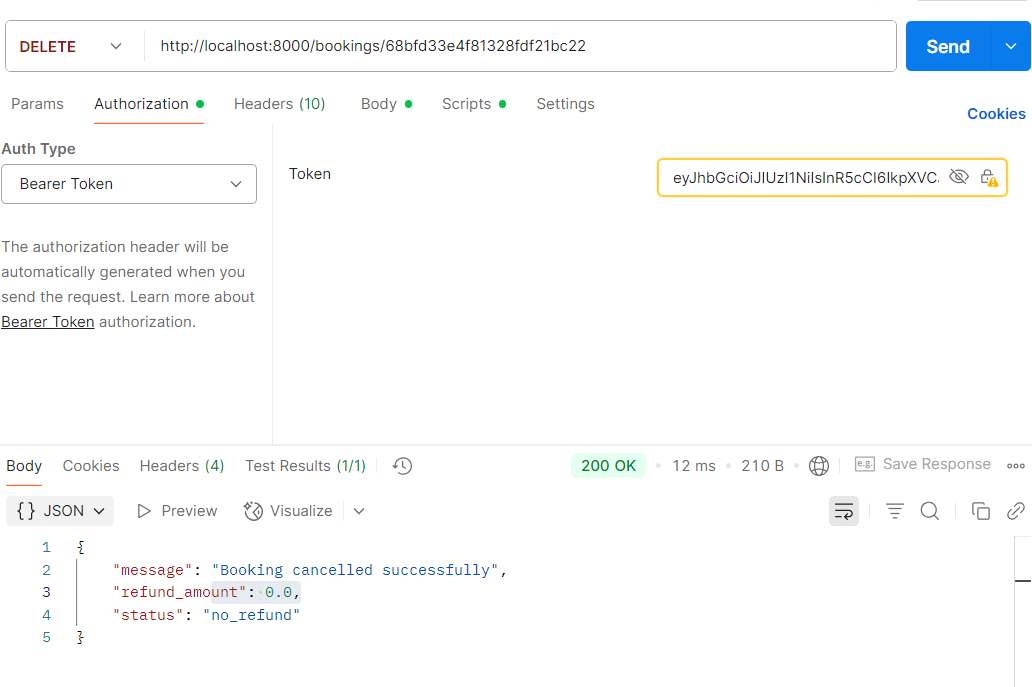


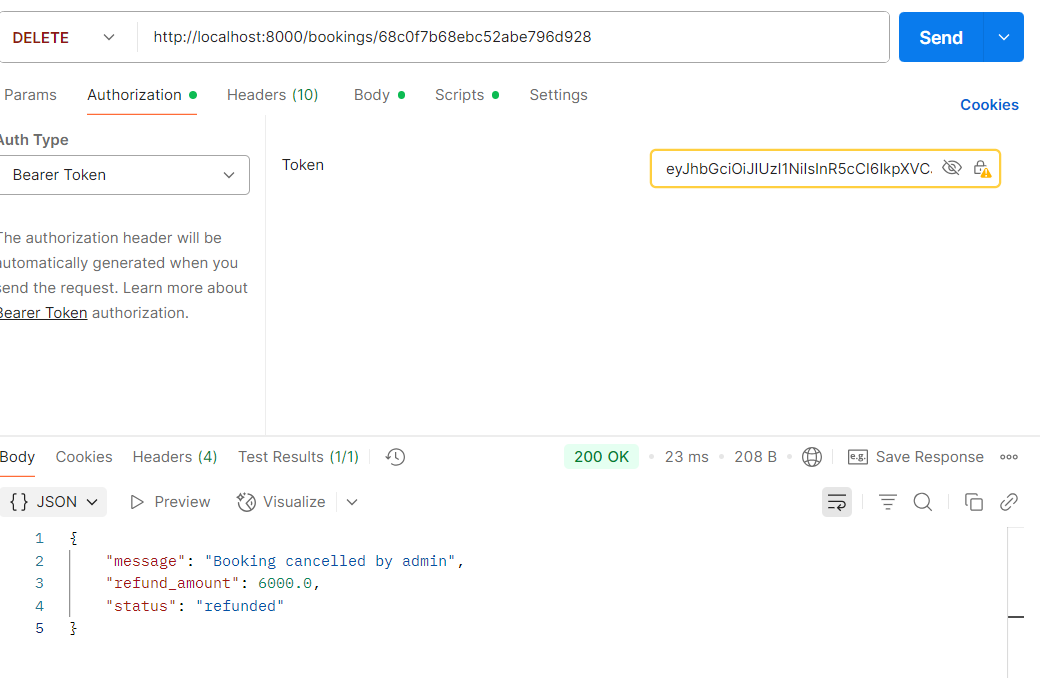
**8.8 Cancel & Refund**

@router.delete("/{booking\_id}")

async def cancel\_booking(booking\_id: str, current\_user: dict = Depends(get\_current\_user)):

    booking = await db.bookings.find\_one({"\_id": ObjectId(booking\_id)})

**8.8(a) User Cancellation**

**8.8(b) Admin Cancellation**