AI-ASSISTED CODING LAB TEST-03

NAME: BANDLA SAI NANDAN REDDY

HALL TICKET: 2403A510C5

BATCH : 05

SET E3

**Q1:**  
Scenario: In the Logistics sector, a company faces a challenge related to backend api development.  
Task: Use AI-assisted tools to solve a problem involving backend api development in this context.  
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

**PROMPT:**

Develop a backend API for a logistics company to manage shipment data, including endpoints for creating, reading, updating, and deleting records. Provide the source code, explanation of AI assistance used, and sample output.

**CODE:**

# app.py

from fastapi import FastAPI, HTTPException, Path, Query, status

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel, Field

from typing import Optional, List

from uuid import uuid4

from datetime import datetime

import sqlite3

import os

DB\_PATH = "shipments.db"

# ------------------ Database Setup ------------------

def init\_db():

    first = not os.path.exists(DB\_PATH)

    conn = sqlite3.connect(DB\_PATH, check\_same\_thread=False)

    if first:

        conn.execute("""

        CREATE TABLE shipments (

            id TEXT PRIMARY KEY,

            origin TEXT NOT NULL,

            destination TEXT NOT NULL,

            status TEXT NOT NULL,

            weight REAL,

            created\_at TEXT NOT NULL,

            updated\_at TEXT NOT NULL

        )

        """)

        conn.execute("""

        CREATE TABLE updates (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            shipment\_id TEXT NOT NULL,

            status TEXT NOT NULL,

            note TEXT,

            timestamp TEXT NOT NULL

        )

        """)

        conn.commit()

    return conn

conn = init\_db()

# ------------------ Pydantic models ------------------

class ShipmentCreate(BaseModel):

    origin: str = Field(..., example="Mumbai, IN")

    destination: str = Field(..., example="Bangalore, IN")

    weight: Optional[float] = Field(None, ge=0, example=12.5)

class Shipment(BaseModel):

    id: str

    origin: str

    destination: str

    status: str

    weight: Optional[float]

    created\_at: datetime

    updated\_at: datetime

class StatusUpdate(BaseModel):

    status: str = Field(..., example="IN\_TRANSIT")

    note: Optional[str] = Field(None, example="Left warehouse")

class UpdateItem(BaseModel):

    id: int

    shipment\_id: str

    status: str

    note: Optional[str]

    timestamp: datetime

# ------------------ App setup ------------------

app = FastAPI(title="Logistics Shipments API", version="1.0")

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=["\*"],

    allow\_credentials=True,

    allow\_methods=["\*"],

    allow\_headers=["\*"],

)

# ------------------ Utility ------------------

def row\_to\_shipment(row):

    return Shipment(

        id=row[0],

        origin=row[1],

        destination=row[2],

        status=row[3],

        weight=row[4],

        created\_at=datetime.fromisoformat(row[5]),

        updated\_at=datetime.fromisoformat(row[6]),

    )

# ------------------ Endpoints ------------------

@app.post("/shipments", response\_model=Shipment, status\_code=status.HTTP\_201\_CREATED)

def create\_shipment(payload: ShipmentCreate):

    shipment\_id = str(uuid4())

    now = datetime.utcnow().isoformat()

    status0 = "CREATED"

    conn.execute(

        "INSERT INTO shipments (id, origin, destination, status, weight, created\_at, updated\_at) VALUES (?, ?, ?, ?, ?, ?, ?)",

        (shipment\_id, payload.origin, payload.destination, status0, payload.weight, now, now),

    )

    conn.commit()

    row = conn.execute("SELECT \* FROM shipments WHERE id = ?", (shipment\_id,)).fetchone()

    return row\_to\_shipment(row)

@app.get("/shipments/{shipment\_id}", response\_model=Shipment)

def get\_shipment(shipment\_id: str = Path(..., description="Shipment UUID")):

    row = conn.execute("SELECT \* FROM shipments WHERE id = ?", (shipment\_id,)).fetchone()

    if not row:

        raise HTTPException(status\_code=404, detail="Shipment not found")

    return row\_to\_shipment(row)

@app.post("/shipments/{shipment\_id}/updates", response\_model=UpdateItem, status\_code=status.HTTP\_201\_CREATED)

def add\_update(shipment\_id: str, payload: StatusUpdate):

    row = conn.execute("SELECT id FROM shipments WHERE id = ?", (shipment\_id,)).fetchone()

    if not row:

        raise HTTPException(status\_code=404, detail="Shipment not found")

    now = datetime.utcnow().isoformat()

    cur = conn.execute(

        "INSERT INTO updates (shipment\_id, status, note, timestamp) VALUES (?, ?, ?, ?)",

        (shipment\_id, payload.status, payload.note, now),

    )

    conn.execute(

        "UPDATE shipments SET status = ?, updated\_at = ? WHERE id = ?",

        (payload.status, now, shipment\_id),

    )

    conn.commit()

    update\_id = cur.lastrowid

    upd\_row = conn.execute("SELECT id, shipment\_id, status, note, timestamp FROM updates WHERE id = ?", (update\_id,)).fetchone()

    return UpdateItem(id=upd\_row[0], shipment\_id=upd\_row[1], status=upd\_row[2], note=upd\_row[3], timestamp=datetime.fromisoformat(upd\_row[4]))

@app.get("/shipments", response\_model=List[Shipment])

def list\_shipments(status: Optional[str] = Query(None), limit: int = Query(50, ge=1, le=200)):

    if status:

        rows = conn.execute("SELECT \* FROM shipments WHERE status = ? ORDER BY updated\_at DESC LIMIT ?", (status, limit)).fetchall()

    else:

        rows = conn.execute("SELECT \* FROM shipments ORDER BY updated\_at DESC LIMIT ?", (limit,)).fetchall()

    return [row\_to\_shipment(r) for r in rows]

@app.get("/shipments/{shipment\_id}/updates", response\_model=List[UpdateItem])

def get\_updates(shipment\_id: str):

    rows = conn.execute("SELECT id, shipment\_id, status, note, timestamp FROM updates WHERE shipment\_id = ? ORDER BY timestamp DESC", (shipment\_id,)).fetchall()

    return [UpdateItem(id=r[0], shipment\_id=r[1], status=r[2], note=r[3], timestamp=datetime.fromisoformat(r[4])) for r in rows]

@app.get("/health")

def health():

    return {"status": "ok", "time": datetime.utcnow().isoformat()}

# Run instructions when executed directly

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("app:app", host="0.0.0.0", port=8000, reload=True)

**OUTPUT :**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Observation :**

The AI tool helped in generating the backend API code quickly. It suggested using FastAPI for development, created the structure of the API (routes, models, and error handling), and provided code examples for CRUD operations. It also explained how to install required packages and test the API using Swagger UI. This saved time and reduced coding errors.

**Q2:**

Scenario: In the Entertainment sector, a company faces a challenge related to algorithms with ai assistance.  
Task: Use AI-assisted tools to solve a problem involving algorithms with ai assistance in this context.  
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

**PROMPT:**

Create a simple Python algorithm using AI-assisted tools for the entertainment industry that recommends movies based on user preferences using cosine similarity.

**CODE:**

from sklearn.metrics.pairwise import cosine\_similarity

from sklearn.feature\_extraction.text import CountVectorizer

# Sample entertainment data

movies = [

    "Action adventure superhero film",

    "Romantic comedy drama",

    "Science fiction adventure space",

    "Horror thriller mystery",

    "Comedy family entertainment"

]

# Create a Count Vectorizer to convert text into numerical vectors

vectorizer = CountVectorizer()

movie\_vectors = vectorizer.fit\_transform(movies)

# User's interest

user\_input = ["I like action and superhero movies"]

# Convert user input to vector

user\_vector = vectorizer.transform(user\_input)

# Compute similarity

similarity\_scores = cosine\_similarity(user\_vector, movie\_vectors)

# Get the index of the best match

recommended\_index = similarity\_scores.argmax()

print("User Input:", user\_input[0])

print("Recommended Movie:", movies[recommended\_index])

OUTPUT:



OBSERVATION:

AI-assisted in designing the recommendation algorithm. The AI suggested using cosine similarity and Count Vectorizer from scikit-learn to measure similarity between user interests and available movies.