Running the Application

Run the application using Uvicorn:
bash
Copy code

```
uvicorn main:app --reload
```

1.
2. **Access the API documentation:**
   Open your browser and navigate to `http://127.0.0.1:8000/docs` to access the interactive API documentation provided by Swagger UI.

API Endpoints

Authentication

- **POST** `/signup`: Create a new user.
- **POST** `/login`: Authenticate a user and receive a JWT token.

Transactions

- **POST** `/transactions`: Create a new transaction.
- **GET** `/transactions`: Retrieve a list of transactions.
- **GET** `/transactions/{transaction_id}`: Retrieve a specific transaction.
- **PUT** `/transactions/{transaction_id}`: Update a transaction.
- **DELETE** `/transactions/{transaction_id}`: Delete a transaction.
- **POST** `/transactions/batch`: Batch create transactions.

Testing the API

Sign Up

bash
Copy code

```
curl -X POST "http://127.0.0.1:8000/signup" \
     -H "Content-Type: application/json" \
     -d '{
           "username": "testuser",
           "password": "testpass"
         }'
```

Log In

bash
Copy code
```
curl -X POST "http://127.0.0.1:8000/login" \
     -H "Content-Type: application/json" \
     -d '{
           "username": "testuser",
           "password": "testpass"
         }'
```

Response:

json
Copy code
```
{
  "access_token": "jwt-token-string",
  "token_type": "bearer"
}
```

Create a Transaction

Use the JWT token received from the login response.

bash
Copy code
```
curl -X POST "http://127.0.0.1:8000/transactions" \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer jwt-token-string" \
     -d '{
           "amount": 1500.00,
           "description": "Monthly product sale",
           "date": "2023-10-01"
         }'
```

Response:

json
Copy code
```
{
  "id": 1,
  "amount": 1500.0,
  "description": "Monthly product sale",
  "date": "2023-10-01",
  "category": "Product Sales",
  "user_id": 1
}
```

Explanation of the Code

`main.py`

- **Imports:**
  - `FastAPI`, `Depends`, `HTTPException`, `status` from FastAPI.
  - `Session` from SQLAlchemy ORM.
  - `List` from typing module.
- **Modules:**
  - `database.py`: Contains the database connection and session management.
  - `models.py`: Defines the database models using SQLAlchemy ORM.
  - `schemas.py`: Defines the Pydantic models for request and response schemas.
  - `auth.py`: Handles authentication, including token creation and verification.
  - `utils.py`: Utility functions like `get_db` dependency.
  - `ml_model.py`: Contains the categorization logic, possibly using a machine learning model.

Database Integration

`database.py`:
python
Copy code
```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "sqlite:///./transactions.db"

engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread":
False}
)
```

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
Base = declarative_base()
```

-

`models.py`:
python
Copy code

```
from sqlalchemy import Column, Integer, String, Float, ForeignKey
from sqlalchemy.orm import relationship
from database import Base

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, index=True)
    hashed_password = Column(String)
    transactions = relationship("Transaction",
back_populates="owner")

class Transaction(Base):
    __tablename__ = "transactions"
    id = Column(Integer, primary_key=True, index=True)
    amount = Column(Float)
    description = Column(String)
    date = Column(String)
    category = Column(String)
    user_id = Column(Integer, ForeignKey("users.id"))
    owner = relationship("User", back_populates="transactions")
```

-

Authentication

`auth.py`:
python
Copy code

```
from fastapi import Depends, HTTPException, status
from jose import JWTError, jwt
from passlib.context import CryptContext
from datetime import datetime, timedelta
from sqlalchemy.orm import Session
from models import User
from utils import get_db

SECRET_KEY = "your-secret-key"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password):
    return pwd_context.hash(password)

def authenticate_user(db, username: str, password: str):
    user = db.query(User).filter(User.username == username).first()
    if not user:
        return False
    if not verify_password(password, user.hashed_password):
        return False
    return user

def create_access_token(data: dict, expires_delta: timedelta =
None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY,
algorithm=ALGORITHM)
    return encoded_jwt

async def get_current_user(token: str = Depends(oauth2_scheme), db:
Session = Depends(get_db)):
    # Token validation logic here
    pass
```

-

Machine Learning Model Integration

`ml_model.py`:
python
Copy code
```
def categorize_revenue(transaction):
    # Enhanced categorization logic using a machine learning model
    description = transaction.description.lower()
    # For simplicity, we'll use the same keyword-based
categorization
    if "sale" in description or "product" in description:
        return "Product Sales"
    elif "service" in description or "consulting" in description:
        return "Service Income"
    elif "interest" in description:
        return "Interest Income"
    elif "rental" in description:
        return "Rental Income"
    else:
        return "Other Income"
```

- **Note:** In a real-world scenario, you'd train a machine learning model using historical transaction data.

Conclusion

This extended Python application provides a comprehensive API for categorizing business revenue, suitable for integration into data pipelines for new businesses. It includes essential features like user authentication, database persistence, CRUD operations, batch processing, and an extendable categorization system.