PYAUTOMATE AI

1.BACKEND

APP

DATA BASE

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "postgresql://postgres:postgres@db:5432/pyautomate"

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

MODEL

```
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import declarative_base
import datetime

Base = declarative_base()

class AutomationWorkflow(Base):
    __tablename__ = "automation_workflows"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, nullable=False)
    created_at = Column(DateTime, default=datetime.datetime.utcnow)
```

ROUTES

AUTOMATION

```
from fastapi import APIRouter, HTTPException
from pydantic import BaseModel
from app.services import rpa_engine, ai_model

router = APIRouter()

class InputText(BaseModel):
```

```
    text: str


@router.get("/run-rpa")
async def trigger_rpa(url: str):
    """Trigger a headless Selenium session that retrieves a page title"""
    try:
        title = rpa_engine.run_web_automation(url)
        return {"status": "success", "page_title": title}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))


@router.post("/predict")
async def run_prediction(payload: InputText):
    """Run a quick ML prediction on input text"""
    try:
        prediction = ai_model.predict_label(payload.text)
        return {"prediction": prediction}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

SERVICES

AI_Model

```
import joblib
import os
from pathlib import Path

MODEL_PATH = Path(__file__).resolve().parent.parent.parent / "ml_models" / "classifier.pkl"

if MODEL_PATH.exists():
    model = joblib.load(MODEL_PATH)
else:
    # simple fallback model
    from sklearn.linear_model import LogisticRegression
```

```
    model = LogisticRegression()
    model.classes_ = ["negative", "positive"]
    model.coef_ = [[0]]
    model.intercept_ = [0]

def predict_label(text: str):
    """Return a dummy prediction if no model exists yet"""
    if hasattr(model, "predict"):
        return model.predict([text])[0]
    return "unknown"
```

RPA_ENGINE

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

def run_web_automation(url: str) -> str:
    """Visit the given URL in a headless browser and return the page title"""
    options = Options()
    options.add_argument("--headless=new")
    driver = webdriver.Chrome(options=options)
    try:
        driver.get(url)
        return driver.title
    finally:
        driver.quit()
```

UTIIS

AUTH

```
from datetime import datetime, timedelta
from typing import Optional
from jose import JWTError, jwt

SECRET_KEY = "change_this_secret"
ALGORITHM = "HS256"
```

```
ACCESS_TOKEN_EXPIRE_MINUTES = 30


def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
    to_encode = data.copy()
    expire = datetime.utcnow() + (expires_delta or
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

OCR_UTILS

```
import pytesseract
from PIL import Image


def extract_text(image_path: str) -> str:
    """Extract text from an image file using Tesseract OCR"""
    img = Image.open(image_path)
    return pytesseract.image_to_string(img)
```

Main

```
from fastapi import FastAPI
from app.routes import automation

app = FastAPI(title="PyAutomate AI")

# include API router
app.include_router(automation.router, prefix="/api/automation", tags=["automation"])

@app.get("/", tags=["root"])
async def read_root():
    return {"message": "Welcome to PyAutomate AI Backend"}
```

REQUIREMENTS

```
fastapi
uvicorn[standard]
selenium
joblib
```

scikit-learn

python-multipart

pydantic

SQLAlchemy

asyncpg

psycopg2-binary

python-jose

pillow

pytesseract

2.CONFIG

Celery config:

```
from celery import Celery
celery_app = Celery(
    "tasks",
    broker="pyamqp://guest@rabbitmq//",
    backend="rpc://"
)
celery_app.conf.task_routes = {
    "tasks.run_automation": {"queue": "automation"},
}
```

3.SETTINGS

```
mport os
from pydantic import BaseSettings
class Settings(BaseSettings):
    app_name: str = "PyAutomate AI"
 database_url: str = os.getenv("DATABASE_URL",
"postgresql://postgres:postgres@localhost/pyautomate")
settings = Settings()
```

4.FRONTEND:

ANALYTICS

```
import plotly.express as px
import pandas as pd

def basic_bar_chart(df: pd.DataFrame, x: str, y: str):
    fig = px.bar(df, x=x, y=y)
    return fig
```

5.DASHBOARD

```
import streamlit as st
import requests
import pandas as pd

st.set_page_config(page_title="PyAutomate AI Dashboard", layout="wide")

st.title("PyAutomate AI Dashboard")

api_url = st.text_input("Enter backend API base URL", "http://localhost:8000")

st.header("Run RPA")
target_url = st.text_input("URL to fetch title", "https://example.com")
if st.button("Run"):
    response = requests.get(f"{api_url}/api/automation/run-rpa", params={"url": target_url})
    if response.ok:
        st.success(f'Page title: {response.json()["page_title"]}')
    else:
        st.error(response.text)

st.header("Predict Text Sentiment")
text_input = st.text_area("Input text")
if st.button("Predict"):
    response = requests.post(f"{api_url}/api/automation/predict", json={"text": text_input})
    if response.ok:
        st.info(f'Prediction: {response.json()["prediction"]}')
```

```
    else:
        st.error(response.text)
```

6.ML_MODEL

--SCRIPTS--

SCRAPER

```
import requests
from bs4 import BeautifulSoup
def simple_scrape(url: str):
    resp = requests.get(url, timeout=10)
    soup = BeautifulSoup(resp.text, 'html.parser')
    return soup.title.string if soup.title else "No title"
```

7.TESTS

TEST_AUTOMATION :

```
rom fastapi.testclient import TestClient
from app.main import app


client = TestClient(app)


def test_root():
    response = client.get("/")
    assert response.status_code == 200
```

.env

```
DATABASE_URL=postgresql://postgres:postgres@db:5432/pyautomate
```

8.DOCKER_COMPOSE

```
version: "3.9"
services:
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: pyautomate
```

```
    volumes:
      - dbdata:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  backend:
    build: .
    command: uvicorn backend.app.main:app --host 0.0.0.0 --port 8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db

  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"

volumes:
  dbdata:
```

9.README

 # 🚀 PyAutomate AI

PyAutomate AI is an AI-powered automation framework that merges RPA, intelligent workflow orchestration, machine-learning-driven decisioning, and rich analytics dashboards.

Quick Start

```bash
```

```
# clone repository and move into directory
pip install -r backend/requirements.txt
uvicorn backend.app.main:app --reload
# in another terminal
streamlit run frontend/dashboard.py
```

For full documentation, open `docs/README.md`.