CX Dashboard — Starter Project

A compact, interview-friendly, end-to-end project scaffold: **FastAPI backend + MySQL storage + scikit-learn sentiment model + OpenAI summarizer + simple Chart.js frontend**. Everything below is intentionally kept small so you can learn and explain every piece in ~4 days.

Quick overview

- Upload small CSV of customer feedback (or POST single feedback).
- Store feedback in MySQL.
- Train a simple TF-IDF + LogisticRegression sentiment model (scikit-learn).
- Serve predictions and KPIs via FastAPI endpoints.
- Generate short, human-readable monthly summaries using the OpenAI API.
- Visualize on a plain `index.html` using Chart.js.

Folder structure (what you'll create)

```
cx-analytics-dashboard/
├── .env.example
├── requirements.txt
├── README.md
├── data/
│   └── sample_feedback.csv
├── train_model.py          # quick script to train model
├── app/
│   ├── main.py            # FastAPI app (routes + start)
│   ├── database.py        # SQLAlchemy engine & session
│   ├── models.py          # SQLAlchemy models
│   ├── schemas.py         # Pydantic schemas
│   ├── crud.py            # DB helper functions
│   ├── ml/
│   │   ├── train.py       # train and save model
│   │   └── predict.py     # helper to load model & predict
│   └── static/
│       ├── index.html
│       ├── app.js
│       └── style.css
```

Files & starter code

> **Note:** paste each code block into the correct file. The implementation is intentionally minimal and synchronous so you can understand every line.

`requirements.txt`

```
fastapi
uvicorn[standard]
sqlalchemy
pydantic
mysql-connector-python
pandas
numpy
scikit-learn
joblib
python-dotenv
openai
python-multipart
chartjs-python # optional helper, not required for frontend
```

(You can remove `chartjs-python` — frontend uses CDN JS.)

`.env.example`

```
# copy to .env and edit
DATABASE_URL=mysql+mysqlconnector://root:password@localhost/feedback_db
OPENAI_API_KEY=your_openai_api_key_here
MODEL_PATH=./models/sentiment_model.joblib
```

`app/database.py`

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
import os
from dotenv import load_dotenv

load_dotenv()
DATABASE_URL = os.getenv("DATABASE_URL")

engine = create_engine(DATABASE_URL, echo=False, future=True)
```

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

# helper to get DB session in FastAPI
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

`app/models.py`

```
from sqlalchemy import Column, Integer, String, DateTime, Text
from sqlalchemy.sql import func
from .database import Base

class Feedback(Base):
    __tablename__ = 'feedback'
    id = Column(Integer, primary_key=True, index=True)
    customer_id = Column(String(64), index=True, nullable=True)
    channel = Column(String(50), nullable=True)
    text = Column(Text, nullable=False)
    sentiment = Column(String(20), nullable=True)  # positive/neutral/negative
    created_at = Column(DateTime(timezone=True), server_default=func.now())
```

`app/schemas.py`

```
from pydantic import BaseModel
from datetime import datetime
from typing import Optional

class FeedbackCreate(BaseModel):
    customer_id: Optional[str]
    channel: Optional[str]
    text: str

class FeedbackOut(FeedbackCreate):
    id: int
    sentiment: Optional[str]
    created_at: datetime
    class Config:
        orm_mode = True
```

---

app/crud.py

```
from sqlalchemy.orm import Session
from . import models, schemas

def create_feedback(db: Session, fb: schemas.FeedbackCreate, sentiment: str |
None = None):
    db_fb = models.Feedback(customer_id=fb.customer_id, channel=fb.channel,
text=fb.text, sentiment=sentiment)
    db.add(db_fb)
    db.commit()
    db.refresh(db_fb)
    return db_fb

def get_feedbacks(db: Session, limit: int = 100):
    return
db.query(models.Feedback).order_by(models.Feedback.created_at.desc()).limit(limit).all()

def get_sentiment_counts(db: Session):
    from sqlalchemy import func
    q = db.query(models.Feedback.sentiment,
func.count(models.Feedback.id)).group_by(models.Feedback.sentiment)
    return {row[0]: row[1] for row in q.all()}

def get_sentiment_trend(db: Session, days: int = 30):
    from sqlalchemy import func, cast, Date
    q = db.query(cast(models.Feedback.created_at, Date),
func.count(models.Feedback.id),
models.Feedback.sentiment).group_by(cast(models.Feedback.created_at, Date),
models.Feedback.sentiment).order_by(cast(models.Feedback.created_at, Date))
    # returns list of (date, count, sentiment)
    return q.all()
```

---

app/ml/train.py

```
# Train a TF-IDF + LogisticRegression pipeline and save it
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import joblib
import os

MODEL_PATH = os.getenv('MODEL_PATH', './models/sentiment_model.joblib')

def train(data_csv='data/sample_feedback.csv'):
    df = pd.read_csv(data_csv)
    # Expect a 'text' column and a 'label' column where label is: positive /
neutral / negative
    df = df.dropna(subset=['text', 'label'])

    X = df['text']
    y = df['label']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    pipe = Pipeline([
        ('tfidf', TfidfVectorizer(max_features=5000, ngram_range=(1,2))),
        ('clf', LogisticRegression(max_iter=1000))
    ])

    pipe.fit(X_train, y_train)
    preds = pipe.predict(X_test)
    print(classification_report(y_test, preds))

    os.makedirs(os.path.dirname(MODEL_PATH), exist_ok=True)
    joblib.dump(pipe, MODEL_PATH)
    print('Model saved to', MODEL_PATH)

if __name__ == '__main__':
    train()
```

`app/ml/predict.py`

```
import joblib
import os

MODEL_PATH = os.getenv('MODEL_PATH', './models/sentiment_model.joblib')

def load_model():
    if not os.path.exists(MODEL_PATH):
        raise FileNotFoundError('Model not found — run train.py first')
    return joblib.load(MODEL_PATH)

_model = None

def predict(texts):
    global _model
    if _model is None:
        _model = load_model()
    # texts: list[str]
    labels = _model.predict(texts)
    return labels
```

`app/main.py` (FastAPI app)

```
from fastapi import FastAPI, Depends, UploadFile, File, HTTPException
from fastapi.responses import HTMLResponse
from sqlalchemy.orm import Session
from . import database, crud, schemas
import csv, io, os
from dotenv import load_dotenv
import openai

load_dotenv()
openai.api_key = os.getenv('OPENAI_API_KEY')

app = FastAPI(title='CX Analytics (student project)')

# create DB tables if not exist
from .database import engine
from . import models
models.Base.metadata.create_all(bind=engine)

@app.post('/ingest/csv')
async def ingest_csv(file: UploadFile = File(...), db: Session =
Depends(database.get_db)):
    content = await file.read()
    s = content.decode('utf-8')
    reader = csv.DictReader(io.StringIO(s))
    count = 0
    from .ml.predict import predict
    texts = []
    rows = []
    for r in reader:
        # expected fields: customer_id, channel, text
        rows.append(r)
        texts.append(r.get('text', ''))

    # run model predictions in batch
    labels = predict(texts)
    for r, lab in zip(rows, labels):
        fb = schemas.FeedbackCreate(customer_id=r.get('customer_id'),
channel=r.get('channel'), text=r.get('text'))
        crud.create_feedback(db, fb, sentiment=str(lab))
        count += 1
    return {'ingested': count}

@app.post('/feedback', response_model=schemas.FeedbackOut)
def create_feedback(fb: schemas.FeedbackCreate, db: Session =
Depends(database.get_db)):
    # predict sentiment
    from .ml.predict import predict
    label = predict([fb.text])[0]
    db_obj = crud.create_feedback(db, fb, sentiment=str(label))
    return db_obj

@app.get('/kpi/sentiment_counts')
def sentiment_counts(db: Session = Depends(database.get_db)):
    return crud.get_sentiment_counts(db)

@app.get('/kpi/sentiment_trend')
def sentiment_trend(db: Session = Depends(database.get_db)):
    data = crud.get_sentiment_trend(db)
    # convert SQLAlchemy tuples to dicts for frontend
    return [{'date': str(row[0]), 'count': int(row[1]), 'sentiment': row[2]} for
row in data]

@app.post('/summary')
def summary(period_days: int = 30, db: Session = Depends(database.get_db)):
    # fetch last N days of feedback text (simple implementation)
    from datetime import datetime, timedelta
    cutoff = datetime.utcnow() - timedelta(days=period_days)
    q = db.query(models.Feedback).filter(models.Feedback.created_at >=
cutoff).order_by(models.Feedback.created_at.desc()).limit(200)
    texts = [f.text for f in q.all()]
    if not texts:
        return {'summary': 'No feedback in the selected period.'}

    # join texts (be careful with token limits in production; here kept small)
    joined = '\n'.join(texts[:50])
    prompt = f"Summarize the following customer feedback into 5 short bullet
points focusing on main issues and suggested actions:\n\n{joined}"
```

```
    resp = openai.ChatCompletion.create(
        model='gpt-3.5-turbo',
        messages=[
            {'role': 'system', 'content': 'You are a helpful summarizer that
writes concise bullet points.'},
            {'role': 'user', 'content': prompt}
        ],
        max_tokens=250,
        temperature=0.2
    )
    summary_text = resp['choices'][0]['message']['content']
    return {'summary': summary_text}

@app.get('/', response_class=HTMLResponse)
def home():
    with open('app/static/index.html', 'r', encoding='utf-8') as f:
        return f.read()
```

`train_model.py` **(root helper script)**

```
# simple runner for training from repository root
from app.ml.train import train

if __name__ == '__main__':
    train()
```

`app/static/index.html`

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>CX Dashboard</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@4.3.0/dist/
chart.umd.min.js"></script>
  <link rel="stylesheet" href="/app/static/style.css">
</head>
<body>
  <h1>CX Dashboard (Student Project)</h1>
  <div>
    <canvas id="pieChart" width="400" height="200"></canvas>
  </div>
  <div>
    <canvas id="lineChart" width="600" height="200"></canvas>
  </div>
  <h3>AI Summary (last 30 days)</h3>
  <pre id="summary">Loading...</pre>

  <script src="/app/static/app.js"></script>
</body>
</html>
```

`app/static/app.js`

```
async function fetchCounts(){
  const res = await fetch('/kpi/sentiment_counts');
  return await res.json();
}

async function fetchTrend(){
  const res = await fetch('/kpi/sentiment_trend');
  return await res.json();
}

async function fetchSummary(){
  const res = await fetch('/summary', {method: 'POST'});
  return await res.json();
}

function drawPie(counts){
  const ctx = document.getElementById('pieChart');
  const labels = Object.keys(counts);
  const data = Object.values(counts);
  new Chart(ctx, {
    type: 'pie',
    data: {labels: labels, datasets:[{data: data}]}
  });
}

function drawLine(trend){
  // trend: [{date, count, sentiment}, ...]
  const ctx = document.getElementById('lineChart');
  // convert to date->sentiment->count map
  const map = {};
  trend.forEach(r => {
    if(!map[r.date]) map[r.date] = {positive:0,neutral:0,negative:0};
    map[r.date][r.sentiment] = r.count;
  });
  const dates = Object.keys(map).sort();
  const pos = dates.map(d=>map[d].positive||0);
  const neu = dates.map(d=>map[d].neutral||0);
  const neg = dates.map(d=>map[d].negative||0);

  new Chart(ctx, {
    type: 'line',
    data: {
      labels: dates,
      datasets: [
        {label: 'Positive', data: pos, tension:0.3},
        {label: 'Neutral', data: neu, tension:0.3},
        {label: 'Negative', data: neg, tension:0.3}
      ]
    }
  });
}

(async ()=>{
  const counts = await fetchCounts();
  drawPie(counts);
  const trend = await fetchTrend();
  drawLine(trend);
  const sm = await fetchSummary();
  document.getElementById('summary').innerText = sm.summary || 'No summary.';
})();
```

`data/sample_feedback.csv`

```
text,label,customer_id,channel
"I waited on hold for 30 minutes, very unhappy.",negative,123,phone
"Great service, quick response.",positive,234,email
"The app crashes sometimes, please fix.",negative,345,app
"Average experience.",neutral,456,chat
"Rep was helpful and solved my issue.",positive,567,phone
```

How it all works — step by step (for interviews)

1. **Ingest data** — you upload a CSV (`/ingest/csv`) or call `/feedback` with a single feedback object. The FastAPI endpoint reads the text and stores it into MySQL via SQLAlchemy.

2. **Model training** — you run `python train_model.py`. That reads a labeled CSV (`data/sample_feedback.csv`), trains a `Pipeline(TfidfVectorizer -> LogisticRegression)` and saves it to `models/sentiment_model.joblib`.

3. **Inference** — FastAPI imports the `predict` helper from `app.ml.predict` which loads the saved model and returns labels for text. Each newly ingested feedback is saved with the predicted sentiment.

4. **KPIs** — API endpoints `/kpi/sentiment_counts` and `/kpi/sentiment_trend` compute aggregated counts and trends which the frontend fetches and renders with Chart.js.

5. **AI summary** — the `/summary` endpoint fetches recent feedback text, concatenates a small amount, and sends it to the OpenAI Chat Completion API to produce a short bullet list of top issues.

6. **Frontend** — a tiny `index.html` calls these endpoints and shows charts + summary. This is what you'll demo in an interview.

How to run locally (concise)

1. `pip install -r requirements.txt`
2. copy `.env.example` to `.env` and edit DB + API KEY
3. create MySQL DB (`CREATE DATABASE feedback_db;`)
4. run `python train_model.py` to create the model
5. start server: `uvicorn app.main:app --reload`
6. open `http://localhost:8000` in browser

Interview talking points (short bullets)

- *Problem*: Vodafone needs automated monitoring of customer feedback.
- *Solution*: Ingest raw feedback → classify sentiment → visualize trends → generate AI summaries.
- *Why this stack*: FastAPI is fast & simple; scikit-learn is reliable for small-to-medium text models; OpenAI summarizes without training; Chart.js makes attractive visuals quickly.
- *Improvements if asked*: add Docker, async ingestion, larger DL model (Keras), RAG for grounded answers.

Next steps I can do for you

1. Create these files and give you a downloadable zip.
2. Add a Docker Compose file to run MySQL + FastAPI.
3. Make the frontend prettier and add sample screenshots.

You can now open this document and copy the code into files.

If you want, I can also: generate the zip of files, or create a minimal Docker Compose + seed script next.

<!-- End of scaffold -->