# Banking Chatbot Project

## 📖 Case Study: AI-Powered Banking Chatbot

A leading bank wants to develop an AI-powered chatbot to assist customers with banking queries, such as account information, loan eligibility, transaction details, and security concerns. The chatbot should:  
✅ Answer common banking FAQs (without calling external APIs).  
✅ Understand and retrieve information from a knowledge base.  
✅ Work locally using an LLM (e.g., Ollama with a GPT model).  
✅ Store and retrieve vectorized FAQs using ChromaDB.  
✅ Expose an API endpoint via FastAPI.  
✅ Provide a user-friendly chat interface using Jinja2 + HTML/JS.

## ⚙ Summary of Steps

1️⃣ Environment Setup: Install required libraries.

2️⃣ Data Preparation: Create an FAQ text file and clean the text.

3️⃣ Tokenization & Embedding: Use Word2Vec to create vector embeddings.

4️⃣ API Development: Implement FastAPI endpoints.

5️⃣ Local LLM Setup: Use Ollama for response enhancement.

6️⃣ Web UI: Develop a simple web interface with Jinja2.

## 📂 Folder Structure

banking\_chatbot/  
│── data/  
│ ├── faqs.txt # FAQ text file  
│── embeddings/  
│ ├── chromadb/ # ChromaDB vector storage  
│── web/  
│ ├── templates/  
│ │ ├── index.html # Jinja2 web template  
│ ├── static/  
│ │ ├── styles.css # Optional CSS file  
│── api/  
│ ├── main.py # FastAPI backend  
│── models/  
│ ├── word2vec\_model # Saved Word2Vec model  
│── README.md

### 1️⃣ Install Required Packages

Run the following command to install dependencies:

pip install fastapi jinja2 chromadb gensim ollama langchain uvicorn

### 2️⃣ Sample Data (FAQs)

Q: How can I check my account balance?  
A: You can check your account balance through mobile banking, internet banking, or by visiting a nearby ATM.  
  
Q: What are the interest rates for savings accounts?  
A: The interest rates vary based on the type of savings account. Please visit our website for current rates.  
  
Q: How do I apply for a loan?  
A: You can apply for a loan online through our website or by visiting a bank branch.

### 3️⃣ Load and Process FAQ Data

import os  
  
# Load FAQ data from a text file  
def load\_faq\_data(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as f:  
 return f.readlines()  
  
faq\_data = load\_faq\_data('data/faqs.txt')

### 4️⃣ Tokenization & Word2Vec Embedding

from gensim.models import Word2Vec  
  
# Tokenize sentences  
sentences = [line.split() for line in faq\_data if line.strip()]  
  
# Train Word2Vec model  
word2vec\_model = Word2Vec(sentences, vector\_size=100, window=5, min\_count=1, workers=4)  
word2vec\_model.save("models/word2vec\_model")

### 5️⃣ Store and Retrieve Vectors using ChromaDB

import chromadb  
  
# Initialize ChromaDB  
chroma\_client = chromadb.PersistentClient(path="embeddings/chromadb")  
  
# Create a collection  
faq\_collection = chroma\_client.create\_collection(name="bank\_faqs")  
  
# Store vectors  
for i, sentence in enumerate(sentences):  
 vector = word2vec\_model.wv[sentence].tolist()  
 faq\_collection.add(id=str(i), embedding=vector, metadata={"text": ' '.join(sentence)})  
  
# Retrieve Similar FAQ  
query\_vector = word2vec\_model.wv["loan application"].tolist()  
results = faq\_collection.query(query\_vector, n\_results=1)  
print("Best Matched FAQ:", results["documents"][0])

### 6️⃣ FastAPI Backend

from fastapi import FastAPI, Query  
import chromadb  
  
app = FastAPI()  
  
# Load ChromaDB  
chroma\_client = chromadb.PersistentClient(path="embeddings/chromadb")  
faq\_collection = chroma\_client.get\_collection(name="bank\_faqs")  
  
@app.get("/chat/")  
async def chatbot\_query(q: str = Query(...)):  
 query\_vector = word2vec\_model.wv[q.split()].tolist()  
 results = faq\_collection.query(query\_vector, n\_results=1)  
 return {"response": results["documents"][0]}  
  
# Run the API using uvicorn main:app --reload

### 7️⃣ Web UI using Jinja2

from fastapi.templating import Jinja2Templates  
from fastapi.responses import HTMLResponse  
from fastapi import Request  
  
templates = Jinja2Templates(directory="web/templates")  
  
@app.get("/", response\_class=HTMLResponse)  
async def home(request: Request):  
 return templates.TemplateResponse("index.html", {"request": request})

### 8️⃣ Jinja2 Web Template (index.html)

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <title>Bank Chatbot</title>  
</head>  
<body>  
 <h2>Ask a Question:</h2>  
 <input type="text" id="user\_input" />  
 <button onclick="askChatbot()">Ask</button>  
 <p id="response"></p>  
 <script>  
 async function askChatbot() {  
 let query = document.getElementById("user\_input").value;  
 let response = await fetch(`/chat/?q=${query}`);  
 let data = await response.json();  
 document.getElementById("response").innerText = data.response;  
 }  
 </script>  
</body>  
</html>