PAWA IT SOLUTIONS — LLM Q&A Assessment

Project structure

```
pawa-llm-assessment/
├─ backend/
│  ├─ app/
│  │  └─ main.py
│  ├─ requirements.txt
│  └─ .env.example
├─ frontend/
│  ├─ package.json
│  ├─ next.config.js
│  ├─ postcss.config.js
│  ├─ tailwind.config.js
│  ├─ app/
│  │  ├─ globals.css
│  │  └─ page.jsx
│  └─ components/
│     ├─ Chat.jsx
│     └─ Message.jsx
└─ README.md
```

---

backend/requirements.txt

```
fastapi
uvicorn[standard]
python-dotenv
pydantic
openai
httpx
```

---

backend/.env.example

```
OPENAI_API_KEY=
OPENAI_MODEL=gpt-4o-mini
```

```
LLM_SYSTEM_PROMPT=You are a helpful assistant. Provide concise, well-structured
answers with sections when appropriate.
```

---

backend/app/main.py

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, BaseSettings
import os
import openai
import asyncio

class Settings(BaseSettings):
    openai_api_key: str
    openai_model: str = "gpt-4o-mini"
    llm_system_prompt: str = "You are a helpful assistant. Provide concise,
well-structured answers with sections when appropriate."
    class Config:
        env_file = ".env"

settings = Settings()
openai.api_key = settings.openai_api_key
app = FastAPI()

class QueryIn(BaseModel):
    query: str

class QueryOut(BaseModel):
    answer: str

@app.post("/api/query", response_model=QueryOut)
async def query_llm(payload: QueryIn):
    if not payload.query or len(payload.query) > 2000:
        raise HTTPException(status_code=400, detail="Invalid query")
    try:
        loop = asyncio.get_event_loop()
        def call_openai():
            resp = openai.ChatCompletion.create(
                model=settings.openai_model,
                messages=[
                    {"role": "system", "content": settings.llm_system_prompt},
                    {"role": "user", "content": payload.query}
                ],
                max_tokens=800,
                temperature=0.2
```

```
            )
            return resp
        resp = await loop.run_in_executor(None, call_openai)
        text = resp.choices[0].message.content.strip()
        return {"answer": text}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

---

frontend/package.json

```
{
  "name": "pawa-llm-frontend",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start"
  },
  "dependencies": {
    "next": "14.0.0",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "swr": "2.2.0"
  },
  "devDependencies": {
    "autoprefixer": "10.4.14",
    "postcss": "8.4.24",
    "tailwindcss": "4.0.0"
  }
}
```

---

frontend/next.config.js

```
const nextConfig = {
  reactStrictMode: true,
}
module.exports = nextConfig
```

---

frontend/postcss.config.js

```
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

---

frontend/tailwind.config.js

```
module.exports = {
  content: ["./app/**/*.{js,jsx,ts,tsx}", "./components/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

---

frontend/app/globals.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

html, body, #__next {
  height: 100%;
}
body {
  @apply bg-gradient-to-br from-blue-50 to-pink-50 text-gray-800;
}
```

---

frontend/app/page.jsx

```
import Chat from '../components/Chat'

export default function Page() {
```

```
  return (
    <main className="min-h-screen flex items-center justify-center p-6">
      <div className="w-full max-w-3xl bg-white/80 backdrop-blur rounded-2xl
shadow-xl p-6">
        <h1 className="text-2xl font-semibold mb-4">PAWA LLM Q&A</h1>
        <Chat />
      </div>
    </main>
  )
}
```

---

frontend/components/Chat.jsx

```
import { useState } from 'react'

export default function Chat() {
  const [query, setQuery] = useState('')
  const [loading, setLoading] = useState(false)
  const [messages, setMessages] = useState([])
  const apiUrl = process.env.NEXT_PUBLIC_API_URL || 'http://localhost:8000/api/
query'

  async function handleSubmit(e) {
    e.preventDefault()
    if (!query.trim()) return
    const userMsg = { role: 'user', text: query }
    setMessages(prev => [...prev, userMsg])
    setQuery('')
    setLoading(true)
    try {
      const res = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ query: userMsg.text })
      })
      const data = await res.json()
      const botMsg = { role: 'bot', text: data.answer }
      setMessages(prev => [...prev, botMsg])
    } catch (err) {
      const errMsg = { role: 'bot', text: 'Something went wrong. Try again.' }
      setMessages(prev => [...prev, errMsg])
    } finally {
      setLoading(false)
    }
```

```
  }

  return (
    <div>
      <div className="space-y-3 max-h-96 overflow-y-auto mb-4">
        {messages.map((m, i) => (
          <div key={i} className={m.role === 'user' ? 'text-right' : 'text-
left'}>
            <div className={`inline-block px-4 py-2 rounded-xl ${m.role ===
'user' ? 'bg-blue-600 text-white' : 'bg-pink-100 text-gray-800'}`}>
              {m.text}
            </div>
          </div>
        ))}
      </div>
      <form onSubmit={handleSubmit} className="flex gap-2">
        <textarea value={query} onChange={e => setQuery(e.target.value)}
rows={2} className="flex-1 p-3 rounded-lg border" placeholder="Ask
anything..." />
        <button type="submit" className="px-4 rounded-lg bg-blue-600 text-
white" disabled={loading}>
          {loading ? 'Thinking...' : 'Send'}
        </button>
      </form>
    </div>
  )
}
```

---

frontend/components/Message.jsx

```
export default function Message({ role, text }) {
  return (
    <div className={role === 'user' ? 'text-right' : 'text-left'}>
      <div className={`inline-block px-4 py-2 rounded-xl ${role === 'user' ?
'bg-blue-600 text-white' : 'bg-pink-100 text-gray-800'}`}>
        {text}
      </div>
    </div>
  )
}
```

README.md

```
# PAWA LLM Q&A Assessment

## Setup backend

1. cd backend
2. python -m venv venv
3. source venv/bin/activate
4. pip install -r requirements.txt
5. create .env from .env.example and fill OPENAI_API_KEY
6. uvicorn app.main:app --reload --port 8000

## Setup frontend

1. cd frontend
2. npm install
3. create .env.local with NEXT_PUBLIC_API_URL pointing to backend
4. npm run dev

## Notes

- The backend exposes POST /api/query accepting JSON {"query": "..."}
- The frontend sends requests to the backend and displays responses
- Include your prompt engineering details inside README before submission
```