COP509 Chat App — Exercise Sheet

You only need to edit `exercise.py`. Do not modify any other file. After each task, save the file, restart the server, and test in the browser at `http://localhost:8000`.

After following the `HowToRun.pdf` setup guide:
To start the server: `python exercise.py`
To stop the server: press **Ctrl** + **C** in the terminal.

Task 0: Get Your API Key and Model IDs

1. Go to `https://openrouter.ai` and sign up (free).
2. Go to **Keys** → **Create Key** → copy it.
3. Browse `https://openrouter.ai/models?q=free` and pick one or more free models. Copy their IDs (e.g. `stepfun/step-3.5-flash:free`).
4. Open the `env` file and fill in your values:

```
OPENROUTER_API_KEY=sk-or-v1-...
MODELS=stepfun/step-3.5-flash:free,another/model
```

5. Rename the file to `.env` (see the How To Run guide for the command).

Task 1: Initialise the OpenAI Client

We use the OpenAI Python SDK to connect to OpenRouter. Find the `TODO 1` section in `exercise.py` and create the client:

```python
client = OpenAI(
    base_url="https://openrouter.ai/api/v1",
    api_key=os.getenv("OPENROUTER_API_KEY")
)
```

How to test

No visible change yet, but errors will disappear in the next task.

Task 2: Stream Responses

Make the LLM respond word-by-word, like ChatGPT does. Replace the `placeholder()` function in the `TODO 2` section with four steps:

Step A: Build your messages list. For now, use just the system prompt and the user's message:

```
messages = [
    {"role": "system", "content": SYSTEM_PROMPT},
    {"role": "user",   "content": user_message},
]
```

Step B: Call the API with streaming.

```
stream = client.chat.completions.create(
    model=model,
    messages=messages,
    stream=True,
)
```

Step C: Write a generator that yields SSE events. The frontend expects each event as `data: {"content": "..."}\n\n`. End the stream with `data: [DONE]`.

```
def generate():
    for chunk in stream:
        content = chunk.choices[0].delta.content
        if content:
            yield f"data: {json.dumps({'content': content})}\n\n"
    yield "data: [DONE]\n\n"
```

Step D: Return a streaming response.

```
return StreamingResponse(generate(), media_type="text/event-stream")
```

Hint

Make sure you delete or replace the existing `placeholder()` function and its `return` statement.

How to test

Send a message in the browser. The response should appear word by word. The bot won't remember previous messages yet. That's the next task.

Task 3: Conversation Memory

Right now the LLM only sees the latest message. The frontend already sends the full conversation history in the `history` variable. Include it in your messages list:

```
messages = (
    [{"role": "system", "content": SYSTEM_PROMPT}]
    + history
    + [{"role": "user", "content": user_message}]
)
```

How to test

Have a multi-turn conversation. Ask “What is my name?” after introducing yourself. The bot should remember.

You now have a working chatbot. The tasks below are extensions.

Task 4 (Extension A): Vision

The frontend sends uploaded images as a base64 string in the `image_data` variable. Currently it is ignored. When an image is present, build a multimodal message instead of a plain string.

Add this **above** your `messages = [...]` line:

```
if image_data:
    user_content = [
        {"type": "text", "text": user_message},
        {"type": "image_url", "image_url": {
            "url": f"data:image/png;base64,{image_data}"
        }},
    ]
else:
    user_content = user_message
```

Then replace `user_message` with `user_content` in your messages list:

```
{"role": "user", "content": user_content}
```

Hint

Use a vision-capable model such as `nvidia/nemotron-nano-12b-v2-vl:free`.

How to test

Click the image button in the chat input, upload a photo, and ask the LLM to describe it.

Tasks 5–6 (Extension B): Conversation Persistence

Save and resume conversations using JSON files and REST endpoints. The endpoint stubs are already in `exercise.py`.

Task 5: Save a Conversation

Fill in the `TODO 5` section. Build a data dictionary and write it to a JSON file:

```
data = {
    "id": conv_id,
    "title": title,
    "messages": messages,
    "updated_at": datetime.now(timezone.utc).isoformat(),
}

with open(CONVERSATIONS_DIR / f"{conv_id}.json", "w") as f:
    json.dump(data, f, indent=2)
```

Task 6: List & Load Conversations

Fill in the two `TODO 6` sections.

GET /conversations: Return a list of all saved conversations, sorted by most recent first:

```
if not CONVERSATIONS_DIR.exists():
    return []

conversations = []
for filepath in CONVERSATIONS_DIR.glob("*.json"):
    with open(filepath) as f:
        data = json.load(f)
    conversations.append({
        "id": data["id"],
        "title": data.get("title", "Untitled"),
        "updated_at": data.get("updated_at", ""),
    })

conversations.sort(key=lambda c: c["updated_at"], reverse=True)
return conversations
```

GET /conversations/{conv_id}: Load a single conversation by ID:

```
filepath = CONVERSATIONS_DIR / f"{conv_id}.json"
if not filepath.exists():
    return JSONResponse({"error": "Not found"}, status_code=404)

with open(filepath) as f:
    data = json.load(f)
return data
```

How to test

Send a few messages, then refresh the page. Your conversations should appear in the History sidebar. Click one to resume it.

Task 7 (Extension C): LLM Benchmarking

Run standardised NLP benchmarks (ARC, GSM8K, HellaSwag) on your model using the `inspect_ai` framework. The frontend shows results live as each question is answered.

Fill in the `run_evaluation()` function. Add the import and paste the code below:

```
from inspect_ai import eval as inspect_eval

logs = inspect_eval(
    task_name,
    model=f"openrouter/{model}",
    limit=limit,
    log_dir=log_dir,
    log_buffer=1,
)

scores = logs[0].results.scores[0].metrics
first_metric = next(iter(scores.values()))
accuracy = round(first_metric.value * 100, 1)

return (logs, accuracy)
```

How to test

Click **New benchmark** in the sidebar, pick a benchmark (e.g. ARC Easy), a model, and 5 questions. Click **Run Benchmark** and watch results appear live. Try 25+ questions for more meaningful accuracy.

Useful References

- OpenRouter docs: https://openrouter.ai/docs
- OpenAI Python SDK: https://github.com/openai/openai-python
- Chat Completions API: https://platform.openai.com/docs/api-reference/chat
- Free models: https://openrouter.ai/models?q=free
- inspect_ai docs: https://inspect.ai-safety-institute.org.uk