―付録―

Python

```
from flask import Flask, request, jsonify, render_template
import openai
import os
import subprocess
import datetime
from dotenv import load_dotenv

# 仮想環境で.env ファイルを読み込む
load_dotenv(dotenv_path="C:/Users/user1/Desktop/tame2/.env")
api_key = os.getenv("OPENAI_API_KEY")
if not api_key:
    print("API キーが取得できません。環境変数を確認してください。")
else:
    print(f"API キー: {api_key[:10]}...（省略）")

app = Flask(__name__)

# ChatGPT API キーの設定（.env ファイルから読み込む）
openai.api_key = os.getenv("OPENAI_API_KEY")

# INP ファイル保存フォルダの設定（.env で定義 or デフォルト: "generated_files"）
SAVE_FOLDER = os.getenv("SAVE_FOLDER", "generated_files")
```

```
# 保存フォルダを作成（存在しない場合）
os.makedirs(SAVE_FOLDER, exist_ok=True)

# ユーザーの入力データを管理する辞書
user_data = {"step": 1, "all_data_collected": False, "file_saved": False}

@app.route("/")
def index():
    return render_template("index.html")

# チャットボットの応答を処理
@app.route("/get_response", methods=["POST"])
def get_response():
    global user_data
    user_message = request.json.get("message")

    # 初回アクセス時には質問を 1 から開始する
    if user_message == "":
        user_data["step"] = 1
        response_message = next_question(user_data["step"])
        return jsonify({"response": response_message})

    # ユーザーのメッセージを step に応じて記録
```

```
    step = user_data.get("step", 1)
    if step == 1:
        user_data["floors"] = user_message
    elif step == 2:
        user_data["columns_vertical"] = user_message
    elif step == 3:
        user_data["columns_horizontal"] = user_message
    elif step == 4:
        user_data["column_height"] = user_message
    elif step == 5:
        user_data["column_spacing"] = user_message
    elif step == 6:
        user_data["section_shape"] = user_message
    elif step == 7:
        user_data["material"] = user_message
    elif step == 8:
        user_data["load_condition"] = user_message
    elif step == 9:
        user_data["boundary_condition"] = user_message
        # すべての情報が揃ったので確認ステップへ移行
        user_data["all_data_collected"] = True
        return jsonify({"response": "すべての情報が揃いました。ChatGPT にデータを送信しますか？（yes/no）"})
```

```
    # 次のステップに進む
    step += 1
    user_data["step"] = step

    # 次の質問を送信
    response_message = next_question(step)
    return jsonify({"response": response_message})

# モデル生成を確認する部分
@app.route("/process_confirmation", methods=["POST"])
def process_confirmation():
    global user_data
    confirmation = request.json.get("message").lower()

    if confirmation == "yes" and user_data["all_data_collected"]:
        # INP ファイルの生成処理
        inp_content = generate_inp_file(user_data)
        if "エラーが発生しました" in inp_content:
            return jsonify({"response": inp_content})
        file_path = save_inp_file(inp_content)  # INP ファイルを保存
        user_data["generated_file_path"] = file_path  # 保存されたファイルのパスを記録
        user_data["file_saved"] = True  # モデルが保存されたフラグを立てる
        return jsonify({"response": f"INP ファイルが生成され、次のパスに保存されました:
{file_path}"})
```

```
    elif confirmation == "no":
        user_data["all_data_collected"] = False
        user_data["step"] = 1
        return jsonify({"response": "操作を中断しました。"})

    else:
        return jsonify({"response": "無効な入力です。'yes' か 'no' で答えてください。"})

@app.route("/process_analysis", methods=["POST"])
def process_analysis():
    global user_data
    confirmation = request.json.get("message").lower()

    if confirmation == "yes":
        # ファイルが保存されているか確認
        if user_data.get("file_saved", False):  # モデルが保存されているか確認
            file_path = user_data["generated_file_path"]
            # ファイルの存在を確認してから解析を実行
            if os.path.exists(file_path):
                analysis_result = run_analysis(file_path)
                return jsonify({"response": analysis_result})
            else:
                return jsonify({"response": "INP ファイルが見つかりません。保存後に再実
```

行してください。"})

else:

return jsonify({"response": "INP ファイルがまだ生成されていません。保存してから解析を実行してください。"})

elif confirmation == "no":

return jsonify({"response": "解析を中断しました。"})

else:

return jsonify({"response": "無効な入力です。'yes' か 'no' で答えてください。"})

INP ファイルを生成する関数

def generate_inp_file(data):

prompt = f"""

これらのデータを使い、下記に示す条件に沿って Abaqus で使用する INP ファイルを生成してください。

- 建物の階数: {data['floors']}
- x 方向の柱の数: {data['columns_vertical']}
- y 方向の柱の数: {data['columns_horizontal']}
- 柱の高さ: {data['column_height']} m
- 柱の間隔: {data['column_spacing']} m
- 断面形状: {data['section_shape']}
- 材料: {data['material']}

- 荷重条件: 最上階の柱の上端の全ての点に Z 方向に 10 の{data['load_condition']}

- 境界条件: z = 0 の点を{data['boundary_condition']}

-条件

- モデルは次に示す内容を順番に生成してください

-NODE,ELEMENT,MATERIAL,BEAM SECTION,NSET,BOUNDARY,STEP,CLOAD,END STEP

- x 方向の柱の数と y 方向の柱の数を上記のデータとおりになるように注意してください

- 地面では、梁が存在しないように ELEMENT で定義しないように注意してください

- ELEMENT で、elset を柱は COLUMN、梁は柱間に X 方向に BEAM1、Y 方向に BEAM2 と分けてそれぞれ定義する

- elset で柱や梁をしっかりと全て定義して、material に進むようにしてください

- 空白などの改行はしないでください

- 説明文は入れないでください

- 地面（原点）から最上階（{data['floors']}階）までの全ての階の柱と梁を正確に生成

-boundary の前に次に示すコードをそのまま 0 や 1 の順番を変えずに反映してください

-*Beam Section, elset=COLUMN, material=Steel, section=RECT

-0.2, 0.6

-1, 0, 0

-*Beam Section, elset=BEAM1, material=Steel, section=RECT

-0.2, 0.6

-0, 1, 0

-*Beam Section, elset=BEAM2, material=Steel, section=RECT

-0.2, 0.6

-1, 0, 0

```
    -boundary では、ENCASTRE ではなく、(1,6) にしてください
    -step は static とする。
    """
    try:
        response = openai.ChatCompletion.create(
            model="gpt-4-turbo",
            messages=[{"role": "system", "content": "INP ファイル生成のアシスタントです。"},
                      {"role": "user", "content": prompt}],
            max_tokens=3000,
            temperature=0.7
        )
        return response.choices[0].message['content'].strip()
    except openai.OpenAIError as e:
        return f"エラーが発生しました: {str(e)}"

# 次の質問を取得する関数
def next_question(step):
    questions = {
        1: "建物の階数は何階ですか？",
        2: "横方向の柱の数はいくつですか？",
        3: "縦方向の柱の数はいくつですか？",
        4: "柱の高さは何メートルですか？",
        5: "柱の間隔は何メートルですか？",
```

```
        6: "断面形状はどうしますか？（例: H 型鋼）",
        7: "使用する材料は何ですか？(例: steel)",
        8: "荷重条件はどうしますか？（集中荷重、等分布荷重など）",
        9: "境界条件はどうしますか？（固定支持、ピン支持など）"
    }
    return questions.get(step, "すべての情報が揃いました。ChatGPT にデータを送信しますか？（yes/no）")

# INP ファイルを保存する関数
def save_inp_file(content):
    try:
        current_time = datetime.datetime.now().strftime("%m%d_%H%M")
        file_name = f"generated_model_{current_time}.inp"
        file_path = os.path.join(SAVE_FOLDER, file_name)
        with open(file_path, "w", encoding="utf-8") as file:
            file.write(content)
        return file_path  # 正常に保存された場合、パスを返す
    except Exception as e:
        return f"エラーが発生しました: {str(e)}"  # 保存失敗の場合、エラーメッセージを返す

def run_analysis(file_path):
    try:
        inp_file_name = os.path.basename(file_path)  # フルパスからファイル名を取得
```

```
        inp_file_dir = os.path.dirname(file_path)  # ファイルのディレクトリパス

        # Abaqus のフルパスを指定（環境変数なしで直接指定）
        abaqus_path = r"C:¥SIMULIA¥CAE¥2023LE¥win_b64¥resources¥install¥le¥launcher.bat" # ここにあなたの Abaqus のインストールパスを設定

        # subprocess.run() を使用して Abaqus を実行
        result = subprocess.run(
            [abaqus_path, "job=" + inp_file_name, "interactive"],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            cwd=inp_file_dir  # カレントディレクトリをファイルのディレクトリに変更
        )

        if result.returncode == 0:
            return f"解析が正常に完了しました。{inp_file_name}の解析が終了しました。"
        else:
            return f"解析が失敗しました。エラーコード: {result.returncode}¥n エラー内容: {result.stderr}"

    except Exception as e:
        return f"解析中にエラーが発生しました: {str(e)}"
```

```
# 会話データを確認するエンドポイント
@app.route("/get_user_data", methods=["GET"])
def get_user_data():
    return jsonify(user_data)


if __name__ == "__main__":
    app.run(debug=True)
```