In [1]:
```python
# imports

import os
import json
from dotenv import load_dotenv
from openai import OpenAI
import gradio as gr
import sqlite3
```

In [2]:
```python
# Initialization

load_dotenv(override=True)

openai_api_key = os.getenv('OPENAI_API_KEY')
if openai_api_key:
    print(f"OpenAI API Key exists and begins {openai_api_key[:8]}")
else:
    print("OpenAI API Key not set")

MODEL = "gpt-4.1-mini"
openai = OpenAI()

DB = "prices.db"
```

```
OpenAI API Key exists and begins sk-proj-
```

In [4]:
```python
system_message = """
You are a helpful assistant for an Airline called FlightAI.
Give short, courteous answers, no more than 1 sentence.
Always be accurate. If you don't know the answer, say so.
"""
```

In [ ]:
```python
# A dictionary to store key-value pairs like cities and prices.
def get_ticket_price(city):
  """
  Looks up the ticket price for a given city.
  """
  ticket_prices = {
      "Paris": 200,
      "London": 150,
      "Tokyo": 800,
      "New York": 450
  }

  # Use .get() for a safe lookup. It returns a default value (None or a message)
  # if the city is not in the dictionary, preventing a KeyError.
  price = ticket_prices.get(city)

  if price is not None:
    return f"The ticket price for {city} is ${price}."
  else:
    return f"Sorry, we don't have a ticket price for {city}."
```

```
# Now that the function is defined, you can call it.
print(get_ticket_price("Paris"))
print(get_ticket_price("Tokyo"))
print(get_ticket_price("Sydney"))
```

```
The ticket price for Paris is $200.
The ticket price for Tokyo is $800.
Sorry, we don't have a ticket price for Sydney.
```

In [6]:
```
get_ticket_price("Paris")
```

Out[6]:
```
'The ticket price for Paris is $200.'
```

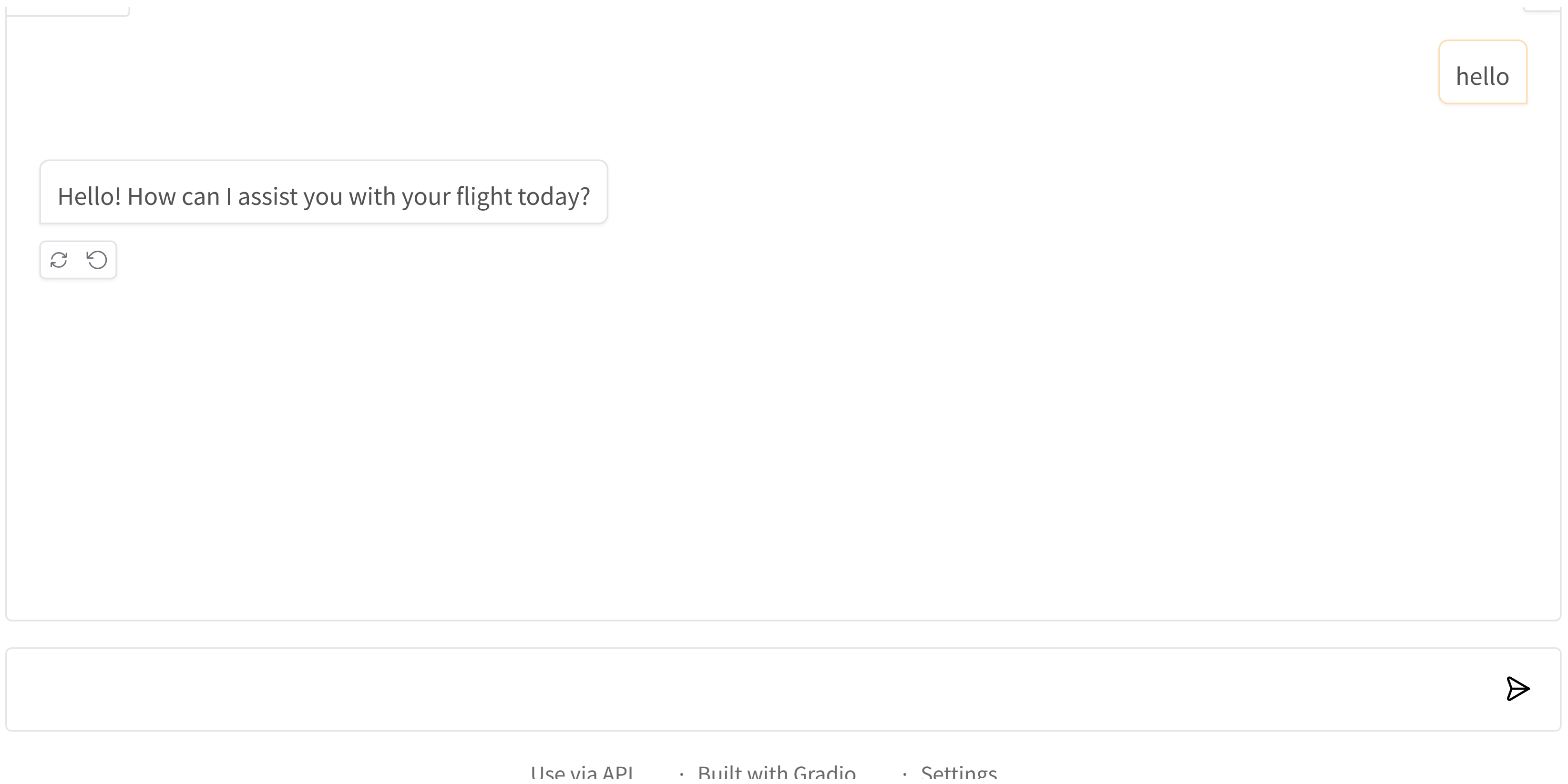
In [7]:
```
price_function = {
    "name": "get_ticket_price",
    "description": "Get the price of a return ticket to the destination city.",
    "parameters": {
        "type": "object",
        "properties": {
            "destination_city": {
                "type": "string",
                "description": "The city that the customer wants to travel to",
            },
        },
        "required": ["destination_city"],
        "additionalProperties": False
    }
}
tools = [{"type": "function", "function": price_function}]
tools
```

Out[7]:
```
[{'type': 'function',
  'function': {'name': 'get_ticket_price',
   'description': 'Get the price of a return ticket to the destination city.',
   'parameters': {'type': 'object',
    'properties': {'destination_city': {'type': 'string',
      'description': 'The city that the customer wants to travel to'}},
    'required': ['destination_city'],
    'additionalProperties': False}}}]
```

In [8]:
```
def chat(message, history):
    history = [{"role": h["role"], "content": h["content"]} for h in history]
    messages = [{"role": "system", "content": system_message}] + history + [{"role": "user", "content": message}]
    response = openai.chat.completions.create(model=MODEL, messages=messages)
    return response.choices[0].message.content

gr.ChatInterface(fn=chat, type="messages").launch()
```

```
* Running on local URL:  http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.
```


hello

Hello! How can I assist you with your flight today?

Out[8]:

In [12]:
```python
def chat(message, history):
    history = [{"role":h["role"], "content":h["content"]} for h in history]
    messages = [{"role": "system", "content": system_message}] + history + [{"role": "user", "content": message}]
    response = openai.chat.completions.create(model=MODEL, messages=messages, tools=tools)

    while response.choices[0].finish_reason=="tool_calls":
        message = response.choices[0].message
        responses = handle_tool_calls(message)
        messages.append(message)
        messages.extend(responses)
        response = openai.chat.completions.create(model=MODEL, messages=messages, tools=tools)

    return response.choices[0].message.content
```

In [13]:
```python
def handle_tool_calls(message):
    responses = []
    for tool_call in message.tool_calls:
        if tool_call.function.name == "get_ticket_price":
            arguments = json.loads(tool_call.function.arguments)
            city = arguments.get('destination_city')
            price_details = get_ticket_price(city)
            responses.append({
                "role": "tool",
                "content": price_details,
                "tool_call_id": tool_call.id
            })
    return responses
```

In [14]:
```
gr.ChatInterface(fn=chat, type="messages").launch()
```

```
* Running on local URL:  http://127.0.0.1:7862
* To create a public link, set `share=True` in `launch()`.
```

Out[14]:

In [ ]:
```
#imports for handling images

import base64
from io import BytesIO
from PIL import Image
```

In [16]:
```
def artist(city):
    image_response = openai.images.generate(
            model="dall-e-3",
            prompt=f"An image representing a vacation in {city}, showing tourist spots and everything unique about {city}, in a vibrant pop-art style",
            size="1024x1024",
            n=1,
            response_format="b64_json",
        )
    image_base64 = image_response.data[0].b64_json
    image_data = base64.b64decode(image_base64)
    return Image.open(BytesIO(image_data))
```

In [18]:
```
image = artist("bangalore")
display(image)
```

In [ ]:
```python
def talker(message):
    response = openai.audio.speech.create(
      model="gpt-4o-mini-tts",
      voice="onyx",
      input=message
    )
    return response.content
```

In [20]:
```python
def chat(history):
    history = [{"role":h["role"], "content":h["content"]} for h in history]
    messages = [{"role": "system", "content": system_message}] + history
    response = openai.chat.completions.create(model=MODEL, messages=messages, tools=tools)
    cities = []
    image = None

    while response.choices[0].finish_reason=="tool_calls":
        message = response.choices[0].message
        responses, cities = handle_tool_calls_and_return_cities(message)
        messages.append(message)
        messages.extend(responses)
        response = openai.chat.completions.create(model=MODEL, messages=messages, tools=tools)

    reply = response.choices[0].message.content
    history += [{"role":"assistant", "content":reply}]

    voice = talker(reply)

    if cities:
        image = artist(cities[0])

    return history, voice, image
```

In [21]:
```python
def handle_tool_calls_and_return_cities(message):
    responses = []
    cities = []
    for tool_call in message.tool_calls:
        if tool_call.function.name == "get_ticket_price":
            arguments = json.loads(tool_call.function.arguments)
            city = arguments.get('destination_city')
            cities.append(city)
            price_details = get_ticket_price(city)
            responses.append({
                "role": "tool",
                "content": price_details,
                "tool_call_id": tool_call.id
            })
    return responses, cities
```

In [23]:
```python
# Callbacks (along with the chat() function above)

def put_message_in_chatbot(message, history):
        return "", history + [{"role":"user", "content":message}]

# UI definition

with gr.Blocks() as ui:
    with gr.Row():
        chatbot = gr.Chatbot(height=500, type="messages")
        image_output = gr.Image(height=500, interactive=False)
    with gr.Row():
        audio_output = gr.Audio(autoplay=True)
    with gr.Row():
        message = gr.Textbox(label="Chat with our AI Assistant:")

# Hooking up events to callbacks

    message.submit(put_message_in_chatbot, inputs=[message, chatbot], outputs=[message, chatbot]).then(
        chat, inputs=chatbot, outputs=[chatbot, audio_output, image_output]
    )

ui.launch(inbrowser=True, auth=("darshith", "Ilovemangoes"))
```

```
* Running on local URL:  http://127.0.0.1:7864
* To create a public link, set `share=True` in `launch()`.
```

```
Out[23]:
```