

BUILDING AND DEPLOYING AI AGENTS

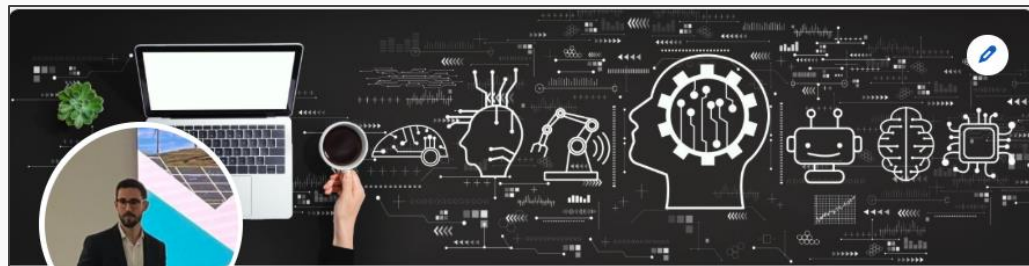

Class 5 - AI Agents in Production





Who the heck is Luis Dias?

Meet the man behind the glasses





Luis Dias ✓

AI practitioner & Leader || AI Solutions Architect - AI Lab@TUI ||
Transforming AI Technology into Strategic Business Value || Driving
Innovation & Impact 🌐 ⚡

Portugal · [Informações de contato](#)

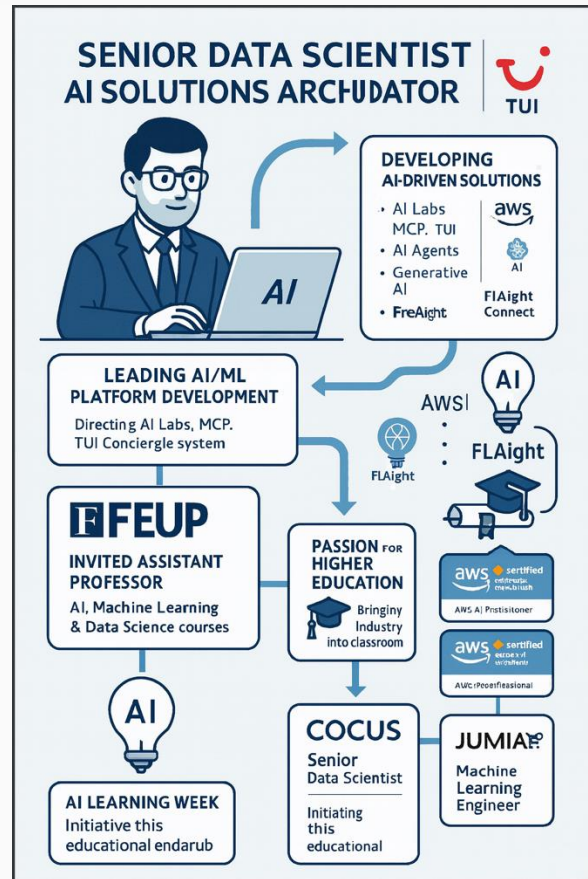
+ de 500 conexões

 TUI

 Faculdade de Engenharia da
Universidade do Porto

 luis.f.s.m.dias@gmail.com

 <https://www.linkedin.com/in/luisfilipedias/>



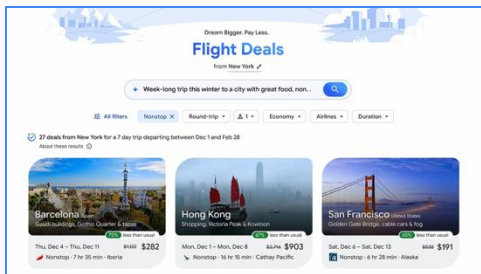
We will focus on popular technologies to make it happen

1. **Agents in Production** – Understanding the moving components.
2. **Streamlit** - Will be used to build the user interface (UI).
3. **Streamlit cloud deployment** - Remote deployment environments for our Streamlit UI.
4. **FastAPI** - Python framework that abstracts the complexity of developing an API.
5. **Render** - Remote deployment environments for our API.
6. **Langfuse** - To set some standard monitoring from the beginning.
7. **Reflection** - Connecting the dots of what we learned today.
8. **Assignment** - Hands-on exercise to consolidate today's learnings.
9. **Wrap up** - Next steps

AI Agents in production

Since the rise of ChatGPT in 2022 we have seen many new AI Applications using AI Agents

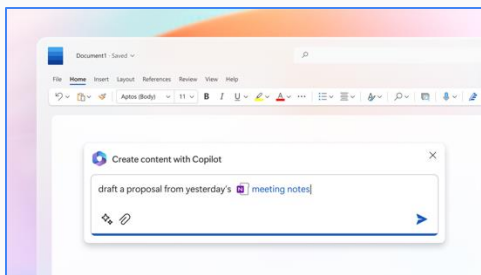
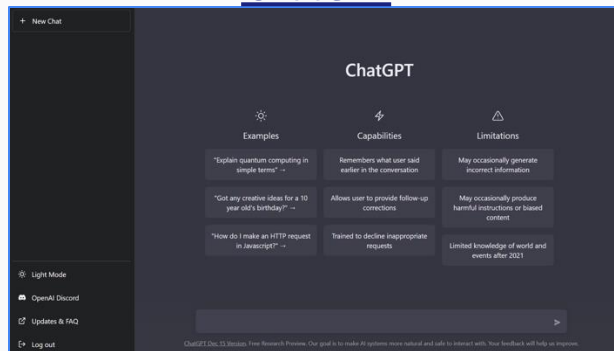
Google flight deals



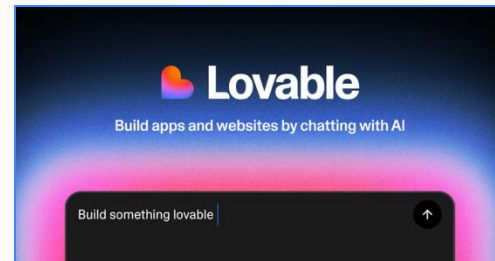
Claude Code



ChatGPT



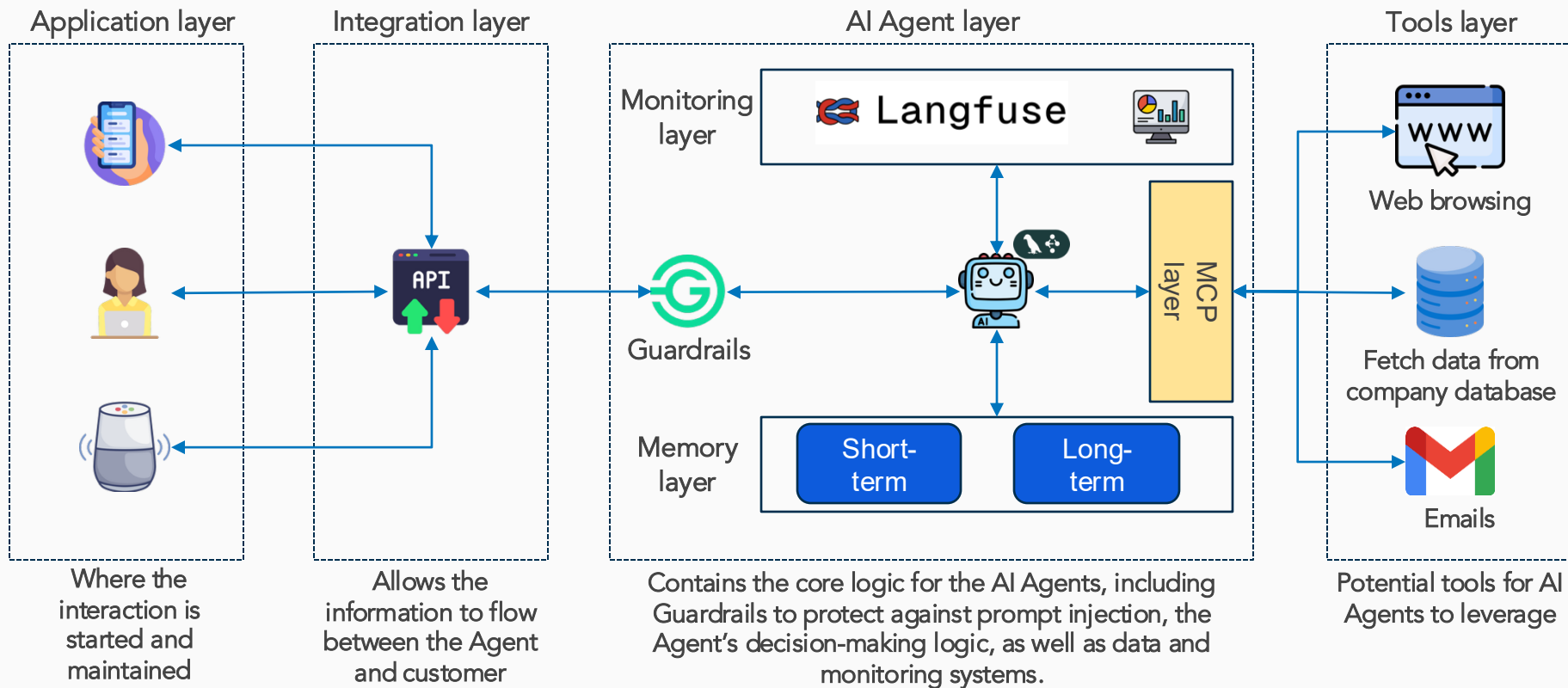
Microsoft Copilot



Lovable

What Does an Agent in Operations Look Like?

High-level overview of the main components of an AI Agent ready for production

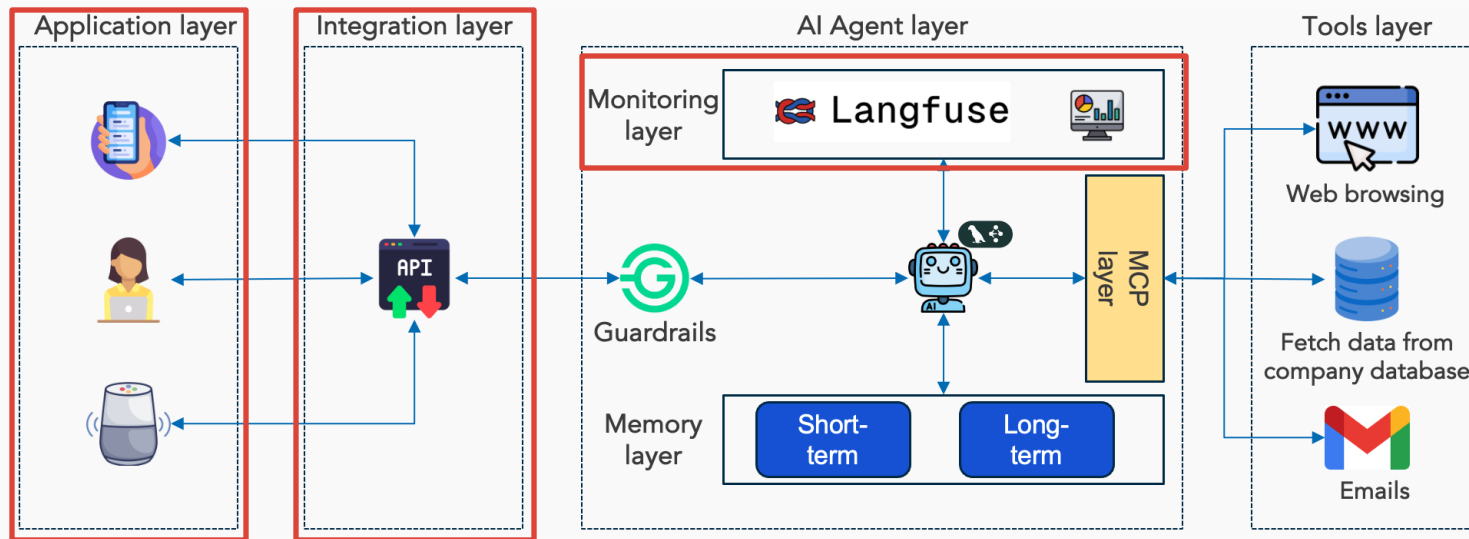


Agents are not useful if they stay in a notebook.

Deployment = making agents usable by people and systems. This means:

- **UI-first:** easy access for end users
- **API-first:** scalable, reusable, workflow integration
- **Monitoring:** track performance and health

Today we will tackle three layers of an AI Agent in Production





<https://github.com/tutai-diaz/buidling-deploying-agents-applications/tree/main>

The screenshot shows the GitHub repository page for 'buidling-deploying-agents-applications' by user 'Luis Dias'. The repository is private and has 0 stars, 0 forks, and 0 watches. It has 1 branch (main) and 0 tags. The repository description is 'Course covering the essentials on how to build and deploy AI Agents'. The file list shows a directory 'classes/class-05-deployment-interfaces/de...' and a file 'resources/streamlit'. The README is titled 'Building & Deploying AI Agents Applications' and describes the repository as a shared workspace for a course on building and deploying AI agent applications. It includes a 'Repository Map' with links to 'classes/' (slide decks, demos, and reference notes), 'assignments/' (lab instructions, reflection prompts, and submission templates), and 'capstone/' (idea bank, weekly reflection prompts, design blueprints, and grading rubric checkpoints). The right sidebar shows 'About', 'Releases', 'Packages', 'Languages' (Python 100.0%), and 'Suggested workflows'.

buidling-deploying-agents-applications Private

Edit Pins Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file Add file Code

Luis Dias Add Streamlit resource hub README and example apps for various functi... d51b7e0 · 16 hours ago 3 Commits

- classes/class-05-deployment-interfaces/de... Add initial project structure with .gitignore, README, and ... 16 hours ago
- resources/streamlit Add Streamlit resource hub README and example apps f... 16 hours ago
- .gitignore Add initial project structure with .gitignore, README, and ... 16 hours ago
- README.md Add initial project structure with .gitignore, README, and ... 16 hours ago

README

Building & Deploying AI Agents Applications

This repository is the shared workspace for the course on building and deploying AI agent applications. Each class drops fresh materials, hands-on labs, and references here so you always know where to look and what to work on next.

Repository Map

- classes/ – slide decks, demos, and reference notes organized per class session.
- assignments/ – lab instructions, reflection prompts, and submission templates.
- capstone/ – idea bank, weekly reflection prompts, design blueprints, and grading rubric checkpoints.

About

Course covering the essentials on how to build and deploy AI Agents

Readme Activity Custom properties 0 stars 0 watching 0 forks

Releases

No releases published [Create a new release](#)

Packages

No packages published [Publish your first package](#)

Languages

Python 100.0%

Suggested workflows

Based on your tech stack

Streamlit

The king of UI in Python for non frontend developers

Why use streamlit?

- Ideal for supporting Data Science, Machine Learning and AI projects
- Enables creation of interactive interfaces
- Designed for beginners – no front-end skills required
- With available widgets and elements, web pages can be built with just a few lines of code
- Compatible with most Python libraries

You can find many examples here: <https://streamlit.io/gallery>

App Gallery

Try out these apps, browse their source code, then fork them and make them your own. Also check out [Streamlit Community Cloud](#) for more.

CATEGORIES

Favorites

Trending

LLMs

Snowflake powered

Data visualization

Geography & society


Sports & fun

Science & technology

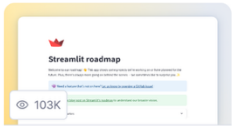
NLP & language

Finance & business


Other




Streamlit extras
arnaudmiribel
View source →



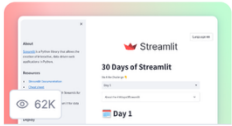
Roadmap
streamlit
View source →




prettypapp
chrieke
View source →



GW Quickview
jkanner
View source →



30Days of Streamlit
streamlit
View source →



Streamlit ECharts Demo
andfanilo
View source →

How to start?

Install the Streamlit library using 'pip install streamlit'. To test the installation, you can run the command 'streamlit hello'

```
(streamlitTutorial) (base) → streamlitTutorial streamlit hello

Welcome to Streamlit. Check out our demo in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.89:8501

Ready to create your own Python apps super quickly?
Head over to https://docs.streamlit.io

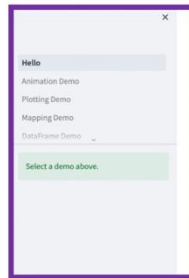
May you create awesome apps!
```

Command to
start Streamlit

URL to reach the web
server at port 8501

Hamburger
menu

Sidebar with access to
sample demos



Welcome to Streamlit! 🍌

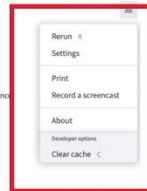
Streamlit is an open-source app framework built specifically for Machine Learning and Data Science projects. 🍌 Select a demo from the sidebar to see some examples of what Streamlit can do!

Want to learn more?

- Check out [streamlit.io](https://docs.streamlit.io)
- Jump into our [documentation](https://docs.streamlit.io)
- Ask a question in our [community forums](https://discuss.streamlit.io)

See more complex demos

- Use a neural net to [analyze the Udacity Self-driving Car Image Dataset](#)
- Explore a [New York City rideshare dataset](#)



Using the 'streamlit run'



Class 5 streamlit-chat-ui demo



```
.venv ~/Desktop/building-deploying-agents-applications/classes/class-05-deployment-interfaces/demos/streamlit-chat-ui git:(main)
streamlit run app.py

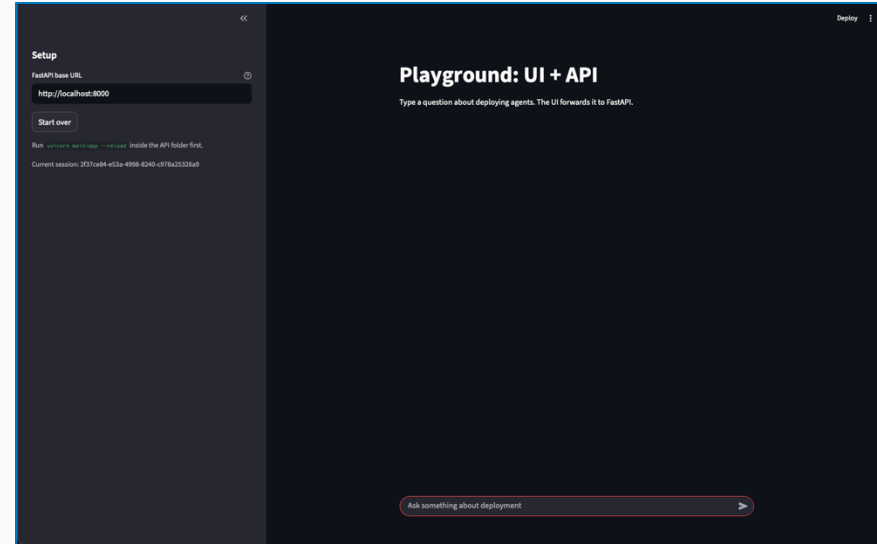
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.25:8501

For better performance, install the Watchdog module:

$ xcode-select --install
$ pip install watchdog
```

To run a Streamlit application, use the command `'streamlit run app.py'` in your terminal, where `app.py` (example) is the Python script containing the Streamlit code.



- Whenever the Python script is saved, the application can be refreshed by clicking '**Rerun**', without needing to restart the server.
- By choosing '**Always rerun**', the app updates automatically with each save, allowing you to see changes immediately.
- Whenever something needs to be updated on the screen (including user interactions), Streamlit re-runs the script from top to bottom.
- You can stop Streamlit with the command '**CTRL+C**'.



Source file changed.

Rerun

Always rerun



Text and titles

```
import streamlit as st

# Add a title
st.title('My First Streamlit App')

# Add header
st.header('This is a header')

# Add subheader
st.subheader('This is a subheader')

# Add normal text
st.text('This is regular text')

# Add markdown
st.markdown('### This is markdown with *emphasis*')

# Display a Python object's information
st.write('Display data or variables:', {'data': 'values'})
```



My First Streamlit App

This is a header

This is a subheader

This is regular text

This is markdown with *emphasis*

Display data or variables:

```
{
  "data" : "values"
}
```


Work with data



```
import streamlit as st
import pandas as pd
import numpy as np

# Create sample data
df = pd.DataFrame({
    'Name': ['John', 'Mary', 'Bob', 'Jane'],
    'Age': [25, 30, 22, 28],
    'City': ['New York', 'Boston', 'Chicago', 'Seattle']
})

# Display the dataframe
st.write('### Sample Data:')
st.dataframe(df)

# Display static table
st.table(df)

# Display statistics
st.write('### Data Statistics:')
st.write(df.describe())
```



Sample Data:

	Name	Age	City
0	John	25	New York
1	Mary	30	Boston
2	Bob	22	Chicago
3	Jane	28	Seattle

	Name	Age	City
0	John	25	New York
1	Mary	30	Boston
2	Bob	22	Chicago
3	Jane	28	Seattle

Data Statistics:

	Age
count	4
mean	26.25
std	3.5
min	22
25%	24.25
50%	26.5
75%	28.5
max	30

Build plots

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px

# Create sample data
chart_data = pd.DataFrame(np.random.randn(20, 3), columns=['A', 'B', 'C'])

# Matplotlib chart
st.write('### Matplotlib Chart')
fig, ax = plt.subplots()
ax.hist(chart_data['A'], bins=20)
st.pyplot(fig)

# Native Streamlit line chart
st.write('### Streamlit Line Chart')
st.line_chart(chart_data)

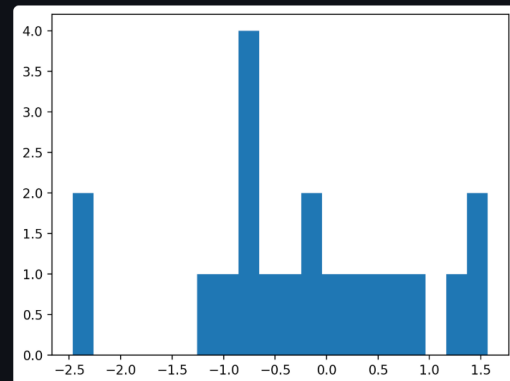
# Native Streamlit area chart
st.write('### Streamlit Area Chart')
st.area_chart(chart_data)

# Native Streamlit bar chart
st.write('### Streamlit Bar Chart')
st.bar_chart(chart_data)

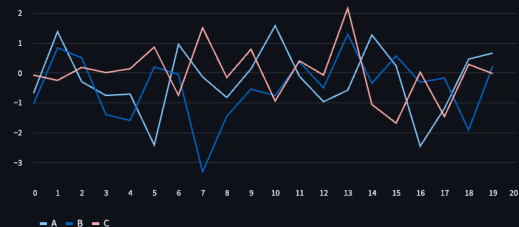
# Plotly chart (more interactive)
st.write('### Plotly Chart')
fig = px.scatter(chart_data, x='A', y='B', color='C', size='C')
st.plotly_chart(fig)
```



Matplotlib Chart



Streamlit Line Chart



Widgets

```
import streamlit as st

# Text input
user_input = st.text_input("Enter some text")
st.write('You entered:', user_input)

# Number input
number = st.number_input("Enter a number", min_value=0, max_value=100, value=50)
st.write('Selected number:', number)

# Slider
slider_value = st.slider("Select a range", 0, 100, 25)
st.write('Slider value:', slider_value)

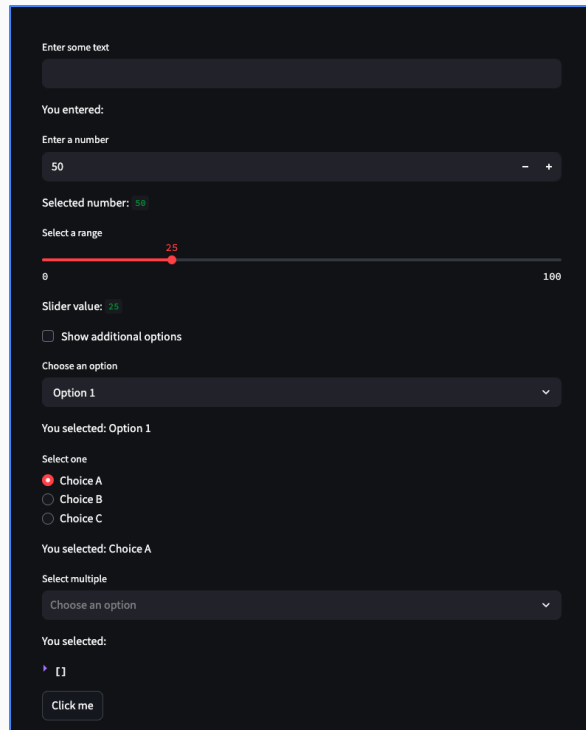
# Checkbox
if st.checkbox("Show additional options"):
    st.write("You selected to show additional options!")

# Selectbox
option = st.selectbox("Choose an option", ["Option 1", "Option 2", "Option 3"])
st.write('You selected:', option)

# Radio buttons
radio_option = st.radio("Select one", ["Choice A", "Choice B", "Choice C"])
st.write('You selected:', radio_option)

# Multiselect
multi_options = st.multiselect("Select multiple", ["Item 1", "Item 2", "Item 3", "Item 4"])
st.write('You selected:', multi_options)

# Button
if st.button("Click me"):
    st.write("Button clicked!")
```



The screenshot displays a Streamlit application interface with the following components:

- Text input:** A label "Enter some text" above a text input field.
- Text output:** The text "You entered:" followed by the content of the text input field.
- Number input:** A label "Enter a number" above a number input field showing the value "50".
- Slider:** A label "Selected number: 98" above a slider widget. The slider range is from 0 to 100, with a current value of 25.
- Checkbox:** A label "Slider value: 25" above a checkbox labeled "Show additional options".
- Selectbox:** A label "Choose an option" above a dropdown menu showing "Option 1".
- Text output:** The text "You selected: Option 1".
- Radio buttons:** A label "Select one" above three radio buttons labeled "Choice A", "Choice B", and "Choice C". "Choice A" is selected.
- Text output:** The text "You selected: Choice A".
- Multiselect:** A label "Select multiple" above a multiselect dropdown menu showing "Choose an option".
- Text output:** The text "You selected:" followed by the content of the multiselect widget.
- Button:** A button labeled "Click me".

Caching to improve the app response time

```
import streamlit as st
import pandas as pd
import time

# Caching data loading
@st.cache_data
def load_large_dataset():
    time.sleep(2) # Simulate long loading time
    return pd.DataFrame({
        'A': range(1000),
        'B': range(1000, 2000)
    })

data = load_large_dataset()
st.write('Loaded data:', data.head())
```



Loaded data:

	A	B
0		1000
1		1001
2		1002
3		1003
4		1004

Layout options

```
import streamlit as st

# Create columns
col1, col2 = st.columns(2)

with col1:
    st.header("Column 1")
    st.write("This is the first column")
    st.image("https://streamlit.io/images/brand/streamlit-mark-color.png", width=200)

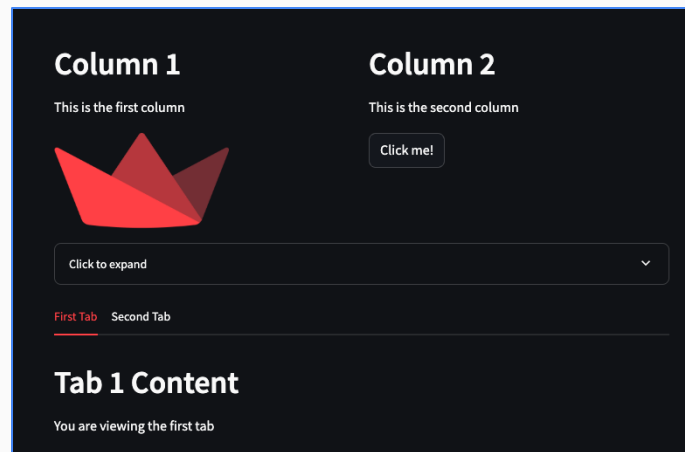
with col2:
    st.header("Column 2")
    st.write("This is the second column")
    if st.button("Click me!"):
        st.write("Thanks for clicking!")

# Create expandable sections
with st.expander("Click to expand"):
    st.write("This content is hidden until expanded")
    st.bar_chart({"data": [1, 5, 2, 6, 2, 1]})

# Create tabs
tab1, tab2 = st.tabs(["First Tab", "Second Tab"])

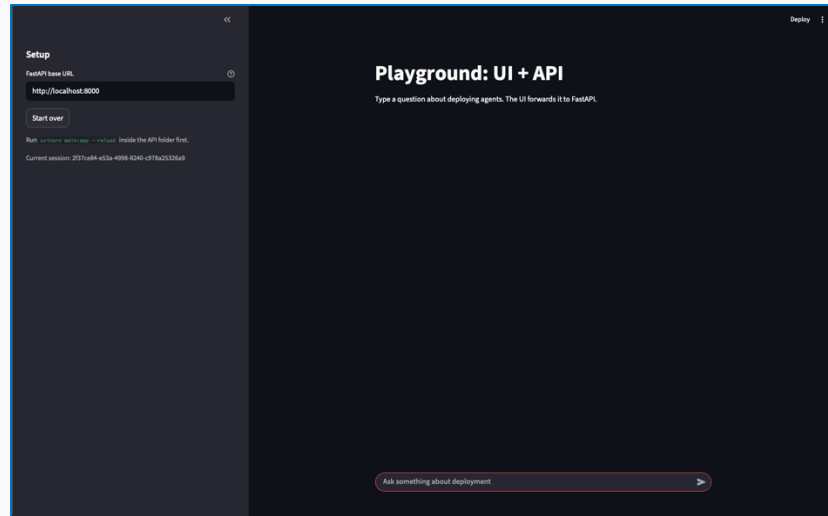
with tab1:
    st.header("Tab 1 Content")
    st.write("You are viewing the first tab")

with tab2:
    st.header("Tab 2 Content")
    st.write("You are viewing the second tab")
```



Today we will deploy a Chat UI in the Streamlit Community cloud

The screenshot shows a GitHub repository page for 'streamlit-chat-ui' within the 'buidling-deploying-agents-applications / classes / class-05-deployment-interfaces / demos' directory. The repository is owned by 'Luis Dias'. A table lists files: 'README.md', 'app.py', 'requirements.txt', and 'README.md' (selected). The selected file's content is visible, showing the title 'Streamlit Chat UI', a description of a beginner-friendly front-end for the Class 5 agent API, instructions on how to launch the UI (ensuring FastAPI is running, staying in the folder, and installing Python 3.10+), and a step-by-step guide to run Streamlit locally (creating a virtual environment and running 'python3 -m venv .venv').



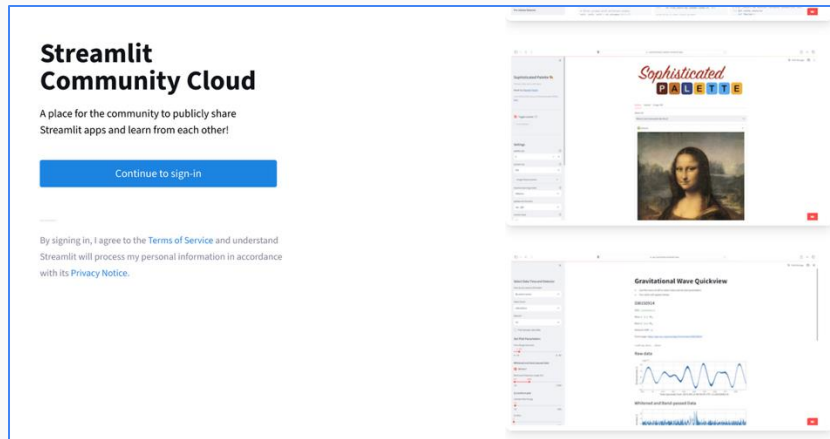
Link to demo: <https://github.com/tutai-diaz/buidling-deploying-agents-applications/tree/main/classes/class-05-deployment-interfaces/demos/streamlit-chat-ui>

Streamlit demo



Make the UI available to everyone

- To share your application with others, Streamlit Community Cloud offers free hosting:
- Push your code to a GitHub repository
- Visit <https://share.streamlit.io/>
- Connect your GitHub account
- Select your repository and main Python file
- Deploy your application



The Streamlit demo was just a quick intro. I advise to go through a dedicated tutorial of Streamlit!

Official documentation:

- [Streamlit Documentation](#) - Full usage details
- [Streamlit GitHub Repository](#) - Streamlit source code

Tutorials:

- [Streamlit Community Forum](#) - Official Streamlit forum
- [30 Days of Streamlit](#) - 30-day course with challenges
- [Streamlit YouTube Channel](#) - Tutorials and video demos

Streamlit App examples:

- [Streamlit Gallery](#) - encompasses community built app
- [GitHub Example Repositories](#) - open-source examples



Modern, fast (high-performance), web framework for building APIs with Python



What is an API?

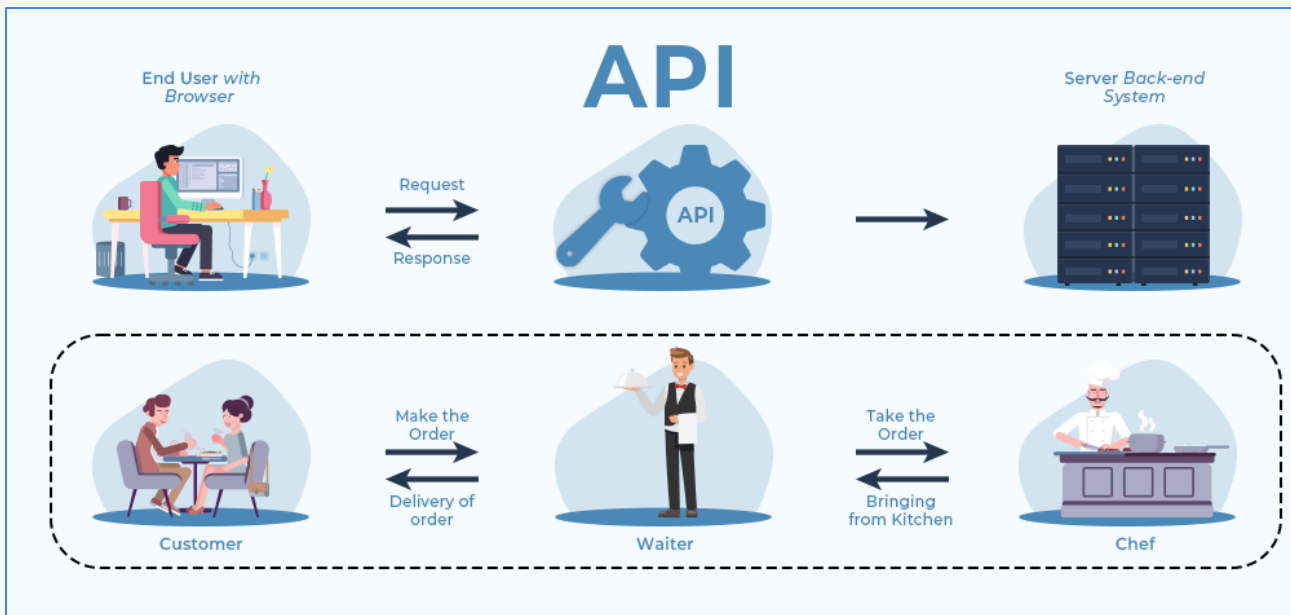
Like APIs themselves, API calls vary according to the specifications outlined in the API documentation. However, an API call follows three basic steps:

1. **The API client initiates the API call, or request for information.** The API client must format the request according to the protocol and schema provided by the API endpoint.
2. **The API endpoint receives the request.** The API endpoint then authenticates the API client and validates the API schema. This helps ensure that a) the call is coming from a verified source, and b) the conditions of the request have been met.
3. **The API endpoint returns the requested information to the API client.** The API schema determines the type of responses that may be returned to the client.

For a more in-depth explanation of API calls, read [What is an API call?](#)

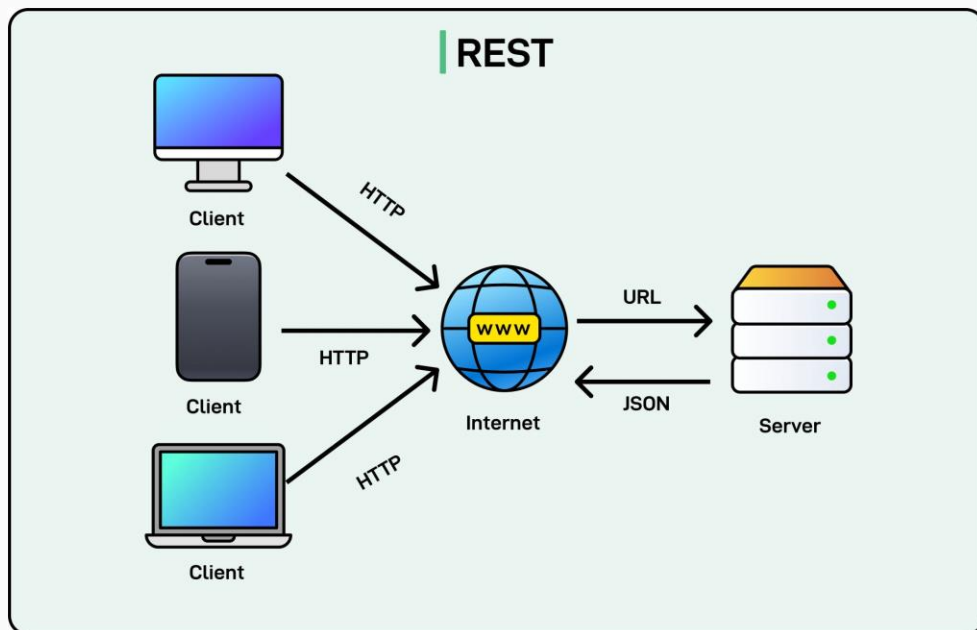
What Is an API (High-Level)?

- **API = Application Programming Interface.**
- Like a “contract” → defines how systems talk to each other.
- Used to integrate agents into apps, websites, workflows.



What Is an API Client?

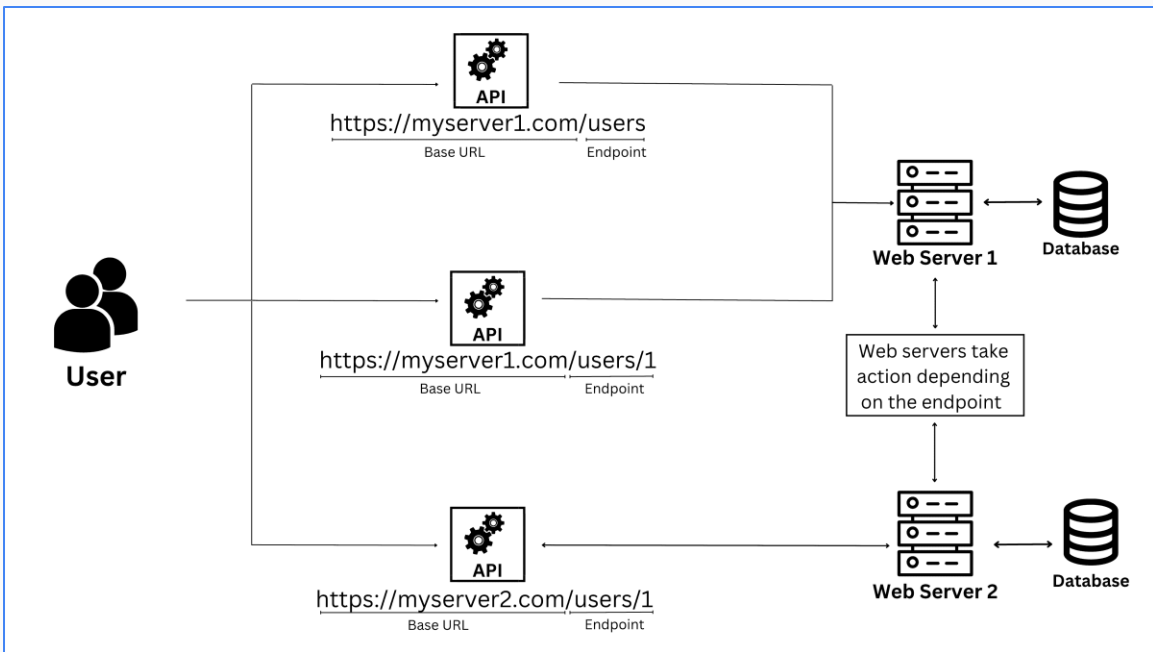
- An **API** defines rules about how two programs talk to each other.
- An **API client** is the program that follows those rules to send requests (like "give me user data") and process responses (like "here's the user data in JSON").



Example: when you open a weather app, the app is the client, it calls a weather API to fetch today's forecast.

What is an API endpoint?

- An **API endpoint** is a specific URL (address) inside the API where you can access or send data.
- It usually corresponds to a **resource** (like /users, /orders, /products).
- Each endpoint can support different **methods**:
 - **GET**: fetch data
 - **POST**: create new data
 - **PUT/PATCH**: update data
 - **DELETE**: remove data

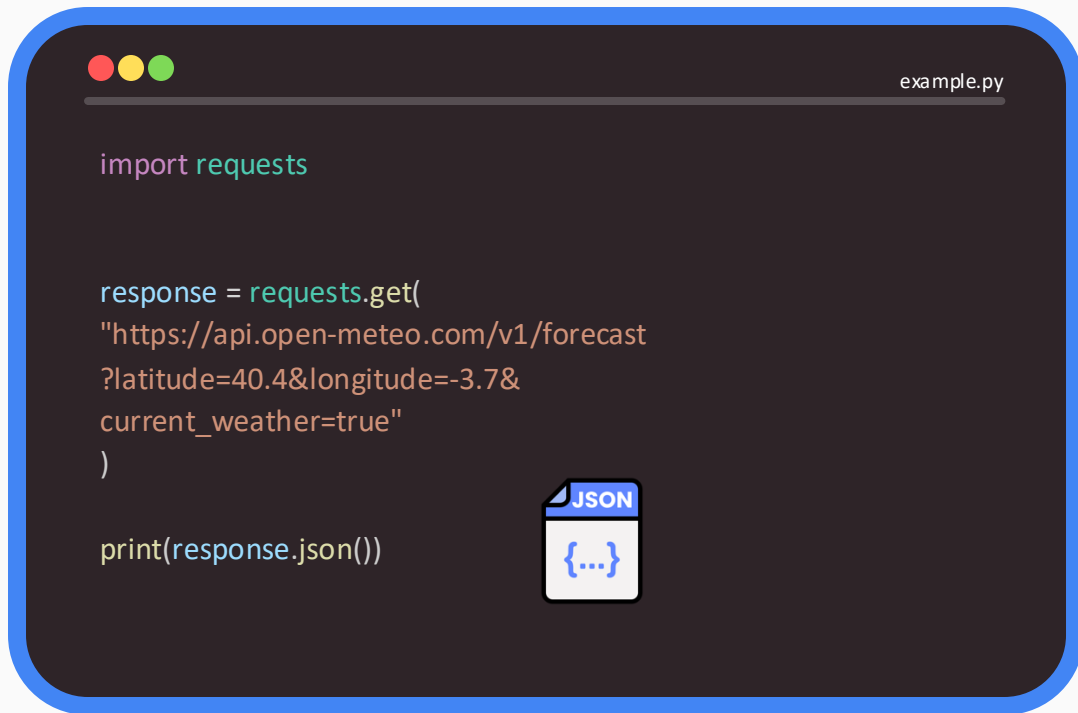


Example: /chat endpoint = system sends text → agent replies.

Example

In Python, we often use libraries like `requests` or `httplib` to act as an API client:

- The Python script = API client.
- The weather API = the server.
- The JSON result = the server's response.



```
import requests

response = requests.get(
    "https://api.open-meteo.com/v1/forecast
    ?latitude=40.4&longitude=-3.7&
    current_weather=true"
)

print(response.json())
```

Overview

- Lightweight framework for building APIs.
- Define endpoints (/, /chat).
- Comes with automatic docs (/docs).
- *Follow FASTAPI example:* <https://fastapi.tiangolo.com/pt/tutorial/>
- We will focus only on:
 1. First steps
 2. Path parameters
 3. Query parameters
 4. Request body

Class 5 fastapi-agent-service demo

Make sure you create a [virtual environment](#), activate it, and then **install FastAPI**:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The title bar of the window says "bash". The terminal shows the command `$ pip install "fastapi[standard]"` being executed. Below the command, a progress bar is shown, consisting of a series of small white squares, with the text "100%" at the end. In the bottom-right corner of the terminal window, the text "restart v" is visible.

```
bash

$ pip install "fastapi[standard]"

100%

restart v
```

The simplest FastAPI file could look like this:

Copy the example to file named 'main.py' and run the server with "fastapi dev main.py". Use the provided API Server URL to GET the "Hello World" message.

```
example.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

```
bash

$ fastapi dev main.py

FastAPI Starting development server 🚀
  Searching for package file structure from directories
  with __init__.py files
  Importing from /home/user/code/awesomeapp

module 🐍 main.py
  code Importing the FastAPI app object from the module with
  the following code:
  from main import app

  app Using import string: main:app

server Server started at http://127.0.0.1:8000
server Documentation at http://127.0.0.1:8000/docs

tip Running in development mode, for production use:
fastapi run

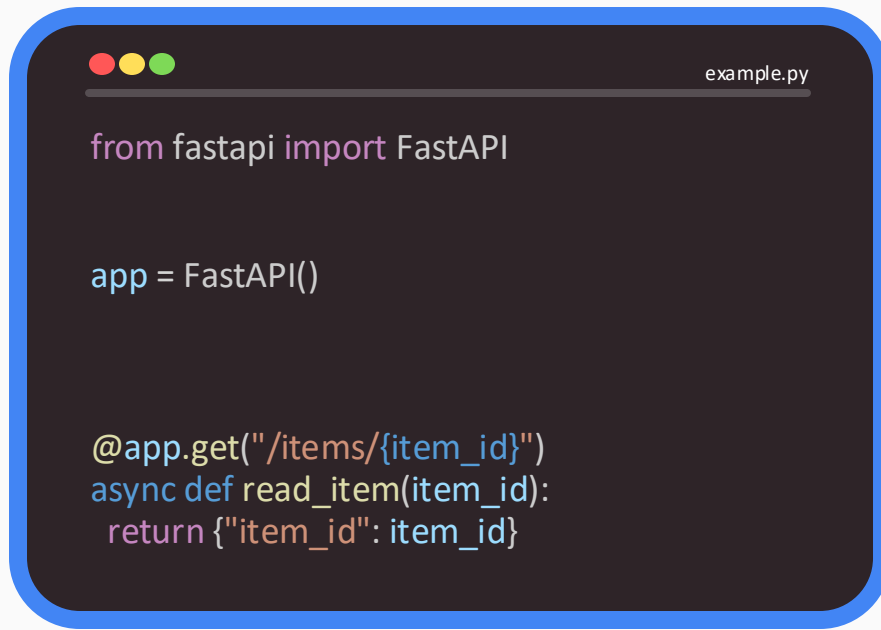
Logs:

INFO Will watch for changes in these directories:
['/home/user/code/awesomeapp']
INFO Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C
to quit)
INFO Started reloader process [383138] using WatchFiles
INFO Started server process [383153]
INFO Waiting for application startup.
INFO Application startup complete.

restart ↺
```

Connecting to the right endpoint

You can declare path "parameters" or "variables" with the same syntax used by Python format strings:



```
from fastapi import FastAPI

app = FastAPI()

@app.get("/items/{item_id}")
async def read_item(item_id):
    return {"item_id": item_id}
```

The value of the path parameter `item_id` will be passed to your function as the argument `item_id`.

Sending data in the path without being on the parameters

When you declare other function parameters that are not part of the path parameters, they are automatically interpreted as "query" parameters.



```
example.py

from fastapi import FastAPI

app = FastAPI()

fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]

@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return fake_items_db[skip : skip + limit]
```

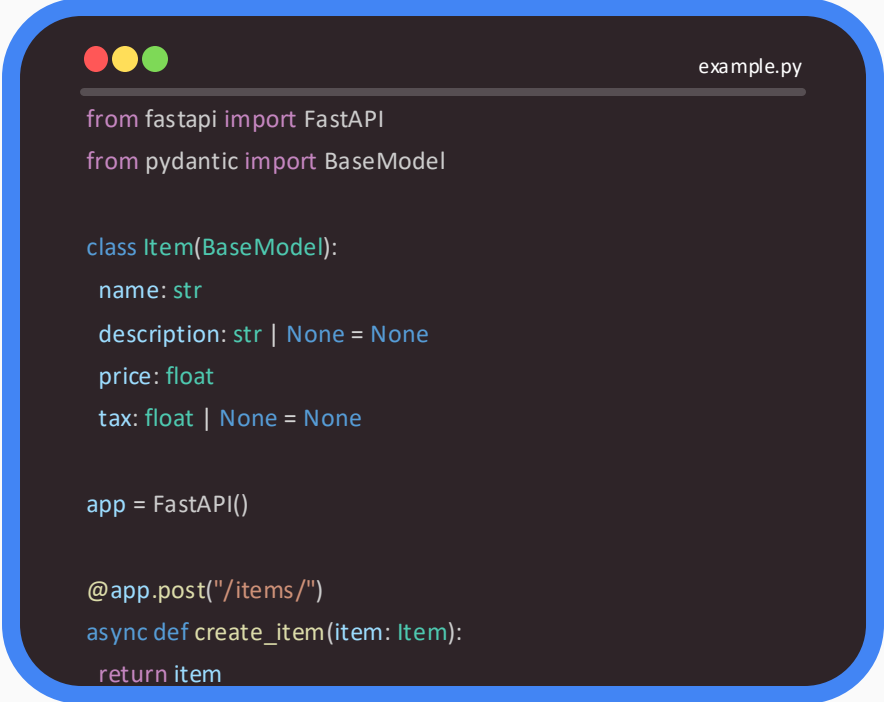
The query is the set of key-value pairs that go after the ? in a URL, separated by & characters.

Sending data in the path without being on the parameters

- When you need to send data from a client (let's say, a browser) to your API, you send it as a **request body**.
- A **request** body is data sent by the client to your API. A **response** body is the data your API sends to the client.
- Your API almost always has to send a **response** body. But clients don't necessarily need to send **request bodies** all the time, sometimes they only request a path, maybe with some query parameters, but don't send a body.

For Body requests we need to use Pydantic's BaseModel

First, you need to import BaseModel from [Pydantic](#) and then you declare your data model as a class that inherits from BaseModel.



```
from fastapi import FastAPI
from pydantic import BaseModel

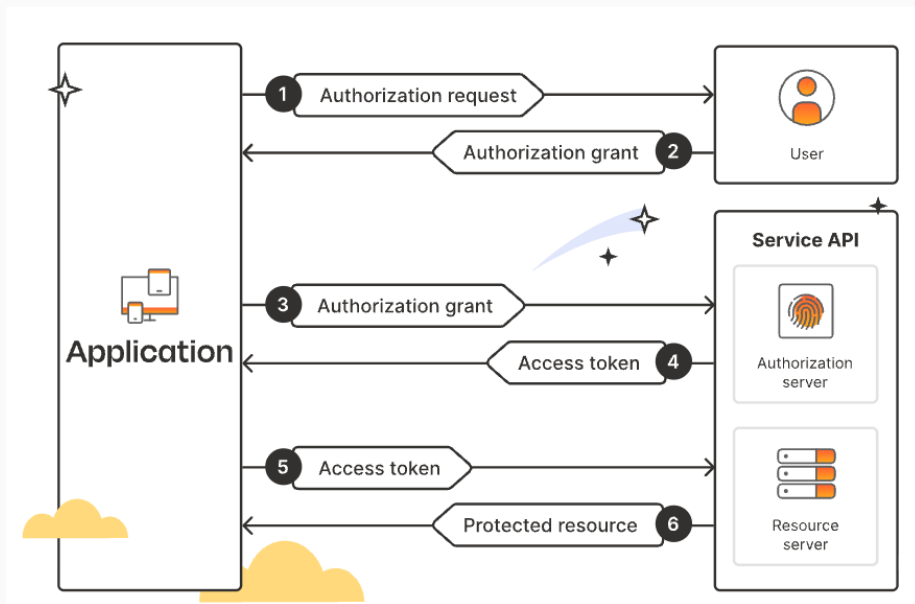
class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None

app = FastAPI()

@app.post("/items/")
async def create_item(item: Item):
    return item
```

It is important that we add security to our applications

[OAuth](#) is a token-based authentication mechanism that enables a user to grant third-party access to their account without having to share their login credentials. OAuth 2.0, which provides greater flexibility and scalability than OAuth 1.0, has become the gold standard for API authentication, and it supports extensive [API integration](#) without putting user data at risk.



FastAPI & Render demo



Today we will deploy an API on Render

The screenshot shows a GitHub repository named 'buidling-deploying-agents-applications' with the path '/classes/class-05-deployment-interfaces/demos/fastapi-agent-service/'. The file tree includes files like .env.example, README.md, demo_client.py, main.py, and requirements.txt. The README.md file is open, showing the title 'FastAPI Agent Service' and a description: 'A small API that powers the Streamlit demo. The entire project fits in this folder so you can see how UI, API, and monitoring connect. The /chat endpoint now uses a LangGraph ReAct agent with two simple tools so you can show students how agents access helper functions.' It also includes a section 'What you need first' with a bullet point: '• Python 3.10 or newer installed on your machine'.

The screenshot shows the Render dashboard. The left sidebar contains navigation links for Projects, Blueprints, Environment Groups, Integrations (Observability, Webhooks, Notifications), Networking (Private Links), Workspace (Billing, Settings), and a New button. The main content area is titled 'Overview' and features a 'Get organized with Projects' section with a '+ Create your first project' button. Below this is the 'Ungrouped Services' section, which displays a table of services. The table has columns for Service Name, Status, Runtime, Region, and Deployed. One service is listed: 'agent-api-render-test', which is 'Suspended by you', running on 'Python 3' in the 'Frankfurt' region, and was deployed '19d' ago.

SERVICE NAME	STATUS	RUNTIME	REGION	DEPLOYED
agent-api-render-test	Suspended by you	Python 3	Frankfurt	19d



<https://github.com/tutai-diaz/buidling-deploying-agents-applications/tree/main/classes/class-05-deployment-interfaces/demos/fastapi-agent-service>

Class 5 fastapi-agent-service demo

1. Use the provided OpenAI API Key so you can use GPT models.
2. Create a new Github repo and follow the step-by-step instructions defined on the README.md
3. Create a new **Web Service** on Render and give Render permission to access your new repo.
4. Provide the following values during service creation:

SETTING	VALUE
Language	Python 3
Build Command	pip install -r requirements.txt
Start Command	uvicorn main:app --host 0.0.0.0 --port \$PORT

Today FastAPI demo was just a quick intro. I advise to go through a dedicated tutorial of FastAPI!

Official documentation:

- [FastAPI Documentation](#) - Full usage details
- [FastAPI GitHub Repository](#) - FastAPI source code

Tutorials:

- [FastAPI Community](#) - Official FastAPI community
- [User Guide](#) - FastAPI step by step tutorial covering all features

FastAPI examples:

- [GitHub Example Repositories](#) - open-source examples

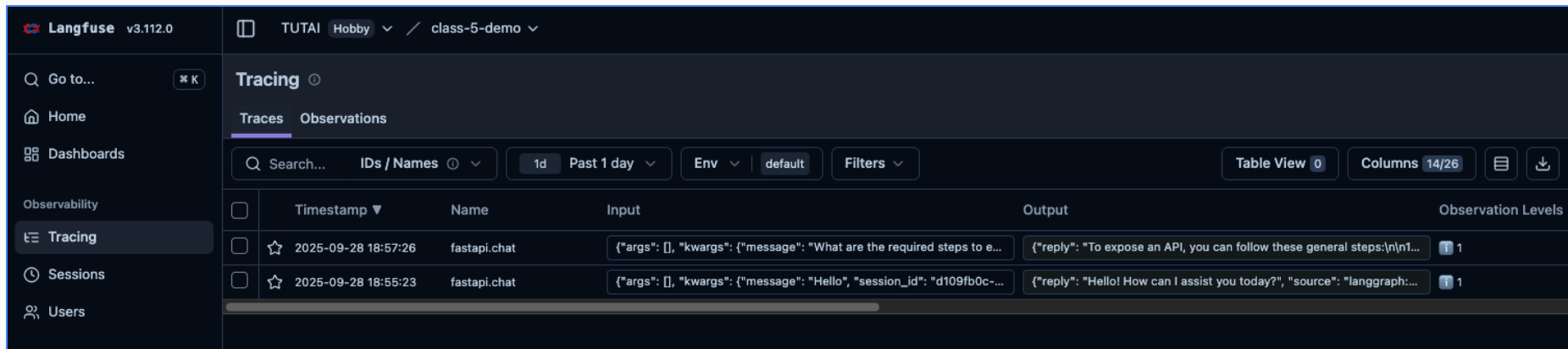
Langfuse

Adding observability to our AI Agents



Let's use Langfuse to monitor our AI Agents applications

Monitoring an AI Agent is key to understand it's usage, debugging and improve the developed agent overtime.



Langfuse v3.112.0

TUTAI Hobby / class-5-demo

Tracing

Traces Observations

Search... IDs / Names 1d Past 1 day Env default Filters Table View 0 Columns 14/26

	Timestamp ▼	Name	Input	Output	Observation Levels
<input type="checkbox"/>	☆ 2025-09-28 18:57:26	fastapi.chat	{"args": [], "kwargs": {"message": "What are the required steps to e..."}}	{"reply": "To expose an API, you can follow these general steps:\n1..."}	1
<input type="checkbox"/>	☆ 2025-09-28 18:55:23	fastapi.chat	{"args": [], "kwargs": {"message": "Hello", "session_id": "d109fb0c-..."}}	{"reply": "Hello! How can I assist you today?", "source": "langgraph:..."}	1

Let's use the Langfuse CallbackHandler() to monitor the LLM calls

We will focus mainly on tracing all calls to the AI Agent

```
def run_agent(message: str) -> Optional[str]:
    """Ask the LangGraph agent for a reply; return None if unavailable."""

    # Initialize Langfuse CallbackHandler for Langchain (tracing)
    langfuse_handler = CallbackHandler()

    if agent_runner is None:
        return None

    try:
        result = agent_runner.invoke({"messages": [("user", message)]}, config={"callbacks": [langfuse_handler]})
    except Exception as exc: # fall back to rule-based helper on errors
        print(f"[LangGraph] agent invocation failed: {exc}")
        return None

    messages = result.get("messages") if isinstance(result, dict) else None
    if not messages:
        return None

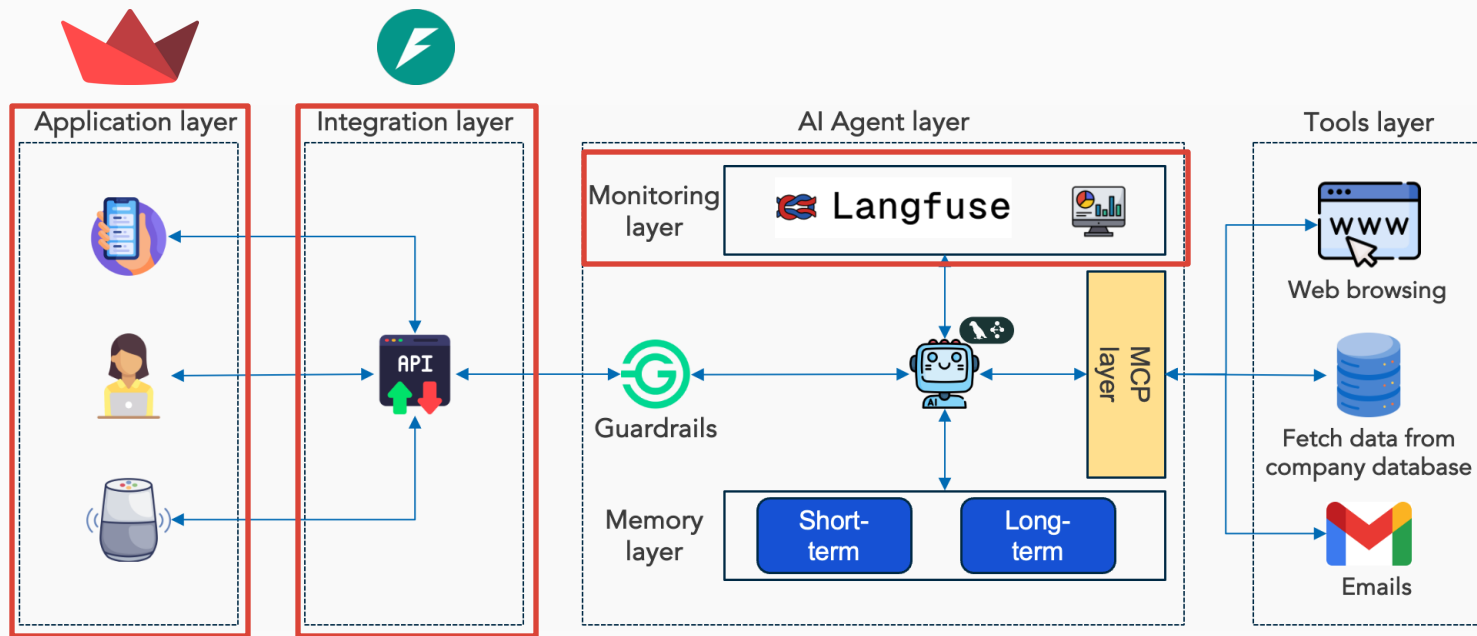
    last_message = messages[-1]
    content = getattr(last_message, "content", last_message)
    return _content_to_text(content)
```

Let's piece everything



Review the final flow

- User → Streamlit UI → FastAPI API → Agent → Langfuse monitoring.



- What new challenges arise when moving an agent from a notebook or prototype into production?
- Which of these layers do you consider most critical for your capstone project, and why?
- How might your capstone project benefit from having a Streamlit UI?
- What is the role of API endpoints and schemas in ensuring reliable communication between the UI and the AI Agent?
- What technical or conceptual gaps do you still need to address before deploying your agent?

Practice Practice Practice



1. Clone the classes repo and use Class 5 demo as the foundation.
2. Set up the FastAPI service.
3. Add the Tavily search tool
4. Prime the agent for the class 5 context.
5. Expose locally the API with monitoring enabled.
6. Update the Streamlit UI.
7. Deploy the API + UI.
8. Document your work.
9. Submit your work by email the:
 - Source code of the assignment.
 - README.md with sample Q&A transcript(s).
 - The Streamlit UI URL.
 - Confirmation (use screenshots or a video) that Tavily, LangGraph, and Langfuse all ran during testing.

Wrap-up



You've deployed an AI agent with

- Streamlit to the cloud.
- Developed an API with FastAPI to integrate with the Streamlit UI.
- Deployed the API to render in order to make it available over the public internet.
- You integrated monitoring with Langfuse in order to assess, debug and improve the agent.

Next: build a simple API and UI to start interacting with the AI Agent your are developing for the Medium article.



Questions?



luiz.f.s.m.dias@gmail.com



<https://www.linkedin.com/in/luisfilipedias/>