# Backend Development Workshop: Generate-a-Joke Application

## 1 Introduction

Welcome to the Backend Development Workshop. In this session, we will build a simple "generate-a-joke" application using FastAPI and Python. This workshop is designed for beginners and will guide you through setting up your environment, understanding APIs, and working with endpoints.

### 1.1 Setup Steps

Before we begin, ensure you have the following setup on your machine:

1. **Install Python:** Download and install Python from `https://www.python.org/downloads/` or follow this tutorial `https://www.datacamp.com/blog/how-to-install-python?dc_referrer=https%3A%2F%2Fwww.google.com%2F`.
2. **Code Editor** Use a code editor of your choosing. If you don't have one installed I would recommend VSCode: `https://code.visualstudio.com/docs/setup/setup-overview`
3. **Install FastAPI:** Use pip3 to install FastAPI by running the command: `pip install "fastapi[standard]"`.

## 2 API's

A REST API (Representational State Transfer Application Programming Interface) is a set of rules that allows programs to communicate with each other over the internet. It uses standard HTTP methods to interact with web resources:

- **GET**: Retrieves data from the server. It is used to request data from a specified resource and should not have any side effects, meaning it does not change the state of the resource. For example, using GET, you might retrieve a list of jokes from a jokes database.
- **POST**: Submits new data to the server. It is typically used to create a new resource. For example, you might use POST to add a new joke to the database. POST requests often include data in the body of the request, which is processed by the server to create the resource.
- **DELETE**: Removes existing data from the server. It is used to delete a specified resource. For example, using DELETE, you might remove a specific joke from the database by its ID.

REST APIs are stateless, meaning each request from a client contains all the information needed to process it, without relying on stored context. They often return data in formats like JSON or XML, which are easy for both humans and machines to read. REST APIs enable the integration of different software systems, making it easy to access data and services across the web.

## 3 Understanding the Skeleton Code

Here is a link to the backend skeleton code to start with :XXX

Let's go through the skeleton code for our "generate-a-joke" application. This code establishes the foundation for building our API using FastAPI.

```python
from fastapi import FastAPI, HTTPException
from typing import List
from pydantic import BaseModel
import random

app = FastAPI()

# Define a joke model
class Joke(BaseModel):
    id: int
    content: str

# In-memory storage for jokes
jokes: List[Joke] = [
    Joke(id=1, content="Joke A"),
    Joke(id=2, content="Joke B"),
    Joke(id=3, content="Joke C")
]

# GET endpoint to retrieve a random joke
@app.get("/api/joke", response_model=Joke)
async def get_joke():
    # Implementation here
    return joke

# POST endpoint to add a new joke
@app.post("/api/add-joke", response_model=Joke)
async def add_joke(joke: str):
    # Implementation here
    return new_joke

# DELETE endpoint to delete a joke by ID
@app.delete("/api/delete-joke/{joke_id}", response_model=Joke)
async def delete_joke(joke_id: int):
    # Implementation here
    return joke
```

## 3.1 Code Explanation

### 3.1.1 Imports

The code begins with importing necessary modules:

- `FastAPI, HTTPException`: These are core components from FastAPI used to create the application and handle HTTP exceptions.
- `List`: A type hint from the `typing` module to specify a list of jokes.
- `BaseModel`: From the `pydantic` library, used to define data models for request and response validation.
- `random`: A standard Python library used for selecting random items.

### 3.1.2 Application Initialization

The FastAPI application is initialized with:

```
app = FastAPI()
```

### 3.1.3 Data Model Definition

A data model named `Joke` is defined using `BaseModel`. This model specifies the structure of a joke with two fields: `id` and `content`.

### 3.1.4 In-Memory Storage

An in-memory list named `jokes` is created to store instances of the `Joke` model. This list is initialized with three jokes.

### 3.1.5 Endpoints

The code includes three API endpoints:

- **GET /api/joke:** Retrieves a random joke from the list. It returns a `Joke` object.
- **POST /api/add-joke:** Accepts a joke as input and adds it to the list. It also returns the newly added `Joke` object.
- **DELETE /api/delete-joke/{joke_id}:** Deletes a joke specified by its ID from the list and returns the deleted `Joke` object.

# 4 Running the code

Paste the skeleton code into a file within the code editor and call the file **main.py** If step 1 has been done successfully you should be able to run this command in the code editor terminal: **fastapi dev main.py**

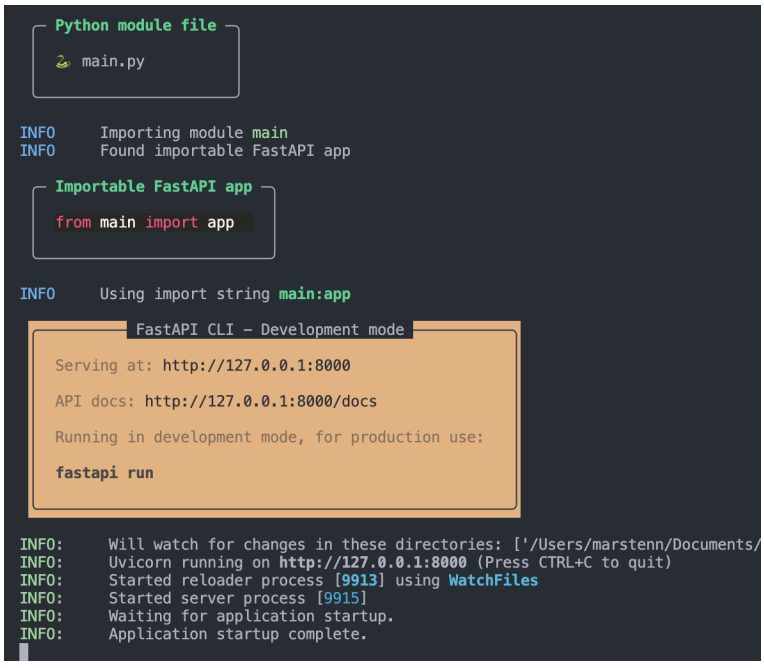


Figure 1: Fast API has started the application

Click on the API docs link and this is what you will see:

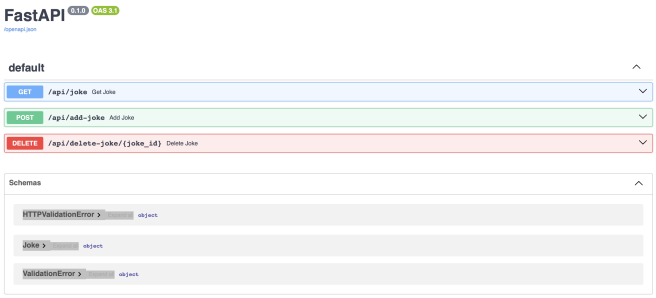


Figure 2: Application Swagger

### 4.1 Understanding Swagger

Swagger is an interface that allows you to interact with your API endpoints. FastAPI automatically generates a Swagger UI to test the endpoints. It can be accessed at `/docs` after starting your FastAPI application.

## 5 Tasks

### 5.1 Task 1: Get a Random Joke

Work on the code in the GET joke endpoint to make it functional. It should return back a random Joke.

```
# GET endpoint to retrieve a random joke
@app.get("/api/joke", response_model=Joke)
async def get_joke():
    # Your code here
```

### 5.2 Task 2: Add a Joke

Add code into this endpoint to add a joke and generate a unique ID for it.

```
# POST endpoint to add a new joke
@app.post("/api/add-joke", response_model=Joke)
async def add_joke(joke: str):
    # Your code here
```

### 5.3 Task 3: Delete a Joke by ID

Implement an endpoint to delete a joke by its unique ID.

```
# DELETE endpoint to delete a joke by ID
@app.delete("/api/delete-joke/{joke_id}", response_model=Joke)
async def delete_joke(joke_id: int):
    # Your code here
```

### 5.4 Task 4: Add Error Handling

Error handling is the process of anticipating and managing errors in a program to ensure it runs smoothly or fails gracefully. It improves user experience by providing meaningful error messages and prevents the program from crashing unexpectedly.

Add error handling to manage situations such as:

- No jokes in the list

- Deleting a non-existent ID

This can be done using **HTTPException**

```
@app.get("/joke")
async def get_random_joke():
    if not jokes:
         # Your code to handle this error
    ...

@app.delete("/joke/{joke_id}")
async def delete_joke(joke_id: int):
    ...
    if not joke:
        # Your code to handle this error
    ...
```

### 5.5 STRETCH Task 5: Add in a Database

As a stretch goal, enhance your simple job API by replacing the in-memory list storage of jokes with a database solution. This involves:

- Setting up a SQLite database to store jokes.
- Modifying the API endpoints to interact with the database instead of the list.
- Ensuring that all CRUD operations (Create, Read, Update, Delete) are supported using database queries.
- Testing the API to verify that all operations work correctly with the database.

This task will help you understand how to integrate persistent storage in your applications, preparing you for real-world scenarios where data needs to be stored and retrieved efficiently.

## 6 Solutions

Solutions for Task 1-4 are present here: `https://github.com/Lm0079/Coding-your-MVP---Intro-to-Back-End-Material/tree/main/solutions/task1-4`

Solutions for Task 5 are present here: `https://github.com/Lm0079/Coding-your-MVP---Intro-to-Back-End-Materia tree/main/solutions/stretch`

## 7 Conclusion

Congratulations on completing the workshop! You have now built a simple backend application using FastAPI. Explore further by adding more features or improving the existing functionality.