A Deep Dive into Containerized Model Serving with FastAPI

PyLadies Amsterdam
June 28, 2022
Nikki van Ommeren
Karlijn Schipper

Agenda

Who are we?

The case: A Deep Dive into Containerized Model Serving with FastAPI

Who are we?

Nikki van Ommeren

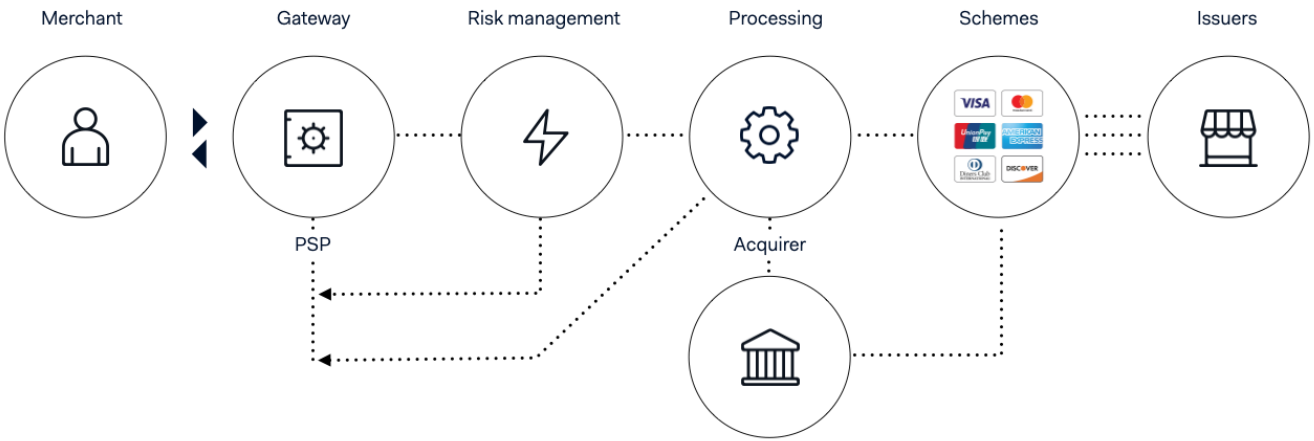
Karlijn Schipper

Nikki van Ommeren

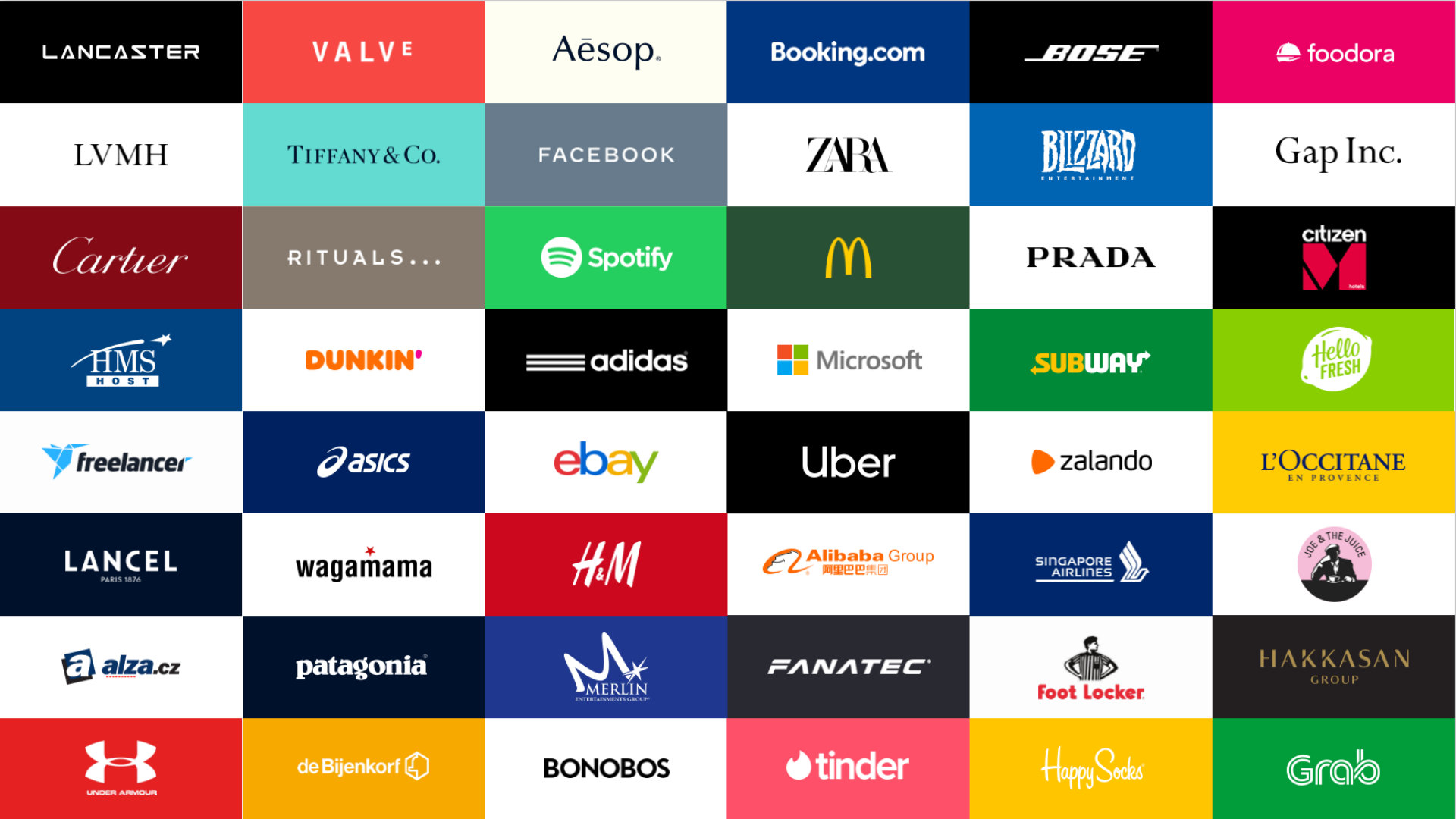
- Data scientist at Adyen
- Enjoys board games, running, cycling

Legacy technology

Traditional value chain



Legacy technology


LANCASTER
VALVE
Aēsop
Booking.com
BOSE
foodora
LVMH
TIFFANY & CO.
FACEBOOK
ZARA
BLIZZARD ENTERTAINMENT
Gap Inc.
Cartier
RITUALS...
Spotify
PRADA
citizen
HMS HOST
DUNKIN'
adidas
Microsoft
SUBWAY
Hello FRESH
freelancer
asics
ebay
Uber
zalando
L'OCCITANE EN PROVENCE
LANCEL PARIS 1876
wagamama
H&M
Alibaba Group 阿里巴巴集团
SINGAPORE AIRLINES
JOE & THE JUICE
alza.cz
patagonia
MERLIN ENTERTAINMENTS GROUP
FANATEC
Foot Locker
HAKKASAN GROUP
UNDER ARMOUR
de Bijenkorf
BONOBOS
tinder
Happy Socks
Grab

Karlijn Schipper

Machine Learning Engineer @ VANTAGE AI

Vantage AI delivers hands-on customisation in the field of ML and AI to solve business problems

- 2-year training program
- 6-12 months assignments

- After this program, we transition to bigdata republic (BDR)
- Added value of BDR:
  - Training
  - Mentorship program
  - Knowledge sessions

Fun stuff

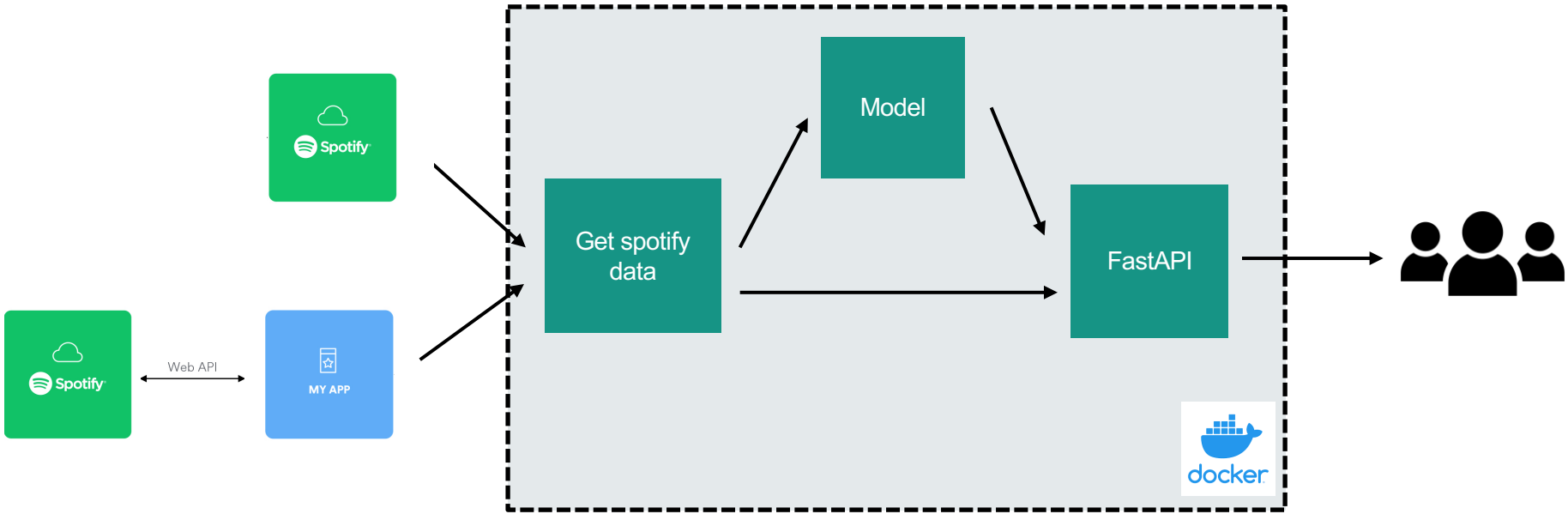
Questions? Find me, or Vantage AI on LinkedIn via this QR code:

Agenda

Who are we?

The case: A Deep Dive into Containerized Model Serving with FastAPI

The goal of this workshop is to create an end-to-end Machine Learning solution that recommends Spotify songs



Case overview

- Part 0 – Preparation
- Part 1 - Create Dockerfile and bash script
- Part 2: Create FastAPI
- Part 3: Requests, Responses, Error handling, Pydantic

Part 0 – Preparation

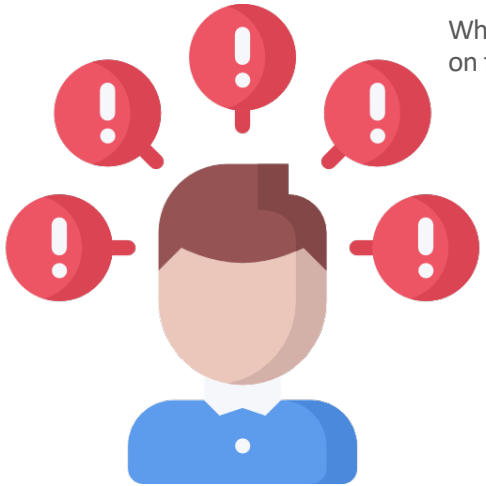
- Create an .env file in the workshop folder
- Fill this .env file in with your Spotify credentials
- You can get these credentials via yourown app (https://developer.spotify.com/dashboard/applications)

Multiple problems can arise if you want to deploy your model

Where to put files on the server?

Different dependencies for the server OS are installed

What other processes etc. are running on the server during your installation?



Different OSs are used when developing the product

What happens when the server changes?

Meet Docker

- Docker is a framework for *containerization*
- Apps run within a virtual container, separated from other apps
- Docker containers run the same on **any** host machine: they have their own virtual OS installed
- Enables microservice architectures: lots of different containers with simple tasks talking through each other through API's

Main benefits of Docker

- Containers run the same on any machine
- Easy to use
- Quickly build and test applications
- Lightweight and quick: very little overhead
- Enables better scalability
- Tools like **Kubernetes** can be used to manage containers in production environments

Dockerfile commands

- **FROM <image:tag>**
  - Select base image
- **RUN <command>**
  - Execute shell command
- **COPY <from> <to>**
  - Copy a file from a local directory to the image
- **ENV <key=value>**
  - Set environment value in the image
- **ARG <key=value>**
  - Set argument in the image
  - `docker build --build-arg key=value`
- **WORKDIR <dir>**
  - Change the current working directory
- **USER <user id>**
  - Change the current user
- **EXPOSE <port>**
  - Expose a port to the docker network
- **ENTRYPOINT [<command>]**
  - configures the container to run as an executable, will always run
- **CMD [<command>]**
  - sets default command and/or parameters, which can be overwritten from the command line

Docker commands

- docker build –t <tag_name> .
- docker run –p 8080:8080 –v <directory>:<directory> <tag_name> bash –c “*command*”
- docker tag <tag_name> <dockerhub_repo>
- docker image push <dockerhub_repo>

Part 1: Create a containerized solution that prints your most listened song

- Complete the Dockerfile
- Complete the bash file that builds a Docker image and runs the container
- Run the bash file
- Authenticate with Spotify API

Solutions - Dockerfile

```
FROM python:3.10

WORKDIR /code

RUN apt-get update -y

COPY requirements.txt /code/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

COPY src /code/src
COPY data /code/data
COPY setup.py /code/setup.py
RUN pip install .
```

Solutions – run_endpoints.sh

```
docker build -t music_style_image .


docker run --env-file=.env -i -t
          -p 8080:8080 --name music_style
                    -v "/${PWD}"/src:/code/src
                              music_style_image bash -c "python src/spotify.py"
```

What is an API?

- Lets applications talk to each other
- You can use API's to:
  - Get model predictions
  - Get Spotify data
  - Send data to your SQL server
- Today we're making an API to *serve* a ML model

What is an API? (II)

- *Representational State Transfer Application Program Interface*
- Allows applications to talk to each other *over the internet* (with http)
- Different kinds of requests (API calls):
  - **GET** request: gets information from the API
  - **POST** requests: sends information and receives a result
  - **PUT/PATCH** are used for replacing or updating information
  - **DELETE**

Return codes

- REST API's work over http, and therefore use http status codes:

- 200 and above: Succesfull responses
  - 200: OK
  - 201: Created
- 300 and above: Redirection
- 400 and above: Client errors:
  - 400 bad request, most generic error
  - 401: Unauthorized
  - 404: Not found
- 500 and above: Server errors:
  - 500: Internal server error
  - 503: service unavailable, etc.


ME
FASTAPI
PERFORMANCE
FLASK
imgflip.com

FastAPI vs. Flask

- GET&POST requests are very similar

```python
import uvicorn

from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":

    uvicorn.run("main:app")
```

```python
from flask import Flask

app = Flask(__name__)

@app.get("/")
def home():
    return {"Hello": "World"}

if __name__ == "__main__":

    app.run()
```

FastAPI vs. Flask

FastAPI

- Automatic, interactive documentation
- In-house data validation with Pydantic
- Is asynchronous - no need to wait for other request to be completed. Flask is synchronous

Flask

- Proven and well-tested solution
- For web-development: Flask automatically looks for a templates folder where you can display your variables

Part 2: Create FastAPI

```
from fastapi import FastAPI
from src.spotify import MusicModel

music_model = MusicModel()

app = FastAPI(description=description, openapi_tags=tags_metadata)


@app.get("/")
def root():

    return {"message": f"Welcome to the Spotify Music app! "
                       f"{music_model.auth_msg}"}
```

Part 2: Create FastAPI

Object created in main.py

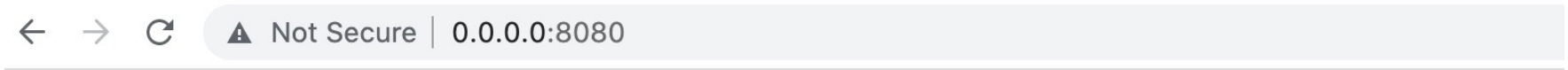
uvicorn src.main:app --host 0.0.0.0 --port 8080 --reload --debug

Refers to file main.py

Makes sure the server restarts after code changes

Part 2: Create FastAPI

```
INFO:     Will watch for changes in these directories: ['/code']
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
INFO:     Started reloader process [1] using WatchFiles
INFO:     Started server process [8]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     172.17.0.1:65272 - "GET / HTTP/1.1" 200 OK
```

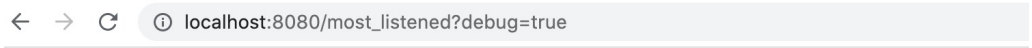


{"message":"Welcome to the Spotify Music app! Successfully connected to the Spotify API."}

Query parameters

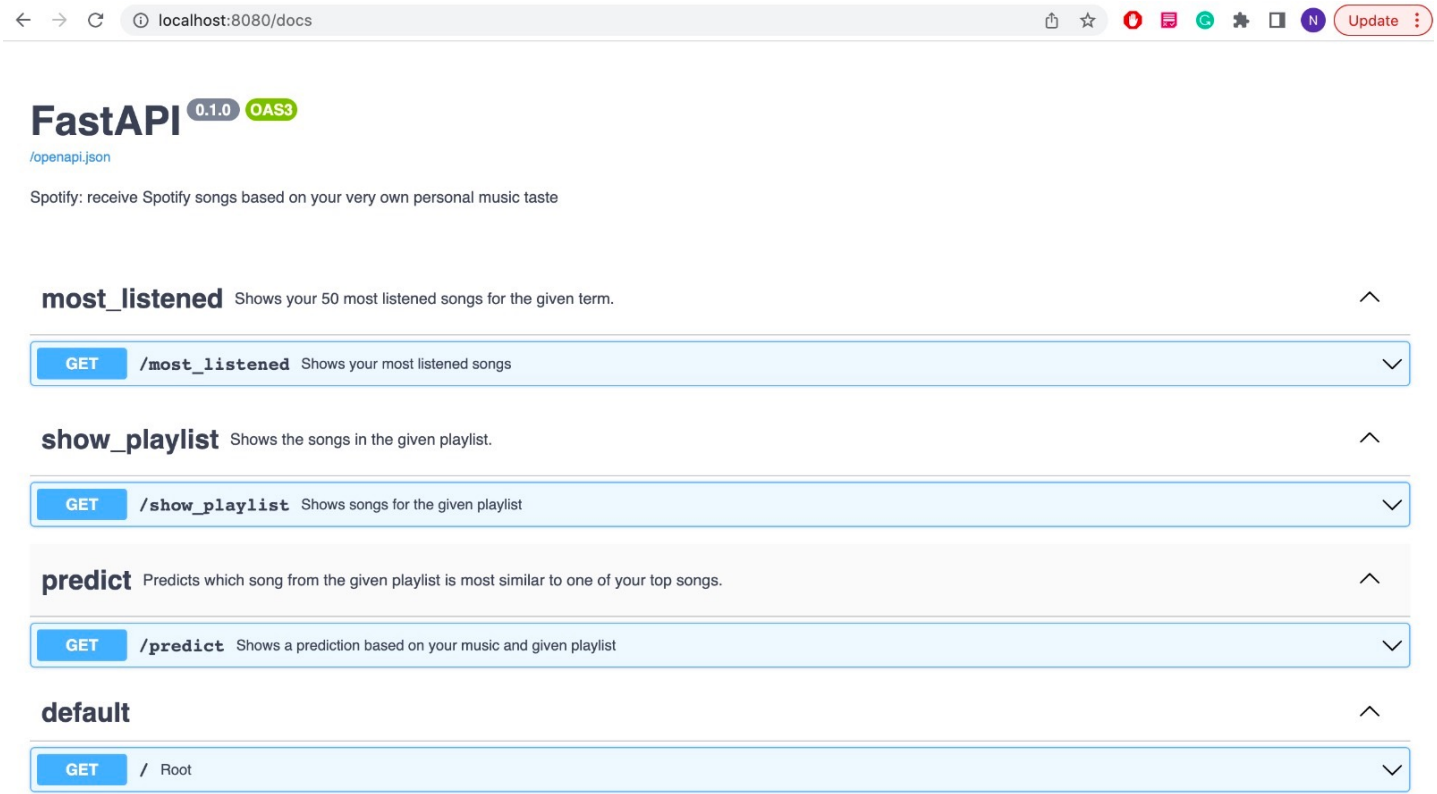
```python
@app.get(
    "/most_listened",
    tags=["most_listened"],
    summary="Shows your most listened songs",
    response_model=List[Song],
)
def get_most_listened_songs(term: Term = "short_term", debug: bool = False):
    """ """
    user_tracks = music_model.read_user_tracks(term)

    if debug:
        return HTMLResponse(
            content=user_tracks[["name", "artists"]].to_html(), status_code=200
        )
    return user_tracks[["name", "artists"]].to_dict(orient="records")
```

Query parameters



	name	artists
0	Schubert: Schwanengesang (Standchen) - Kate Simko Rework	Kate Simko, London Electronic Orchestra
1	Always Remember Us This Way	Lady Gaga
2	Come Online	Kid Francescoli, Julia Minkin
3	Strange Entity	Oscar and the Wolf
4	Onbezonnen	Froukje
5	Figure Out	Tess van der Velde
6	Adem Je In	S10
7	Zonder Gezicht	Froukje, S10
8	Kings	Native Young
9	I'm Gonna Leave You - The Cinematic Orchestra Remix	Melanie De Biasio, The Cinematic Orchestra
10	Tegen De Klippen Op	MEAU
11	Dogs - Oliver Koletzki Remix	HVOB
12	Want You In My Soul	Lovebirds, Stee Downes
13	Coming Back to You	Lane 8, J. F. July
14	Stir Me Up	Lane 8
15	She Knows	Jet Hammer
16	Giulia	Alberto Bof, Stephanie Gilmore
17	Desert Days	Sonny Ism
18	I Would Stay	Krezip
19	Pompidou	Portico Quartet

Interactive docs



Exercise 2

1. Get your most listened songs by calling ‘most_listened’ with different values for the variable query variable term.
2. Try the functions in the interactive docs
3. Add a ‘limit’ parameter to get_most_listened_songs, limiting the number of records it returns.

Solutions

Add a ‘limit’ parameter to get_most_listened_songs, limiting the number of records it returns:

```
@app.get(
    "/most_listened",
    tags=["most_listened"],
    summary="Shows your most listened songs",
    response_model=List[Song],
)
def get_most_listened_songs(term: Term = Query("short_term"), limit: int = 50, debug: bool = False):
    """ """
    user_tracks = music_model.read_user_tracks(term)[:limit]

    if debug:
        return HTMLResponse(
            content=user_tracks[["name", "artists"]].to_html(), status_code=200
        )
    return user_tracks[["name", "artists"]].to_dict(orient="records")
```

Part 3: Response model & Error handling

- A **request** body is sent from the client to the API. A **response** body is the data from your API to the client.
- For calls containing a **request** body, POST is the most common request.

- To declare a **request** or response body, you can use Pydantic; which comes pre-installed with FastAPI.

Response model

```python
from pydantic import BaseModel

class Song(BaseModel):
    name: str
    artists: str

@app.get(
    "/most_listened",
    response_model=List[Song],
)
def get_most_listened_songs(term: Term = Query("short_term"), limit: int = 50, debug: bool = False):
    """ """
    user_tracks = music_model.read_user_tracks(term)[:limit]

    if debug:
        return HTMLResponse(
            content=user_tracks[["name", "artists"]].to_html(), status_code=200
        )
    return user_tracks[["name", "artists"]].to_dict(orient="records")
```

Response model

Schemas

HTTPValidationError ›

Song ⌄ {

name*	string	
	title: Name	
artists*	string	
	title: Artists	

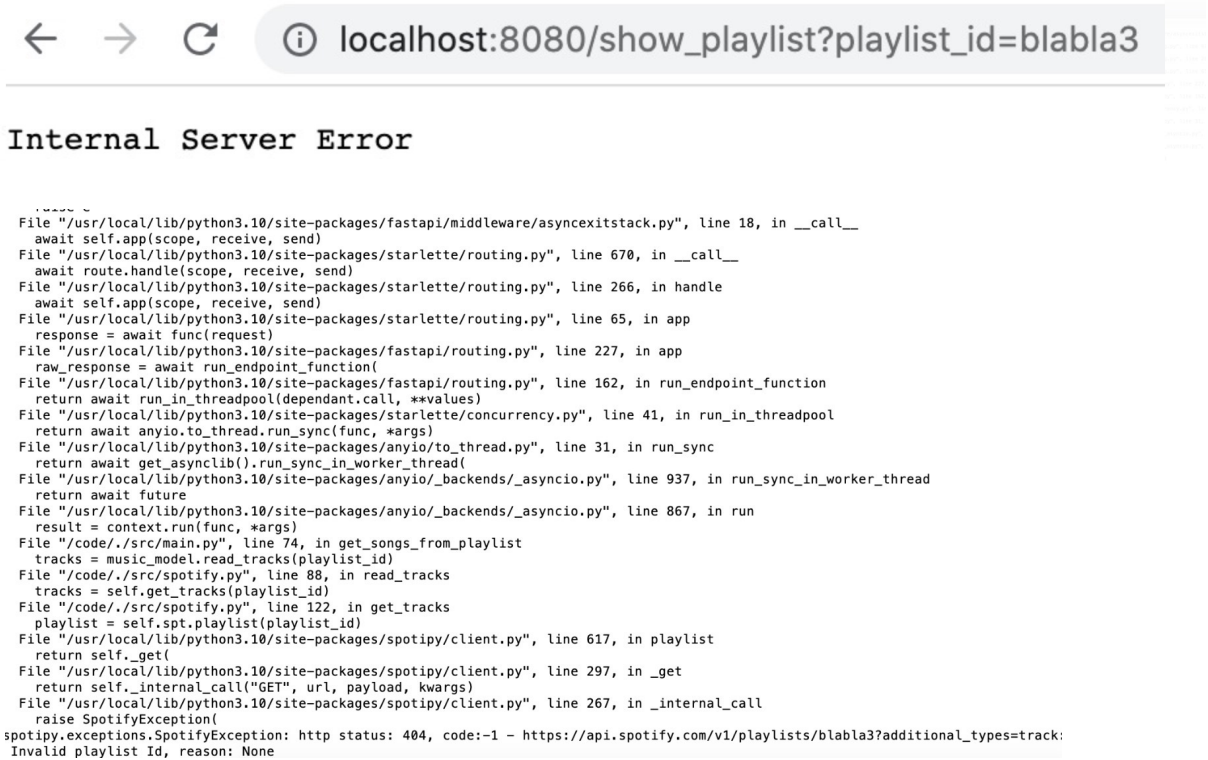
}

example: OrderedMap { "name": "De Diepte", "artists": "S10" }

Term ›

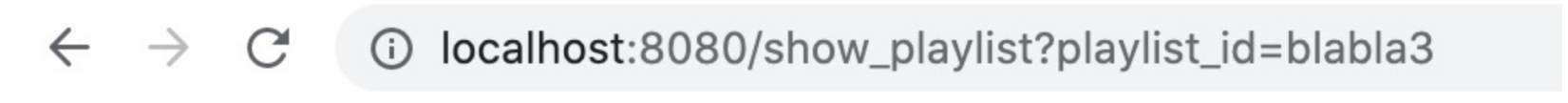
ValidationError ›

Error handling



Error handling

```
raise HTTPException(status_code=404, detail="Playlist id not found")
```




```
{"detail":"Playlist id not found"}
```

```
INFO:     172.17.0.1:64332 - "GET /show_playlist?playlist_id=blabla3 HTTP/1.1" 404 Not Found
```

Exercise 3

1. Create your own prediction function with term and playlist_id as input.
2. Create your own PyDantic class definition for the predicted output
3. Bonus: Throw an HTTP error when the user attempts to ask for a playlist when no Spotify credentials are provided

Looking for an extra challenge? Try adding a function that adds your recommended song to one of your own playlist. This function still needs to be added to the MusicModel class in ‘spotify.py’

Solutions

1. Create your own prediction function with term and playlist_id as input.

```
@app.get(
    "/predict",
    tags=["predict"],
    summary="Shows a prediction based on your music and given playlist",
    response_model=PredOut,
)
def get_prediction(
    term: Term = Query("short_term"),
    playlist_id: str = "37i9dQZF1DXb5BKLTO7ULa",
):

    return music_model.predict(term=term, playlist_id=playlist_id)
```

Solutions

Create your own PyDantic class definition for the predicted output

```
class PredOut(BaseModel):
    favourite_song: str
    most_similar_song: str
    distance: float

    class Config:
        schema_extra = {
            "example": {
                "favourite_song": "Always Remember Us This Way - Lady Gaga",
                "most_similar_song": "De Diepte - S10",
                "distance": 0.15,
            }
        }
```

Solutions

Bonus: Throw an HTTP error when the user attempts to ask for a playlist when no Spotify credentials are provided

```
def get_songs_from_playlist(
    playlist_id: str = "37i9dQZF1DXb5BKLTO7ULa",
    debug: bool = False,
):
    """ """
    try:
        tracks = music_model.read_tracks(playlist_id)
    except SpotifyException:
        raise HTTPException(status_code=404, detail="Playlist id not found")

    if debug:
        return HTMLResponse(
            content=tracks[["name", "artists"]].to_html(), status_code=200
        )

    return tracks[["name", "artists"]].to_dict(orient="records")
```

Recap

- How to use docker with a docker file
- FastAPI is great for type checking, interactive docs and easy API development
- How to use the Spotify Web API

- More questions? Connect to us on LinkedIn!

- https://www.linkedin.com/in/karlijn-schipper-889538111/
- https://www.linkedin.com/in/nikki-van-ommeren-43a806a2/

Questions?