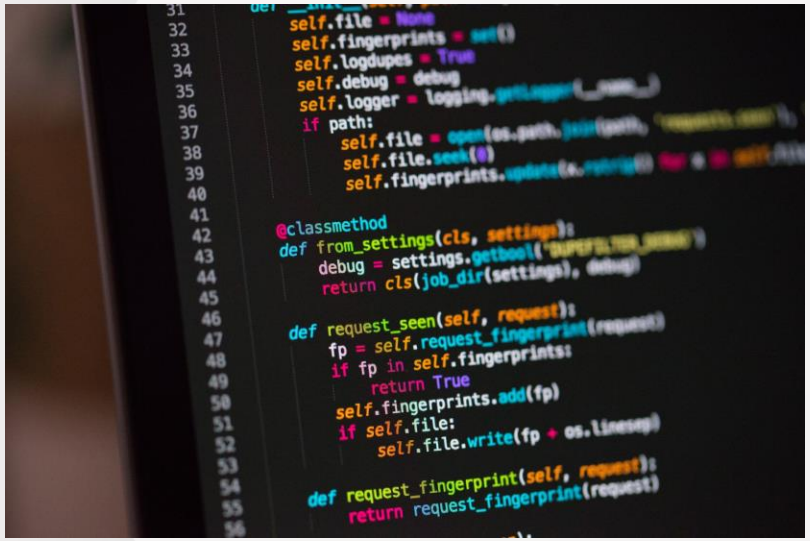
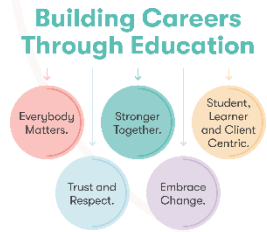
Level 5 Data Engineer
Module 6 Topic 3

APIs and Microservices for Data Engineers



L5 Data Engineer Higher Apprenticeship
Module 6 / 12 (“**Data Collection and Ingestion pt. 1**”)
Topic 3 / 4

Ice breaker: Discussion



A bit of fun to start…

- If you could automate any task in your daily routine using an API, what would it be and why?
- Imagine you had a microservice for any personal task. What would it do and how would it improve your life?
- How do you think microservices can benefit data engineering projects in terms of scalability and flexibility?

Submit your responses to the chat or turn on your microphone

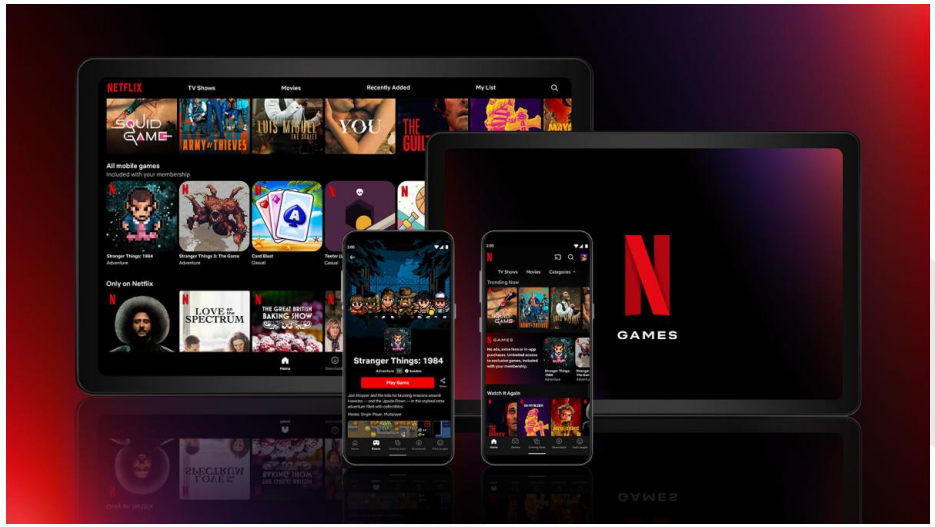
BPP

Case study

Netflix’s Use of Microservices and APIs

Overview:

- Netflix uses microservices and APIs to manage its vast streaming service.
- Each service (e.g., user authentication, content recommendation, streaming) operates independently but communicates through APIs.



Impact:

- Allows Netflix to scale individual services as needed.
- Enables deployment of updates without affecting the entire system.
- Improves fault tolerance.
- Enhances ability to handle high traffic volumes.
- Provides a seamless user experience.

Knowledge check poll

What is a primary benefit of using microservices architectures in data engineering?

A. It simplifies the data ingestion process.

B. It allows for real-time data processing.

C. It enables independent scaling and deployment of services.

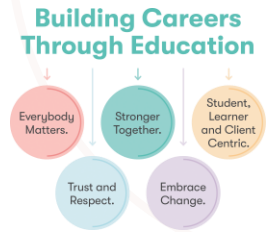
D. It reduces the need for data validation.

Submit your responses to the chat!

Feedback: C – It enables independent scaling and deployment of services.

BPP

Session aim and objectives



Completion of this topic supports the following outcomes:

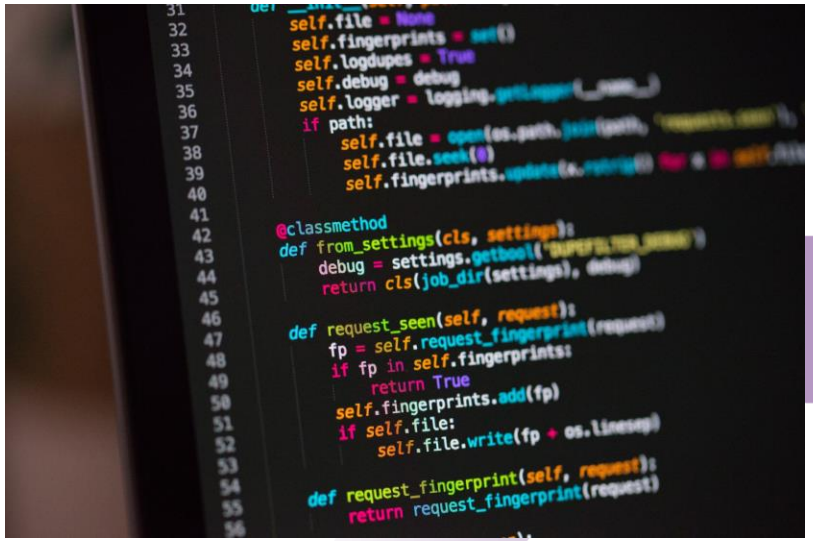
1. Explain the benefits of microservices architectures in data engineering

2. Demonstrate ability to use an API to ingest data

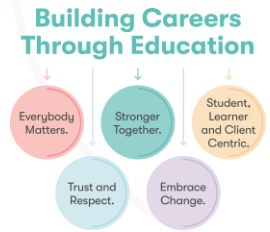
3. Design a data ingestion architecture using APIs and microservices

BPP

E-learning Recap



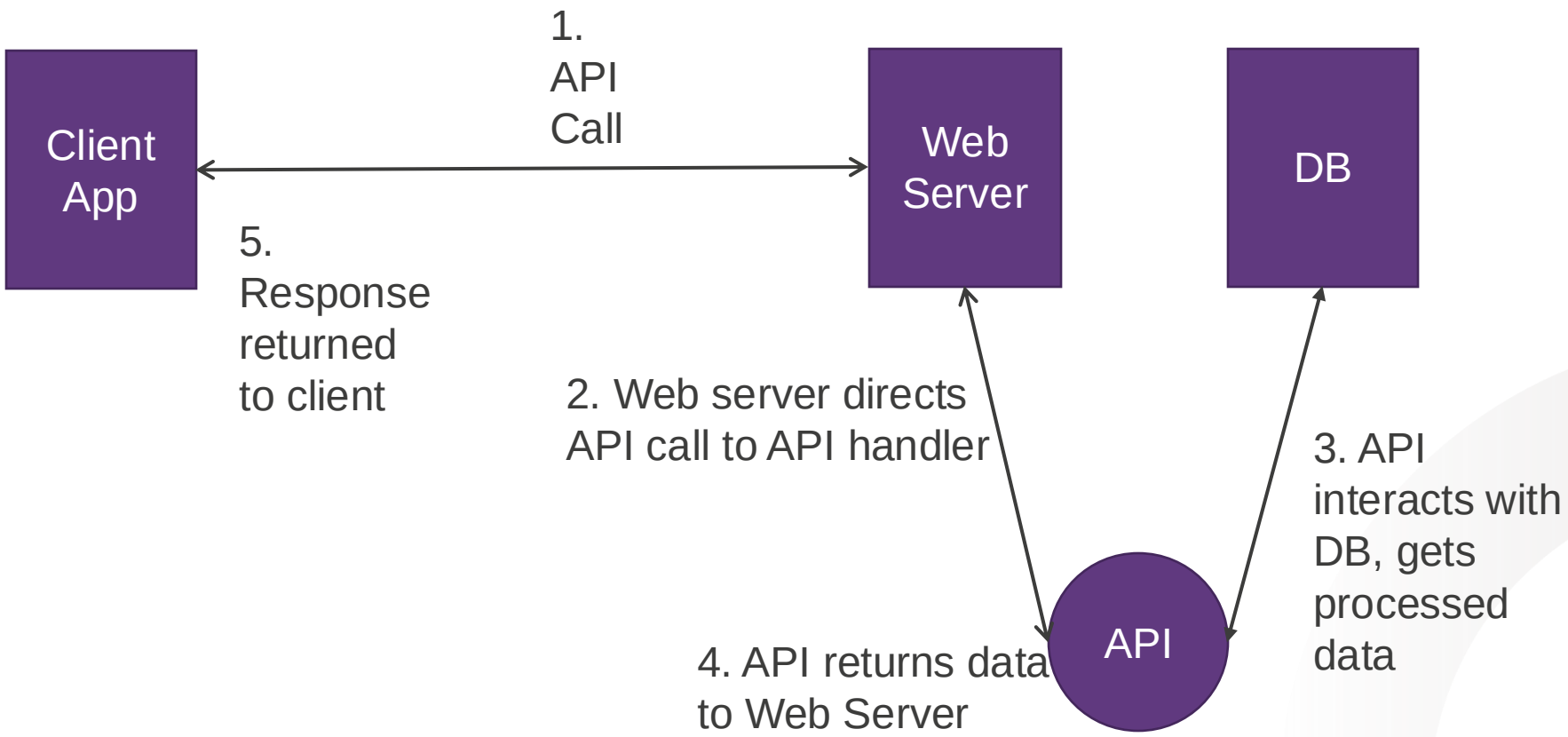
Recap discussion



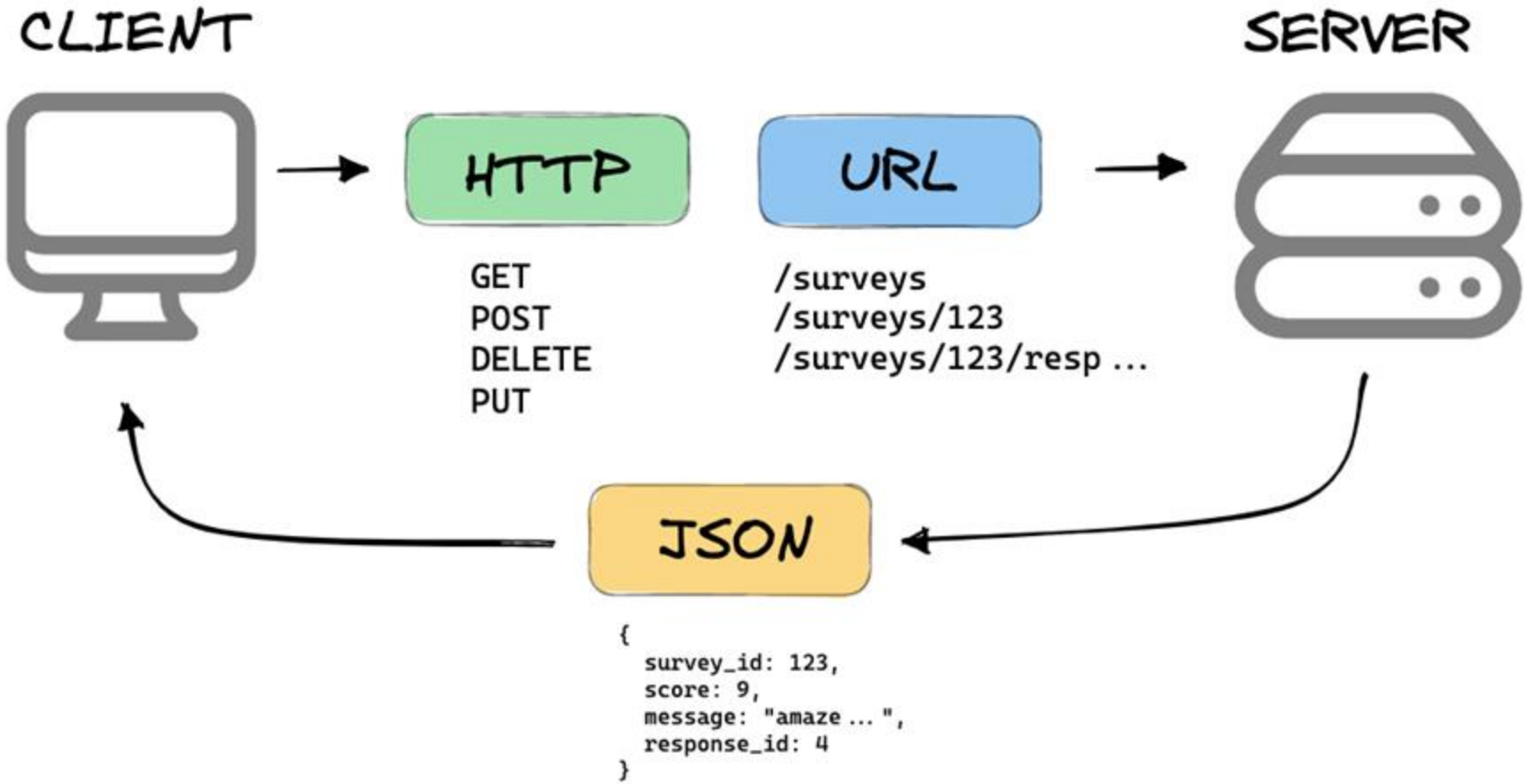
- Can you recall the key benefits of using microservices architectures?
- What are API keys used for?
- What are API gateways?
- What is a service mesh?

BPP

APIs at a glance



What is a REST API?

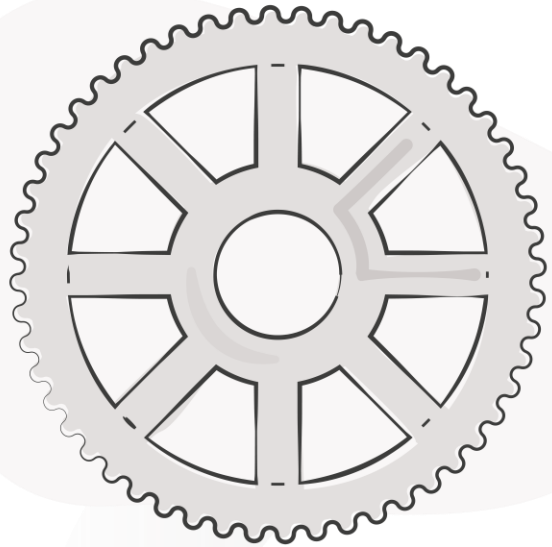


Practical application

Building Careers Through Education

Everybody Matters. Stronger Together. Student, Learner and Client Centric. Trust and Respect. Embrace Change.

Your tutor will now walk you through the Food Hygiene Rating Scheme API - https://api.ratings.food.gov.uk/help



Food Standards Agency

ABOUT THE API | API INDEX | BEST PRACTICES | FAQS | STATUS

Food Hygiene Rating Scheme API (Version 2)

Welcome!

Welcome to the Food Hygiene Rating Scheme Application Programming Interface (FHRS API). The FHRS API provides free programmatic access to the Food Standards Agency Rating Data for England, Scotland, Wales and Northern Ireland. Developers can leverage FSA data to provide their own services and websites, allowing you to build and develop custom solutions that are consistent with the data being held and displayed by the FSA. As a developer you get access to the same data and the majority of the same functionality that is used to deliver the Food Hygiene Rating Scheme website.

BPP

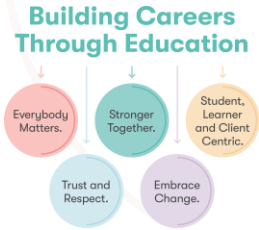
Things we learned from the code walkthrough

We will now recap the following functions we learned from the Python code walkthrough:

- import requests
- import json
- Headers
- Response status codes
- Raising errors and handling exceptions in Python

BPP

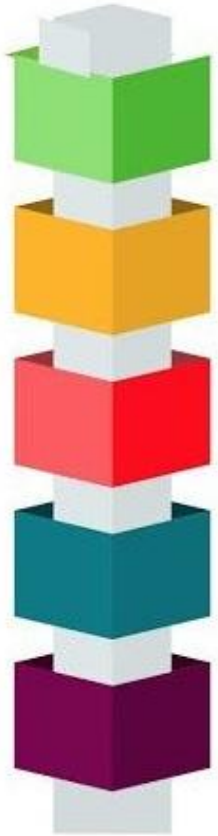
Working with ‘requests’




```
>>> import requests

>>> r = requests.get('https://api.github.com/events')
>>> r.text
u'[{"repository":{"open_issues":0,"url":"https://github.com/…

>>> r = requests.post('http://httpbin.org/post', data = {'key':'value'})

>>> requests.get('https://api.github.com/user', auth=('user', 'pass'))
<Response [200]>
```

BPP

HTTP Status Codes



1XX
INFORMATIONAL

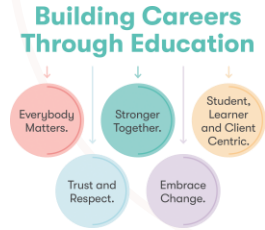
2XX
SUCCESS

3XX
REDIRECTION

4XX
CLIENT ERROR

5XX
SERVER ERROR

API Response Codes



GET

+ 200 OK
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 404 Not Found
+ 500 Internal Server Error

POST

+ 201 Created
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 409 Conflict
+ 500 Internal Server Error

PUT

+ 200 OK
+ 201 Created*
+ 204 No Content
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 500 Internal Server Error

DELETE

+ 204 No Content
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 500 Internal Server Error

BPP

Exceptions

You can write your program to handle certain exceptions

One way to do this is *try*

```
>>> while True:
... try:
... x = int(input("Please enter a number: "))
... break
... except ValueError:
... print("Oops! That was no valid number. Try again...") ...
```

BPP

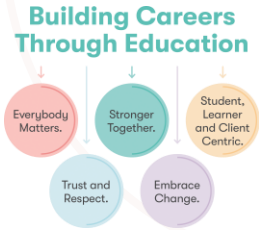
Try statements

A try statement may have more than one except clause to allow multiple exception types to by tried.

An except statement may name exceptions as a parenthesised triple

- Ex :... **except** (RuntimeError, TypeError, NameError):
  ... **pass**

Wildcard exceptions



You may leave the last except clause blank to be used as a wild card.

```
import sys
try:
 f = open('myfile.txt')
 s = f.readline()
i = int(s.strip())
except OSError as err:
print("OS error: {0}".format(err))
 except ValueError:
print("Could not convert data to an integer.")
 except:
 print("Unexpected error:", sys.exc_info()[0])
raise
```

BPP

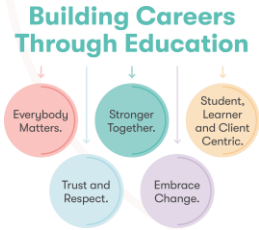
Else clause

Follows all except clauses

Useful for code that needs to be executed if an exception does not come up in the try clause

```
for arg in sys.argv[1:]:
try:
f = open(arg, 'r')
 except IOError: print('cannot open', arg)
else:
print(arg, 'has', len(f.readlines()), 'lines')
f.close()
```

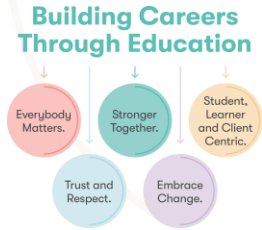
Exception argument



- When an exception occurs it may have a value, known as the exception argument.
- The type of the argument depends on the exception type.
- After the except clause you can specify a variable after the exception name. This variable gets bound to instance.args

BPP

Exception arguments - example




```
>>> try:
 ... raise Exception('spam', 'eggs')
 ... except Exception as inst:
... print(type(inst))
... print(inst.args)
... print(inst)
... x, y = inst.args
... print('x =', x)
... print('y =', y)
<class 'Exception'>
 ('spam', 'eggs')
('spam', 'eggs')
 x = spam
y = eggs
```

BPP

Raising exceptions

The raise statement allows a specific exception to be forced

>>> raise NameError('HiThere’)

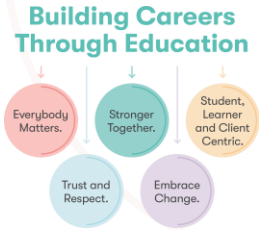
Traceback (most recent call last):

File "<stdin>", line 1, in ?

NameError: HiThere

BPP

Clean-up actions



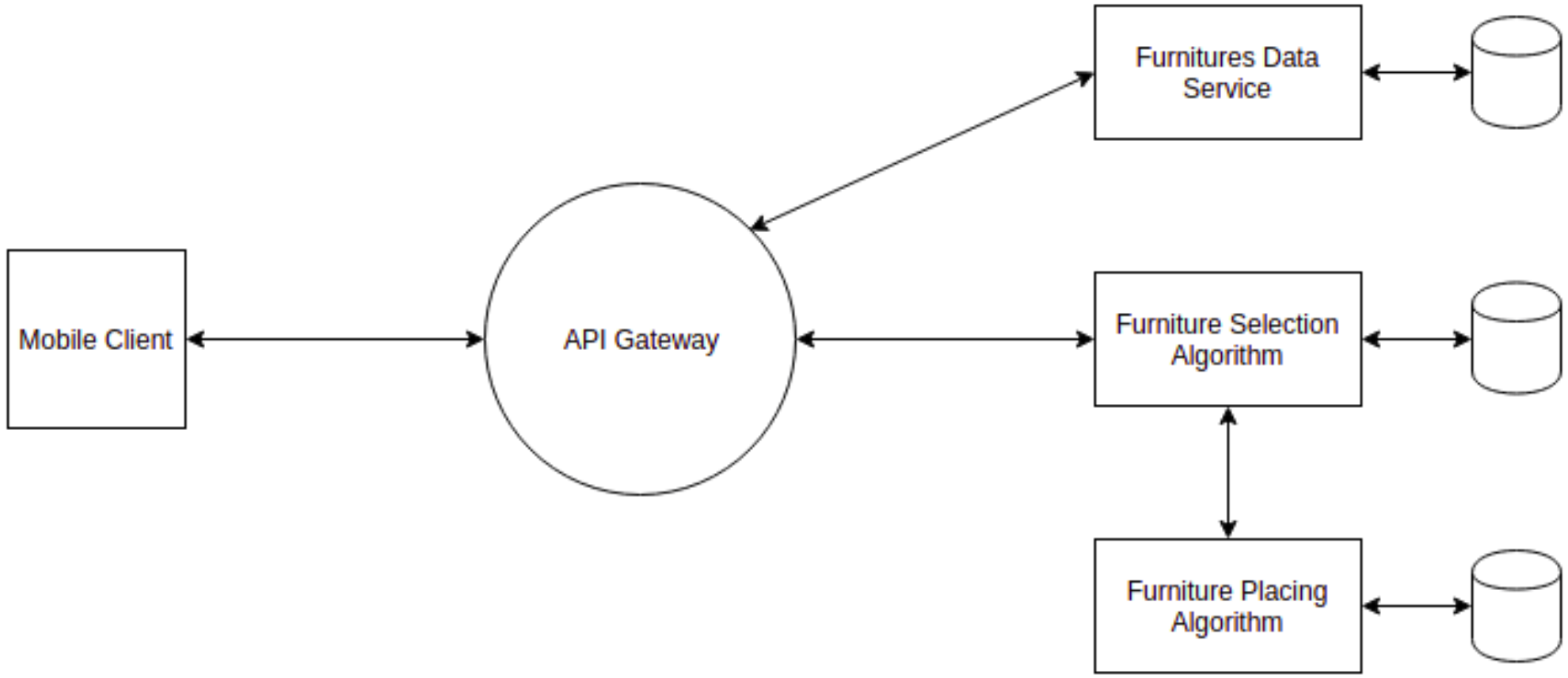
Try statements have an optional clause to define clean up actions:

```
>>> try:
... raise KeyboardInterrupt
 ... finally:
... print('Goodbye, world!')
 ... Goodbye, world!
 KeyboardInterrupt
```

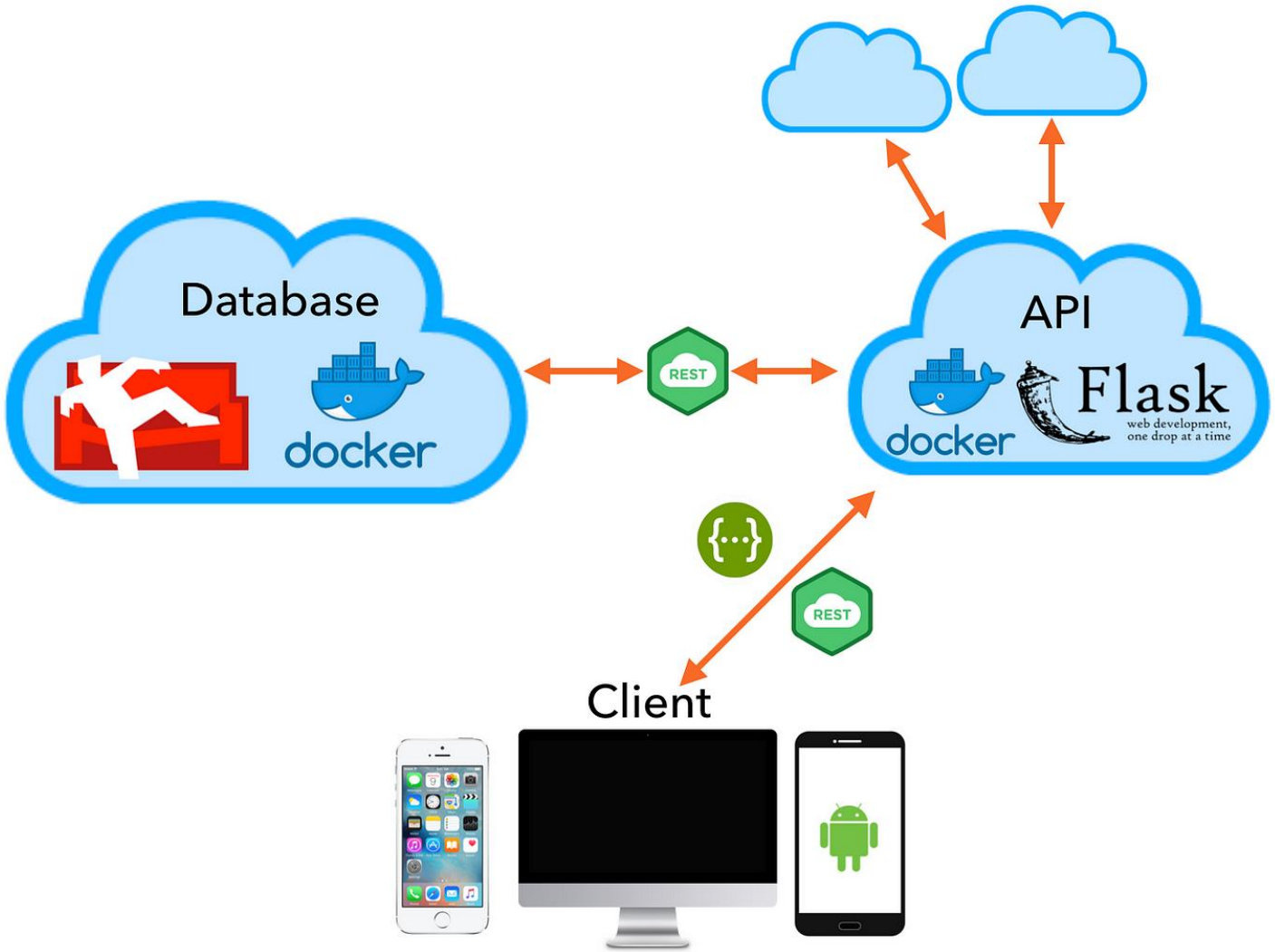
A finally statement is executed at the end of the try statement whether or not an exception occurred.

BPP

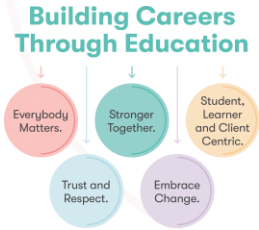
Recap: microservices example



Flask



Flask Code Example

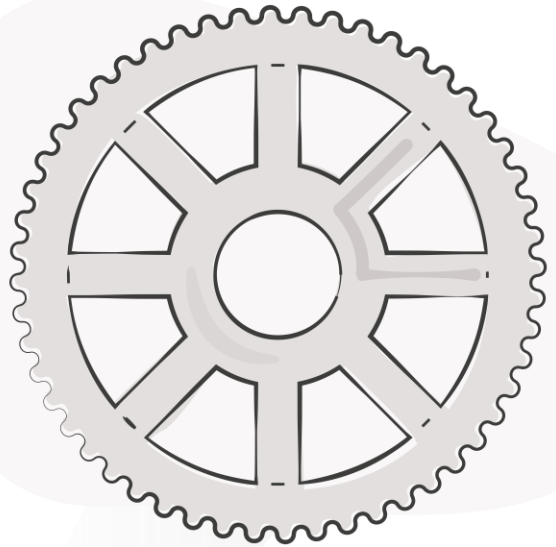



```python
from flask import Blueprint, request, jsonify
from models import Product, products

routes = Blueprint('routes', __name__)

@routes.route('/products', methods=['POST'])
def add_product():
  name = request.json['name']
  description = request.json['description']
  price = request.json['price']
  id = len(products) + 1
  new_product = Product(id, name, description, price)

  products.append(new_product.__dict__)

  return jsonify(new_product.__dict__)

@routes.route('/products', methods=['GET'])
def get_products():
  return jsonify(products)
```

BPP

Practical application

Your tutor will now walk you through building a microservices API with Python and Flask



Practical considerations – data accessibility

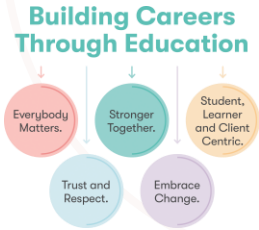
API aggregators are services that combine multiple APIs into a single, cohesive interface, simplifying developer access to diverse data sources. They reduce the complexity of dealing with various APIs by providing a unified way to retrieve and manage data.

Benefits: Streamlines development, reduces latency, and improves data consistency across services.

Examples: Platforms like Zapier and MuleSoft allow for integration of multiple services, enhancing workflow automation.

BPP

Practical considerations – data access



OAuth is an open-standard authorisation protocol that allows third-party services to exchange web resources on behalf of a user. OAuth allows users to grant a secure delegated access to their server resources without sharing their credentials. Commonly used by major online services such as Google, Facebook, and Microsoft to permit the sharing of information between resources securely.

Secure API Access: OAuth is widely used to protect APIs by ensuring that the calls made to an API are authorised. For instance, when a mobile app requests user data from a server, OAuth can ensure that the request is made by an authenticated user and that the app is authorised to access such data.

Scope Control: OAuth scopes allow defining the extent of access that applications have. This is crucial for APIs as it restricts the extent of operations, for example restricting writing user data.

BPP

OAuth and API Gateways

- OAuth's role in securing APIs and microservices is foundational to modern software architecture, enabling secure, scalable, and flexible integrations across disparate systems.

- An API Gateway in a microservices architecture can act as the OAuth 2.0 client and handle user authentication and authorisation.

- It then passes an OAuth token along to individual services, which use the token to decide whether to process the request or deny it based on the token’s scopes.

BPP

JWT

JWT Overview: JSON Web Tokens (JWT) are a compact,
URL-safe means of representing claims to be transferred between two parties.

- **Header**: Specifies the token type (JWT) and the signing algorithm
- **Payload**: Contains the claims, which are statements about an entity (typically, the user).
- **Signature**: ensuring the token's integrity and authenticity.

Usage in APIs and microservices:

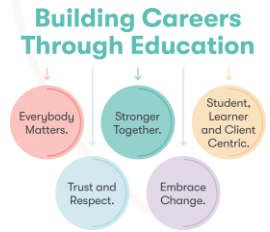
- **Authentication**: JWTs can be used to authenticate subsequent requests after a user logs in,
- reducing the need to query the database repeatedly.
- **Authorisation**: Enables services to verify the token's validity and grant access to resources
- based on the token's claims.

Advantages:

- **Efficiency**: Because JWTs are self-contained, they reduce the need for round-trip
- database lookups for session information.
- **Scalability**: Facilitates scalability in applications by allowing different servers to verify the token
- independently using a shared secret or public/private key pair.

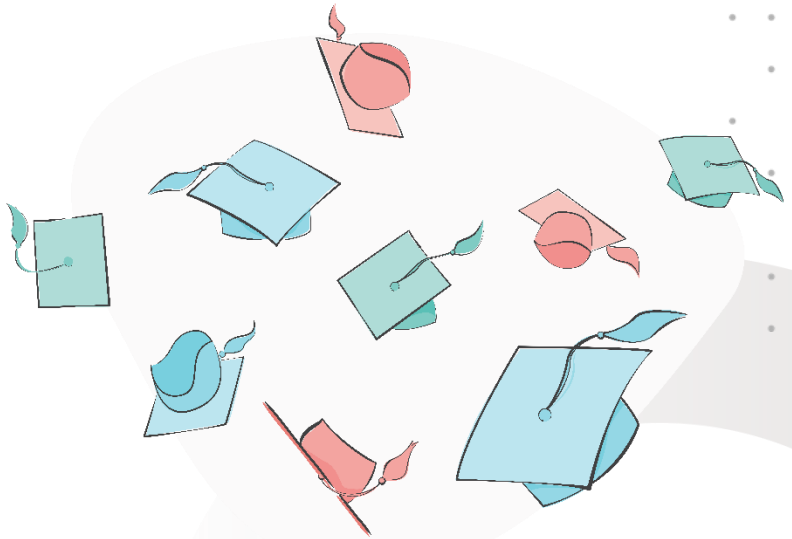
BPP

Key Learning Summary



The key takeaways from this session are as follows:

- **Benefits of Microservices Architectures:** Microservices offer scalability, flexibility, and fault tolerance by allowing independent deployment and scaling of services.
- **Using APIs for Data Ingestion:** APIs facilitate the integration of various data sources, enabling efficient data ingestion and real-time data processing.
- **Designing Data Ingestion Architectures:** Effective data ingestion architectures using APIs and microservices improve data flow, enhance system performance, and support dynamic scaling.



BPP

Thank you

Do you have any questions, comments, or feedback?