Concepts in Micro-Services

This section covers a few important aspects in microservices.

1. Authentication using JWT.
2. Database Management.
3. File Management.
4. API Versioning.
5. Requests between Microservices.

JWT (JSON Web Token)

Json Web Token is a standard to help multiple components interact securely.

A token is issued by a component, which indicates to all parts of the system that the person providing the token can access certain parts of the application.

In a typical application,

A user accesses himself using a username and password, followed by which a jwt token is issued with the permissions to access the application for a certain period of time.

Token format:

`<header:base64string>.<payload:base64string>.<signing_data:base64string>`

Example:

```
    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9(header).eyJpZCI6MjMsImlzcyI6IkRTQ0UiLCJzdWIiOiJFbXBsb3llZSBNaWN
```

The pyjwt library helps install the jwt management library.

Installation

pip install pyjwt

Generating a token

```python
import jwt

# Creates a token for 1 year.

payload = {'sub': 'Employee Token', 'permissions': ['emp.create', 'emp.list'], 'iss': 'My Company',
'exp': 365 * 24 * 24 * 3600 * 24}

token = jwt.encode(payload, 'signing_key')
```

Decoding a token

```python
import jwt

token_payload = jwt.decode('token_string', 'signing_key', algorithm = ['HS256'])
```

A sample flask application which generates a JWT token and returns it in the response

In [2]:
```python
from flask import Flask
from flask import request, make_response

import jwt


class BaseException(Exception):
    status = 400
    message = ""

    def __init__(self, status, message) -> None:
        super().__init__()
        self.status = status
        self.message = message

    def __str__(self):
        return str({'status': self.status, 'message': self.message})

class TokenGenerationError(BaseException):
    def __init__(self) -> None:
        super().__init__(500, "Unable to generate the token")



app = Flask(__name__)

@app.route('/api/token/<int:employee_id>', methods=['POST'])
def fetch_token(employee_id):
    log_message = {
        'operation': 'fetch token',
        'status': 'processing'
    }

    app.logger.info(str(log_message))

    payload = {
        'id': employee_id,
        'iss': 'DSCE',
        'sub': 'Employee Microservice Token'
    }

    try:
        token = jwt.encode(payload,
                           key="mysecretkey")

    except:
        log_message['status'] = 'unsuccessful'
        log_message['reason'] = 'Token generation error'
        app.logger.error(str(log_message))
        err = exceptions.TokenGenerationError()
        return str(err), err.status

    app.logger.info(str(log_message))
    return {'token': token}, 200
```

Exercise

Add an endpoint to the above application to decode the jwt token and return the payload .

Managing Files

Just like how we used

request.json to fetch the JSON payload,

Uploading a file

request.files can be used to access the files uploaded to the server using a multipart request.

Note:

A multipart request has to be used.

Accessing the file object

`file = request.files['filename']`

Save the file to the local file system

`file.save(path)`

Downloading the file

To create a downloadable file, the `send_file` method must be used and a local path must be passed.

This ensures that flask creates a download pathway.

`return save_file(path), 200`

In [4]:
```
import os

from flask import Flask
from flask import request, send_file


class BaseException(Exception):
    status = 400
    message = ""

    def __init__(self, status, message) -> None:
        super().__init__()
        self.status = status
        self.message = message

    def __str__(self):
        return str({'status': self.status, 'message': self.message})

class FileExistsException(BaseException):
    def __init__(self) -> None:
        super().__init__(400, "File with id already exists")

class FileDoesNotExistsException(BaseException):
    def __init__(self) -> None:
        super().__init__(400, "File with id does not exist")

app = Flask(__name__)

@app.route('/api/files/', methods=['POST'])
def create_file():
    log_message = {
        'operation': 'create file',
        'status': 'processing'
    }

    app.logger.info(str(log_message))


    # Access the text part of the mulipart request.
    data = dict(request.form)
    file_name = data.get('name')
    file_id = data.get('file_id')

    file_path = os.getcwd() + '\storage\\'
    # Access the file object
    file = request.files['file']

    overwrite = data.get('overwrite', False)

    if not overwrite and os.path.isfile(file_id):
        log_message['status'] = 'unsuccessful'
        log_message['reason'] = 'File Already exists'
        app.logger.error(str(log_message))
        err = exceptions.FileExistsException()
        return str(err), err.status
    else:
        log_message['status'] = 'successful'
        app.logger.info(str(log_message))
        file.save(file_path + file_id)
        return "", 200


@app.route('/api/files/<file_id>', methods=['GET'])
def get_file(file_id):
    log_message = {
        'operation': 'get file',
        'status': 'processing'
    }

    app.logger.info(str(log_message))
    file_path = os.getcwd() + '\storage\\{}'.format(file_id)

    # Check if the file exists.
    if not os.path.isfile(file_path):
        log_message['status'] = 'unsuccessful'
        log_message['reason'] = 'File Does not exist'
        app.logger.error(str(log_message))
        err = exceptions.FileDoesNotExistsException()
        return str(err), err.status
    else:
        log_message['status'] = 'successful'
        app.logger.info(log_message)
        return send_file(file_path), 200

@app.route('/api/files/<file_id>', methods=['DELETE'])
def remove_file(file_id):
    log_message = {
        'operation': 'remove file',
        'status': 'processing'
    }

    app.logger.info(str(log_message))

    file_path = os.getcwd() + '\storage\\{}'.format(file_id)

    # Check if the file exists.
    if not os.path.isfile(file_path):
        log_message['status'] = 'unsuccessful'
        log_message['reason'] = 'File Does not exist'
        app.logger.error(str(log_message))
        err = exceptions.FileDoesNotExistsException()
        return str(err), err.status
    else:
        # Delete the file.
        os.remove(file_path)
        log_message['status'] = 'successful'
        app.logger.info(log_message)
        return "", 200
```

Exercise:

Check if the created program can manage multiple file types, such as PDF, Videos, Images etc.

Managing Data using databases.

ORMs (Obejct Relational Mappers), are used to manage the database interactions directly from a class.

SQLAlchemy is an ORM, simple, efficient and with built in transaction support.

Installing SQLAlchemy

pip install SQLAlchemy

dress) < 1024 return val_name and val_addr

engine)

)

Initialize the database

In [9]:
```
import sqlalchemy as db
from sqlalchemy.orm import declarative_base
from sqlalchemy.orm import sessionmaker

engine = db.create_engine("sqlite:///employees.sqlite")
conn = engine.connect()
```

Creating a table

In [15]:
```
metadata = db.MetaData()

employee = db.Table('Employee', metadata,
                db.Column('id', db.Integer(), primary_key=True),
                db.Column('name', db.String(255), nullable=False),
                db.Column('address', db.String(1024), default="Nammane"),
                db.Column('pic_id', db.String(1024), default="default")
            )

metadata.create_all(engine)
```

In [12]:
```
# Initialize a db session
Base = declarative_base()
session = sessionmaker(bind=engine)()
```

In [30]:
```
# Save an entry to DB.
class Employeers_Company(Base):
    __tablename__ = "employers"

    name = Column(String)
    id = Column(Integer, primary_key=True)
    address = Column(String)
    pic_id = Column(String)

    def __init__(self, name, id, address, pic_id='default'):
        super().__init__()
        self.name = name
        self.id = id
        self.address = address
        self.pic_id = 'default'

#
#employee4 = Employee_company("Srihari", 2, "JP Nagara", "default_pic")
#session.add(employee4)
#session.commit()
```

```
C:\Users\Prashanth\AppData\Local\Temp\ipykernel_19308\1372814538.py:2: SAWarning: This declarative base already
contains a class with the same class name and module name as __main__.Employeers_Company, and will be replaced i
n the string-lookup table.
  class Employeers_Company(Base):
```

In [36]:
```
# Get an entry by id from DB.
# employee_obj = session.query(Employeers_Company).filter_by(id=12).first()
```

In [ ]:
```
# Remove an entry


# emp = session.query(Employee).filter_by(id=self.id).first()
# session.delete(emp)
# session.commit()
```

In [ ]:
```
# Class for managing
```

In [6]:
```
from sqlalchemy import Column, Integer, String
from flask import request
from app import session, Base


class Employee(Base):
    __tablename__ = "employee"

    name = Column(String)
    id = Column(Integer, primary_key=True)
    address = Column(String)
    pic_id = Column(String)

    def __init__(self, name, id, address, pic_id='default'):
        super().__init__()
        self.name = name
        self.id = id
        self.address = address
        self.pic_id = 'default'

    def __str__(self):
        return str({
            "id": self.id,
            "name": self.name,
            "address": self.address
        })

    def __cache__(self):
        return {
            'key': 'emp_{}'.format(self.id),
            'value': str(self)
        }
    def save(self):
        session.add(self)
        session.commit()

    def get_by_id(self):
        emp = session.query(Employee).filter_by(id=self.id).first()
        return emp

    def delete(self):
        emp = session.query(Employee).filter_by(id=self.id).first()
        session.delete(emp)
        session.commit()

    def update(self):
        emp = session.query(Employee).filter_by(id=id).first()
        emp.name = self.name or emp.name
        emp.address = self.address or emp.address
        return session.commit()

    def validate(self):
        val_name = len(self.name) > 0 and len(self.name) < 256
        val_addr = len(self.address) > 0 and len(self.address) < 1024
        return val_name and val_addr
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[6], line 3
      1 from sqlalchemy import Column, Integer, String
      2 from flask import request
----> 3 from app import session, Base
      6 class Employee(Base):
      7     __tablename__ = "employee"

ModuleNotFoundError: No module named 'app'
```

In [7]:
```
# Ignore the above error.
```

Exercises

Modify the library application to save the books into the database.

Calling other microservices

Requests library helps us to provide communication between microservices.

Installation

pip install requests

In [44]:
```
# Calling another microservice using get method

import requests

headers = {
    'X-Auth-Header': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjMsImlzcyI6IkRTQ0UiLCJzdWIiOiJFbXBsb3llZSBNa
}

response = requests.get('http://localhost:5000/api/employee/1', headers=headers)
response.text
```

Out[44]:
```
"{'id': 1, 'name': 'Prashanth', 'address': 'JP Nagar'}"
```

In [51]:
```
# Calling another microservice using post method

import requests

data = {
    'name': 'Srihari',
    'address': 'Tirupati'
}

headers = {
    'Content-Type': 'application/json',
    'X-Auth-Header': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjMsImlzcyI6IkRTQ0UiLCJzdWIiOiJFbXBsb3llZSBNa
}

response = requests.post('http://localhost:5000/api/employee/', json=data, headers=headers)
```

```
response.text
```

Out[51]:

```
"{'id': 1003, 'name': 'Srihari', 'address': 'Tirupati'}"
```

In [55]:

```python
# Calling another microservice using PUT method

import requests

data = {
    'name': 'Srihari M',
    'address': 'Tirupati'
}

headers = {
    'Content-Type': 'application/json',
    'X-Auth-Header': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjMsImlzcyI6IkRTQ0UiLCJzdWIiOiJFbXBsb3llZSBNa
}

response = requests.put('http://localhost:5000/api/employee/1001', json=data, headers=headers)
response.text
```

Out[55]:

```
"{'id': 1001, 'name': 'Srihari M', 'address': 'Tirupati'}"
```

In [56]:

```python
# Calling another microservice using DELETE method

import requests

headers = {
    'X-Auth-Header': 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjMsImlzcyI6IkRTQ0UiLCJzdWIiOiJFbXBsb3llZSBNa
}

response = requests.delete('http://localhost:5000/api/employee/1001', headers=headers)
response.text
```

Out[56]:

```
''
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js