**Name**: Omkar Deshpande

**Roll No.**: 43212

**Batch**: Q-10

**Divisions**: BE 10

# Assignment 7

**Title:** To develop Microservices framework based distributed application

**Tools:** Python using Flask framework.

**Theory:**

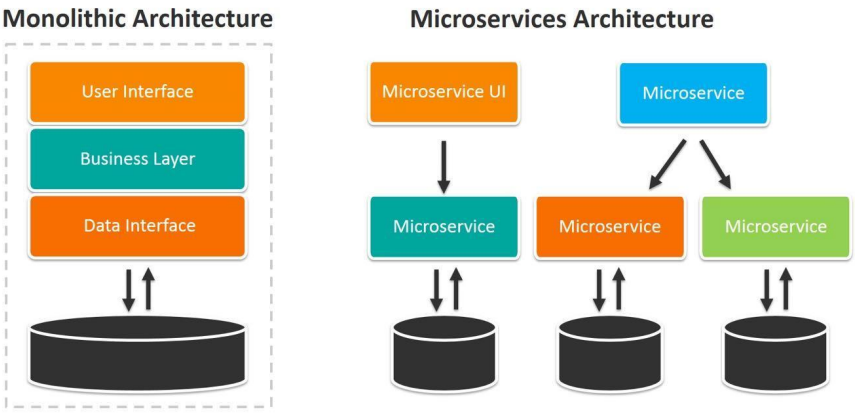
**Microservices:**

- The way developers worked to build applications is changing. In the past, software was built as a large monolithic application where a team of developers would take months to construct a large application built on a common code base.
- Define an architecture that structures the application as a set of loosely coupled, collaborating services. Each service implements a set of narrowly, related functions.
- For example, an application might consist of services such as the order. management service, customer management service, etc.
- Each service has its own database in order to be decoupled from other services. Data consistency between services is maintained.
- Microservices - also known as the microservice architecture - is an architectural a style that structures an application as a collection of services that are

    1. Highly maintainable and testable
    2. Loosely coupled
    3. Independently deployable
    4. Organized around business capabilities.

**Monolithic Design vs Microservice:**

- Traditional application design is often called “monolithic” because the whole thing is developed in one piece.
- Even if the logic of the application is modular it’s deployed as one group, like a Java application as a JAR file for example.
- This monolith eventually becomes so difficult to manage as the larger applications require longer and longer deployment timeframes.
- In contrast, a team designing a microservices architecture for their application will split all of the major functions of an application into independent services.
- Each independent service is usually packaged as an API so it can interact with the rest of the application elements.




**Web frameworks**

- Flask

  Flask (source code) is a Python web framework built with a small core and easy-to-extend philosophy. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine.

- WSGI
  Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

- Werkzeug

  It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

**Common web framework functionality:**

Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include

1. URL routing
2. Input form handling and validation
3. HTML, XML, JSON, and other output formats with a templating engine
4. Database connection configuration and persistent data manipulation through an object-relational mapper (ORM)
5. Web security against Cross-site request forgery (CSRF), SQL Injection, Cross-site Scripting (XSS), and other common malicious attacks 6. Session storage and retrieval.

**Virtual Environment:**

In Python, by default, every project on the system will use the same directories to store and retrieve site packages (third-party libraries) and system packages (packages that are part of the standard Python library).
Consider the scenario where there are two projects: ProjectA and ProjectB, both have a dependency on the same library, ProjectC. The problem becomes apparent when we start requiring different versions of projects. Maybe ProjectA needs v1.0.0, while ProjectB requires the newer v2.0.0,

for example.
Since projects are stored in the site-packages directory according to their name and can't differentiate between versions, both projects, ProjectA, and ProjectB would be required to use the same version which is unacceptable in many cases and hence the virtual environment.
The main purpose of Python virtual environments is to create an isolated environment for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has.

**Using Flask to Create Microservices:**

1. **Using Virtual Environments:** Install virtualenv for a development environment. virtualenv is a virtual Python environment builder. It helps a user to create multiple Python environments side-by-side. Thereby, it can avoid compatibility issues between the different versions of the libraries.

   **The following command installs virtualenv:** `sudo apt-get install virtualenv`

2. **Flask Module:** Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as an argument. The route() function of the Flask class is a decorator, which tells the application which URL should call the associated function.

   **Route decorator:**

   The route() decorator in Flask is used to bind the URL to a function.
   For example −

```
@app.route('/hello')
def hello_world():
return 'hello world'
```

   Here, the URL '/hello' rule is bound to the hello_world() function. As a result, if a user visits http://localhost:5000/hello URL, the output of the hello_world() function will be rendered in the browser.

**Conclusion:**

Thus, in this assignment, I studied microservices, virtual environments, and web frameworks.