Unit 9: Developing an API for a Distributed Environment

1. Create an API and use it to create and read records.

In the world of modern software development, APIs play a foundational role. They serve as the software codes that facilitate communication between two programs, offering developers a straightforward set of methods for accessing data across multiple channels. APIs are a potent tool, providing authorisation and access to the data that users and other applications require. The most prevalent API architectures are Representational State Transfer (REST) and Simple Object Access Protocol (SOAP) (Lutkevich & Nolle, N.D).

Lutkevich & Nolle (N.D) state that APIs have significantly transformed software development, enabling businesses to streamline their internal development processes, collaborate more efficiently with other developers, and enhance customer experience. By providing a standardised set of rules for writing application code, APIs have helped businesses reduce development time and cost while minimising the risk of errors. APIs also allow companies to securely manage their data and service functionality, expand their customer base, and generate revenue by monetising their services. This has created new business opportunities to grow and thrive in the digital age.

However, for Gillis (N.D.), building an API from scratch can be challenging and involves many complex steps. These can entail setting up several endpoints, configuring a server, managing requests and responses, and connecting to a database. It calls for a strong command of programming concepts and procedures and web development experience.

Based on the concepts of Fabisiak (2020), the following code is an example of a RESTful API for creating and reading records using Python and the Flask framework:

1. Installing Flask and Flask_RESTful

```
pip install Flask
```

```
pip install Flask-RESTful
```

2. Create and initialise the file

```
"""Import modules"""
from flask import Flask
from flask_restful import Resource, Api, reqparse

app = Flask(__name__)
api = Api(app)

BOOKS = {}

# Sample data
books = [
    {"id": 1, "title": "Software Architecture with Python", "author": "Anand Balachandran Pillai"},
    {"id": 2, "title": "Python 3 Object-Oriented Programming", "author": "Dusty Phillips"},
]

# Books List class
class BooksList(Resource):
    """methods get"""
    def get(self):
        """Method that returns available books"""
        return BOOKS

    def post(self):
        """Method to add new books """
        parser = reqparse.RequestParser()
        parser.add_argument("title")
        parser.add_argument("author")
        args = parser.parse_args()
        book_id = str(int(max(BOOKS.keys()) if BOOKS else 0) + 1)
        BOOKS[book_id] = {
            "title": args["title"],
            "author": args["author"],
        }
        return BOOKS[book_id], 201


api.add_resource(BooksList, '/books/')

# Book class
class Book(Resource):
    def get(self, book_id):
        """Books ID"""
        if book_id in BOOKS:
            return BOOKS[book_id]
        return "Not found", 404

    def put(self, book_id):
        """Method to update books id"""
        if book_id not in BOOKS:
            return "Record not found", 404
        parser = reqparse.RequestParser()
```

```
        parser.add_argument("title")
        parser.add_argument("author")
        args = parser.parse_args()
        book = BOOKS[book_id]
        book["title"] = args["title"] if args["title"] is not None else book["title"]
        book["author"] = args["author"] if args["author"] is not None else book["author"]
        return book, 200

    def delete(self, book_id):
        """Delete method to delete existent book"""
        if book_id in BOOKS:
            del BOOKS[book_id]
            return '', 204
        return "Not found", 404


api.add_resource(Book, '/book/<string:book_id>')

if __name__ == "__main__":
    app.run(debug=True)
```

3. Output:

```
C:\Users\hcham\anaconda3\envs\pythonProject-SSD\python.exe
C:\Users\hcham\PycharmProjects\pythonProject-SSD\record_api.py
  Serving Flask app 'record_api'
  Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production
WSGI server instead.
  Running on http://127.0.0.1:5000
Press CTRL+C to quit
  Restarting with stat
  Debugger is active!
  Debugger PIN: 251-742-674
```

2. Become familiar with the capabilities of Python's flask library.

Flask is a Python web framework that enables developers to develop web applications with complete control over data access quickly. It provides simplified REST API development, versatility for various projects, minimal boilerplate code, essential components, and extensibility through Flask extensions (Agrawal, 2021). Therefore, Webcrome Software (2023) states that Flask provides the tools and libraries to build web applications without imposing specific patterns or structures. For instance, here

are some of the critical capabilities and features of Flask based on ideas of Ibrar (2023):

1. **Starting with Flask:** To build a web app with Flask, first install Flask using pip, the Python package manager.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run()
```

2. **Routing:** Flask provides an elegant way to define routes using decorators. It also supports dynamic routing through URL pattern parameters:

```
@app.route('/user/<username>')
def show_user_profile(username):
    return f'User: {username}'
```

3. **Templates:** Flask includes a templating that allows the creation of dynamic HTML templates for rendering data:

```
@app.route('/template')
def render_template_example():
    return render_template('template.html', name='John')
```

4. Response Handling: Flask simplifies handling form input by providing a response object and request.form dictionary:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/form', methods=['GET', 'POST'])
def form_example():
    if request.method == 'POST':
        name = request.form['name']
```

```
        return f'Hello, {name}!'
    return render_template('form.html')

if __name__ == '__main__':
    app.run()
```

In addition, Webcrome Software (2023) highlight that the Flask micro web framework provides the essentials for building web applications without imposing a strict structure or set of tools. The minimalist approach allows developers to choose the necessary components while keeping the framework lightweight. Key features include minimalist design, modularity, simple routing mechanism, Jinja2 templates, built-on Werkzeug WSGI toolkit, RESTful support, vibrant community, and lightweight design. However, developers often use more full-featured web frameworks like Django for larger and more complex applications, according to Majdak (2023). On the other hand, Flask is an excellent choice for smaller applications or when specific libraries are needed due to its lightweight nature, making it more minimalist and potentially faster.

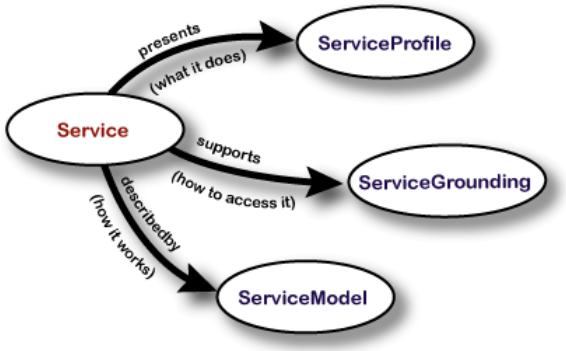
3. Design an ontology which can be used in standardised service deployments.

Ontology formally represents knowledge that defines concepts and their relationships in a domain. It is used to share and reuse knowledge and facilitate communication and reasoning among people or computer systems. Ontologies use a graph model, where nodes represent concepts and edges represent relationships. They provide standardisation in representing and sharing knowledge, improving communication and collaboration across different organisations and systems (graph.build, N.D.).

According to Hara Gopal & Bharati (2017), service ontology is a valuable tool for creating end products or services by combining goods and services. It is a component-

based description of services that streamlines the electronic design and production of services. Service ontology involves conceptualising the service, identified by a service name and description and organised into three layers: top-level, domain knowledge level, and concrete deck. The generic service ontology is further categorised into three top-level ontological distinctions: the customer-value perspective, the supply-side perspective, and the joint operationalisation of these viewpoints in the actual service production process.

Furthermore, W3-Org (2004) provided an Upper Ontology for Services, which aims to provide users with vital information about a web service's functionality, usage, and interaction methods. The ServiceProfile describes the providers' services and the requests' needs. It consists of provider contact information, functional description of the service, and service properties.



Overview of an OWL-S Description (W3-Org, 2004)

Gruber proposed modelling ontologies using frames and first-order logic. He identified five components: classes, relations, functions, formal axioms and instances. Classes represent concepts, associations connect ideas, functions compute values, standard axioms define knowledge that cannot be described otherwise, and instances represent individual ontology elements (CORDIS, 2022).

To establish a standardised ontology for service deployments, we can adhere to the guidelines proposed by (Slimani, N.D.) by following these steps:

1. Conduct a thorough domain analysis to identify the crucial concepts and their interrelationships. This involves identifying different types of services that can be deployed, components of a service deployment, and the associations between these elements.

2. Utilize axioms to formulate logical statements that explain the relationships between the concepts within the domain. For instance, we could define an axiom that states that a service deployment must have at least one service component.

3. Categorize the concepts and relationships into a hierarchy to enhance the ontology's modularity and reusability. For example, we could have a top-level category for "Service Deployment" and subcategories for "Service Component," "Infrastructure Component," and "Configuration."

4. Assign unique identifiers to all the ontology concepts and relationships to ensure they are referred to unambiguously.

5. Document the ontology by providing comprehensive descriptions of the concepts and relationships and examples of their application. This step guarantees that the ontology is easily comprehended and helpful to others.

For instance, below is an outline example of an ontology for standardised service deployments based on the ideas of Chamekh & Le Mouël (2007):

Top-level concepts:

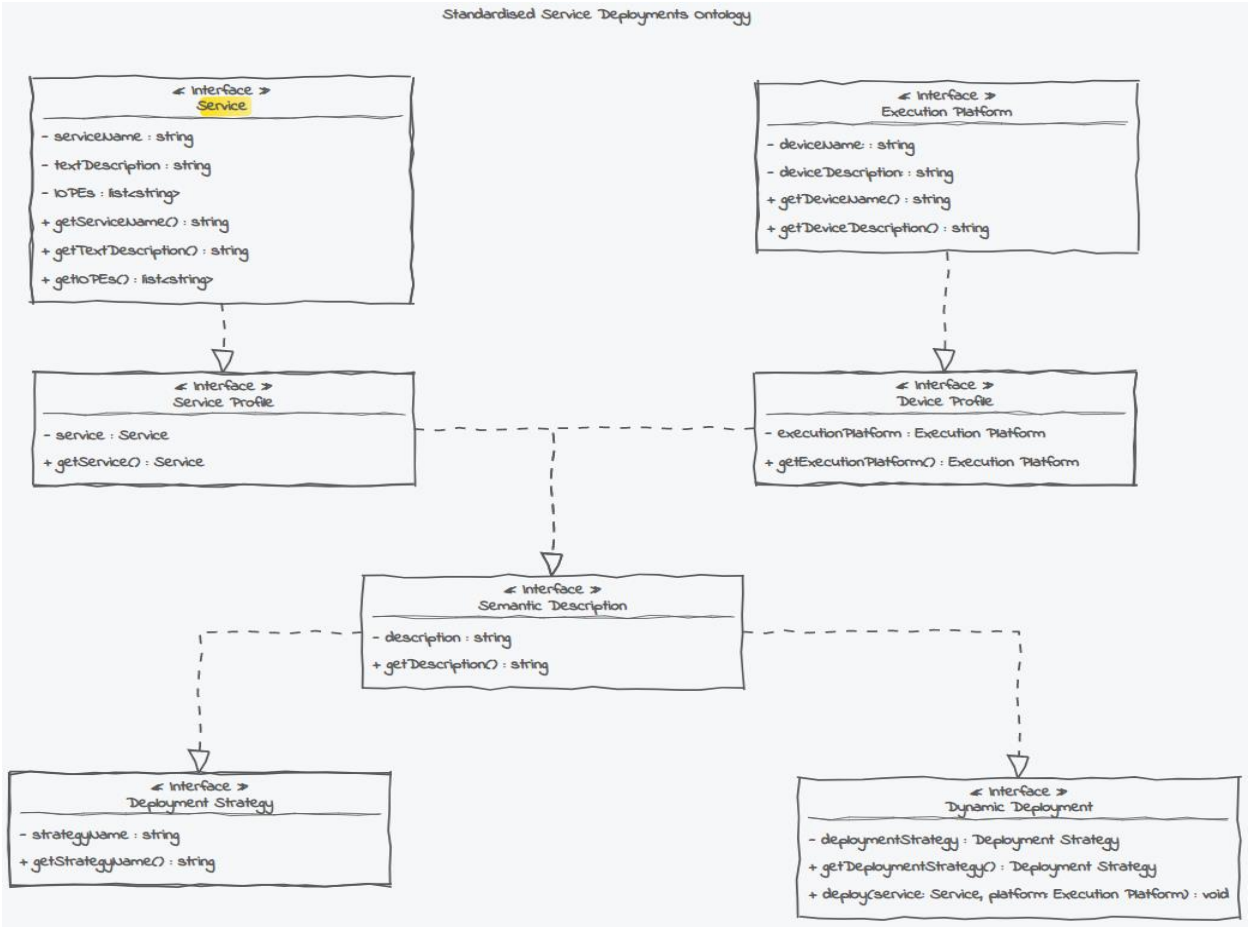
- Service
- Execution platform
- Semantic context
- Dynamic deployment

Sub-concepts:

- Service profile (serviceName, textDescription, IOPEs)
- Process model (composite process, atomic process)
- Device profile
- Deployment strategy (OptimizedDeployment, LocationTrackingDeployment, UserCustomizedDeployment)

Relationships:

- Service has a semantic description
- Execution platform has a semantic description
- Semantic context describes services and execution platforms
- Dynamic deployment uses semantic descriptions to deploy services to execution platforms
- Service profile describes a service
- Process model describes a service
- Device profile describes an execution platform
- Deployment strategy uses semantic descriptions to deploy services to execution platforms



References

Lutkevich, B. & Nolle, T. (N.D.). *What Is an API (Application Program Interface)?* [online] SearchAppArchitecture. Available at: https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API.

Gillis, A. S (N.D.). *What is REST API (RESTful API)?* [online] Available at: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API.

Fabisiak, R. (2020). *Create a simple REST API with Python and Flask in 5 minutes*. [online] Duomly - Programming courses online. Available at: https://medium.com/duomly-blockchain-online-courses/how-to-create-a-simple-rest-api-with-python-and-flask-in-5-minutes-94bb88f74a23.

Agrawal, R. (2021). *What is Flask | A Comprehensive Guide on using Flask for Data Science*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-using-flask-for-data-science/.

Software, W. (2023). *WHAT IS FLASK IN PYTHON?* [online] Medium. Available at: https://medium.com/@inquiry_85245/what-is-flask-in-python-911e90fcfa61.

Ibrar, M. (2023). *Building Web Applications with Flask: A Pythonic Journey*. [online] Available at: https://dev.to/moiz697/building-web-applications-with-flask-a-pythonic-journey-2nc.

Majdak, M. (2023). *Flask vs Django: Different Python Web Frameworks*. [online] Available at: https://startup-house.com/blog/flask-v-django.

graph.build (N.D.). *Ontology in Graph Models and Knowledge Graphs*. [online] Available at: https://graph.build/resources/ontology.

Hara Gopal, V., & Bharati, K. (2017). An Overview of Service Ontology in Semantic Service Search. *International Journal of Applied Engineering Research*, [online] 12, pp.5905–5909. Available at: https://www.ripublication.com/ijaer17/ijaerv12n16_64.pdf.

W3.org. (2023). *OWL-S: Semantic Markup for Web Services*. [online] Available at: https://www.w3.org/submissions/OWL-S/#4.

CORDIS, cordis.europa.eu (2022). *Ecosystem for Collaborative Manufacturing Processes – Intra- and Interfactory Integration and Automation*. [online] CORDIS | European Commission. Available at: https://cordis.europa.eu/project/id/723145.

Vob, J. (2013). *The Service Ontology*. [online] Available at: https://dini-ag-kim.github.io/service-ontology/service.html.

Slimani, T. (N.D.). *A Study Investigating Typical Concepts and Guidelines for Ontology Building*. [online] Available at: https://arxiv.org/ftp/arxiv/papers/1509/1509.05434.pdf.

Hajer Chamekh and Frédéric Le Mouël (2007). An Ontology-based Approach to Semantically Deploy Services in Pervasive Environments. doi:https://doi.org/10.1109/perser.2007.4283947.