REST API

# Task

## Background

vCare-as-a-Service is a service exposed by a virtual eCare (vCare) system through an API to allow the integration with 3rd party systems. For example, a patient who is user of platform X (a 3rd party rehabilitation/coaching system e.g. Imaginary, CC2U,…), even if he/she is using a number of services, none of them will be linked with a clinical pathway, as supported by vCare. By means of the vCare-as-a-Service, the user of platform X will be able to subscribe to the vCare platform and “link” his/her disease with the recommended clinical pathway. vCare services will be “consumed” by platform X that implements the vCare API and in this way the internal decisions making of platform X for rehabilitation/coaching will be more efficient. Platform X will therefore need to associate its services with the input coming from the vCare and orchestrate their behavior.

## Description

1. Study how an API can be developed without knowing the full extent of endpoints offered by a service and propose solutions for achieving this.
2. Develop a basic API that polls a service bus and first acquires a list of service IDs, then, according to service IDs, consumes the data from specific service endpoints.
3. Test basic API.

 Write a report to present all the finding and conclusions.

## Requirements and Assessment

1. API study (60% of the total mark).
2. Development and testing of API. (40% of the total mark).

REST stands for representational state transfer and is a software architecture style that defines a pattern for client and server communications over a network. REST provides a set of constraints for software architecture to promote performance, scalability, simplicity, and reliability in the system.

REST defines the following architectural constraints:

1. Stateless: The server won’t maintain any state between requests from the client.
2. Client-server: The client and server must be decoupled from each other, allowing each to develop independently.
3. Cacheable: The data retrieved from the server should be cacheable either by the client or by the server.
4. Uniform interface: The server will provide a uniform interface for accessing resources without defining their representation.
5. Layered system: The client may access the resources on the server indirectly through other layers such as a proxy or load balancer.
6. Code on demand (optional): The server may transfer code to the client that it can run, such as JavaScript for a single-page application.

HTTP Methods

REST APIs listen for [HTTP methods](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) like GET, POST, and DELETE to know which operations to perform on the web service’s resources. A **resource** is any data available in the web service that can be accessed and manipulated with **HTTP requests** to the REST API. The HTTP method tells the API which action to perform on the resource.

While there are many HTTP methods, the five methods listed below are the most commonly used with REST APIs:

| **HTTP method** | **Description** |
| --- | --- |
| GET | Retrieve an existing resource. |
| POST | Create a new resource. |
| PUT | Update an existing resource. |
| PATCH | Partially update an existing resource. |
| DELETE | Delete a resource. |

A REST API client application can use these five HTTP methods to manage the state of resources in the web service.

Status Codes

Once a REST API receives and processes an HTTP request, it will return an **HTTP response**. Included in this response is an **HTTP status code**. This code provides information about the results of the request. An application sending requests to the API can check the status code and perform actions based on the result. These actions could include handling errors or displaying a success message to a user.

Below is a list of the most common status codes returned by REST APIs:

| **Code** | **Meaning** | **Description** |
| --- | --- | --- |
| 200 | OK | The requested action was successful. |
| 201 | Created | A new resource was created. |
| 202 | Accepted | The request was received, but no modification has been made yet. |
| 204 | No Content | The request was successful, but the response has no content. |
| 400 | Bad Request | The request was malformed. |
| 401 | Unauthorized | The client is not authorized to perform the requested action. |
| 404 | Not Found | The requested resource was not found. |
| 415 | Unsupported Media Type | The request data format is not supported by the server. |
| 422 | Unprocessable Entity | The request data was properly formatted but contained invalid or missing data. |
| 500 | Internal Server Error | The server threw an error when processing the request. |

These ten status codes represent only a small subset of the available [HTTP status codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes). Status codes are numbered based on the category of the result:

| **Code range** | **Category** |
| --- | --- |
| 2xx | Successful operation |
| 3xx | Redirection |
| 4xx | Client error |
| 5xx | Server error |

HTTP status codes come in handy when working with REST APIs as you’ll often need to perform different logic based on the results of the request.

One method to interact with REST API using python is using [Python’s requests library](https://realpython.com/python-requests/). This library enable us to send HTTPS requests.

# Study how an API can be developed without knowing the full extent of endpoints offered by a service and propose solutions for achieving this

When I develop an API client without knowing the full set of endpoints in advance, I need a discovery mechanism that allows the client to adapt dynamically. There are several possible approaches:

**Capabilities / HATEOAS** – the service provides a stable endpoint (e.g., /services/{id}/capabilities) that lists the available operations and their URIs. By following these links, I can dynamically discover what actions are supported. This method is simple and allows the client to adapt when new endpoints are added.

**OpenAPI / Swagger Introspection** – some services expose an OpenAPI specification (commonly at /openapi.json). By parsing this specification, I can identify all endpoints, methods, and parameters, and even generate code automatically. This gives me complete visibility, but it requires the provider to publish and maintain the specification.

**Centralized Service Registry** – in this approach, I would first query a central registry that contains the list of all services and their endpoints. After retrieving this information, I can connect to the correct service. This option simplifies discovery from my side but creates a single point of failure and adds another dependency.

**My choice:**  
For my implementation, I chose the **capabilities endpoint** approach. I find it the most straightforward, standardized, and extensible solution, because it allows me to automatically adapt to changes without relying on an external registry or a full OpenAPI specification.

# Develop a basic API that polls a service bus and first acquires a list of service IDs, then, according to service IDs, consumes the data from specific service endpoints

I am using FAST Api and uvicorn to develop a custom API

Used uvicorn to run a server, FastApi to create an app and then started writing the endpoints.

We can use the endpoint “docs”, and FastAPI enables us to test the API and to have an interactive documentation