# 24_May_Flask_Assignment

July 18, 2025

1. What is a Web API

A Web API (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate with each other over the web using HTTP. It enables client applications (like browsers or mobile apps) to interact with server-side services or data.

2. How does a Web API differ from a web service

| Feature | Web API | Web Service |
|---|---|---|
| Protocol | Usually uses HTTP/HTTPS | Often uses SOAP over HTTP |
| Format | JSON, XML, or other formats | Usually XML |
| Flexibility | More flexible and lightweight | Heavier and more rigid |
| Examples | RESTful APIs, GraphQL | SOAP, XML-RPC |

```
Web API is a broader term, while Web Service is a specific kind of API that adheres to specifi
```

3. What are the benefits of using Web APIs in software development

Interoperability between systems

Separation of concerns (frontend vs backend)

Scalability and easier updates

Reusability across multiple platforms

Faster development through modular services

4. Explain the difference between SOAP and RESTful APIs

| Feature | SOAP | RESTful API |
|---|---|---|
| Protocol | Strict protocol (SOAP) | Uses HTTP methods |
| Format | XML only | JSON, XML, etc. |
| Flexibility | Less flexible, more verbose | Lightweight and faster |
| State | Stateful | Stateless |
| Usage | Enterprise apps (banking, etc.) | Web/mobile apps |

5. What is JSON and how is it commonly used in Web APIs

JSON (JavaScript Object Notation) is a lightweight data-interchange format that's easy to read and write for humans and easy to parse for machines. In Web APIs, JSON is used to structure request and response data, especially in RESTful APIs.

6. Can you name some popular Web API protocols other than REST

SOAP – XML-based protocol

GraphQL – Query-based API by Facebook

gRPC – High-performance RPC protocol by Google

OData – Protocol for querying and updating data

WebSockets – For real-time, bidirectional communication

7. What role do HTTP methods (GET, POST, PUT, DELETE, etc.) play in Web API development

`HTTP methods define the type of operation a client wants to perform:`

GET – Retrieve data

POST – Create new data

PUT – Update existing data

DELETE – Remove data

PATCH – Partial update They support CRUD operations in RESTful APIs.

8. What is the purpose of authentication and authorization in Web APIs

Authentication: Verifies who the user is (e.g., using API keys, OAuth, JWT).

Authorization: Determines what the authenticated user is allowed to do. This ensures security, prevents unauthorized access, and maintains data integrity.

9. How can you handle versioning in Web API development

URI Versioning: /api/v1/resource

Header Versioning: Accept: application/vnd.api+json; version=2

Query Parameters: /resource?version=1

Content Negotiation: Via Accept and Content-Type headers

Versioning allows backward compatibility and controlled upgrades.

10. What are the main components of an HTTP request and response in the context of Web APIs

`HTTP Request:`

Method: GET, POST, PUT, DELETE

URL: Endpoint to access the resource

Headers: Metadata (e.g., Content-Type, Authorization)

Body: Data sent to the server (usually in POST/PUT)

`HTTP Response:`

Status Code: Indicates result (e.g., 200 OK, 404 Not Found)

Headers: Metadata (e.g., Content-Type)

Body: Returned data (e.g., JSON response)

11. Describe the concept of rate limiting in the context of Web APIs

Rate limiting restricts the number of API requests a client can make within a specific time window.

`It helps:`

Prevent abuse and denial-of-service attacks

Protect server resources

Ensure fair usage across users

`Example: Limit: 100 requests per minute per user.`

12. How can you handle errors and exceptions in Web API responses

`Standard error handling includes:`

HTTP status codes (e.g., 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error)

Consistent error messages in the response body (usually in JSON)

Error logging on the server side

Validation of input data to catch issues early

`Example JSON error response:`

{

“status”: 400,

“error”: “Invalid input”,

“message”: “Email field is required”

}

13. Explain the concept of statelessness in RESTful Web APIs |

`In REST, statelessness means:`

Each API call is independent.

The server does not store client state between requests.

All required data (like authentication or session info) must be sent with every request.

`This makes REST scalable and simplifies server design.`

14. What are the best practices for designing and documenting Web APIs

Use RESTful conventions (GET, POST, PUT, DELETE)

Use nouns, not verbs in endpoints (e.g., /users not /getUsers)

Version your APIs from the start (e.g., /api/v1/)

Use consistent naming and response structure

Return appropriate HTTP status codes

Secure your APIs (auth, rate limiting, etc.)

Document using tools like Swagger (OpenAPI), Postman, or Redoc

15. What role do API keys and tokens play in securing Web APIs

API Keys: Identify the calling application; simple, but less secure.

Tokens (e.g., OAuth2, JWT): Authenticated and often time-limited; offer better security.

`They help with:`

Authentication: Who are you?

Authorization: What can you do?

Rate limiting and monitoring usage

16. What is REST, and what are its key principles

REST (Representational State Transfer) is an architectural style for designing networked applications.

`Key principles:`

Statelessness

Client-Server architecture

Cacheability

Uniform interface

Layered system

Resource-based URIs

17. Explain the difference between RESTful APIs and traditional web services

| Feature | RESTful APIs | Traditional Web Services (SOAP) |
|---|---|---|
| Protocol | HTTP | SOAP over HTTP or other protocols |
| Format | JSON, XML | XML only |
| Complexity | Simple and lightweight | Complex with strict contracts |
| Flexibility | High | Limited |
| Statelessness | Yes | Often stateful |

18. Describe the concept of statelessness in RESTful APIs

`RESTful APIs are stateless, meaning:`

Each request from client to server must contain all the information needed to understand and process it.

No session or context is stored on the server between requests.

19. What are the main HTTP methods used in RESTful architecture, and what are their purposes

GET – Retrieve a resource (e.g., get user data)

POST – Create a new resource (e.g., add new user)

PUT – Update an entire resource

PATCH – Partially update a resource

DELETE – Remove a resource

OPTIONS – Returns supported methods for a resource

`These map directly to CRUD operations.`

20. What is the significance of URIs (Uniform Resource Identifiers) in RESTful API design

`In REST:`

URIs uniquely identify resources, not actions.

URIs should be clear, predictable, and resource-based.

`Examples:`

/users/123 – good RESTful design

/getUserById?id=123 – action-oriented (less RESTful)

`URIs are central to routing and clarity in RESTful architecture.`

21. Explain the role of hypermedia in RESTful APIs. How does it relate to HATEOAS

Hypermedia is data that contains links to other resources.

HATEOAS (Hypermedia As The Engine Of Application State) is a REST constraint where the server provides navigational links in the response so the client knows what actions are available next.

`Example:`

[ ]:
```
{
  "user": {
    "id": 1,
    "name": "Udit",
    "links": [
      { "rel": "self", "href": "/users/1" },
      { "rel": "orders", "href": "/users/1/orders" }
    ]
  }
}
```

`This allows dynamic discovery of actions, improving decoupling between client and server.`

22. What are the benefits of using RESTful APIs over other architectural styles

Simplicity (uses HTTP)

Statelessness improves scalability

Language-agnostic (works across platforms)

Flexible format support (JSON, XML, etc.)

Cacheable responses

Wide adoption and community support

23. Discuss the concept of resource representations in RESTful APIs

A resource (e.g., a user, order, product) can have multiple representations, such as:

JSON (default for most REST APIs)

XML, YAML, HTML, etc.

`The representation is what's sent to and from the client, not the resource itself. Clients use`

Accept: application/json

24. How does REST handle communication between clients and servers

Communication is over HTTP

REST uses stateless requests

The client sends requests with all required data

The server processes and responds with a resource representation

It's asynchronous, scalable, and platform-independent

25. What are the common data formats used in RESTful API communication

JSON (most common) – lightweight, human-readable

XML – used in legacy systems or strict schema needs

HTML – for web clients

YAML, CSV, and protobuf (less common)

`Content-Type and Accept headers determine format:`

Content-Type: application/json

26. Explain the importance of status codes in RESTful API responses

`Status codes:`

Indicate the result of a request

Help the client know what happened and how to respond

`Common examples:`

200 OK – Successful request

201 Created – Resource successfully created

400 Bad Request – Invalid input

401 Unauthorized – Auth needed

404 Not Found – Resource missing

500 Internal Server Error – Server-side issue

27. Describe the process of versioning in RESTful API development

API versioning ensures backward compatibility as APIs evolve.

`Common versioning strategies:`

URI versioning: /api/v1/users

Header versioning: Accept: application/vnd.api+json; version=2

Query parameter: /users?version=1

`Best practice: version from day one and document clearly.`

28. How can you ensure security in RESTful API development? What are common authentication methods?

`Best practices:`

Use HTTPS

Implement Authentication & Authorization

Rate limiting, input validation, and CORS controls

Encrypt sensitive data

`Common Authentication Methods:`

API Keys (simple)

OAuth 2.0 (industry standard for delegated access)

JWT (JSON Web Tokens) (stateless, compact)

Basic Auth (base64 encoded, not secure without HTTPS)

29. What are some best practices for documenting RESTful APIs?

Use OpenAPI (Swagger) or Postman for interactive documentation

Include:

`Endpoint URLs and methods`

`Required parameters`

`Sample requests/responses`

`Authentication requirements`

`Status codes and error messages`

Keep it up-to-date with the code

Provide examples for clarity

30. What considerations should be made for error handling in RESTful APIs?

`Best practices:`

Use proper HTTP status codes

Return consistent error structure:

[ ]:
```
{
  "status": 400,
  "error": "InvalidRequest",
  "message": "Email is required."
}
```

Avoid exposing internal server details

Provide clear, actionable messages

Log errors on the server for debugging

Support error codes and trace IDs for tracking

31. What is SOAP, and how does it differ from REST?

SOAP (Simple Object Access Protocol) is a protocol used to exchange structured information in web services using XML over HTTP, SMTP, or other protocols.

| Feature | SOAP | REST |
|---|---|---|
| Protocol | Strict protocol | Architectural style |
| Format | XML only | JSON, XML, etc. |
| Transport | HTTP, SMTP, etc. | HTTP only |
| Security | Built-in (WS-Security) | Uses HTTPS & external tokens |
| Flexibility | Less flexible, more verbose | Lightweight and flexible |

32. Describe the structure of a SOAP message.

A SOAP message is an XML document with the following structure:

```
<soap:Envelope>
<soap:Header>
    <!-- Optional metadata like authentication -->
</soap:Header>
<soap:Body>
    <!-- Actual message/request/response -->
</soap:Body>
</soap:Envelope>
```

Envelope: Root element

Header: Optional, for metadata (auth, routing, etc.)

Body: Required, contains message or fault info

33. How does SOAP handle communication between clients and servers?

SOAP uses XML-based messaging over various transport protocols (typically HTTP).

It defines a contract (WSDL) that clients must follow.

Communication is usually synchronous, but it can also support asynchronous messaging.

SOAP ensures strict validation, type safety, and extensibility.

34. What are the advantages and disadvantages of using SOAP-based web services?

`Advantages:`

Platform-agnostic and language-independent

Built-in standards for security (WS-Security), transactions, etc.

WSDL contract ensures strict typing

Can use multiple transport protocols (HTTP, SMTP, etc.)

`Disadvantages:`

Verbose due to XML

Slower than REST

Harder to debug and develop

Complex setup and maintenance

35. How does SOAP ensure security in web service communication?

`SOAP supports a specification called WS-Security, which includes:`

Message encryption (XML Encryption)

Digital signatures (XML Signature)

Username/password tokens

Security tokens (e.g., SAML)

`SOAP can enforce message-level security, making it suitable for enterprise-level apps like ban`

36. What is Flask, and what makes it different from other web frameworks?

Flask is a lightweight and flexible Python web framework used for building web applications and APIs.

`Key features:`

Minimalist and modular

No default ORM or form validation (unlike Django)

Uses Jinja2 templating

Great for microservices or small-to-medium apps

37. Describe the basic structure of a Flask application.

```
[ ]: #Minimal example:

from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"

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

#Typical structure:

/myapp
  |-- app.py
  |-- /templates
  |      |-- home.html
  |-- /static
         |-- style.css
```

38. How do you install Flask on your local machine?

```
[ ]: #Using pip (Python package manager):

pip install Flask

#Create a virtual environment (recommended):

python -m venv venv
source venv/bin/activate      # Linux/macOS
venv\Scripts\activate.bat     # Windows
pip install Flask
```

39. Explain the concept of routing in Flask.

Routing maps a URL path to a specific function (called a view).

Example:

```
[ ]: @app.route('/about')
def about():
    return "This is the About page"
```

You define routes using the @app.route() decorator.

Flask also supports dynamic routes:

[ ]:
```python
@app.route('/user/<username>')
def profile(username):
    return f"Hello, {username}!"
```

40. What are Flask templates, and how are they used in web development?

Templates in Flask (using Jinja2) let you create dynamic HTML with placeholders for variables and logic.

Example (templates/home.html):

[ ]:
```
<h1>Welcome {{ name }}!</h1>
```

[ ]:
```python
#In your Flask app:

from flask import render_template

@app.route("/home")
def home():
    return render_template("home.html", name="Udit")
```

This separates logic (Python) from presentation (HTML), improving code organization and reusabi