PARULINSTITUTEOF ENGINEERING &TECHNOLOGY
FACULTY OF ENGINEERING & TECHNOLOGY
PARULUNIVERSITY

303105257 - Programming in Python with Full Stack Development

RESTful API'S

Computer Science & Engineering

Shaikh Mohd Maaz(Lecturer)

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

Introduction to RESTful APIs and the REST Architectural Style

What is REST

REST (Representational State Transfer) is an architectural style for designing networked applications. It relies on stateless, client-server communication protocols, typically HTTP, to allow systems to interact in a scalable and decoupled manner

A Restful system consists of a:

•A client who requests for the resources.

•server who has the resources.

It is important to create REST API according to industry standards which results in ease of development and increase client adoption.

1.Resource Identification:
•Resources are the main entities of REST.
•Each resource is identified by a unique URI (Uniform Resource Identifier), such as https://api.example.com/users/123.

2.Stateless Communication:
•Each client request contains all the necessary information for the server to fulfill it.
•No client context is stored on the server between requests, ensuring scalability and simplicity.

3.Representation of Resources:
•Resources are represented in formats such as JSON, XML, or HTML.
•A client interacts with the representation of a resource, not the resource itself.

4.Uniform Interface:

•REST defines a consistent set of conventions for interacting with resources:

- •**GET**: Retrieve a resource.
- •**POST**: Create a new resource.
- •**PUT**: Update an existing resource.
- •**DELETE**: Remove a resource.

•These methods ensure simplicity and predictability in API interactions.

5.Stateless and Layered System:

•Each request from the client to the server must include all information required to process the request.

•The system can include intermediate layers (e.g., caching servers, load balancers) that do not alter the client-server interaction.

6.Cacheability:
•Responses from the server must define whether they are cacheable.
•This improves performance and reduces server load.

7.Code on Demand (Optional):
•Servers can extend client functionality by providing executable code (e.g., JavaScript) to the client.

Benefits of RESTful APIs

1.Scalability: Statelessness and caching make it easier to scale systems.
2.Flexibility: Resource representations can be in different formats, supporting diverse clients.
3.Simplicity: Uniform interface makes it easier for developers to understand and implement.
4.Interoperability: Uses standard web protocols, ensuring compatibility with various platforms.

Understanding the HTTP protocol and its role in RESTful APIs

HTTP (Hypertext Transfer Protocol) plays a vital role in API (Application Programming Interface) development as it facilitates communication between clients and servers. Here's an in-depth look at how HTTP is used in API development:

HTTP Methods

HTTP defines various request methods that clients use to interact with servers. Commonly used methods include:

- **GET:** Retrieves data from the server.
- **POST:** Sends data to the server to create a new resource.
- **PUT**: Updates an existing resource on the server.
- **DELETE:** Removes a resource from the server.

```
GET /api/users        // Fetches a list of users
POST /api/users       // Creates a new user
PUT /api/users/123    // Updates user with ID 123
DELETE /api/users/123 // Deletes user with ID 123
```

HTTP Headers

HTTP headers provide additional information about the request or response. In API development, headers are used for various purposes such as authentication, content negotiation, caching, and more. Common headers include:

- **Authorization:** Provides credentials for authentication.
- **Content-Type:** Specifies the format of the data being sent (e.g., JSON, XML).
- **Accept:** Indicates the preferred response format accepted by the client.

```
GET /api/users
Authorization: Bearer <token>
Accept: application/json
```

HTTP Status Codes

HTTP status codes are crucial in API responses to indicate the status of a request. Some common status codes include:

•200 OK: Successful request.

•201 Created: Successful resource creation.

•400 Bad Request: Invalid request from the client.

•401 Unauthorized: Authentication is required.

•404 Not Found: Resource not found on the server.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 123,
  "name": "John Doe"
}
```

HTTP Body

The HTTP body contains the actual content transmitted in requests or replies.

- **Data Formats:** It can be structured as JSON, XML, HTML, or plain text for different purposes.
- **Content-Length:** The Content-Length header specifies the size of the message body in bytes.
- **Schema Compliance:** Developers need to adhere to the API's schema for data consistency and integrity.

```
// HTTP body Syntax

{
  "name": "Raj Veer",
  "email": "rajveer@example.com",
  "age": 30
}
```

HTTP Parameters

Parameters help replace or filter requests or responses, allowing for more flexible interactions.

- **URL Parameters:** Shown in the URL to indicate resource identifiers or query constraints.
- **Query Parameters:** Used in the query string to add criteria or preferences to search queries.
- **Body Parameters:** Sent along with requests for resource creation or modification.

```
POST /api/users
Content-Type: application/json

{
  "name": "Raj",
  "email": "raj@example.com",
  "age": 25
}
```

Parul® University

NAAC ACCREDITED UNIVERSITY A++

Designing and implementing RESTful APIs using common HTTP methods, such as GET, POST, PUT, and DELETE

HTTP Method	Purpose	Example Endpoint	Description
GET	Retrieve data	`/users` or `/users/1`	Fetches a list or a specific resource.
POST	Create a resource	`/users`	Adds a new resource.
PUT	Update a resource	`/users/1`	Replaces a resource entirely.
DELETE	Remove a resource	`/users/1`	Deletes a specific resource.

Step 1: Define the Resource Model
For example, consider a **User** resource:

```json
{
  "id": 1,
  "name": "John Doe",
  "email": "johndoe@example.com",
  "createdAt": "2024-12-29T12:00:00Z"
}
```

Step 2: Create Endpoints for the Resource

GET: Fetch a List or Single Resource

- **Endpoint**: /users
- **Description**: Retrieves all users.
- **Implementation**:

```
# Fetch all users
@app.route('/users', methods=['GET'])
def get_users():
    return jsonify(users), 200
```

- **Endpoint**: /users/{id}
- **Description**: Retrieves a specific user.
- **Implementation**:

```
# Fetch a single user by ID
@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    user = next((u for u in users if u['id'] == id), None)
    if user:
        return jsonify(user), 200
    return jsonify({"error": "User not found"}), 404
```

POST: Create a New Resource

- **Endpoint**: /users
- **Description**: Adds a new user.
- **Implementation**

```
# Add a new user
@app.route('/users', methods=['POST'])
def create_user():
    new_user = request.json
    new_user['id'] = len(users) + 1
    users.append(new_user)
    return jsonify(new_user), 201
```

PUT: Update an Existing Resource

- **Endpoint**: /users/{id}
- **Description**: Updates or replaces the user with the given ID.
- **Implementation**:

```
# Update an existing user
@app.route('/users/<int:id>', methods=['PUT'])
def update_user(id):
    user = next((u for u in users if u['id'] == id), None)
    if user:
        user.update(request.json)
        return jsonify(user), 200
    return jsonify({"error": "User not found"}), 404
```

DELETE: Remove a Resource

- **Endpoint**: /users/{id}
- **Description**: Deletes a user by ID.
- **Implementation**:

```python
# Delete a user
@app.route('/users/<int:id>', methods=['DELETE'])
def delete_user(id):
    global users
    users = [u for u in users if u['id'] != id]
    return jsonify({"message": "User deleted"}), 200
```

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

Using URLs and resource representations to identify and transfer data in RESTful APIs

Designing RESTful URLs

Key Points:

- Use **nouns**, not verbs, in URLs (e.g., /users, not /getUsers).
- Organize URLs in a **hierarchical structure** to show relationships.
- Use **plural nouns** for collections (e.g., /books) and IDs for specific resources (e.g., /books/123).
- Use query parameters for filtering and sorting (e.g., /books?genre=fiction&page=1).

Resource Representations

Key Points:

•Representations transfer the state of a resource between client and server.
•Common formats:

- **JSON** (most popular): Easy to read and parse.
- **XML**: Used in legacy systems.
- **HTML/Text**: For human-readable outputs.

```json
{
  "id": 1,
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald"
}
```

Transferring Data via HTTP

Key Points:

- **GET**: Retrieve resources (e.g., /books).
- **POST**: Create a new resource (e.g., /books).
- **PUT**: Update a resource (e.g., /books/123).
- **DELETE**: Remove a resource (e.g., /books/123).

Implementing best practices for designing and implementing RESTful APIs

1. Using HTTP Status Codes

HTTP status codes communicate the result of a client's request. They help developers understand the outcome without parsing responses.

Code	Category	Description
200	**OK**	Request succeeded (e.g., data fetched).
201	**Created**	Resource created successfully.
204	**No Content**	Successful operation, no content.
400	**Bad Request**	Invalid input or missing parameters.
401	**Unauthorized**	Authentication required.
403	**Forbidden**	Client lacks permission.
404	**Not Found**	Resource does not exist.
500	**Internal Server Error**	↓ ver-side error occurred.

2. API Versioning

API versioning ensures backward compatibility and allows developers to update or deprecate features without breaking existing clients.

Strategies for Versioning

1.URI Versioning (Most Common):
- Example: /v1/users

2.Header Versioning:
- Example: Add a custom header like API-Version: 1.

3.Query Parameter Versioning:
- Example: /users?version=1.

Error Handling

Proper error handling improves the developer experience by providing clear and actionable feedback.

Best Practices

1.Use appropriate HTTP status codes for errors.

2.Return detailed but non-sensitive error messages.

3.Include a consistent error structure.

Parul® University
NAAC A++ ACCREDITED UNIVERSITY

Consuming RESTful APIs using common tools and libraries

1. Introduction to RESTful API Consumption

•RESTful APIs allow clients to interact with server-side resources using HTTP methods like **GET**, **POST**, **PUT**, and **DELETE**.

•Common tools and libraries for API consumption:

- **cURL**: Command-line tool for making HTTP requests.
- **Postman**: GUI-based tool for testing and automating API calls.
- **Requests Library (Python)**: Python package for sending HTTP requests programmatically.

2. Using cURL

What is cURL?

- A command-line tool to send HTTP requests.
- Pre-installed on many operating systems

Examples

1. GET Request

```bash
curl -X GET "https://api.example.com/books/1"
```

Response:

```json
{
  "id": 1,
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald"
}
```

2. **POST Request**

```bash
curl -X POST "https://api.example.com/books" \
     -H "Content-Type: application/json" \
     -d '{"title": "1984", "author": "George Orwell"}'
```

Response:

```json
{
  "id": 2,
  "message": "Book created successfully"
}
```

3. Using Postman

What is Postman?

•A GUI-based API client for testing, debugging, and automating API calls.

Steps to Use Postman

1.Install Postman: Download from Postman.

2.Create a Request:

- •Select HTTP method (GET, POST, etc.).
- •Enter the API endpoint URL.
- •Add headers (e.g., Authorization: Bearer <token>).
- •Provide a request body for POST/PUT requests.

3.Send the Request: Click **Send** to execute.

4.Inspect Response:

- •View the response body, status code, and headers.
- •Use response to verify functionality.

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

THANK YOU

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversi