RESTFUL WEB SERVICES

REKHA RANI

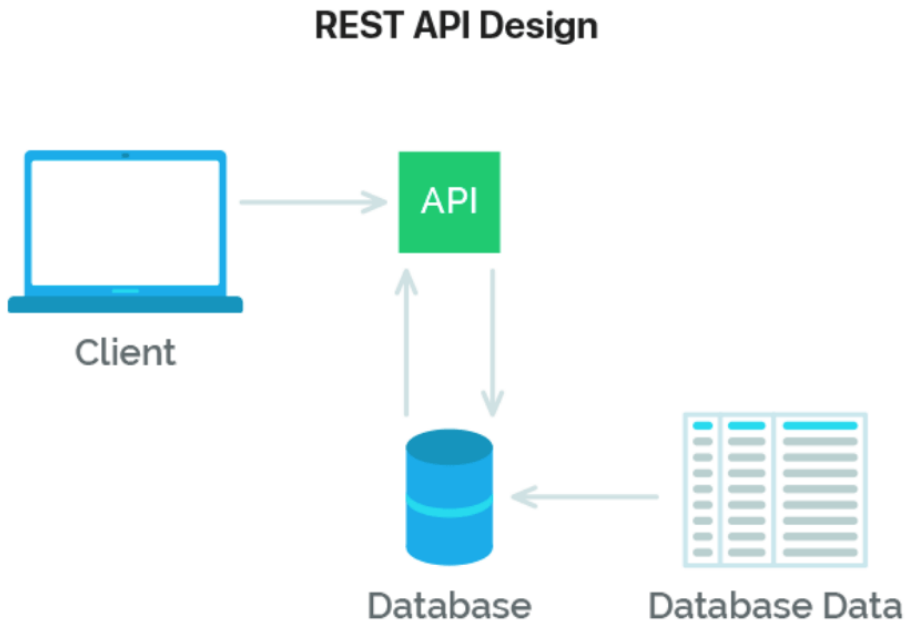
CONTENTS

- Introduction to RESTful APIs
- Why REST Matters
- Principles of REST
- HTTP Methods
- RESTful URI Design
- Request and Response
- Status Codes
- Best Practices
- Tools and Libraries
- Case Studies
- Demonstration - Applying REST Concepts
- Conclusion

INTRODUCTION TO RESTFUL APIS

- **Origin:** REST was introduced by Roy Fielding in his 2000 doctoral dissertation.
- **Purpose:** Addressing scalability and simplicity challenges in distributed systems.
- **Evolution:** Became a standard for designing networked applications and web services.
- **Definition**
  - *Concept by Roy Fielding:* REST stands for Representational State Transfer, a set of architectural principles for designing networked applications.
  - *Resource-Oriented:* Emphasizes resources (data or services) that are identified by URIs and can be manipulated using standard methods.
- **Key Characteristics:**

Stateless, Client-Server Architecture, Catchability, Uniform Interface, Layered System, Code on Demand (Optional)

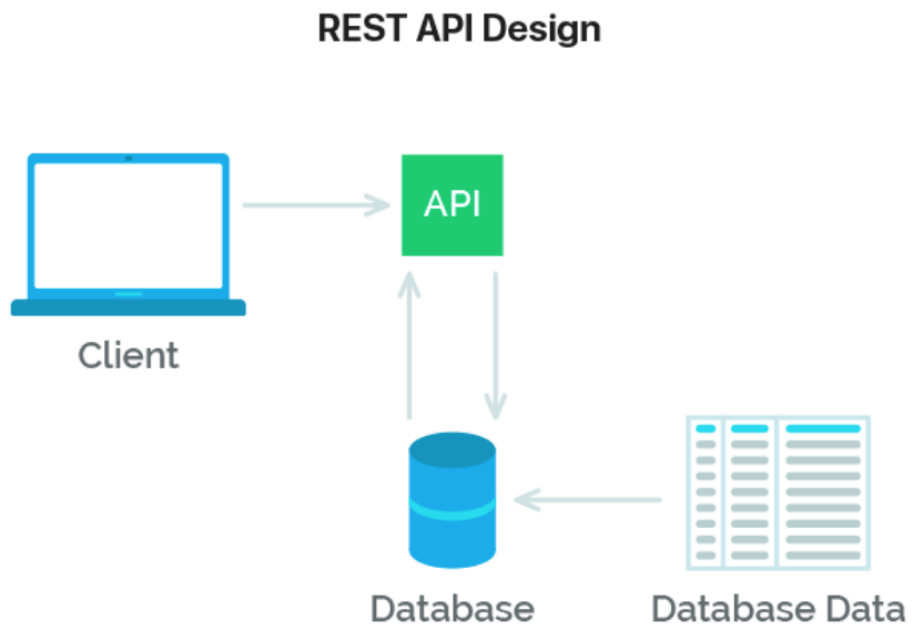


WHY REST MATTERS

- Simplicity
- Scalability

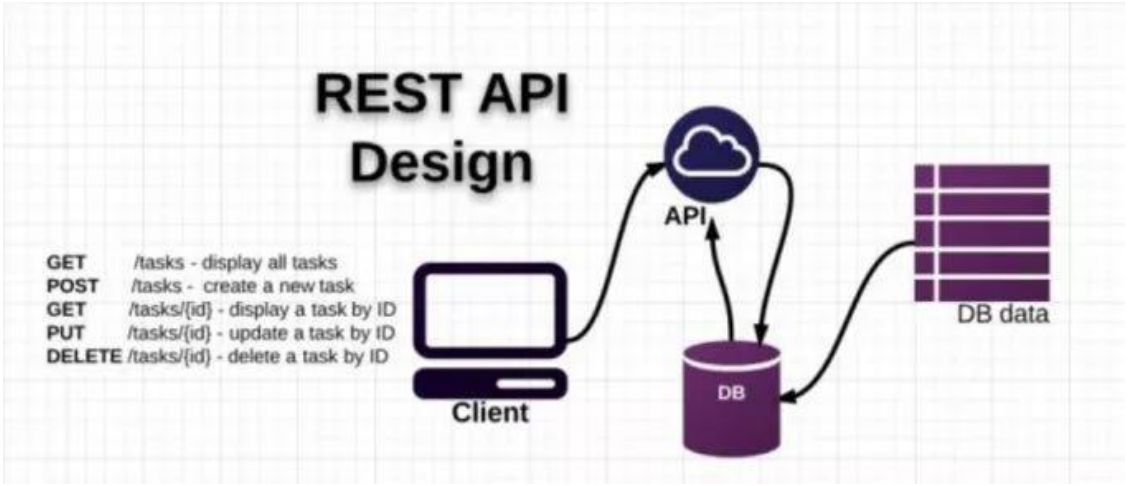
- Prevalence in Web Development and Integration
  1.Web Development:
  2. Integration:

- Key Takeaways
  - Developer-Friendly
  - Scalability
  - Versatile Integration



PRINCIPLES OF REST

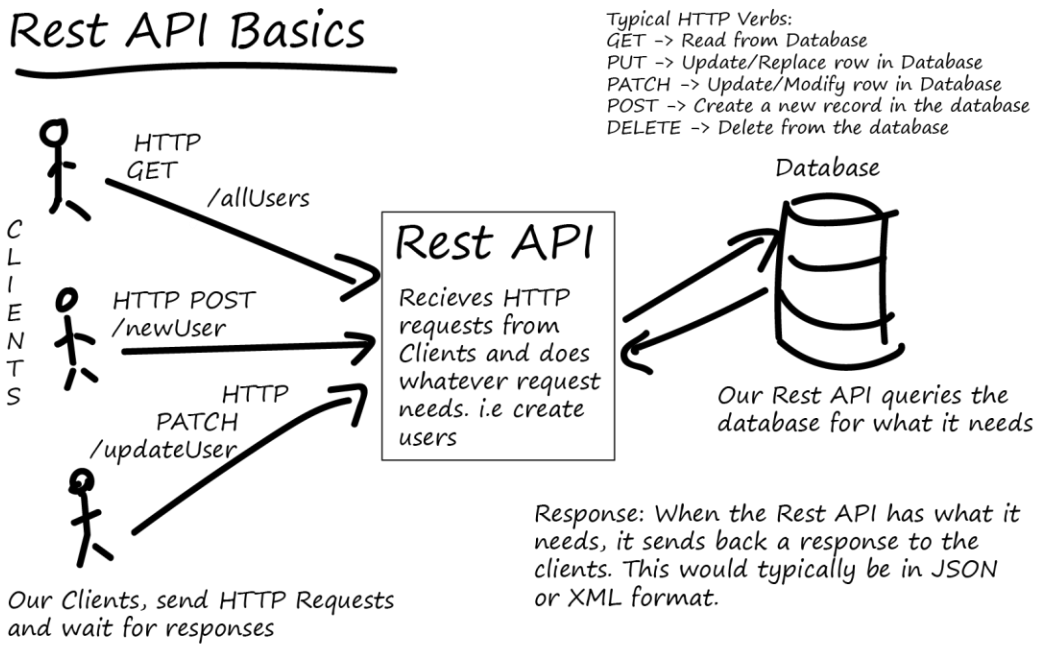
- Stateless: Each request from a client to a server must contain all the information needed to understand and fulfil the request.
- Client-Server: Separation of concerns between the client and server.
- Uniform Interface: Consistent and uniform way to interact with resources.



PRINCIPLES OF REST

The Core Principles and Significance in Modern Development

- REST is not just a set of rules; it's a guiding architectural style for designing networked applications. At its core, REST is about simplicity, scalability, and statelessness.
- RESTful APIs play a pivotal role in building applications that are not only scalable but also maintainable over time.
- By adhering to REST principles, developers can create APIs that are intuitive, flexible, and adhere to standards.
- One of the key strengths of REST lies in its ability to enable seamless communication between different software systems.
- Through standard HTTP methods, such as GET, POST, PUT, and DELETE, REST APIs facilitate the transfer of data and actions across distributed environments.



HTTP METHODS

- GET: Retrieve a resource
- POST: Create a new resource
- PUT: Update an existing resource
- DELETE: Remove a resource
- Other methods: PATCH, OPTIONS, HEAD, etc.

REST : HTTP Methods

SQL	REST
SELECT	GET
INSERT	POST
UPDATE	PUT
DELETE	DELETE

HEAD : get meta-data

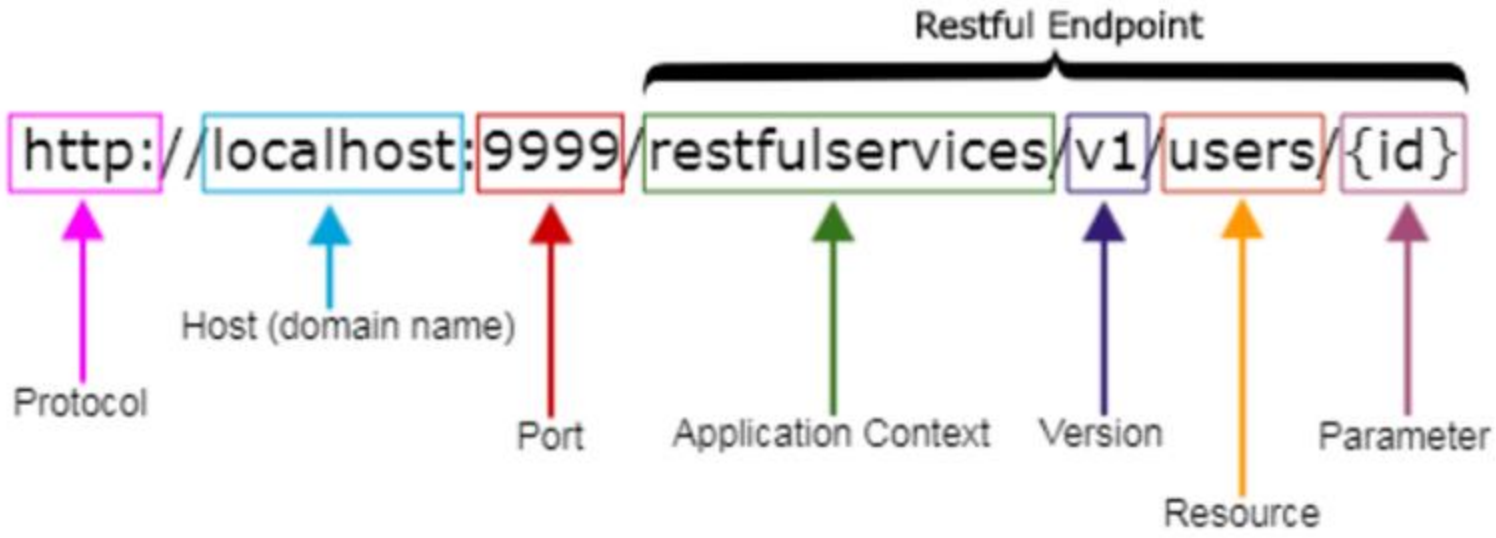
OPTIONS : to get details about a resource

TRACE : used to debug proxies

CONNECT : Forward some other protocol through HTTP proxy

RESTFUL URI DESIGN

- Meaningful and resource-oriented URIs
- Use of nouns to represent resources
- Hierarchy and relationships in URIs
- Examples of good and bad URI design



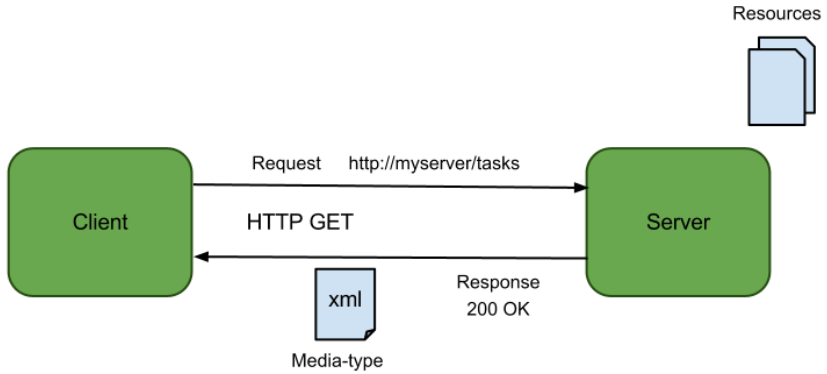
REQUEST AND RESPONSE

- ➢ Structure of a RESTful request
- ❖ Components:
- HTTP Method
- Headers
- Body (Optional

- ➢ Structure of a RESTful response
- ❖ Components
- Status Code
- Headers
- Body (Optional

- ➢ Headers and their significance
- ❖ Common Headers:
- Content-Type
- Authorization
- Accept

- **Example Request:**
  - **HTTP Method:** GET
  - **URI: /users/123**
  - **Headers:**
    - Accept: application/json
    - Authorization: Bearer [token]
- **Example Response:**
  - **Status Code:** 200 OK
  - **Headers:**
    - Content-Type: application/json
    - Cache-Control: max-age=3600




```
{
  "userId": 123,
  "username": "john_doe",
  "email": "john@example.com"
}
```

STATUS CODES

- Overview of HTTP status codes

HTTP status codes are three-digit numbers that indicate the outcome of a request-response between a client and a server.
Categories: 1xx (Informational, 2xx (Success), 3xx (Redirection), 4xx (Client Error),5xx (Server Error)

- Common status codes:
  (200 OK, 201 Created, 204 No Content, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error)

- When and how to use different status codes:

•200 Series: Indicate success. Use these when the request was successful.
•300 Series: Indicate redirection. Use when the client needs to take additional action.
•400 Series: Indicate client errors. Use when the request cannot be fulfilled due to client-side issues.
•500 Series: Indicate server errors. Use when the server fails to fulfill a valid request.

BEST PRACTICES

- Use of versioning
- Pagination
- Error handling and reporting
- Security best practices (HTTPS, authentication, authorization)
- Caching strategies

TOOLS AND LIBRARIES

- Popular tools and libraries for building and consuming RESTful APIs:
- Purpose: Automated API documentation generation.
- Features:
  - Describes API endpoints, request/response formats, and authentication.
  - Interactive documentation for easy testing and exploration.

- Swagger/Open API for documentation Postman for testing Client:
- Purpose: Comprehensive API development environment.
- Features:
  - Allows sending requests to API endpoints and inspecting responses.
  - Supports automated testing with pre-defined scripts.
  - Collaboration features for teams.

- Libraries in various programming languages:
- Purpose: Simplify API consumption for developers in different languages.
- Benefits:
  - Abstracts away low-level HTTP details.
  - Provides native language support for making API requests.
  - Reduces development time and potential errors.

CASE STUDIES

Real-world examples of successful RESTful APIs:

- **1. Twitter API**
  - Use Case: Enables developers to access Twitter data and functionality.
- **2. GitHub API**
  - Use Case: Allows developers to interact with GitHub repositories and user data.
- **3. Google Maps API**
  - Use Case: Provides mapping and location-based services.

CASE STUDIES

Learnings from Popular APIs

- **Consistent Resource Naming:** Successful APIs have clear and consistent resource naming conventions.
- **Effective Rate Limiting:** Implementing rate limiting to prevent abuse while providing fair access to resources.
- **Comprehensive Documentation:** Well-documented APIs with clear examples and use cases.
- **Versioning Strategies:** Thoughtful versioning to maintain backward compatibility and ease transitions.

DEMONSTRATION - APPLYING REST CONCEPTS

- **Tool Selection:** Postman
- **Scenario:** Retrieving User Information
- **GitHub Repository:**

CONCLUSION

- Recap of key points
- Importance of RESTful APIs in modern development
- Encouragement for further exploration

THANK YOU

SOMEONE@EXAMPLE.COM