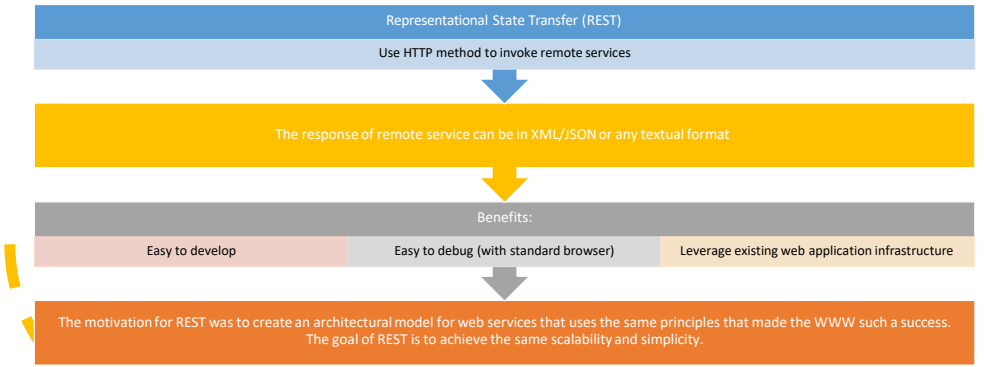
REST Constraints

Ahmad Hamo
24-3-2-25

REST API (also known as RESTful API)

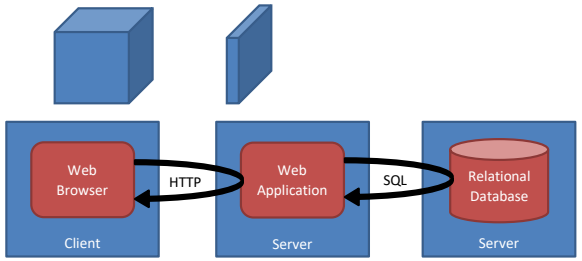


What is REST?

An architectural style for *distributed hypermedia systems* described by **Roy Thomas Fielding** in his doctoral dissertation 2000.

Consists of constraints:

1. Client - Server
2. Stateless
3. Cache
4. Uniform Interface
5. Layered System
6. Code-On-Demand



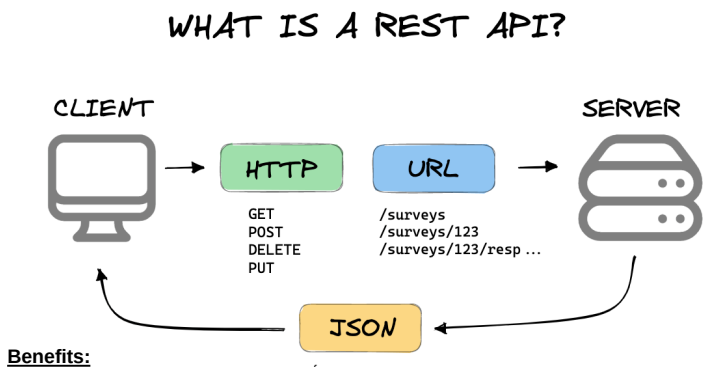
What does REST mean?

The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting ***links*** *(state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.*

From Roy's dissertation.

What does “Representational” mean in REST?

- When interacting with resources over a network (such as the web), clients and servers do not directly exchange the actual resource itself. Instead, they exchange **representations** of those resources.
- A **representation** is a concrete depiction or encoding of the state of a resource at a particular moment in time.
- **User object** isn't sent across the network. Instead, the server sends a **representation** of that user in a specific format (like JSON or XML).



- The motivation for REST was to create an architectural model for web services that uses the same principles that made the WWW such a success.
- The goal of REST is to achieve the same scalability and simplicity.

REST & SOAP

- Representational state transfer (REST) is a set of architectural principles.
- Simple object access protocol (SOAP) is an official protocol maintained by the World Wide Web Consortium (W3C).
- The main difference is that SOAP is a **protocol** while REST is not.
- Typically, an API will adhere to either REST or SOAP, depending on the use case and preferences of the developer.
- In general, REST **is easier** to use for the most part and is **more flexible**.
- It has the following advantages when compared to SOAP:
  - Uses easy to understand standards like swagger and OpenAPI Specification 3.0
  - Smaller learning curve
  - Efficient (SOAP uses XML for all messages, REST mostly uses smaller message formats like JSON)
  - Closer to other Web technologies in design philosophy

REST

The basic REST HTTP requests methods or **verbs** which are commonly used POST, GET, PUT, and DELETE. In addition to HEAD, OPTIONS, TRACE and PATCH requests as well. Example from the Swagger Pet Store API:

- Sending a GET request to /pet/{petId} would retrieve pets with a specified ID from the database.
- Sending a POST request to /pet/{petId}/uploadImage would add a new image of the pet.
- Sending a PUT request to /pet/{petId} would update the attributes of an existing pet, identified by a specified id.
- Sending a DELETE request to /pet/{petId} would delete a specified pet.

GET	Read or retrieve data
POST	Add new data
PUT	Update data that already exists
DELETE	Remove data

- **Swagger** is an Interface Description Language for describing RESTful APIs expressed using JSON/ YAML . Swagger is used together with a set of open-source software tools to design, build, document, and use RESTful web services. Swagger includes automated documentation, code generation, and test-case generation. Wikipedia
- **YAML** is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted. Wikipedia

HTTP response status codes

1.Informational responses (100–199)

102 Processing (WebDAV) This code indicates that the server has received and is processing the request, but no response is available yet.

Successful responses (200–299)

200 OK The request succeeded.

3.Redirection messages (300–399)

301 Moved Permanently The URL of the requested resource has been changed permanently. The new URL is given in the response.

4.Client error responses (400–499)

401 Unauthorized Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

5.Server error responses (500–599)

500 Internal Server Error The server has encountered a situation it does not know how to handle.

SOAP vs. REST

- **SOAP is a protocol**, meaning it has strict rules for message structure, security, and transactions.
- **REST is an architectural style**, meaning it provides guidelines rather than strict rules.

Feature	SOAP	REST
Type	Protocol	Architectural style
Message Format	XML	JSON, XML, plain text, etc.
Transport Protocol	HTTP, SMTP, TCP, etc.	HTTP (mainly)
Complexity	More complex (strict standards)	Simpler and lightweight
Performance	Slower (due to XML overhead)	Faster (uses JSON, less overhead)
Security	Built-in security (WS-Security)	Uses HTTPS and OAuth for security
Statefulness	Can be stateful	Stateless by default
Best Used For	Enterprise applications, banking, transactions	Web services, mobile apps, public APIs

Which One to Use?

- Use **SOAP** for **high-security applications** like banking and payment processing.
- Use **REST** for **web services, mobile apps, and lightweight applications**.

REST API Architectural Constraints

- **Cacheable:** Well-managed caching partially or eliminates some client-server interactions, further improving scalability and performance. In REST, caching shall be applied to resources when applicable, and then these resources **MUST** declare themselves cacheable. Caching can be implemented on the server or client-side.

- **Client-Server**: This constraint essentially means that client applications and server applications **MUST** be able to evolve separately without any dependency on each other. A client should know only resource URIs, and that's all. Today, this is standard practice in web development, so nothing fancy is required from your side. Keep it simple.

  Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered.

REST Architectural Constraints (restfulapi.net)

REST API Architectural Constraints

- **Uniform Interface:** It is a key constraint that differentiate between a REST API and Non-REST API. Once a developer becomes familiar with one of your APIs, he should be able to follow a similar approach for other APIs. (show rest-examples)

- **Stateless:** It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session.

  If the client application needs to be a stateful application for the end-user, where the user logs in once and does other authorized operations after that, then each request from the client should contain all the information necessary to service the request – including authentication and authorization details.

  Question: What if one of the server instances goes down?
  Answer: Since the server is stateless, it does not store any session data. This means that if one server instance goes down, the other instances can still handle incoming requests because they do not rely on shared session data.

REST Architectural Constraints (restfulapi.net)

REST API Architectural Constraints

- **Layered system:** REST allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client cannot ordinarily tell whether it is connected directly to the end server or an intermediary along the way.

- It allows for a system to be composed of multiple layers, each with a specific role or responsibility. The client interacts with the system **without** necessarily knowing the internal structure or how many intermediary layers are involved in processing the request.

REST Architectural Constraints (restfulapi.net)

Benefits of a Layered System in REST

- **Improved Scalability :**
  - By separating concerns into different layers, you can scale each component independently.
- **Enhanced Security :**
  - Layers can provide additional security measures. For instance, an intermediary layer could filter malicious requests before they reach the core application logic.
- **Flexibility and Maintainability :**
  - Since each layer has a specific responsibility, changes in one layer (e.g., switching to a new database system) do not necessarily affect other layers.
- **Fault Tolerance :**
  - If one layer fails (e.g., a database server goes down), other layers may still continue to function. For example, a caching layer could serve stale data while the database is being restored.
- **Performance Optimization :**
  - Layers like caching servers or content delivery networks (CDNs) can store frequently accessed data closer to the client, reducing latency and improving performance.

REST API Architectural Constraints

- **Code on demand (Optional):** is an optional REST constraint that allows servers to send executable code (e.g., JavaScript) to clients, enabling dynamic functionality, but it is **rarely** used in modern APIs due to **security concerns**.
- The examples of code on demand may include the compiled components such as **Java applets**, **UI widget** rendering code and **client-side scripts** such as JavaScript.

REST Architectural Constraints (restfulapi.net)

REST API Architectural Constraints

- **All** these constraints help you build a **truly RESTful API**, and you should follow them.
- Still, at times, you may find yourself **violating one or two** constraints. Do not worry; you are still making a RESTful API – but **not "truly RESTful."**

Safe Methods

Request methods are considered ***safe*** if their defined semantics are essentially **read-only**. The client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource.

The GET, HEAD, OPTIONS, and TRACE methods are considered safe methods. As per HTTP specification, the GET and HEAD methods should be used only for retrieval of resource representations – and they do not update/delete the resource on the server.

The **purpose of distinguishing between safe and unsafe methods** is to allow automated retrieval processes (spiders) and cache performance optimization (pre-fetching) to work without fear of causing harm.

Safe methods allow user agents to represent other methods, such as *POST, PUT and DELETE,* in a unique way so that the user is made aware of the fact that a possibly unsafe action is being requested – and they can update/delete the resource on the server and so should be used carefully.

HTTP Methods - REST API Tutorial (restfulapi.net)

Safe Idempotent Methods

The term **idempotent** is used more comprehensively to describe an **operation that will produce the same results if executed once or multiple times**.

In HTTP specification, the **PUT, DELETE and safe methods (GET, HEAD, OPTIONS, TRACE) are idempotent methods**.

Idempotence is a handy property in many situations, as it means that an operation can be repeated or retried as often as necessary without causing unintended effects.

With non-idempotent operations, the algorithm may have to keep track of whether the operation was already performed or not.

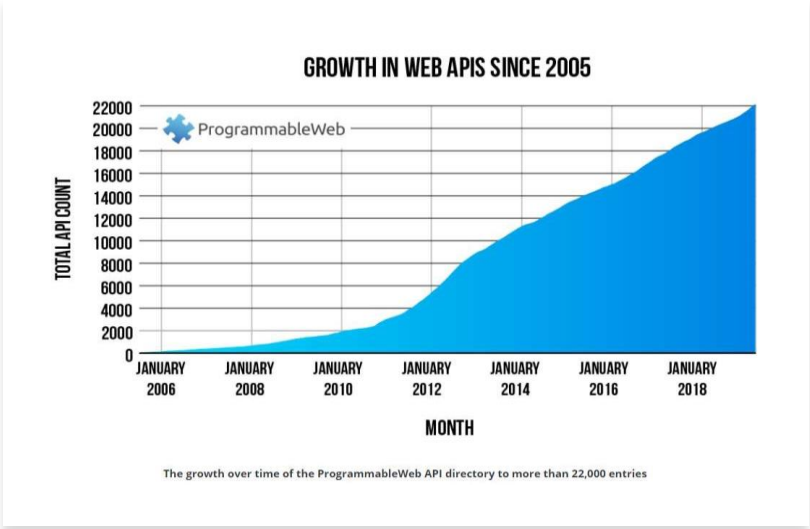
POST is not an idempotent method since calling it multiple times may result in incorrect updates. Usually, POST APIs create new resources on the server. If POST/payment endpoint is called with an identical request body, you will create multiple records. **To avoid this, you must have your own custom logic preventing duplicate records.**

HTTP Methods - REST API Tutorial (restfulapi.net)

HTTP response codes

Must read the following resources:

1 HTTP Methods - REST API Tutorial (restfulapi.net)
2 https://restfulapi.net/http-status-codes/
3 https://blog.hubspot.com/website/idempotent-api



XML APIs vs. JSON APIs

OpenAPI Specification (OAS)

- The OpenAPI Specification (OAS) defines a **standard**, **language-agnostic** **interface to HTTP APIs** which allows both humans and computers to discover and understand the capabilities of the service **without** access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.
- An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

You must read and try following (we will use OpenAPI V 3.1.0):

1 **Specification:** https://spec.openapis.org/oas/v3.1.0
2 **Example:** https://petstore3.swagger.io/
3 **Swagger editor:** **https://editor-next.swagger.io/** (go to File->Load Example menu)

Samples

Online REST API for Testing and Prototyping | GO REST

Dummy sample rest api - dummy.restapiexample.com

SOAPUI/ReadyAPI demo

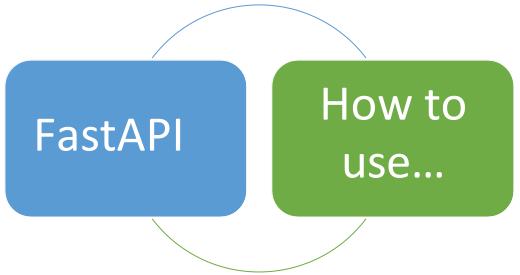
Watch both videos on the following link:

Download Free ReadyAPI Trial | SoapUI

Tools

- MySQL database
- Docker which used to package and run the application with its dependencies inside container.
- SoupUI and Postman
- Git
- Create your account on
  - https://github.com/
  - https://hub.docker.com/

Next



First Assignment

Details will be sent separately.

References

- RESTful Web APIs: Services for a Changing World, By Leonard Richardson, Mike Amundsen, Sam Ruby 2020.
- https://www.w3.org/
- https://developer.mozilla.org/
- Web server vs. Application server (educative.io)
- The Next Dimension of Enterprise Computing, Dr. Billy B. L. Lim, School of Information Technology, Illinois State University
- SOAP vs REST APIs: Which Is Right For You? | SoapUI
- REST vs. SOAP (redhat.com)