**WEB SERVICES**

420-511-VA

#### **LAB-3-**

REST Architectural Style

We have looked into multiple elements that are used to establish client server communication, namely:

* Web server
* Client (involving Browser, or client-side code)
* HTTP as a transfer protocol
* HTTP Request and Response with a way to specify the resource and return the status of request processing
* HTTP headers where additional information can be communicated
* HTTP body to transfer the resource
* Hypermedia which represents a resource and points to other resources
* URL to locate resources
* PHP as a back-end programming language

We have also identified that in order to make the access to the web service standard across clients we need to:

* Define URL as the entry points, known as endpoints, of the API. Meaning the URLs that the clients should use to access the service?
* Define the actions that can be taken by the web service.
* Define the format of the request that can be accepted by the web service and the format of the response that is returned by the webservice.
* Define the acceptable data formats as part of the request or that will be provided in the response.

Here comes the role of architectural styles with guidelines on how to use these elements to communicate resources between the client and the server and address the points that we have identified for building the API.

We will next look into REST.

**PART-1-**

Go through the material on the following link and provide answers to the questions below:

**http://www.xfront.com/sld003.htm**

The student is supposed to give a brief and comprehensive answer in his/her own words.

1. Who suggested the REST web services architectural style?

|  |
| --- |
| Roy Fielding suggested the REST web services architectural style. |

1. What do each of the Representation, State and Transfer keywords stand for or represent?

|  |
| --- |
| * Representation: represents the resource that is returned after the client references a Web resource using a URL. * State: represents the state of the client application after a representation of a resource is returned. * Transfer: represents the client application changes states with each resource representation |

1. The readings refers to a state machine, what is a state machine (not defined in the readings)?

|  |
| --- |
| A state machine refers to the concept that describes a machine that has a set of states. |

1. What was the motivation behind REST?

|  |
| --- |
| “The motivation for developing REST was to create an architectural model for  how the Web should work, such that it could serve as the guiding framework  for the Web protocol standards.” |

1. Is REST a Standard?

|  |
| --- |
| No, because W3 doesn’t put out a REST specification. REST is just an architectural style that people could use to design their Web services. |

1. In what format does the material suggest that the REST returns the data?

|  |
| --- |
| XML/HTML/GIF/JPEG/etc (Resource Representations), text/xml, text/html, image/gif, image/gif, image/jpeg, etc(Resource Types, MIME Types) |

1. From what the material describes if we want a REST URL for a Parts data to be returned in JSON what would be the format of the URL?

|  |
| --- |
| http://www.parts-depot.com/parts?flavor=json |

1. What technique is the most commonly used for client server interaction in REST web service PULL or PUSH?

|  |
| --- |
| A pull-based interaction style: consuming components pull representations. |

1. What are the characteristics of a REST-based network?

|  |
| --- |
| Client-server: a pull-based interaction style, stateless, cache, uniform interface, named resources and interconnected resource representations. |

1. Is REST stateless? What does that mean?

|  |
| --- |
| Yes, it means that each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. |

1. In practical implementations which service architectural style is implemented using a single entry point SOAP or REST?

|  |
| --- |
| SOAP, because it can use the same URL/single for all transactions. |

1. In what service architectural style, the semantics is hidden from HTTP and what style relies on HTTP? REST or SOAP? and How, justify your response?

|  |
| --- |
| In the SOAP service architectural style, the semantics is hidden from HTTP. REST relies on the HTTP because the HTTP method in the REST identifies the desired operation (e.g., HTTP GET). With SOAP, both the targeted resource, as well as the requested method is nested within the SOAP envelope, thus making it much more difficult for intermediaries (e.g., proxy server) to do their job. |

1. In what architectural style is it easier to use a proxy server that can act on the semantics of the service requests REST or SOAP? and why, justify your response?

|  |
| --- |
| In the REST architectural style, it is easier to use a proxy server that can act on the semantics of the service request because the resource it is targeting is not hidden inside like SOAP. With SOAP, it will require the proxy server to understand the semantics of every SOAP application. |

1. Could caching be used with REST? What is cashed?

|  |
| --- |
| Yes, caching can be used with REST. The data of resource/server response is cached. |

1. Does REST conform to the semantic web?

|  |
| --- |
| Yes, because with REST style, every resource has its own logical URI. |

1. What are the standards that REST advocates for?

|  |
| --- |
| The standards that REST advocates for are:   * Addressing and naming resources: URI * Generic resource interface: HTTP GET, POST, PUT, DELETE * Resource representations: HTML, XML, GIF, JPEG, etc * Media Types: MIME tyupes(text/html, text/plain, etc) |

1. What does the Payload refer to?

|  |
| --- |
| Payload refers to the body in the request and response message. |

1. With REST, in the context of the material, does the client have to be able to process payload in XML?

|  |
| --- |
| Yes, because client applications need to know the semantics of the data. Client applications needs to understand request/response payload. |

**PART-2-**

In this part of the LAB you will go through some of the API development guidelines and take notes of certain aspects that should be considered when developing an API in general or specifically a RESTful API.

#### There is no implementation required, go through the following articles by Microsoft and write one statement for each guideline, briefly summarizing the guideline, using sections and subsections where needed:

**RESTful web API design**

* <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design>
* For this article include the whole table for the section: “Define API operations in terms of HTTP methods”

**API Implementation**

* <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-implementation>
* For this article you don’t need to discuss the section: “Using Azure API Management”

**RESTful web API design**

**What is REST?**

* REST (Representational State Transfer) is an architectural style for designing web services that is proposed by Roy Fielding.
  + Main design principles of RESTful APIs using HTTP:
    - Designed around resources
    - A resource has an identifier (URI)
    - Clients interact with a service by exchanging representations of resources
    - Uniform interface (GET, POST, PUT, PATCH, and DELETE)
    - Stateless request model
    - Driven by hypermedia links that are contained in the representation

**Organize the API design around resources**

* Resource URIs should be based on nouns (the resource) and not verbs (operations on the resource)
* Avoid creating APIs that mirror the internal structure of a database
* Use plural nouns for URIs that reference collections.
* Avoid requiring resource URIs more complex than collection/item/collection
* Avoid web APIs that expose a large number of small resources
* Think of web API as an abstraction of the database.
* It might not be possible to map every operation implemented by a web API to a specific resource.

**Define API operations in terms of HTTP methods**

* GET retrieves a representation of the resource at the specified URI.
* POST creates a new resource at the specified URI
* PUT either creates or replaces the resource at the specified URI
* PATCH performs a partial update of a resource
* DELETE removes the resource at the specified URI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resource | POST | GET | PUT | DELETE |
| /customers | Create a new customer | Retrieves all customers | Bulk update of customers | Remove all customers |
| /customers/1 | Error | Retrieve the details for customer 1 | Update the details of customer 1 if it exists | Remove customer 1 |
| /customers/1/orders | Create a new order for customer 1 | Retrieve all orders for customer 1 | Bulk update of orders for customer 1 | Remove all orders for customer 1 |

Difference between POST, PUT and PATCH

* **POST** request creates a resource
* **PUT** request creates a resource or updates an existing resource
* **PATCH** request performs a partial update to an existing resource.

**Conform to HTTP semantics**

Conforms to HTTP semantics includes Media Types/MIME types, GET methods, POST methods, PUT methods, PATCH methods, and DELETE methods

**Asynchronous operations**

Consider making an operation asynchronous when a POST, PUT, PATCH, or DELETE operation might require processing that takes a while to complete.

**Filter and paginate data**

When only a subset of information is required, passing a filter in the query string of the URI can help limit the amount of data returned by any single request.

**Support partial responses for large binary resources**

A resource that contains large binary fields, such as files or images can enable resources to be retrieved in chunks by having the web API support the Accept-Ranges header for GET requests for large resources. Also, HTTP HEAD requests (similar to GET request except it only returns the HTTP headers that describes the resource with an empty message body) to determine whether to fetch a resource by using partial GET requests.

**Use HATEOAS to enable navigation to related resources**

HATEOAS is a principle that views the system as a finite state machine, and that the response to each request contains the information necessary to move from one state to another; no other information should be necessary.

**Versioning a RESTful web API:**

* **Types of versioning:**
* No versioning: adding content to existing resources might not present a breaking change as client applications that are not expecting to see this content will ignore it.
* URI versioning: add a version number to the URI for each resource such as https://adventure-works.com/v2/customers/3
* Query string versioning: specify the version of the resource by using a parameter within the query string appended to the HTTP request, such as https://adventure-works.com/customers/3?version=2
* header versioning: implement a custom header that indicates the version of the resource. As example,

GET https://adventure-works.com/customers/3 HTTP/1.1

Custom-Header: api-version=1

* media type versioning: when a client application sends an HTTP GET request to a web server it should stipulate the format of the content that it can handle by using an Accept header (allow the client application to specify whether the body of the response should be XML, JSON, or some other common format that the client can parse). As example, GET https://adventure-works.com/customers/3 HTTP/1.1

Accept: application/vnd.adventure-works.v1+json

**Open API Initiative:** standardizes REST API descriptions across vendors

**Web API Implementation**

* **Processing requests**
  + Code that implements GET, PUT, DELETE, HEAD, and PATCH requests should not impose any side-effects. The same request repeated over the same resource should result in the same state
  + POST actions that create new resources should not have unrelated side-effects. The effects of the request should be limited to the new resource (and any directly related resources that forms a linkage).
  + Avoid implementing chatty POST, PUT, and delete operations
    - POST request can contain the details for multiple new resources and add them all to the same collection.
    - PUT request can replace the entire set of resources in a collection
    - DELETE request can remove an entire collection
  + Follow the HTTP specification when sending a response
    - A web API must return messages that contain the correct HTTP status code, appropriate HTTP headers, and a formatted body.
  + Support content negotiations

The body of a response message may contain data in different formats (media types). Use Accept header to lists the media types that the client can handle and use Content-Type header to specify the format of the result.

* + Provide links to support HATEOAS-style navigation and discovery of resources
    - The HATEOAS approach allows clients to navigate and discover resources from their initial starting point which is accomplished using links containing URIs.
* **Handling EXCEPTIONS**
  + Capture exceptions and return a meaningful response to clients

If an exception makes it impossible to complete the operation successfully, the exception can be passed back in the response message, but it should include a meaningful description of the error that caused the exception.

* + Handle exceptions consistently and log information about errors

Consider implementing a global error handling strategy across the entire web API and incorporate error logging which captures the full details of each exception.

* + Distinguish between client-side errors and server-side errors

HTTP protocol distinguishes between errors that occur due to the client application (the HTTP 4xx status codes), and errors that are caused by a mishap on the server (the HTTP 5xx status codes).

* **Optimizing client-side data access**
  + Aim to minimize the amount of traffic that flows across the network through the help of the following points when implementing the code to retrieve and maintain data:
    - Support client-side caching
      * HTTP 1.1 protocol supports caching in clients and intermediate servers through which a request is routed using the Cache-Control header.
    - Provide Etags to optimize query processing
      * When a client application retrieves an object, the response message can also include an ETag (Entity Tag), an opaque string that indicates the version of a resource.
    - Use Etags to Support Optimistic Concurrency
      * HTTP protocol supports an optimistic concurrency strategy to enable updates over previously cached data. If, after fetching and caching a resource, the client application subsequently sends a PUT or DELETE request to change or remove the resource, it should include in If-Match header that references the ETag.
* **Handling large requests and responses**
  + When a client application needs to issue requests that send or receive data that may be several megabytes (or bigger) in size, consider the following points:
    - Optimize requests and responses that involve large objects
    - Implement partial responses for clients that do not support asynchronous operations
    - Avoid sending unnecessary 100-Continue status messages in client applications
    - Support pagination for request that may return large numbers of objects
* **Maintaining responsiveness, scalability, and availability**
  + The same web API might be used by many client applications running anywhere in the world, so it is important to ensure that the web API is responsive, scalable and available which is determined by meeting the following requirements:
    - Provide asynchronous support for long-running requests
    - Ensure that each request is stateless
    - Track clients and implement throttling to reduce the chances of DOS attacks
    - Manage persistent HTTP connections carefully
* **Publishing and managing a web API**
  + To make a web API available for client applications, the web API must be deployed to a host environment which is typically a web server.
* **Testing a web API**
  + A web API should be tested as thoroughly as any other piece of software, and this is done by creating unit tests to validate the functionality.
* **Supporting client-side developers**
  + Document the REST operations for a web API
  + Implement a client SDK: provide an SDK that wraps the REST interface and abstracts low-level details inside a more functional set of methods.
* **Monitoring a web API**
  + Monitoring a web API directly
    - if web API is implemented using the ASP.NET Web API template and Visual Studio 2012, you can gather availability, performance, and usage data by using ASP.NET Application Insights (a package that transparently tracks and records information about requests and responses when the web API is deployed to the cloud).
  + Monitoring a web API through the API Management Service
    - if web API is published using the API Management service, a package that transparently tracks and records information about requests and responses when the web API is deployed to the cloud.