RESTful Web Services

Lecture 10

Assignment Two

- Assignment 2 is at 5pm, Friday, Week 13 (21/5/2021)
  - The assignment involves development of a Web application
  - You **must** **re-use and develop your Node.js server from assignment one**
  - You **must** use the port number given to you in week 1; **do not use port 80 or any other port number**

Murdoch UNIVERSITY

Assignment Two

- Your application will be required to parse and process XML AND JSON documents
- Full details of the requirements are explained in the assignment question
  - Also check the QandA regularly
- All students should submit their assignment on LMS AND have an identical application in their account on ceto, according to the instructions in the assignment question
- Late submission penalties will apply - refer to the unit guide and the assignment question

Murdoch UNIVERSITY

Learning Objectives

- In the scheme of what we are doing in this unit:
  - We are studying how to use XML / JSON as important Internet technologies for solutions in different areas
  - It is likely that any work in this industry will involve the use of Web Services
- This week's lecture is aimed at learning about RESTful Web Services

Learning Objectives

- Learn why RESTful Web Services are an important development in Internet technologies
- Learn about Rest Architecture
- Learn about the basics of RESTful Web Services
  - What is a resource?
  - Messages and Addressing
  - Statelessness and Caching
  - Security

Lecture Outline

- Overview: setting the scene
- The REST architecture
- RESTful Web Services
- Components of RESTful Web Services
- The role of JSON

Introduction

- "Web resources" were first defined on the World Wide Web as documents or files identified by their Uniform Resource Location (URL)
- Today we have a much more generic and abstract definition encompassing every thing or entity that can be identified, named, addressed or handled, in any way whatsoever, on the Web

Introduction

- RESTful Web services are one way of providing inter-operability between computer systems on the Internet
- REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and pre-defined set of stateless operations

Introduction

- The term REST was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation
- Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI)

Introduction

- REST refers to a network of Web resources (a virtual state-machine) where the user progresses through the application by selecting links and operations such as GET or DELETE
  - i.e. transitioning through states
- This results in the next resource (representing the next state of the application) being transferred to the user for their use

Introduction

- In a RESTful Web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or some other defined format
- The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources

Introduction

- Using HTTP, the kind of operations available include those pre-defined by the HTTP verbs GET, POST, PUT, DELETE and so on
- By making use of a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and scalability
  - This can be achieved by re-using components that can be managed and updated without affecting the system as a whole, even while it is running

What is REST Architecture?

- REST is an acronym for **RE**presentational **S**tate **T**ransfer
- REST is an architectural style for networked applications
  - It can be considered a collection of principles or a set of Web standards, which uses HTTP
- The concept of REST revolves around a ‘resource’ where:
  - Every component is a resource, and
  - A resource is accessed by a common interface using HTTP standard methods

What is REST Architecture?

- In REST architecture, a REST Server simply provides access to resources and a REST client accesses and modifies the resources
- Here each resource is identified by a URI
- REST uses various formats to represent a resource like text, HTML, JSON, XML
  - JSON is the most popular and commonly used

What is REST Architecture?

- The REST architecture describes six constraints:
  - Uniform interface – URI must uniquely identify the resource
  - Stateless – any necessary state to handle a request is in the request itself (querystring, body, headers) or the response itself (headers, status, body)
  - Cacheable – to prevent clients re-using stale or inappropriate data in further requests

What is REST Architecture?

- The REST architecture describes six constraints (cont.):
  - Client-Server – separation of responsibilities; the client is not concerned about data storage, and the server is not concerned about user interface/state
  - Layered system – the client is unaware if it is connected to the end server or an intermediary
  - Code on demand (optional) – allows server to temporarily extend client functionality by transferring executable logic; eg: client-side scripts and applets

What is REST Architecture?

- Difference between **resource** and **state**:
  - *Resource or resource state* is data that defines the resource representation
    - This is constant across every client who requests it
  - *State or application state* is data for the current session or request that is required by the server
    - This could vary by client and per request

REST: HTTP Methods

- The following five HTTP methods are commonly used in REST-based architecture:
  - GET - Read only access to a resource
  - PUT - Used to create a new resource
  - DELETE - Used to remove a resource
  - POST - Used to update/modify an existing resource or create a new resource
  - OPTIONS - Used to get the supported operations on a resource

RESTful Web Services

- Web services based on REST Architecture are known as **RESTful** Web Services
- Such web services use HTTP methods to implement the concept of REST architecture
- A RESTful Web Service usually defines a Uniform Resource Identifier (URI), where a service provides resource representation such as JSON and set of HTTP Methods

RESTful Web Services

- A RESTful Web Service is a collection of open protocols and standards for exchanging data between applications or systems
- Software applications, written in various programming languages and running on various platforms, can use RESTful Web Services to exchange data over computer networks in a manner similar to inter-process communication on a single computer

RESTful Web Services: What is a Resource?

- In REST architecture, everything is a resource
  - These resources can be text files, html pages, images, videos or dynamic business data
- A RESTful Web Server simply provides access to resources
- A RESTful Web Client accesses and modifies the resources

RESTful Web Services: What is a Resource?

- Each resource is identified by URIs
- REST uses various representations to represent a resource like text, JSON, XML
  - JSON is the most popular representations of resources

RESTful Web Services: Representation of Resources

- A resource in REST is similar to an Object in Object Oriented Programming or similar to an Entity in a Database
- Once a resource is identified, its representation must be decided upon, using a standard format so that a server can send the resource (in one of the previously mentioned formats), and the client can understand the same format

Characteristics of a Good Resource Representation

- In REST, there is no restriction on the format of a resource representation
- One client may ask for a JSON representation of a resource, whereas another client may ask (the same server) for an XML representation of the same resource, and so on
- It is responsibility of the REST server to pass the client the resource in the format that the client understands or requested

Characteristics of a Good Resource Representation

- When designing a representation format for a resource in a RESTful Web Service, the following are important considerations:
  - Understandability: Both Server and Client should be able to understand and utilize the representation format of the resource
  - Linkability: A resource can have a linkage to another resource, and a format should be able to handle such situations
  - Completeness: the format should be able to represent a resource completely

RESTful Web Services: Messages

- RESTful Web Services make use of HTTP protocol as a medium of communication between client and server
- A client sends a message in the form of a HTTP Request and server responds in the form of a HTTP Response
- This technique is termed “Messaging”
  - These messages contain message data and metadata (i.e. information about the message itself)

RESTful Web Services: Messages

- HTTP Request message has 5 major parts:
  - Verb – Indicates an HTTP method such as GET, POST, PUT, DELETE, OPTIONS
  - URI – Contains the URI to identify the resource on the server
  - HTTP Version – Indicates the HTTP version
  - Request Header – Contains metadata for the HTTP Request message as key-value pairs; eg: browser type, format supported by client, …
  - Request Body – Message content or resource representation (resource can contain another resource format which should be able to represent simple as well as complex structures of a resource)

RESTful Web Services: Messages

- HTTP Response message has 4 parts:
  - Status/Response Code – Indicates Server status for the requested resource; eg: 404 (resource not found) and 200 (ok)
  - HTTP Version – Indicates the HTTP version
  - Response Header – Contains metadata for the HTTP Response message as key-value pairs; eg: content length, content type, response date, server type, etc.
  - Response Body – Response message content or resource representation

RESTful Web Services: Addressing

- Addressing refers to locating a resource or multiple resources existent on a server
  - This is analogous to locating a postal address for a person
- Each resource in REST architecture is identified by its URI
- A URI is of following format:

```
<protocol>://<service-name>/
        <ResourceType>/<ResourceID>
```

RESTful Web Services: Addressing

- The purpose of an URI is to locate a resources on the server hosting the web service
- Another important attribute of a request is the `VERB`, which identifies the operation to be performed on the resource (achieved by the HTTP methods)

Constructing a Standard URI

- Important points to be considered:
  - Use Plural Nouns: to define resources; eg: the example that follows uses `users` to identify users as a resource
  - Avoid using spaces: instead use underscore (_) or hyphen (-) when using a long resource name
  - Use lowercase letters: although URI is case-insensitive, it is good practice to keep the URI in lower case letters only

Constructing a Standard URI

- Important points to be considered (cont.):
  - Maintain Backward Compatibility: as a Web Service is a public service, a URI once made public should always be available
    - In case the URI gets updated, redirect the older URI to new URI using HTTP Status code 300
  - Use an HTTP Verb: always use HTTP Verb like GET, PUT, and DELETE to do the operations on the resource
    - It is **not** good to use operations names in the URI

Constructing a Standard URI

- Important points to be considered (cont.):
  - For example, the following is a poor URI to fetch a user:
    `http://localhost:8080/UserManagement/rest/UserService/**getUser**/1`
  - The following is an example of a good URI to fetch a user:
  - `http://localhost:8080/UserManagement/rest/UserService/**users**/1`

RESTful Web Services: Statelessness

- As per REST architecture, a RESTful Web Service should not keep a client state on the server
- This restriction is called **statelessness**
- It is the responsibility of the client to pass its context to the server and then the server can store this context to process a client's further request

RESTful Web Services: Statelessness

- For example, a session maintained by a server is identified by a session identifier passed by the client
- RESTful Web services should adhere to this restriction
- In RESTful Web Services, web service methods do not store any information from the client they are invoked from

RESTful Web Services: Advantages of Statelessness

- Web services can treat each method request independently
- Web services do not need to maintain a client's previous interactions
  - This simplifies application design
- As HTTP is itself a stateless protocol, RESTful Web Services work seamlessly with HTTP protocol

RESTful Web Services: Disadvantages of Statelessness

- Web service servers need to get:
  - Extra client information in every request
  - The client's state in order to serve the client interaction

RESTful Web Services: Caching

- Caching refers to storing the server response in the client so that a client does not need to make a server request for the same resource again and again
- A server response should have information about how particular caching is to be done, so that a client caches a server response for a period of time OR never caches the server response

RESTful Web Services: Caching

- Following are the headers by which a server response can configure a client's caching:
  1. Date – Date and Time of when the resource was created
  2. Last Modified – Date and Time of when the resource was last modified
  3. Cache-Control – Primary header to control caching (see next slide)
  4. Expires – Expiration date and time of caching
  5. Age Duration in seconds from when resource was fetched from the server

RESTful Web Services: Cache-Control Header Directives

1. Public – Indicates that the resource is cacheable by any component
2. Private – Indicates that the resource is cacheable by only client and server, no intermediary can cache the resource
3. no-cache/no-store – Indicates that the resource is not cacheable
4. max-age – Indicates that caching is valid up to max-age in seconds (after this, the client has to make another request)
5. must-revalidate – Indication to the server to re-validate the resource if max-age has passed

RESTful Web Services: Caching Best Practices

- Always keep static contents cacheable, with expiration date of 2 to 3 days
  - For example images, css, JavaScript, etc.
- Never set expiration date too high
- Dynamic contents should be cached for a few hours only

RESTful Web Services: Security

- RESTful Web Services utilize HTTP, so it is very important to safeguard the URL paths of a RESTful Web Service
  - Just like it is important to secure a typical website
- Following are the best practices that should be considered when designing a RESTful Web Service:

RESTful Web Services: Security

- Validation – Validate all inputs on the server. Protect your server against SQL or NoSQL injection attacks
- Session based authentication – Use session based authentication to authenticate a user whenever a request is made to a Web Service method
- No sensitive data in URL – Never use username, password or session token in the URL; these values should be passed to the Web Service via the POST method

RESTful Web Services: Security

- Restriction on Method execution – Allow restricted use of methods like GET, POST, DELETE (GET method should not be able to delete data)
- Validate Malformed XML/JSON – Check for well formed input passed to a Web Service method
- Throw generic Error Messages – A Web Service method should use HTTP error messages
  - Eg: 403 to show access forbidden, etc.

RESTful Web Services: Security

- Always use standard HTTP codes when returning HTTP response to the client
- Some codes you may not be familiar with:
  - 201 CREATED, when a resource is successfully created using POST or PUT request (returns a link to newly created resource using location header)
  - 204 NO CONTENT, when a response body is empty (for example: a DELETE request)
  - 304 NOT MODIFIED, used to reduce network bandwidth usage in case of conditional GET requests (response body should be empty; Headers should have date, location, etc.)

RESTful Web Services: Security

- 401 UNAUTHORIZED, states that the user is using invalid or wrong authentication token
- 403 FORBIDDEN, states that the user does not have access to the method being used (for example: delete access without admin rights)
- 409 CONFLICT, states a conflict situation when executing the method (for example: adding duplicate entry)
- 500 INTERNAL SERVER ERROR, states that the server has thrown some exception while executing the method

REST: Example Operation

- A simple Web-based social application:
  1. A user visits the home page of an application by entering the address in the browser
  2. The browser submits an HTTP request to the server
  3. The server responds with an HTML document, with a form and some links
  4. The user enters data in the form and submits
  5. The browser submits another HTTP request to the server (with the form data)
  6. The server processes the request and responds with another page

REST: Example Operation

- The cycle continues until the user stops
  - Along the way, there could be numerous exceptions in the form of error messages
- So how does this application relate to REST?
- What the user types into the browser (point 1) is called the Uniform Resource Identifier
  - URI is more general than a URL and refers to a resource location or a resource name
  - A URI is an identifier of a resource

REST: Example Operation

- A **resource** is anything that can be identified by a URI
- In point 1 above, the user entry was a resource to a static webpage
- In point 6, the server processing of data submitted via the form is another resource
  - The form used to submit the data has the URI of this resource encoded as the value of the `action` attribute of the `form` element

REST: Example Operation

- The HTML pages returned by the server (points 3 and 6) are representations of a resource
- A **representation** is an encapsulation of the information of the resource encoded in a format such as XML, JSON, HTML
  - This information could consist of state, data, or mark-up
- A resource may have one or more representations

REST: Example Operation

- Clients and servers use **media types** to denote the type of representation
- Two common types are:
  - **`text/html`** for **HTML** format
  - **`application/x-www-form-urlencoded`** for URI-encoded format (used for **form** submission)

REST: Example Operation

- Clients use HTTP to submit requests for resources and to receive responses:
  - In point 1, the request uses GET to fetch an HTML page (which includes a form)
  - In point 4, the submission of the form is achieved with a POST containing the data
- These methods are part of HTTP's *uniform interface*, which makes communication self-describing and visible
  - The interface also includes other HTTP methods

REST: Example Operation

- Each representation the client receives from the server represents the state of the user's interaction with the application
- For example, when a user submits a form (which may invoke a response from the server), the user changes the state of the application

REST: Example Operation

- Similarly, when a user is just browsing a webpage and clicks on a link (to load another page), this also changes the state of the application
- Thus, in this context, HTML is a **hypermedia** format allowing links and forms to control the application flow and thereby change the state of the application
- This is known as the **hypermedia constraint**

References

- RESTFUL WEB SERVICES - QUICK GUIDE
  - http://www.tutorialspoint.com/restful/restful_quick_guide.htm
- RESTful Service Best Practices: Recommendations for Creating Web Services, 2012. Fredich, T.
  - www.restapitutorial.com
- RESTful Web Services. Richardson, L. and Ruby, S. O'Reilly, 2008.
- RESTful Web Services Cookbook. Allarmaraju, S. O'Reilly, 2010.
- RESTful Web APIs. Richardson, L. and Amundsen, M. O'Reilly, 2013.