Reminder of Midterm Next Week

- Midterm covers chapters 1-7. 30 questions and 0.5 for each question. Total mark: 15
- Make sure you try the dummy exam with the lockdown browser and are able to take the test
- The exam will be available 4:30-5:45 PM ET Wednesday, October 8th on Blackboard
- Send me on Piazza via **private posts** on at most five recommended questions by the end of Monday, October 5th
- Using the lockdown browser, blackboard will record your picture, your photo ID and your activity during exam
- If you have difficulty to access the exam at blackboard, you can contact me via slack or email
- During exam, it is difficult to discuss questions with me. Just answer the question based on your understanding. If you have questions, you can contact me after exam
- Because we are having midterm next week, the deadline for this chapter's homework and exercise is Wednesday after midterm
  - Working on homework and exercise should still help your exam preparation

Discussion #4

- CSS VS XSLT

	CSS	XSLT
Difference 1	Multiple stylesheet types with cascading priorities	One stylesheet type
Difference 2	Used for HTML	Used for structured document
Difference 3	Only client side	Server side and client side
Difference 4	Its own syntax	XML syntax
Difference 5	Same content, change representation	Much powerful. Transformation result could only have less data
Commonality 1	Both can be used for html representation	
…		

IS 651: Distributed Systems
Chapter 7: REST Web Services

Jianwu Wang

Fall 2020

Learning Outcomes

- After learning chapter 7, you should be able to
  - Understand the features of REST Web service and its differences from SOAP based Web service
  - Know how to use proper method to call a REST Web service
  - Know JSON and its differences from XML
  - Know the cross-domain restriction and how to work around it

REST Basics

- REST is a term coined by Roy Fielding in his Ph.D. dissertation to describe an architecture style of networked systems. REST is an acronym standing for Representational State Transfer. It is easier to understand using representational resource state transfer.
  - Resource: Resources are any addressable object (as something with a URI on the web), such as a book or student record
  - Representation: Resource representations for client and resources at server side are separated
  - REST: access and manipulate resource states using a representational approach. We don’t care how the resource is actually stored/managed on the server, We only care its representation from the client view

REST Basics (2)

- REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages.
- An HTTP REST Web service follows three basic design principles
  - *Use HTTP methods explicitly (HTTP is stateless)*
  - *Expose directory structure-like URIs*
  - *Transfer XML, JavaScript Object Notation (JSON), or both*
- There are no standards/specifications for REST Web service

Use HTTP methods explicitly

- Methods
  - *POST:* create, sending data
  - *GET:* read, list, retrieve
  - *PUT:* replace, update
  - *DELETE:* delete
- GET Examples
  - From your web browser: https://api.targetlock.io/v1/post-code/21250
  - Curl command: curl -v https://api.targetlock.io/v1/post-code/21250

POST vs. GET

- POST should be used for creating resources
- Common error:
  - Wrong*: GET /adduser?name=Robert HTTP/1.1*
  - Correct

```
POST /users HTTP/1.1
Host: myserver Content-Type: application/xml
<?xml version="1.0"?>
<user>
<name>Robert</name>
</user>
```

GET vs. PUT

- GET is for data retrieval only. GET is an operation that should be free of side effects, a property also known as **idempotence**.

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

- Common error:
  - Wrong: *GET /updateuser?name=Robert&newname=Bob HTTP/1.1*
  - Correct:

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
<name>Bob</name>
</user>
```

- Similarly, DELETE should be used rather than a deleteuser function with GET.

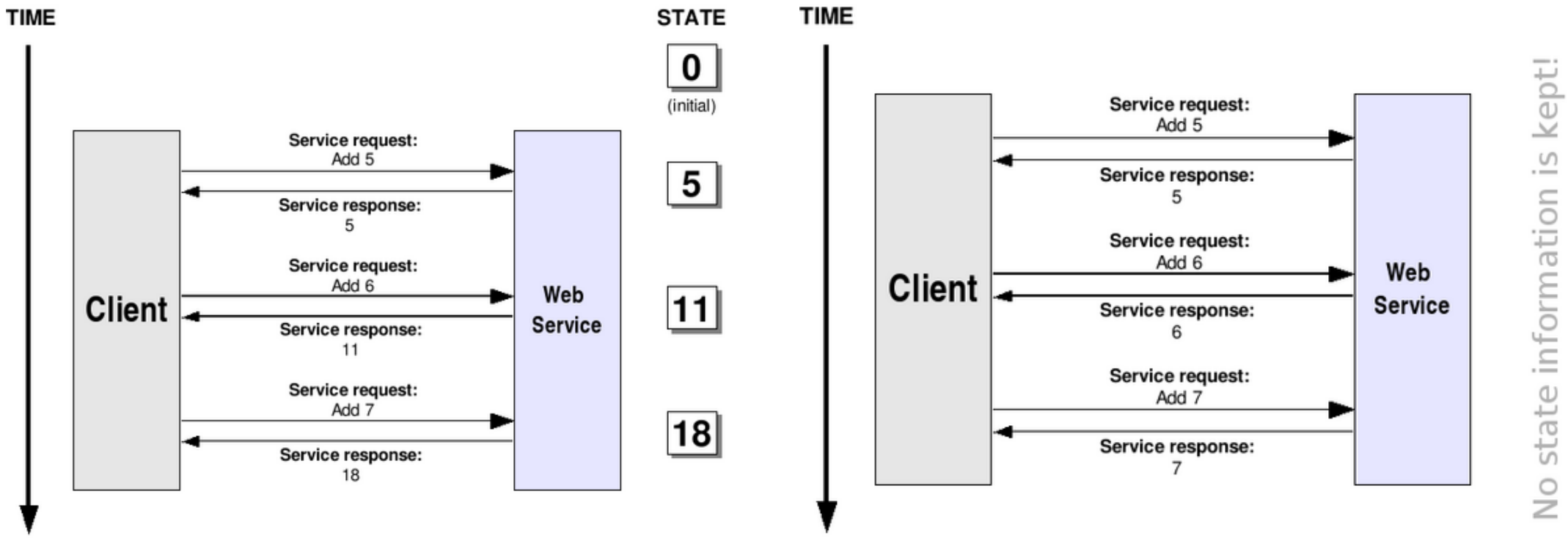
Be Stateless

- A complete, independent request doesn't require the server, while processing the request, to retrieve any kind of **application/client context or state**.
  - Treats each request as an independent transaction that is unrelated to any **previous** request
- A REST Web service application/client includes, within the HTTP headers and body of a request, all of the parameters, context, and data needed by the server-side component to generate a response.
- Stateless server-side components are less complicated to design, write, and distribute across load-balanced servers.

Be Stateless (2)

- A stateless service not only performs better, it shifts most of the responsibility of maintaining state to the client application.
- In a RESTful Web service, the server is responsible for generating responses and for providing an interface that enables the client to maintain application state on its own.
- For example, in the request for a multipage result set, the client should include the actual page number to retrieve instead of simply asking for next.
- The principle of *loose-coupling* implies statelessness.

Compare a Stateless and Stateful Service



Expose Directory Structure-like URIs

- REST Web service URIs should be intuitive to the point where they are easy to guess.
  - Think of an URI as a kind of self-documenting interface that requires little, if any, explanation or reference for a developer to understand what it points to and to derive related resources.

Directory Structure-like URI Examples

- Example URIs from the book (not real):
  - *http://www.w3schools.com/catalog/cds (for cd list)*
  - *http://www.w3schools.com/catalog/cds/2 (for the detailed info of the cd)*
  - *http://www.w3schools.com/getCD.php?cd=2 (not the best url, but same as above)*
- Examples from the #5 reference (not real):
  - *http://www.parts-depot.com/parts (for a parts list)*
  - *http://www.parts-depot.com/parts/00345 (for a part)*
  - *http://www.parts-depot.com/parts/getPart?id=00345 (not the best url, but same as above)*

Transfer XML, JSON, or both

- Example: A REST service makes available a URL to submit a purchase order (PO)
  - The client creates an PO instance document which conforms to the PO schema that Parts Depot has designed (and publicized in a WSDL document)
  - The client submits PO.xml as the payload (i.e. entity body) of an HTTP POST message
- The payload (HTTP entity body) should be in XML or JSON
- Both XML and JSON are semi-structured data, *a.k.a.* self-describing structure
  - Does not obey the tabular structure like relational database
  - Contains tags to separate semantic elements and enforce hierarchies of records and fields within the data

JSON

- Alternative serialization

```
{"menu": {
 "id": "file",
 "value": "File:",
 "popup": {
    "menuitem": [
        {"value": "New", "onclick": "CreateNewDoc()"},
        {"value": "Open", "onclick": "OpenDoc()"},
        {"value": "Close", "onclick": "CloseDoc()"} ]
 }
}}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()"/>
    <menuitem value="Open" onclick="OpenDoc()"/>
    <menuitem value="Close" onclick="CloseDoc()"/>
  </popup>
</menu>
```

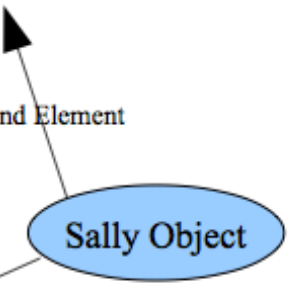
- A useful website to format/view data in json: https://jsonformatter.org/json-viewer

JSON Example

- The figure shows how JSON works in JavaScript.
  - Program link
- The JSON is included directly in the program here, but could easily be the result of a REST query.
- It results in the simple text output to the browser: *Sally Green 27*

```
<script>
var employees = { "accounting" : [   // accounting is an array in employees.
                        { "firstName" : "John",  // First element
                         "lastName"  : "Doe",
                         "age"       : 23 },
                        { "firstName" : "Mary",  // Second Element
                         "lastName"  : "Smith",
                         "age"       : 32 }
                       ], // End "accounting" array.


            "sales"     : [ // Sales is another array in employees.
                        { "firstName" : "Sally", // First Element
                         "lastName"  : "Green",
                         "age"       : 27 },
                        { "firstName" : "Jim",   // Second Element
                         "lastName"  : "Galley",
                         "age"       : 41 }
                       ] // End "sales" Array.
          }; // End Employees
document.write(employees.sales[0].firstName + ' ');
// native js method write outputs Sally concatenated to a space
document.write(employees.sales[0].lastName + ' ');
// native js method write outputs Green concatenated to a space
document.write(employees.sales[0].age);
// native js method write outputs 27
</script>
```



Guardian API

- Most Web applications offer a REST API such as Twitter, Flickr, YAHOO, Facebook, The New York Times, NPR, and the Guardian Newspaper.
- The Guardian News
- Guardian Open Platform API docs
- Result result in JSON, DEMO
  - http://content.guardianapis.com/search?q=syria&section=news&from-date=2013-09-01&api-key=xyz
- Result result in XML, DEMO
  - http://content.guardianapis.com/search?q=syria&section=news&from-date=2013-09-01&format=xml&api-key=xyz
- You can use command to call the Rest web service
  - curl -v "http://content.guardianapis.com/search?q=syria&section=news&from-date=2013-09-01&api-key=xyz"
- You need to replace the xyz in the last two links with your api-key to make them work.

Cross-Domain Restriction

- One problem that comes up is that web browsers have a security limitation that requires any program running in the browser (using JavaScript with AJAX) can only return results from the same domain that the original web page came from.
  - This is called the **cross-domain restriction**
  - Our Guardian example would therefore not work for a web page we created on gl since our web page is from umbc.edu and Guardian is on the guardian.com
- Demo: https://userpages.umbc.edu/~jianwu/is651/programs/ch7/cross_domain_restriction.html

Work Around the Cross-Domain Restriction

- We can work around the restriction by having 1) a server-side program to fetch data and 2) a client-side program to parse and present the fetched data

Client-side program running on your computer's web browser

Server-side program running on GL machine

REST service hosted at Guardian

CURL and PHP in Server-Side Program

- Use a server-side program to retrieve the XML or JSON from Guardian and then send it back to the user that requested the web page from gl.
  - In order to do this, we will use PHP to issue the request using the Curl library.
  - The Curl library offers a way to send a URL programmatically and handle the response.

CURL and PHP Example

```
<?php
	$querystring='q=debates&section=news&from-date=2013-09-01&api-key=xyz';
	$host ='https://content.guardianapis.com/search';
	$request = $host.'?'.$querystring;

	$ch = curl_init();
	curl_setopt($ch, CURLOPT_URL, $request); // set url
	curl_setopt($ch, CURLOPT_FAILONERROR, 1); // fail the request if return code is > 400
	curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);// allow redirects
	curl_setopt($ch, CURLOPT_RETURNTRANSFER,1); // return variable
	curl_setopt($ch, CURLOPT_TIMEOUT, 4); // times out after 4s
	curl_setopt($ch, CURLOPT_HTTPGET, true); // set GET method

	$json = curl_exec($ch);
	curl_close($ch);
	header("Content-type: application/json"); //send http header
	echo "jsonProcessFn(".$json.");"; //wraps result in a function call
?>
```

- guardian.php
  - Program link
  - Source code link

JSONP (JSON with Padding) in Client-Side Program

- The client-side program can use JSONP to call the PHP and process the result.
  - JSONP uses the <script> tag, instead of the XMLHttpRequest object
  - JSONP requires the data is wrapped by function name
- Example: guardian.html

guardian.html Example – Part 1

```
<!DOCTYPE html>
<html>
<style>
    h3{background:red;}
    span{color:green;}
</style>
<body>
<p id="output"></p>
<script>
function jsonProcessFn(data) {
  var h3 = document.createElement("h3"); //create an h3 element
  h3.innerHTML = "User Tier = "+ data.response.userTier;
  //output element is defined in the html.
  document.getElementById("output").appendChild(h3);
  //response and results are at php results.
  var arr = data.response.results;
  for (var i = 0, len = arr.length; i < len; i++) {
    newsProcessFn(arr[i]);
  }
}
```

guardian.html Example – Part 2

```
function newsProcessFn(news) {
  var li = document.createElement("li");
  var span1 = document.createElement("span");
  span1.innerHTML = "Date= ";
  var span2 = document.createElement("span");
  span2.innerHTML = "Title= ";
  li.appendChild(span1);
  li.appendChild(document.createTextNode(news.webPublicationDate + " "));
  li.appendChild(span2);
  li.appendChild(document.createTextNode(news.webTitle));
  document.getElementById("output").appendChild(li);
} </script>
<script src="guardian.php"></script>
</body>
</html>
```