Note on Case study

- Post at the Piazza post on your topic on your choice for your papers/projects by 04/16
- Providing multiple candidates will help me suggest the best one for you
- You can directly upload papers you selected
- You can utilize additional resources, including multimedia resources, to explain/demonstrate the concepts and working mechanism
- Add references/acknowledgement if you use resources from others and make sure you can explain them well
- Analyze the techniques objectively, including its strength, weakness, applicable conditions
- Make connections with what we have learned, especially Chapter8 (Distributed System Basics)
- Make sure to sign up your presentation's time slot via the spreadsheet

Discussion #8

- Briefly explain how the implementation of catalog application looks like without following service-oriented architecture, and its differences from the current one
  - Implementation without SOA
    - MEAN/LAMP stack for web application
    - 3 layers: application, business logic, database
    - Mostly for human, not programs, to use
  - Differences
    - Using SOA have an additional layer: service layer. It hurts latency but supports easy integration with other applications through service API and loose-coupling
    - If the implementation follows object-oriented programing, it is easy to convert it to follow SOA using third-party libraries

IS 651: Distributed Systems
Chapter 11: REST Revisited

Jianwu Wang

Spring 2021

Learning Outcomes

- After learning this chapter, you should be able to
  - Learn RSS and display RSS feed programmatically
  - Understand different XML parsers and their differences
  - Understand and build mashup applications
  - Implement REST services w/o CodeIgniter

REST Revisited

- REST defines a set of architectural principles by which you can design Web services that focus on a system's resources
  - How resource states are addressed and transferred over HTTP
- An HTTP REST Web service follows three basic design principles:
  - *Use HTTP methods (GET, POST, PUT, DELETE) explicitly (HTTP is stateless)*
  - *Expose directory structure-like URIs*
  - *Transfer XML, JavaScript Object Notation (JSON), or both*
- REST should be stateless
  - But sometimes, it needs to handle some states

Flickr API Images

- In Chapter 7, we only see xml in return by calling Flickr API, not the actual image

```
<rsp stat="ok">
<photos page="1" pages="10357" perpage="3" total="51783">
 <photo id="5794206993" owner="42836099@N07" secret="861560fe53"
server="3232" farm="4" title="DSC10509 (crop)" ispublic="1" isfriend="0"
isfamily="0" />
<photo id="5794178809" owner="34902206@N02" secret="c7f9c421ae"
server="2022" farm="3" title="IMG_6796" ispublic="1" isfriend="0" isfamily="0" />
<photo id="5794733362" owner="34902206@N02" secret="b2bc0b1a1d"
server="3299" farm="4" title="IMG_6795" ispublic="1" isfriend="0" isfamily="0" />
</photos>
</rsp>
```

- We can construct image based on the reply using structure: http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
  - http://farm4.static.flickr.com/3232/5794206993_861560fe53.jpg

phpFlickr: a third-party library for Flickr

- Comments of the demo program
  1. All the variables above this comment initialize a variable for use in the Flickr API URL. The user IDs for the Flickr are in a format like 45883295@N04
  2. This program is for the Flickr API flickr.photos.search entry and uses the previous variables
  3. This block loops through all the results of the search and creates the image links
  4. Note how the mapping we see in the demo/exercise is done automagically here! (The line should not wrap.)
- The REST service link to retrieve the same images without phpFlickr library

```
<?php
  require_once("phpFlickr.php");
  $f = new phpFlickr("YOUR_FLICKR_API_KEY");
  $tag = "YOUR_TAG";
  $num = "NUMBER_OF_RESULT_TO_RETURN";
  $nsid = "USER_ID"; //1
  $photos = $f -> photos_search(array("user_id" =>
$nsid, "tags" => $tag, "per_page" => $num,
    "sort" => "date-posted-desc")); //2
  if(count($photos) == 0) { //counting the size of
array
    echo "<p>Sorry, the requested photo(s) could
      not be found.</p>";
  } else {
    foreach($photos['photo'] as $photo) { //3
      echo "<a href='http://www.flickr.com/photos/" .
        $photo['owner'] . "/" . $photo['id'] .
        "/' title='View larger'>";
      echo "<img src='" . $f->buildPhotoURL($photo,
      "Medium") . "' alt='$photo[title]'
       border='0'></a>"; //4
    }
  }
?>
```

RSS (Really Simple Syndication)

- A different family of XML vocabularies for blogs and other uses
- All RSS is XML with various schemas depending on version
- An RSS document, called "feed" or "channel", includes text and metadata of the text
- A feed/channel contains one or more items

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
    <title>RSS Title</title>
    <description>This is an example of an RSS feed</description>
    <link>http://www.someexamplerssdomain.com/main.html</link>
    <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
    <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
    <item>
        <title>Example entry</title>
        <description>
            Here is some text containing an interesting description.
        </description>
        <link>http://en.wikipedia.org/wiki/RSS/</link>
        <guid>unique string per item</guid>
        <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
    </item>
</channel>
</rss>
```

Display RSS Feeds using Magpie library

- Comments of the demo program
  1. As usual, we must include the library code for use in the program.
  2. I used an RSS feed from the Washington Post about politics
     - You can substitute the URL for any feed
     - You can go to the feed URL and see that it is typically transformed by default in your web browser, but the program receives the XML
     - You can see this by using curl from the command-line. It will return the XML feed rather than the HTML that your browser displays

```php
<?php
    require_once 'magpie/rss_fetch.inc'; //1
    $url =
'http://feeds.washingtonpost.com/rss/politics'; //2
    $rss = fetch_rss($url);
    $i=0;
    echo "Site: ", $rss->channel['title'], "<br>"; //3
    foreach ($rss->items as $item) { //4
        $title = $item['title'];
        $url = $item['link'];
        echo "<a href=$url>$title</a></li><br>";
        if(++$i==3) break;
    }
?>
```

Display RSS Feeds using Magpie library (2)

- Comments of the demo program
  3. Magpie parses the XML of the feed into PHP arrays
     - That is how the code can address tags using array notation such as channel['title']
     - Magpie returns a PHP object and then uses the arrow notation to access each tag as in: $rss->channel['title']
  4. The foreach loop gets the desired subset of these tag variables for display

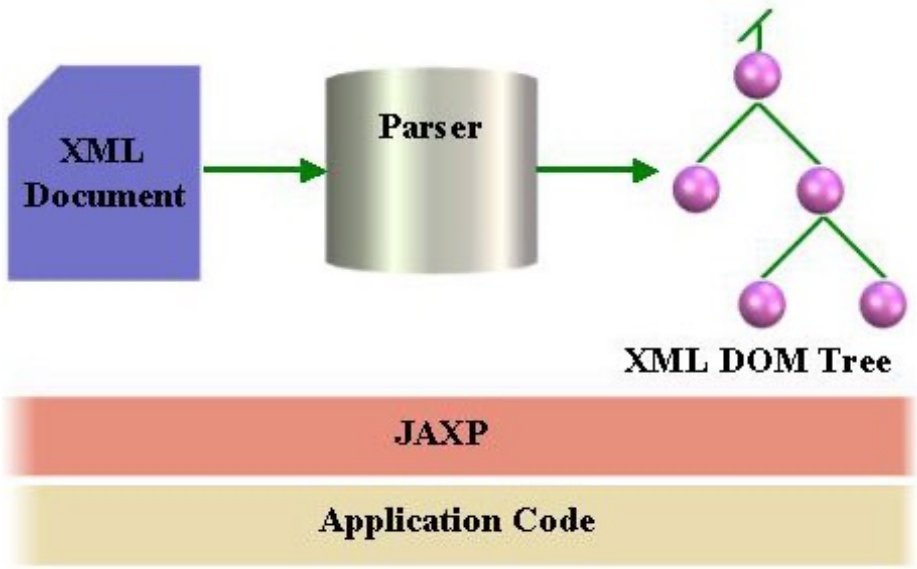
```php
<?php
    require_once 'magpie/rss_fetch.inc'; //1
    $url =
'http://feeds.washingtonpost.com/rss/politics'; //2
    $rss = fetch_rss($url);
    $i=0;
    echo "Site: ", $rss->channel['title'], "<br>"; //3
    foreach ($rss->items as $item) { //4
        $title = $item['title'];
        $url = $item['link'];
        echo "<a href=$url>$title</a></li><br>";
        if(++$i==3) break;
    }
?>
```

Quick Question

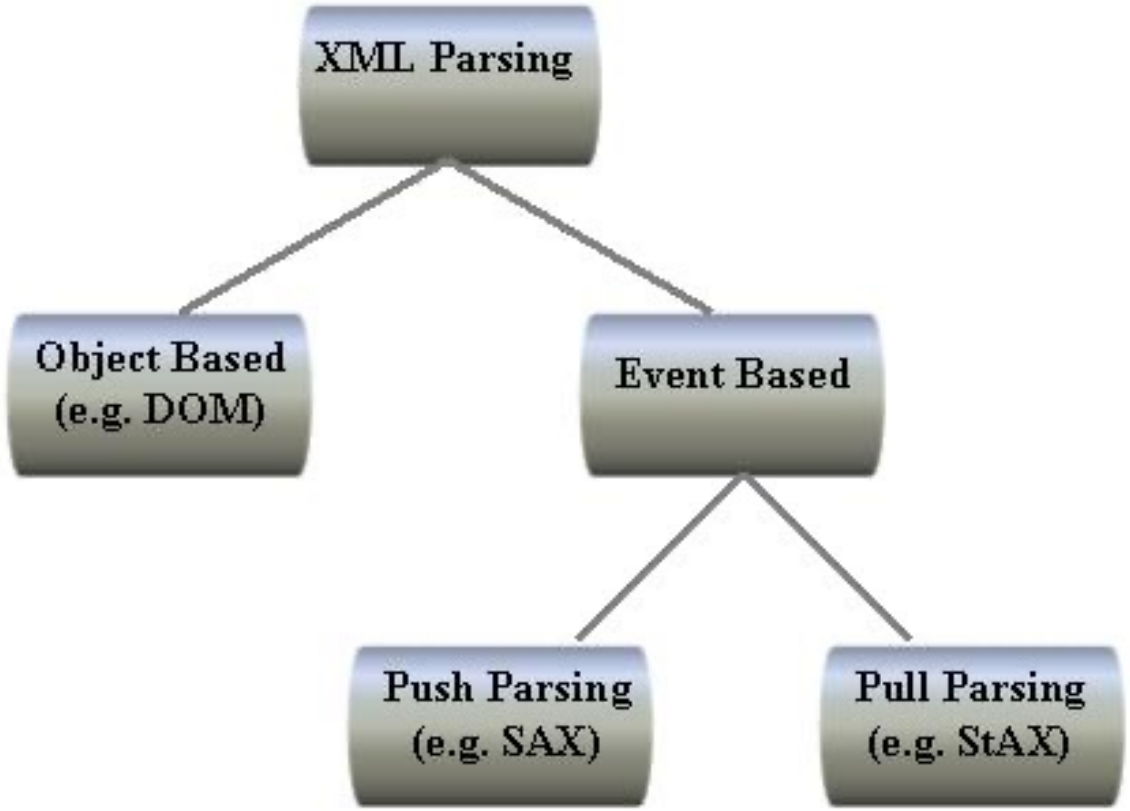
- RSS feeds are really the first REST web service (before we have REST web service concept), why?

XML Parsers

- Object-based (DOM)
- Event-based
  - Push Parsers (SAX)
  - Pull Parsers (StAX)



XML DOM Parser



XML Parser Classification

Mashups

- A mashup is a web-based application that uses and combines data from two or more sources to create new services
- Mashup of the Movie Blogger rss feed and The Open Movie Database (Omdbapi) for movie data
- Final Demo (the movies listed change over time)
  - Demos with source codes: Demo 1, Demo 2, Demo 3
- Helper Function Code - to do the call to omdbapi and return the data
  - You can use it unchanged, but be sure you understand it
  - Change .txt to .php and put in your gl account

Creating a REST API using CodeIgniter

- Demo
- GET has function index() in the controller
  - Get all the records from the database with the URL: http://host/CI/index.php?/Rest/
  - Get a specific record using the id as: http://host/CI/index.php?/Rest/index/1
  - The if statement checks for the id using the CI URI Class. The segments are numbered with #1 being rest. So #3 would the id=1 above
  - postmodel is the model implementation to fetch data. Fetched data is returned to client
- POST has function create()
  - One can create a record with a URL such as: http://host/CI/index.php?/Rest/create/Motorhead/Hammered/Rock
  - The id can be ignored as it will be auto-incremented by the database
  - postmodel function to save data. A message array is returned to client

```
<?php  class Rest extends Controller
{       function Rest()
        {       parent::Controller();
                $this->load->model('postmodel');        }
        function index() //1. get
        {       if($this->uri->segment(3)==FALSE)
                {       $data = $this->postmodel->get_post();
                        $this->response($data);
                }else{
                        $id=$this->uri->segment("3");
                        $data = $this->postmodel->get_post($id);
                        $this->response($data);         }       }
        function create() //2. post
        {       $data = array('artist' => $this->uri->segment(3),
                        'title' => $this->uri->segment(4),
                        'genre' => $this->uri->segment(5));
                $this->postmodel->create_post($data);
                $message = array('message' => 'Added!');
                $this->response($message);   }
```

Creating a REST API using CodeIgniter (2)

- PUT has the function edit () since it updates an existing record.
  - Sample Url: http://host/CI/index.php?/Rest/edit/Motorhead/Hammered/Metal/2 if it has id=2
- DELETE has the function delete () and deletes records by id
  - Sample Url: http://host/CI/index.php?/Rest/delete/2

```
function edit() //3. put
{       $id = $this->uri->segment(6);
        $put_data = array('artist' => $this->uri->segment(3),
                          'title' => $this->uri->segment(4),
                          'genre' => $this->uri->segment(5));
        $this->postmodel->update_post($put_data, $id);
        $message = array('id' => $id, 'message' => 'Edited!');
        $this->response($message);  }
function delete() //4. delete
{       $id = $this->uri->segment(3);
        $this->postmodel->delete_post($id);
        $message = array('message' => 'Deleted!');
        $this->response($message);  }        …  ?>
```

The Problem?

- The problem is we have not really implemented a uniform HTTP method interface
  - We can see this by issuing a GET command using curl to update. It works!
    - curl -v -X GET http://host/CI/index.php?/Rest/edit/Motorhead/Hammered/Metal/2
  - This is bad because updates are supposed to use PUT

```
...
> GET /CI/index.php?/Rest/edit/Motorhead/Hammered/Metal/2 HTTP/1.1
...
< HTTP/1.1 200 OK
...
<?xml version="1.0" encoding="UTF-8"?>
* Connection #0 to host localhost left intact
* Closing connection #0
<root><id>25</id><message>Edited!</message></root>
```

Creating a *(correct)* REST API

- Demo
- So now we have a complete REST API for the cdStore service that meets all the requirements for a RESTful architecture in CodeIgniter.

```
function index() //1
{   $request_method = $_SERVER['REQUEST_METHOD']; //2
    $id = (int) $this->uri->segment("3");
    if($id == NULL)
    {   switch($request_method) //3
        {   case "GET":
                $data = $this->postmodel->get_post();
                $this->response($data);
                break;
            case "POST":
                $this->postmodel->create_post($_POST);//4
                $message = array('message' => 'Added!');
                $this->response($message);
                break;   }   }   else
    {   switch($request_method)
        {   case "GET":
                $data = $this->postmodel->get_post($id);
                $this->response($data);
                break;
            case "PUT":
                parse_str(file_get_contents("php://input"), $put_data); //5
                $this->postmodel->update_post($put_data, $id); //6
                $message = array('id' => $id, 'message' => 'Edited!');
                $this->response($message);
                break;
            case "DELETE":
                $this->postmodel->delete_post($id);
                $message = array('message' => 'Deleted!');
                $this->response($message);
                break; //7
        }   }   }
```

Creating a REST API without CodeIgniter

- You will implement one for your own catalog in the homework from scratch - not using CodeIgniter
- Scratch API - use this for the homework