XML and JSON

Structured Object Representations

Objectives

- Understand the syntax of both representations
- Practical Use of XML & JSON
  - **Access third party web services**
  - *Android UI layout, Drawables, Animation sequence (.xml)*
  - *iOS Storyboard (.xib)*
  - *Qt screen design*
  - *… other SDKs ...*
- Use PHP functions for parsing XML and JSON strings
- Use JavaScript functions for parsing JSON strings

XML Example

(Car Rental)

Renter Name: **John Smith**
In State Rental**: Yes**
Car License: **ABC 7654**
Rental Start Date: **2016-05-22**
Rental Duration: **7**
Additional Drivers:

1. **Bob Smith** (**S123456789**)
2. **Chuck deGroot** (**D444555111**)

```
<?xml version=”1.0” encoding=”UTF-8”?>
<rental>
  <renter>John Smith</renter>
  <in-state>true</in-state>
  <car-license>ABC 7654</car-license>
  <start-date>2016-05-22</start-date>
  <duration>7</duration>
  <extra-drivers>
    <driver>
      <name>Bob Smith</name>
      <drv-license>S123456789</drv-license>
    </driver>
    <driver>
      <name>Chuck deGroot</name>
      <drv-license>D444555111</drv-license>
    </driver>
  </extra-drivers>
</rental>
```

JSON Example

(Car Rental)

Renter Name: **John Smith**
In State Rental**: Yes**
Car License: **ABC 7654**
Rental Start Date: **2016-05-22**
Rental Duration: **7**
Additional Drivers:

1. **Bob Smith** (**S123456789**)
2. **Chuck deGroot** (**D444555111**)

```
{
  “renter”: “John Smith”,
  “in-state”: true,
  “car-license”: “ABC 7654”,
  “start-date”: “2016-05-22”,
  “duration”: 7,
  “extra-drivers”: [
    {
      “name”: “Bob Smith”,
      “drv-license”: “S123456789”
    },
    {
      “name”: “Chuck deGroot”,
      “drv-license”: “D444555111”
    }
  ]
}
```

XML Example (using attributes)

(Car Rental)

Renter Name: **John Smith**
In State Rental**: Yes**
Car License: **ABC 7654**
Rental Start Date: **2016-05-22**
Rental Duration: **7**
Additional Drivers:

1. **Bob Smith** (**S123456789**)
2. **Chuck deGroot** (**D444555111**)

```
<?xml version="1.0" encoding="UTF-8"?>
<rental start="2016-05-22" duration="7">
  <renter name="John Smith" />
  <in-state>true</in-state>
  <car-license number="ABC 7654" />
  <extra-drivers>
    <driver name="Bob Smith"
        license="S123456789" />
    <driver name="Chuck deGroot"
        license="D444555111" />
  </extra-drivers>
</rental>
```

XML vs. JSON

XML

- Structure (parent-child) can be validated using DTD
- Very verbose
- No specific notation for arrays

JSON

- No formal description for structure validation
- About 30% shorter
- Distinct notations for arrays and objects

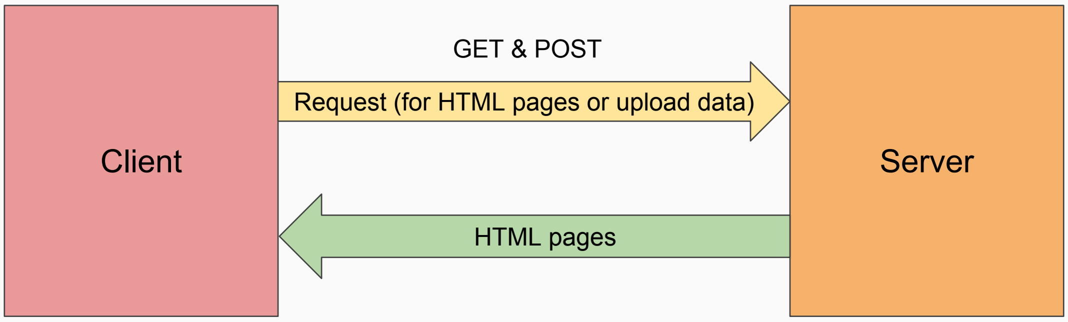
Third Party Web Services

- Your WebApp may have to obtain additional data from external sources
  - Current Weather Conditions
  - Currency Exchange Rate
  - Twitter Posts
  - Flickr Photos
  - ...
- Response formats from *modern* third-party web services (Web APIs) are either XML or JSON

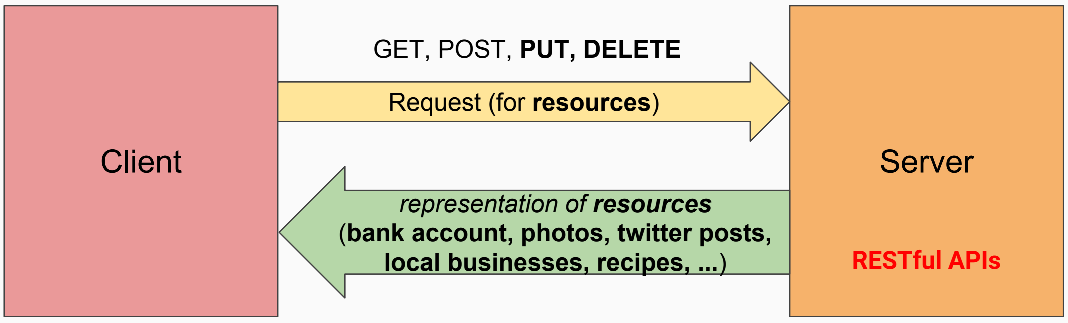
REST Web APIs

- Roy Fielding Doctoral Dissertation
  - Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architecture*. Doctoral dissertation, University of California, Irvine, 2000
- REpresentational State Transfer
  - Stateless Data ("resources") Exchange between Client and Server
  - Uniform Resource Identifiers (URIs)
  - Resource: distributed hypermedia

What we **have done**:



What we **want to accomplish next**:


Client
GET, POST, **PUT, DELETE**
Request (for **resources**)
Server
representation of ***resources***
(**bank account, photos, twitter posts, local businesses, recipes, ...**)
RESTful APIs

Examples of Web Services

- ApiGee Console (https://apigee.com/console)
- Programmable Web: http://www.programmableweb.com
- Tons of them!!!

Demo: Weather Underground Web Services

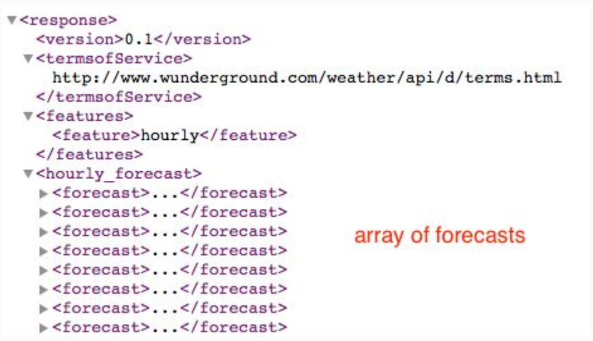
XML Parsing in PHP

XML/JSON Parsing in PHP

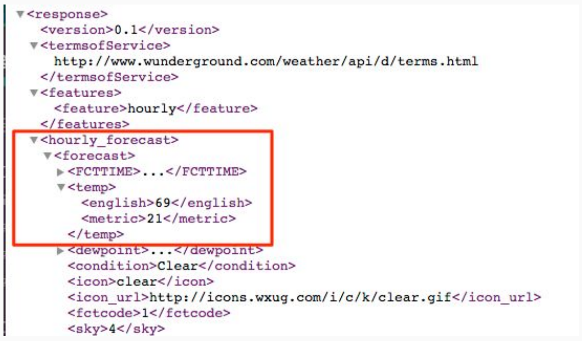
- Easier to use, non-customizable
  - `file_get_contents()`
- Parser
  - `simplexml_load_string()`: convert XML strings to PHP associative arrays
  - `json_decode()`: convert JSON strings to PHP associative arrays

Web Service: Weather Underground (hourly forecast)

```
http://api.wunderground.com/api/YOUR_API_KEY_HERE/hourly/q/MI/Allendale.xml
```



```
▼<response>
   <version>0.1</version>
 ▼<termsofService>
     http://www.wunderground.com/weather/api/d/terms.html
   </termsofService>
 ▼<features>
     <feature>hourly</feature>
   </features>
 ▼<hourly_forecast>
   ▶<forecast>...</forecast>
   ▶<forecast>...</forecast>
   ▶<forecast>...</forecast>
   ▶<forecast>...</forecast>
   ▶<forecast>...</forecast>
   ▶<forecast>...</forecast>
   ▶<forecast>...</forecast>
   ▶<forecast>...</forecast>
```



```
▼<response>
   <version>0.1</version>
 ▼<termsofService>
     http://www.wunderground.com/weather/api/d/terms.html
   </termsofService>
 ▼<features>
     <feature>hourly</feature>
   </features>
 ▼<hourly_forecast>
   ▼<forecast>
     ▶<FCTTIME>...</FCTTIME>
     ▼<temp>
        <english>69</english>
        <metric>21</metric>
      </temp>
     ▶<dewpoint>...</dewpoint>
      <condition>Clear</condition>
      <icon>clear</icon>
      <icon_url>http://icons.wxug.com/i/c/k/clear.gif</icon_url>
      <fctcode>1</fctcode>
      <sky>4</sky>
```

response ⇒ hourly_forecast ⇒ forecast[*pos*] ⇒ temp ⇒ metric

Parsing of Weather Underground XML Response

```
<!-- wunderkey.php -->
<?php
  $apiKey = "my_api_key_for_weather_underground";
?>
```

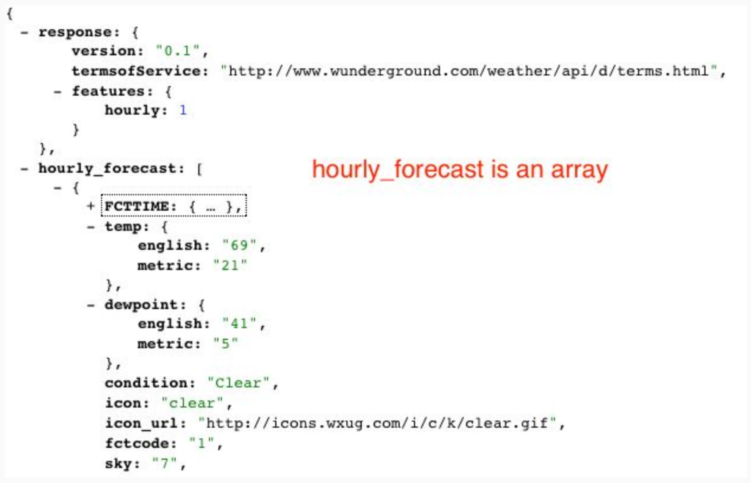
```
<?php
  require_once 'wunderkey.php';
  $result = file_get_contents("https://api.wunderground.com/api/$apiKey" .
     "/hourly/q/MI/Allendale.xml");
  $weather = simplexml_load_string ($result);
  print_r ($weather);

  print_r ($weather->hourly_forecast->forecast[0]->temp->metric);
?>
```

JSON Parsing in PHP

Web Service: Weather Underground (hourly forecast)

```
http://api.wunderground.com/api/YOUR_API_KEY_HERE/hourly/q/MI/Allendale.json
```



```
{
  - response: {
        version: "0.1",
        termsofService: "http://www.wunderground.com/weather/api/d/terms.html",
      - features: {
            hourly: 1
        }
    },
  - hourly_forecast: [
      - {
          + FCTTIME: { … },
          - temp: {
                english: "69",
                metric: "21"
            },
          - dewpoint: {
                english: "41",
                metric: "5"
            },
            condition: "Clear",
            icon: "clear",
            icon_url: "http://icons.wxug.com/i/c/k/clear.gif",
            fctcode: "1",
            sky: "7",
```

hourly_forecast is an array

response ⇒ hourly_forecast[*pos*] ⇒ temp ⇒ metric

Parsing of Weather Underground JSON Response in PHP

```
<!-- wunderkey.php -->
<?php
  $apiKey = “my_api_key_for_weather_underground”;
?>
```

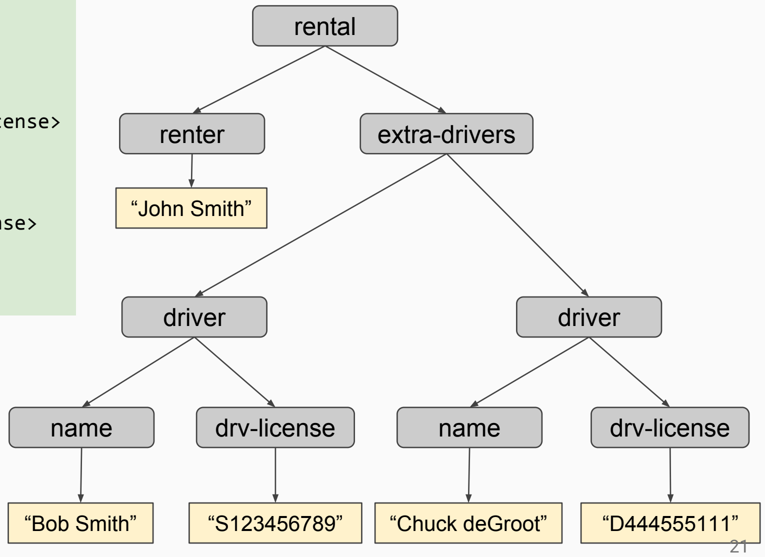
```
<?php
  require_once ‘wunderkey.php’;
  $result = file_get_contents(“https://api.wunderground.com/api/$apiKey” .
     “/hourly/q/MI/Allendale.json”);
  $weather = json_decode ($result);
  print_r ($weather);

  print_r ($weather->hourly_forecast[0]->temp->metric);
?>
```

DOMParser:
XML Parsing in JavaScript

XML and DOM Tree

```
<rental>
  <renter>John Smith</renter>
  <extra-drivers>
    <driver>
        <name>Bob Smith</name>
        <drv-license>S123456789</drv-license>
    </driver>
    <driver>
      <name>Chuck deGroot</name>
      <drv-license>D444555111</drv-license>
    </driver>
  </extra-drivers>
</rental>
```



JavaScript DOMParser class for parsing XML

```
var parser = new DOMParser();
// Use BACKQUOTES for multiline strings in JavaScript!!!
var xmlString = `<rental>
  <renter>John Smith</renter>
  <extra-drivers>
    <driver>
        <name>Bob Smith</name>    <drv-license>S123456789</drv-license>
    </driver>
    <driver>
      <name>Chuck deGroot</name>  <drv-license>D444555111</drv-license>
    </driver>
  </extra-drivers>
</rental>`;

var xmldoc = parser.parseFromString(xmlString, 'text/xml');
// You can now use HTML document functions: getElementsByTagName(), etc...
var client = xmldoc.getElementsByTagName('renter')[0];
console.log(client.firstChild);                        // Output "John Smith"

var drivers = xmldoc.getElementsByTagName('driver');
for (var k = 0; k < drivers.length; k++) {
   var name = drivers[k].children[0].firstChild;           // driver's name
   var lics = drivers[k].children[1].firstChild;           // driver's license
}
```

JSON.Parse(): Parsing JSON in JavaScript

JavaScript JSON.parse()

```
var jsonString =    // use backquotes for multiline strings in JavaScript
`{
  "renter": "John Smith",
  "duration": 7,
  "extra-drivers": [
    { "name": "Bob Smith",      "drv-license": "S123456789" },
    { "name": "Chuck deGroot", "drv-license": "D444555111" }
  ]
}`;
var jsonDoc = JSON.parse(jsonString);
console.log(jsonDoc.renter);                             // output “John Smith”
for (var k = 0; k < jsonDoc['extra-drivers'].length; k++) {
    var dr = jsonDoc['extra-drivers'][k];
    console.log(dr.name);
    console.log(dr['drv-license']);
}
```

XML Namespace

- Primary use of XML is for encoding machine readable data/resources
- XML-encoded data from multiple providers may have **naming conflict**
  - `<loan>` : book loan from library? Financial loan from bank?
  - `<temperature>`: chemical reaction or weather?
- Avoid naming conflict using XML namespaces (`xmlns`)
  - C++: `using namespace std;`
  - Java: `package edu.gvsu.cis.lakermobile;`

XML namespaces

```
<lib:loan
  xmlns:lib=”http://www.gvsu.edu/lib”>

  <lib:student>G00012345</lib:student>
  <lib:book_id>TS.646.3</lib:book_id>

</lib:loan>
```

```
<cu:loan
  xmlns:cu=”http://www.lmcu.org/fin”>
  <cu:amount>25000</cu:amount>
  <cu:apr>0.031</cu:apr>
</cu:loan>
```

The URLs (http://www.gvsu.edu/lib and http://www.lmcu.org/fin) do not refer to any physical document. They are used by the parser only to distinguish the two <loan> elements