JSON

Unit 2- Refer e-book JSON by O'Reilly uploaded on teams
-Keypoints by Faculty: Fiona Coutinho

Pls Note students: the contents in this pdf or any other document uploaded by me has only the keypoints,summary and some extra information in some topics,So students have to refer to their textbooks and syllabus while studying…Don't rely only on uploaded documents

JSON: Javascript object Notation

- JSON: Basics of JSON,JSON syntax, JSON data types,JSON schemas
- The JavaScript XmlHttpRequest and Web APIs: Web APIs,The JavaScript XMLHttpRequest.
- JSON, Client-side frameworks, JSON on the server side: Serializing, Deserializing and Requesting JSON:PHP.

JSON

- JSON basics
- JSON schema
- JSON php
- JSON javascript
- XMLHttpRequest
- Client-Side Frameworks

JSON basics

JSON

- What is JSON?

JSON stands for JavaScript Object Notation. It is data saved in a .json file, and consists of a series of key/value pairs.

{ "key": "value" }

JSON

- The value of any JSON key can be a string, Boolean, number, null, array, or object.
- **Comments are not allowed in JSON.**
- Although JSON resembles an object or an array, **JSON is a string.**
- A serialized string, which means it can later be parsed and decoded into data types.

JSON Data Types

Valid Data Types

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

JSON Data Types

1. JSON Strings: **{"name":"John"}**
2. JSON Numbers: **{"age":30}** or **{“height":6.3}**
3. JSON Objects:

{"employee":{"name":"John", "age":30, "city":"New York"}}

4. JSON Arrays: **{"employees":["John", "Anna", "Peter"]}**
5. JSON Booleans: **{"sale":true}**
6. JSON null: **{"middlename":null}**

JSON schema

JSON Schema

- A JSON Schema is written with JSON

JSON Schema can answer the following questions:

- Are the data types of the values correct? We can specify that a value has to be a number, string, etc.
- Does this include the required data?We can specify what data is required, and what is not.
- Are the values in the format that I require?We can specify ranges, minimum and maximum.
- **In our very first name-value pair of our JSON, we must declare it as a schema document**

  Example - name for this declaration will always be “$schema,” and the value will always be the link for the draft version

```
{
"$schema": "http://json-schema.org/draft-04/schema#"
}
```

JSON Schema

- **The second name-value pair in our JSON Schema Document will be the title .**

  Example - Format for a document that represents a cat

```
{
"$schema": "http://json-schema.org/draft-04/schema#",
"title": "Cat"
}
```

- **In the third name-value pair of our JSON Schema Document, we will define the properties that we want to be included in the JSON.**
  - The property value is a skeleton of the name-value pairs of the JSON we want.

    Example-

```
{
"$schema": "http://json-schema.org/draft-04/schema#",
"title": "Cat",
"properties": {
"name": {"type": "string"},
"age": {"type": "number","description": "Your cat's age in years."},
"declawed": {"type": "boolean“}
                        }
}
```

JSON Schema: Defining the properties for a cat

- We can then validate that our JSON conforms to the JSON Schema

{"name": "Fluffy","age": 2,"declawed": false}

- In next Example we add a fourth name-value pair, “required,” with an array of required values for its value.
- Name, age, and declawed are required, so we add them to this list.

JSON Schema:Defining the properties for a cat

```
{"$schema": "http://json-schema.org/draft-04/schema#",
"title": "Cat",
"properties": {
"name": {"type": "string"},
"age": {"type": "number","description": "Your cat's age in
years."},
"declawed": {"type": "boolean"},
"description": {"type": "string"}
},
"required": ["name","age","declawed"]
}
```

JSON Schema: Defining the properties for a cat

Valid JSON

```
{
"name": "Fluffy",
"age": 2,
"declawed": false,
"description" : "Fluffy loves to sleep all day."
}
```

JSON Schema: Defining the properties for a cat

In our JSON Schema, we can define a minimum length and a maximum length for a string, and a minimum for a number.

In Ex. Below validation has been added to ensure that the cat's name is a minimum of 3 characters and a maximum of 20 characters. Additionally, we ensure that the age of the cat submitted is not a negative number.

```
{
"$schema": "http://json-schema.org/draft-04/schema#",
"title": "Cat",
"properties": {
"name": {"type": "string","minLength": 3,"maxLength" : 20},
"age": {"type": "number","description": "Your cat's age in years.","minimum" : 0},
"declawed": {"type": "boolean“},
```

JSON Schema: Defining the properties for a cat

```
"description": {"type": "string"}
},
"required": ["name","age","declawed"]
}
```

JSON Schema: Defining the properties

The JSON in Example below is invalid with the cat JSON Schema because the name value exceeds the maxLength, and the age value precedes the minimum.

Example of Invalid JSON

{"name": "Fluffy the greatest cat in the whole wide world","age": -2,"declawed": false,
"description" : "Fluffy loves to sleep all day."
}

The JSON below is valid with the cat JSON Schema and conforms to the requirements for the values.

Example of valid JSON

{"name": "Fluffy","age": 2,"declawed": false,"description" : "Fluffy loves to sleep all day."
}

JSON Schema

JSON Schema Validator - Newtonsoft

JSON Schema Lint :: JSON Schema Validator

Nested JSON


```
[
{ "id": 1, "name": "Max Adams", "born": "21 Oct 2001",
 "cars": [ {"make": "Mercedes", "model": "GLS" }, {"make": "Audi",
  "model": "Q8" } ], "favourite": { "colour": "Blue", "movie": "Toy Story"
  } },
{ "id": 2, "name": "Steven Miller", "born": "28 Jul 2004",
"cars": [ {"make": "Tesla", "model": "Model S"} ], "favourite": { "colour":
  "Red", "movie": "Mulan" } },
{ "id": 3, "name": "Daniel James", "born": "18 Dec 2005",
"cars": [ {"make": "Bugatti", "model": "Chiron"} ], "favourite": { "colour":
  "Yellow", "movie": "Frozen", "toy": "Teddy" } }
]
```

Nested JSON

```
$filepath = './persons.txt';
$json_string = file_get_contents($filepath);
$json = json_decode($json_string, true);
foreach($json_array as $elem) {
echo($elem['id']. ", ".$elem['name'] ); echo("<br/>"); }
//access nested objects inside array
foreach($json as $elem) { echo( $elem['name']." -
  ".$elem['favourite']['colour'] ); echo("<br/>"); }
//access json arrays inside nested json
foreach($json as $elem) { echo( $elem['name']. " : " );
foreach ($elem['cars'] as $car ) { echo($car['make']. ", "); }
  echo("<br/>"); }
```

JSON :PHP

json_encode() turns PHP into JSON

json_decode() turns JSON into php

```
$data = [
        'name' => 'Aragorn',
        'race' => 'Human'
];
echo json_encode($data);
```

JSON :Using data from JSON with PHP

```
<?php
//json saved in php string:
$data = '{
        "name": "Aragorn",
        "race": "Human"
}';
$character = json_decode($data);//convert json to php
echo $character->name;//fetch json data using php

?>
```

JSON :Accessing a JSON feed from a URL or file-data.json

```
[  {
     name: 'Aragorn',
     race: 'Human',
   },
   {
     name: 'Legolas',
     race: 'Elf',
   },
   {
     name: 'Gimli',
     race: 'Dwarf',
   },
]
```

JSON :Accessing a JSON feed from a URL or file-data.json

we'll extract that data in PHP.

```
<?php
$url = 'data.json'; // path to your JSON file
$data = file_get_contents($url); // put the contents of the file into a variable
$characters = json_decode($data); // decode the JSON feed
echo $characters[0]->name;
foreach ($characters as $character) {
        echo $character->name . '<br>';
}
```

JSON PHP

- A common use of JSON is to read data from a web server, and display the data in a web page.
- PHP has some built-in functions to handle JSON.
- Objects in PHP can be converted into JSON by using the PHP function json_encode():

Run program phparraytojsonobject1.php

JSON PHP

Run program phparraytojsonobject1.php:

```
<?php
$myObj->name = "John";
$myObj->age = 30;
$myObj->city = "New York";

$myJSON = json_encode($myObj);

echo $myJSON;
?>
```

JSON PHP

Eg2:

```
<?php
$myArr = array("John", "Mary", "Peter", "Sally");

$myJSON = json_encode($myArr);

echo $myJSON;
?>
```

The Client JavaScript

- Here is a JavaScript on the client, using an AJAX call to request the demo_file.php PHP file:

```
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myObj = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myObj.name;
}
xmlhttp.open("GET", "demo_file.php");
xmlhttp.send();
```

demo_file.php

```
<?php
$myObj = new stdClass();
$myObj->name = "John";
$myObj->age = 30;
$myObj->city = "New York";

$myJSON = json_encode($myObj);

echo $myJSON;
?>
```

JSON.parse()

- A common use of JSON is to exchange data to/from a web server.
- When receiving data from a web server, the data is always a string.
- Parse the data with JSON.parse(), and the data becomes a JavaScript object.

JSON.parse()

- Imagine we received this text from a web server:

  '{"name":"John", "age":30, "city":"New York"}‘

- Use the JavaScript function JSON.parse() to convert text into a JavaScript object:

```
const obj = JSON.parse('{"name":"John", "age":30, "city":"New York"}');
```

- Now Use the JavaScript object in your page as follows:

JSON.parse()

```
<!DOCTYPE html>
<html>
<body>
<h2>Creating an Object from a JSON String</h2>
<p id="demo"></p>
<script>
const txt = '{"name":"John", "age":30, "city":"New York"}'
const obj = JSON.parse(txt);
document.getElementById("demo").innerHTML = obj.name + ", " +
obj.age;
</script>
</body>
</html>
```

Store and retrieve data from local storage using JSON

```
<!DOCTYPE html>
<html>
<body>
<h2>Store and retrieve data from local storage.</h2>
<p id="demo"></p>
<script>
// Storing data:
const myObj = { name: "John", age: 31, city: "New York" };
const myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
// Retrieving data:
let text = localStorage.getItem("testJSON");
let obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
</script>
</body>
</html>
```

Stringify:Converts javascript to JSON

- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:
- const myObj = {name: "John", age: 31, city: "New York"};
  const myJSON = JSON.stringify(myObj);
  window.location = "demo_json.php?x=" + myJSON;

Receiving Data

- If you receive data in JSON format, you can easily convert it into a JavaScript object:
- const myJSON = '{"name":"John", "age":31, "city":"New York"}';
  const myObj = JSON.parse(myJSON);
  document.getElementById("demo").innerHTML = myObj.name;

JSON From a Server

- You can request JSON from the server by using an AJAX request
- As long as the response from the server is written in JSON format, you can parse the string into a JavaScript object.
- Use the XMLHttpRequest to get data from the server:

```
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function()
 {  const myObj = JSON.parse(this.responseText);//to JS object
  document.getElementById("demo").innerHTML = myObj.name;
};
xmlhttp.open("GET", "json_demo.txt");
xmlhttp.send();
```

json_demo.txt

```
{
    "name":"John",
    "age":31,
    "pets":[
        { "animal":"dog", "name":"Fido" },
        { "animal":"cat", "name":"Felix" },
        { "animal":"hamster", "name":"Lightning" }
    ]
}
```

Array as JSON:JSON returned from a server as an array:

```
<!DOCTYPE html>
<html>
<body>
<h2>Fetch a JSON file with XMLHttpRequest</h2>
<p>Content written as an JSON array will be converted into a JavaScript array.</p>
<p id="demo"></p>
<script>
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload = function() {
  const myArr = JSON.parse(this.responseText);
  document.getElementById("demo").innerHTML = myArr[0];
}
xmlhttp.open("GET", "json_demo_array.txt", true);
xmlhttp.send();
</script>
</body>
</html>
```

Array as JSON:JSON returned from a server as an array:

json_demo_array.txt:

["Ford", "BMW", "Audi", "Fiat"]

JSON :Javascript

```
var data =  '[
                { "name": "Aragorn", "race": "Human" },
                { "name": "Gimli", "race": "Dwarf" }
              ]‘
data = JSON.parse(data);
Now we can access the data like a regular JavaScript object.
console.log(data[1].name)//Gimli
//And we can loop through each iteration with a for loop.
for (var i = 0; i < data.length; i++) {
  console.log(data[i].name + ' is a ' + data[i].race + '.')
}
```

JSON :Javascript

Lets take JSON string and put it in data.json.

```
[
  {
    name: 'Aragorn',
    race: 'Human',
  },
  {
    name: 'Gimli',
    race: 'Dwarf',
  },
]
```

JSON :Javascript

```
var request = new XMLHttpRequest()
request.open('GET', 'data.json', true)
request.onload = function () {
  // begin accessing JSON data here
  var data = JSON.parse(this.response)
  for (var i = 0; i < data.length; i++) {
    console.log(data[i].name + ' is a ' + data[i].race + '.')
  }
}
request.send()
```

JSON :Javascript

Using jQuery

- As you can see, it's not too difficult to retrieve a JSON feed with plain JavaScript. However, it's even easier with jQuery, using the getJSON() function.

```
$(document).ready(function () {
  $.getJSON('data.json', function (data) {console.log(data[0].name)
})})
```

- Also jQuery can access JSON via an AJAX request

```
$(document).ready(function () {  var data$.ajax({dataType: 'json', url:
'data.json',data: data,
    success: function (data) {console.log(data[0].name)},
  })
})
```

XMLHttpRequest

The JavaScript XMLHttpRequest

- AJAX stands for Asynchronous JavaScript and XML.
- When we are making behind-the-scenes requests for JSON, this would technically be Asynchronous JavaScript and JSON (AJAJ).
- Let's put AJAX with JSON and the JavaScript XMLHttpRequest Together next.

The JavaScript XMLHttpRequest

Lets creates a new XMLHttpRequest object and gets JSON from the Open Weather Map API.

- Example:

```
var myXMLHttpRequest = new XMLHttpRequest();
var url =
"http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139";
myXMLHttpRequest.onreadystatechange = function() {
if (myXMLHttpRequest.readyState === 4 && myXMLHttpRequest.status
=== 200) {
var myObject = JSON.parse(myXMLHttpRequest.responseText);
var myJSON = JSON.stringify(myObject);
}
}
myXMLHttpRequest.open("GET", url, true);
myXMLHttpRequest.send();
```

The JavaScript XMLHttpRequest

- In the example a function is assigned to the onreadystate-change property of myXMLHttpRequest.
- This function will be executed every time the readyState property changes.
- Here we check if the readyState is 4 and that the HTTP status is 200
- If both are true then parse the JSON into a JSON object.
- JSON is turned into text from an object and then from an object back into text are: serialization and deserialization.
- Serialization is the act of converting the object into text.
- Deserialization is the act of converting the text back into an object.

The JavaScript XMLHttpRequest

- The JavaScript is deserializing with JSON.parse.
- In the responseText,the JSON is simply text as a data interchange format. Once it is parsed by JSON.parse,it is no longer JSON, but a JavaScript object.
- Deserialization Example:-

```
var myJSON = JSON.parse(myXMLHttpRequest.responseText);
```

This deserialization with JSON.parse is necessary because the JSON is not yet an object. Remember that JSON stands for JavaScript Object Notation. While it is in its JSON form, it is a literal representation of an object in the form of text. In order for JSON to become a real object, it must be deserialized. In JavaScript, we can also serialize the JSON with JSON.stringify().

- In Example below, the myObject variable gets the deserialized JSON. This is now an object.
- The myJSON variable gets the serialized object. This is now JSON.

Example Object deserialized and then serialized

```
/var myObject = JSON.parse(myXMLHttpRequest.responseText);//the JSON response deserialized
var myJSON = JSON.stringify(myObject); // The object serialized
```

Finally, the last two lines in my example are setting up the request and then sending it via the HTTP protocol

```
myXMLHttpRequest.open("GET", url, true);
myXMLHttpRequest.send();
```

The JavaScript XMLHttpRequest

- The function value of the onreadystatechange property is an EventHandler.
- The underlying JavaScript engine (not my code) has logic to access the properties value (my function) every time the ready state changes. The ready states from 0-4 are:
- 0 for UNSENT:The state before the open() function has been executed
- 1 for OPENED:The state after the open() function has been executed but before send() has been executed
- 2 for HEADERS_RECEIVED:The state after the send() function has been executed and headers and status are available
- 3 for LOADING:The headers have been received but the response text is still being retrieved
- 4 for DONE:Complete; the full message with headers and body has been received

Client-side frameworks

- Jquery
- AngularJS

Client-Side Frameworks-jQuery with JSON:

- jQuery is an abstraction tool to manipulate the HTML Document Object Model (DOM).
- Here the HTML doc is treated as an object and a set of nodes that can be enumerated, accessed,and manipulated.
- Manipulation of the DOM is possible in JavaScript but it take several lines of code.
- For example to hide an HTML element(button)
- In javascript:<button id="myButton">My Button</button>
  document.getElementById("myButton").style.display = "none";
- In JQuery:$("#myButton").hide();
- few characters of code needed and Saves production time

– jQuery also deals with Internet browser compatibility issues ..In older versions of Internet Explorer, Firefox,and Chrome, JSON.parse() is not supported

- jQuery has a function for parsing JSON: jQuery.parseJSON.

Client-Side Frameworks-jQuery with JSON:

- Example of Parsing JSON in JavaScript with JSON.parse()
  var myAnimal = JSON.parse('{ "animal" : "cat" }');
- Example of Parsing JSON with jQuery using jQuery.parseJSON
  var myAnimal = jQuery.parseJSON('{ "animal" : "cat" }');
- In this example jQuery.parseJSON uses attempts to use the native JSON.parse() function. If its not available in the browser (for old versions) it falls back to using new Function().
- It has some logic for checking invalid characters present in a script in an injection attack attempt, in which case an error is thrown.

Client-Side Frameworks-jQuery with JSON:

- There is a function for making an HTTP request for JSON. If this is done with JavaScript it can require a good chunk of code as shown next:
- Example creates a new XMLHttpRequest object and gets JSON from the Open Weather Map API

```
var myXMLHttpRequest = new XMLHttpRequest();
var url = "http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139";
myXMLHttpRequest.onreadystatechange = function() {
if (myXMLHttpRequest.readyState === 4 && myXMLHttpRequest.status === 200)
{
var myObject = JSON.parse(myXMLHttpRequest.responseText);
var myJSON = JSON.stringify(myObject);
}
}
myXMLHttpRequest.open("GET", url, true);
myXMLHttpRequest.send();
```

Client-Side Frameworks-jQuery with JSON:

- Same example in Jquery:

```
var url =
"http://api.openweathermap.org/data/2.5/weather?lat=35&lon
=139";
$.getJSON(url, function(data) {
// do something with weather data
});
```

Client-Side Frameworks- AngularJS with JSON:

- The AngularJS framework is an abstraction tool that caters to single-page applications.
- Single-page web applications are web pages that break away from the traditional multipage web application experience into a single page.
- The traditional multipage web application experience is heavily tied the human interaction between client and server.
- The human either types in or clicks a URL that makes a request for a resource with HTTP.
- This dance between the client and server and the human in a traditional web application involves each step “moving” to a new web page.

Client-Side Frameworks- AngularJS with JSON:

- Before the Internet grew up most people used.
- Single-page web applications give you the seamless experience just like desktop applications right in your Internet browser using JavaScript and XMLHttpRequest.
- Instead of the human hopping from URLs and links to URLs, the background code handles the resource requests while the human remains on one page.
- The single-page web application is a complex system so it provides a framework based on the model-view-controller (MVC) architectural concept to built it.
- AngularJS implements the MVC concept as follows:

1. Model:JavaScript objects are the data model
2. View:HTML
3. Controller:The JavaScript files that use the AngularJS syntax defining and handling the interactions with the Model and the view

Client-Side Frameworks- AngularJS with JSON:

- AngularJS can have a steep learning curve
- It requires a shift in thinking about how to interact with the DOM.
- Angular decouples DOM manipulation from application logic.
- For example, I need to change a message on a page from the generic “Hello, stranger” to a personalized greeting when a user signs in
- In JavaScript, we would achieve this by manipulating the Document Object Model.

Client-Side Frameworks- AngularJS with JSON:

- Example HTML message when the user is not signed in

```
<h1 id="message">Hello, stranger!</h1>
```

- Example JavaScript to change the message if the user is signed in

```
if (signedIn) {
var message = "Hello, " + userName + "!";
document.getElementById("message").innerHTML = message;
}
```

- AngularJS Example

```
<body ng-app="myApp">
<div id="wrapper" ng-controller="myAppController as ctrl">
<h1 id="message">Hello {{ ctrl.username }}!</h1>
</div>
</body>
angular.module('myApp', [])
.controller('myAppController',[ function() { var self=this;
$self.username="Stranger";
if(signedIn){$self.username = "Bob";}]);
```

Client-Side Frameworks- AngularJS with JSON:

- The “ng-app” and “ng-controller” attributes on the html tags (<body> and) are syntax for setting up the view to support data binding.
- Data binding is done using two open curly brackets “{{” and two closing curly brackets “}}”
- The controller file instead of manipulating the DOM manipulates data model(username).
- We can also design complex application such as a page that allows users to perform create, read, update, and delete (CRUD) operations on their user info, the Angular way simplifies development.

Client-Side Frameworks- AngularJS with JSON:

- JSON is the notation for JavaScript object literals. So where does JSON fit in with AngularJS?
- In an AngularJS data model, the most common vehicle for getting
- that data from the database to the data model is with JSON.
- This is often a JSON resource requested over HTTP, a client-server relationship.
- AngularJS leverages this relationship for retrieving data models easily with a core AngularJS service: $http.

Client-Side Frameworks- AngularJS with JSON:

- In this Example the AngularJS $http is used to get weather data from the Open Weather Map API; the JSON is added to the global scope as an object named “weatherData.”
- Example of Getting weather data from the Open Weather Map API

```
angular.module('myApp', [])
.controller('myAppController', function($scope, $http) {
$http.get('http://api.openweathermap.org/data/2.5/weather?lat=
35&lon=139').
success(function(data, status, headers, config) {
$scope.weatherData = data;
});
});
```

Client-Side Frameworks- AngularJS with JSON:

JSON response from the Open Weather Map API

{"coord": {"lon": 139,"lat": 35},
"sys": {"message": 0.0099,"country": "JP","sunrise": 1431805135,"sunset": 1431855710},
"weather": [{"id": 800,"main": "Clear","description": "sky is clear","icon": "02n"}],
"base": "stations","main": {"temp": 291.116,"temp_min": 291.116,"temp_max": 291.116,"pressure": 1020.61,"sea_level": 1028.58,"grnd_level": 1020.61,"humidity": 95},
"wind": {"speed": 1.51,"deg": 339.501},
"clouds": {"all": 8},
"dt": 1431874405,"id": 1851632,"name": "Shuzenji","cod": 200
}

Client-Side Frameworks- AngularJS with JSON:

- The JSON from the Open Weather Map API is deserialized by the AngularJS $http.
- Once its added to the global scope as an object named “weatherData,” it becomes our data model
- Example below displays The weather description (weather property) which contains an array of objects, with only one value. I select the value at index 0 with “weather[0]” and then the property “description.”
- Example Binding the weather description

```
<body ng-app="myApp">
<div id="wrapper" ng-controller="myAppController">
<div>{{ weatherData.weather[0].description }}
</div>
</div>
</body>
```

Web APIs

- The JavaScript XMLHttpRequest is the client making the requests, and the web API is the server sending the responses.
- Unlike humans, code doesn't have a pair of eyes to read or view.
- Code needs to view that "something" in a format that it can read (parse).
- This is where a data interchange format

Web APIs

JSON weather data from the OpenWeatherMap web API


```
{"dt": 1433383200,"temp":
{
"day": 293.5,
"min": 293.5,
"max": 293.5,
"night": 293.5,
"eve": 293.5,
"morn": 293.5
},
"pressure": 1015.06,"humidity": 98,
"weather": [{"id": 802,"main": "Clouds","description": "scattered clouds","icon":
"03n"}],
"speed": 2.86,
"deg": 134,
"clouds": 44
}
```

Web APIs

- The JSON weather data example can be “read” by any code capable of parsing JSON.
- This JSON resource can be requested with a URL :
- http://api.openweathermap.org/data/2.5/forecast/daily?lat=35&lon=139&cnt=10&mode=json
- Though many public web APIs like OpenWeatherMap are for “reading,” many APIs such as the PayPal API are more interactive.
- A web API is a set of instructions and standards for interacting with a service over HTTP. Interaction can include create, read, update, and delete (CRUD) operations and the web API will have a reference outlining these instructions and standards.
- For example, according to the PayPal API reference, I can create a new invoice with the PayPal API by posting JSON to the URL: https://api.sandbox.paypal.com/v1/invoicing/invoices
- In Example 6-2, we have JSON representing an invoice to be sent as a request to the PayPal API

Web APIs

Example 6-2. An invoice for the PayPal API

{"merchant_info": {"email": "bob@bob.com","first_name": "Bob","last_name": "Bobberson","business_name": "Bob Equipment, LLC","phone": { "country_code": "001","national_number": "5555555555"}, "address": {"line1": "123 Fake St.","city": "Somewhere","state": "OR","postal_code": "97520","country_code": "US"}}, "billing_info": [{"email": "someguy@someguy.com"}], "items": [{"name": "Widgets","quantity": 20,"unit_price": {

Web APIs

"currency": "USD","value": 89}}],
"note": "Special Widgets Order!","payment_term": {"term_type":
"NET_45"},
"shipping_info": {"first_name": "Some","last_name": "Guy",
"business_name": "Not applicable","address": {
"line1": "456 Real Fake Dr","city": "Some Place","state": "OR",
"postal_code": "97501","country_code": "US"}}}

Web APIs

- With the PayPal API, once the invoice is created, I can request (read), update, and delete that invoice.
- Operations done behind the scenes in JavaScript, such as the request for the weather data, are referred to as asynchronous.
- Asynchronous operations are operations that take place in the background without interrupting the main transmission.
- In the case of JavaScript asynchronous operations, the “main transmission” would be the display of the web browser. For example, a news web page might include a sidebar with real-time weather data. While you’re reading your news article, code in the background could asynchronously update the weather display every 60 seconds.
- This operation would not require the page to reload, nor would it interrupt your page scrolling to read your article. The only thing on the page to change would be the sidebar with the weather data.
- The asynchronous (background) operations of JavaScript are referred to as AJAX.