KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Module 5

- JSON Data Interchange Format -Syntax, Data Types, Object, JSON Schema, Manipulating JSON data with PHP.

JSON Is a Data Interchange Format

- A data interchange format is a text format used to exchange data between platforms.
- Another data interchange format you may already have heard of is XML.
- The world needs data interchange formats, like XML and JSON, to exchange data between very different systems.

- JSON stands for JavaScript Object Notation.

-“Object” is a common programming concept, in particular to object-oriented programming (OOP)

-Notation implies a system of characters for representing data such as numbers or elements.

- JSON Is Programming Language Independent
- JSON is based on JavaScript object literals.
- SON is a **text format** for storing and transporting data
- JSON is "self-describing" and easy to understand

Key Terms and Concepts

This chapter covered the following key terms:

JSON
: JavaScript Object Notation

Notation
: A system of characters for representing data such as numbers or elements

Data interchange format
: Text used to exchange data between platforms or systems

Portability
: Transferring information between platforms in a way that is compatible with both systems

We also discussed these key concepts:

- JSON is a data interchange format
- JSON is programming language independent (JavaScript is not required to use it)
- JSON is based on the object literal notation of JavaScript (emphasis on the word "notation")
- JSON represents data in a way that is friendly to universal programming concepts

JSON Syntax

- JSON Is Based on JavaScript Object Literals

x = 5;

x=x+5;

- we know the value of x is 10, but we don't see 10. In this example, x was the variable, and 5 was a literal or a number literal.

Example 2-1. Using JSON to describe the shoes I'm wearing right now

```
{
    "brand": "Crocs",
    "color": "pink",
    "size": 9,
    "hasLaces": false
}
```

JSON Syntax

- The main point of the shoe example is that you (even a human) can literally read the attributes of my shoe.
- The data type for my JSON shoe example is object.
- We can see the literal value of the object which exposes the properties or attributes in a way which we can see (and read).
- These attributes or properties of the shoe object are represented as name-value pairs.
- The way that JSON is based on JavaScript object literals is purely in the syntactic representation of the object literal and its properties. This representation of properties is achieved with name-value pairs.

JSON Syntax

Name-Value Pairs

- They are called by other names as well: key-value pairs, attribute-value pairs, and field-value pairs.
- In a name-value pair, you first declare the name. For example, “animal.” Now, pair implies two things: a name and a value.
- With name-value pairs in JSON, the value can also be a number, a boolean, null, an array, or an object or string value.

"animal" : "cat“

- JSON uses the colon character (:) to separate the names and values. The name is always on the left and the value is always on the right

"animal" : "horse"

"animal" : "dog"

Proper JSON Syntax

- The name, which in our example is “animal,” is always surrounded in double quotes.
- The name in the double quotes can be any valid string. So, you could have a name that looks like this, and it would be perfectly valid JSON

"My animal": "cat“

- You can even place an apostrophe in the name:

"Lindsay's animal": "cat“

- In JSON-“transferring information between platforms in a way that is compatible with both systems.”➔ Portability
- it is important to avoid spaces or special characters for maximum portability

"lindsaysAnimal": "cat“

"myAnimal": "cat"

- Better not to include ‘_’ even.
- The “cat” value in the example has double quotes.
- Unlike the name in the name-value pair, the value does not always have double quotes.
- If our value is a string data type, we must have double quotes.
- In JSON, the remaining data types are number, boolean, array, object, and null. These will not be surrounded in double quotes

- We need curly brackets surrounding our name value pair to make it an object.
- In JSON, multiple name-value pairs are separated by a comma. So, to extend the ani- mal/cat example, let's add a color:

```
{ "animal" : "cat", "color" : "orange" }
```

- { (left curly bracket) says "begin object"
- } (right curly bracket) says "end object"
- [(left square bracket) says "begin array"
-] (right square bracket) says "end array"
- : (colon) says "separating a name and a value in a name-value pair"
- , (comma) says "separating a name-value pair in an object" or "separating a value in an array"; can also be read as "here comes another one"

JSON Syntax

```
{
    title : "This is my title.",
    body : "This is the body."
}
```

```
{
    'title': 'This is my title.',
    'body': 'This is the body.'
}
```

```
{
    "title": "This is my title.",
    "body": "This is the body."
}
```

JSON Syntax

- In JSON, only double quotes are used, and they are absolutely required around the name of the name-value pair.
- Syntax Validation

JSON Formatter & Validator

A formatting tool with options, and a beautiful UI that highlights errors. The processed JSON displays in a window that doubles as a tree/node style visualization tool and a window to copy/paste your formatted code from.

JSON Editor Online

An all-in-one validation, formatting, and visualization tool for JSON. An error indicator is displayed on the line of the error. Upon validation, helpful parsing error information is displayed. The visualization tool displays your JSON in a tree/node format.

JSONLint

A no-bells-and-whistles validation tool for JSON. Simply copy, paste, and click "validate." It also kindly formats your JSON.

- Syntax validation concerns the form of JSON itself, whereas conformity validation concerns a unique data structure.
- For Example syntax validation would be concerned that our JSON is correct (sur- rounded in curly brackets, dividing our name-value pairs with commas).
- Conformity validation would be concerned that our data included a name, breed, and age. Additionally the conformity validation would be concerned that the value of age is a number, and the value of name is a string.

Literal
A value that is written precisely as it is meant to be interpreted

Variable
A value that can be changed and is represented by an identifier, such as *x*

Maximum portability (in data interchange)
Transcending the base portability of the data format by ensuring the data itself will be compatible across systems or platforms

Name-Value Pair
A name-value pair (or key-value pair) is a property or attribute with a name, and a corresponding value

Syntax Validation
Validation concerned with the form of JSON

Conformity Validation
Validation concerned with the unique data structure

←Summary

We also discussed these key concepts:

- JSON is based on the syntactic representation of the properties of JavaScript object literals. This *does not* include the functions of JavaScript object literals.
- In the JSON name-value pair, the name is always surrounded by double quotes.
- In the JSON name-value pair, the value can be a string, number, boolean, null, object, or array.
- The list of name-value pairs in JSON is surrounded by curly brackets.
- In JSON, multiple name-value pairs are separated by a comma.
- JSON files use the *.json* extension.
- The JSON media type is application/json.

JSON Data Types

The common Classification:

1)Primitive

- Numbers (e.g., 5 or 5.09)

— Integer

— Floating-point number

— Fixed-point number

- Characters and strings (e.g., “a” or “A” or “apple”)
- Booleans (i.e., true or false)

2)Composite

-Enumeration data types

Example 3-1. “Let me enumerate the fine qualities of your personality” could be represented in programming as an array literal

```
[
    "witty",
    "charming",
    "brave",
    "bold"
]
```

JSON Data Types

- JSON Data types:
- Object
- String
- Number
- Boolean
- Null
- Array

The JSON Object Data Type

- The JSON object data type is simple. JSON, at its root, is an object. It is a list of name value pairs surrounded in curly braces. When you create a name-value pair within your JSON that is also an object, your JSON will begin to look nested.

Example 3-3. Nested objects

```
{
    "person": {
        "name": "Lindsay Bassett",
        "heightInInches": 66,
        "head": {
            "hair": {
                "color": "light blond",
                "length": "short",
                "style": "A-line"
            },
            "eyes": "green"
        }
    }
}
```

The JSON String Data Type

- The JSON string can be comprised of any of the Unicode characters, and all the characters in that promotional text are valid. A string value must always be surrounded in double quotes

Example 3-5. This code won't work

```
{
    "promo": "Say "Bob's the best!" at checkout for free 8oz bag of kibble."
}
```


- There are quotes inside of the value, and the parser is going to read that first quote character in front of "Bob" in the promotional text as the end of the string.
- Then, when the parser finds the remainder of the text just hanging out there and not belonging to a name value pair, it will produce an error.

The JSON String Data Type

Example 3-6. Using a backslash character to escape quotes inside of strings fixes the problem

```
{
    "promo": "Say \"Bob's the best!\" at checkout for free 8oz bag of kibble."
}
```


- To deal with this, we must escape our quote inside of any string value by preceding it with a back slash character
  - This backslash character will tell the parser that the quote is not the end of the string.
  - Once the parser actually loads the string into memory, any backslash character that precedes a quote character will be removed and the text will come out on the other side as intended.

The JSON String Data Type

- For example, the JSON shown in Example 3-7, which is meant to communicate the location of my Program Files directory, will produce an error.
- To fix this problem, we must escape the backslash character by adding another backslash character, as shown in Example 3-8.

Example 3-7. The backslash used in this code will throw an error

```
{
    "location": "C:\Program Files"
}
```

Example 3-8. The backslash character must be escaped with another backslash character

```
{
    "location": "C:\\Program Files"
}
```

The JSON String Data Type

- In addition to the double quote and backslash characters, you must escape the following characters:
  - \/ (forward slash)
  - \b (backspace)
  - \f (form feed)
  - \t (tab)
  - \n (new line)
  - \r (carriage return)
  - \u followed by hexadecimal characters (e.g., the smiley emoticon \u263A)

The JSON Number Data Type

Example 3-11. Representing numbers in JSON

```
{
    "widgetInventory": 289,
    "sadSavingsAccount": 22.59,
    "seattleLatitude": 47.606209,
    "seattleLongitude": -122.332071,
    "earthsMass": 5.97219e+24
}
```

A number in JSON can be an integer, decimal, negative number, or an exponent

The JSON Boolean Data Type

Example 3-12. Preferences

```
{
    "toastWithBreakfast": false,
    "breadWithLunch": true
}
```

The JSON Null Data Type

- When we have nothing of something, you might think it appropriate to say there is zero of that something.

Example 3-13. Next-door neighbor Bob's might look like this


```
{
    "freckleCount": 0,
    "hairy": true,
    "watchColor": "blue"
}
```

Example 3-14. Mine would look like this

```
{
    "freckleCount": 1,
    "hairy": false,
    "watchColor": null
}
```

The JSON Array Data Type

JSON Schema

- It is a JSON media type for defining the structure of JSON data.
- A declarative language that allows you to annotate and validate JSON Documents.
- JSON schema enables the confident and reliable use of the JSON data format.
- It is a standard providing a format for what JSON data is required for a given application and how to interact with it.
- Applying these standards for a JSON document lets you enforce consistency and data validity across similar JSON data.
- A JSON Validator provides syntax validation, where JSON Schema provides conformity validation.
- JSON Schema can serve as a first line of defense in accepting data or a time (and sanity) saving tool for the party providing the data that ensures their data will conform to what is accepted.

JSON Schema

Benefits

- Describes your existing data format(s)
- Provides clear human and machine readable documentation.
- Validates data which is useful for:

-Automated testing

-Ensuring quality of client submitted data.

JSON Schema

Example 4-1. The 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#"
}
```

The second name-value pair in our JSON Schema Document will be the title (see Example 4-2).

Example 4-2. Format for a document that represents a cat

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

JSON Schema

- 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 essentially a skeleton of the name-value pairs of the JSON we want.
- Instead of a literal value, we have an object that defines the data type, and optionally the description

JSON Schema

Example 4-3. 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"

          }
       }
    }
```

JSON Schema

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

Example 4-4. This JSON conforms to our JSON Schema for "Cat"

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


- A JSON Schema can answer the following three questions for conformity validation:

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.

JSON Schema

- When we ask for data, there are often properties (or fields) that we must have values for, and others that are optional.

Example:

- *When we create a new account on a shopping website, we need to complete a shipping address form. That address form requires our name, street, city, state, and zip code.*
- *Optionally, we can include a company name, apartment number, and a second line for a street address. If we leave out one of the required fields, we cannot move forward with the account creation*

JSON Schema

- To achieve this required logic in JSON schema, we add a fourth name-value pair after $schema, title, and properties.
- This name-value pair has the name “required” and a value of the array data type. The array includes the fields we require.
- *Example:*

We first add another field for “description.” Next, 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. We leave out description because it’s not required

JSON Schema

Example 4-5. Filling out required fields

```
{
  "$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"
  ]
}
```

Example 4-6. Valid JSON

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

This JSON conforms to our JSON Schema for "Cat" with the required fields of name, age and declawed. We are including the optional name-value pair, "description."

Example 4-7. Valid JSON without the description field

```
{
  "name": "Fluffy",
  "age": 2,

 "declawed": false
}
```

We may also leave out the description field, as it's not included in the list of required fields. The JSON in Example 4-7 conforms to our JSON Schema for "Cat" with the required fields of name, age, and declawed.

JSON Schema

- *In Example 4-9, 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.*

Example 4-9. Validating the cat JSON

```
{
  "$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": {

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

Example 4-10. 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 in Example 4-10 is not valid with the cat JSON Schema because the name value exceeds the maxLength, and the age value precedes the minimum.

Example 4-11. This JSON is valid

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

Manipulating JSON data with PHP.

PHP includes the object data type. Objects are defined with a class

Example 9-9. A PHP class representing a cat

```
class Cat
{
    public $name;
    public $breed;
    public $age;
    public $declawed;
}
```


When a class is instantiated, an object is created. The object can then be used in programming logic

Example 9-10. The class is instantiated, and the properties set. An object is created. The last line of code will display "Fluffy Boo."

```
$cat = new Cat();
$cat->name = "Fluffy Boo";
$cat->breed = "Maine Coon";
$cat->age = 2.5;
$cat->declawed = false;

echo $cat->name;
```

Manipulating JSON data with PHP

- PHP also includes built-in support for serializing and deserializing JSON. PHP refers to this as encoding and decoding JSON.
- When we encode something, we convert it to a coded (unreadable) form.
- When we decode something, we convert it back into a readable form.
- From the perspective of PHP, JSON is in a coded format. Therefore, to serialize JSON, the "***json_encode***" function is called, and to deserialize JSON, the "***json_decode***" function is called.

Manipulating JSON data with PHP

- Serializing JSON (PHP→JSON)
- In PHP, we can quickly serialize PHP objects with the built-in support for JSON.
- In this section, we will create a PHP object representing an address, and serialize to JSON.

In Example 9-11, we first create a class representing account. Then, we create a new instance of the class for Bob Barker's account An account object is created. Finally, on the last line, we call the built-in function "json_encode" to serialize the account object (results shown in Example 9-12).

Manipulating JSON data with PHP

Serializing JSON (PHP→JSON)

Example 9-11. Creating and serializing an address

```
<?php
class Account {
    public $firstName;
    public $lastName;
    public $phone;
    public $gender;
    public $addresses;
    public $famous;
}

class Address {
    public $street;
    public $city;
    public $state;
    public $zip;
}

$address1 = new Address();
$address1->street = "123 fakey st";
$address1->city = "Somewhere";
$address1->state = "CA";
$address1->zip = 96027;

$address2 = new Address();
$address2->street = "456 fake dr";
$address2->city = "Some Place";
$address2->state = "CA";
$address2->zip = 96345;

$account = new Account();
$account->firstName = "Bob";
$account->lastName = "Barker";
$account->gender = "male";
$account->phone = "555-555-5555";
$account->famous = true;
$account->addresses = array ($address1, $address2);

$json = json_encode($account);

?>
```

Example 9-12. The JSON result from json_encode($account)

```
{
    "firstName": "Bob",
    "lastName": "Barker",
    "phone": "555-555-5555",
    "gender": "male",
    "addresses": [
        {
            "street": "123 fakey st",
            "city": "Somewhere",
            "state": "CA",
            "zip": 96027
        },
        {
            "street": "456 fake dr",
            "city": "Some Place",
            "state": "CA",
            "zip": 96345
        }
    ],
    "famous": true
}
```

Manipulating JSON data with PHP

Deserializing JSON (JSON→PHP)

- To deserialize JSON, we use the json_decode function.
- This does not have built-in support for deserializing the JSON to a specified PHP object, such as the Account class.
- So, we must do a little processing to reshape our data back into the PHP object.
- Let's add a new function to the account object, to load up its properties from a JSON string

Manipulating JSON data with PHP

Deserializing JSON (JSON→PHP)

Example 9-13. The "loadFromJSON" function accepts a JSON string for a parameter, calls the built in "json_decode" function to deserialize to a generic PHP object, and maps the name-value pairs to the Account properties by name in the foreach loop

```
class Account {
    public $firstName;
    public $lastName;
    public $phone;
    public $gender;
    public $addresses;
    public $famous;

    public function loadFromJSON($json)
    {
        $object = json_decode($json);
        foreach ($object AS $name => $value)
        {
            $this->{$name} = $value;
        }
    }
}
```

Next, we can create a new Address object, and call the new "loadFromJSON" function.

Example 9-14. Calling our new "loadFromJSON" function to deserialize the account JSON back into the Address object. The last line will display "Bob Barker."

```
$json = json_encode($account);

$deserializedAccount = new Account();
$deserializedAccount->loadFromJSON($json);

echo $deserializedAccount->firstName . " " . $deserializedAccount->lastName;
```

Manipulating JSON data with PHP

Requesting JSON

- To make an HTTP request for a resource with PHP, we can use the built in function "file_get_contents."
- This function returns the resource body as a string.
- We can the deserialize the string to a PHP object.

Example 9-15. Resource at URL http://localhost:5984/accounts/ddc14efcf71396463f53c0f8800019ea from my local CouchDB API.

```
{
    "_id": "ddc14efcf71396463f53c0f8800019ea",
    "_rev": "6-69fd853972074668f99b88a86aa6a083",
    "address": {
        "street": "123 fakey ln",
        "city": "Some Place",
        "state": "CA",
        "zip": "96037"
    },
    "gender": "female",
    "famous": false,
    "age": 28,
    "firstName": "Mary",
    "lastName": "Thomas"
}
```


Example 9-16. Calling the built in "file_get_contents" function to get the account JSON resource from the CouchDB API. Next, a new account object is created and our "loadFromJSON" function is called to deserialize. The last line will display "Mary Thomas."

```
$url = "http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f700076c";
$json = file_get_contents($url);

$deserializedAccount = new Account();
$deserializedAccount->loadFromJSON($json);

echo $deserializedAccount->firstName . " " . $deserializedAccount->lastName;
```