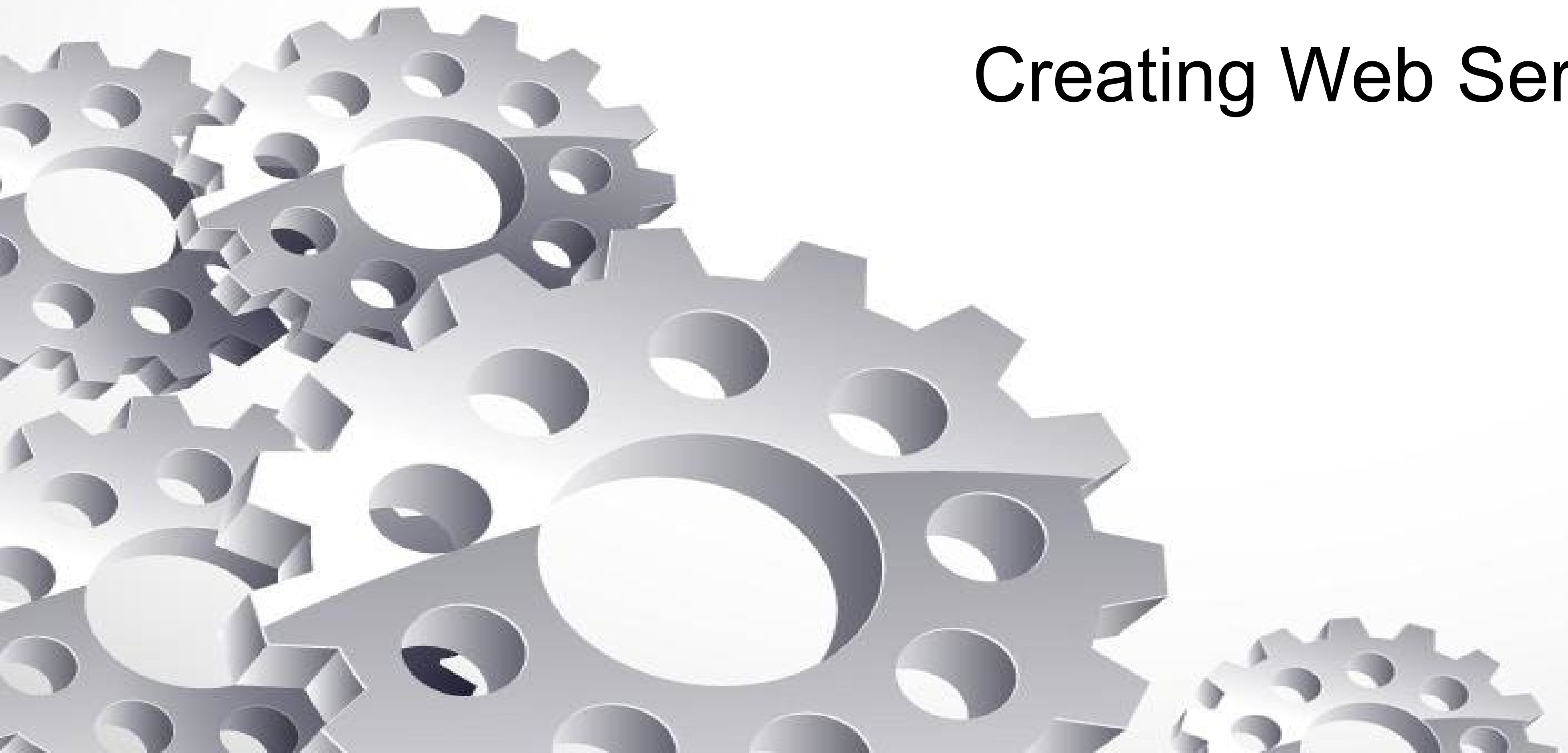
Web BackEnd

Creating Web Service using PHP



Introduction to Web Services

Normally: a method of communications over the WWW

W3C Definition: a software system

- Designed to support interoperable machine-to-machine interaction
- Over network
- An interface with a certain format
- Other systems interact with the Web Service

Means of exposing functionality or data

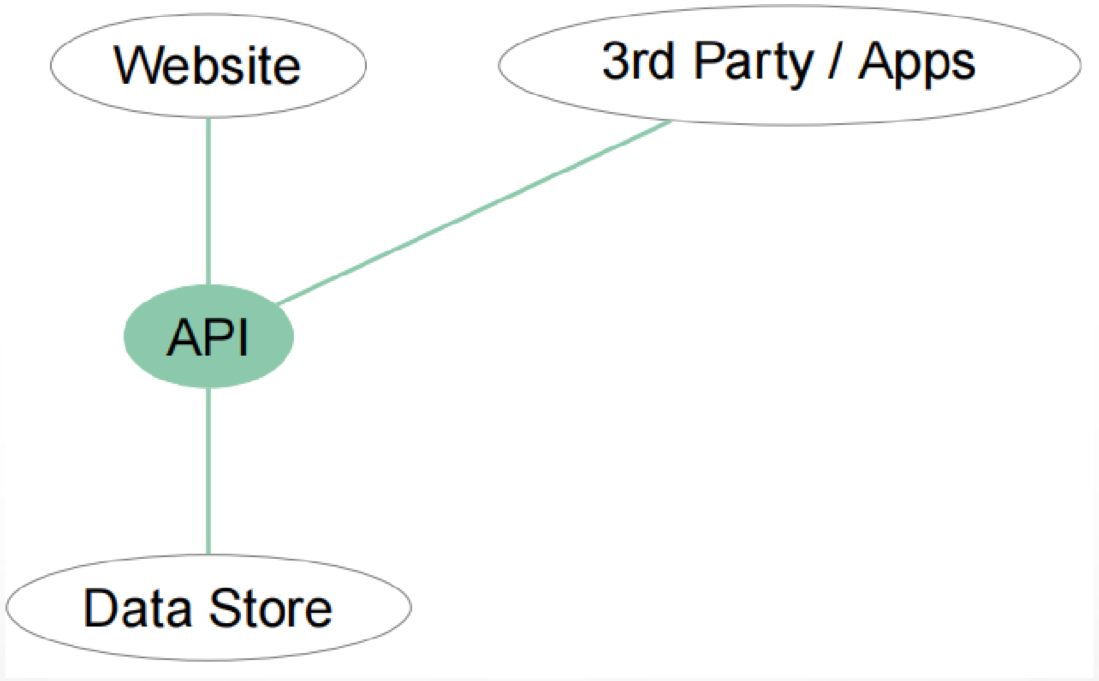
A lot like a web page, but the target is for other machines to consume

Integration between applications

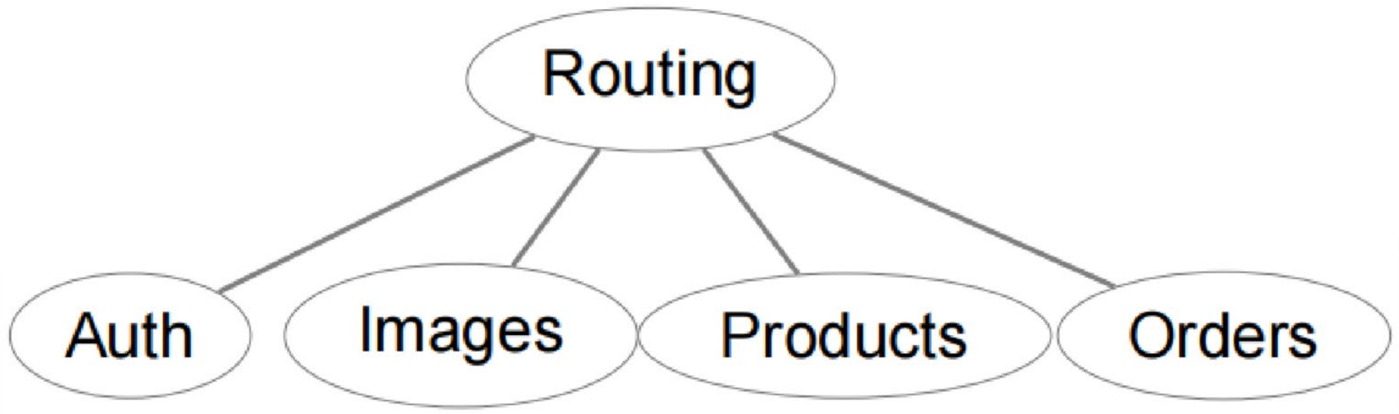
Separation within an application

Concept of Integration Between Applications

- It is common to use an API internally as well as exposing it



Concept of Separation Within an Application



Different ways of using web service:

- RPC: Remote Procedure Call
  - Strong Coupling
- SOA: Service-oriented Architecture
  - This is a good choice.
- REST: Representational State Transfer
  - Another choice. Covered in this topic

Building Blocks

You can make an API from any tools you like

- Existing MVC setup
- Simple PHP (covered in this topic)
- Framework modules
- Component library
- etc...

REST & RESTful WEB SERVICE

REST stands for Representational State Transfer

- It is an architectural pattern for developing web services as opposed to a specification
- REST web services communicate over the HTTP specification, using HTTP vocabulary
  - Methods (GET, POST, etc.)
  - HTTP URI syntax (paths, parameters, etc.)
  - HTTP Response codes.
  - Media types (xml, json, html, plain text, etc)

HTTP-REST Request Basics

The HTTP request is sent from the client.

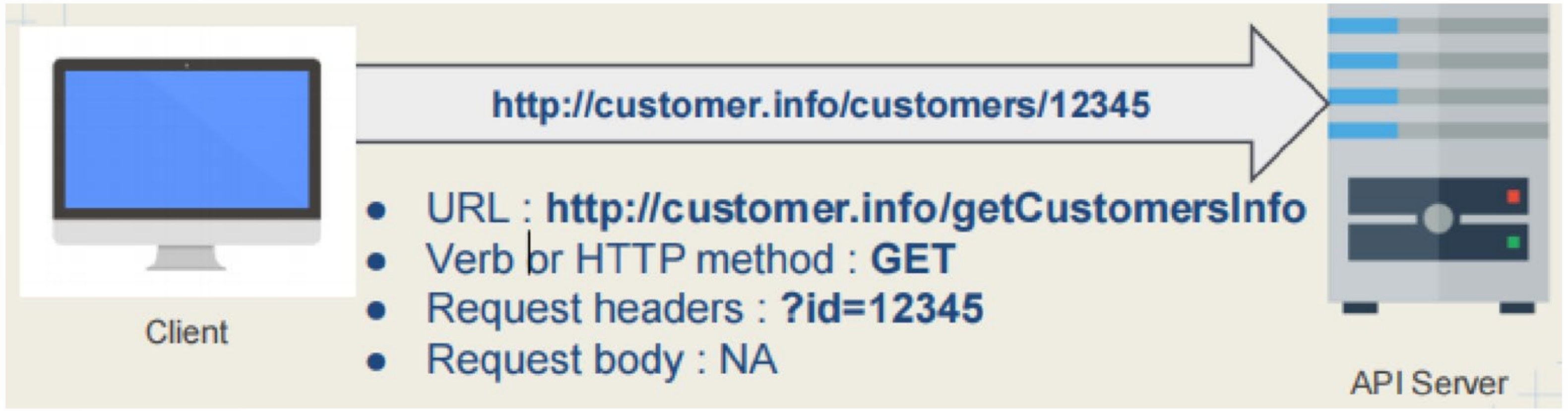
– Identifies the location of a resource (URL) ie http://localhost/api/getAllUsers.php

– Specifies the verb, or HTTP method (GET, POST, etc). ie ‘get’ or ‘post’

– Supplies optional request headers (GET - e.g. name-value pairs)

– Supplies an optional request body (POST - e.g. form parameters, attachments, etc.)

HTTP-REST Request Basics



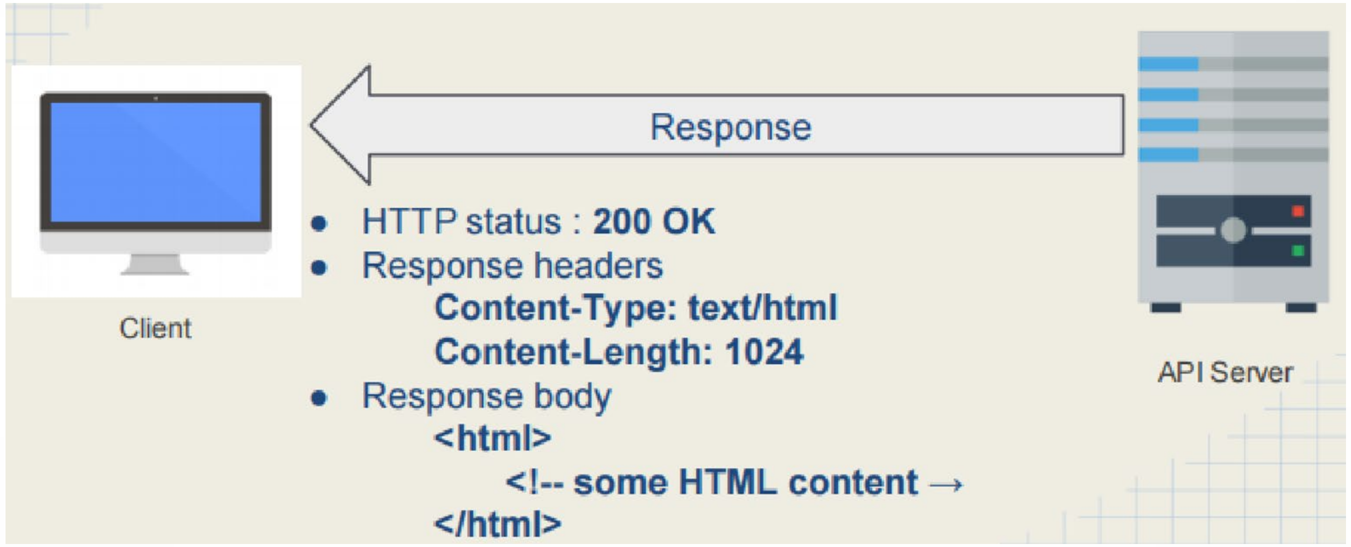
- URL : **http://customer.info/getCustomersInfo**
- Verb or HTTP method : **GET**
- Request headers : **?id=12345**
- Request body : NA

HTTP-REST Response Basics

The HTTP response is sent from the server.

- Gives the status of the processed request. (HTTP code)
- Supplies response headers (name-value pairs) that
- provide additional information about the response.
- Supplies an optional response body that identifies
- additional data to be downloaded to the client (html, xml, binary data, etc.)

HTTP-REST Response Basics



- HTTP status : **200 OK**
- Response headers
  Content-Type: text/html
  Content-Length: 1024
- Response body

```
<html>
    <!-- some HTML content →
</html>
```

REST Characteristics

Resources

- URI: Every resource is uniquely addressable using URIs.

Constraints

- Client-Server
- Stateless
- Cacheable
- Layered
- Code on demand (optional)
- Uniform Interface
  - Method : Use only HTTP methods (GET, POST, PUT, DELETE)
  - Representation : json, xml, etc

cont...

Client-Server

1. Separation of concerns.
2. Client and server are independent from each other.
3. Client doesn’t know anything about the resource which is kept in the server.
4. Server responds as long as the right requests come in.

➔ Goal: Platform independency and to improve scalability.

Stateless

1. Each request is independent from other requests.
2. No client session data or any context stored on the server.
3. Every request from client stores the required information, so that the server can respond.
4. If there are needs for session-specific data, it should be held and maintained by the client and transferred to the server with each request as needed.
5. A service layer which doesn’t have to maintain client sessions is much easier to scale.

Cacheable

1. HTTP responses must be cacheable by the clients.
2. Important for performance.
3. If a new request for the resources comes within a while, then the cached response will be returned.

Layered System

1. There can be many intermediaries between you and the server you are connecting to.
2. Actually, a client does not know if it is connected to the last server or an intermediary server.
3. Intermediaries may improve performance by caching and message passing.
4. Intermediary servers can increase scalability by load-balancing and can force clients to form some sort of security policies.
5. This structure can be used when encapsulation is needed.

Code on Demand (optional)

1. Servers can send some kind of executable scripts to the client_x0002_side in order to increase or change the functionality on the client side.

2. This may cause low visibility, so it is the only optional constraint.

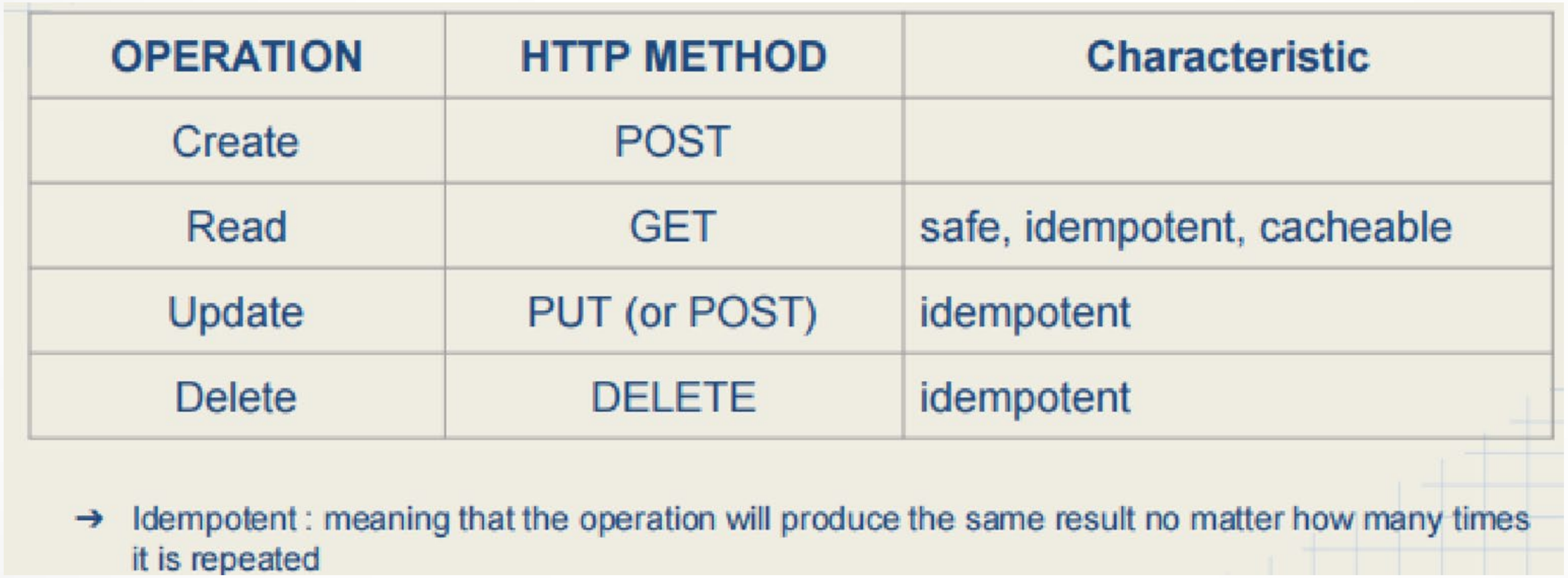
Uniform Interface

1. All resources share a uniform interface for the transfer of state between client and resource, consisting of

    o HTTP Method

    o HTTP Status Code

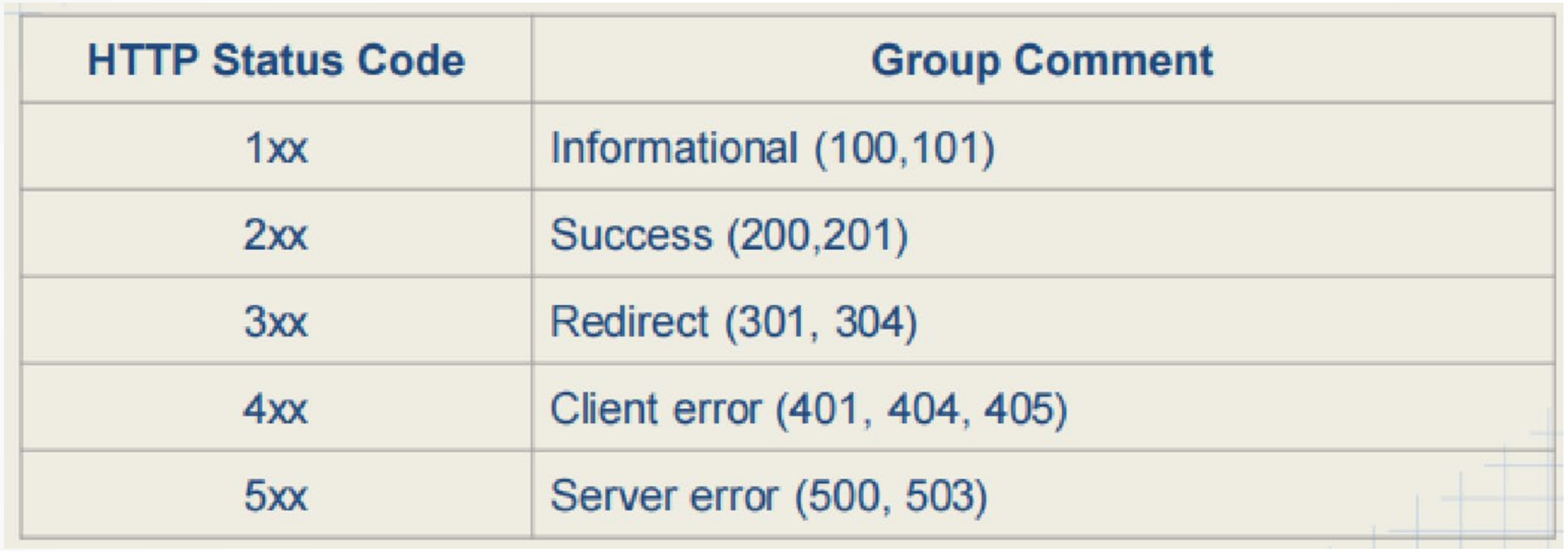
HTTP Methods



OPERATION	HTTP METHOD	Characteristic
Create	POST	
Read	GET	safe, idempotent, cacheable
Update	PUT (or POST)	idempotent
Delete	DELETE	idempotent

→ Idempotent : meaning that the operation will produce the same result no matter how many times it is repeated

HTTP Status Code



HTTP Status Code	Group Comment
1xx	Informational (100,101)
2xx	Success (200,201)
3xx	Redirect (301, 304)
4xx	Client error (401, 404, 405)
5xx	Server error (500, 503)

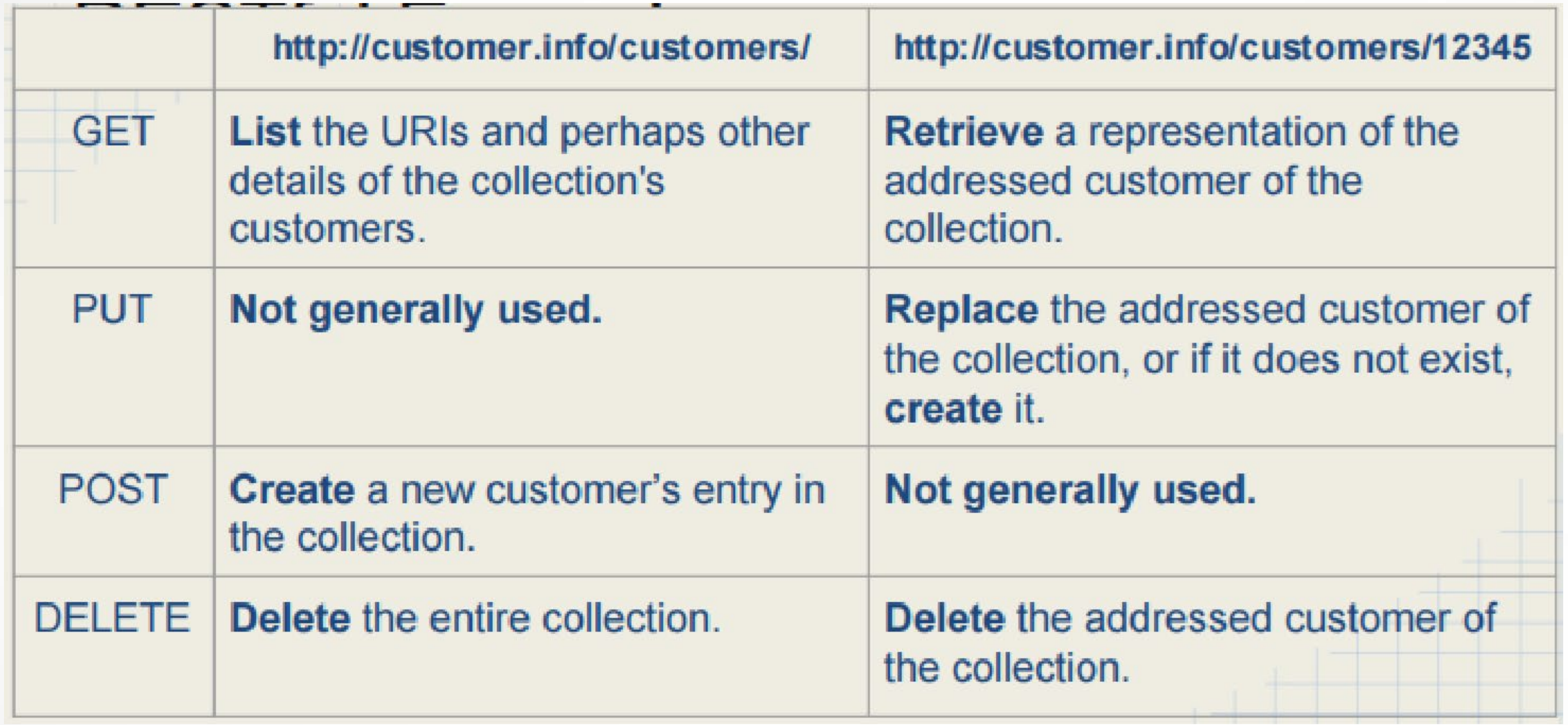
RESTful WEB SERVICE

- (will be covered with more detail in the next topic)

Constraints

- Client-Server
- Stateless
- Cacheable
- Layered
- Code on demand (optional)
- Uniform Interface

➔ If a service does not include all constraints out of «Code on Demand», it is not a RESTful web service



	http://customer.info/customers/	http://customer.info/customers/12345
GET	**List** the URIs and perhaps other details of the collection's customers.	**Retrieve** a representation of the addressed customer of the collection.
PUT	**Not generally used.**	**Replace** the addressed customer of the collection, or if it does not exist, **create** it.
POST	**Create** a new customer's entry in the collection.	**Not generally used.**
DELETE	**Delete** the entire collection.	**Delete** the addressed customer of the collection.

Let’s Begin

You can make an API from any tools you like

- Existing MVC setup
- Simple PHP (as in these examples)
- Framework modules
- Component library

*we will create rest api using simple PHP in this example

My First Web Service using PHP

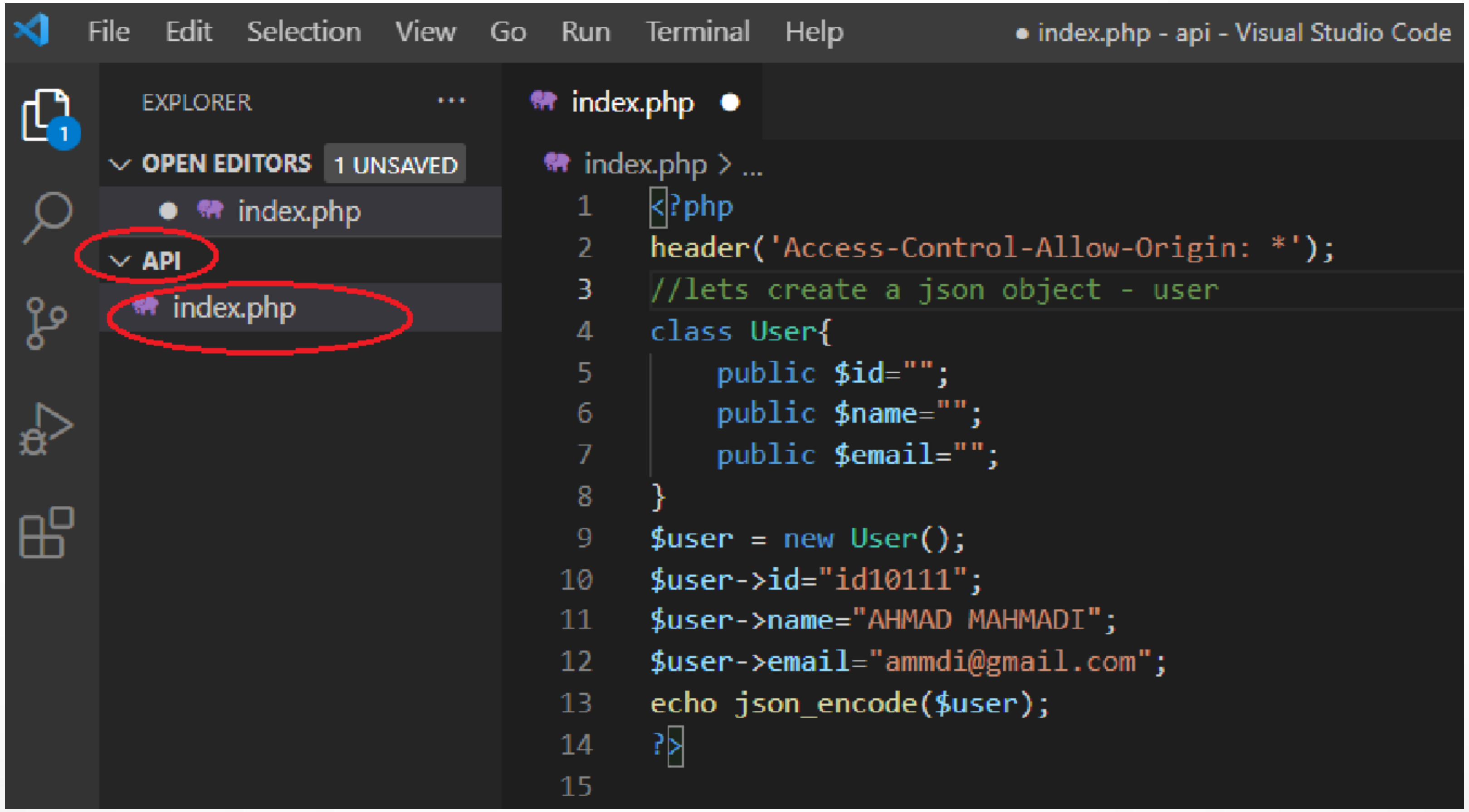
Create the first web service that return JSON data format using PHP

Steps:

1. Create a new folder name “api” in xampp htdocs
2. create new php file “index.php” in folder “api”
3. Write this code in index.php

    - PHP has some built-in functions to handle JSON.

    - Objects in PHP can be converted into --JSON by using the PHP function json_encode();



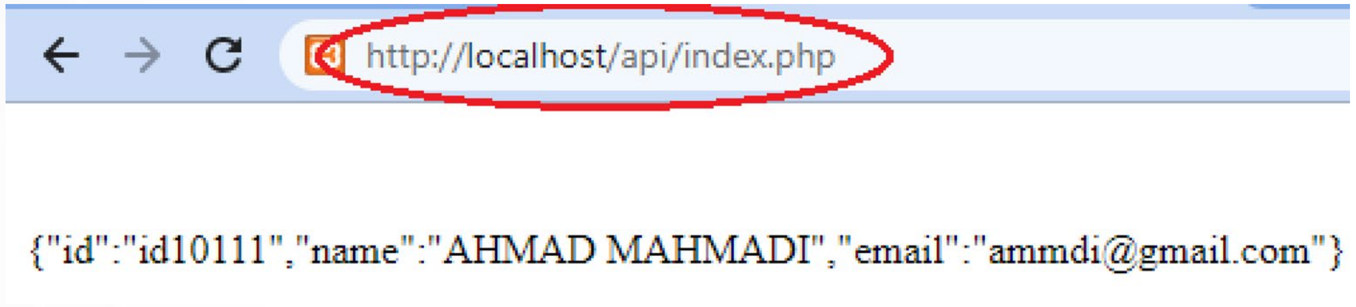
```php
<?php
header('Access-Control-Allow-Origin: *');
//lets create a json object - user
class User{
    public $id="";
    public $name="";
    public $email="";
}
$user = new User();
$user->id="id10111";
$user->name="AHMAD MAHMADI";
$user->email="ammdi@gmail.com";
echo json_encode($user);
?>
```

cont...

3. Write this code in index.php

```
<?php
header('Access-Control-Allow-Origin: *');
  class User{                      //lets create a json object - user
    public $id="";
    public $name="";
    public $email="";
}
$user = new User();
$user->id="id10111";
$user->name="AHMAD MAHMADI";
$user->email="ammdi@gmail.com";
echo json_encode($user);
?>
```

Consume Your Web Service : using web browser - output



```
{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"}
```

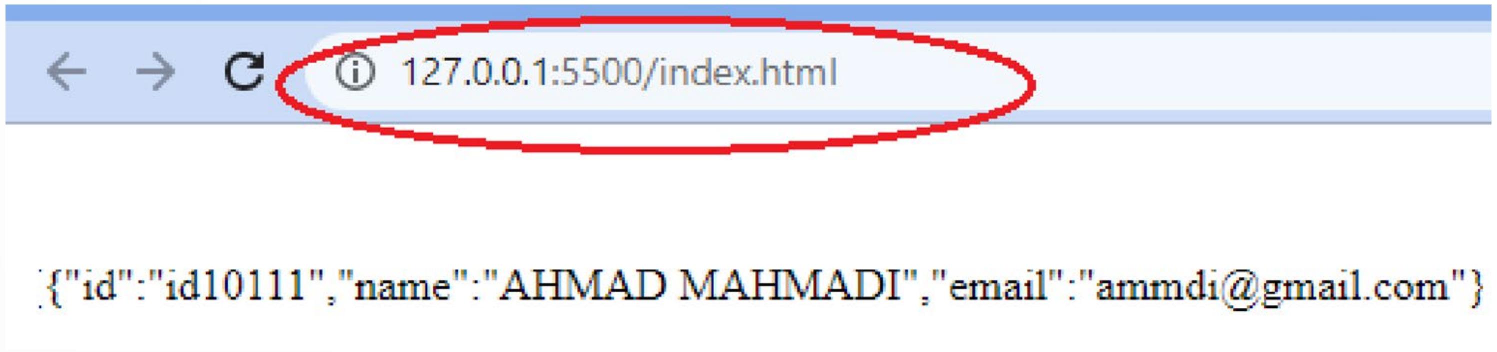
Consume Your Web Service : using JavaScript in “consumer.html”

- Create new html file --> “consumer.html”
- Write this JavaScript on the client html
- using an AJAX call to request the PHP file from the example above:
- Use JSON.parse() to convert the result into a JavaScript object:

Consume Your Web Service : (consumer.html) using JavaScript

```
<html>
<body>
<h3>hello index</h3>
<p id="consume_webservice">demo</p>
</body>
<script>
            const xhr = new XMLHttpRequest();
            xhr.open('get','http://localhost/api/index.php',true);
            xhr.send();
            xhr.onload = function(){
                console.log(xhr.responseText);
                document.querySelector('#consume_webservice').innerHTML = xhr.responseText;
           }
</script>
```

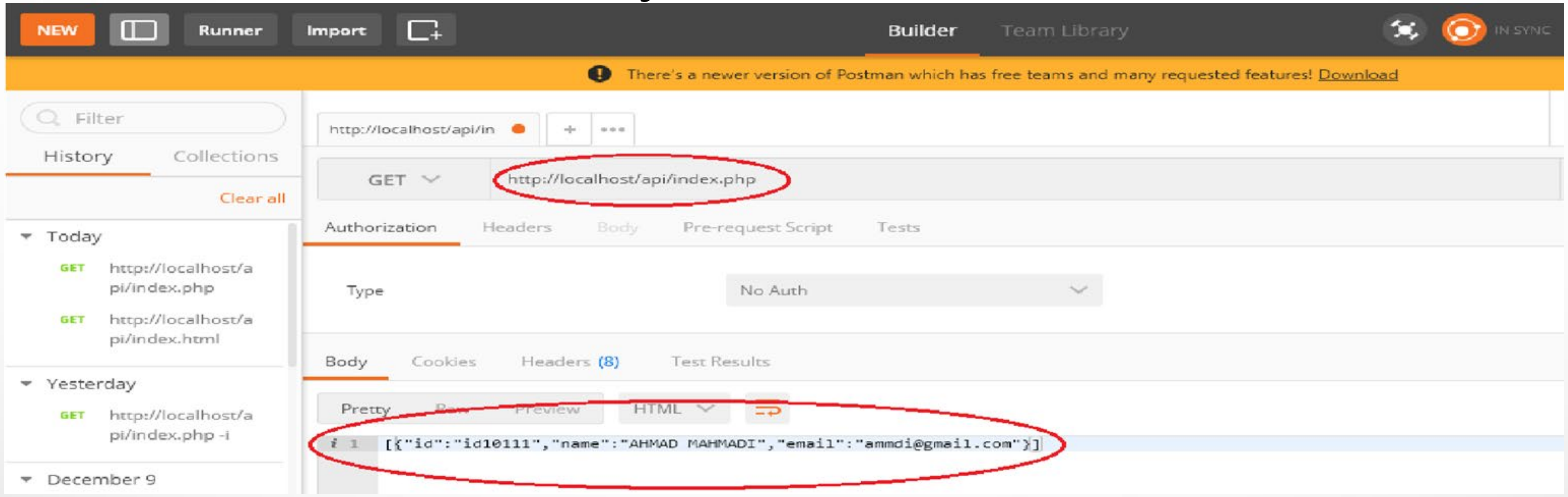
Consume Your Web Service : using JavaScript in “consumer.html” - output



```
{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"}
```

Consume Your Web Service : using Postman

- Postman is an interactive and automatic tool for verifying the APIs
- other than Postman, you can also use ie cURL, 25
-



There's a newer version of Postman which has free teams and many requested features! Download

GET http://localhost/api/index.php

GET http://localhost/api/index.html

GET http://localhost/api/index.php -i

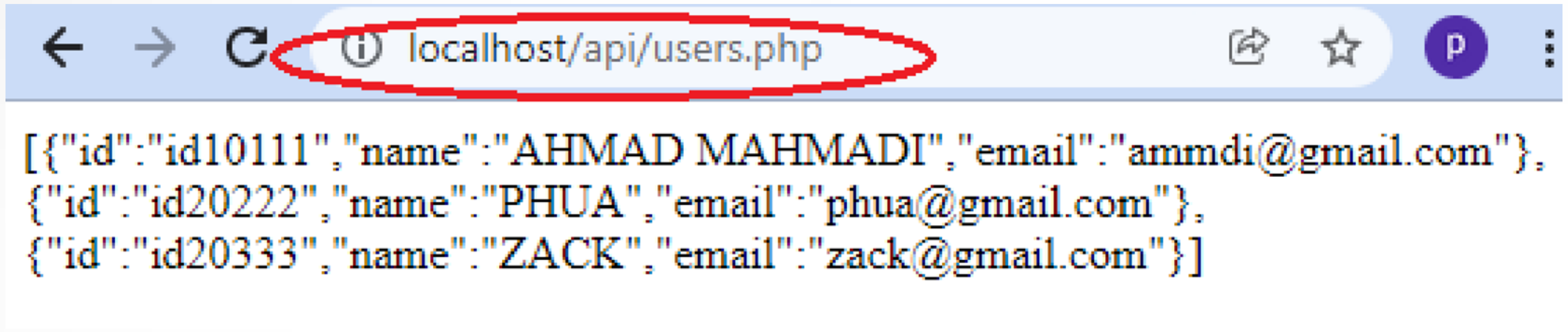
Type: No Auth

```
[{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"}]
```

Let’s code “users.php” that returns array of user (json format)

```
<?php
header('Access-Control-Allow-Origin: *');

//lets create a json object - user
class User{
    public $id="";
    public $name="";
    public $email="";
}

$users = array();
$user = new User();
$user->id="id10111";
$user->name="AHMAD MAHMADI";
$user->email="ammdi@gmail.com";
array_push($users,$user);
$user = new User();
$user->id="id20222";
$user->name="PHUA";
$user->email="phua@gmail.com";
array_push($users,$user);
$user = new User();
$user->id="id20333";
$user->name="ZACK";
$user->email="zack@gmail.com";
array_push($users,$user);

echo json_encode($users);
?>
```

Consume Your Web Service : using web browser



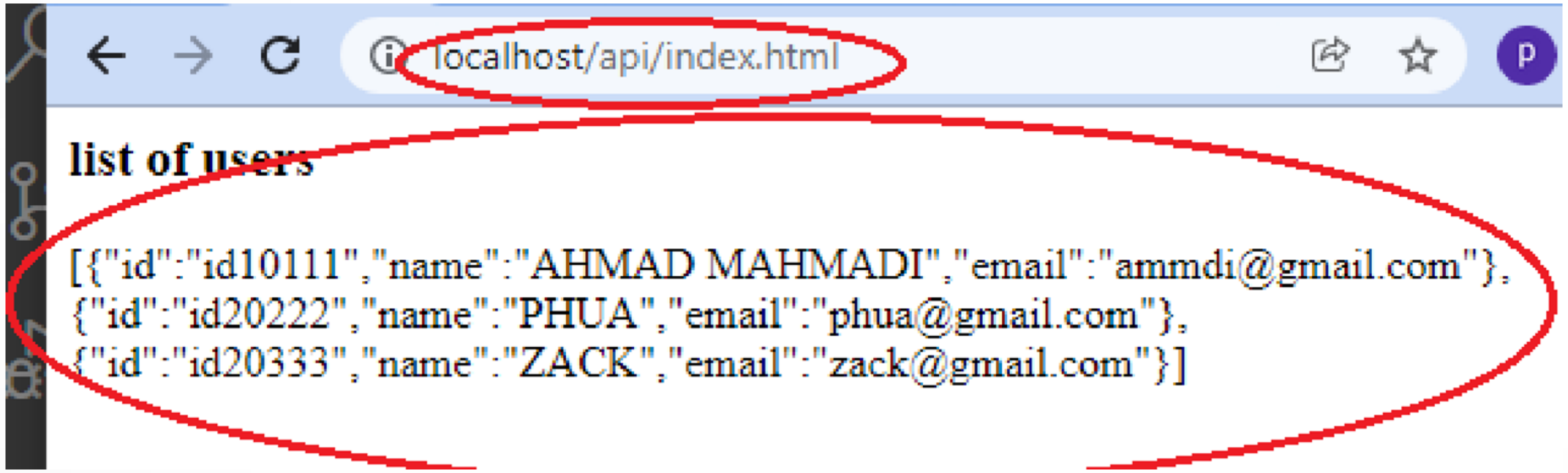
localhost/api/users.php

```
[{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"},
{"id":"id20222","name":"PHUA","email":"phua@gmail.com"},
{"id":"id20333","name":"ZACK","email":"zack@gmail.com"}]
```

Consume Your Web Service : (consumer.html) using JavaScript

```
<html>
<body>
<h3>hello index</h3>
<p id="consume_webservice">demo</p>
</body>
<script>
            const xhr = new XMLHttpRequest();
            xhr.open('get','http://localhost/api/users.php',true);
            xhr.send();
            xhr.onload = function(){
                console.log(xhr.responseText);
                document.querySelector('#consume_webservice').innerHTML = xhr.responseText;
           }
</script>
```

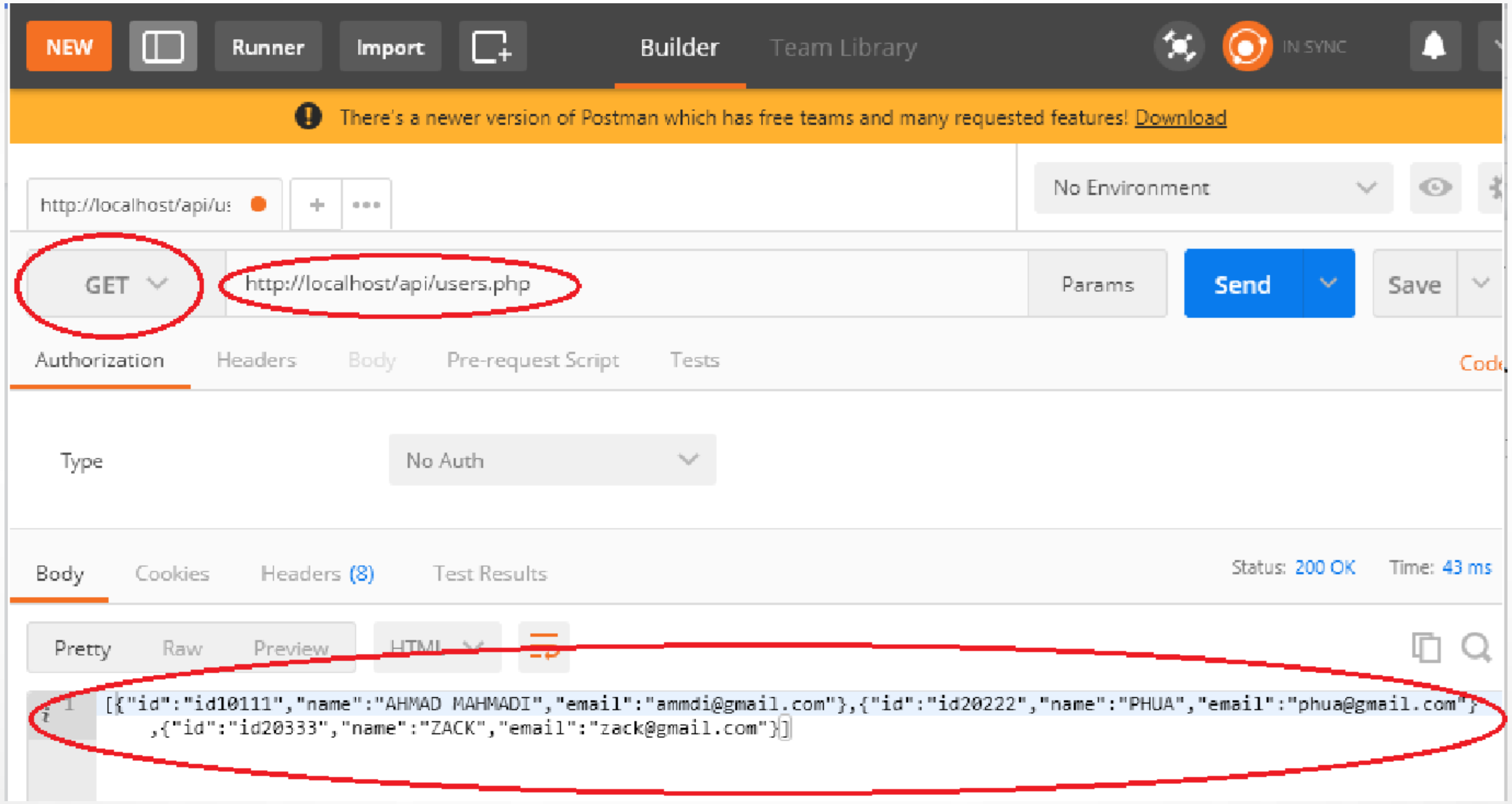
Consume Your Web Service : using JavaScript in "consumer.html" - output



list of users

```
[{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"},
{"id":"id20222","name":"PHUA","email":"phua@gmail.com"},
{"id":"id20333","name":"ZACK","email":"zack@gmail.com"}]
```

Consume Your Web Service : using Postman



There's a newer version of Postman which has free teams and many requested features! Download

http://localhost/api/users.php

GET http://localhost/api/users.php

Type: No Auth

Status: 200 OK Time: 43 ms

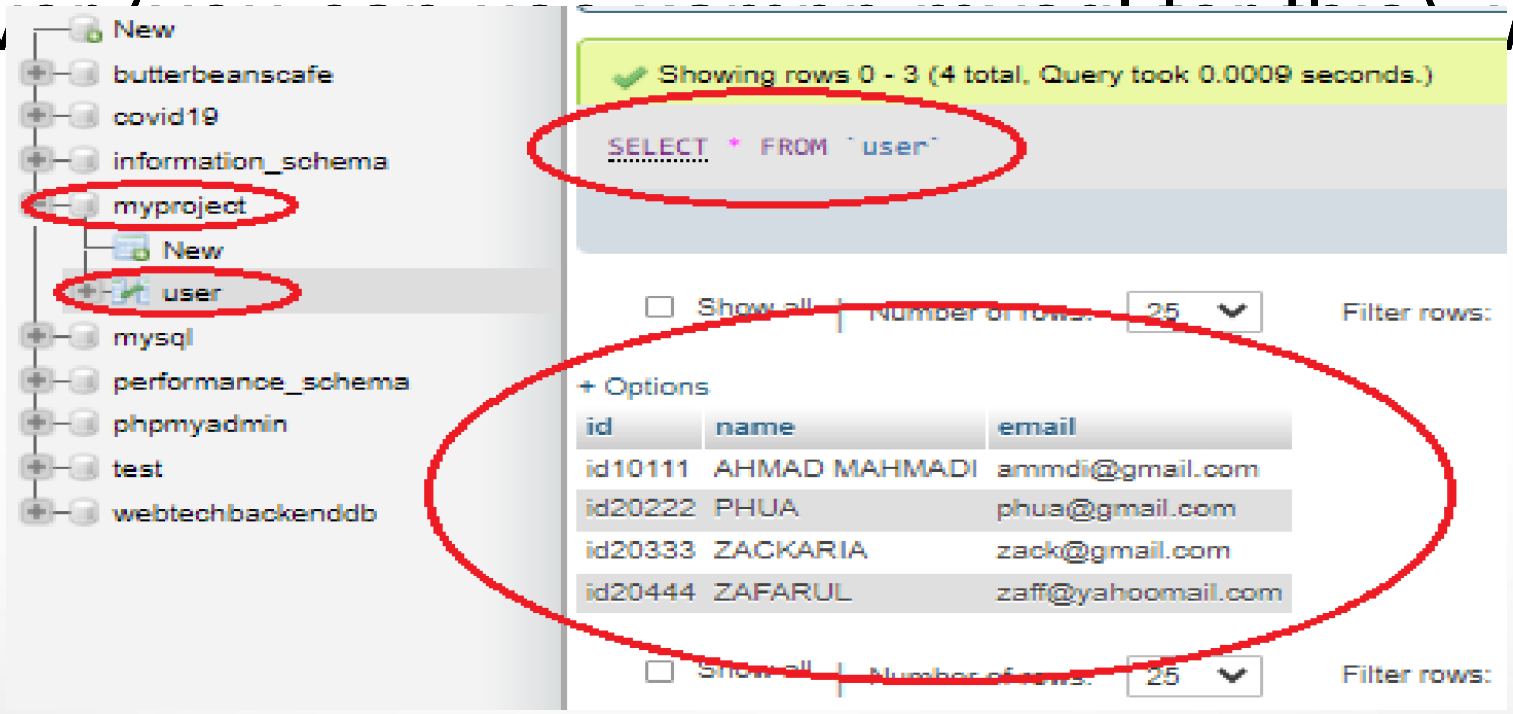
```
[{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"},{"id":"id20222","name":"PHUA","email":"phua@gmail.com"}
,{"id":"id20333","name":"ZACK","email":"zack@gmail.com"}]
```

Let’s Create A Web Services (5) using PHP that connect with Database

1. getAllUser_db.php - that will return all user records in table “user”
2. insertUser_db.php - that will insert the user info into table “user”
3. deleteUser_db.php - that will delete user record from table “user” based on user id
4. updateUser_db.php - that will update the user data for the particular user id
5. getSingleUser_db.php - that will return single user record based on user id

Create a Web Service with PHP Database

- PHP is a server side programming language, and can be used to access a database.
- Please prepare a database (name “MyProject”) on your serv which have a ta



New
butterbeanscafe
covid19
information_schema
myproject
New
user
mysql
performance_schema
phpmyadmin
test
webtechbackenddb

Showing rows 0 - 3 (4 total, Query took 0.0009 seconds.)

```
SELECT * FROM `user`
```

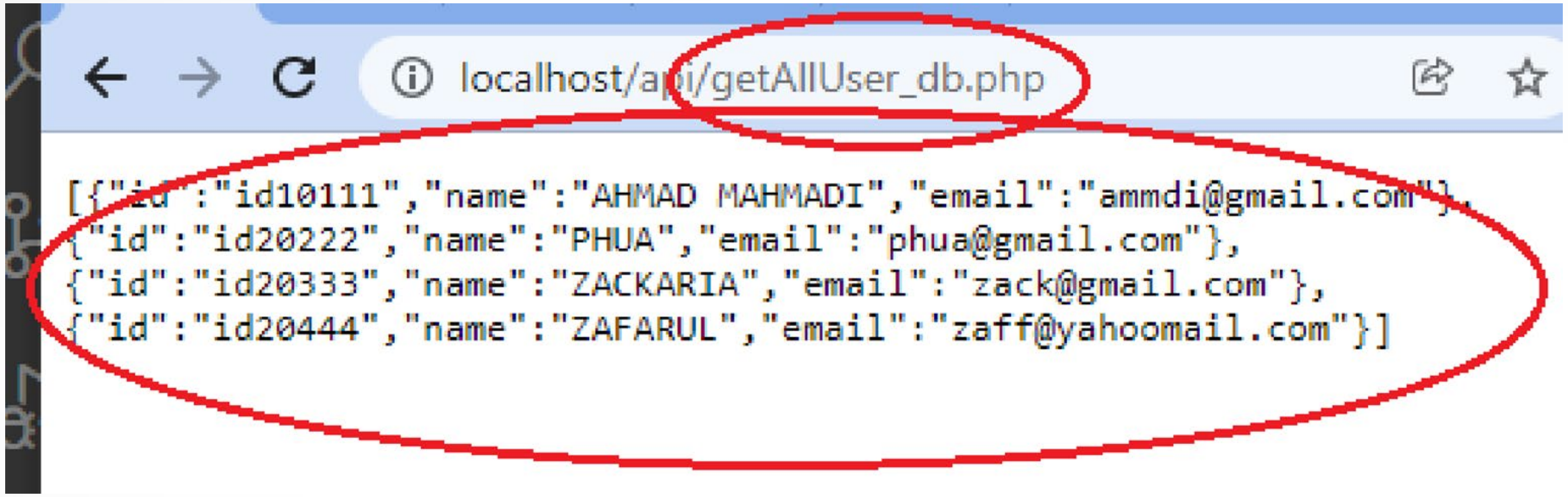
id	name	email
id10111	AHMAD MAHMADI	ammdi@gmail.com
id20222	PHUA	phua@gmail.com
id20333	ZACKARIA	zack@gmail.com
id20444	ZAFARUL	zaff@yahoomail.com

code in “getAllUser_db.php”

```php
<?php
header('Access-Control-Allow-Origin: *');
header("Content-Type: application/json; charset=UTF-8");

    class db{
        // Properties
        private $host = 'localhost';
        private $user = 'root';
        private $password = '';
        private $dbname = 'myproject';

        // Connect
        public function connect(){
            $mysql_connect_str = "mysql:host=$this->host;dbname=$this->dbname";
            $dbConnection = new PDO($mysql_connect_str, $this->user, $this->password);
            $dbConnection->setAttribute( PDO::ATTR_ERRMODE,
                                         PDO::ERRMODE_EXCEPTION);

            return $dbConnection;
        }
    }
```

```php
$sql = "SELECT * FROM user";

try {
    // Get DB Object
    $db = new db();
    // Connect
    $db = $db->connect();

    $stmt = $db->query($sql);
    $user = $stmt->fetchAll(PDO::FETCH_OBJ);
    $db = null;
    echo json_encode($user);
} catch (PDOException $e) {
    $data = array(
        "status" => "fail"
    );
    echo json_encode($data);
}


?>
```

Consume Your Web Service : using web browser getAppUser_db.php

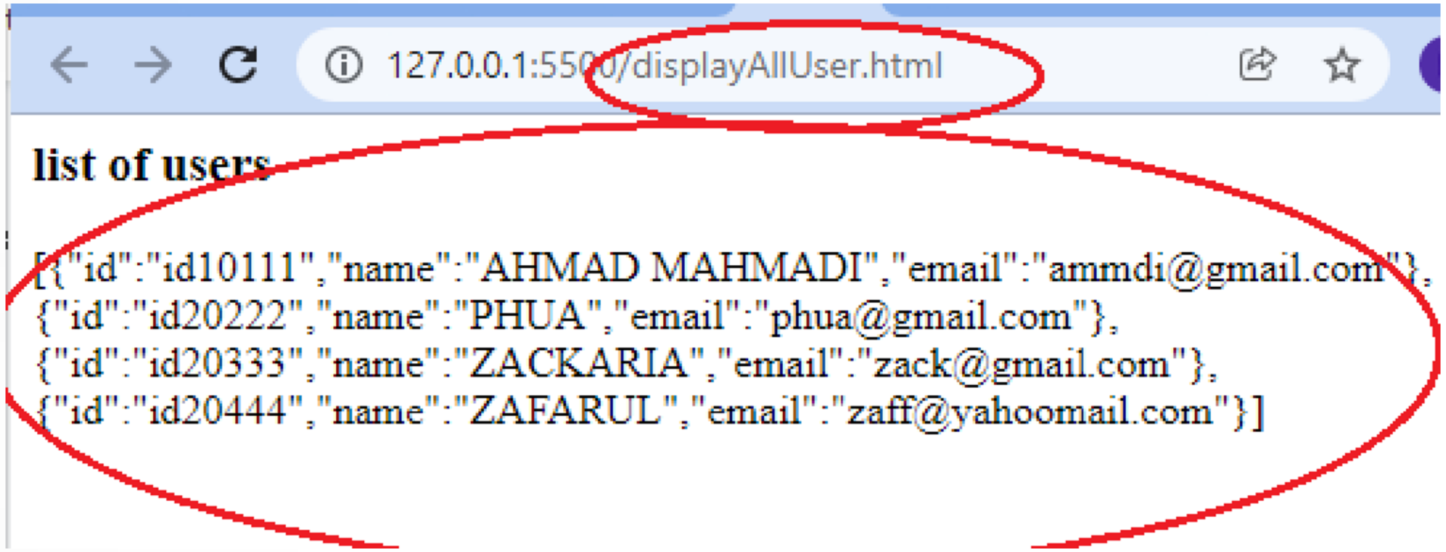


```
[{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"},
{"id":"id20222","name":"PHUA","email":"phua@gmail.com"},
{"id":"id20333","name":"ZACKARIA","email":"zack@gmail.com"},
{"id":"id20444","name":"ZAFARUL","email":"zaff@yahoomail.com"}]
```

Consume Your Web Service : using JavaScript in “displayAllUser.html” - code

```
<html>
<body>
<h3>list of users</h3>
<p id="consume_webservice">demo</p>
</body>
<script>
        const xhr = new XMLHttpRequest();
        xhr.open('get','http://localhost/api/getAllUser_db.php',true);
        xhr.send();
        xhr.onload = function(){
            console.log(xhr.responseText);
            document.querySelector('#consume_webservice').innerHTML = xhr.responseText;
        }
</script>
```

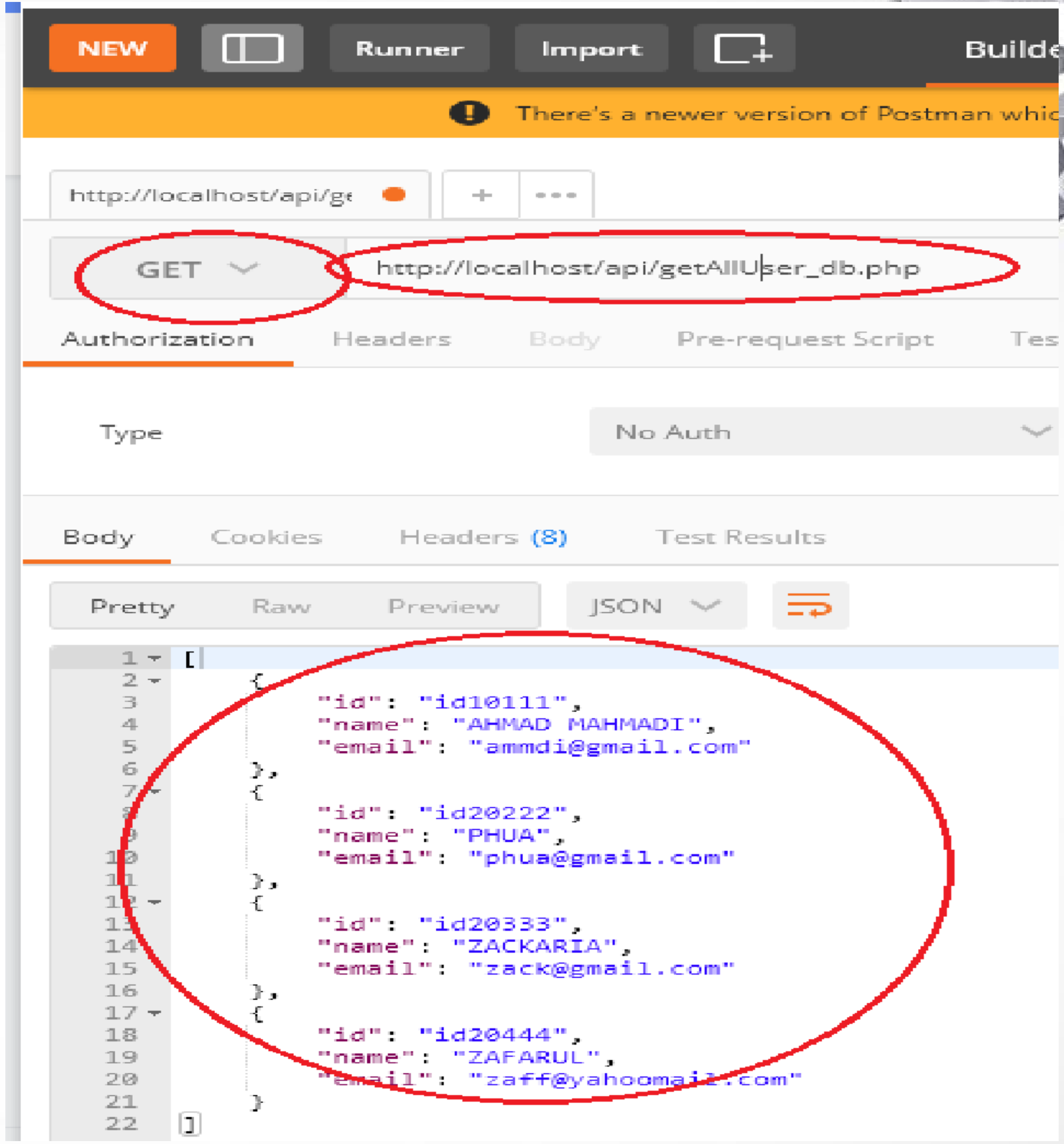
Consume Your Web Service : using JavaScript in “displayAllUser.html” - output



list of users

```
[{"id":"id10111","name":"AHMAD MAHMADI","email":"ammdi@gmail.com"},
{"id":"id20222","name":"PHUA","email":"phua@gmail.com"},
{"id":"id20333","name":"ZACKARIA","email":"zack@gmail.com"},
{"id":"id20444","name":"ZAFARUL","email":"zaff@yahoomail.com"}]
```

Consume Your Web Service : using Postman



GET http://localhost/api/getAllUser_db.php

Type: No Auth

```
[
    {
        "id": "id10111",
        "name": "AHMAD MAHMADI",
        "email": "ammdi@gmail.com"
    },
    {
        "id": "id20222",
        "name": "PHUA",
        "email": "phua@gmail.com"
    },
    {
        "id": "id20333",
        "name": "ZACKARIA",
        "email": "zack@gmail.com"
    },
    {
        "id": "id20444",
        "name": "ZAFARUL",
        "email": "zaff@yahoomail.com"
    }
]
```

code in “insertUser_db.php”

```
<?php
Header('Access-Control-Allow-Origin: *'); //for allow any domain, insecure
Header('Access-Control-Allow-Headers: *'); //for allow any headers, insecure
Header('Access-Control-Allow-Methods: GET, POST, OPTIONS, PUT, DELETE');

   class db{
      // Properties
      private $host = 'localhost';
      private $user = 'root';
      private $password = '';
      private $dbname = 'myproject';

      // Connect
      public function connect(){
         $mysql_connect_str = "mysql:host=$this->host;dbname=$this->dbname";
         $dbConnection = new PDO($mysql_connect_str, $this->user, $this-
>password);
         $dbConnection->setAttribute(PDO::ATTR_ERRMODE,
                                                PDO::ERRMODE_EXCEPTION);

         return $dbConnection;
      }
   }
```

```
   $id = $_POST["id"];
   $name = $_POST["name"];
   $email = $_POST["email"];

try {
   $sql = "INSERT INTO user (name,id,email) VALUES
                                              (:name,:id,:email)";
   $db = new db();
   // Connect
   $db = $db->connect();
   $stmt = $db->prepare($sql);
   $stmt->bindValue(':name', $name);
   $stmt->bindValue(':id', $id);
   $stmt->bindValue(':email', $email);

   $stmt->execute();
   $count = $stmt->rowCount();
   $db = null;

   $data = array(
      "status" => "success",
      "rowcount" =>$count
   );
   echo json_encode($data);
} catch (PDOException $e) {
   $data = array(
      "status" => "fail"
   );
   echo json_encode($data);
}
?>
```

Consume Your Web Service : using web browser insertUser_db.php

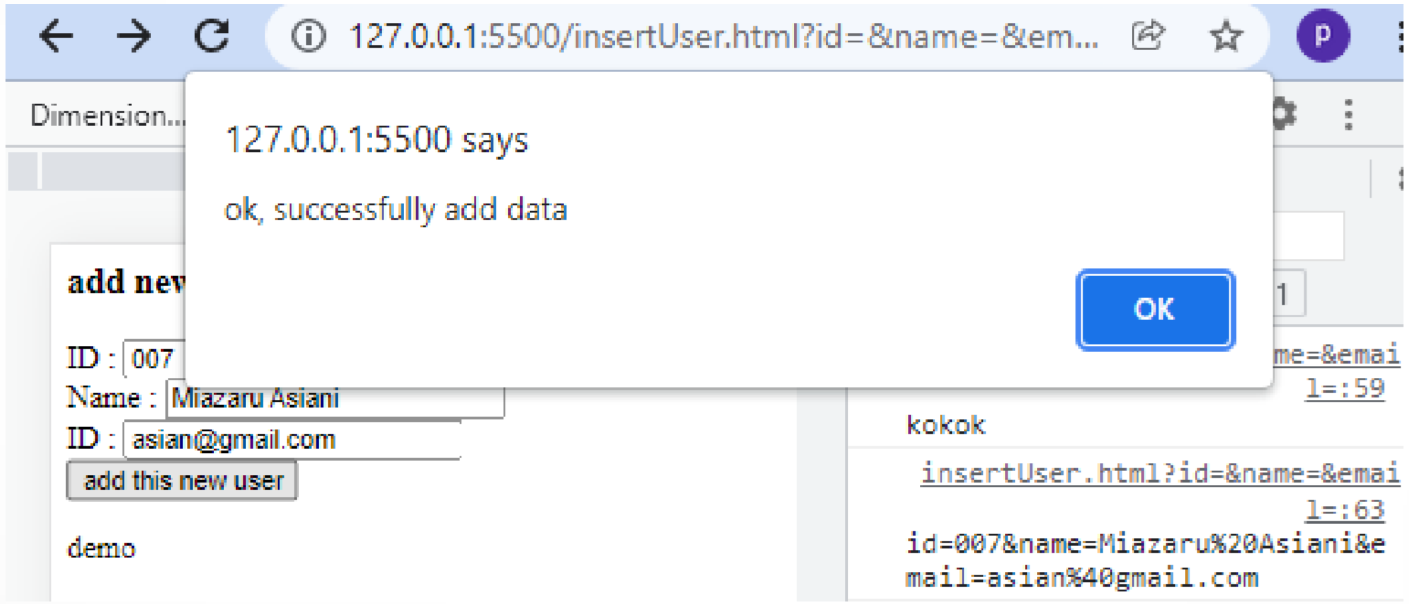
- we can’t invoke the post method using web browser url

Consume Your Web Service : using JavaScript in "insertUser.html" - code

```
<html>
<head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.1/jquery.js"></script>
</head>
<body>
    <h3>add new user</h3>
    <form id="addpatient">
        ID : <input type="text" name="id" placeholder="create id"> <br>
        Name : <input type="text" name="name" placeholder="write fullname"> <br>
        ID : <input type="text" name="email" placeholder="enter email"> <br>
        <input type="submit" id="submit" value="add this new user">

    </form>



    <p id="consume_webservice">demo</p>
</body>
```

```
<script>
    $("#addpatient").submit(function (event) {
        event.preventDefault();
        //use this technique for ajax data if not using RESTFul
        //NOTE: this will capture the name attribute (not id) in our form
        var formData = $(this).serialize();
        console.log(formData); // check using console to make sure all the form data values are correctly serialized

        $.ajax({
            type: "POST",
            url: "http://localhost/api/insertUser_db.php",
            data: formData,
            dataType: "json",

            success: function (data, status, xhr) {
                if (data.rowcount > 0) {
                    alert("ok, successfully add data");
                } else {
                    alert("fail to insert, " + data.errormessage);
                }
            },
            error: function (xhr, resp, text) {
                alert("error " + xhr + ", " + resp + ", " + text);
            },
        });
    });
</script>
```

Consume Your Web Service : using JavaScript in “insertUser.html” - output



127.0.0.1:5500 says

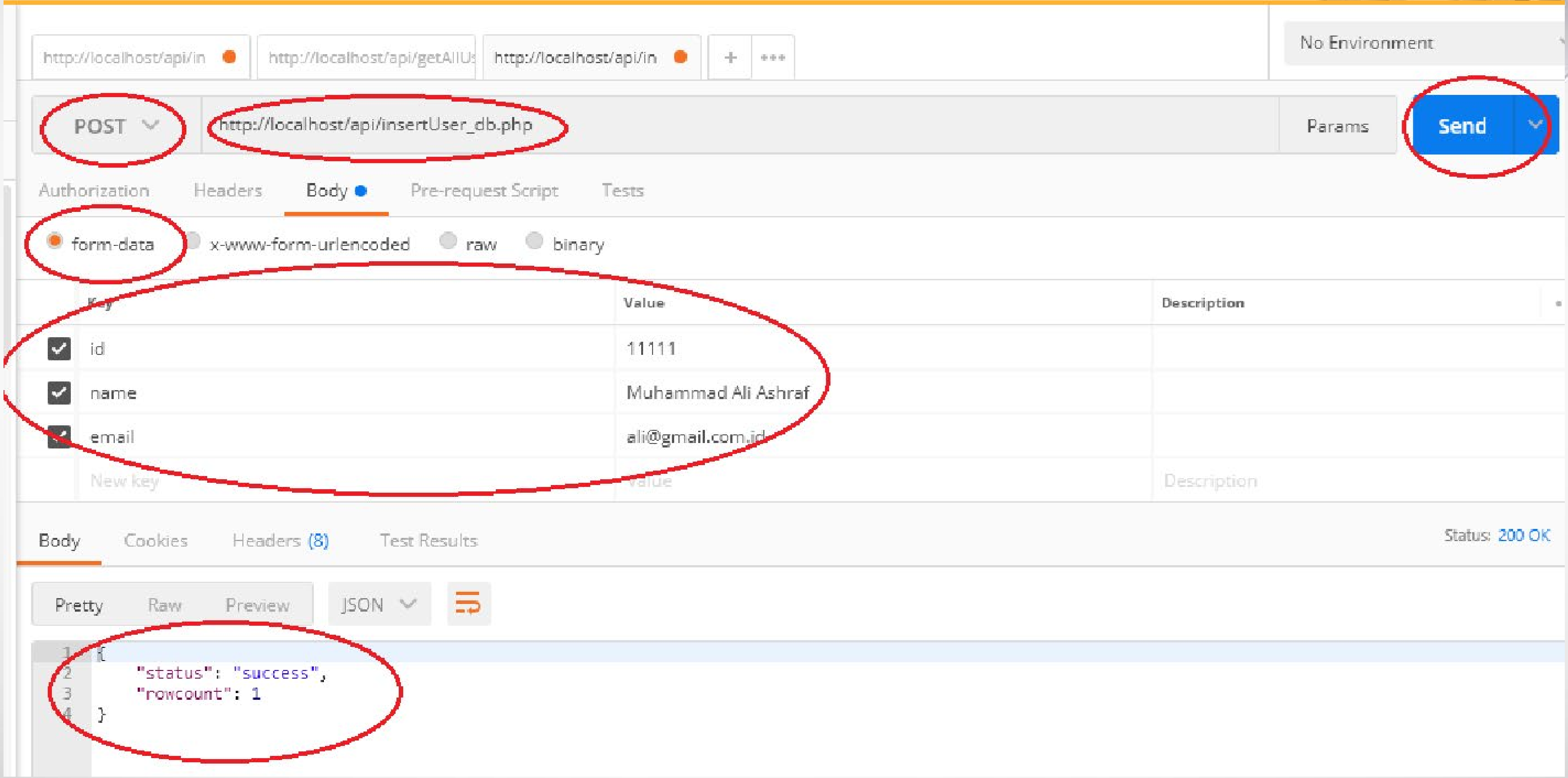
ok, successfully add data

ID : 007
Name : Miazaru Asiani
ID : asian@gmail.com
add this new user

demo

```
me=&emai
      l=:59
kokok
 insertUser.html?id=&name=&emai
      l=:63
id=007&name=Miazaru%20Asiani&e
mail=asian%40gmail.com
```

Consume Your Web Service : using Postman



http://localhost/api/in | http://localhost/api/getAllUs | http://localhost/api/in

No Environment

POST http://localhost/api/insertUser_db.php

Params Send

Authorization Headers Body Pre-request Script Tests

[✓ form-data] x-www-form-urlencoded raw binary

	Key	Value	Description
✓	id	11111	
✓	name	Muhammad Ali Ashraf	
✓	email	ali@gmail.com.id	
	New key	Value	Description

Body Cookies Headers (8) Test Results

Status: 200 OK

Pretty Raw Preview JSON

```
{
    "status": "success",
    "rowcount": 1
}
```

Your Tasks:

- Complete the implementation/code for
  - getSingleUser_db.php
  - updateUser_db.php
  - deleteUser_db.php

end of slides