How to create and document REST APIs

Presented by Ragnar Storstrøm
E-mail: rstorstrom@gmail.com
GitHub: https://github.com/lgs1971

If you wish to download the code…

- …the URL is: https://github.com/lgs1971/RESTAPISamples
- The code is fully documented*

*I have removed the comments in the presentation to make it shorter

What is REST?

- “REST is an acronym for **RE**presentational **S**tate **T**ransfer and an architectural style for **distributed hypermedia systems**. Roy Fielding first presented it in 2000 in his famous dissertation.”
  Source:
  https://restfulapi.net/
- A Web API (or Web Service) conforming to the REST architectural style is a *REST API*.
- HTTP error codes:
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Original use case

- My employer has a customer that have implemented Microsoft's Local Administrator Password Solution (LAPS). The customer wanted a form in User Application where users could request to get the local Administrator for their PC for 1-2 hours
- To implement this, we needed two things:
  - A REST API to interface with Microsoft AD
  - A form that could interface with the REST API

Prerequisite operations

- Install a Web server
  Windows:
  - Download and install Xampp
  Linux:
  - Install LAMP
  Note:
  - You only need to install Apache, PHP and MySQL server
  - You should configure PHP.INI to load **ldap** and **pdo_mysql**
- Install the following sample PHP database:
  https://www.mysqltutorial.org/how-to-load-sample-database-into-mysql-database-server.aspx

Recommended operations

- Use one of the following links to configure a virtual host:
  https://www.cloudways.com/blog/configure-virtual-host-on-windows-10-for-wordpress/
  https://towardsdatascience.com/how-to-host-multiple-website-with-apache-virtual-hosts-4423bd0aefbf
- Note:
  Remember to add the hostname to your DNS/hosts file!

Note: My example application use **demorestapi.localhost** as the virtual host

How to install Composer

- Open a new command line window
- Change to the root directory of your web server by running e.g.:
```
cd \Xampp\htdocs
```
- Run the following script to install and configure Composer:
```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');»
php -r "if (hash_file('sha384', 'composer-setup.php') === '906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac310e8cba5b96ac8b64115ac402c8cd292b8a03482574915d1a8') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;«
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Note: If you are installing on Windows, there is a Windows installer you can use

How to install the Slim Framework

- Run the following command to install Slim framework version 3:
  `composer require slim/slim:“3.*“`

How to install Swagger UI

- Use the Docker installation instructions found here: https://hub.docker.com/r/swaggerapi/swagger-editor/

Note:

- For this demo, I am using **zen_mclaren swaggerapi/swagger-editor**

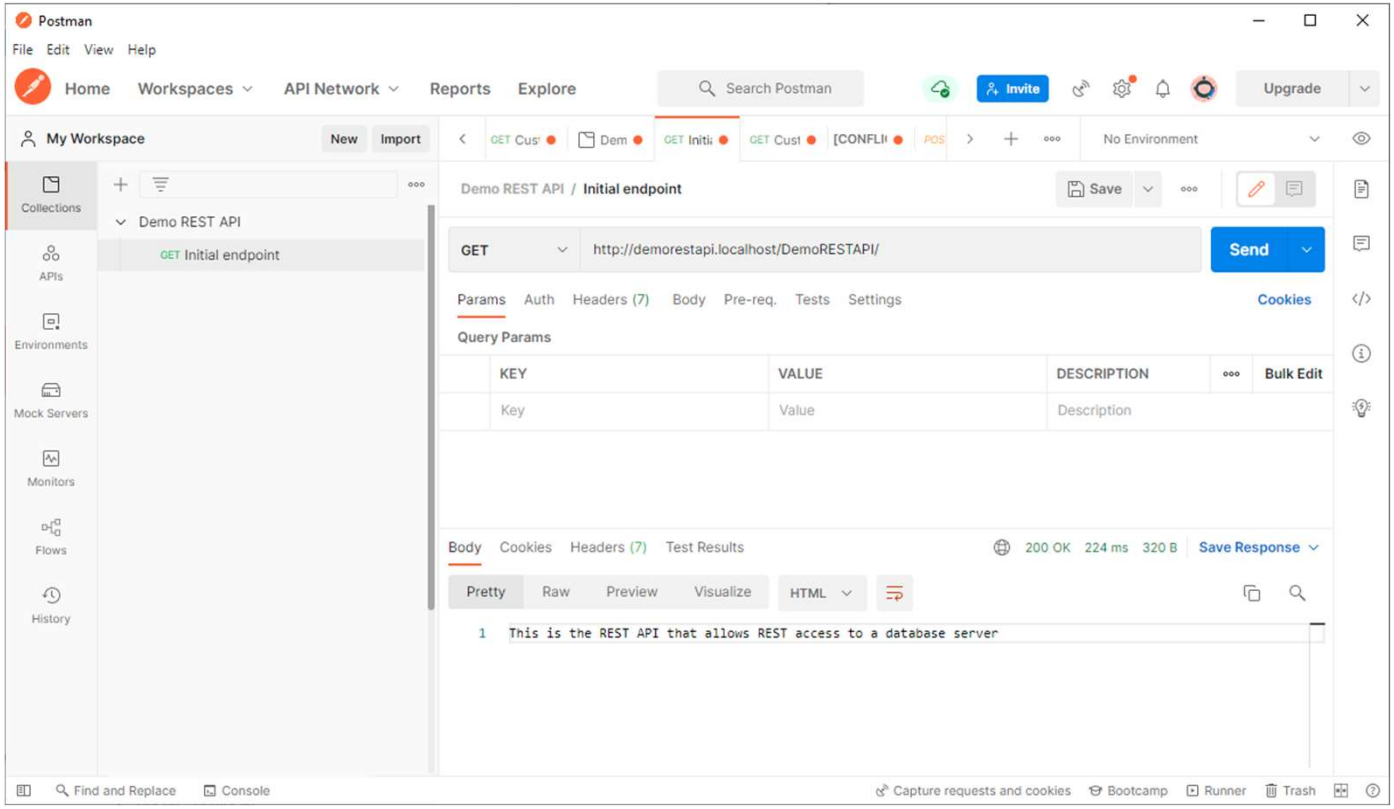
01 - How to create your first REST API

- Create a new directory for your new project, e.g.:
  `md DemoRESTAPI`
- Download the following files and place it in the directory:
  https://github.com/lgs1971/RESTAPISamples/Excercise_01.php
  https://github.com/lgs1971/RESTAPISamples/Excercise_01.yaml
- Open the file in your favorite code editor and review the code
- When you are done, save the file as **index.php**

01 - Code comments

- The following code line loads the Slim Framework:
  `require_once '../vendor/autoload.php';`
- These lines loads Slim Framework library files but change the names:
  `use \Psr\Http\Message\ServerRequestInterface as Request;`
  `use \Psr\Http\Message\ResponseInterface as Response;`
- This line creates a hashtable to hold the REST API configuration:
  `$config['displayErrorDetails'] = true;`
- This line creates a new Slim Framework application:
  `$app = new \Slim\App(['settings' => $config]);`
- This line configures a new REST endpoint:
  `$app->get('/', function (Request $request, Response $response)`
- This line starts the application:
  `$app->run();`

01 - How to create your first REST API (demo)



01 – Documenting the REST API in Swagger

- The following code lines provide the basic setup for documenting a REST API in Swagger:

```
openapi: 3.0.0
info:
  title: REST API for accessing the classicmodels database
  description: This REST API was created by Ragnar Storstrøm as a demo on how to write a REST API using the Slim framework
  version: 0.0.1

servers:
  - url: http://demorestapi.localhost/DemoRESTAPI
    description: REST API server in my internal lab
```

01 – Documenting the REST API in Swagger

- The following code lines document the initial REST endpoint in Swagger:

```
paths:
  /:
    get:
      summary: Returns information about which REST API you have accessed
      description: This is a test endpoint designed to prove you can access the demo REST API
    tags:
       - REST API
    responses:
       '200':
        description: A text describing the REST API is returned
        content:
          application/text:
            schema:
              type: string
              example:
                This is the REST API that allows REST access to a database server
```

02 – Connecting to your database

- Download the following file and place it in the directory:
  https://github.com/lgs1971/RESTAPISamples/Excercise_02.php
  https://github.com/lgs1971/RESTAPISamples/Excercise_02.yaml
- Open the file in your favorite code editor and review the code
- When you are done, save the file as **index.php** (overwriting the current one)

02 – Code comments

- The following code lines defines a Container object to hold the database connection:

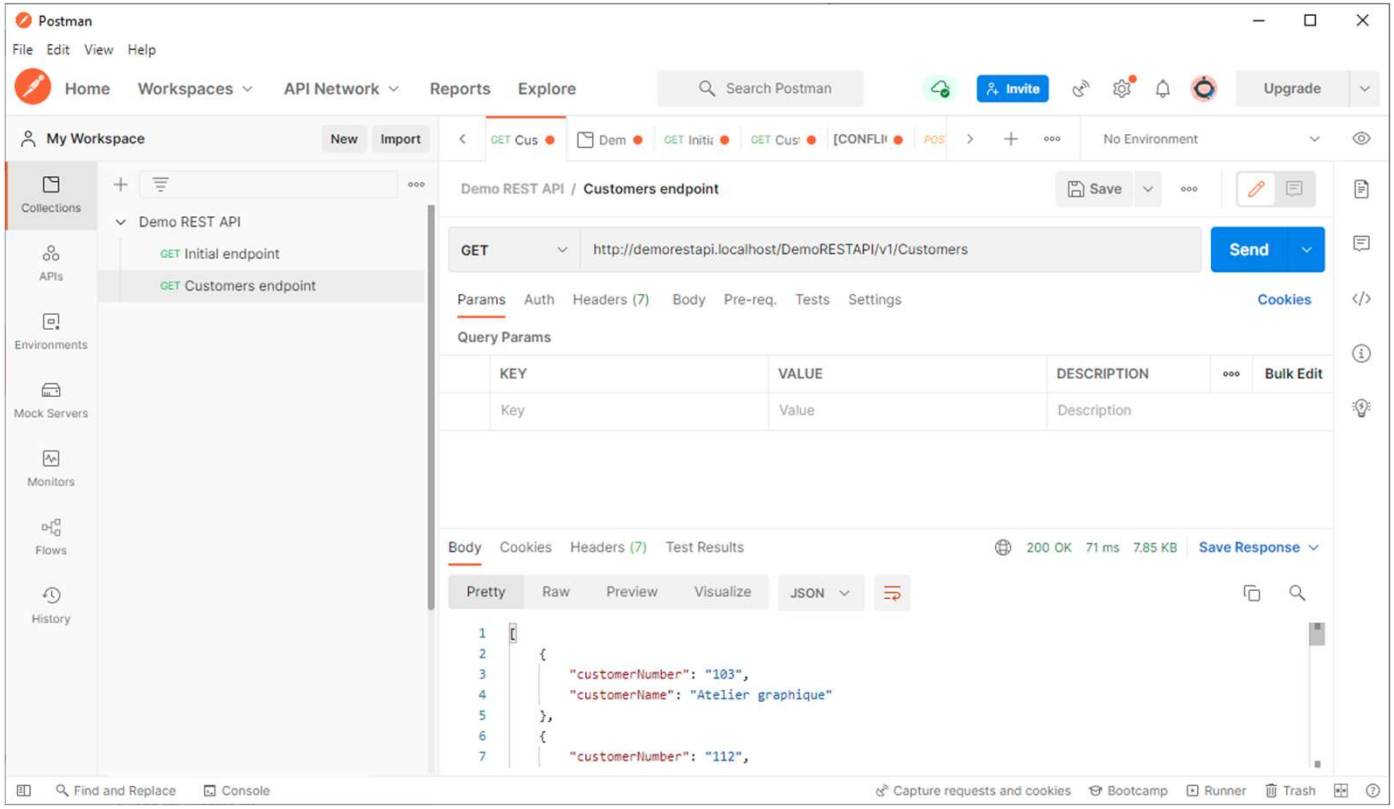
```
global $container;
$container = $app->getContainer();
$container['db'] = function ($c)
{
  $settings = $c['settings']['db’];
  $pdo      = new PDO('mysql:host=' . $settings['host'] . ';dbname=‘ .  $settings['dbname'],
$settings['user'], $settings['pass’]);
  $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
  return $pdo;
};
```

02 – Code comments

- The following code lines defines a new REST API calles v1/Customers:

```
$app->group('/v1', function ()
 {
   $this->get('/Customers', function (Request $Request, Response $Response, $Args)
   {
     $PDOObject    = $this->get('db’);
     $SQLStatement = $PDOObject->prepare("SELECT customerNumber, customerName FROM
customers");
     $QueryRes     = $SQLStatement->execute();
     if ($QueryRes === true)
     {
       $Result = $SQLStatement->fetchAll();
       return $Response->withJson($Result, 200);
     }
   });
 }
);
```

02 – Connecting to your database (demo)



02 – Documenting the REST API in Swagger

- The following code lines document the v1/Customers GET REST endpoint in Swagger:

```
/v1/Customers:
   get:
    summary: Returns all records in the customers table in the database
    description: This endpoint takes an object DN as a parameter and returns a database ID
    tags:
     - Customers
    responses:
     '200':
      description: The customer number and names from the customer database is returned
```

02 – Documenting the REST API in Swagger

- The following code lines document the v1/Customers GET REST endpoint in Swagger:

```
content:
   application/json:
    schema:
     type: array
     items:
      type: object
      properties:
       customerNumber:
        type: integer
       customerName:
        type: string
     example:
      customerNumber: 103
      customerName: Atelier graphique
```

03 – Looking up a record

- Download the following file and place it in the directory:
  https://github.com/lgs1971/RESTAPISamples/Excercise_03.php
  https://github.com/lgs1971/RESTAPISamples/Excercise_03.yaml
- Open the file in your favorite code editor and review the code
- When you are done, save the file as **index.php** (overwriting the current one)

03 – Code comments

- The following code lines defines a new REST API calles v1/Customer with a parameter for the customer number to look up:

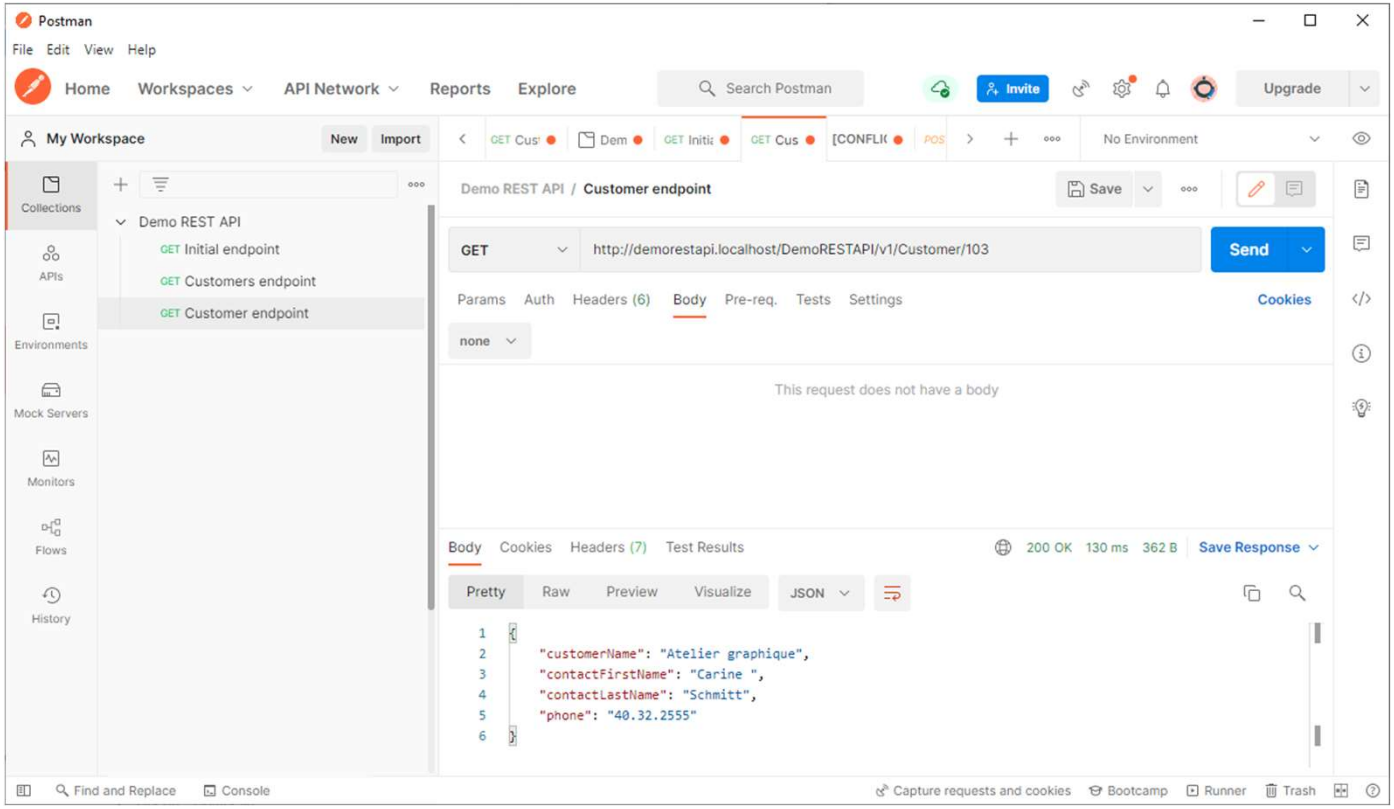
```
$this->group('/Customer', function()
    {
     $this->get('/{CustomerNumber}', function (Request $Request, Response $Response, $Args)
      {
       $CustomerNumber = html_entity_decode($Args['CustomerNumber']);
       $PDOObject      = $this->get('db');
       $SQLStatement   = $PDOObject->prepare("SELECT customerName, contactFirstName,
contactLastName, phone FROM customers WHERE customerNumber = :CustomerNumber");
       $QueryRes       = $SQLStatement->execute(['CustomerNumber' => $CustomerNumber]);
       if ($QueryRes === true)
       {
        $Customer = $SQLStatement->fetch();
```

03 – Code comments

- The following code lines defines a new REST API calles v1/Customer with a parameter for the customer number to look up:

```
            if ($Customer)
            {
              $Response = $Response->withJson($Customer, 200);
            } else
            {
              $Response = $Response->withJson("ERROR: No customer with number '" . $CustomerNumber
. "' could be found!", 204);
            }
          } else
          {
            $Response->write(json_encode("ERROR: Select returned error '" . $SQLStatement->errorCode()
. "'!"));
            $Response = $Response->withStatus(500);
          }
          return $Response;
        }
      );
    });
```

03 – Looking up a record (demo)



03 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/{CustomerNumber} GET REST endpoint in Swagger:

```
/v1/Customer/{CustomerNumber}:
   get:
     summary: Returns the customer name in the database for the supplied customer number
     description: This endpoint takes a customer number as a parameter and returns a customer
name
     tags:
      - Customer
     parameters:
      - in: path
        name: CustomerNumber
        schema:
          type: integer
        required: true
        description: Customer number to get the name of
```

03 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/{CustomerNumber} GET REST endpoint in Swagger:

```
responses:
    '200':
      description: The customer name for the supplied customer number is returned
      content:
        application/json:
          schema:
            type: string
            example:
              customerName: Atelier graphique
    '204':
      description: No customer was found in the database with this customer number
      content:
        application/json:
          schema:
            type: string
            example:
              No customer with number xxx could be found!
```

03 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/{CustomerNumber} GET REST endpoint in Swagger:

```
'500':
  description: A database error occurred
  content:
    application/json:
      schema:
        type: string
        example:
          ERROR: Select returned error xxx!
```

04 – Adding logging

- Install the slim-basic-auth module by running the following command:
  composer require monolog/monolog
- Download the following file and place it in the directory:
  https://github.com/lgs1971/RESTAPISamples/Excercise_04.php
- Open the file in your favorite code editor and review the code
- When you are done, save the file as **index.php** (overwriting the current one)

04 – Code comments

- These lines loads Monologger library files but change the names:

```
use \Monolog\Logger as MonologLogger;
use \Monolog\Handler\StreamHandler as MonologStreamHandler;
```

- The following code lines defines a Container object to hold the monologger:

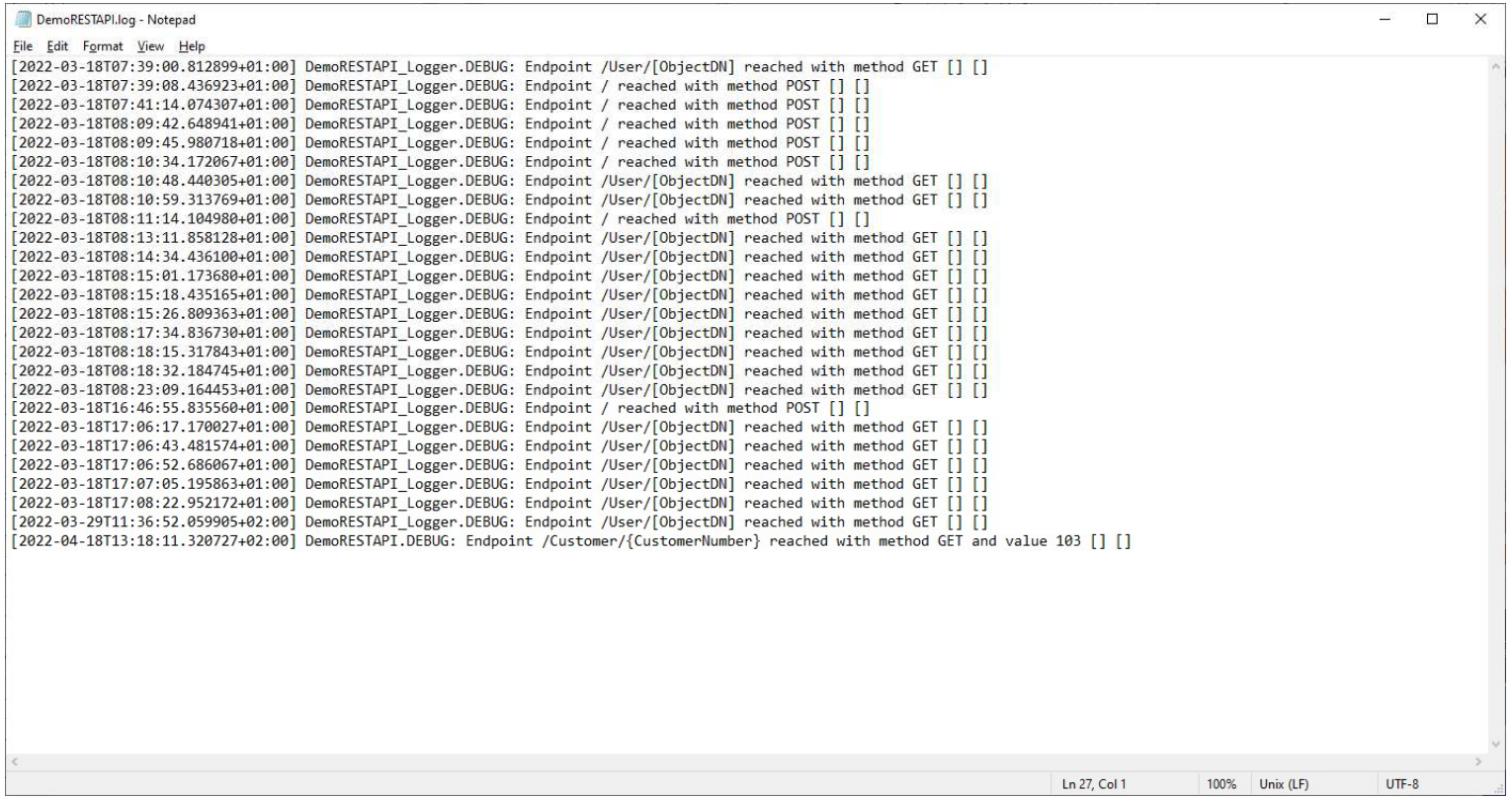
```
$container['logger'] = function($c)
{
 $Logger      = new MonologLogger('DemoRESTAPI’);
 $FileHandler = new MonologStreamHandler("../logs/DemoRESTAPI.log");
 $Logger->pushHandler($FileHandler);
 return $Logger;
};
```

04 – Code comments

- The following lines add comments to the log file when REST endpoints are reached:

```
$this->logger->debug("Endpoint /Customers reached with method GET");
$this->logger->debug("Endpoint /Customer/{CustomerNumber} reached with method GET and value $CustomerNumber");
```

04 – Adding logging (demo)



04 – Documenting the REST API in Swagger

- Since we only added logging, there is no new code to add to the Swagger file

05 – Adding a database record

- Download the following file and place it in the directory:
  https://github.com/lgs1971/RESTAPISamples/Excercise_05.php
  https://github.com/lgs1971/RESTAPISamples/Excercise_05.yaml
- Open the file in your favorite code editor and review the code
- When you are done, save the file as **index.php** (overwriting the current one)

05 – Code comments

- These lines define a new REST endpoint with method POST:

```
$this->post('/', function(Request $Request, Response $Response, $Args)
        {
          $this->logger->debug("Endpoint /Customer reached with method POST");
```

- The following code lines gets the data from the POST operation:

```
          $FormData = $Request->getParsedBody();
          $CustomerNumber       = filter_var($FormData['customerNumber'],
FILTER_SANITIZE_STRING);
          $CustomerName        = filter_var($FormData['customerName'],
FILTER_SANITIZE_STRING);
          …
```

05 – Code comments

- These lines checks if we have the required values:

```
if ((!is_null($CustomerNumber)) && (!is_null($CustomerName)) &&
    (!is_null($ContactLastName)) && (!is_null($ContactFirstName)) &&
    (!is_null($Phone)) && (!is_null($AddressLine1))&& (!is_null($City)) && (!is_null($Country)))
{
```

- These lines prepares the values for creating a new customer record:

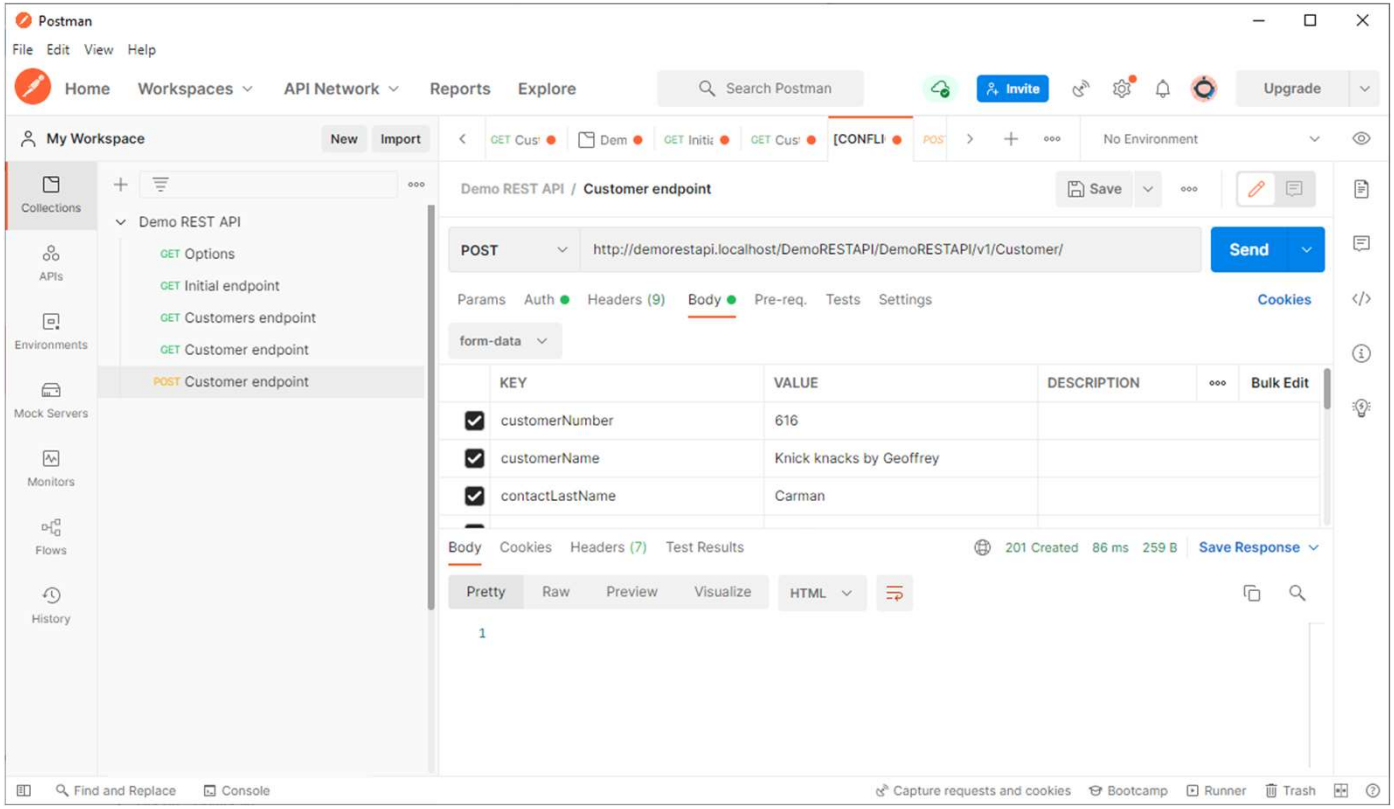
```
$Data = [
            'CustomerNumber' => $CustomerNumber,
            'CustomerName' => $CustomerName,
            …
```

05 – Code comments

- These lines store the values in the database:

```
$PDOObject      = $this->get('db’);
$SQLStatement = $PDOObject->prepare("INSERT INTO customers (customerNumber,
customerName, contactLastName, contactFirstName, phone, addressLine1, addressLine2, city,
state, postalCode, country, salesRepEmployeeNumber, creditLimit) VALUES (:CustomerNumber,
:CustomerName, :ContactLastName, :ContactFirstName, :Phone, :AddressLine1, :AddressLine2,
:City, :State, :PostalCode, :Country, :SalesRepEmployeeNumber, :CreditLimit)");
$Result              = $SQLStatement->execute($Data);
if ($Result === true)
{
  $Response = $Response->withStatus(201);
} else
{
  $Response = $Response->withJson("ERROR: Insert returned error '" . $SQLStatement-
>errorCode() . "'!", 400);
}
```

05 – Adding a database record (demo)



05 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/ POST REST endpoint in Swagger:

```
/v1/Customer/:
  post:
    summary: Creates the supplied customer in the database
    description: This endpoint takes customer information as a parameters
    tags:
      - Customer
    requestBody:
      required: true
      content:
        multipart/form-data:
          schema:
            type: object
            properties:
```

05 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/ POST REST endpoint in Swagger:

```
customerNumber:
  type: integer
customerName:
  type: string
contactLastName:
  type: string
contactFirstName:
  type: string
phone:
  type: string
addressLine1:
  type: string
addressLine2:
  type: string
```

05 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/ POST REST endpoint in Swagger:

```
city:
  type: string
state:
  type: string
postalCode:
  type: string
country:
  type: string
salesRepEmployeeNumber:
  type: integer
creditLimit:
  type: integer
```

05 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/ POST REST endpoint in Swagger:

```
responses:
   '201':
    description: Object created
   '400':
    description: Insert returned error xxx!
    content:
     application/json:
      schema:
       type: string
       example:
        ERROR: Insert returned error xxx!
```

05 – Documenting the REST API in Swagger

- The following code lines document the v1/Customer/ POST REST endpoint in Swagger:

```
'406':
  description: Mandatory parameter missing
  content:
   application/json:
    schema:
     type: string
     example:
      ERROR: Mandatory parameter missing
```

06 – Adding authentication to the POST

- Download the following file and place it in the directory:
  https://github.com/lgs1971/RESTAPISamples/Excercise_06.php
  https://github.com/lgs1971/RESTAPISamples/Excercise_06.yaml
- Open the file in your favorite code editor and review the code
- When you are done, save the file as **index.php** (overwriting the current one)

06 – Code comments

- These lines store the values in the database:

```
        # Log to file which REST endpoint was reached and which method was used
        $this->logger->debug("Endpoint /Customer reached with method POST");
        # Prepare to access the container
        global $container;
        # Get the environment variables
        $environment = $container['environment’];
        # Log the authentication information to file
        $this->logger->debug("User: " . $environment["PHP_AUTH_USER"] . " - Password:
********");
        # If authentication information was provided...
        if (($environment["PHP_AUTH_USER"] != "") and ($environment["PHP_AUTH_PW"] != ""))
        {
          # ...then open a connection to the LDAP directory service
          $LDAPConn = $this->get('ldap’);
```

06 – Code comments

- These lines store the values in the database:

```
# If the connection was successful...
if ($LDAPConn)
{
  # ...then prepare to bind to the LDAP directory service
  $LDAPBind = false;
  # Prepare to get the LDAP configuration information
  global $config;
  # If TLS should be used...
  if ($config['ldap']['usetls'] == true)
  {
    # ...then if StartTLS was successful...
    if (ldap_start_tls($LDAPConn))
    {
      # ...then bind to the LDAP directory service using the credentials in the Basic
authentication
      $LDAPBind = @ldap_bind($LDAPConn, $environment["PHP_AUTH_USER"],
$environment["PHP_AUTH_PW"]);
    } else
```

06 – Code comments

- These lines store the values in the database:

```
                {
                  # ...else convert the error information to JSON format and return a 401 - Unauthorized
error code
                  $LDAPErr = ldap_error($LDAPConn);
                  ldap_get_option($LDAPConn, LDAP_OPT_DIAGNOSTIC_MESSAGE, $LDAPOpt);
                   $Response = $Response->withJson("ERROR: Start TLS failed (" . $LDAPErr . ", " .
$LDAPOpt . ")!", 412);
                }
              } else
              {
                # ...else bind to the LDAP directory service using the credentials in the Basic
authentication
                $LDAPBind = @ldap_bind($LDAPConn, $environment["PHP_AUTH_USER"],
$environment["PHP_AUTH_PW"]);
              }
              # If the LDAP bind was successful...
             if ($LDAPBind)
             {
               # ...then get the form data from the POST request
               $FormData = $Request->getParsedBody();
```

06 – Code comments

- These lines store the values in the database:

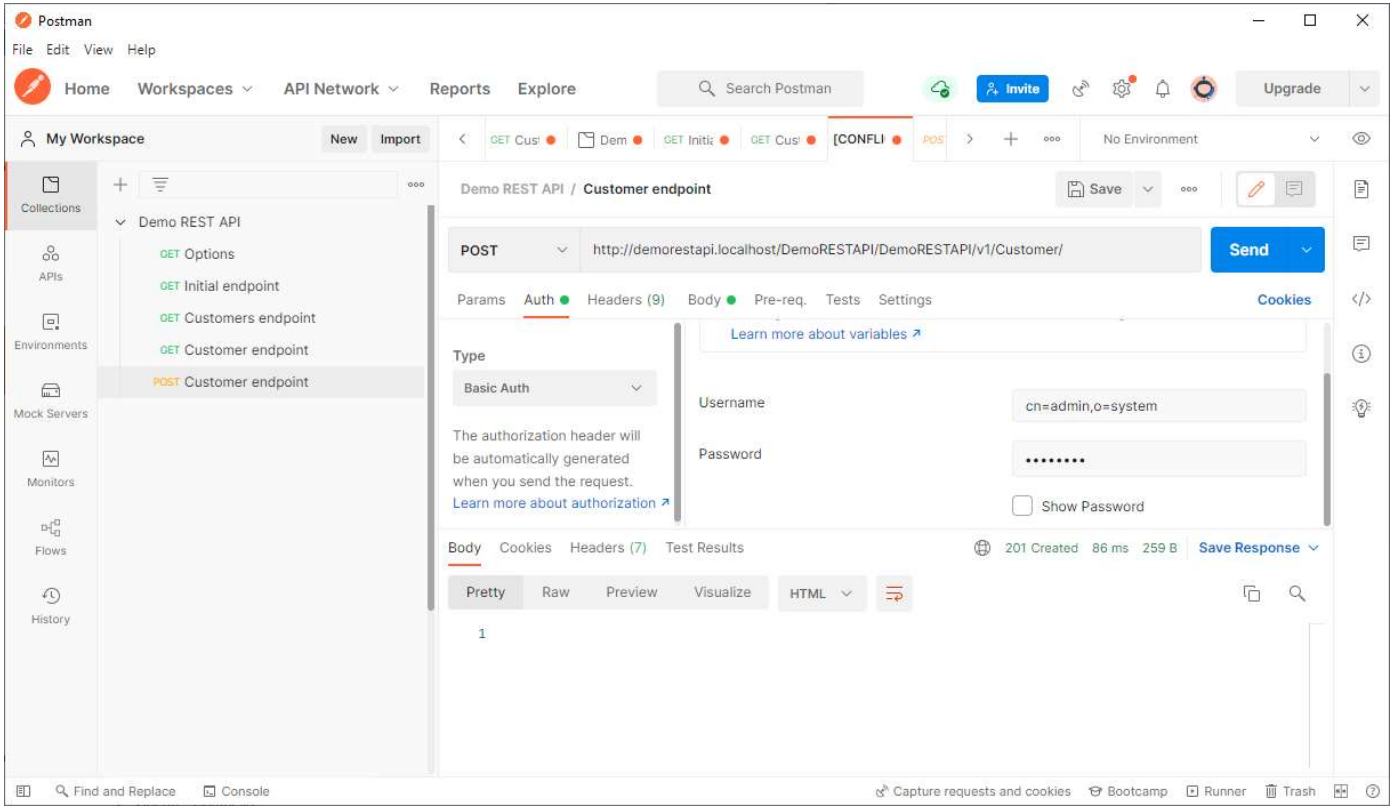
```
                } else
                {
                  # ...else convert the error information to JSON format and return a 401 - Unauthorized
error code
                  $LDAPErr   = ldap_error($LDAPConn);
                  ldap_get_option($LDAPConn, LDAP_OPT_DIAGNOSTIC_MESSAGE, $LDAPOpt);
                  $ErrorsMsg = "ERROR: LDAP bind failed (" . $LDAPErr . ", " . $LDAPOpt . ")!";
                  $Response  = $Response->withJson($ErrorsMsg, 401);
                  $this->logger->debug($ErrorsMsg);
                }
              } else
              {
                # ...else convert the error information to JSON format and return a 401 - Unauthorized
error code
                $LDAPErr = ldap_error($LDAPConn);
                ldap_get_option($LDAPConn, LDAP_OPT_DIAGNOSTIC_MESSAGE, $LDAPOpt);
                $ErrorsMsg = "ERROR: LDAP connect failed (" . $LDAPErr . ", " . $LDAPOpt . ")!";
                $Response  = $Response->withJson($ErrorsMsg, 401);
                $this->logger->debug($ErrorsMsg);
              }
```

06 – Code comments

- These lines store the values in the database:

```
        } else
        {
          # ...else convert the error information to JSON format and return a 401 - Unauthorized
error code
          $ErrorsMsg = "ERROR: Authorization is required to submit data!";
          $Response  = $Response->withJson($ErrorsMsg, 401);
          $this->logger->debug($ErrorsMsg);
        }
```

06 – Adding authentication to the POST (demo)



06 – Documenting the REST API in Swagger

- The following code lines document the Basic Authentication for the v1/Customer/ POST REST endpoint in Swagger:

```
servers:
  - url: http://demorestapi.localhost/DemoRESTAPI
    description: REST API server in my internal lab

components:
  securitySchemes:
    basicAuth:     # <-- arbitrary name for the security scheme
      type: http
      scheme: basic
```

06 – Documenting the REST API in Swagger

- The following code lines document the Basic Authentication for the v1/Customer/ POST REST endpoint in Swagger:

```
/v1/Customer/:
  post:
    summary: Creates the supplied customer in the database
    description: This endpoint takes customer information as a parameters
    security:
      - basicAuth: []
```

How is the Slim Framework licensed?

- The Slim Framework is using the MIT license: https://github.com/slimphp/Slim/blob/4.x/LICENSE.md
- In short, this means that you are allowed to sell products based on the Slim Framework as long as you keep the license information in the product

Does the Slim Framework have any security issues?

- The only CVE I could find was this one for v2.50 and below: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2171
- But the blog also mentions something that was fixed in version 3.0: https://www.slimframework.com/blog/

- As you can see, the developers seems to have done a good job of securing their product. And if you belive this list, even MicroFocus is using it (academy.vertica.com): https://trends.builtwith.com/websitelist/Slim-Framework