EEE 4482 Server Installation and Programming

Worksheet 7b – Writing REST API with PHP Framework

Objective: To create APIs more efficiently using a PHP framework (SLIM4)

Tools: Windows PC

Software: Oracle VM Virtual Box version 6.1.12
CentOS 7
Filezilla
Visual Studio Code
Chrome
Postman

Topics covered:

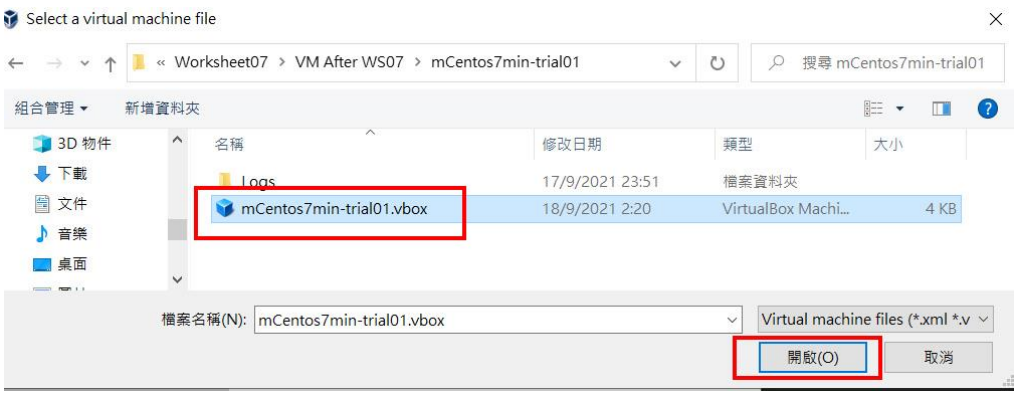
- Writing APIs with PHP framwork
- Testing APIs with suitable tool (i.e. Postman)

Component list:

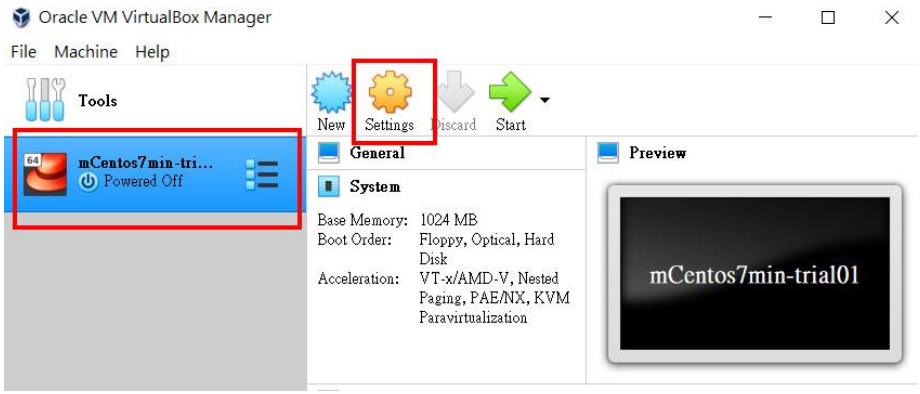
mCentos7min-trial01_AfterWS07_v2.zip - A virtual machine zip file with Apache, MariaDB and PHP7.4 installed

Testing the installation of PHP server

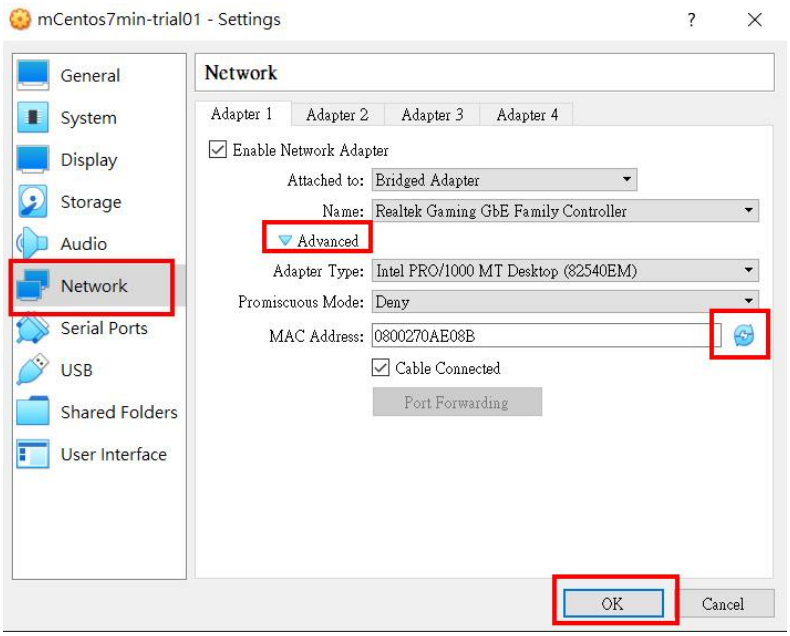
1. Download and unzip the provide virtual machine zip file.
2. Open Oracle VM Virtual Box. Open the unzipped virtual machine.



3. We need to change the MAC address of the network card inside the virtual machine. Highlight the added virtual machine. Press the “Settings” button.

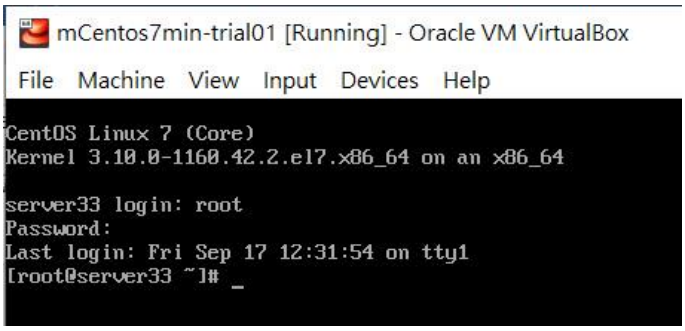


Select the “Network”. Click the “Advanced” word. In the line “MAC Address”, press the “Generates a new random MAC address” button as shown below. Finally, press “OK” button.



4. Press the green arrow button to start the virtual machine.

5. After booting the Linux, login the system using "**root**" and password "**netlab123**".



6. We need to get the IP address of the Virtual Machine. We enter the following command to get the allocated IP address.

   ip a | grep inet

   You should get similar response as shown below.

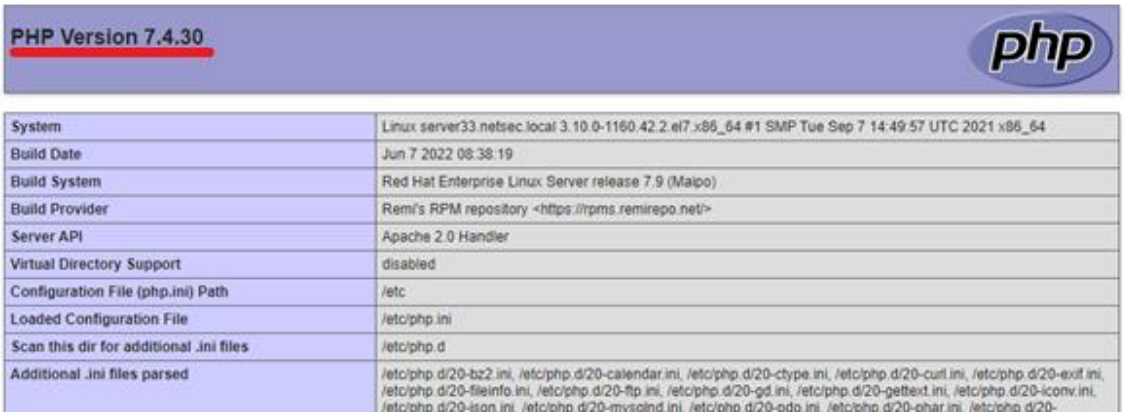
```
[root@server33 ~]#
[root@server33 ~]# ip a | grep inet
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 192.168.33.22/24 brd 192.168.33.255 scope global noprefixroute dynamic enp0s3
    inet6 fe80::8151:fa6f:e9b9:dcc4/64 scope link tentative noprefixroute dadfailed
    inet6 fe80::6a32:8396:f590:4825/64 scope link tentative noprefixroute dadfailed
    inet6 fe80::13f0:a5ff:7174:a8f8/64 scope link tentative noprefixroute dadfailed
[root@server33 ~]# _
```

In this example, the IP is 192.168.33.22. Please check your IP address and write it down.
IP address of the virtual machine is _____

7. Now, we can test if the installation of PHP is ok or not. Open a browser and enter the following link.

   http:// [your IP address]/info.php

   For example, in my case, I type http://192.168.33.22/info.php. I get the following page. That means the PHP server installation is properly installed.



PHP Version 7.4.30

System	Linux server33.netsec.local 3.10.0-1160.42.2.el7.x86_64 #1 SMP Tue Sep 7 14:49:57 UTC 2021 x86_64
Build Date	Jun 7 2022 08:38:19
Build System	Red Hat Enterprise Linux Server release 7.9 (Maipo)
Build Provider	Remi's RPM repository <https://rpms.remirepo.net/>
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gd.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-json.ini, /etc/php.d/20-mysqlnd.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-

Install PHP composer on CentOS7

In this workshop, more PHP packages and dependencies are required to develop our application. The Composer is a useful tool to help us by tracking and managing the dependencies of our project. It pulls in all the required PHP packages that our project depends on and manage them for us.

1. Update the local repository before download and install anything on your system. It may take some time to download for the first time.

```
yum – y update
```

2. Install the required software dependencies.

```
yum -y install php-cli php-zip wget unzip
```

3. Download the installer script in the current directory for installation of Composer

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
```

4. We need to verify the integrity of the installer script by checking whether the SHA-384 hash matches the installer signature.

```
HASH="$(wget -q -O - https://composer.github.io/installer.sig)"

php -r "if (hash_file('SHA384', 'composer-setup.php') === '$HASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
```

If the two signatures match, the output shows the message: **Installer verified**.

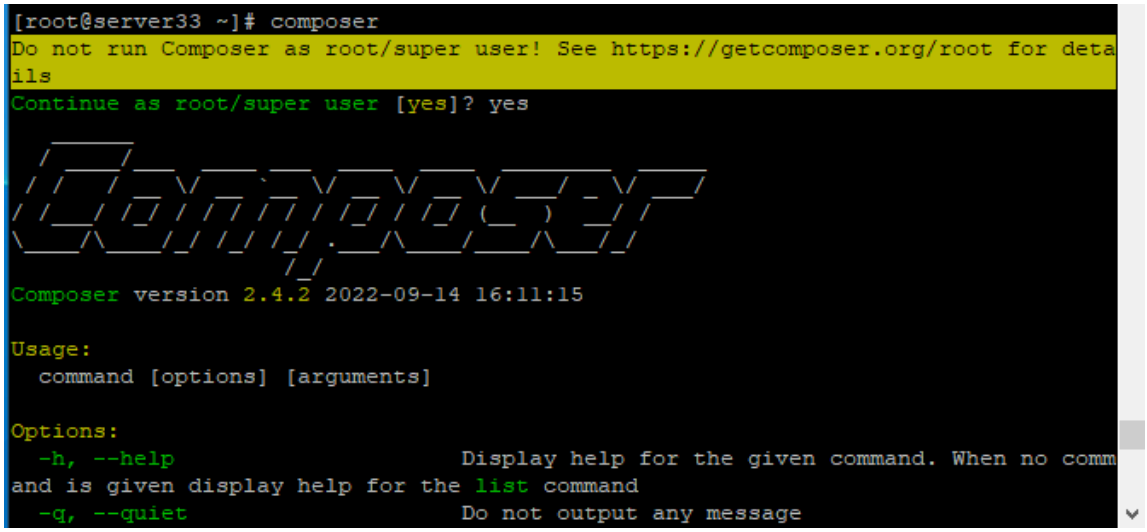
```
[root@server33 ~]# php -r "copy('https://getcomposer.org/installer', 'composer-s
etup.php');"
[root@server33 ~]# HASH="$(wget -q -O - https://composer.github.io/installer.sig
)"
[root@server33 ~]#
[root@server33 ~]# php -r "if (hash_file('SHA384', 'composer-setup.php') === '$H
ASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('co
mposer-setup.php'); } echo PHP_EOL;"
Installer verified
[root@server33 ~]#
[root@server33 ~]#
```

5. We can install the Composer by use the command:

```
php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

6. After the installation completed, we can input the following command to see if it can run correctly. Input "yes" if you are asked to continue as root/super user.

```
composer
```



Install Slim4 on PHP7.4

Slim is a micro PHP framework designed as a middleware to receive HTTP request. It routes the requests to the relevant controllers, and return the corresponding HTTP responses. Therefore, it is highly suitable for building of both micro web services and APIs that consume, repurpose, or publish data.

1. In your project's root directory ("***/var/www/html***"), create a folder, named "***api***". You can do this by running the following commands in terminal.

```
cd /var/www/html
mkdir api
cd api
```

2. In the "api" directory, you have to create two more folders named "***src***" and "***public***".

```
mkdir -p public src
```

3. Create a file named "***Db.php*** "in "***src***" directory. This file will store username and password for login to your database.

```
touch src/Db.php
```

4. Then, create an "***index.php"*** file in "***public***" directory. This file will contain the application's middleware and routes for setting the APIs.

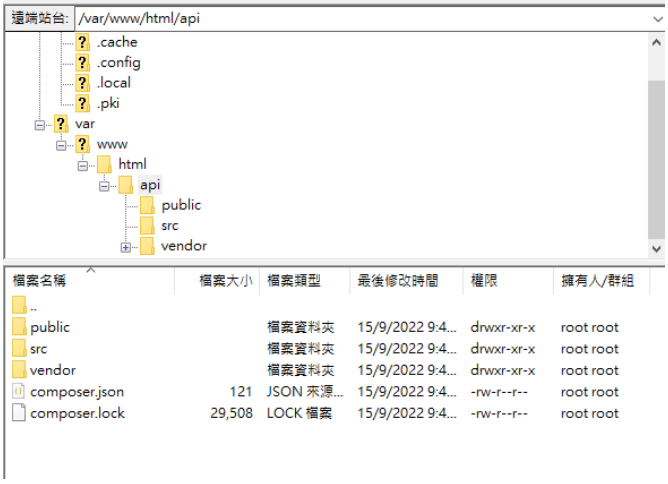
```
touch public/index.php
```

7. Finally, we have to install Slim4 and the required dependencies in your project directory ("***/var/www/html/api***"). Input "yes" if you are asked to continue as root/super user.

```
composer require --with-all-dependencies slim/slim:"4.*"
composer require --with-all-dependencies slim/psr7
composer require --with-all-dependencies selective/basepath
```

```
  - Installing nikic/fast-route (v1.3.0): Extracting archive
  - Installing slim/slim (4.10.0): Extracting archive
1 package suggestions were added by new dependencies, use `composer suggest` to s
ee details.
Generating autoload files
2 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found
[root@server33 api]#
```

8. When you login to server with FileZilla, you will see the following directory structure at "***/var/www/html/api***".



Enable .htaccess for running Slim4 in a subdirectory of Apache server

Let's consider the following situation:

- The main website is in the directory "***/var/www/html***" and is accessed at "***https:// [your IP] /***".
- The Slim 4 app is installed in the directory "***/var/www/html/api***", and to be accessed at "***https:// [your IP] /api***".
- The Slim 4 "***index.php***" file is stored in "***/var/www/html/api***".

In order to run Slim 4 in a subdirectory of the website, we need to add ".htaccess" files to the subdirectory for modifying the access control.

An ".htaccess" file is a directory-level configuration file for Apache HTTP server, which allows one to override the web server's system-wide settings without modifying the global configuration. For example, per-directory access control, password protection, URL redirection can be configured in the ".htaccess" file.

1. In default setting, the Apache HTTP server will ignored all ".htaccess" files. We need to explicitly enable it by editing the "***/etc/httpd/conf/httpd.conf***" file in Vim.

```
vi /etc/httpd/conf/httpd.conf
```

2. In "***httpd.conf***" file, enable ***mod_rewrite*** by adding the following script at the top line of the file.

```
LoadModule rewrite_module modules/mod_rewrite.so
...
...
...
```

3. In "***httpd.conf***" file, enable "***.htaccess"*** by adding the highlighted script in the file.

```
. . .
<Directory /var/www/html>
. . .
 #
 # AllowOverride controls what directives may be placed in .htaccess files.
 # It can be "All", "None", or any combination of the keywords:
 # Options FileInfo AuthConfig Limit
 #
 AllowOverride All
. . .
</Directory>
. . .
```

```
    Options Indexes FollowSymLinks

    #
    # AllowOverride controls what directives may be placed in .htaccess files.
    # It can be "All", "None", or any combination of the keywords:
    #   Options FileInfo AuthConfig Limit
    #
    AllowOverride All

    #
    # Controls who can get stuff from this server.
    #
    Require all granted
</Directory>
```

4. Use "***:w!***" command to overwrite the file and "***:q!***" command to leave in Vim.

5. Use "restorecon" command to restore the default context of the directory by reading the default rules set.

```
restorecon -r /var/www/html
```

6. Restart your Apache server

```
systemctl restart httpd
```

Adding .htaccess file to subdirectory

There are two "***.htaccess***" files that you are going to create.

1. Create the 1st "***.htaccess***" file in "***/var/www/html/api/public/***". The "***.htaccess***" file and "***index.php***" files should be in the same public-accessible directory. The "***.htaccess***" file should contain the following scripts. You can create this file by a code editor in Windows, and then upload it to the server by **FileZilla**.

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^ index.php [QSA,L]
```

2. Create the 2nd "***.htaccess***" file in "***/var/www/html/api/***" with the internal redirect rule to ensure that the "***public/***" directory does not appear in the URL. The "***.htaccess***" file should contain the following scripts.

```
RewriteEngine on
RewriteRule ^$ public/ [L]
RewriteRule (.*) public/$1 [L]
```

3. Restart Apache server to reload the configuration using following command.

```
systemctl restart httpd
```

Prepare a database for the application

In the last workshop, we learnt how to create a MySQL database and a data table by running PHP files. Before building our APIs, we need to prepare another database for storing the necessary data. The format of the database is shown below.

Database Name	elibrary				
Table Name	users				
Name	**Data Type**	**Property**	**Unique?**	**Null?**	**Key?**
user_id	Int(6)	Unsigned	Yes	No	Yes
username	Varchar(50)		Yes	No	No
password	Varchar(50)		No	No	No
is_admin	Int(6)	DEFAULT 0	No	No	No

Database Name	elibrary				
Table Name	books				
Name	**Data Type**	**Property**	**Unique?**	**Null?**	**Key?**
book_id	Int(6)	Unsigned	Yes	No	Yes
title	Varchar(50)		Yes	No	No
authors	Varchar(50)		No	No	No
publishers	Varchar(50)		No	No	No
date	Varchar(50)		No	No	No
isbn	Varchar(50)		Yes	No	No
status	int(6)	DEFAULT 0	No	No	No
borrowed_by	int(6)	DEFAULT -1	No	Yes	No
last_updated	timestamp	on update CURRENT_TIMESTAMP	No	No	No

1. Create a php file named "***CreateDb.php***" for creation of the required database.

```php
<?php
$servername = "localhost";
$username = "root";
$password = "netlab123";
$dbname = "elibrary";
try {
    $conn = new PDO("mysql:host=$servername", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "CREATE DATABASE $dbname";
    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Database has been created successfully<br>";
    // Close connection
    $conn = null;
} catch(PDOException $e) {
    echo $sql . "<br>" . $e->getMessage();
    // Close connection
    $conn = null;
}
$conn = null;
?>
```

2. Create a php file named "***CreateTableBooks.php***" for storing books' records in database.

```php
<?php

$servername = "localhost";
$username = "root";
$password = "netlab123";
$dbname = "elibrary";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // sql to create table
    $sql = "CREATE TABLE books (
        book_id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        title VARCHAR(50) NOT NULL UNIQUE,
        authors VARCHAR(50) NOT NULL,
        publishers VARCHAR(50) NOT NULL,
        date VARCHAR(50) NOT NULL,
        isbn VARCHAR(50) NOT NULL UNIQUE,
        status INT(6) NOT NULL DEFAULT 0,
        borrowed_by  INT(6) DEFAULT -1,
        last_updated  TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
    )";
    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Table books has been created successfully";

    // Close connection
    $conn = null;
} catch(PDOException $e) {
    echo $sql . "<br>" . $e->getMessage();
    // Close connection
    $conn = null;
}

$conn = null;
?>
```

3. Create a php file named "***CreateTableUsers.php***" for storing usernames and passwords.

```php
<?php
    $servername = "localhost";
    $username = "root";
    $password = "netlab123";
    $dbname = "elibrary";
    try {
        $conn = new PDO("mysql:host=$servername;dbname=$dbname",
        $username, $password);
        // set the PDO error mode to exception
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        // sql to create table
        $sql = "CREATE TABLE users (
            user_id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
            username VARCHAR(50) NOT NULL UNIQUE,
            password VARCHAR(50) NOT NULL,
            is_admin INT(6) NOT NULL DEFAULT 0
        )";
        // use exec() because no results are returned
        $conn->exec($sql);
        echo "Table users has been created successfully";
        // Close connection
        $conn = null;
    } catch(PDOException $e) {
        echo $sql . "<br>" . $e->getMessage();
        // Close connection
        $conn = null;
    }
    $conn = null;
?>
```

4. Upload the file "***CreateDb.php***" to the folder "***/var/www/html/***" in the virtual machine.

   Open a browser and input the link as follows.
   http:// [your IP] / CreateDb.php
   e.g. http://192.168.0.101/ CreateDb.php

5. Repeat step 4 for "***CreateTableBooks.php***" file and "***CreateTableUsers.php***" file.
   - http:// [your IP] / ***CreateTableBooks***.php
   - http:// [your IP] / ***CreateTableUsers***.php

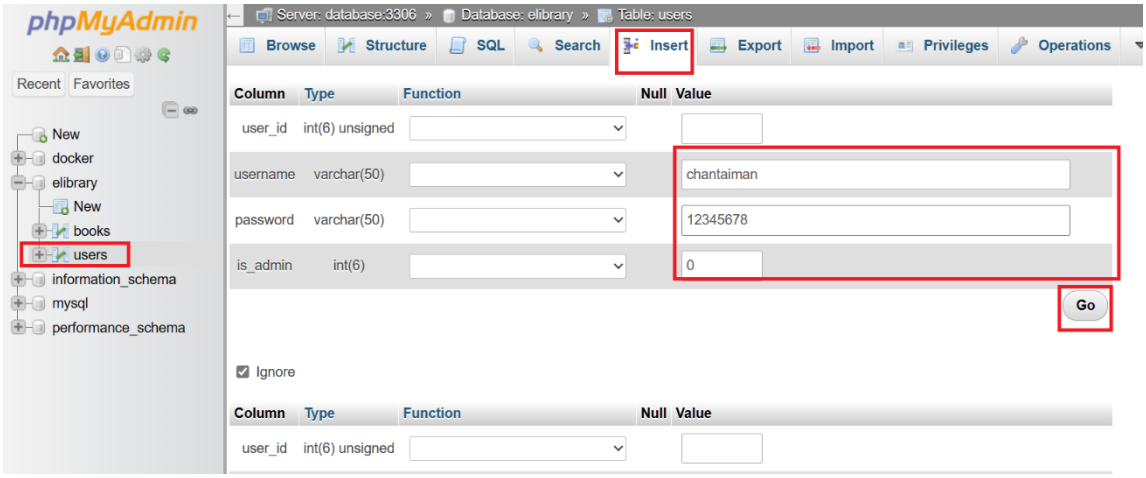
6. Login to the ***phpMyAdmin*** admin panel to check if the database is created correctly.
   Open a browser and input the link "***http:// [your IP] /phpmyadmin***/".
   Username: **root**
   Password: **netlab123**

What is phpMyAdmin?

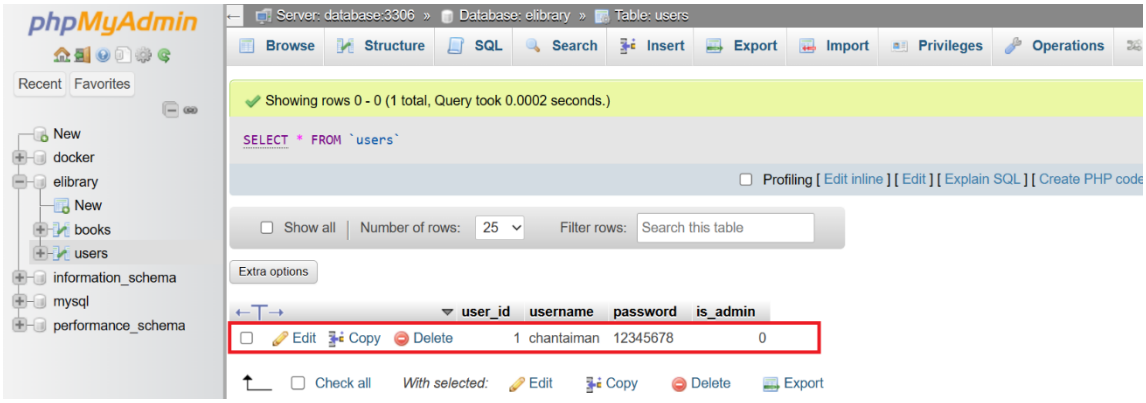
phpMyAdmin is a web-based GUI tool written in PHP that allows users to manage MySQL or MariaDB databases. It offers a user-friendly interface accessible through a web browser, enabling tasks like creating and managing databases, executing SQL queries, importing and exporting data, and managing users and permissions. With its intuitive features, phpMyAdmin simplifies database administration for developers and administrators, providing a convenient and visual way to interact with MySQL or MariaDB databases without the need for extensive knowledge of command-line tools or SQL syntax.

7. In ***phpMyAdmin***, you can manually insert a new record with user inputs.

elibrary -> users -> Insert:

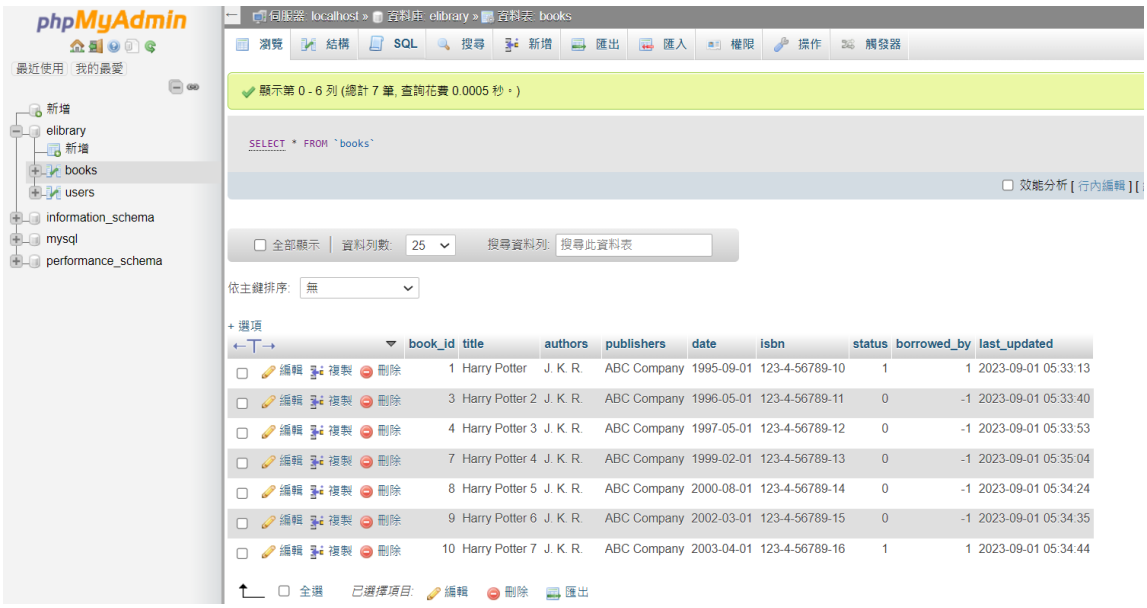


8. Go back to the home page of the "users" table, you will see a new record displayed on the table.



9. Repeat step 8 to create books' records in "books" table as shown below.

book_id	title	authors	publishers	date	isbn	status	borrowed_by
1	Harry Potter	J. K. R.	ABC Company	1995-09-01	123-4-56789-10	1	1
2	Harry Potter 2	J. K. R.	ABC Company	1996-05-01	123-4-56789-11	0	-1
3	Harry Potter 3	J. K. R.	ABC Company	1997-05-01	123-4-56789-12	0	-1
4	Harry Potter 4	J. K. R.	ABC Company	1999-02-01	123-4-56789-13	0	-1
5	Harry Potter 5	J. K. R.	ABC Company	2000-08-01	123-4-56789-14	0	-1
6	Harry Potter 6	J. K. R.	ABC Company	2002-03-01	123-4-56789-15	1	1
7	Harry Potter 7	J. K. R.	ABC Company	2003-04-01	123-4-56789-16	0	-1



10. You can delete the php files "***CreateDb.php***", "***CreateTableBooks.php***" and "***CreateTableUsers.php***" file by **FileZilla** after the database is successfully created.

Create the routes for HTTP requests

Let's start to add some codes into the directory "***/var/www/html/api/***" in server for routing the HTTP request for your APIs.

1. Create a file "***Db.php***" and write the following codes using your code editor in Windows.

```php
<?php
namespace App;
use \PDO;

class Db
{
  private $host = 'localhost';
  private $user = 'root';
  private $pass = 'netlab123';
  private $dbname = 'elibrary';

  public function connect()
  {
    $conn_str = "mysql:host=$this->host;dbname=$this->dbname";
    $conn = new PDO($conn_str, $this->user, $this->pass);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    return $conn;
  }
}
?>
```

2. Upload the file to the directory "***/var/www/html/api/src***" by **FileZilla**.

3. Create a new file named "***index.php***" and add the following template codes to the file in your code editor.

```php
<?php

require_once __DIR__ . '/../vendor/autoload.php';

$app = Slim\Factory\AppFactory::create();

//handle json format
$app->addBodyParsingMiddleware();

// Add Slim routing middleware
$app->addRoutingMiddleware();

// Set the base path to run the app in a subdirectory.
// This path is used in urlFor().
$app->add(new Selective\BasePath\BasePathMiddleware($app));

$app->addErrorMiddleware(true, true, true);

// Define app routes here




// Run app
$app->run();
?>
```

4. Add a default route to test if the routing function is working properly. Your code of routes should be put before the "***$app->run();***".

```
16 $app->addErrorMiddleware(true, true, true);
17 // Define app routes here
18
19 $app->get('/', function ($request, $response) {
20     $response->getBody()->write('Hello, World!');
21     return $response;
22 })->setName('root'); //<<<set root
23
24 // Run app
25 $app->run();
26 ?>
27
```

5. Upload the "***index.php"*** file to the server's directory "***/var/www/html/api/public/"*** to overwrite the existing ***"index.php"*** file via FileZilla. Open a browser and input the url as follows.

   http:// [your IP] / api/
   E.g. http://192.168.0.101/ api/

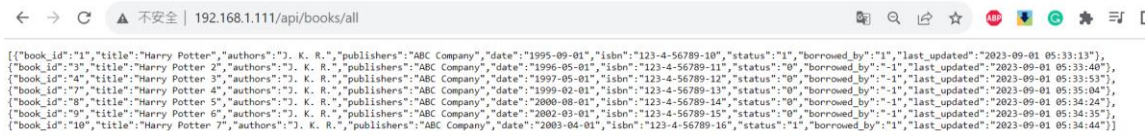
   You should get the following response from the server.

6. Add the following code below the default route in step 4 to handle another HTTP Get request. This API allows user to read all entries in the data table "books".

```
25 use App\Db;
26 require_once __DIR__ . '/../src/Db.php';
27 $app->get('/books/all', function ($request, $response) {
28     $sql = "SELECT * FROM books";
29
30     try {
31         $db = new Db();
32         $conn = $db->connect();
33
34         $stmt = $conn->query($sql);
35         $data = $stmt->fetchAll(PDO::FETCH_ASSOC);
36
37         // Close connection
38         $conn = null;
39         $db = null;
40
41         $response->getBody()->write(json_encode($data));
42         return $response
43             ->withHeader('content-type', 'application/json')
44             ->withStatus(200);
45     } catch (PDOException $e) {
46         $error = array(
47         "message" => $e->getMessage()
48         );
49
50         $response->getBody()->write(json_encode($error));
51         return $response
52             ->withHeader('content-type', 'application/json')
53             ->withStatus(500);
54     }
55 });
```

In line 25 – 26: The "*Db.php*" is imported to the code for the class **Db** which embedded with credentials information of the database.

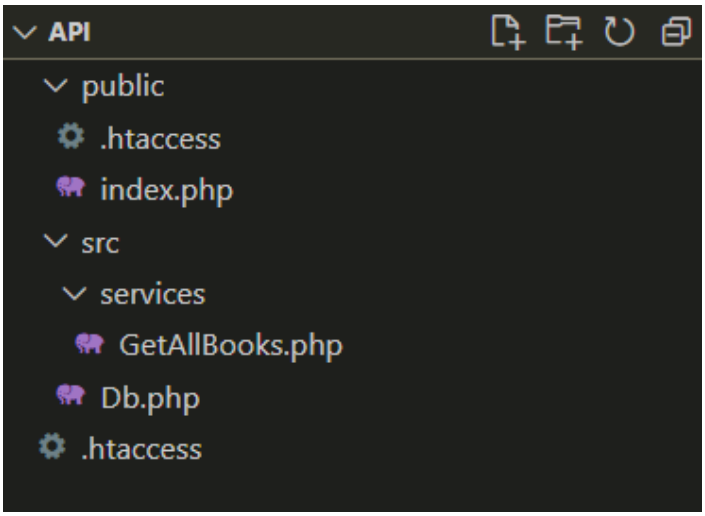
7. Open a browser and input "**http:// [your IP] /api/books/all**" to test the API.



If your API server is running correctly, you can read all books' records in json format.

8. Before adding more services into the server, we would like to rearrange the structure of codes in the "*index.php*". If we put all other services into a single PHP file, it will be easily messed up and significantly increase the difficulty of maintenance later.

   Let's put the details of web services into separate PHP files and only define the corresponding routes in the "*index.php*" with the following directory structure.



The details of API is put into a new file named "*GetAllBooks.php*" at "/var/www/html/api/src/services".

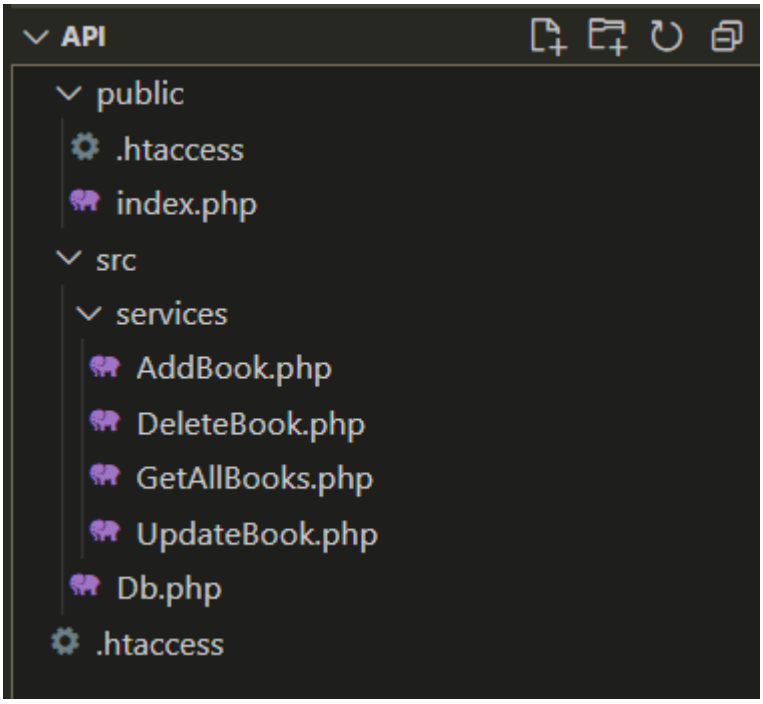
```php
<?php
namespace App\Services;  // to be recognized by server

use \PDO;
use App\Db;
require_once __DIR__ . '/../Db.php';

function GetAllBooks($request, $response) {
    $sql = "SELECT * FROM books";
    try {
        $db = new Db();
        $conn = $db->connect();

        $stmt = $conn->query($sql);
        $data = $stmt->fetchAll(PDO::FETCH_ASSOC);

        // Close connection
        $conn = null;
        $db = null;

        $response->getBody()->write(json_encode($data));
        return $response
            ->withHeader('content-type', 'application/json')
            ->withStatus(200);
    } catch (PDOException $e) {
        $error = array(
        "message" => $e->getMessage()
        );

        $response->getBody()->write(json_encode($error));
        return $response
            ->withHeader('content-type', 'application/json')
            ->withStatus(500);
    }
}
?>
```

Then, we have to update the route setting in "*index.php*" accordingly as shown below.

```
18
19  // Define app routes here
20  $app->get('/', function ($request, $response) {
21      $response->getBody()->write('Hello, World!');
22      return $response;
23  })->setName('root'); //<<<set root
24
25  //require all php files in /../src/services
26  foreach (glob(__DIR__ . '/../src/services/*.php') as $filename) {
27      // var_dump($filename); // for debug only
28      require_once($filename);
29  }
30  $app->get('/books/all', 'App\Services\GetAllBooks');
31
32
```

```
//require all php files in /../src/services
foreach (glob(__DIR__ . '/../src/services/*.php') as $filename) {
    // var_dump($filename); // for debug only
    require_once($filename);
}
$app->get('/books/all', 'App\Services\GetAllBooks');
```

9. We need to add other APIs to the server for **Add**, **Update** and **Delete** services of the books in the e-Library.



No.	Route	Filename	HTTP Method
1	"/books/all"	GetAllBook.php	GET
2	"/books/add"	AddBook.php	POST
3	"/books/update/{book_id}"	UpdateBook.php	PUT
4	"/books/delete/{book_id}"	DeleteBook.php	DELETE

For the **Add** service, you can put the following code in the "*AddBook.php*" file. Then add the "/books/add" route in the "*index.php*" for handling the HTTP Post request that allow user to insert a book's record to the database.

In "*AddBook.php*":

```php
<?php

namespace App\Services;

use \PDO;
use App\Db;
require_once __DIR__ . '/../Db.php';

function AddBook($request, $response, $args) {
    $data = $request->getParsedBody();
    $title = $data["title"];
    $authors = $data["authors"];
    $publishers = $data["publishers"];
    $date = $data["date"];
    $isbn = $data["isbn"];

    try {
            $db = new Db();
            $conn = $db->connect();

            // begin the transaction
            $conn->beginTransaction();

            // our SQL statements
            $sql = "INSERT INTO books (title, authors, publishers, date, isbn)
                    VALUES ('$title', '$authors', '$publishers', '$date', '$isbn')";
            $conn->exec($sql);

            // commit the transaction
            $result = $conn->commit();

            // Close connection
            $conn = null;
            $db = null;

            $response->getBody()->write(json_encode($result));
            return $response
                    ->withHeader('content-type', 'application/json')
                    ->withStatus(200);
    } catch (PDOException $e) {
            $error = array(
                    "message" => $e->getMessage()
            );

            $response->getBody()->write(json_encode($error));
            return $response
                    ->withHeader('content-type', 'application/json')
                    ->withStatus(500);
    }
}
```

For the **Update** service, you can put the following code in "*UpdateBook.php*" file. Then add the "/books/update" route in the "*index.php*" for handling the HTTP Put request that allow user to update a particular book's record in the database.

In "*UpdateBook.php*":

```php
<?php
namespace App\Services;

use \PDO;
use App\Db;
require_once __DIR__ . '/../Db.php';

function UpdateBook($request, $response, $args){
    $book_id = $request->getAttribute('book_id');
    $data = $request->getParsedBody();
    $title = $data["title"];
    $authors = $data["authors"];
    $publishers = $data["publishers"];
    $date = $data["date"];
    $isbn = $data["isbn"];
    try {
        $db = new Db();
        $conn = $db->connect();

        // begin the transaction
        $conn->beginTransaction();

        // our SQL statements
        $sql = "UPDATE books SET
                title = '$title',
                authors = '$authors',
                publishers = '$publishers',
                date = '$date',
                isbn = '$isbn'
                WHERE book_id = '$book_id'";

        $conn->exec($sql);

        // echo($sql); //for debug only

        // commit the transaction
        $result = $conn->commit();

        // Close connection
        $conn = null;
        $db = null;

        $response->getBody()->write(json_encode($result));
        return $response
            ->withHeader('content-type', 'application/json')
            ->withStatus(200);
    } catch (PDOException $e) {
        $error = array(
            "message" => $e->getMessage()
        );
        $response->getBody()->write(json_encode($error));
        return $response
            ->withHeader('content-type', 'application/json')
            ->withStatus(500);
    }
}
?>
```

For the **Delete** service, you can put the following code into "*DeleteBook.php*" file. Then add the "/books/delete" route in the "*index.php*" for handling the HTTP Delete request that allow user to delete a particular book's record in the database.

In "*DeleteBook.php*":

```php
<?php
namespace App\Services;
use \PDO;
use App\Db;
require_once __DIR__ . '/../Db.php';

function DeleteBook($request, $response, $args) {
    $book_id = $args["book_id"];
    try {
        $db = new Db();
        $conn = $db->connect();

        $sql = "DELETE FROM books WHERE book_id = $book_id";

        $stmt = $conn->prepare($sql);
        $result = $stmt->execute();

        $db = null;
        $response->getBody()->write(json_encode($result));
        return $response
            ->withHeader('content-type', 'application/json')
            ->withStatus(200);
    } catch (PDOException $e) {
        $error = array(
        "message" => $e->getMessage()
        );
        $response->getBody()->write(json_encode($error));
        return $response
            ->withHeader('content-type', 'application/json')
            ->withStatus(500);
    }
}
?>
```

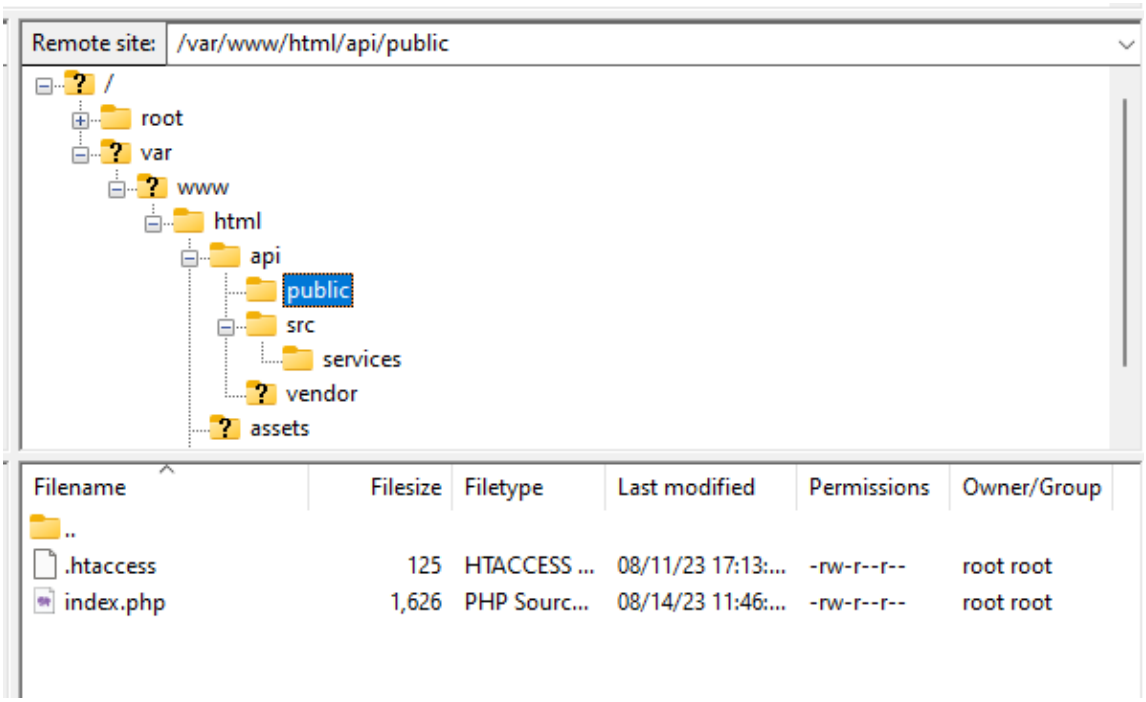
In "*index.php*":

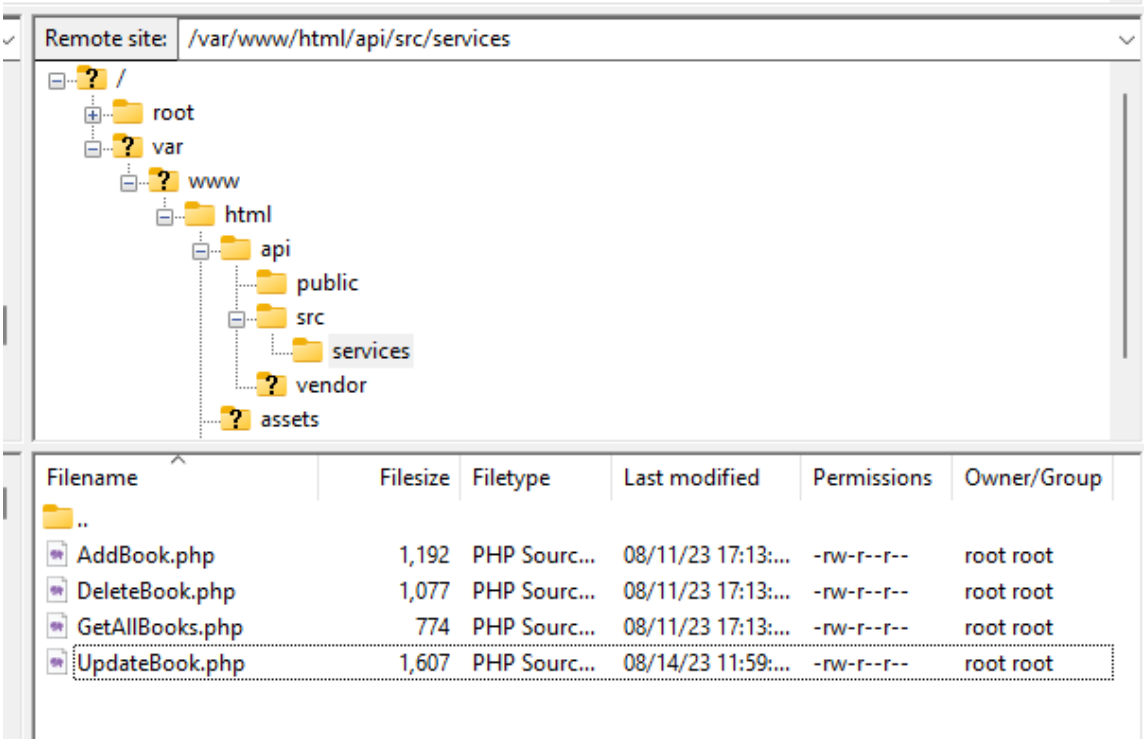
```php
<?php

require_once __DIR__ . '/../vendor/autoload.php';

$app = Slim\Factory\AppFactory::create();

//handle json format
$app->addBodyParsingMiddleware();

// Add Slim routing middleware
$app->addRoutingMiddleware();

// Set the base path to run the app in a subdirectory.
// This path is used in urlFor().
$app->add(new Selective\BasePath\BasePathMiddleware($app));

$app->addErrorMiddleware(true, true, true);

// Define app routes here
$app->get('/', function ($request, $response) {
    $response->getBody()->write('Hello, World!');
    return $response;
})->setName('root'); //<<<set root

//require all php files in /../src/services
foreach (glob(__DIR__ . '/../src/services/*.php') as $filename) {
    // var_dump($filename); // for debug only
    require_once($filename);
}
$app->get('/books/all', 'App\Services\GetAllBooks');
$app->post('/books/add', 'App\Services\AddBook');
$app->put('/books/update/{book_id}', 'App\Services\UpdateBook');
$app->delete('/books/delete/{book_id}', 'App\Services\DeleteBook');

// Run app
$app->run();
?>
```

10. Please ensure that all of your API routes are placed before the "***$app->run();***" function in the php file as shown.

```php
...

// Define app routes here

...

... //your API routes

...


// Run app
$app->run();
?>
```

11. Finally, the file "***index.php***" should be uploaded to the directory "***/var/www/html/public***" to overwrite the existing ***"index.php"*** file via FileZilla.

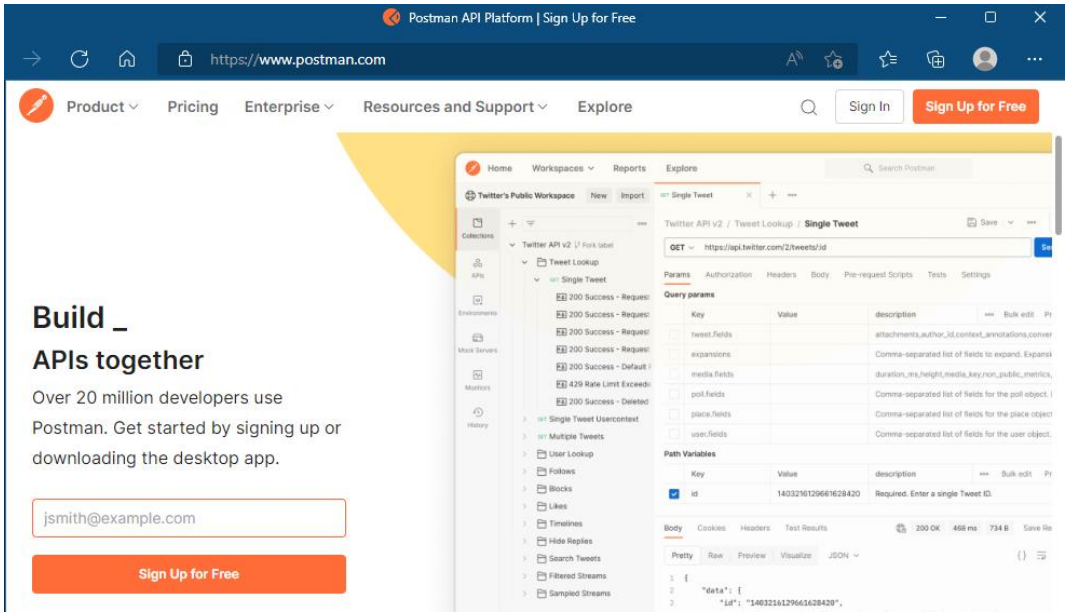




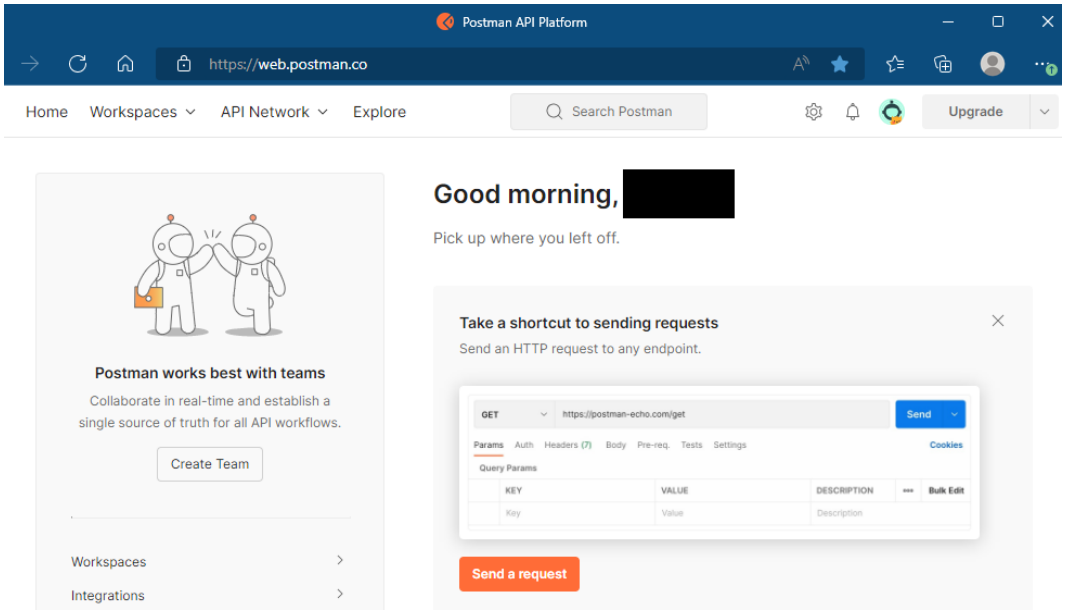
Testing your APIs by Postman

After completion of your APIs code, you would like to test it before building your web application. Postman is one of the most famous tools for testing and managing web APIs. It provides an API repository that you can easily store, catalog, and collaborate around all your APIs on one central platform. You can also make use of the toolset to help in design, testing and documentation of your APIs.

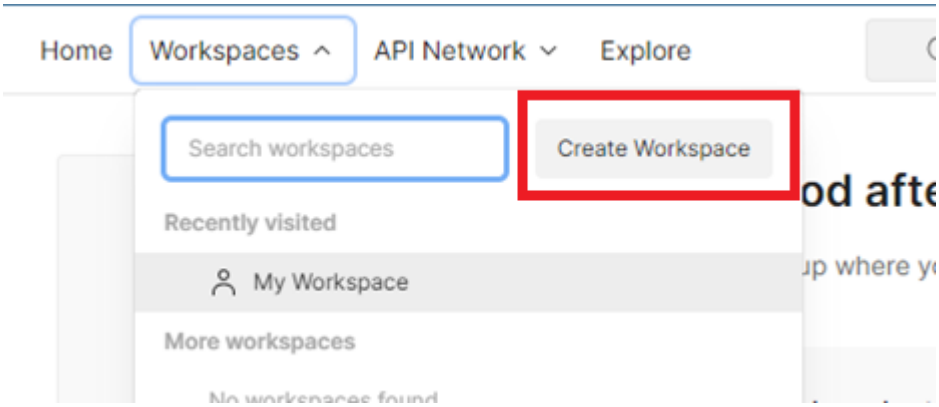
1. Open a browser and go to the website: https://www.postman.com/



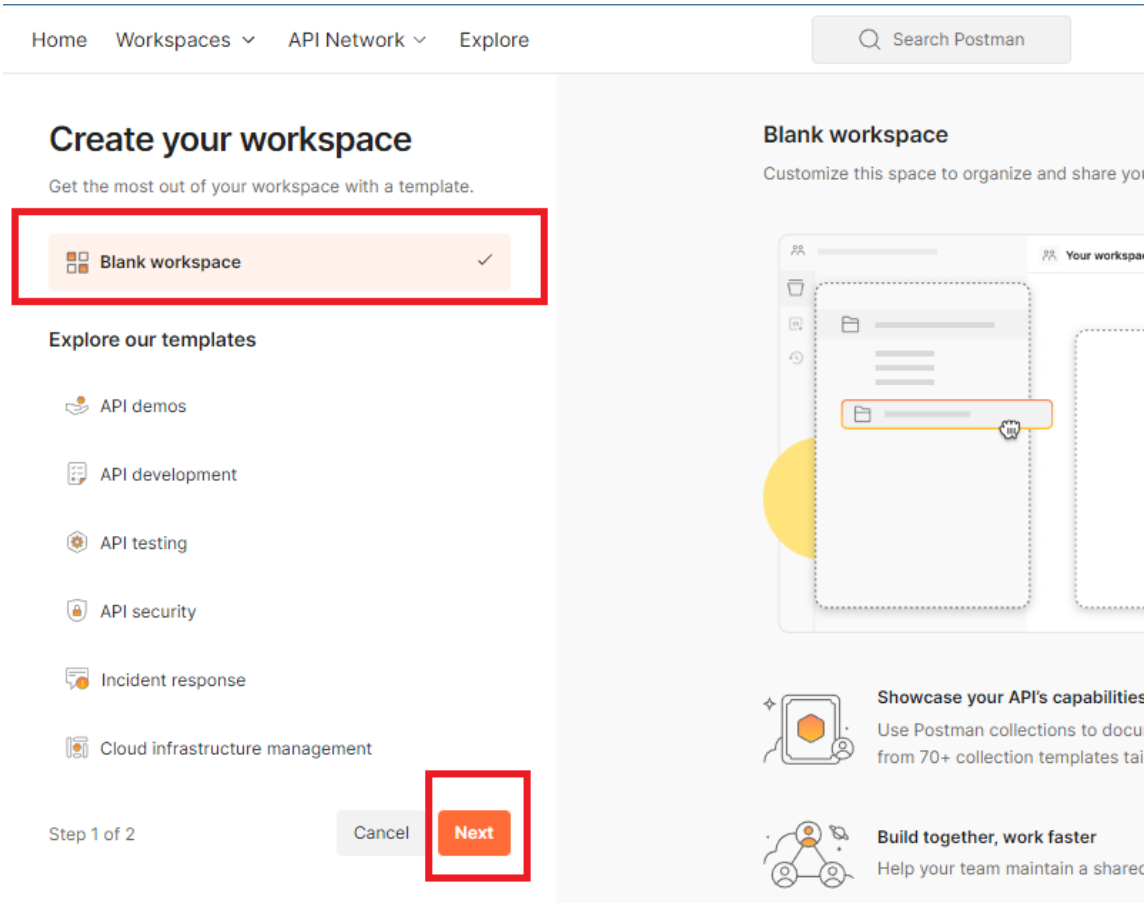
2. Sign in the platform with your account. You can create a new account or just login to your Google account.



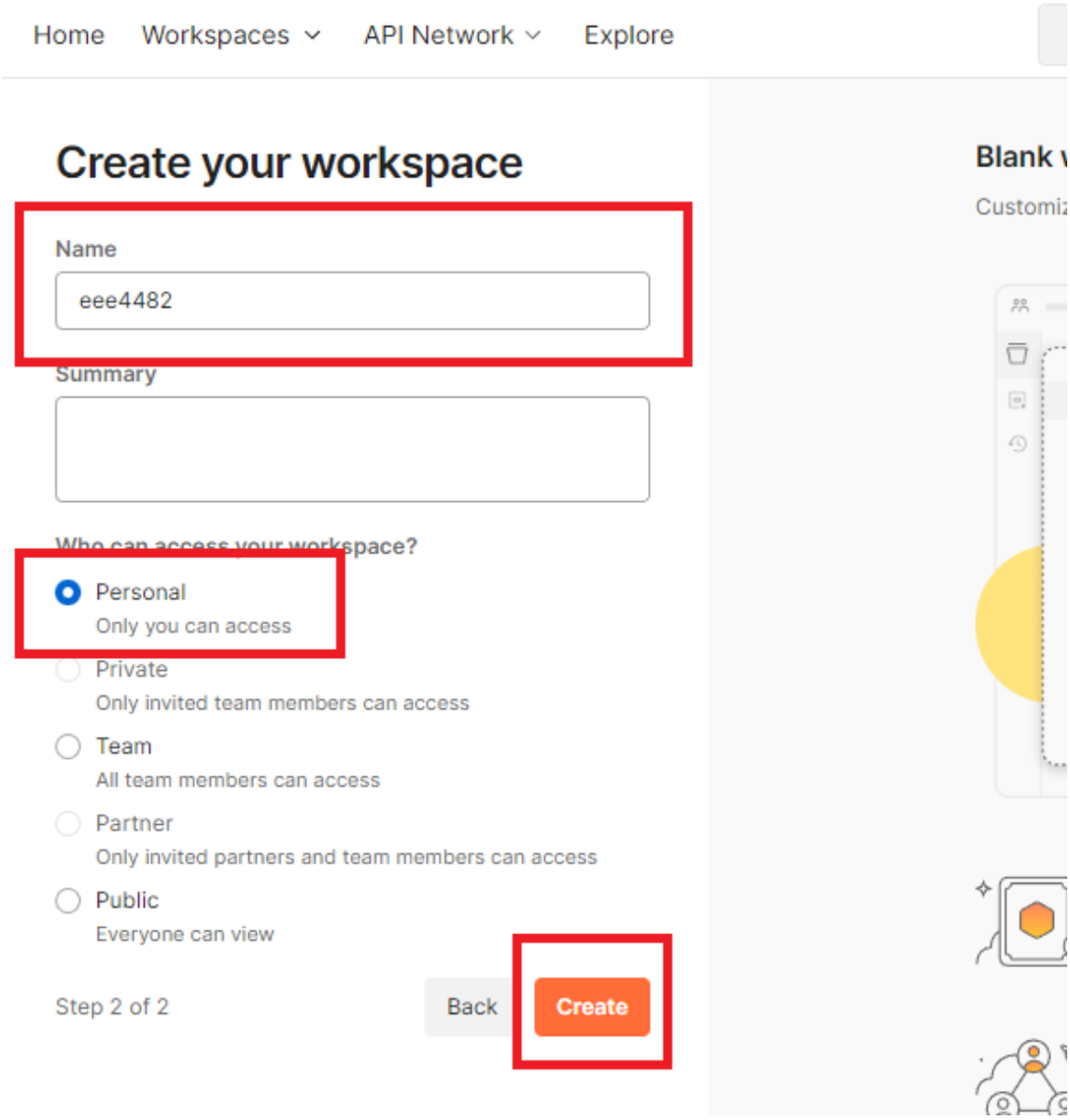
3. Create a workspace from the top-left menu.



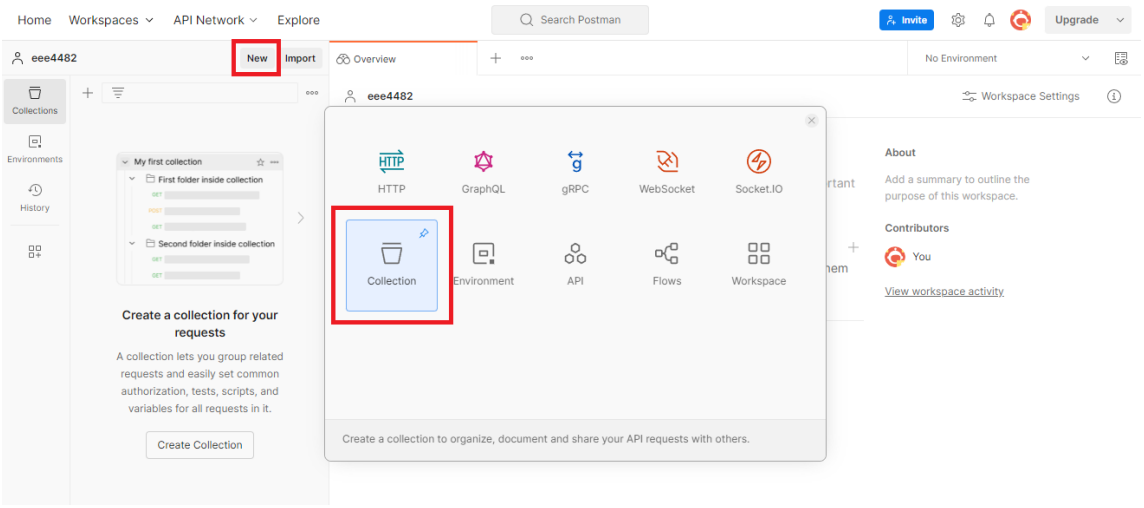
Select “Blank workspace”



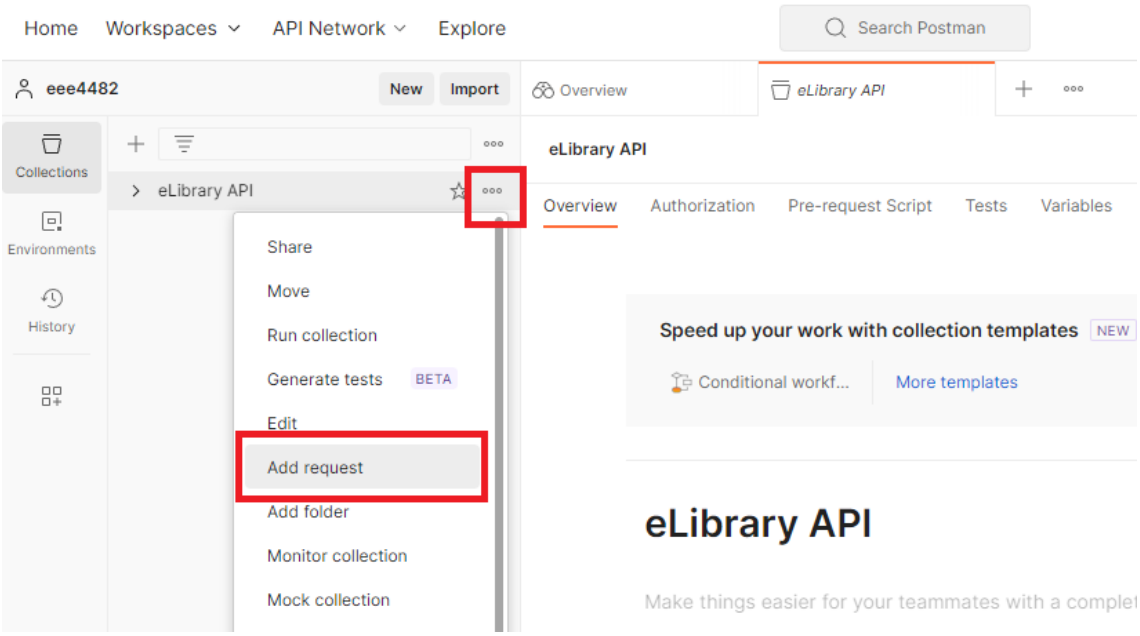
Input the name “eee4482” and select “Personal” for the visibility.

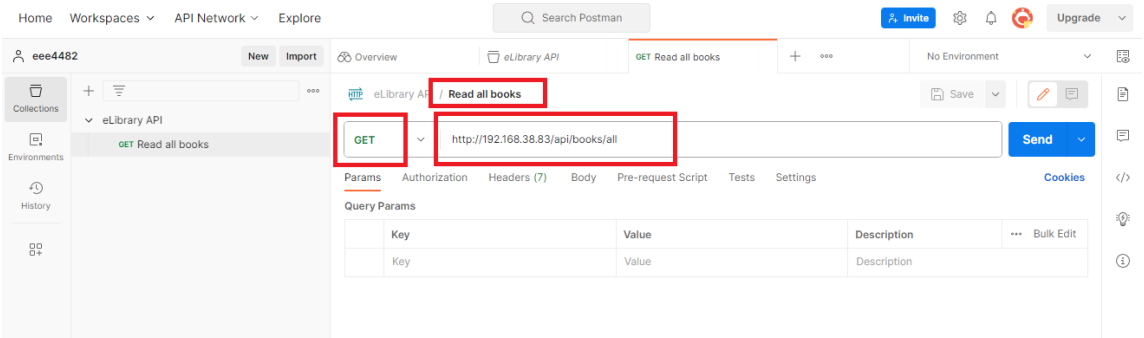


4. Create a **Collection** in your workspace. You can give a name to your **Collection**, e.g. “eLibrary API”.

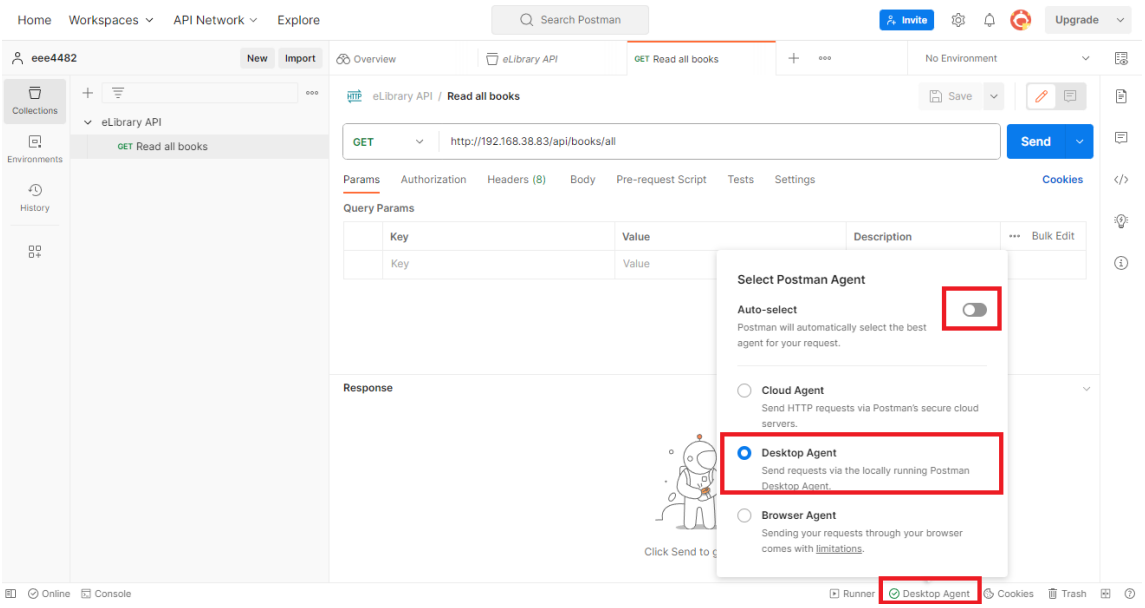


5. In your Collection, click "Add request" to create a HTTP request for testing your API. Input the name and URL as the picture shown. The HTTP GET method should be selected for the request. You can leave the Query Params as default values. **You must click "Save" button or press "ctrl + S" after completing the settings.**

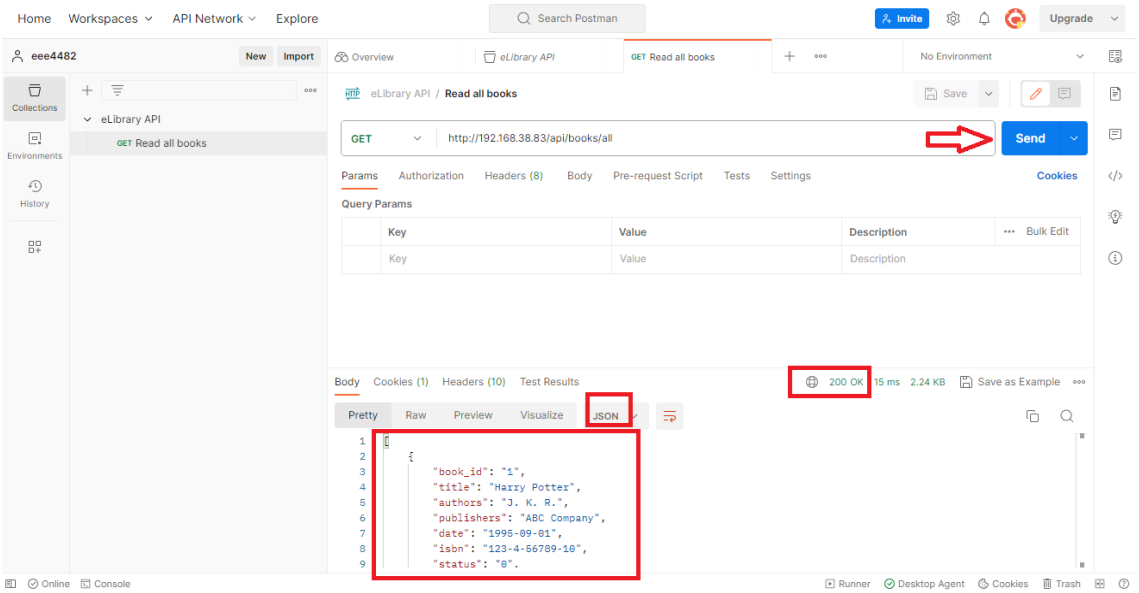




6. Before clicking on the "Send" button, you have to switch the platform agent into "**Desktop Agent**" by selecting it at the bottom tool bar. You may be asked to install the desktop agent before activating the function.

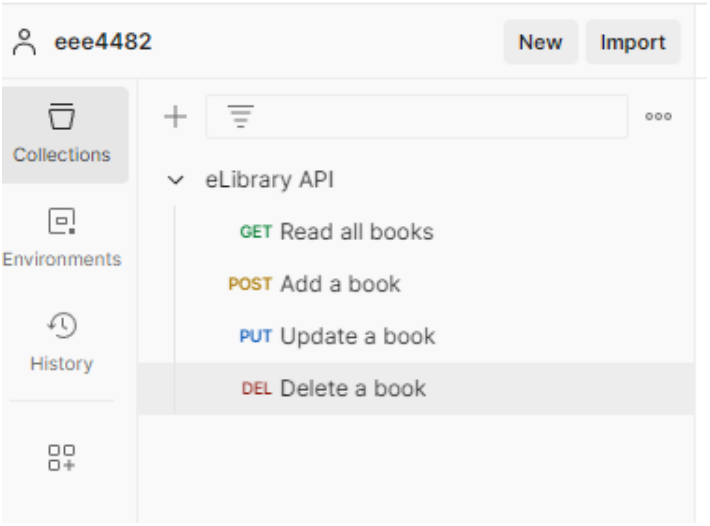


7. Click the "Send" button, the platform will make the HTTP request to your API endpoint. If your request is successful, a "200" response code will be shown on the page, and the details of book record will be listed in the box.



8. Then we are going to add other APIs to the Postman, you can input the names and URLs as the following table.

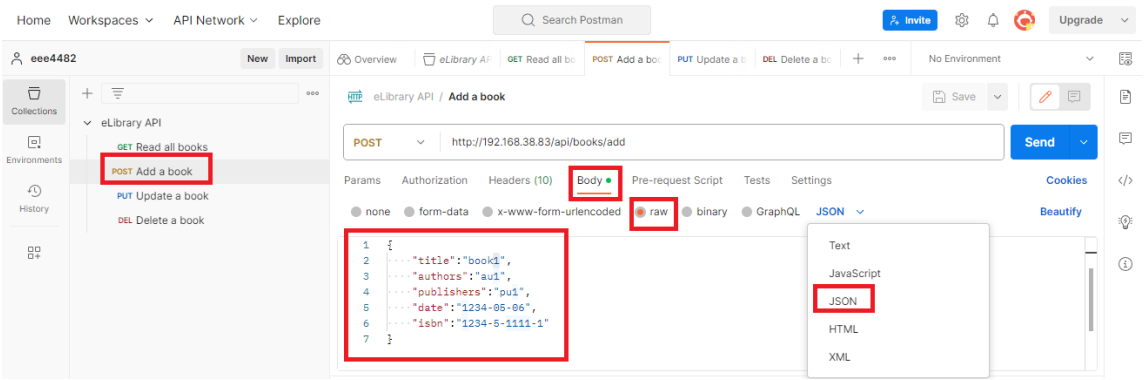
Name	**HTTP Method**	**URL**
Read all books	GET	http:// [your IP] /api/books/all e.g. http://192.168.38.83/api/books/all
Add a book	POST	http:// [your IP] /api/books/add e.g. http://192.168.38.83/api/books/add
Update a book	PUT	http:// [your IP] /api/books/update/[book_id] e.g. http://192.168.38.83/api/books/update/1
Delete a book	DELETE	http:// [your IP] /api/books/delete/[book_id] e.g. http://192.168.38.83/api/books/delete/1



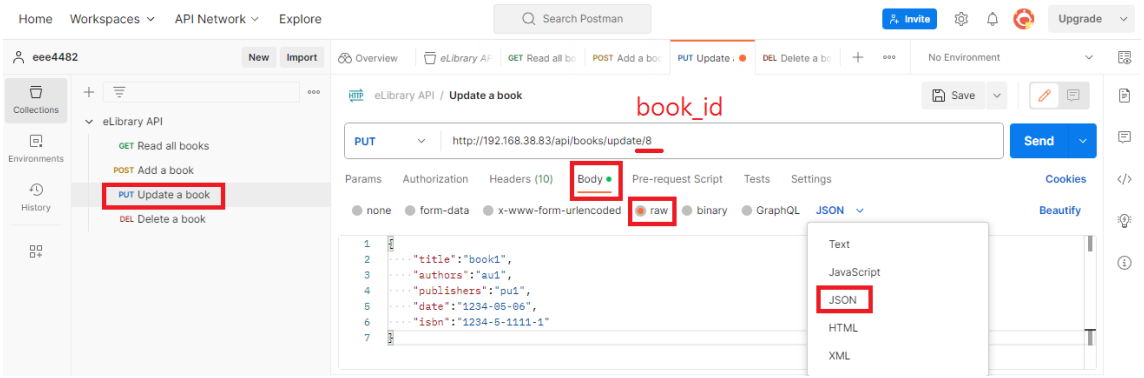
(The logo of APIs will be updated only after you clicked the save button.)

9. In the POST and PUT request, you are required to add contents (e.g. book's title, authors, publishers and etc.) to the request **body** for using the API. The contents should be set as JSON format. You can edit the request body by following the below pictures.

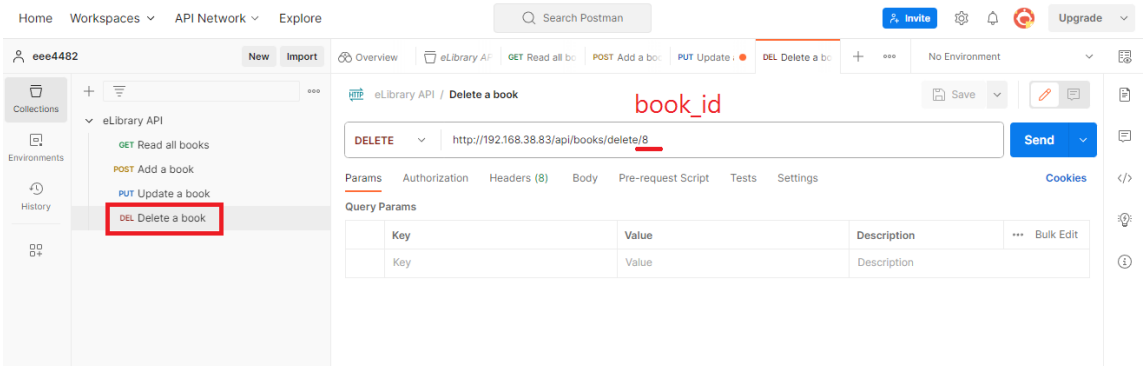
For “Add a book”,



Similarly, if you want to “Update a book”,



If you want to “Delete a book”,



Please note that you need to add the **[book_id]** of the target book to the end of the URL when using “Update” and “Delete” APIs.

10. Now, you can add more books and update their records in the database through APIs. You can verify the result at the **phpMyAdmin** panel.

Step	Description	JSON Parameter
1	Add book “book2” to the database.	{ "title":"book2",

		"authors":"au2", "publishers":"pu2", "date":"1234-05-07", "isbn":"1234-5-1111-2" }

2	Add book "book3" to the database.	{
		"title":"book3",
		"authors":"au3",
		"publishers":"pu3",
		"date":"1234-05-08",
		"isbn":"1234-5-1111-3"
		}

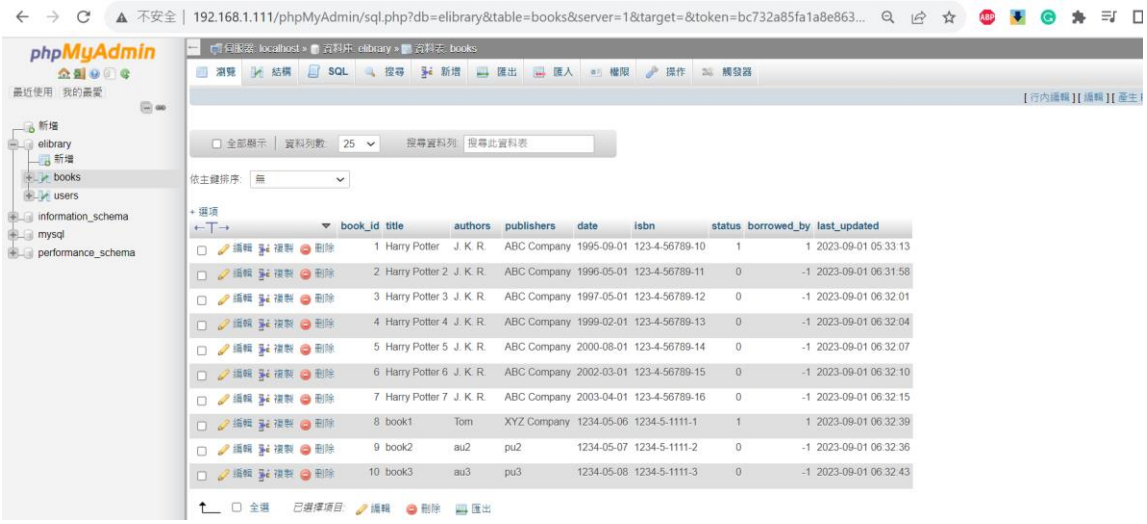
3	Get all books info. You should record the ***book_id*** of the books for later usage.	NA

5	Update book1's record with *book_id* = 8
	
	"authors": "au1" => "Tom"
	"publishers": "pu1" => "XYZ Company"
	
	You should add the **[*book_id*]** of book1 to the end of the URL endpoint.
		{
		"title":"book1",
		"authors":"Tom",
		"publishers":"XYZ Company",
		"date":"1234-05-06",
		"isbn":"1234-5-1111-1"
		}

6	Update book3's record with *book_id* = 10
	
	"authors": "au3" => "Peter"
	"publishers": "pu3" => "123 Company"
		{
		"title":"book3",
		"authors":"Peter",
		"publishers":"123 Company",
		"date":"1234-05-08",
		"isbn":"1234-5-1111-3"
		}

	You should add the **[*book_id*]** of book3 to the end of the URL endpoint.	
7	Get all books info. Check if the book records are updated correctly.	NA
8	Delete book2 record with *book_id* = 9 You should add the **[*book_id*]** of book2 to the end of the URL endpoint.	NA
9	Get all books info	NA

Sample result at ***phpMyAdmin***:



(The *book_id* of *books* table may be different to your database records)

References

https://www.w3schools.com/php/
https://www.w3schools.com/sql/
https://www.slimframework.com/docs/v4/