Fusio Training Notes

Open Source PHP-Based API Management Platform

Prepared by Taming Tech Sdn Bhd

About Fusio

Fusio is an open source API management platform which helps to build and manage REST APIs. Fusio provides all the tools to quickly build an API from different data sources yet it is possible to create complete customized responses.

Fusio is being developed by Christoph Kappestein. More information about Fusio is available at https://www.fusio-project.org/about.

Taming Tech Sdn Bhd
239A Jalan Bandar 13
Taman Melawati
53100 Kuala Lumpur

iszuddin@tamingtech.my
011-5878 9190

https://tamingtech.my

Introduction

This document was created as a set of tutorials for exercise to be carried out in an in-class training environment. The tutorials in each part helps participants experience and understand the concept in building a REST API with Fusio.

Because of this, the tutorial are not in-depth and does not explain every aspect of Fusio. This also means that you are not supposed to use this document as a reference manual.

Other reading sources that can help you with your Fusio learning is listed below:

Fusio Manual
https://docs.fusio-project.org/

PHP API Reference
https://docs.fusio-project.org/docs/concepts/php_api/

Github
https://github.com/apioo/fusio

Doctrine DBAL
https://www.doctrine-project.org/projects/doctrine-dbal/en/current/index.html

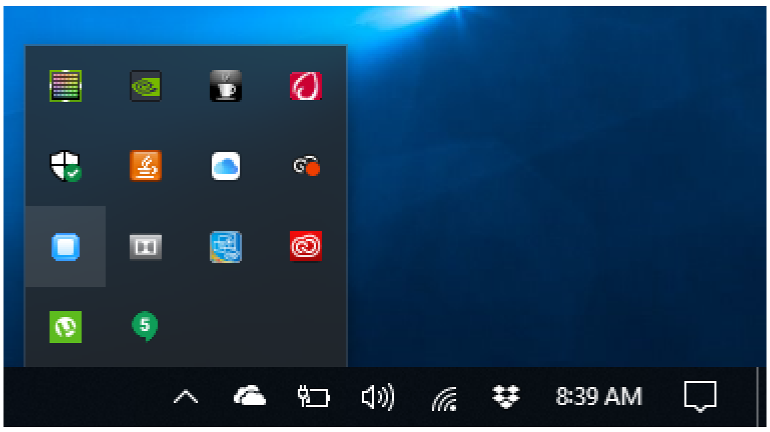
How to Install Fusio on Laragon

This tutorial will walk you through installing Fusio on your Laragon. It will assume that you already have Laragon installed on your computer.

Disabling Auto Virtual Host

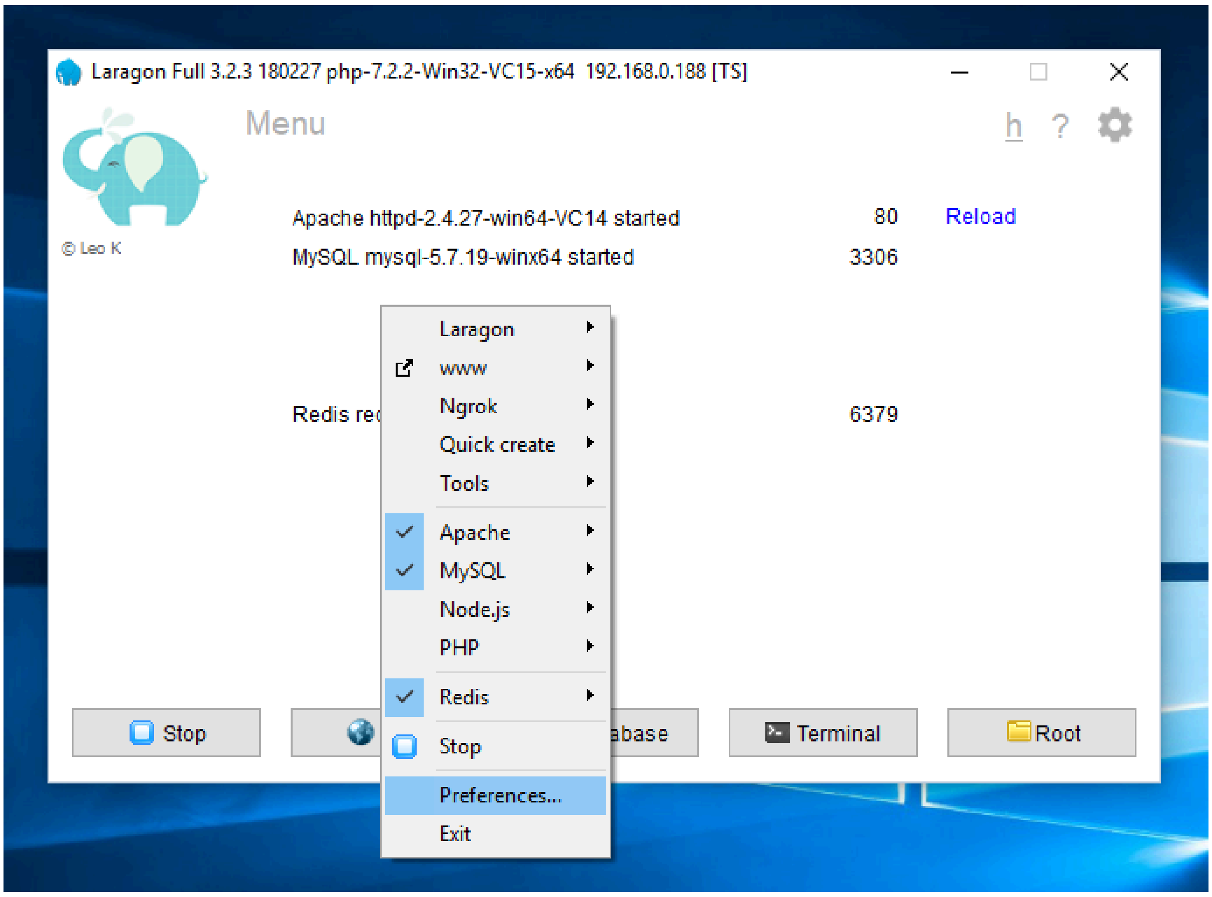
When installing a new project, it is better to have a unique URL for it. Laragon has a feature where it can automatically create a new virtual host when you create a new Project. But I prefer to handle my virtual hosts manually.

To do that, bring up your Laragon Control Panel. If you have not launched your Laragon, please do so. Then you should be able to launch the Control Panel from the right side of your taskbar.



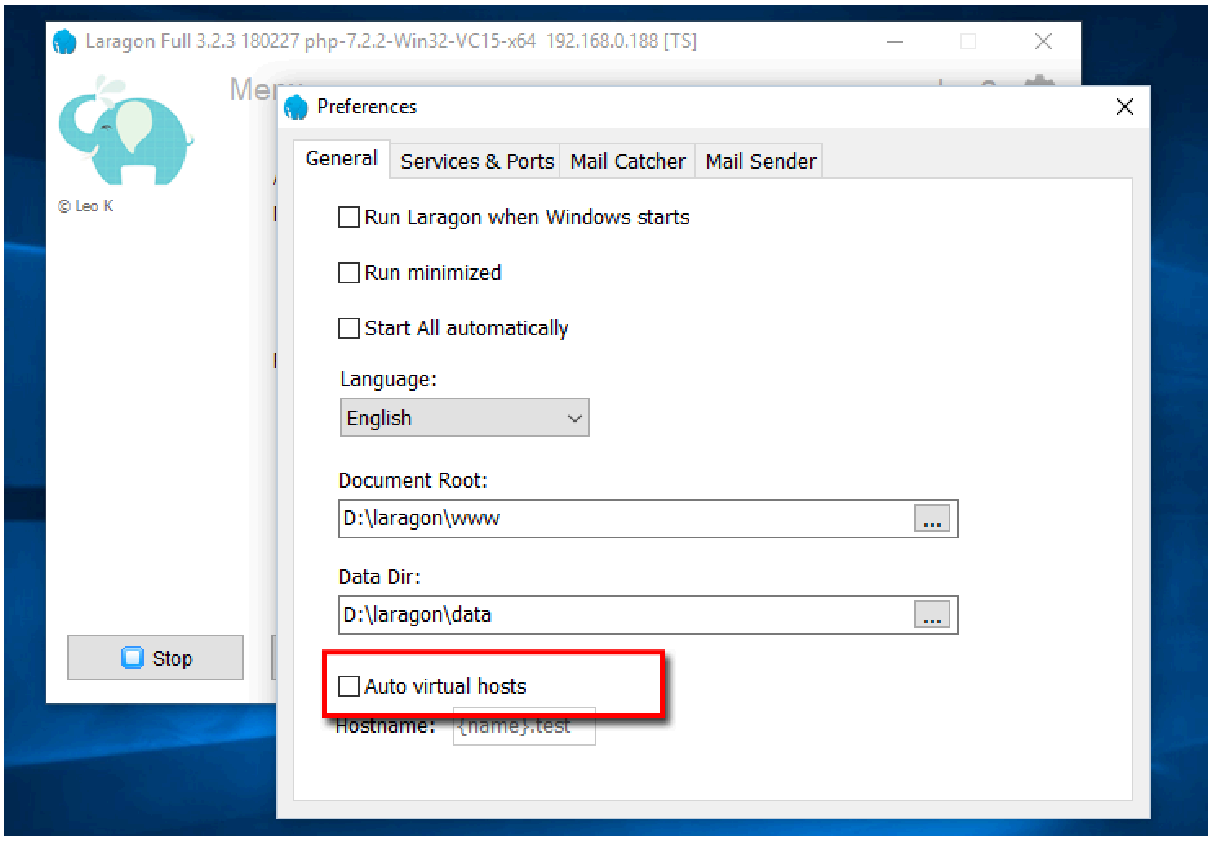
Click on the Laragon icon, to launch the Control Panel.

First click on Start All button to launch all services. Then make a right click anywhere on its surface. It should bring up the menu.


Laragon Full 3.2.3 180227 php-7.2.2-Win32-VC15-x64 192.168.0.188 [TS]

Apache httpd-2.4.27-win64-VC14 started 80 Reload
MySQL mysql-5.7.19-winx64 started 3306
Redis rea 6379

Now click on Preferences. Here you will see the main settings for your Laragon. Make sure that “Auto virtual hosts” is **unchecked**.



Laragon Full 3.2.3 180227 php-7.2.2-Win32-VC15-x64 192.168.0.188 [TS]

Preferences

General | Services & Ports | Mail Catcher | Mail Sender

☐ Run Laragon when Windows starts

☐ Run minimized

☐ Start All automatically

Language: English

Document Root: D:\laragon\www

Data Dir: D:\laragon\data

☐ Auto virtual hosts

Hostname: {name}.test

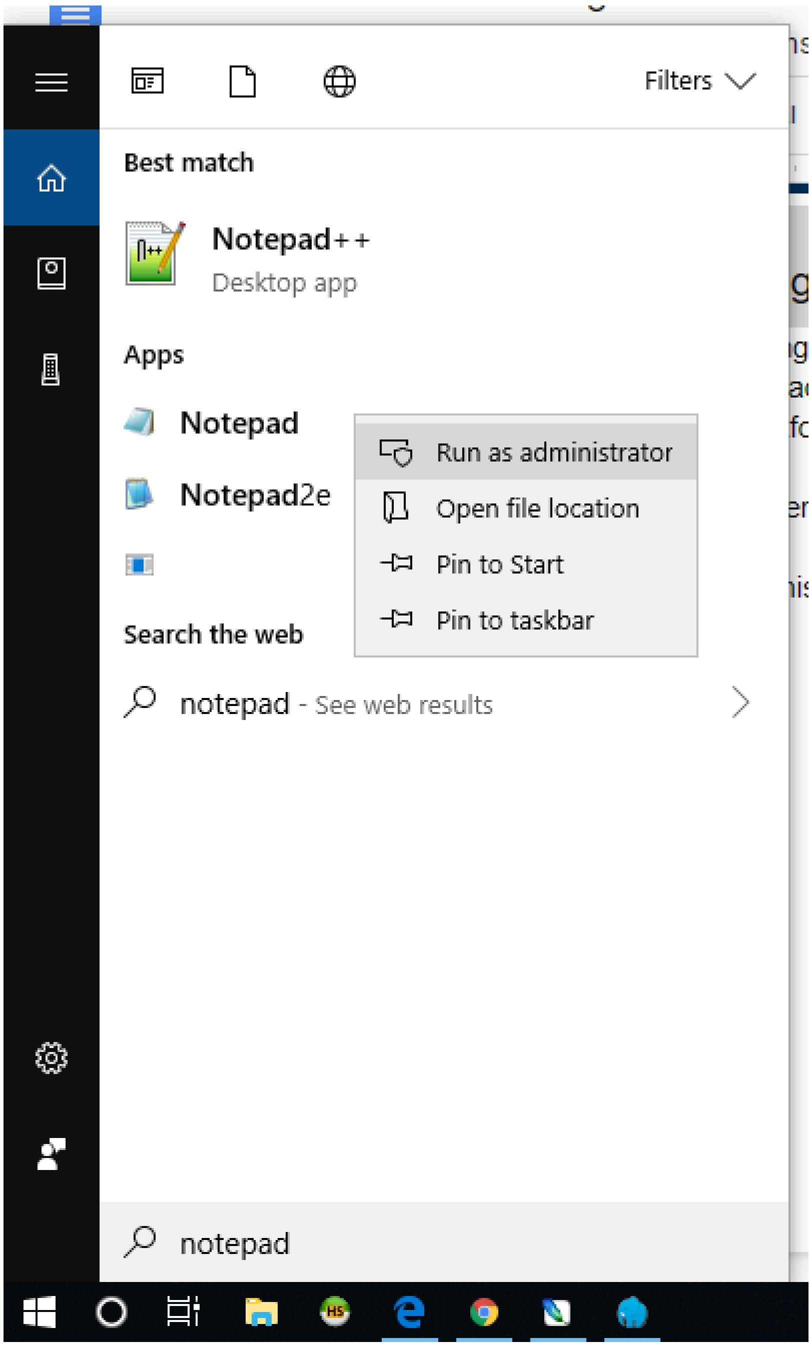
Modifying Windows ***hosts*** file

When doing development on a local machine, we want the address of our project to mimic the real world address as much as possible but will still be able to know that this is a development or test platform.

We will later access our Fusio project from “api.mylocal.test”.

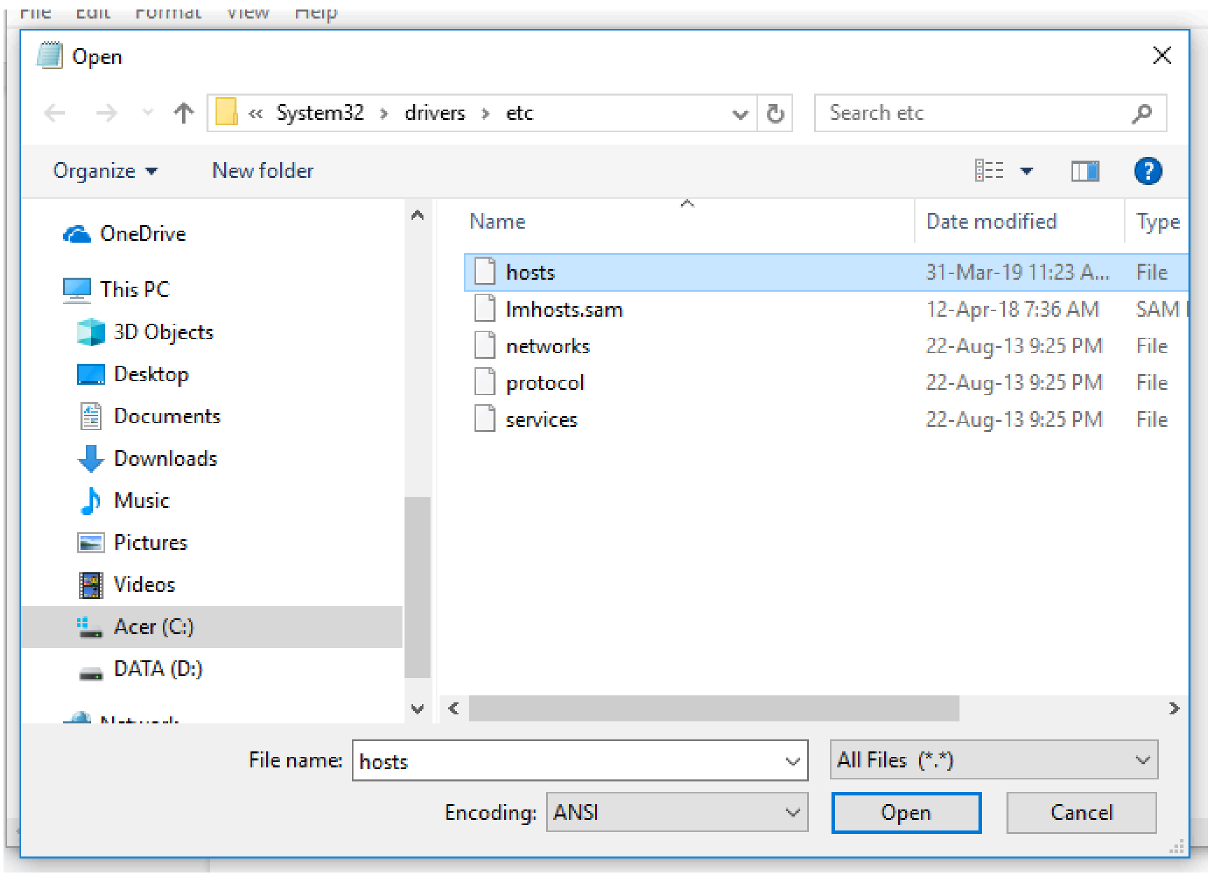
To make this address accessible, we need to modify the hosts file on Windows. The hosts file acts like a local DNS system where you can tell Windows specific URL and their IP address. To be able to edit this file, you need to launch your text editor with Administrator mode.

From the Search panel on the Windows button, search for Notepad. Then right click on the icon and click “Run as Administrator”.



Click Yes when prompted for confirmation.

Now click on File > Open and browse to **C:\Windows\System32\drivers\etc**. Make sure you have **“All Files (*.*)”** selected if you don’t see the ***hosts*** file. Select the hosts file to edit it.



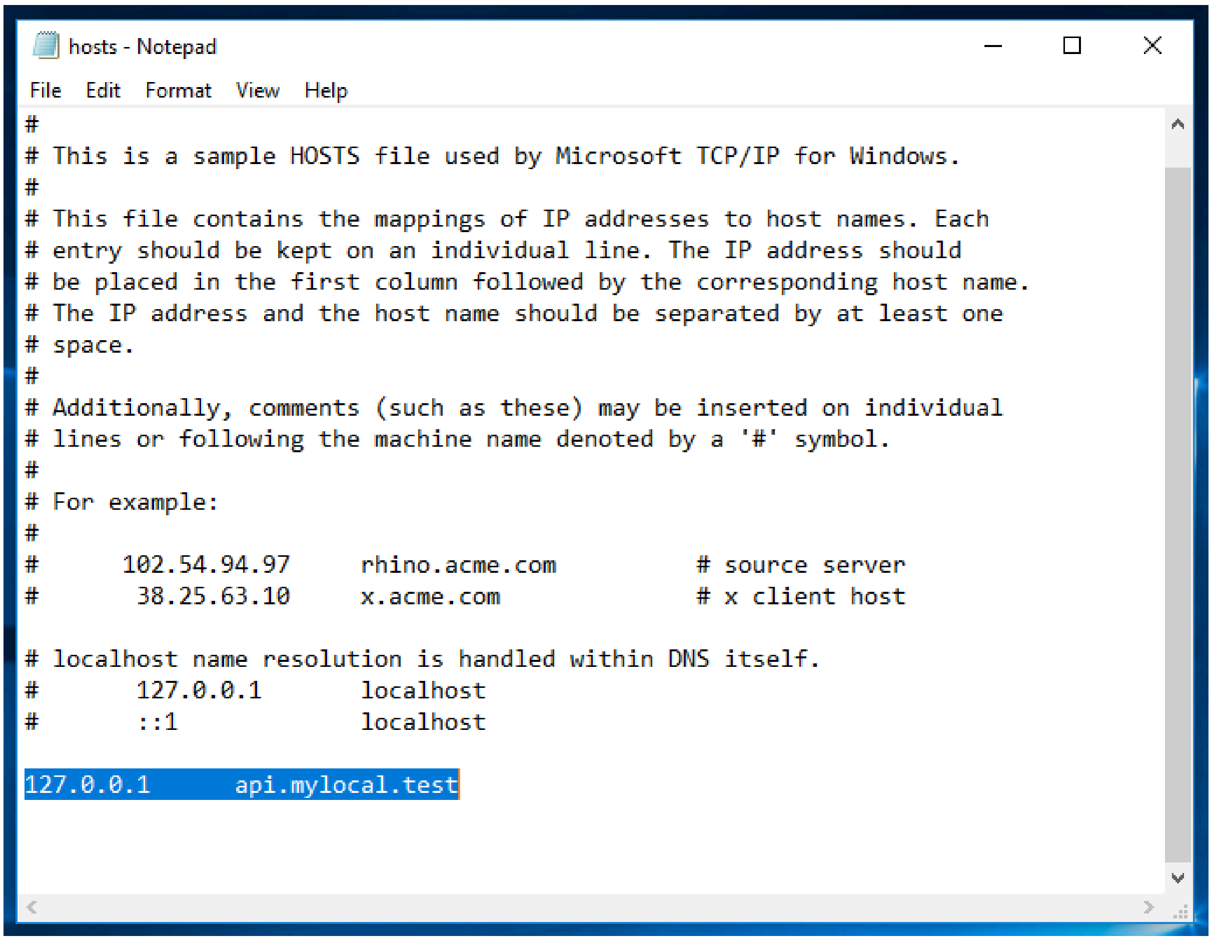
Name	Date modified	Type
hosts	31-Mar-19 11:23 A...	File
lmhosts.sam	12-Apr-18 7:36 AM	SAM
networks	22-Aug-13 9:25 PM	File
protocol	22-Aug-13 9:25 PM	File
services	22-Aug-13 9:25 PM	File

File name: hosts
All Files (*.*)
Encoding: ANSI

Edit your hosts file by adding a new line at the end with the following text.

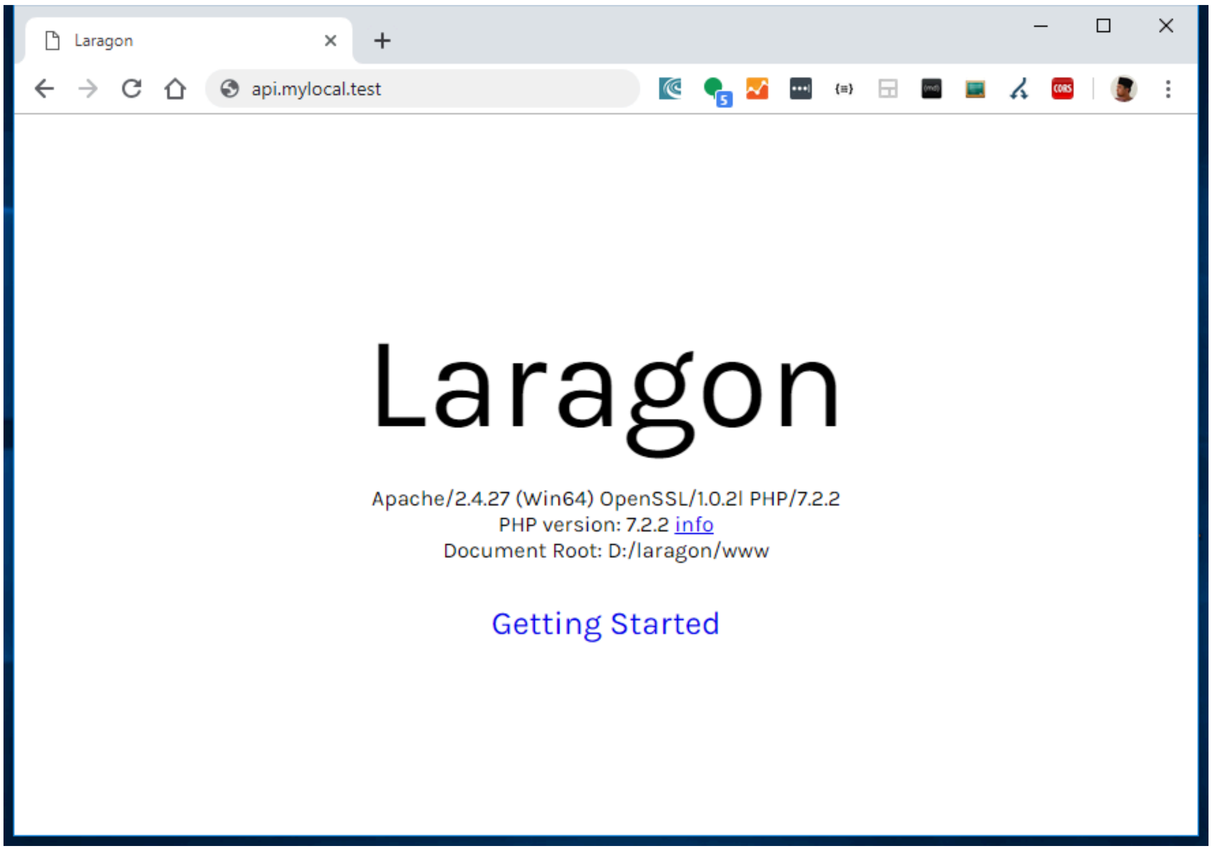
```
127.0.0.1      api.mylocal.test
```

That means, you are adding 127.0.0.1 (your local machine IP address) and giving it a name address, which is api.mylocal.test. Now click File > Save.



```
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost

127.0.0.1       api.mylocal.test
```

By now, if you type this address, **http://api.mylocal.test**, into your browser, you should be able to get Laragon's default website. Be sure that you have clicked on Start All on Laragon and launched your web server.



Laragon

Apache/2.4.27 (Win64) OpenSSL/1.0.2l PHP/7.2.2
PHP version: 7.2.2 info
Document Root: D:/laragon/www

Getting Started

Now we are going to create a virtual host and its own directory folder so we can later install Fusio in it.

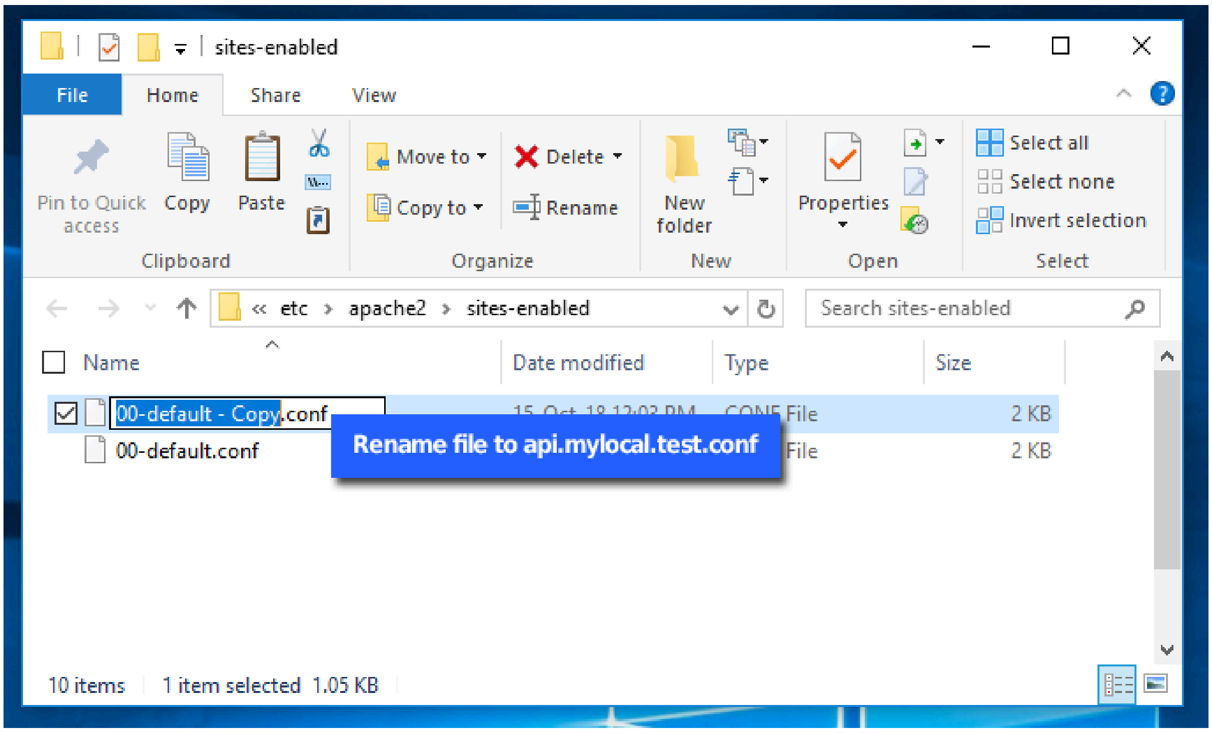
Creating a Virtual Host

From Laragon Control Panel, do a right-click to bring up the menu and browse for

Apache > sites enabled > dir: sites enabled

This will launch a folder that contains configurations for virtual hosts.

Copy and paste **00-default.conf** file. That should make a copy of the file with the name **00-default - Copy.conf**. Rename this file to **api.mylocal.test.conf**.



Now we will edit this file. Use whatever text editor that you are comfortable with to edit this. Laragon comes with Notepad++. You should be able to right-click on the file and select ***Edit with Notepad++***.

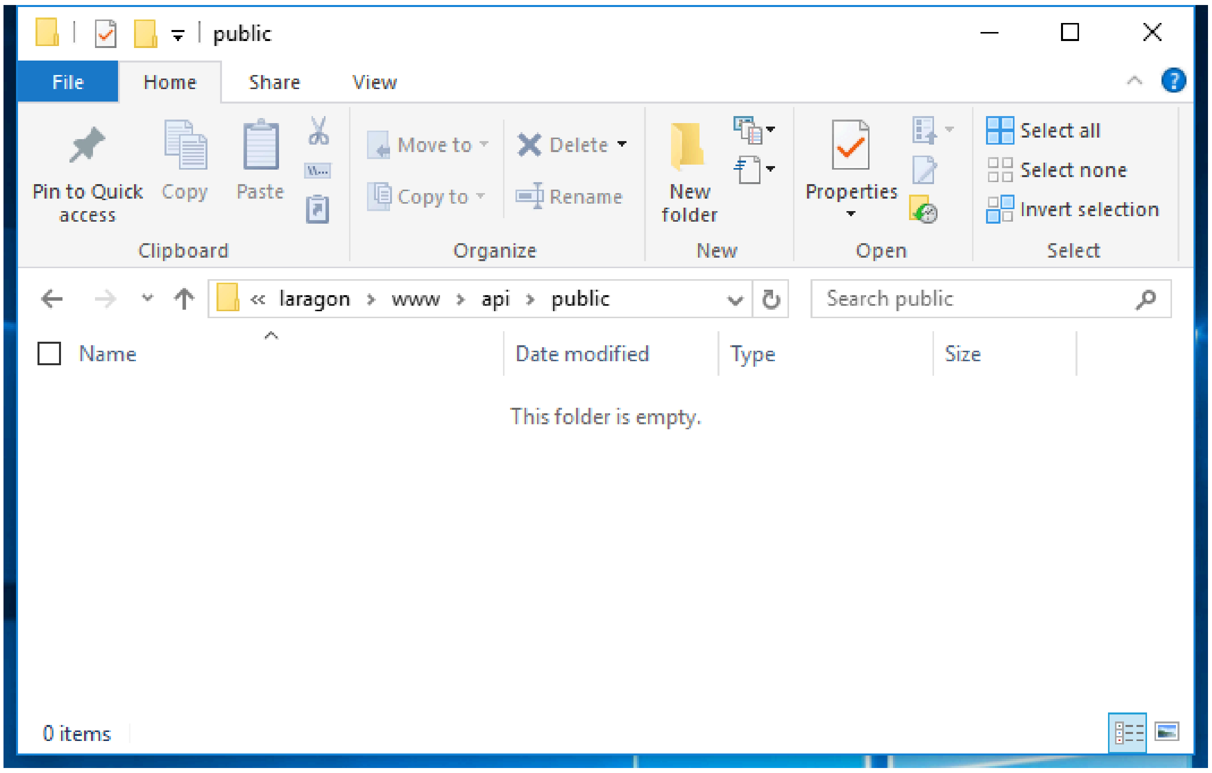
Replace the whole content with the following.

```
<VirtualHost *:80>
    DocumentRoot "C:/laragon/www/api/public/"
    ServerName api.mylocal.test
    ServerAlias *.api.mylocal.test
    <Directory "C:/laragon/www/api/public/">
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

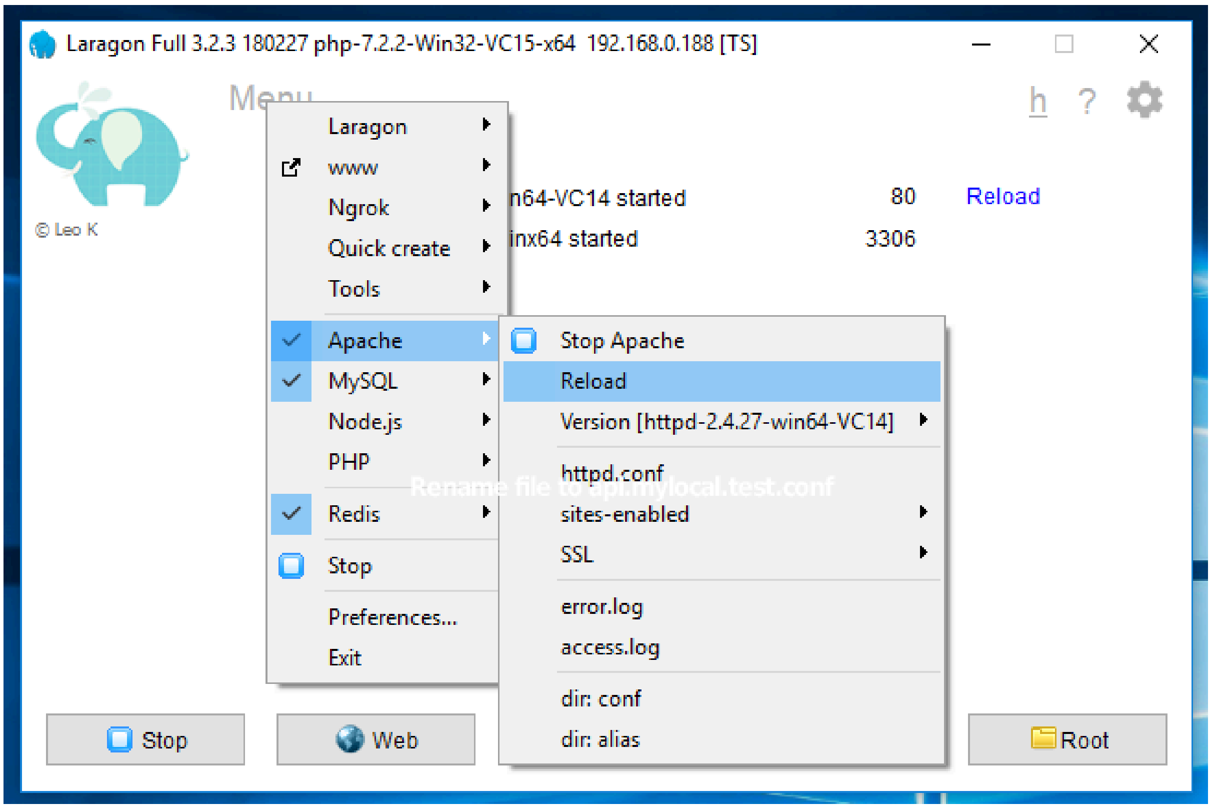
This is assuming that you have Laragon installed on your C: drive. Make necessary adjustments to reflect the correct Laragon installation folder.

Also, you will see that added a path of api/public at the end of the www folder. These paths are not created yet. So we need to create them now.

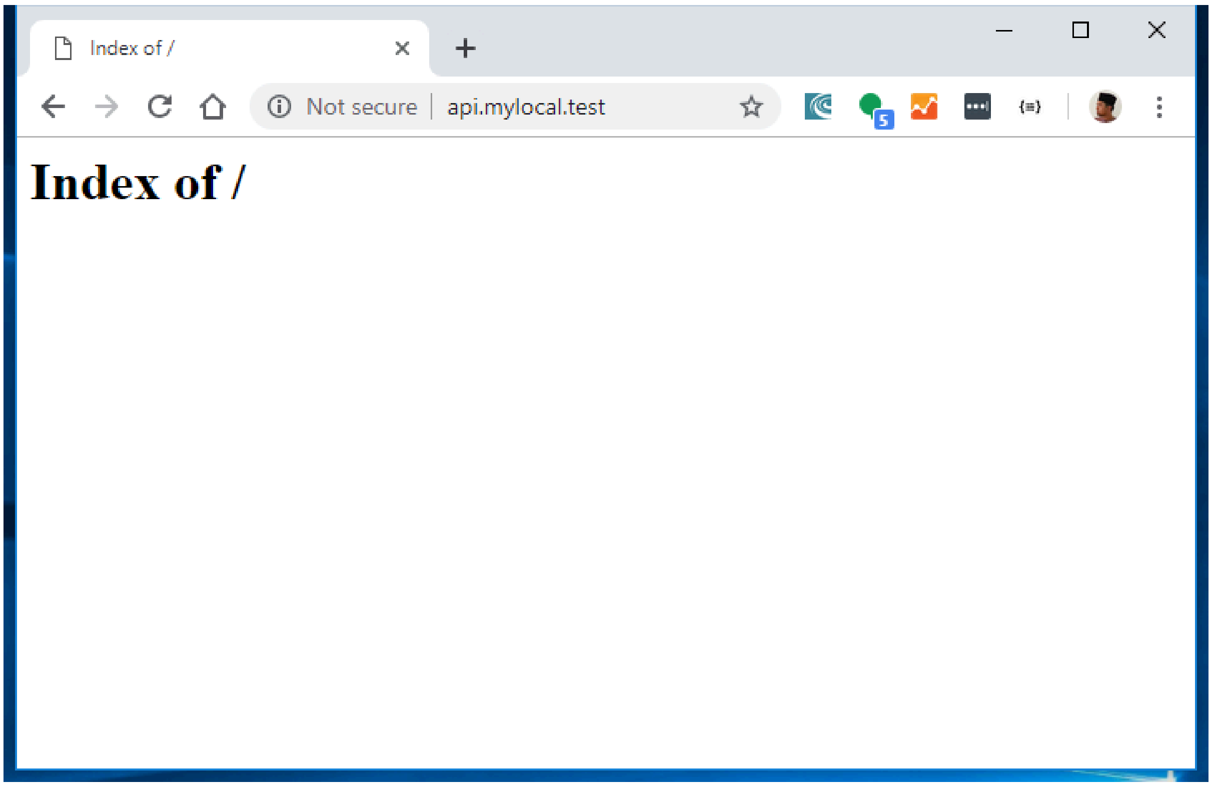
- Browse over to your Laragon's **www** folder and create this path.
- Create a folder **api** folder
- Create a **public** folder inside the api folder



To make sure that this new virtual host takes effect, launch the Laragon Control Panel, right-click to bring up the menu and select Apache > Reload.



Now when you enter the http://api.mylocal.test in your browser, you should get blank empty page because there is no file in the folder.



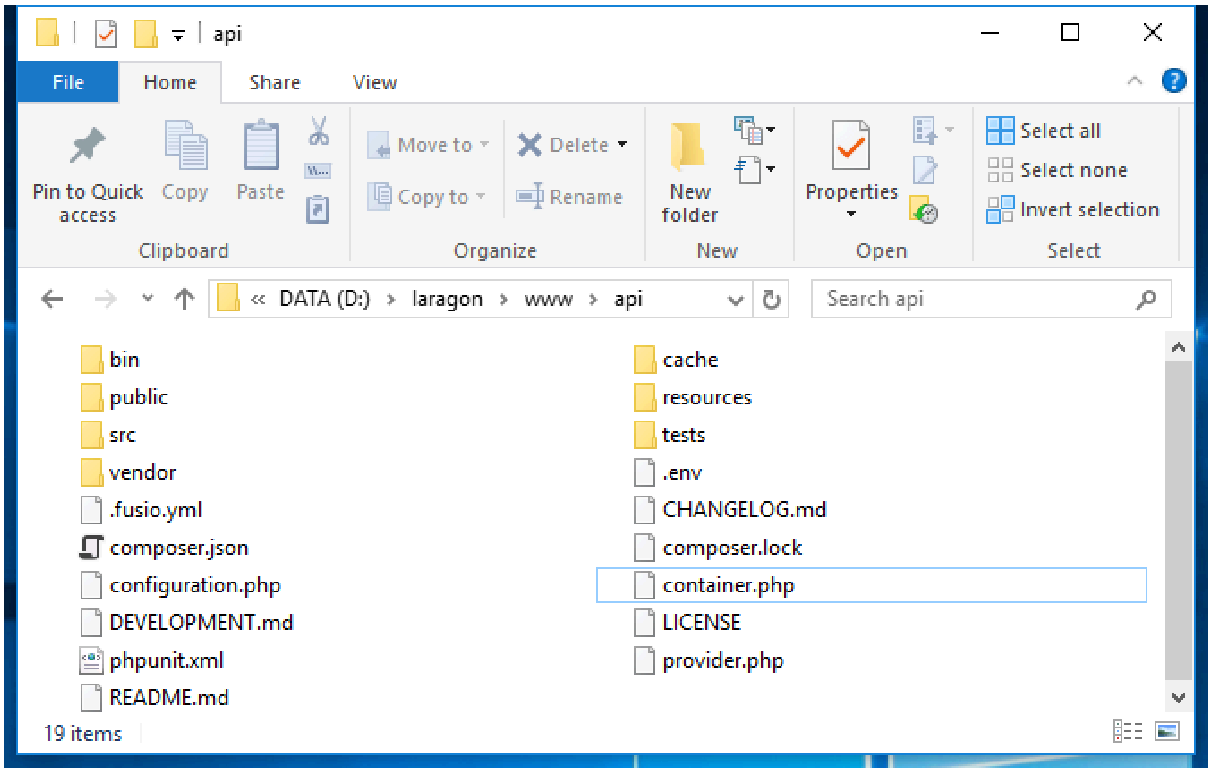
Index of /

Now we can begin installing Fusio.

Downloading Fusio

Browse to https://www.fusio-project.org/download to get the latest release of Fusio. Once downloaded extract the ZIP file into the virtual host folder that we created in previous steps. But please take note that we **will not** extract it into the **public** folder, but the folder above it, **api**.

If requested to merge the public folder, just choose Yes.



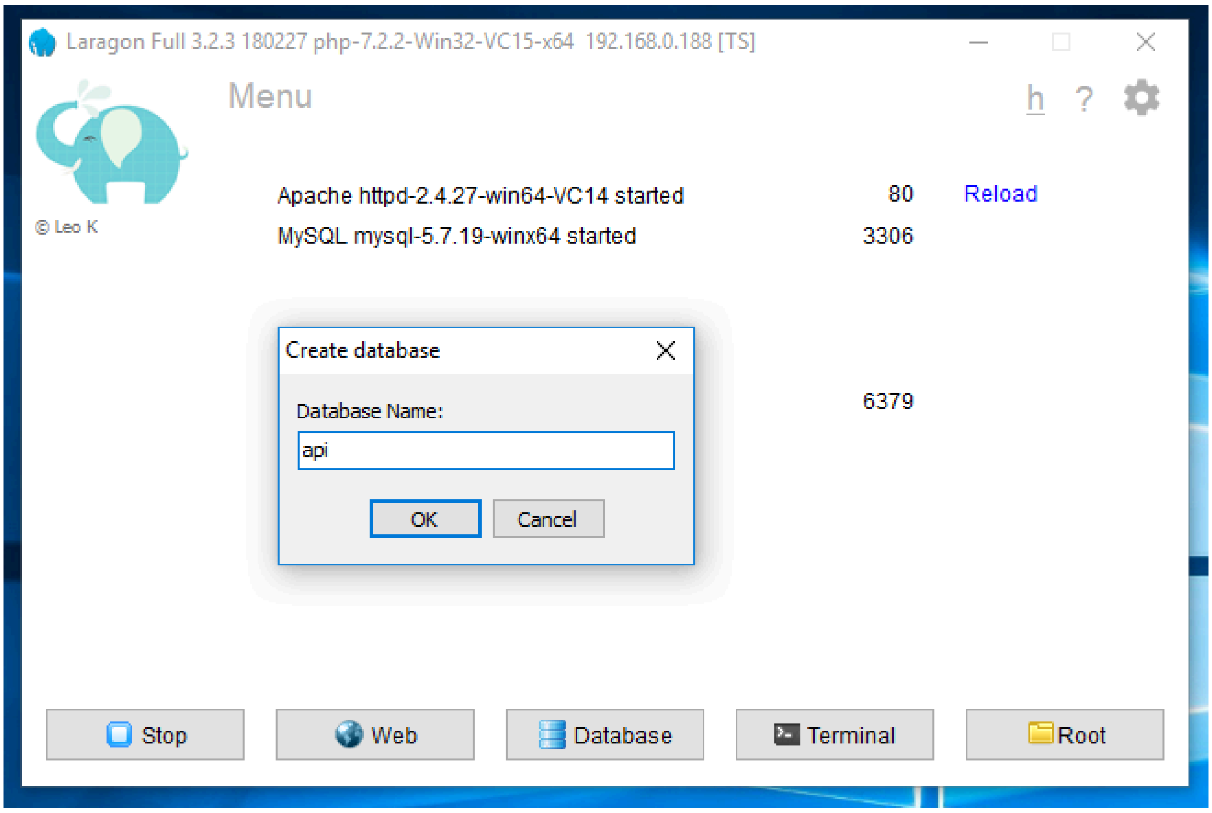
bin
public
src
vendor
.fusio.yml
composer.json
configuration.php
DEVELOPMENT.md
phpunit.xml
README.md
cache
resources
tests
.env
CHANGELOG.md
composer.lock
container.php
LICENSE
provider.php

19 items

Note: *My Laragon is installed on the D: drive. You should extract the ZIP file into your own Laragon installation, and to the virtual host folder that you created in the previous steps.*

Create a New Empty Database for Fusio

Launch Laragon Control Panel, right-click and select MySQL > Create database. Enter “fusio” as database name. Click OK.


Laragon Full 3.2.3 180227 php-7.2.2-Win32-VC15-x64 192.168.0.188 [TS]
Menu
© Leo K
Reload
Create database
OK
Cancel
Stop
Web
Database
Terminal
Root

Apache httpd-2.4.27-win64-VC14 started 80
MySQL mysql-5.7.19-winx64 started 3306
6379

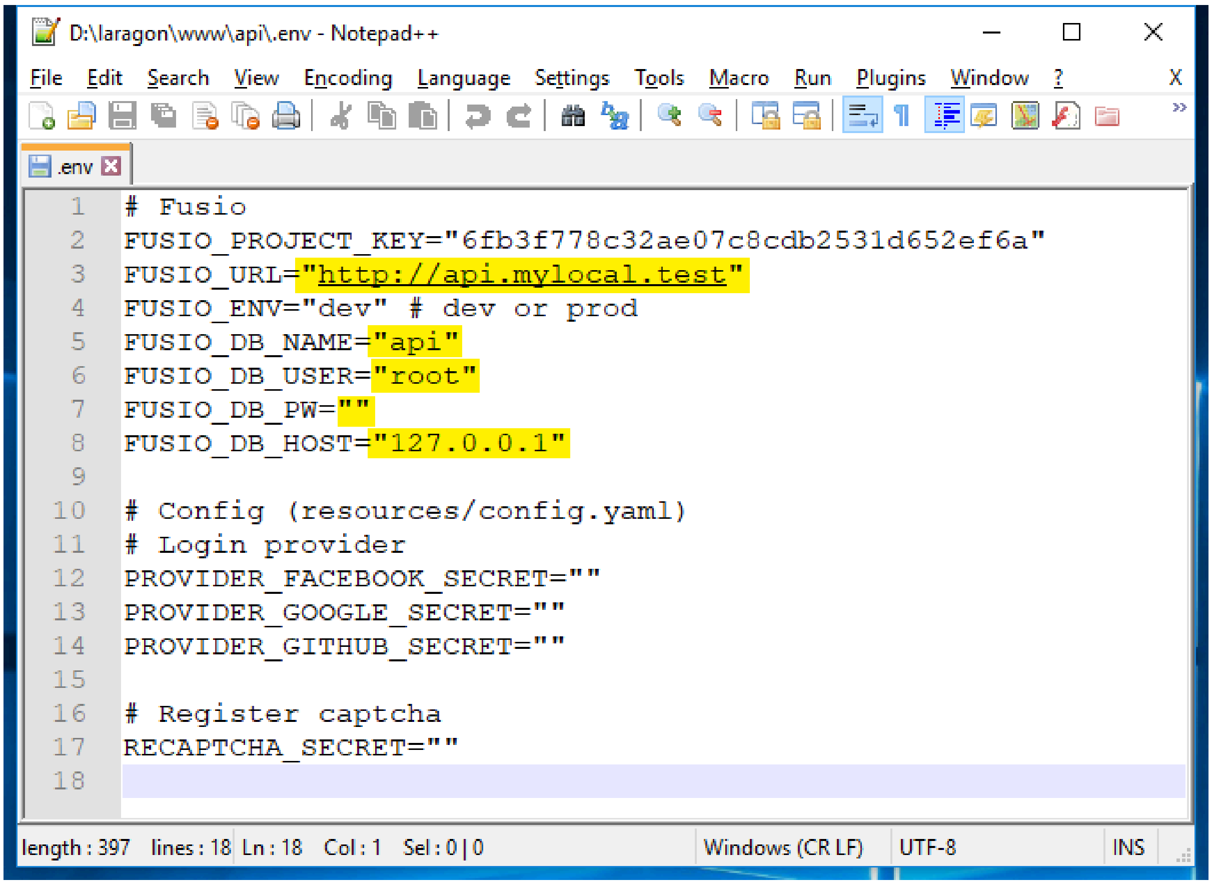
Database Name: api

You can access this database later with the default root username and empty password.

Edit Fusio’s .env File

Browse to your Fusio installation folder (C:\laragon\www\api). Find the .env file, right-click and select Edit with Notepad++.

Edit the file to reflect your own Fusio installation. When done, click File > Save.

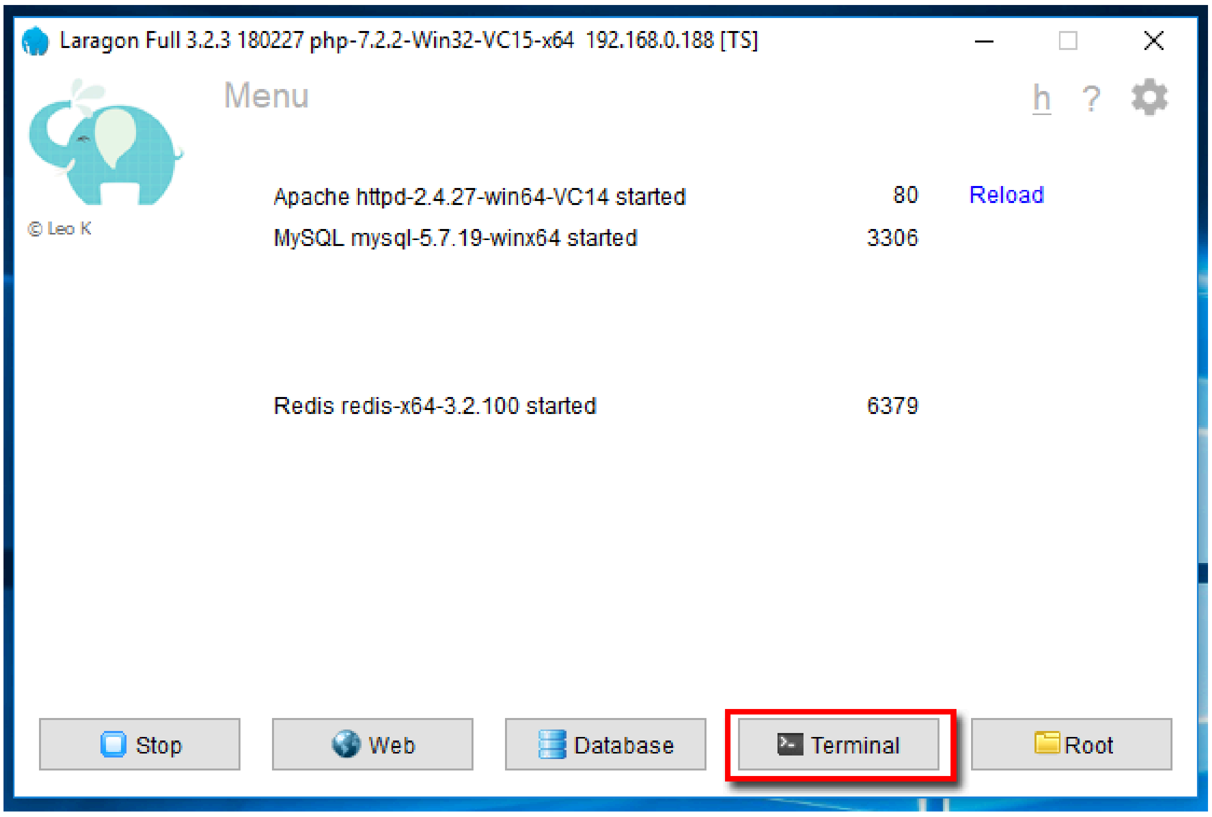

D:\laragon\www\api\.env - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
.env

```
# Fusio
FUSIO_PROJECT_KEY="6fb3f778c32ae07c8cdb2531d652ef6a"
FUSIO_URL="http://api.mylocal.test"
FUSIO_ENV="dev" # dev or prod
FUSIO_DB_NAME="api"
FUSIO_DB_USER="root"
FUSIO_DB_PW=""
FUSIO_DB_HOST="127.0.0.1"

# Config (resources/config.yaml)
# Login provider
PROVIDER_FACEBOOK_SECRET=""
PROVIDER_GOOGLE_SECRET=""
PROVIDER_GITHUB_SECRET=""

# Register captcha
RECAPTCHA_SECRET=""

```

length : 397 lines : 18 Ln : 18 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

Run Fusio Installation Command

The next part will require us to type some commands in the terminal or console. To launch the terminal, launch Laragon's Control Panel and click on the Terminal button at the bottom.

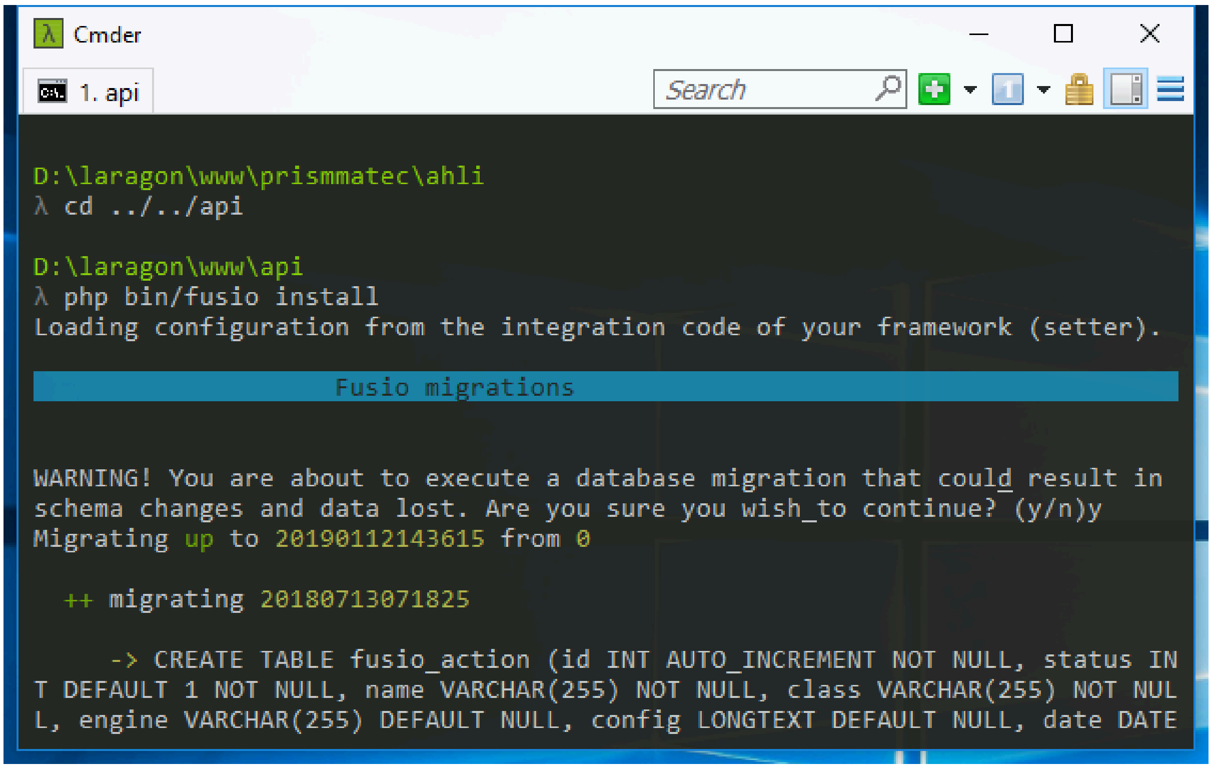


```
Laragon Full 3.2.3 180227 php-7.2.2-Win32-VC15-x64  192.168.0.188 [TS]

Apache httpd-2.4.27-win64-VC14 started     80     Reload
MySQL mysql-5.7.19-winx64 started          3306

Redis redis-x64-3.2.100 started            6379
```

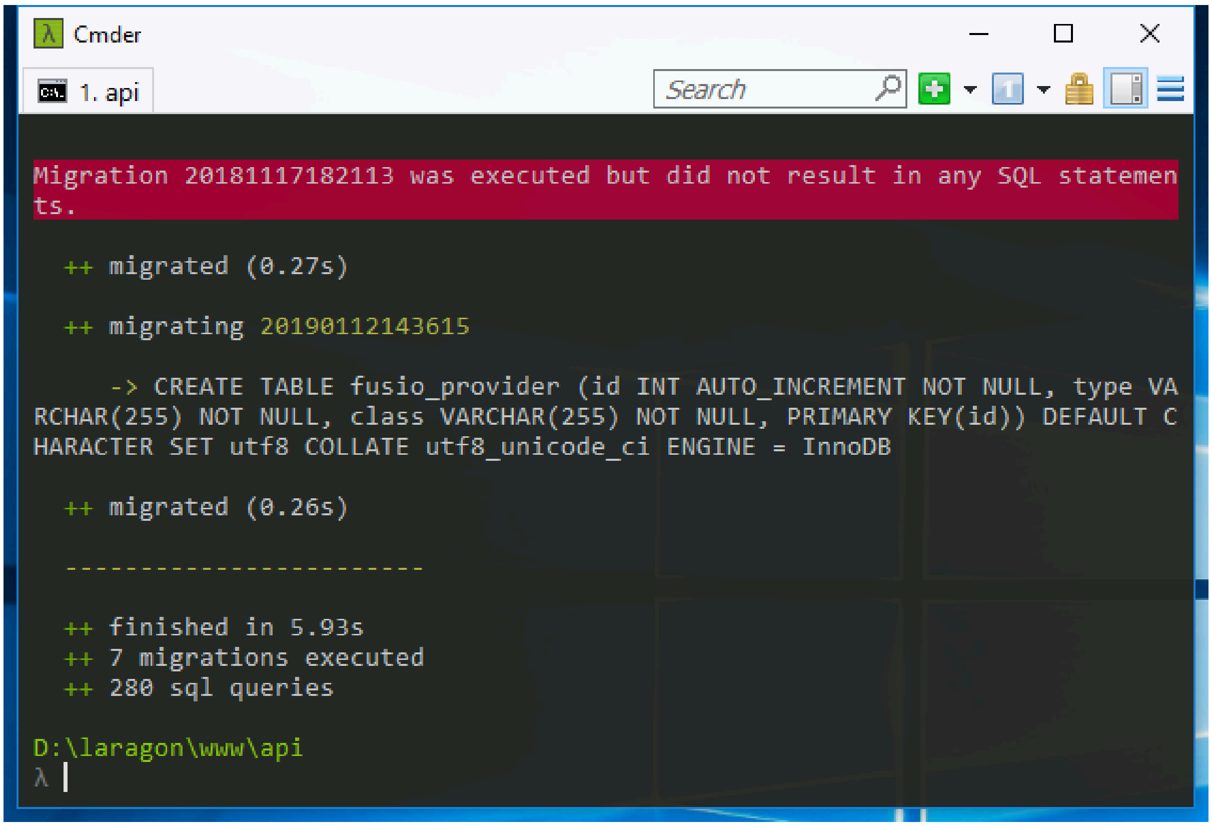
On the terminal, use the **`cd`** command to move to the **api** folder under **www**. Then enter the following command to install Fusio.

```
php bin/fusio install
```

When prompted, enter **y**.

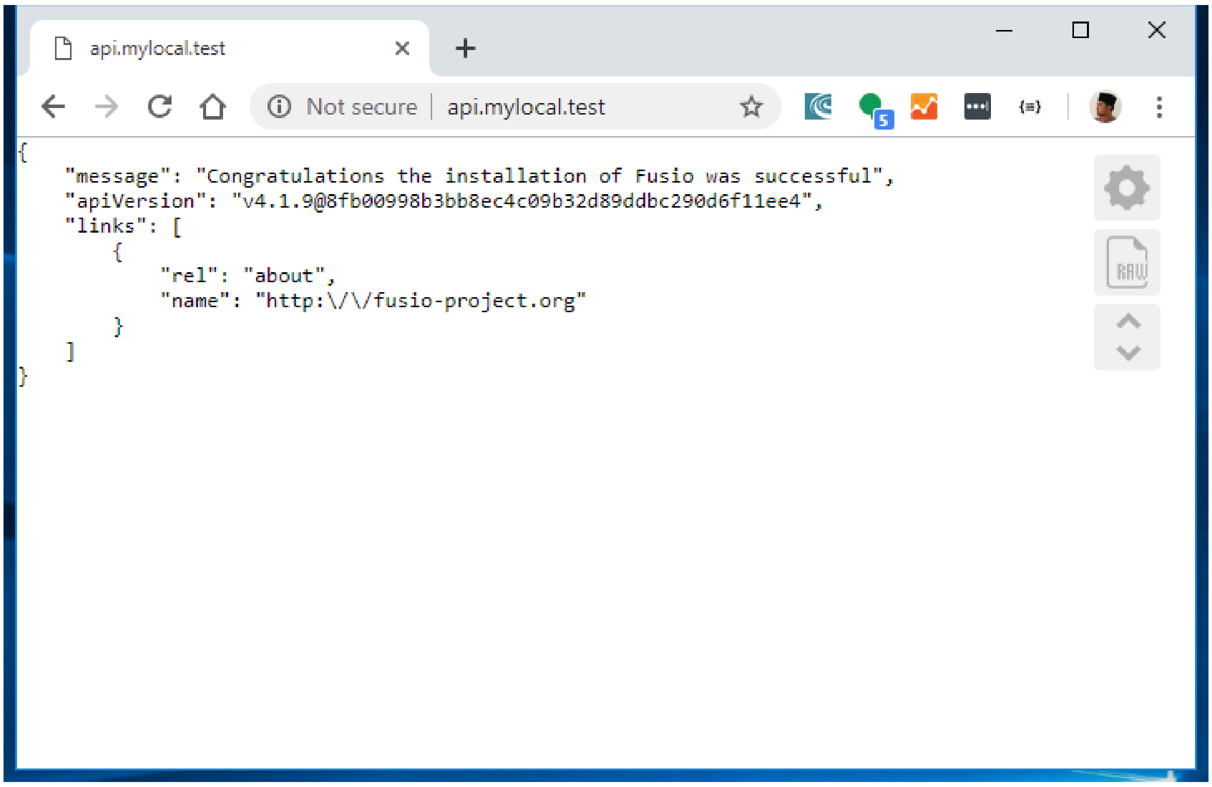


```
D:\laragon\www\prismmatec\ahli
λ cd ../../api

D:\laragon\www\api
λ php bin/fusio install
Loading configuration from the integration code of your framework (setter).

                    Fusio migrations


WARNING! You are about to execute a database migration that could result in
schema changes and data lost. Are you sure you wish to continue? (y/n)y
Migrating up to 20190112143615 from 0

  ++ migrating 20180713071825

     -> CREATE TABLE fusio_action (id INT AUTO_INCREMENT NOT NULL, status IN
T DEFAULT 1 NOT NULL, name VARCHAR(255) NOT NULL, class VARCHAR(255) NOT NUL
L, engine VARCHAR(255) DEFAULT NULL, config LONGTEXT DEFAULT NULL, date DATE
```



```
Migration 20181117182113 was executed but did not result in any SQL statemen
ts.

  ++ migrated (0.27s)

  ++ migrating 20190112143615

     -> CREATE TABLE fusio_provider (id INT AUTO_INCREMENT NOT NULL, type VA
RCHAR(255) NOT NULL, class VARCHAR(255) NOT NULL, PRIMARY KEY(id)) DEFAULT C
HARACTER SET utf8 COLLATE utf8_unicode_ci ENGINE = InnoDB

  ++ migrated (0.26s)

  ------------------------

  ++ finished in 5.93s
  ++ 7 migrations executed
  ++ 280 sql queries

D:\laragon\www\api
λ
```

Once completed, we can try the **http://api.mylocal.test** in the browser again.



```
{
    "message": "Congratulations the installation of Fusio was successful",
    "apiVersion": "v4.1.9@8fb00998b3bb8ec4c09b32d89ddbc290d6f11ee4",
    "links": [
        {
            "rel": "about",
            "name": "http:\/\/fusio-project.org"
        }
    ]
}
```

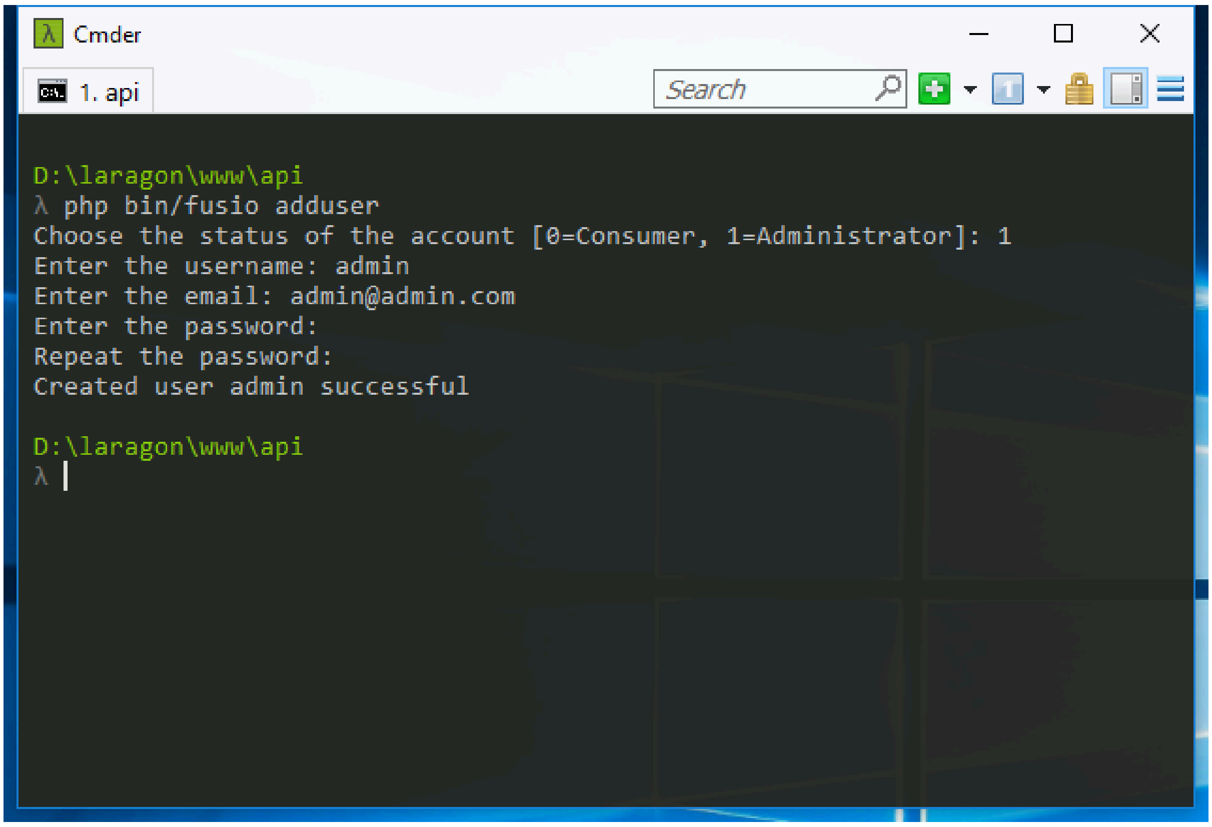
This is the default API response from Fusio. You can see here that what you are getting is a JSON response with information about the installation.

Creating an Administrator User

To be able to manage your API with Fusio, we need to create an Administrator user. Go back to the terminal and enter the following command.

```
php bin/fusio adduser
```

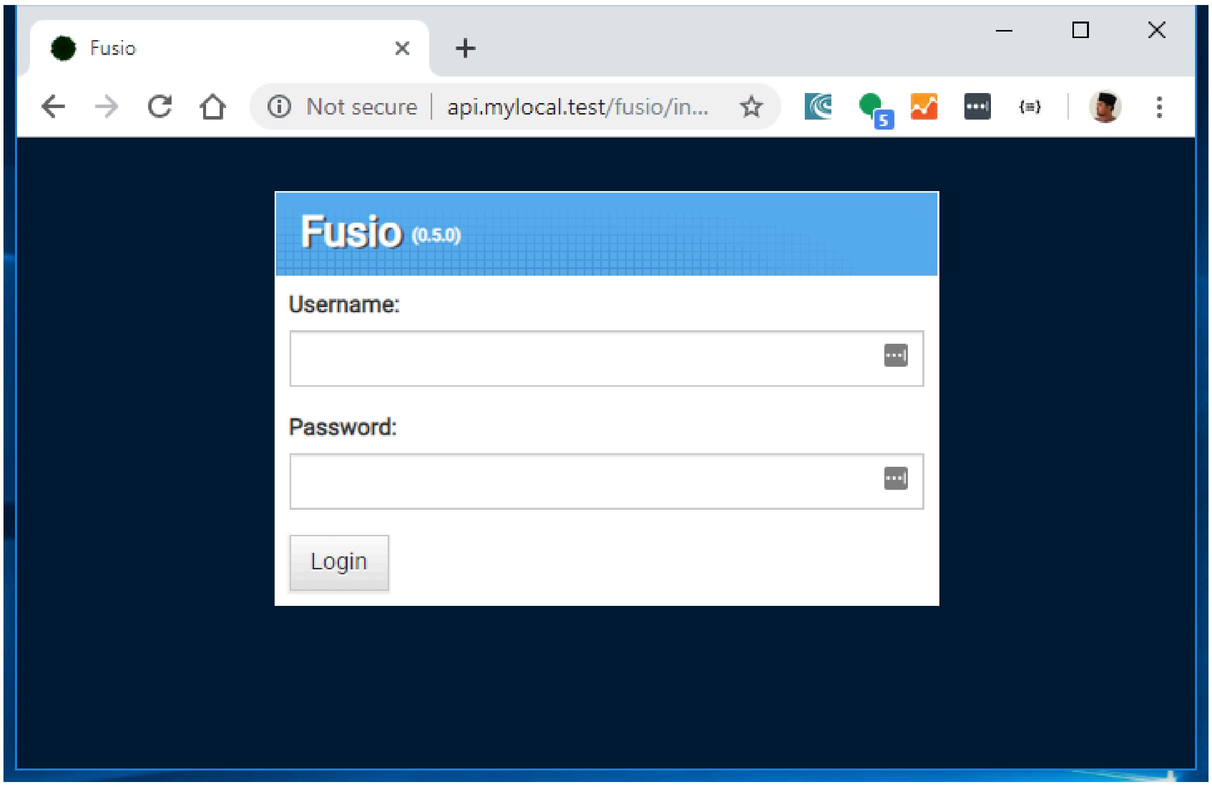
- Enter **1** to create an Administrator user.
- For the sake of this tutorial, enter **admin** as username
- For the sake of this tutorial, enter **admin@admin.com** as email
- For the sake of this tutorial, enter **admin1234** as password

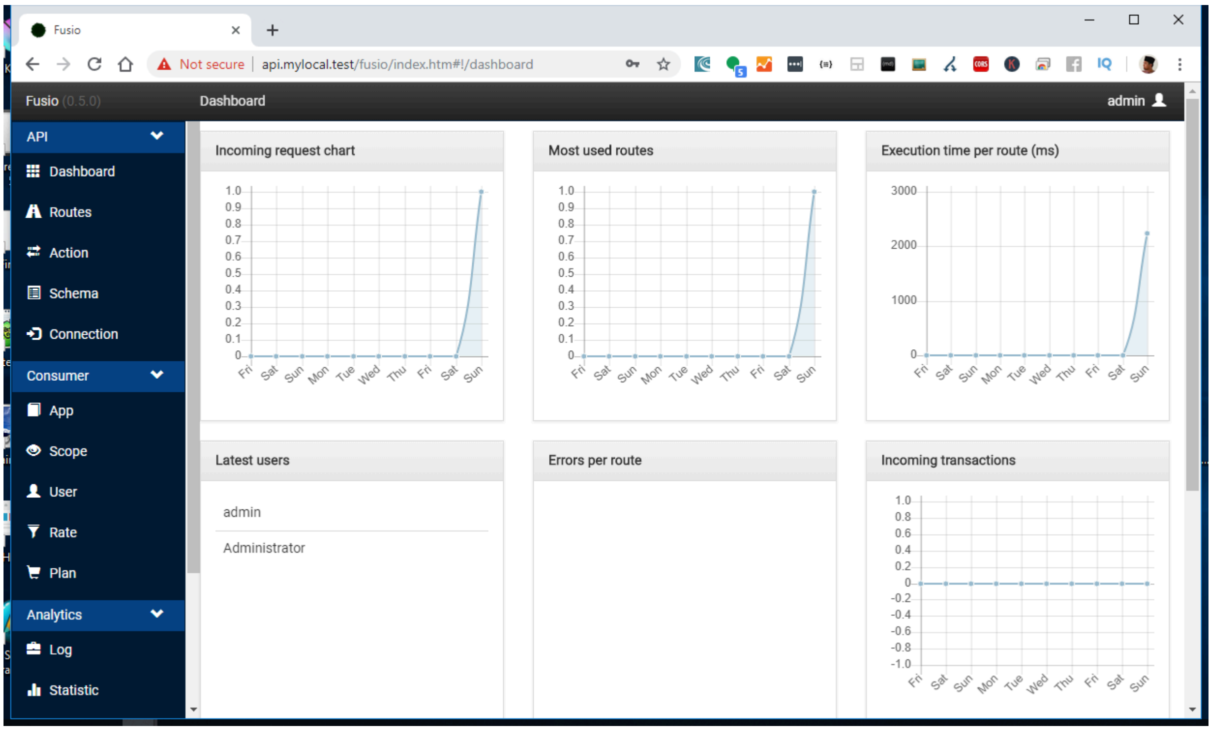


```
D:\laragon\www\api
λ php bin/fusio adduser
Choose the status of the account [0=Consumer, 1=Administrator]: 1
Enter the username: admin
Enter the email: admin@admin.com
Enter the password:
Repeat the password:
Created user admin successful

D:\laragon\www\api
λ
```

Launch your browser and enter the address **http://api.mylocal.test/fusio/index.htm**. You should see a login screen.

Enter admin as username and admin1234 as password, as created with the previous step.

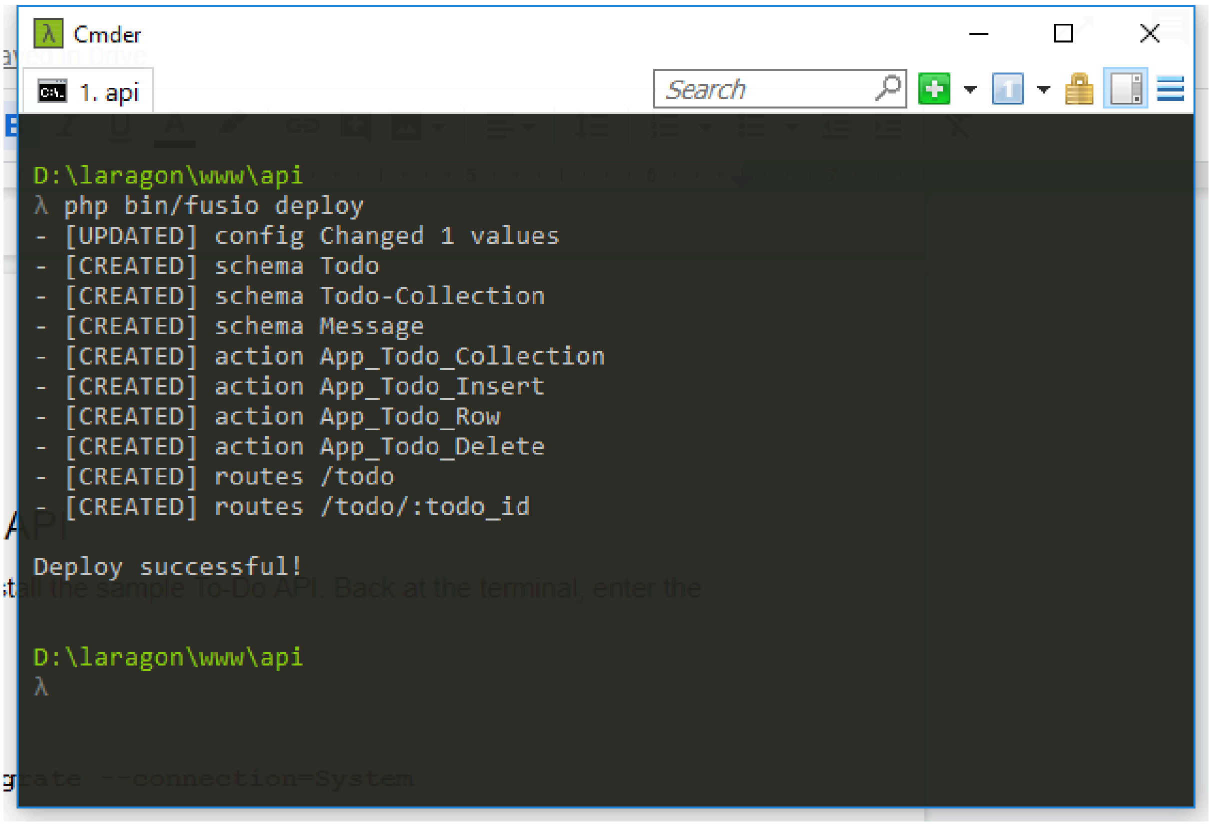




Installing Sample To-Do API

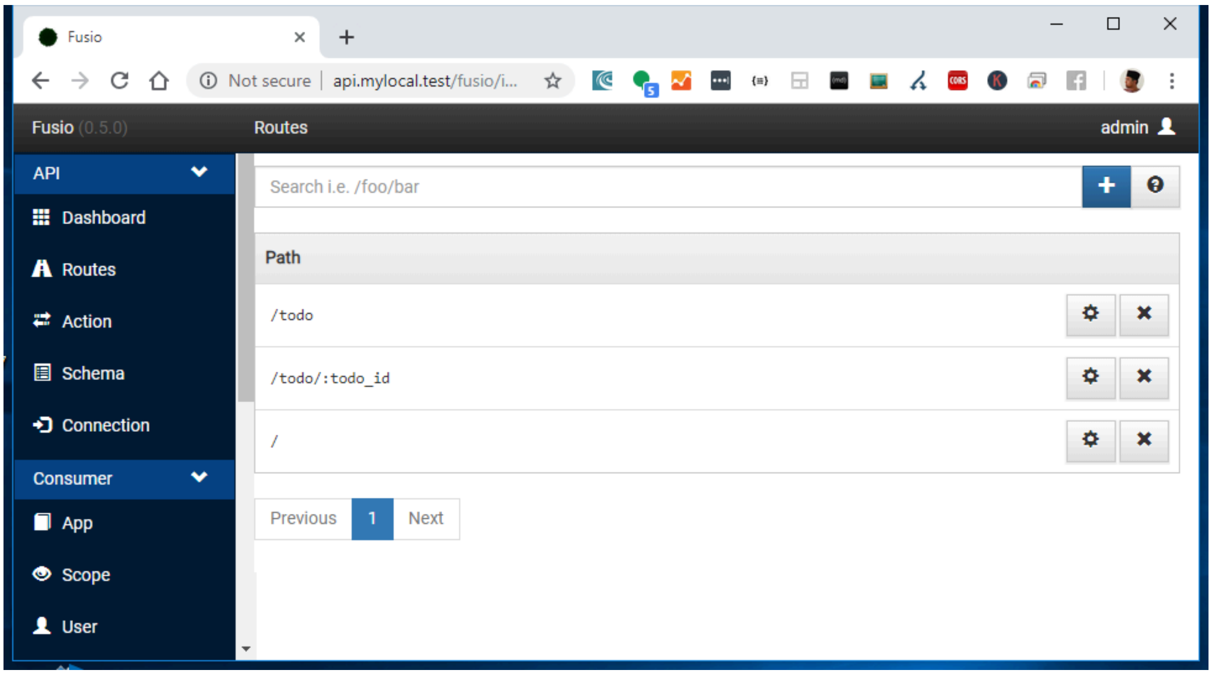
To continue with the lesson, let’s install the sample To-Do API. Back at the terminal, enter the following command.

```
php bin/fusio deploy
```



```
D:\laragon\www\api
λ php bin/fusio deploy
- [UPDATED] config Changed 1 values
- [CREATED] schema Todo
- [CREATED] schema Todo-Collection
- [CREATED] schema Message
- [CREATED] action App_Todo_Collection
- [CREATED] action App_Todo_Insert
- [CREATED] action App_Todo_Row
- [CREATED] action App_Todo_Delete
- [CREATED] routes /todo
- [CREATED] routes /todo/:todo_id

Deploy successful!


D:\laragon\www\api
λ
```

This will install some new Routes and API. We can verify this by clicking the **Routes** menu from Fusio Administration Panel.

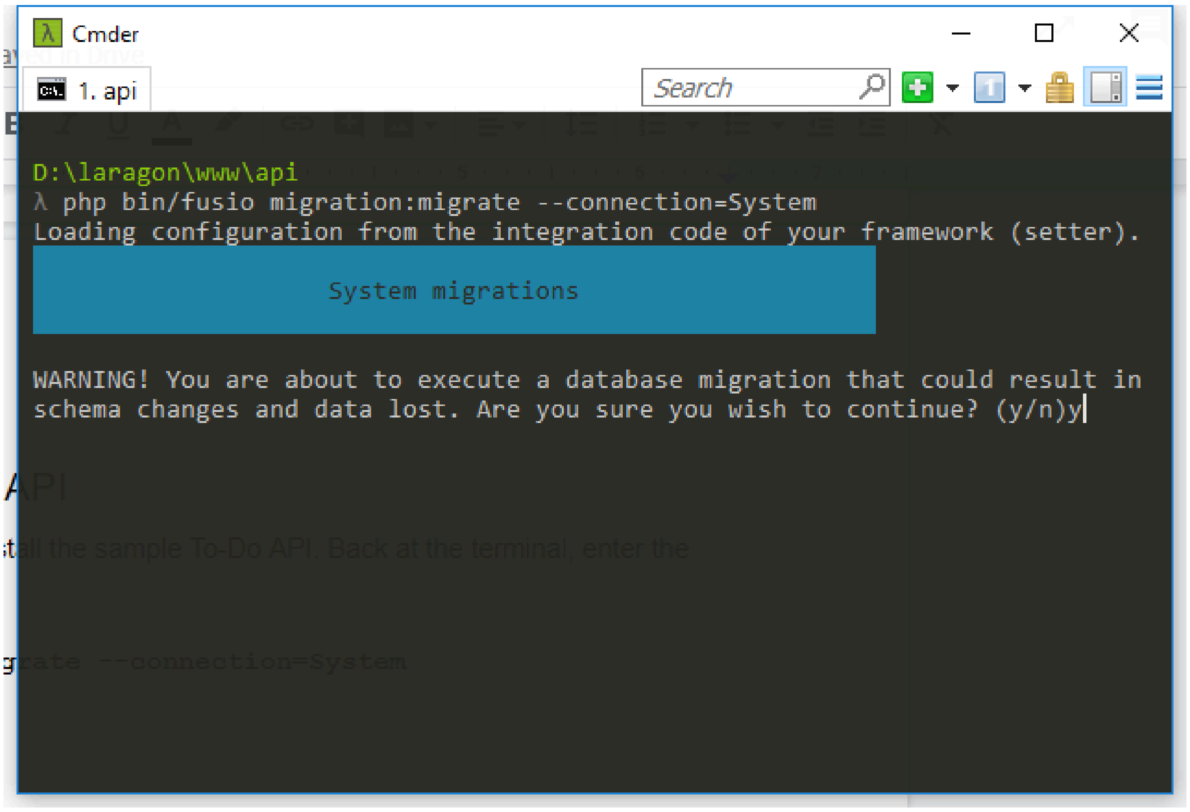


Path
/todo
/todo/:todo_id
/

However, we have not completed the process yet. We still have to create the database for the sample To-Do API. Return back to the terminal and enter the following command.

```
php bin/fusio migration:migrate --connection=System
```

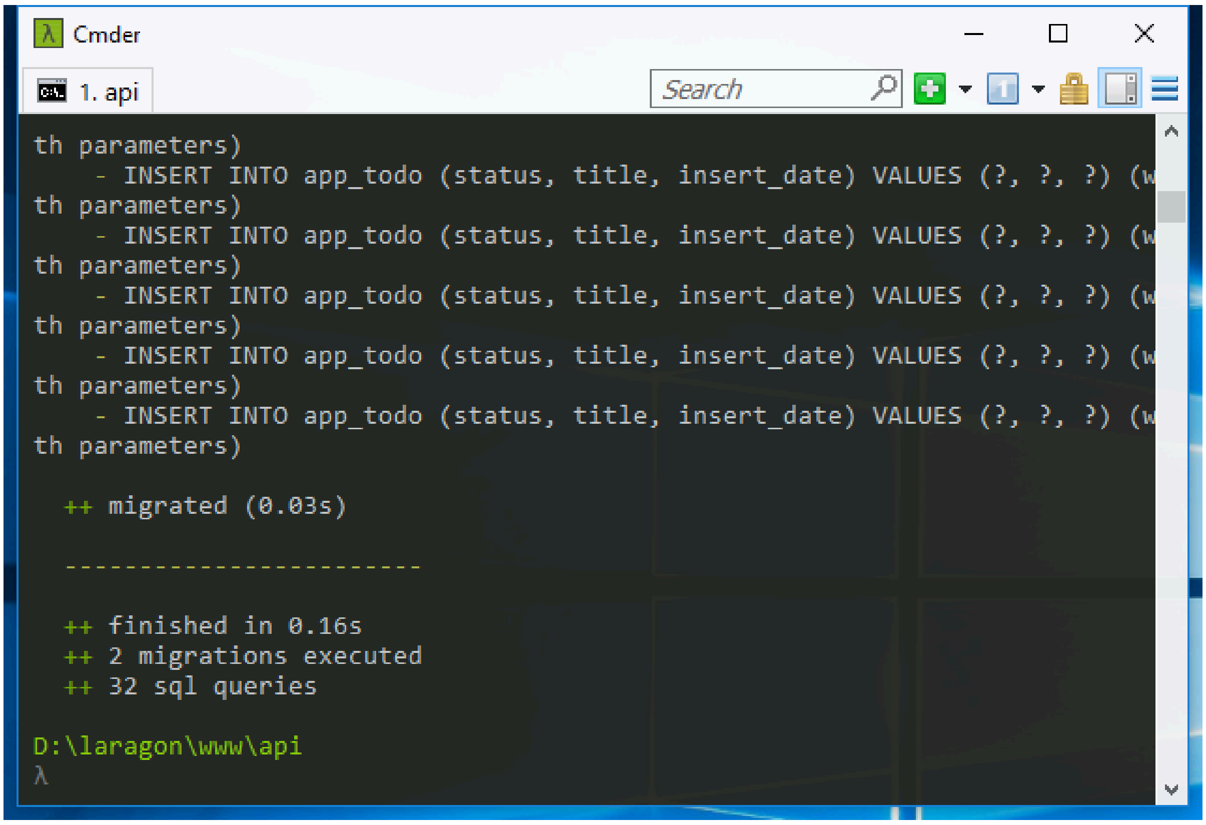
When prompted, enter **y**.



Cmder

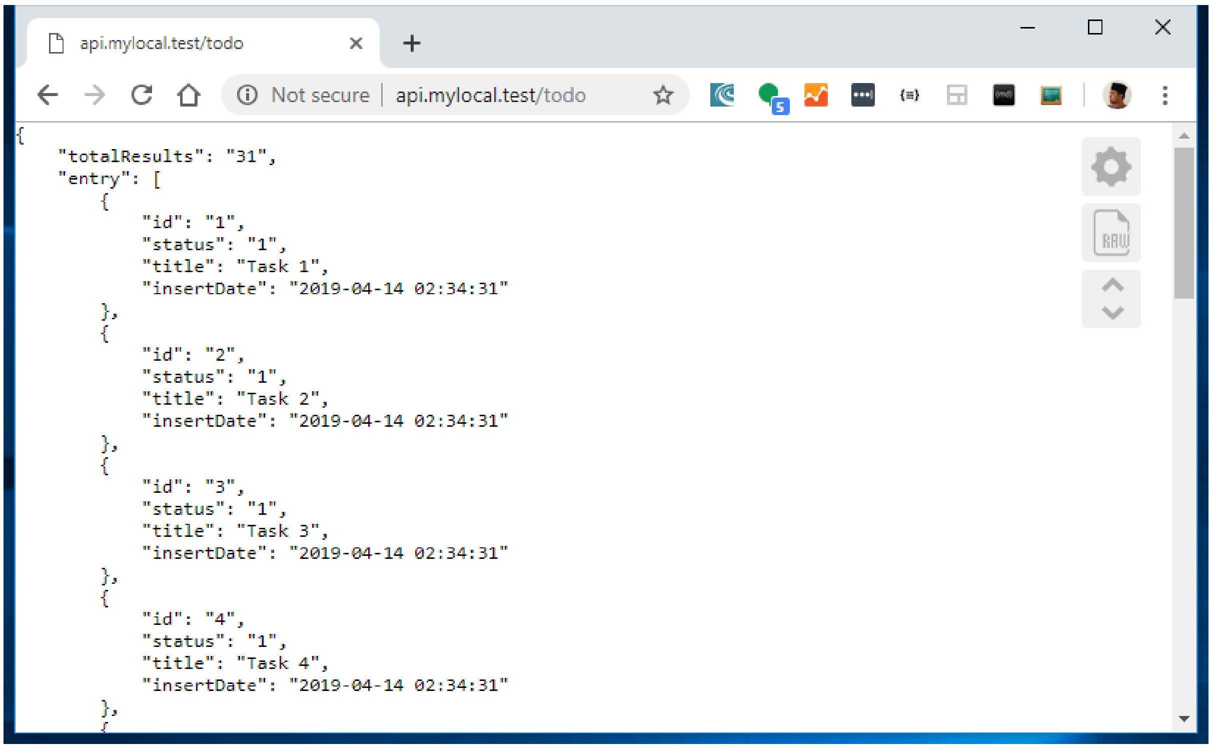
1. api

```
D:\laragon\www\api
λ php bin/fusio migration:migrate --connection=System
Loading configuration from the integration code of your framework (setter).

                    System migrations

WARNING! You are about to execute a database migration that could result in
schema changes and data lost. Are you sure you wish to continue? (y/n)y
```



```
th parameters)
    - INSERT INTO app_todo (status, title, insert_date) VALUES (?, ?, ?) (w
th parameters)
    - INSERT INTO app_todo (status, title, insert_date) VALUES (?, ?, ?) (w
th parameters)
    - INSERT INTO app_todo (status, title, insert_date) VALUES (?, ?, ?) (w
th parameters)
    - INSERT INTO app_todo (status, title, insert_date) VALUES (?, ?, ?) (w
th parameters)
    - INSERT INTO app_todo (status, title, insert_date) VALUES (?, ?, ?) (w
th parameters)

  ++ migrated (0.03s)

  ------------------------

  ++ finished in 0.16s
  ++ 2 migrations executed
  ++ 32 sql queries

D:\laragon\www\api
λ
```

By now we should be able to try out the To-Do API. Enter **http://api.mylocal.test/todo** in your browser and see the result.



```
{
    "totalResults": "31",
    "entry": [
        {
            "id": "1",
            "status": "1",
            "title": "Task 1",
            "insertDate": "2019-04-14 02:34:31"
        },
        {
            "id": "2",
            "status": "1",
            "title": "Task 2",
            "insertDate": "2019-04-14 02:34:31"
        },
        {
            "id": "3",
            "status": "1",
            "title": "Task 3",
            "insertDate": "2019-04-14 02:34:31"
        },
        {
            "id": "4",
            "status": "1",
            "title": "Task 4",
            "insertDate": "2019-04-14 02:34:31"
        },
        {
```

This is the result returned by the API, which is a set of JSON data.

Summary

In this tutorial, you have just completed and learned the following:

- Creating a virtual host with a local domain address on Laragon
- Downloaded and installed Fusio
- Created an administrator user and logged into Fusio Administration Panel
- Deployed the sample To-Do API
- Accessed some API endpoints on Fusio

Fusio Sample To-Do API

Fusio comes with a sample To-Do API to demonstrate how an API endpoint is built with Fusio.

To proceed, we are making the assumption that you have Fusio installed properly and you are able to login to your Fusio administration panel.

Before you checkout the API, make sure that you deployed the API using the following command on our console.

```
php bin/fusio deploy
```

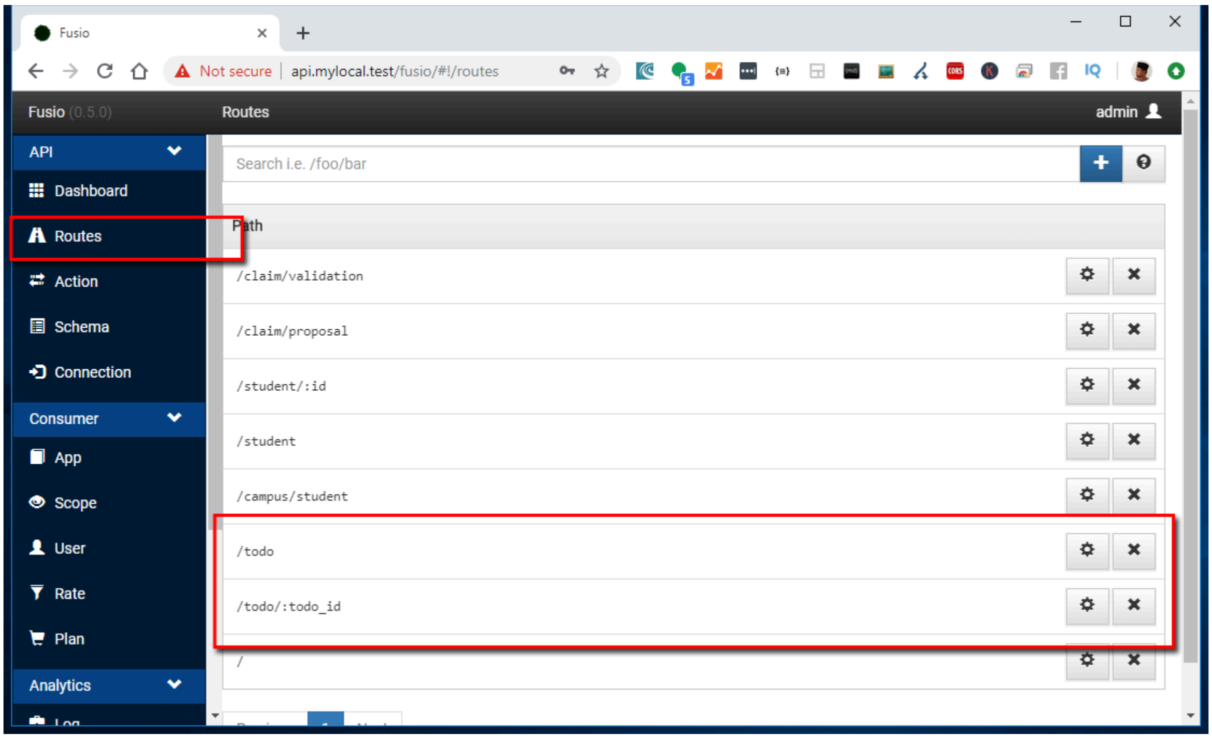
```
php bin/fusio migration:migrate --connection=System
```

The following section, we will be looking into understanding Routes

Routes

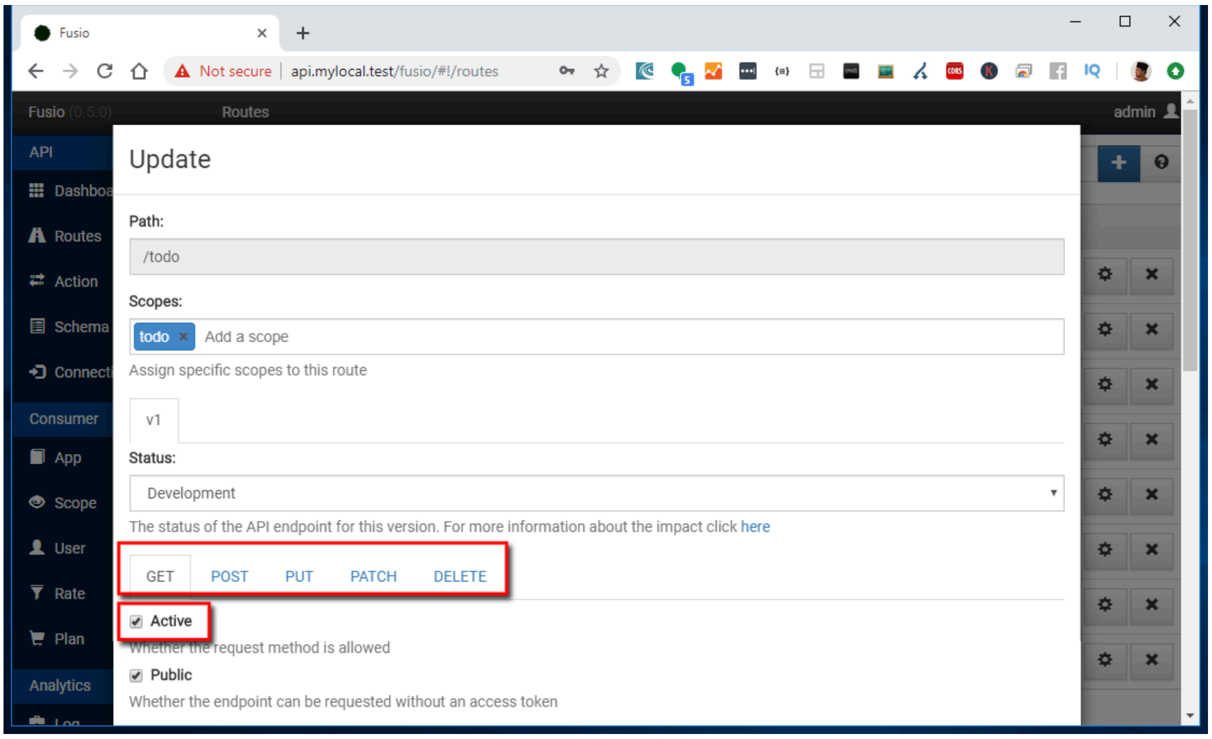
Routes determine the URL or address for your API.

If you deployed your To-Do Sample API, login to your Fusio administration panel and click on Routes from the left menu.



```
/claim/validation
/claim/proposal
/student/:id
/student
/campus/student
/todo
/todo/:todo_id
/
```

You should be able to see the two routes that starts with /todo. Even though you see two routes, there are actually four APIs within these two routes. Let's click on the "gear" ⚙ icon on the right to see why.



Update

Path:
/todo

Scopes:
todo × Add a scope
Assign specific scopes to this route

v1

Status:
Development
The status of the API endpoint for this version. For more information about the impact click here

GET POST PUT PATCH DELETE

☑ Active
Whether the request method is allowed
☑ Public
Whether the endpoint can be requested without an access token

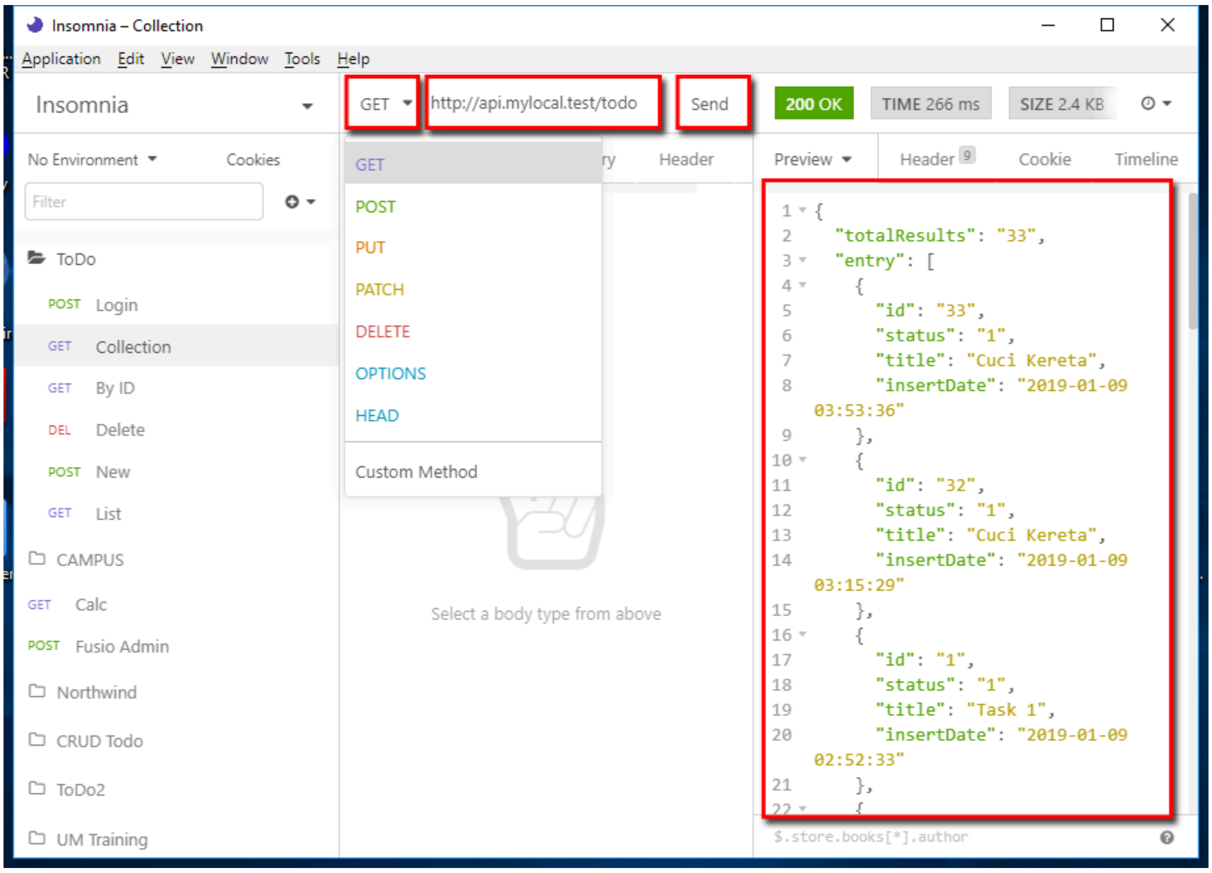
Here you will see that each route gives you the option of 5 HTTP methods to work with, which are GET, POST, PUT, PATCH and DELETE. If you check out the tabs one by one, you will see if they are active or not.

If you check out both routes /todo and /todo/:todo_id, you will see that you actually have four APIs which are:

1. **/todo - GET**
2. /todo - POST
3. /todo/:todo_id - GET
4. /todo/:todo_id - DELETE

Let's Try Out the API

When testing APIs, normally we use an API client software like Postman or Insomnia. This tutorial will use Insomnia. You can download Insomnia for free from https://insomnia.rest/



```
{
  "totalResults": "33",
  "entry": [
    {
      "id": "33",
      "status": "1",
      "title": "Cuci Kereta",
      "insertDate": "2019-01-09 03:53:36"
    },
    {
      "id": "32",
      "status": "1",
      "title": "Cuci Kereta",
      "insertDate": "2019-01-09 03:15:29"
    },
    {
      "id": "1",
      "status": "1",
      "title": "Task 1",
      "insertDate": "2019-01-09 02:52:33"
    },
    {
```

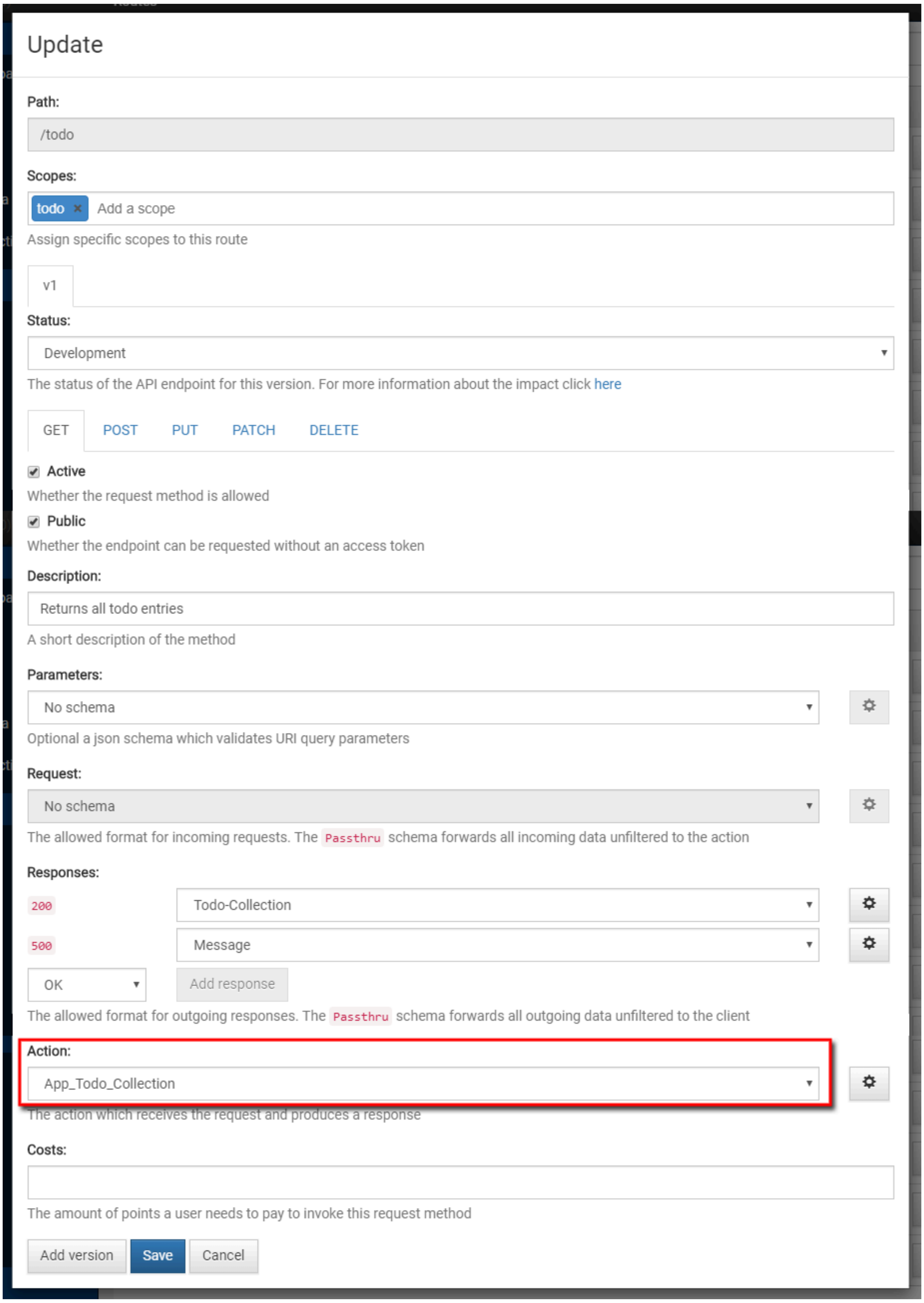
Like in the image above, since we are testing the first API (GET - todo), make sure you select the GET method and enter the URL correctly on the address area. If you are ready, click **Send**. The right is the response that you get from the API. As you can see here, you get a JSON data with a list of to-dos.

Now you know what this API endpoint does. It returns a list of to-do items, You can try out the other APIs. However the rest of the tutorial will focus on the GET - /todo API.

Actions

We will look at the Routes from the Fusio administration panel. Click on the “gear” icon for /todo again and look into the GET tab.

Let’s ignore the rests and look into Action for now.



Update

Path: /todo

Scopes: todo × Add a scope
Assign specific scopes to this route

v1

Status: Development
The status of the API endpoint for this version. For more information about the impact click here

☑ Active
Whether the request method is allowed

☑ Public
Whether the endpoint can be requested without an access token

Description: Returns all todo entries
A short description of the method

Parameters: No schema
Optional a json schema which validates URI query parameters

Request: No schema
The allowed format for incoming requests. The `Passthru` schema forwards all incoming data unfiltered to the action

Responses:

200	Todo-Collection
500	Message

The allowed format for outgoing responses. The `Passthru` schema forwards all outgoing data unfiltered to the client

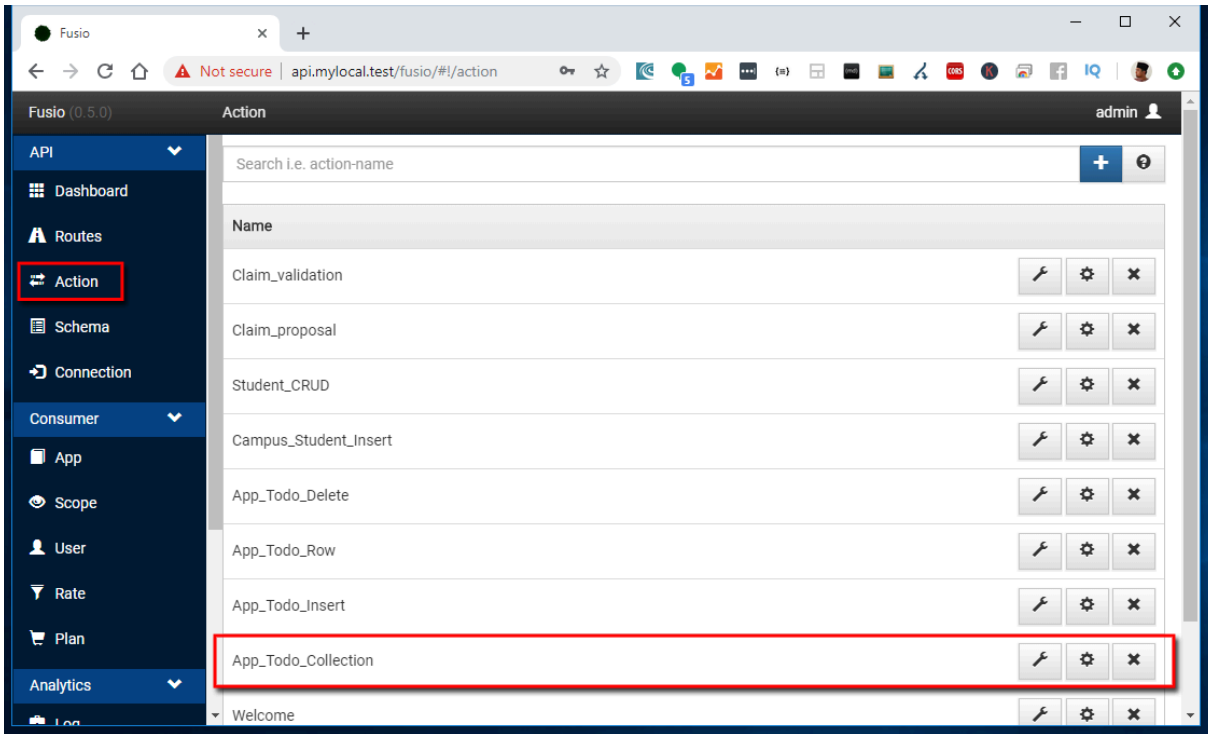
Action: App_Todo_Collection
The action which receives the request and produces a response

Costs:
The amount of points a user needs to pay to invoke this request method

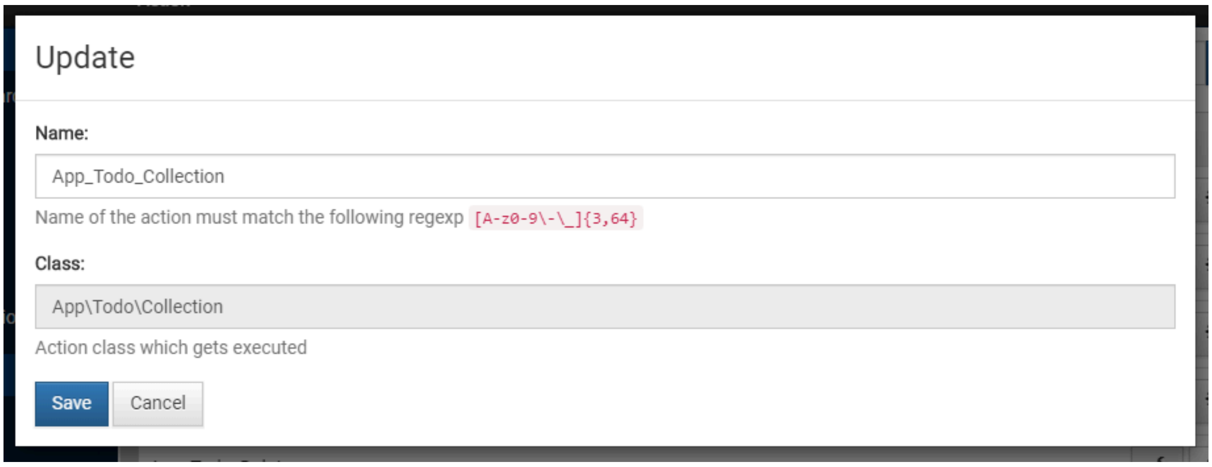
For this /todo with the GET method, it says that it is using the App_Todo_Collection.

With Fusio, when creating an API endpoint, you need to define your Route. And in your Route setting, you need to define your Action.

An Action is the component that will handle the API request. Let's go into Action. Click on Action from the left menu. Here, you should be able to find the App_Todo_Collection action, just like you saw it from the Route settings.



Click on the "gear" icon for this action. You will be able to see the following settings dialog box.


Update

Name:

App_Todo_Collection

Name of the action must match the following regexp `[A-z0-9\-\_]{3,64}`

Class:

App\Todo\Collection

Action class which gets executed

When creating an Action, there are eight ways you can go about doing it (Class). They are:

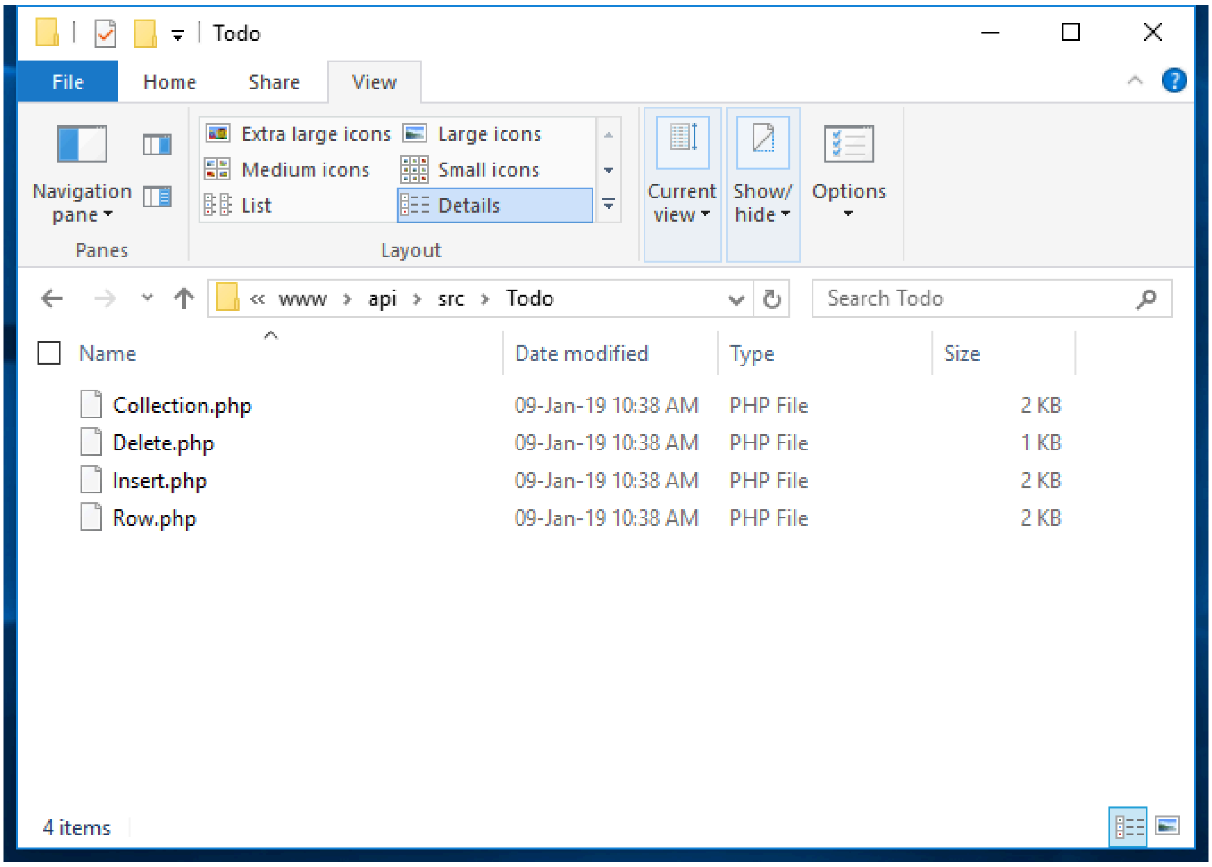
1. File Processor (for sending content of static file)

2. HTTP Processor (fetching and passing content from an external URL)
3. PHP Processor (running a PHP script)
4. PHP Sandbox (Fusio has a built in editor for PHP scripting)
5. SQL Table (Fusio built-in feature for easy CRUD operations on a table)
6. Util Static Response (where you can enter a static JSON response)
7. V8 Processor (running a Javascript)
8. PHP Class Name

The To-Do sample API uses the PHP Class Name method for creating the handler for the Action. You can see that the class defined here is **App\Todo\Collection**.

When creating a PHP Class for your Action, it stored in your Fusio installation folder under the /src folder.

It is common practice to create a specific folder to group your API Action Classes. In the case of the To-Do Sample API, all the classes are in the /src/Todo folder. Here you will find the Collection.php file.



Name	Date modified	Type	Size
Collection.php	09-Jan-19 10:38 AM	PHP File	2 KB
Delete.php	09-Jan-19 10:38 AM	PHP File	1 KB
Insert.php	09-Jan-19 10:38 AM	PHP File	2 KB
Row.php	09-Jan-19 10:38 AM	PHP File	2 KB

Let's open the Collection.php file and see the content.

PHP Class Name file (for your Action)

Below is the content of Collection.php

```php
<?php

namespace App\Todo;

use Fusio\Engine\ActionAbstract;
use Fusio\Engine\ContextInterface;
use Fusio\Engine\ParametersInterface;
use Fusio\Engine\RequestInterface;

class Collection extends ActionAbstract
{
    public function handle(RequestInterface $request,
ParametersInterface $configuration, ContextInterface $context)
    {
        /** @var \Doctrine\DBAL\Connection $connection */
        $connection = $this->connector->getConnection('System');

        $sql = 'SELECT id,
                       status,
                       title,
                       insert_date AS insertDate
                  FROM app_todo
                 WHERE status = 1
              ORDER BY insert_date DESC';

        $sql =
$connection->getDatabasePlatform()->modifyLimitQuery($sql, 16);

        $count   = $connection->fetchColumn('SELECT COUNT(*) FROM
app_todo');
        $entries = $connection->fetchAll($sql);

        return $this->response->build(200, [], [
            'totalResults' => $count,
            'entry' => $entries,
        ]);
    }
}
```

What is important to note here:

- `namespace App\Todo;`

When creating your own API later, you need to give it a namespace. Also make sure that you create a folder with the same name.

-
  ```
  /** @var \Doctrine\DBAL\Connection $connection */
  $connection = $this->connector->getConnection('System');
  ```

  First, take note that Fusio provides you with the Doctrine DBAL library for database operations. More information about Doctrine DBAL is available at https://www.doctrine-project.org/projects/doctrine-dbal/en/2.9/index.html

  Second, this script is using the **System** database connection. This is the database connection as defined in your .env file. When creating your own API, you can add more database connection from Fusio administration panel, and give it a unique name. You can later use them in your PHP Name Class.

-
  ```
  class Collection extends ActionAbstract
  {
      public function handle(RequestInterface $request,
  ParametersInterface $configuration, ContextInterface $context)
      { ...
      }
  }
  ```

  Your class extends ActionAbstract class.

  You must have the `handle` function to handle the API request.

  Fusio gives you access to $request, $configuration and $context objects, which provides to other information and methods. More information about it is available at https://www.fusio-project.org/documentation/php

-
  ```
  return $this->response->build(200, [], [
      'totalResults' => $count,
      'entry' => $entries,
  ]);
  ```

  This is how you send data back. The `build` function receives three parameters

  1. HTTP code
  2. Extra HTTP headers
  3. PHP array for the data to be sent back, which will be sent as JSON

Summary

If you follow this tutorial closely, you will understand how Fusio works, and how you can create your own API using Fusio. As a quick summary, below are the steps when creating your API in Fusio.

1. Create your PHP class and store it in /src/<namespace>
2. Define your Action in Fusio administration panel and specify where your PHP class is

3. Define your Route
    a. Define your Path
    b. Choose your method and make sure you check Active
    c. Select your Action

Exercise

Go through the other API endpoints for the To-Do sample API and see what you can find. See if you can find out where the PHP classes are and try to understand what each line of codes do.

Creating Your First API on Fusio

By now you should have installed Fusio, deployed the sample To-Do API and tested some API endpoints on your Fusio installation.

This tutorial will show you how you can create your first API on Fusio.

Learning from the To-Do Sample API in Fusio, the know that:

- Database is used in **Connections**
- **Connections** is used in your **PHP Class** script
- **PHP Class** is used in **Actions**
- **Action** is used in **Routes**

Following this sequence, we will be creating this one by one.

Database → Connection → PHP Class → Action → Route

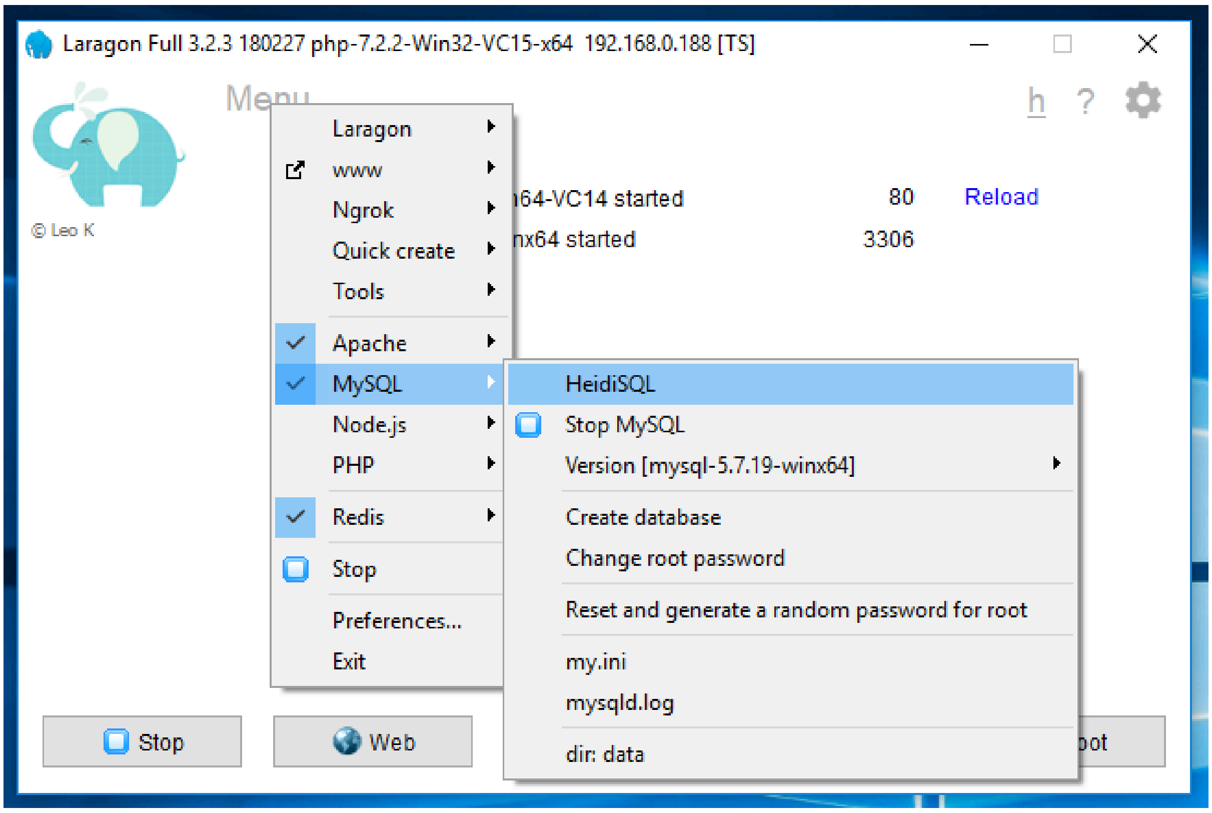
Once we completed this step, we will have a working API.

For this tutorial, our goal is to create the following API.

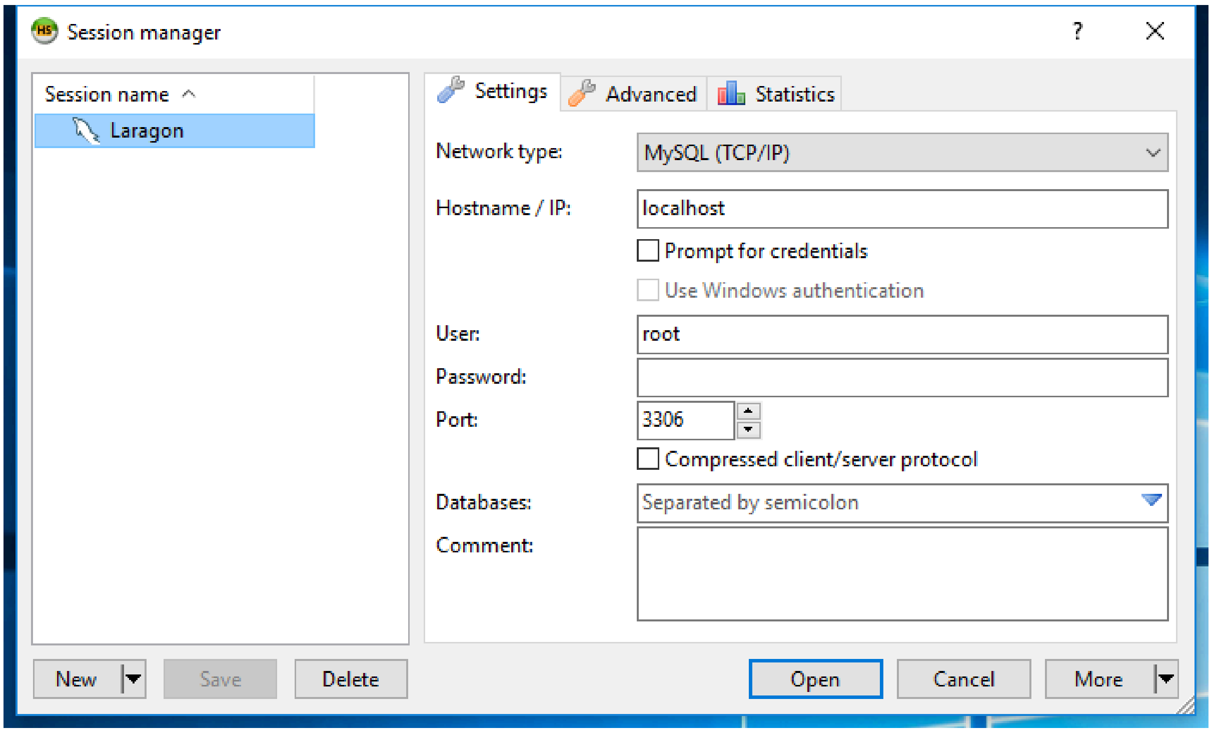
Path	/school/teacher
Method	GET
Result	Returns a JSON formatted data that lists the teachers from the database

Creating your Database

Laragon comes equipped with HeidiSQL, which is a database client for database servers like MySQL. Launch your Laragon Control Panel, right-click and select MySQL > HeidSQL.

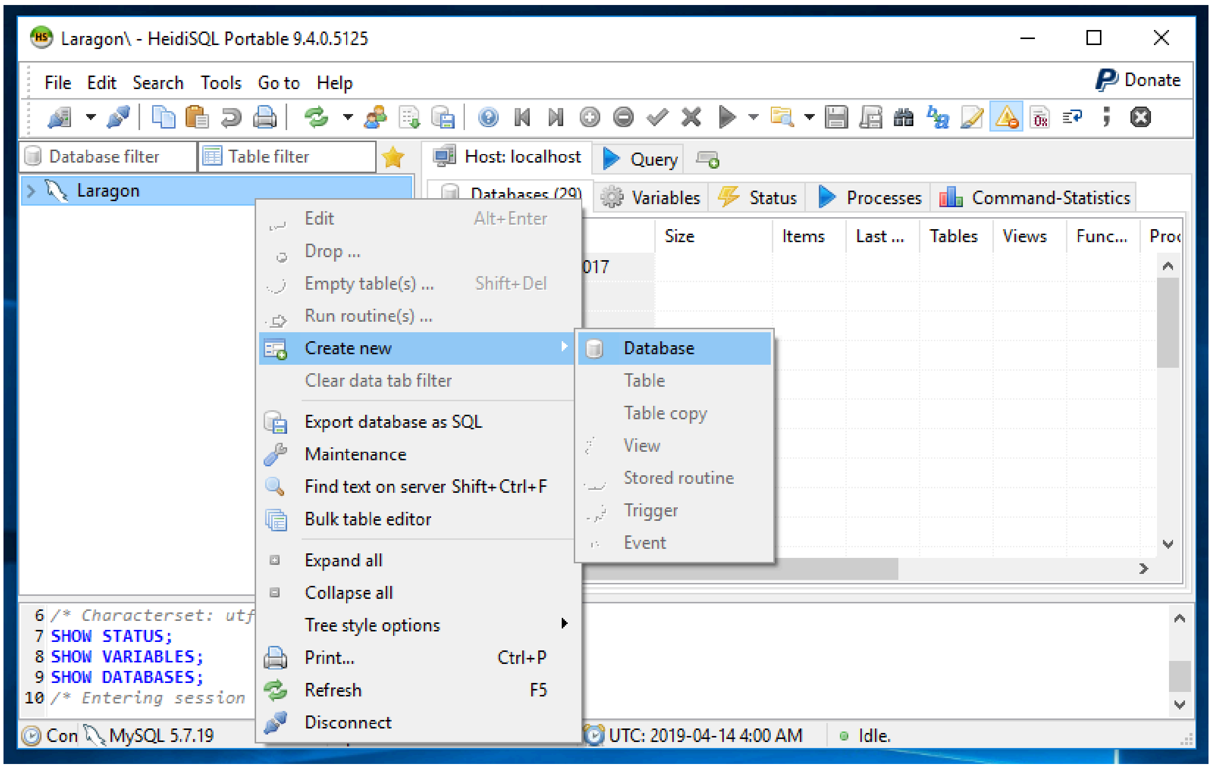


Select Laragon from the Session list and click Open.



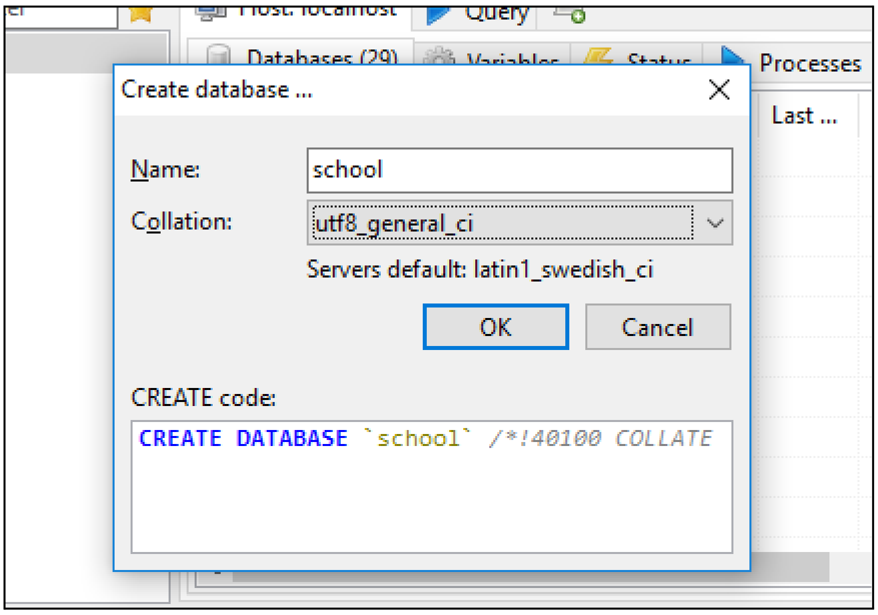
Network type: MySQL (TCP/IP)
Hostname / IP: localhost
Prompt for credentials
Use Windows authentication
User: root
Password:
Port: 3306
Compressed client/server protocol
Databases: Separated by semicolon
Comment:

We will start by creating an empty database. Right-click on the Laragon session and select Create New > Database.



```
/* Characterset: utf
SHOW STATUS;
SHOW VARIABLES;
SHOW DATABASES;
/* Entering session
```

Enter **school** for database name. Select **utf8_general_ci** as collation.



Name: school

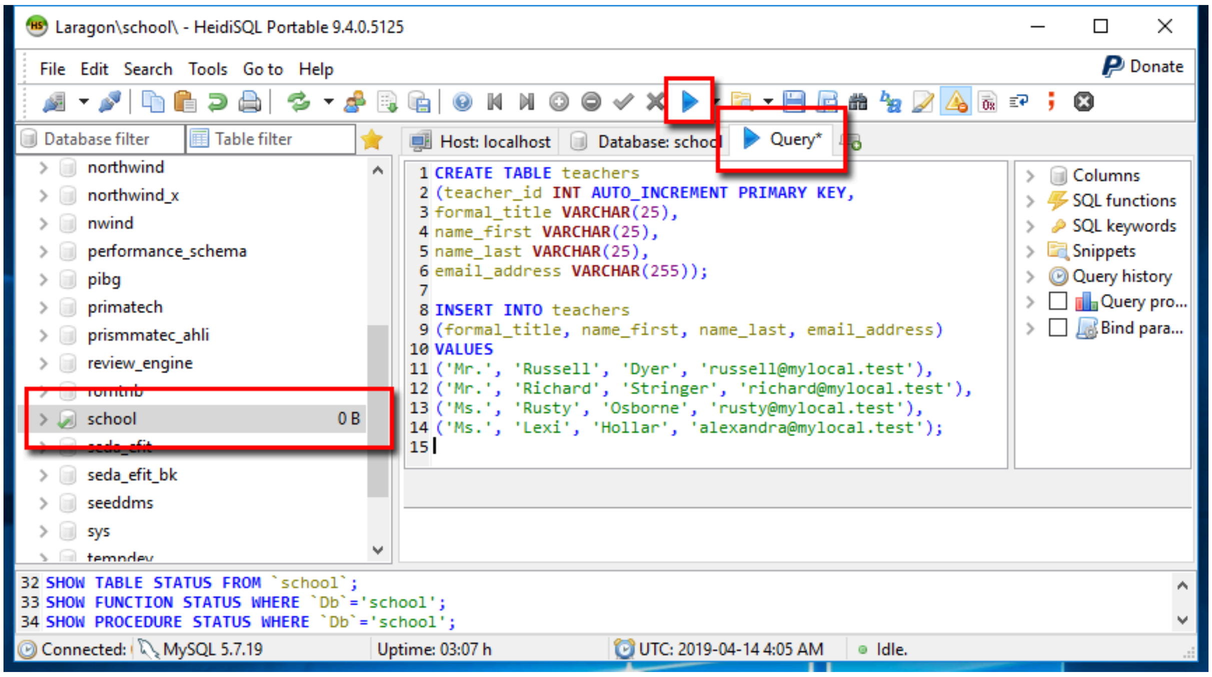
Collation: utf8_general_ci

Servers default: latin1_swedish_ci

CREATE code:

```
CREATE DATABASE `school` /*!40100 COLLATE
```

Now make sure you have the **school** database selected from the left panel. Then click on the Query tab.



```
CREATE TABLE teachers
(teacher_id INT AUTO_INCREMENT PRIMARY KEY,
formal_title VARCHAR(25),
name_first VARCHAR(25),
name_last VARCHAR(25),
email_address VARCHAR(255));

INSERT INTO teachers
(formal_title, name_first, name_last, email_address)
VALUES
('Mr.', 'Russell', 'Dyer', 'russell@mylocal.test'),
('Mr.', 'Richard', 'Stringer', 'richard@mylocal.test'),
('Ms.', 'Rusty', 'Osborne', 'rusty@mylocal.test'),
('Ms.', 'Lexi', 'Hollar', 'alexandra@mylocal.test');
```

```
SHOW TABLE STATUS FROM `school`;
SHOW FUNCTION STATUS WHERE `Db`='school';
SHOW PROCEDURE STATUS WHERE `Db`='school';
```

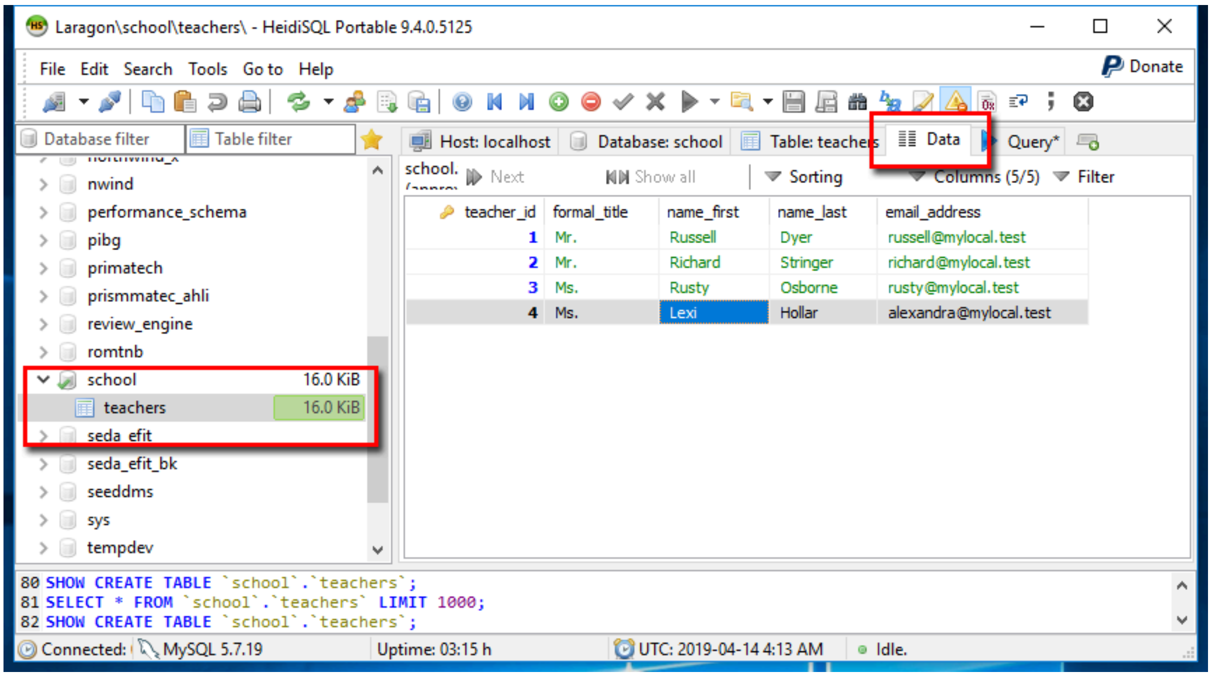
We will run some SQL commands to create some data in the school database.

Enter the following SQL command into the Query area. And then click on the button with the blue triangle icon to execute the SQL commands.

This will create one table, teachers, and insert four records into the table.

```
CREATE TABLE teachers
(teacher_id INT AUTO_INCREMENT PRIMARY KEY,
formal_title VARCHAR(25),
name_first VARCHAR(100),
name_last VARCHAR(100),
email_address VARCHAR(255));

INSERT INTO teachers
(formal_title, name_first, name_last, email_address)
VALUES
('Mr.', 'Russell', 'Dyer', 'russell@mylocal.test'),
('Mr.', 'Richard', 'Stringer', 'richard@mylocal.test'),
('Ms.', 'Rusty', 'Osborne', 'rusty@mylocal.test'),
('Ms.', 'Lexi', 'Hollar', 'alexandra@mylocal.test');
```

You can hit the F5 button to reload HeidiSQL. Then from the left panel, expand the **school** database and select the **teachers** table. On the right panel, find and click on the Data tab panel. You should be able to see the **teachers** table with its data.



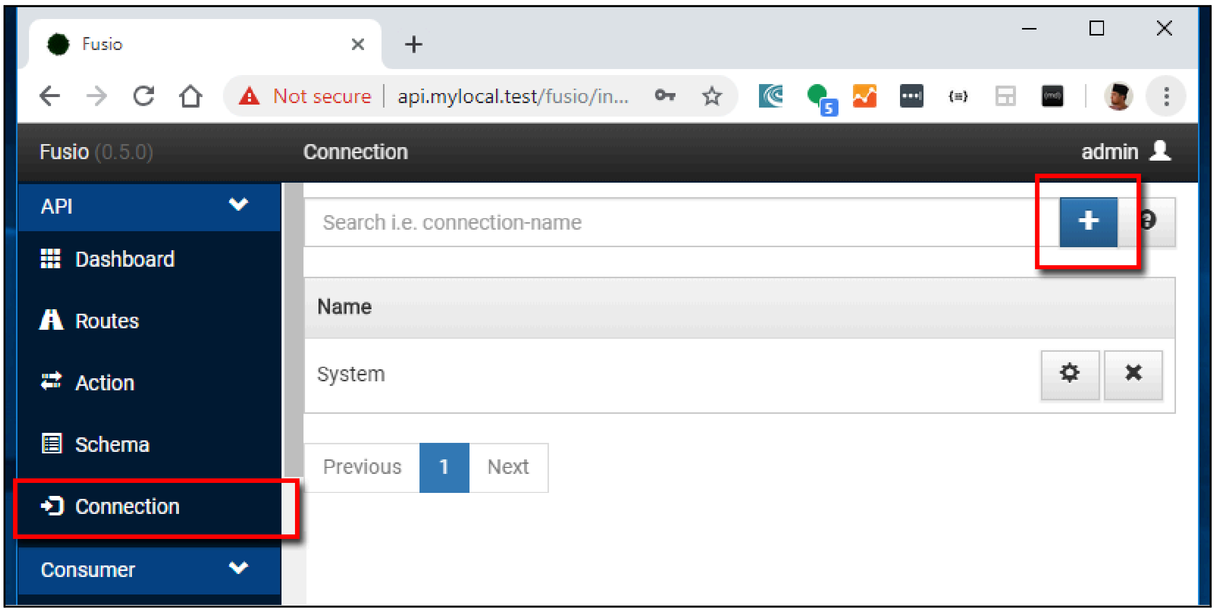
teacher_id	formal_title	name_first	name_last	email_address
1	Mr.	Russell	Dyer	russell@mylocal.test
2	Mr.	Richard	Stringer	richard@mylocal.test
3	Ms.	Rusty	Osborne	rusty@mylocal.test
4	Ms.	Lexi	Hollar	alexandra@mylocal.test

```
SHOW CREATE TABLE `school`.`teachers`;
SELECT * FROM `school`.`teachers` LIMIT 1000;
SHOW CREATE TABLE `school`.`teachers`;
```

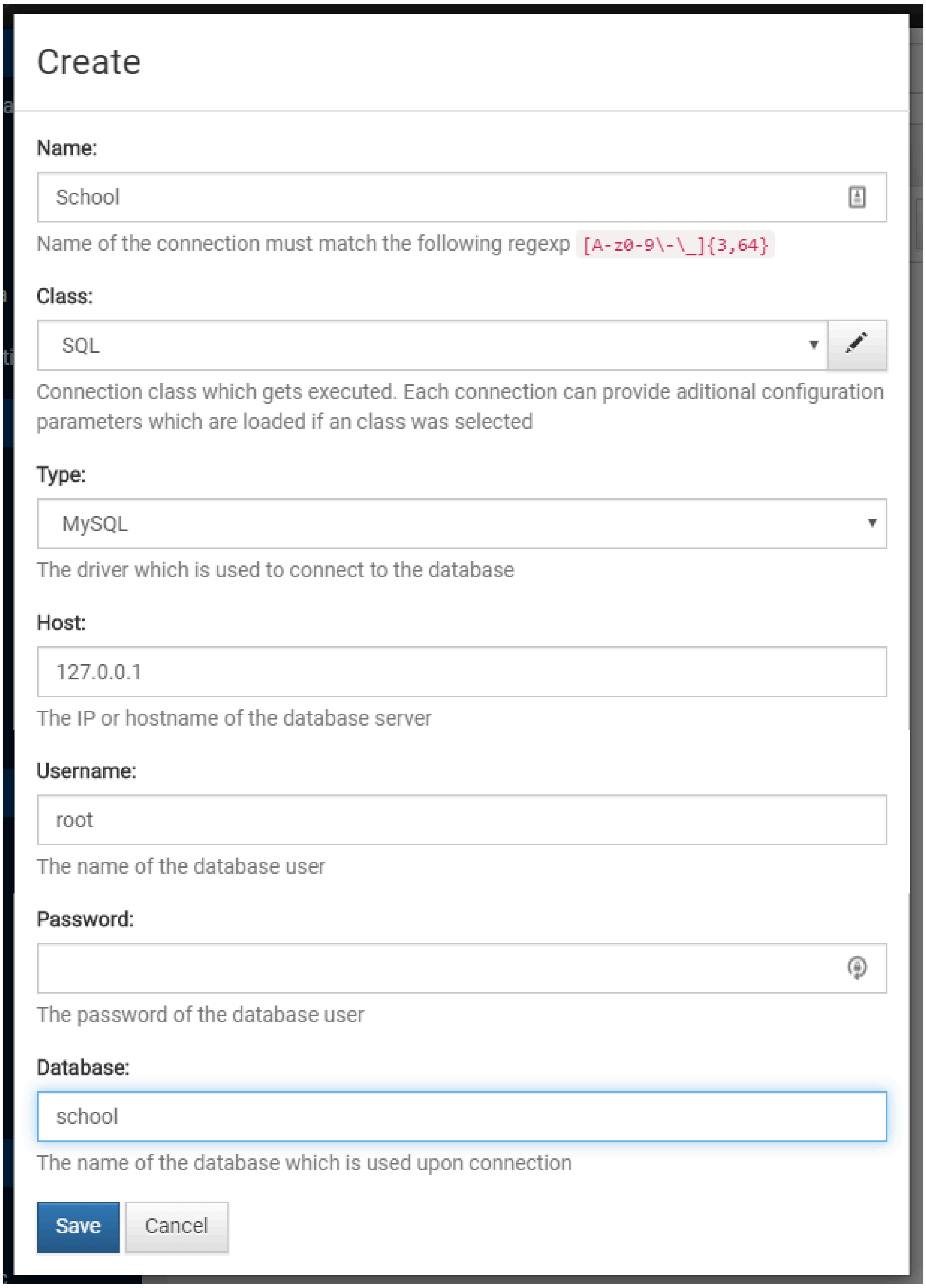
Connected: MySQL 5.7.19 Uptime: 03:15 h UTC: 2019-04-14 4:13 AM Idle.

Creating a Database Connection

We will now create a Database connection using Fusio. Proceed to your Fusio Administration Panel. Click on the Connection from the left menu. Click on the blue Add button with the plus (+) icon.



We will be creating a Connection instance to the school database that we created earlier. Enter your **Connection** details as below. Click **Save**.



Create

Name:

School

Name of the connection must match the following regexp `[A-z0-9\-\_]{3,64}`

Class:

SQL

Connection class which gets executed. Each connection can provide aditional configuration parameters which are loaded if an class was selected

Type:

MySQL

The driver which is used to connect to the database

Host:

127.0.0.1

The IP or hostname of the database server

Username:

root

The name of the database user

Password:

The password of the database user

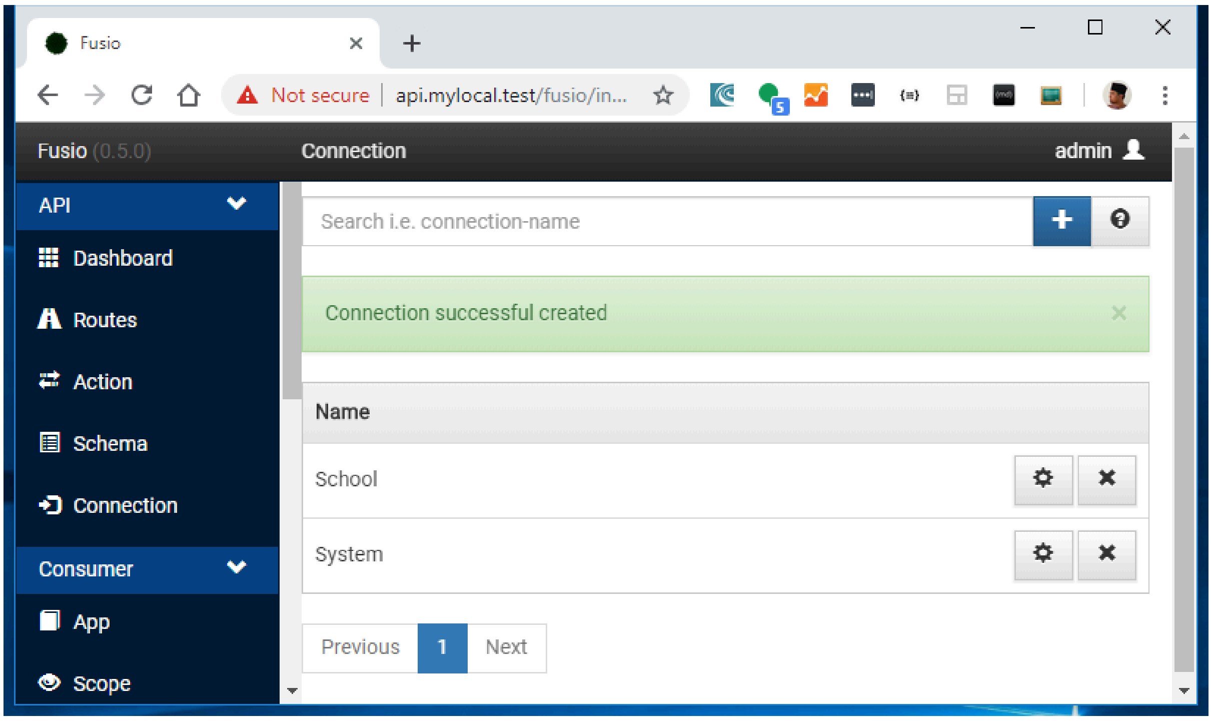
Database:

school

The name of the database which is used upon connection

Name	School
Class (select)	SQL
Type (select)	MySQL
Host	127.0.0.1
Username	root
Password	(leave blank - default root password)
Database	school

When done, you should be able to see your Connection listed.



Connection successful created

Name
School
System

Creating the PHP Class

We will take the example from the Collection.php from To-Do API which is available in src/Todo/Collection.php. Create the following file and save it as src/School/Teacher/Collection.php.

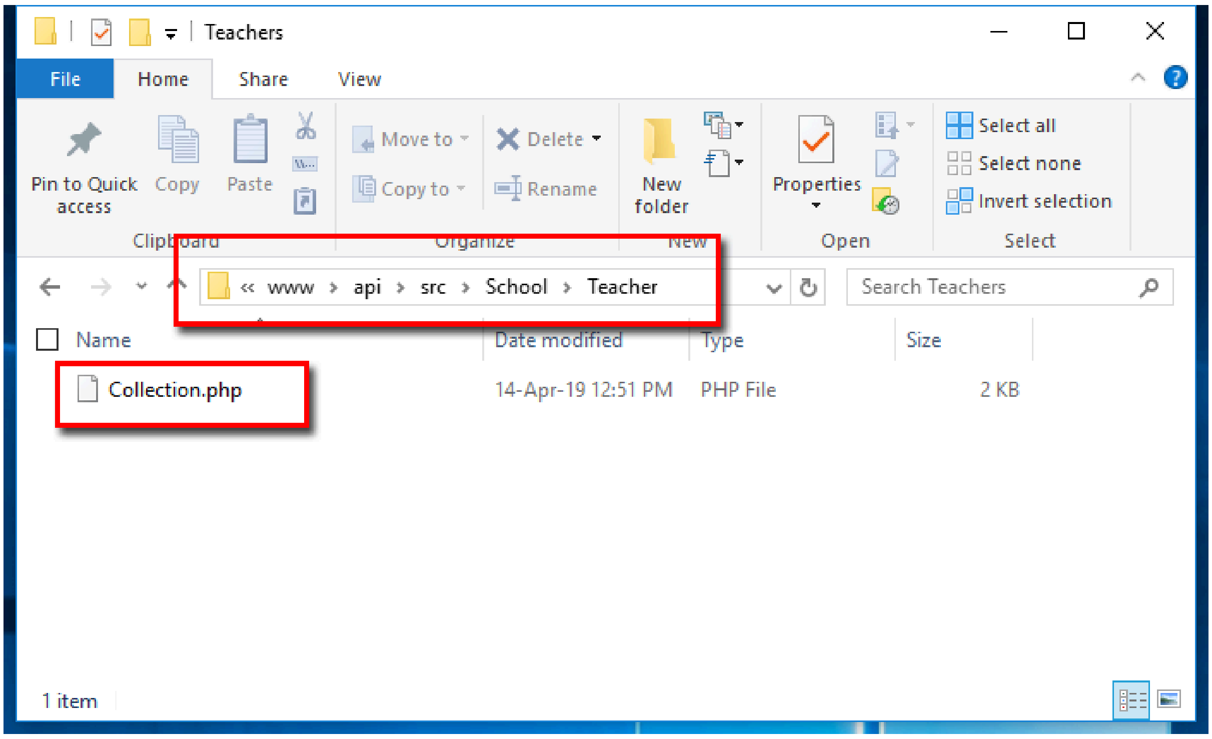
```
<?php

namespace App\School\Teacher;
```

```
use Fusio\Engine\ActionAbstract;
use Fusio\Engine\ContextInterface;
use Fusio\Engine\ParametersInterface;
use Fusio\Engine\RequestInterface;

class Collection extends ActionAbstract
{
    public function handle(RequestInterface $request,
ParametersInterface $configuration, ContextInterface $context)
    {
        /** @var \Doctrine\DBAL\Connection $connection */
        $connection = $this->connector->getConnection('School');

        $sql = 'SELECT teacher_id,
                       formal_title,
                       name_first,
                       name_last,
                       email_address
                  FROM teachers
              ORDER BY teacher_id DESC';

        $sql =
$connection->getDatabasePlatform()->modifyLimitQuery($sql, 16);

        $count   = $connection->fetchColumn('SELECT COUNT(*) FROM
teachers');
        $entries = $connection->fetchAll($sql);

        return $this->response->build(200, [], [
            'totalResults' => $count,
            'entry' => $entries,
        ]);
    }
}
```

Your folder should look like this. Pay attention to your path.

Your PHP Class for this is **App\School\Teacher\Collection**.

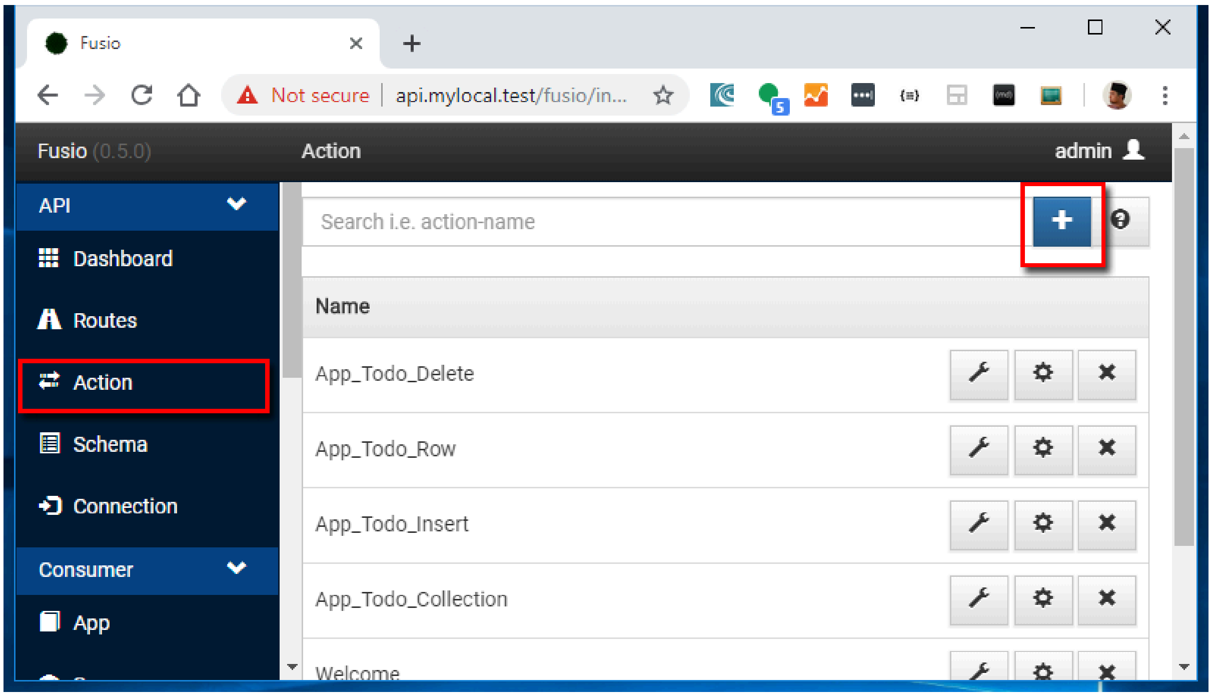
We put it in this manner with the assumption that the system will later have other tables and APIs such as Student, Course, and others.



Name	Date modified	Type	Size
Collection.php	14-Apr-19 12:51 PM	PHP File	2 KB

Creating Action

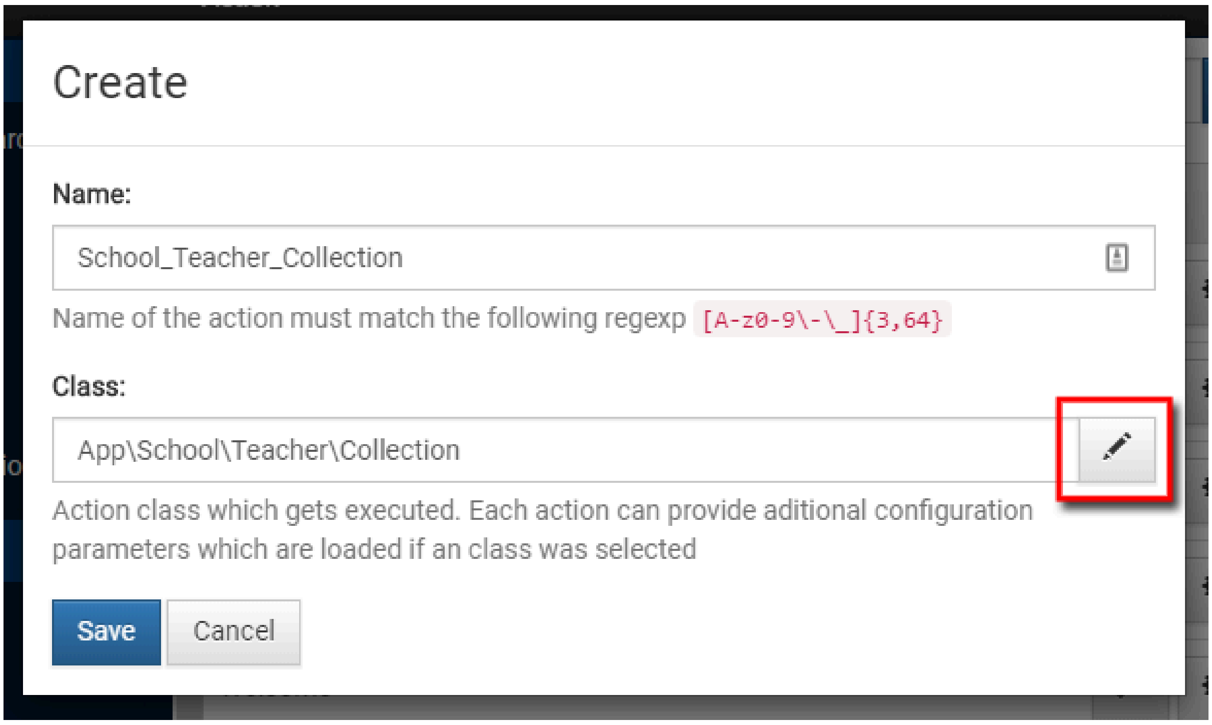
When we are ready with the PHP Class, we are ready to create an Action. Login to your Fusio Administration Panel and click on Action from the left menu. And then click on the blue Add New button, with the plus (+) icon.



Name
App_Todo_Delete
App_Todo_Row
App_Todo_Insert
App_Todo_Collection
Welcome

Enter the following into the Create modal dialog.

Name	School_Teacher_Collection
Class (click on the pencil button)	App\School\Teacher\Collection


Create

Name:

School_Teacher_Collection

Name of the action must match the following regexp `[A-z0-9\-\_]{3,64}`

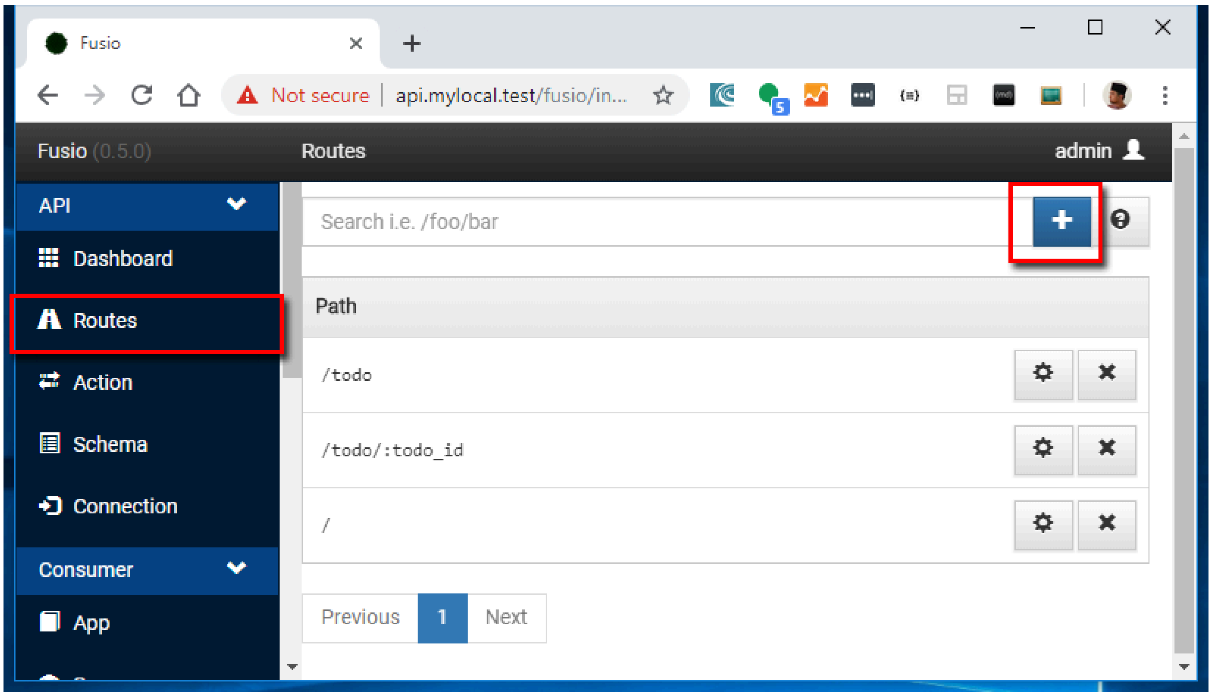
Class:

App\School\Teacher\Collection

Action class which gets executed. Each action can provide aditional configuration parameters which are loaded if an class was selected

Create Route

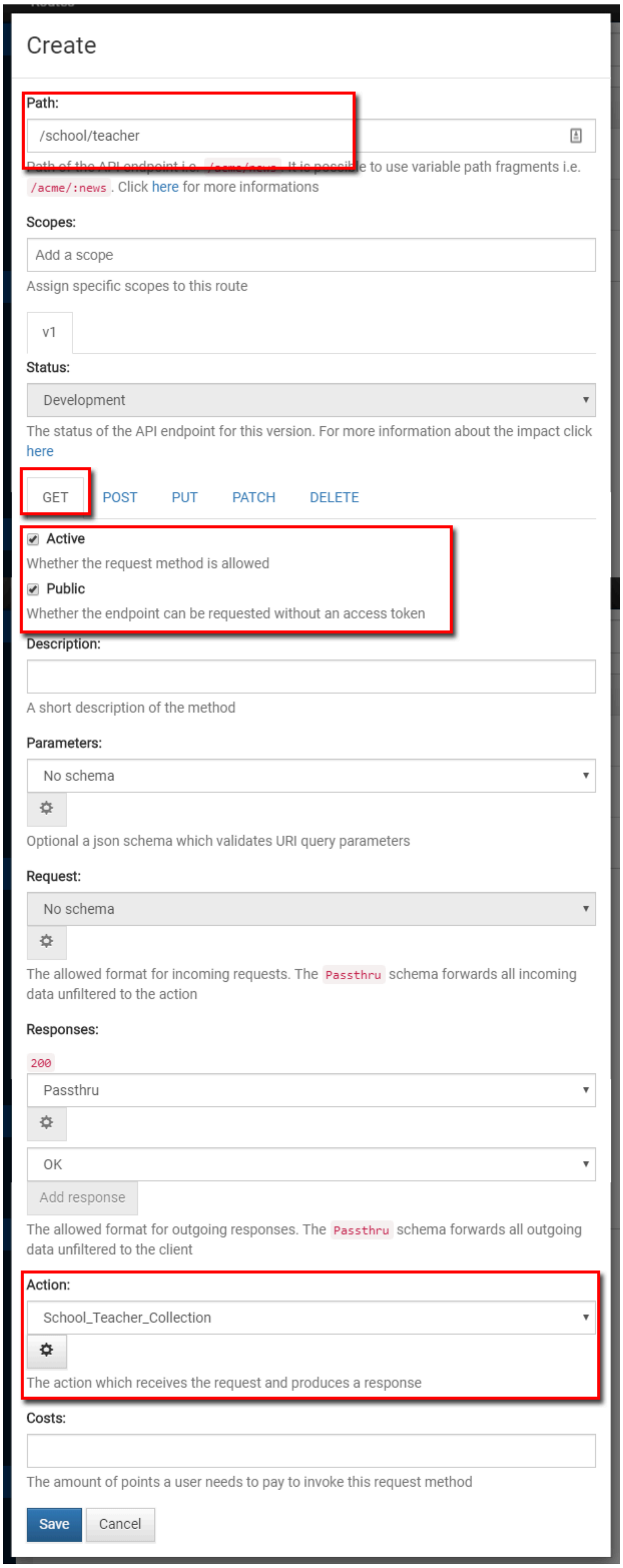
The final step to creating your API is the Route. Click on Routes from the menu on the left in the Fusio Administration Panel. And then click on the blue Add New button with the plus (+) icon.



Enter the following details in the Create modal dialog. While there are many options available when creating a Route, we will ignore some options for now and focus on the following.

Path	/school/teacher
GET (make sure this tab is active)	
Active	(checked)
Public	(checked)
Action (select)	School_Teacher_Collection

Once complete, click **Save**.



Create

Path:

/school/teacher

Path of the API endpoint i.e. `/acme/news`. It is possible to use variable path fragments i.e. `/acme/:news`. Click here for more informations

Scopes:

Add a scope

Assign specific scopes to this route

v1

Status:

Development

The status of the API endpoint for this version. For more information about the impact click here

GET POST PUT PATCH DELETE

☑ Active

Whether the request method is allowed

☑ Public

Whether the endpoint can be requested without an access token

Description:

A short description of the method

Parameters:

No schema

Optional a json schema which validates URI query parameters

Request:

No schema

The allowed format for incoming requests. The `Passthru` schema forwards all incoming data unfiltered to the action

Responses:

`200`

Passthru

OK

Add response

The allowed format for outgoing responses. The `Passthru` schema forwards all outgoing data unfiltered to the client

Action:

School_Teacher_Collection

The action which receives the request and produces a response

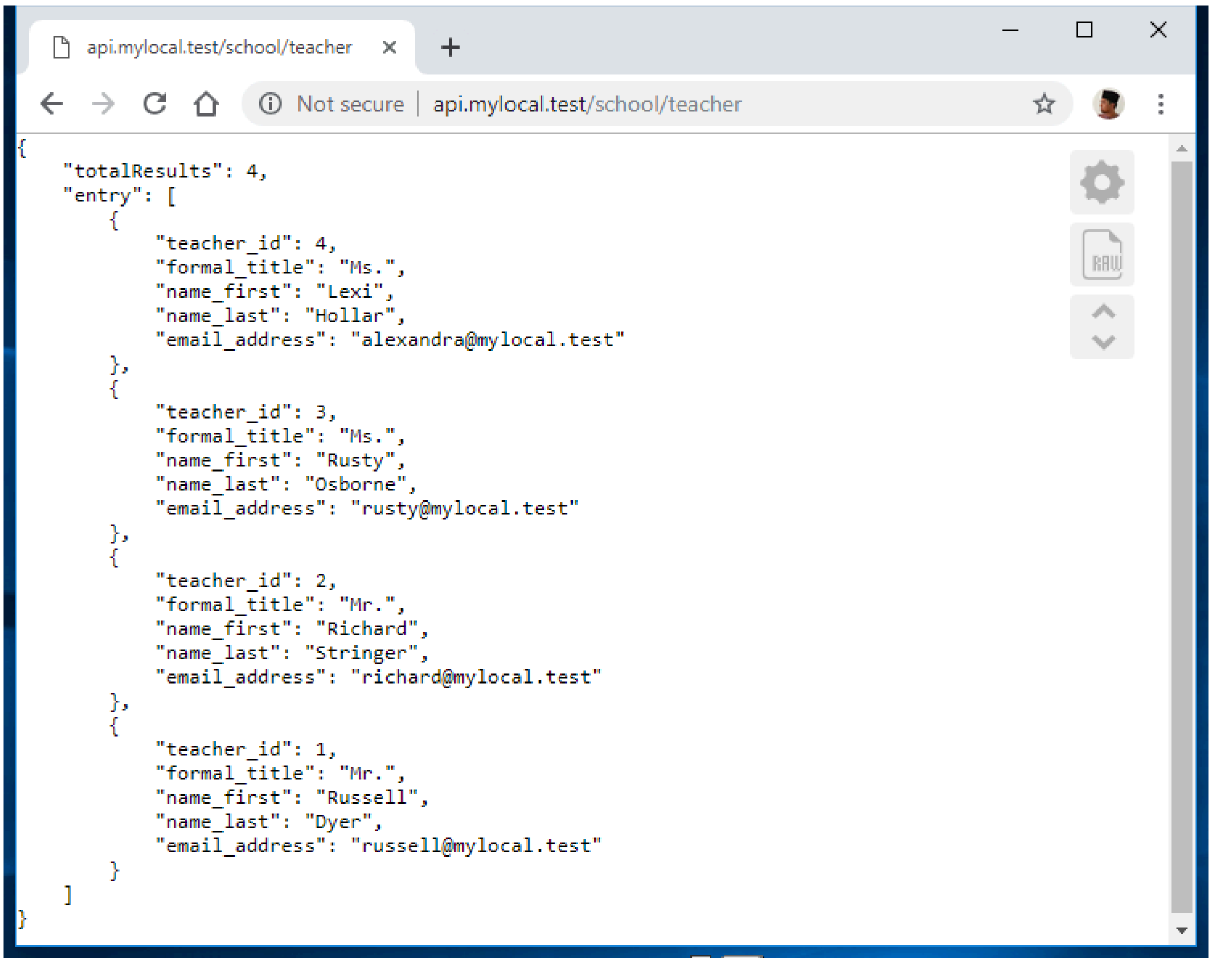
Costs:

The amount of points a user needs to pay to invoke this request method

Save Cancel

Testing Your API

If you made this far, we should have a working API. Because this is a simple GET API that does not require any data submission nor any ID, we can simply enter the **http://api.mylocal.test/school/teacher** into the browser.


api.mylocal.test/school/teacher
Not secure
api.mylocal.test/school/teacher
RAW

```
{
    "totalResults": 4,
    "entry": [
        {
            "teacher_id": 4,
            "formal_title": "Ms.",
            "name_first": "Lexi",
            "name_last": "Hollar",
            "email_address": "alexandra@mylocal.test"
        },
        {
            "teacher_id": 3,
            "formal_title": "Ms.",
            "name_first": "Rusty",
            "name_last": "Osborne",
            "email_address": "rusty@mylocal.test"
        },
        {
            "teacher_id": 2,
            "formal_title": "Mr.",
            "name_first": "Richard",
            "name_last": "Stringer",
            "email_address": "richard@mylocal.test"
        },
        {
            "teacher_id": 1,
            "formal_title": "Mr.",
            "name_first": "Russell",
            "name_last": "Dyer",
            "email_address": "russell@mylocal.test"
        }
    ]
}
```

Summary

You should have learned:

- How to write your PHP Class
- Create a new Action with your PHP Class
- Create a new API endpoint with Routes using your Action
- Under that you have the option to handle GET, POST, PUT, DELETE, PATCH for each API endpoint path

API & Working with Data

In the previous tutorial, we created a simple API that only reads data from the database and display them as response. In this tutorial we will create an API that will be able to receive data from the external party and update the specified record.

Our goal is to create the following API.

Path	/school/teacher/:teacher_id	**:teacher_id** is the record id Example /school/teacher/2
Method	POST	

Parameters	New data formatted in JSON format	Example:
		`{`
		`  "name_first" : "Howard",`
		`  "name_first" : "Harkness"`
		`}`

Result	Data in JSON format	Example:
		`{`
		`  "success" : true`
		`}`

In this tutorial, we will be working with the same **school** database and the **teacher** table. Therefore, there is no need to create any database and connection. We can continue with creating our PHP Class.

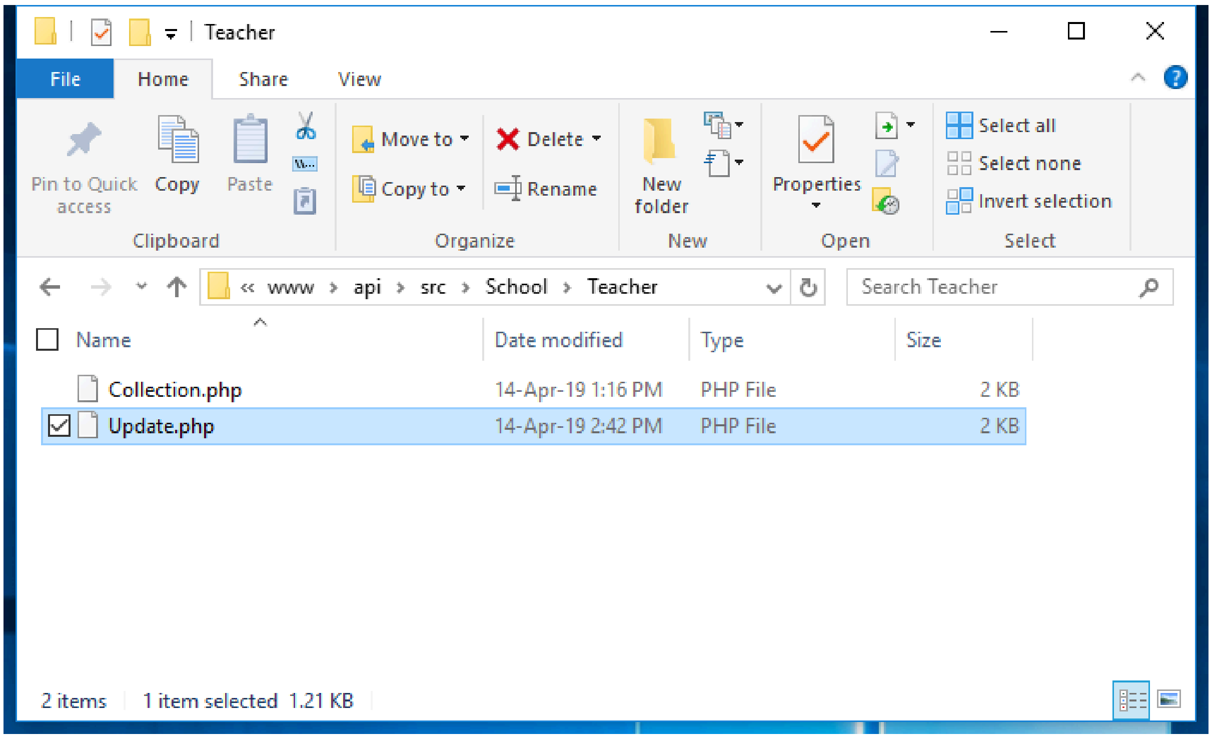
Create PHP Class

We will take the example from the Delete.php from To-Do API which is available in src/Todo/Delete.php. Create the following file and save it as **src/School/Teacher/Update.php**.

I should also tell you that the Delete.php from To-Do is actually an update operation, which changes the status of the record. So we can use that to start with.

```
<?php

namespace App\School\Teacher;

use Fusio\Engine\ActionAbstract;
use Fusio\Engine\ContextInterface;
use Fusio\Engine\ParametersInterface;
use Fusio\Engine\RequestInterface;
use PSX\Http\Exception as StatusCode;

class Update extends ActionAbstract
{
    public function handle(RequestInterface $request,
ParametersInterface $configuration, ContextInterface $context)
    {
        /** @var \Doctrine\DBAL\Connection $connection */
        $connection = $this->connector->getConnection('School');
        $tosave = [];
        $data = $request->getBody();

        if (isset($data->formal_title))
            { $tosave['formal_title'] = $data->formal_title; }
        if (isset($data->name_first))
            { $tosave['name_first'] = $data->name_first; }
        if (isset($data->name_last))
            { $tosave['name_last'] = $data->name_last; }
        if (isset($data->email_address))
            { $tosave['email_address'] = $data->email_address; }

        $affected = $connection->update('teachers', $tosave, [
            'teacher_id' => $request->getUriFragment('teacher_id')
        ]);

        if (empty($affected)) {
            throw new StatusCode\NotFoundException('Entry not
available');
        }

        return $this->response->build(200, [], [
            'success' => true,
            'message' => 'Update successful',
        ]);
    }
}
```

Now we should have something like this in the Teacher folder.



Name	Date modified	Type	Size
Collection.php	14-Apr-19 1:16 PM	PHP File	2 KB
Update.php	14-Apr-19 2:42 PM	PHP File	2 KB

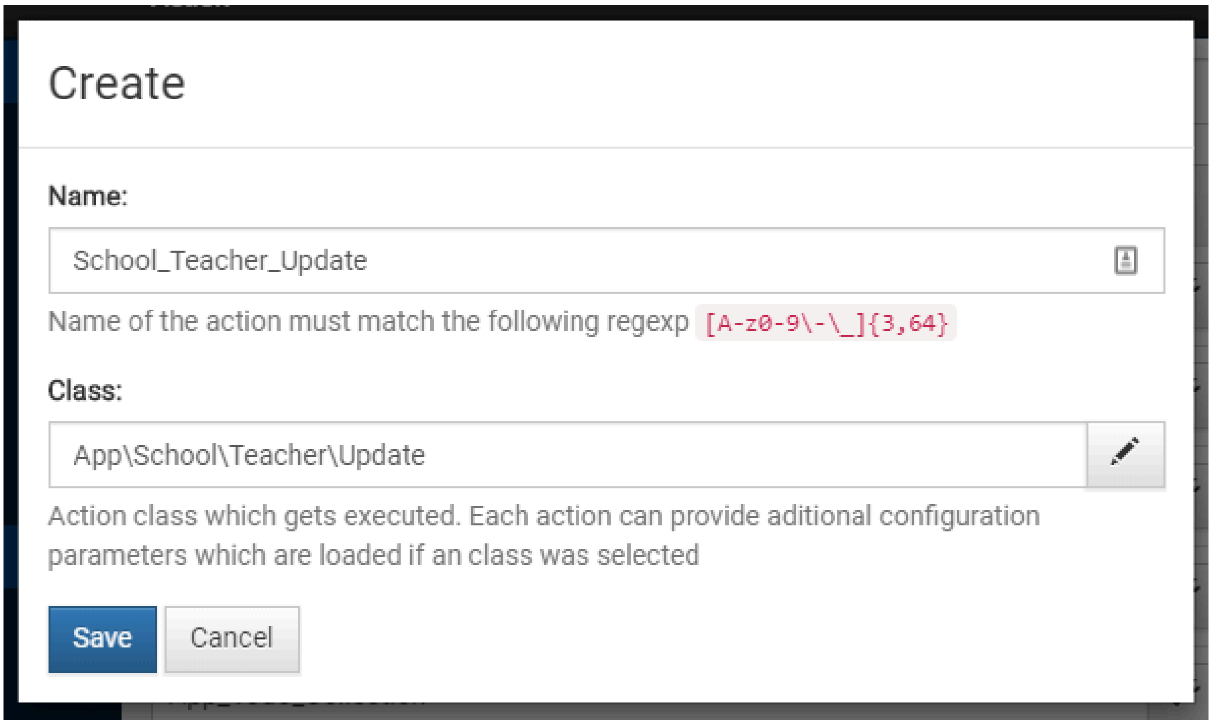
2 items 1 item selected 1.21 KB

Create Action

In your Fusio Administration Panel, click on **Action** on the left menu. Click on the blue Add New button with the plus (+) icon.

Enter the following details. Then click **Save**.

Name	School_Teacher_Update
Class (click on the pencil button)	App\School\Teacher\Update


Create

Name:

School_Teacher_Update

Name of the action must match the following regexp `[A-z0-9\-\_]{3,64}`

Class:

App\School\Teacher\Update

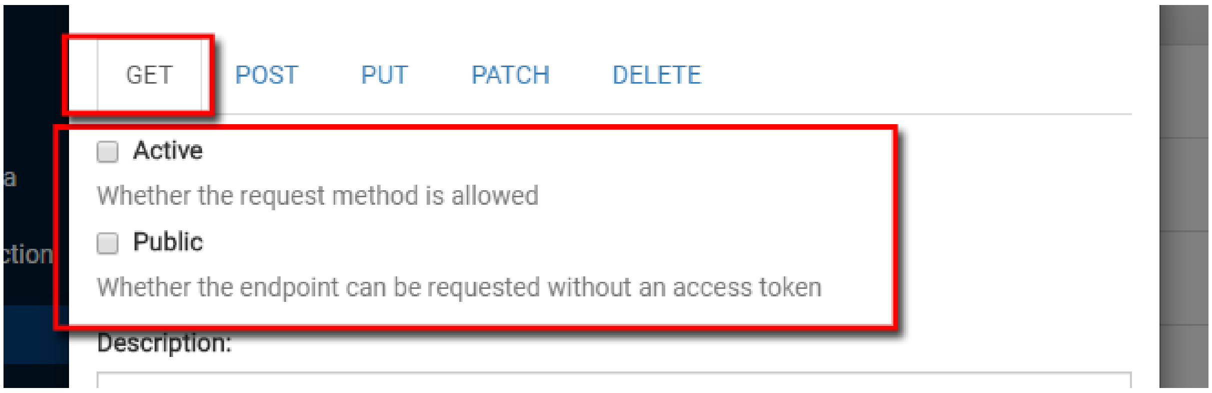
Action class which gets executed. Each action can provide aditional configuration parameters which are loaded if an class was selected

Once done here, we are ready to create the **Route**.

Create Route

The final step to creating your API is the Route. Click on Routes from the menu on the left in the Fusio Administration Panel. And then click on the blue Add New button with the plus (+) icon.

Since we are only creating an endpoint for the POST method, make sure you uncheck the Active from the GET tab.



☐ Active

Whether the request method is allowed

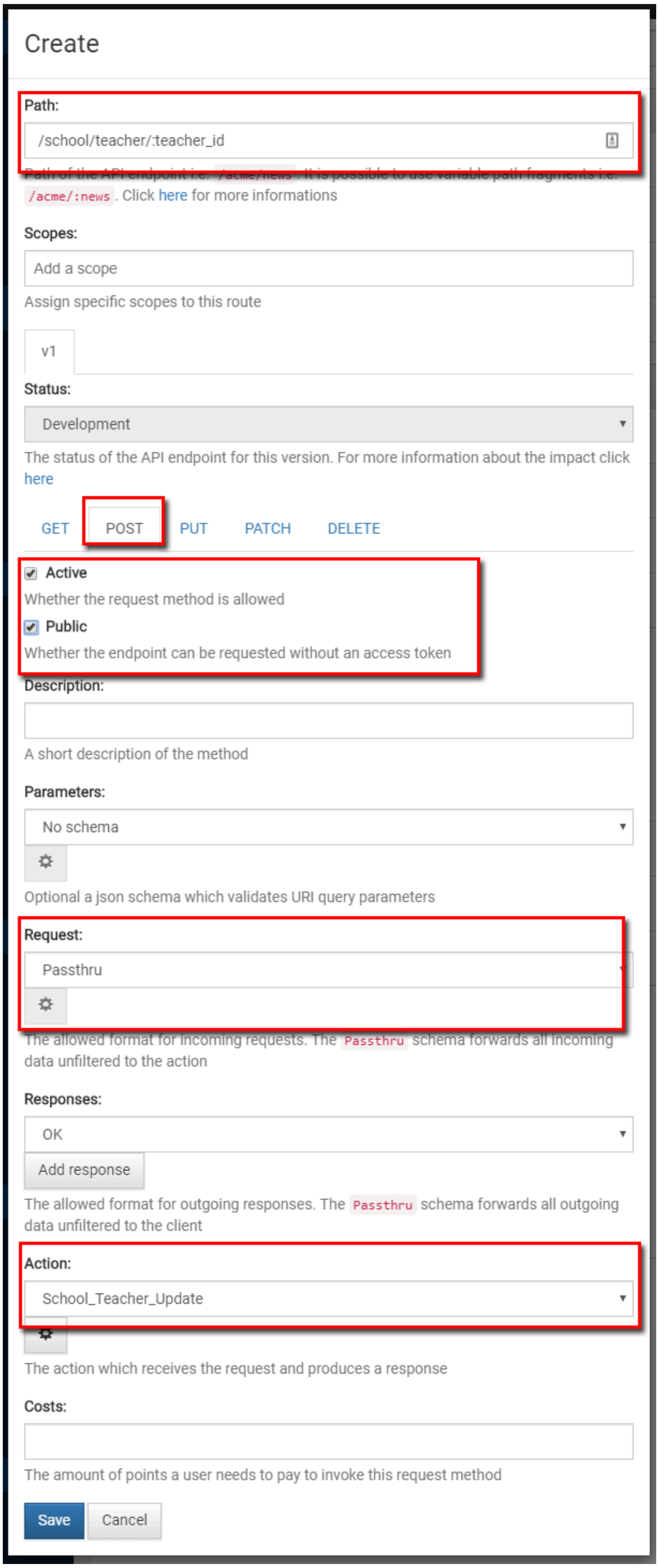
☐ Public

Whether the endpoint can be requested without an access token

Description:

Now, proceed to the POST method tab and enter the following details. We can ignore some of the rest and leave it with the default values.

Path	/school/teacher/:teacher_id	:teacher_id would allow the code in PHP class to fetch this information with `$request->getUriFragment('teacher_id')`
POST		
Active	(checked)	
Public	(checked)	
Request (select)	Passthru	This would allow data to be sent to the PHP class codes with `$data = $request->getBody();`
Action (select)	School_Teacher_Update	



Create

Path:

/school/teacher/:teacher_id

Path of the API endpoint i.e. `/acme/news`. It is possible to use variable path fragments i.e. `/acme/:news`. Click here for more informations

Scopes:

Add a scope

Assign specific scopes to this route

v1

Status:

Development

The status of the API endpoint for this version. For more information about the impact click here

GET POST PUT PATCH DELETE

☑ Active

Whether the request method is allowed

☑ Public

Whether the endpoint can be requested without an access token

Description:

A short description of the method

Parameters:

No schema

Optional a json schema which validates URI query parameters

Request:

Passthru

The allowed format for incoming requests. The `Passthru` schema forwards all incoming data unfiltered to the action

Responses:

OK

Add response

The allowed format for outgoing responses. The `Passthru` schema forwards all outgoing data unfiltered to the client

Action:

School_Teacher_Update

The action which receives the request and produces a response

Costs:

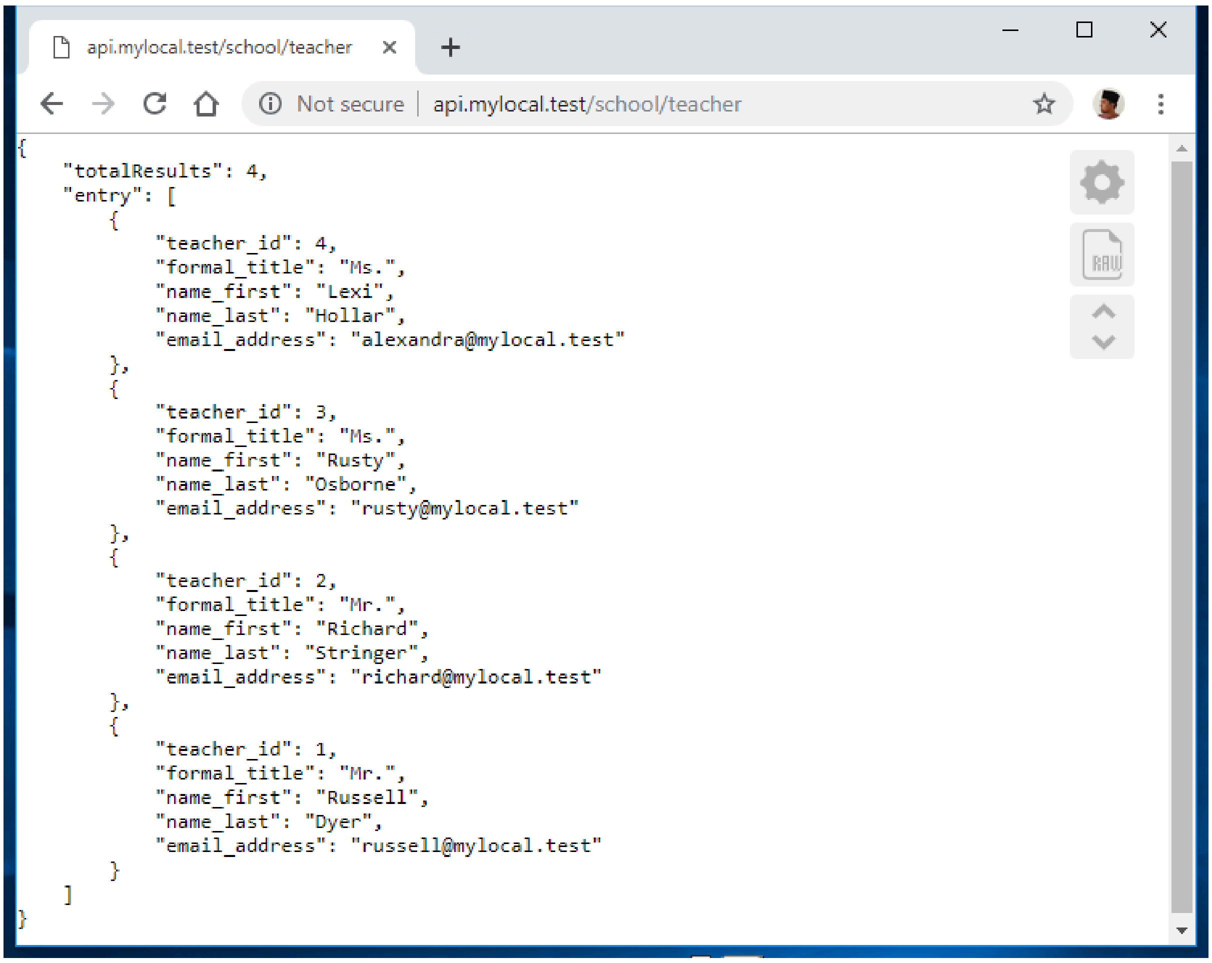
The amount of points a user needs to pay to invoke this request method

Save Cancel

Testing the API

To test this API, we will be using Insomnia. Before we do that, let's take a look at how the current data looks like.

If you remember from the previous tutorial, you can check the data of the table using the previous API. Enter **http://api.mylocal.test/school/teacher** into the browser.

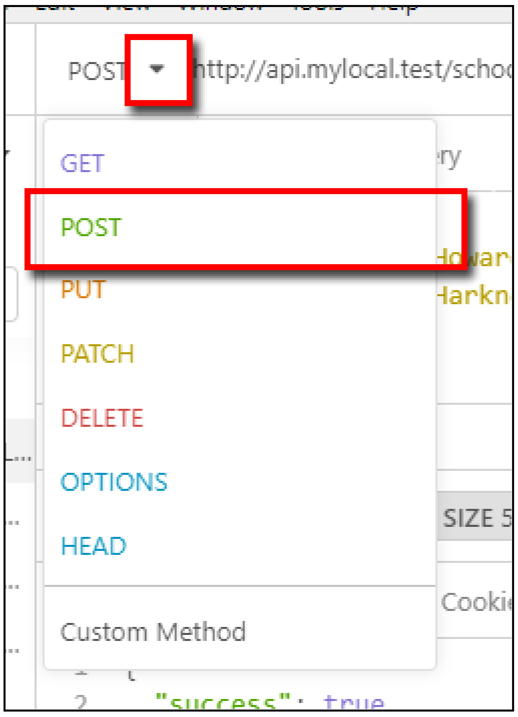

api.mylocal.test/school/teacher
Not secure
api.mylocal.test/school/teacher
RAW

```
{
    "totalResults": 4,
    "entry": [
        {
            "teacher_id": 4,
            "formal_title": "Ms.",
            "name_first": "Lexi",
            "name_last": "Hollar",
            "email_address": "alexandra@mylocal.test"
        },
        {
            "teacher_id": 3,
            "formal_title": "Ms.",
            "name_first": "Rusty",
            "name_last": "Osborne",
            "email_address": "rusty@mylocal.test"
        },
        {
            "teacher_id": 2,
            "formal_title": "Mr.",
            "name_first": "Richard",
            "name_last": "Stringer",
            "email_address": "richard@mylocal.test"
        },
        {
            "teacher_id": 1,
            "formal_title": "Mr.",
            "name_first": "Russell",
            "name_last": "Dyer",
            "email_address": "russell@mylocal.test"
        }
    ]
}
```

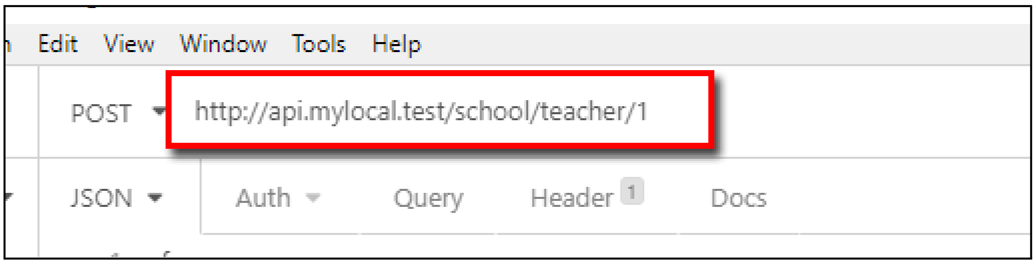
We will be editing the first record, which is Mr. Russell Dyer.

Now Launch Insomnia and follow these steps.

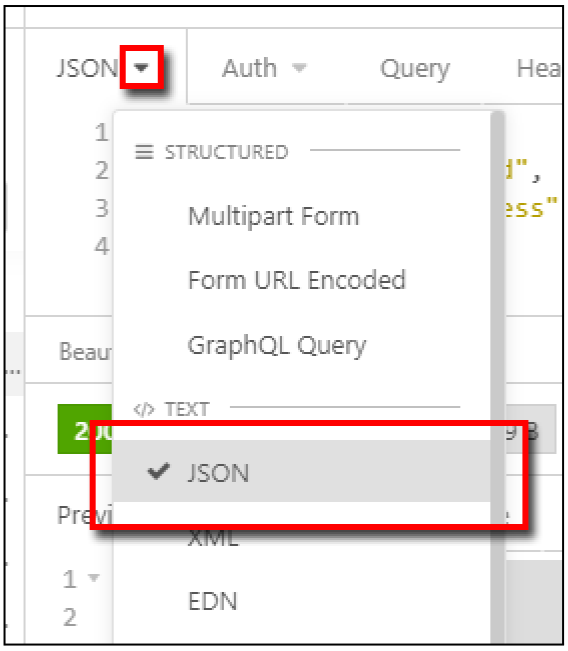
1. Select POST as the method. Click on the little triangle to make the options appear.



2. Enter the URI path, **http://api.mylocal.test/school/teacher/1**. This means we are editing record number 1.

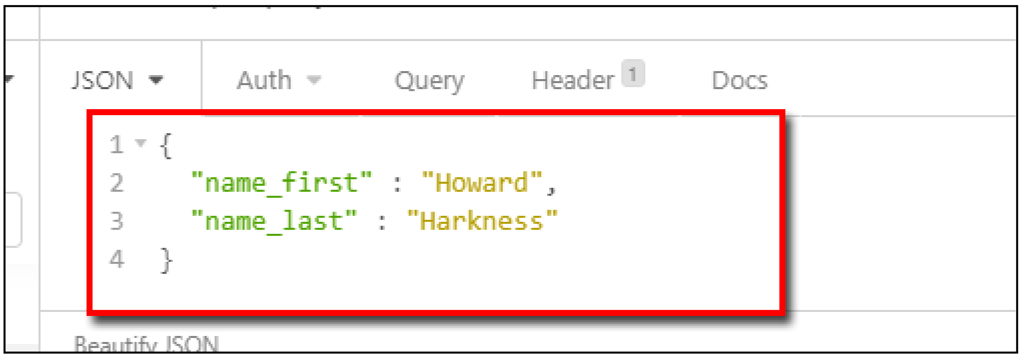

Edit View Window Tools Help
POST
http://api.mylocal.test/school/teacher/1
JSON
Auth
Query
Header 1
Docs

3. Select JSON for body content. Click on the little triangle to make options appear.



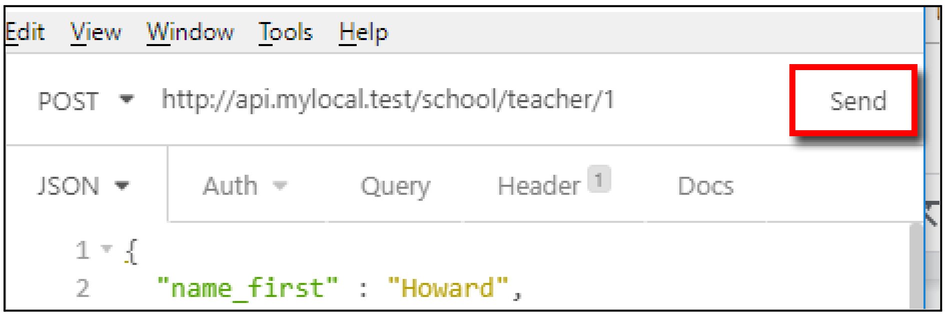
4. Enter the following as body content.

```
{
  "name_first" : "Howard",
  "name_last" : "Harkness"
}
```



```
{
   "name_first" : "Howard",
   "name_last" : "Harkness"
}
```

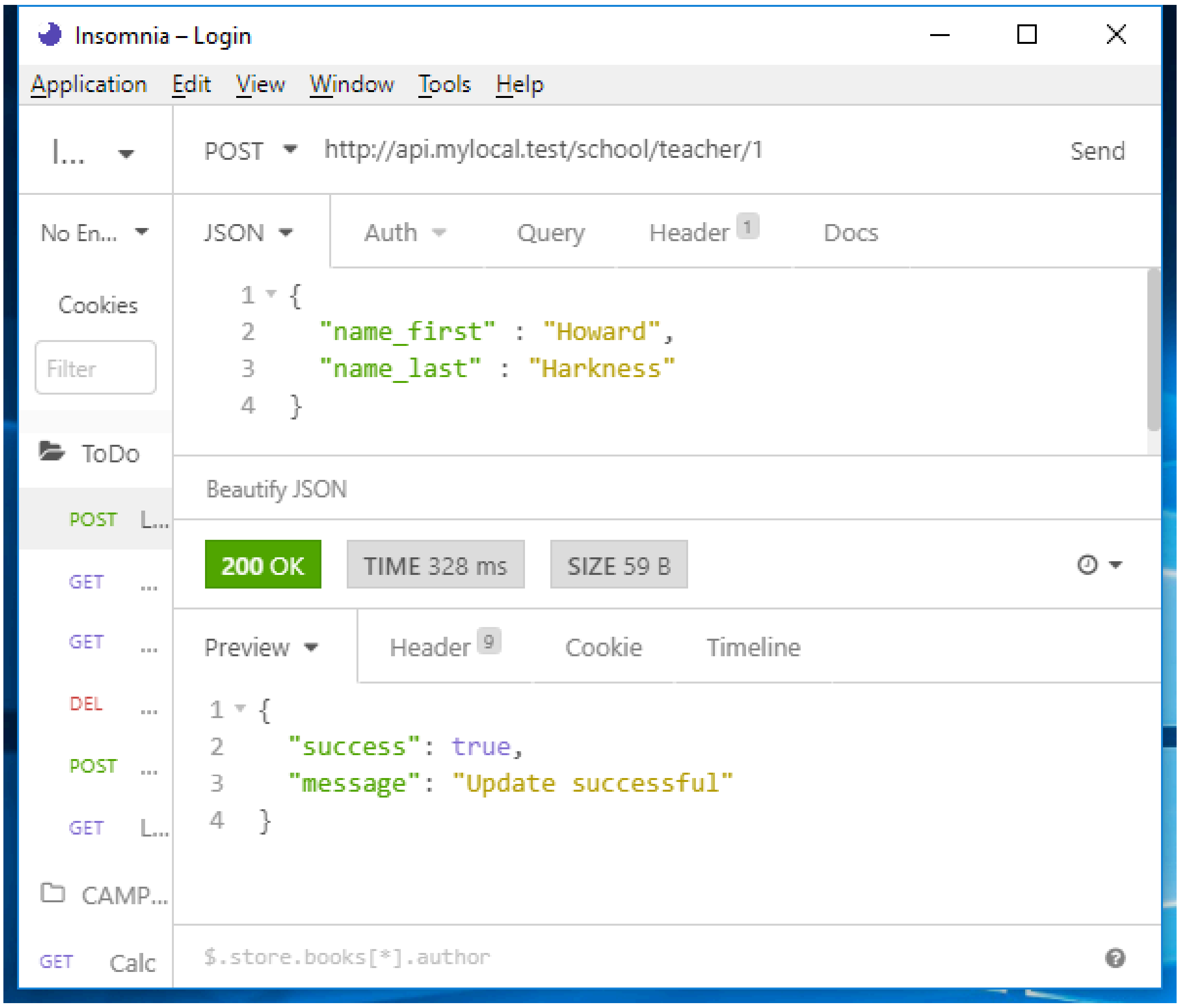
5. Hit **Send**.



http://api.mylocal.test/school/teacher/1

```
{
  "name_first" : "Howard",
```

If all goes well, you should be getting a response from the API.



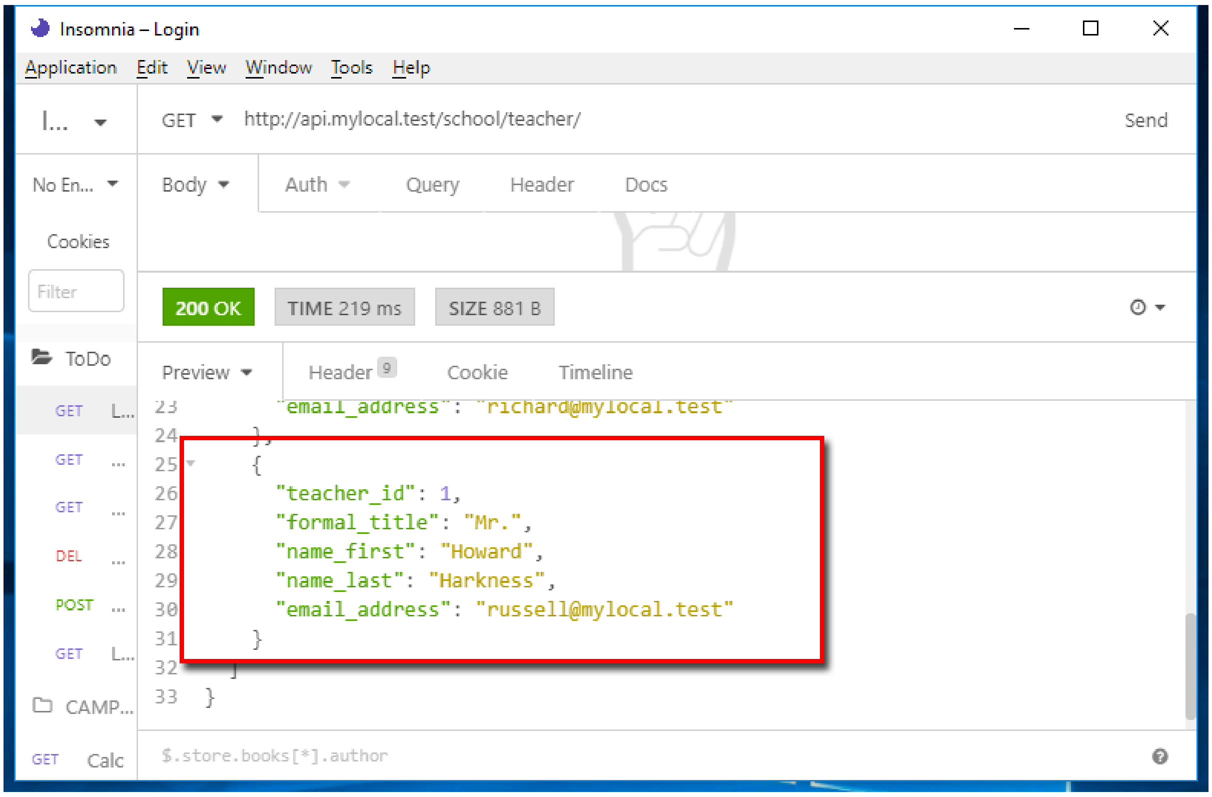
```
POST http://api.mylocal.test/school/teacher/1
```

```
{
   "name_first" : "Howard",
   "name_last" : "Harkness"
 }
```

```
{
   "success": true,
   "message": "Update successful"
 }
```

Let’s check the previous API to see if record has been updated. We continue with Insomnia with this. Just change the method to **GET** and the address to **http://api.mylocal.test/school/teacher** and hit **Send**.

You should be able to see that the record number 1 has been updated.



http://api.mylocal.test/school/teacher/

```
        "email_address": "richard@mylocal.test"
    },
    {
        "teacher_id": 1,
        "formal_title": "Mr.",
        "name_first": "Howard",
        "name_last": "Harkness",
        "email_address": "russell@mylocal.test"
    }
  ]
}
```

$.store.books[*].author

Summary

In this tutorial, you should have learned:

- How to create an API endpoint that takes data from the URL
- How to read the value (ie, teacher_id) from the URL in your PHP code
- How to read JSON data sent via the body
- How to use Insomnia to test your API with JSON data as body content

Users, Scope & Private API

In the tutorial, we will learn how to manage users and the API that they can access. This tutorial will go through the following process.

1. Making API private
2. Creating a Scope and grouping APIs to Scope
3. Creating a Consumer user and assigning a Scope
4. Using Private API with JSON Web Token (JWT)

Making API Private

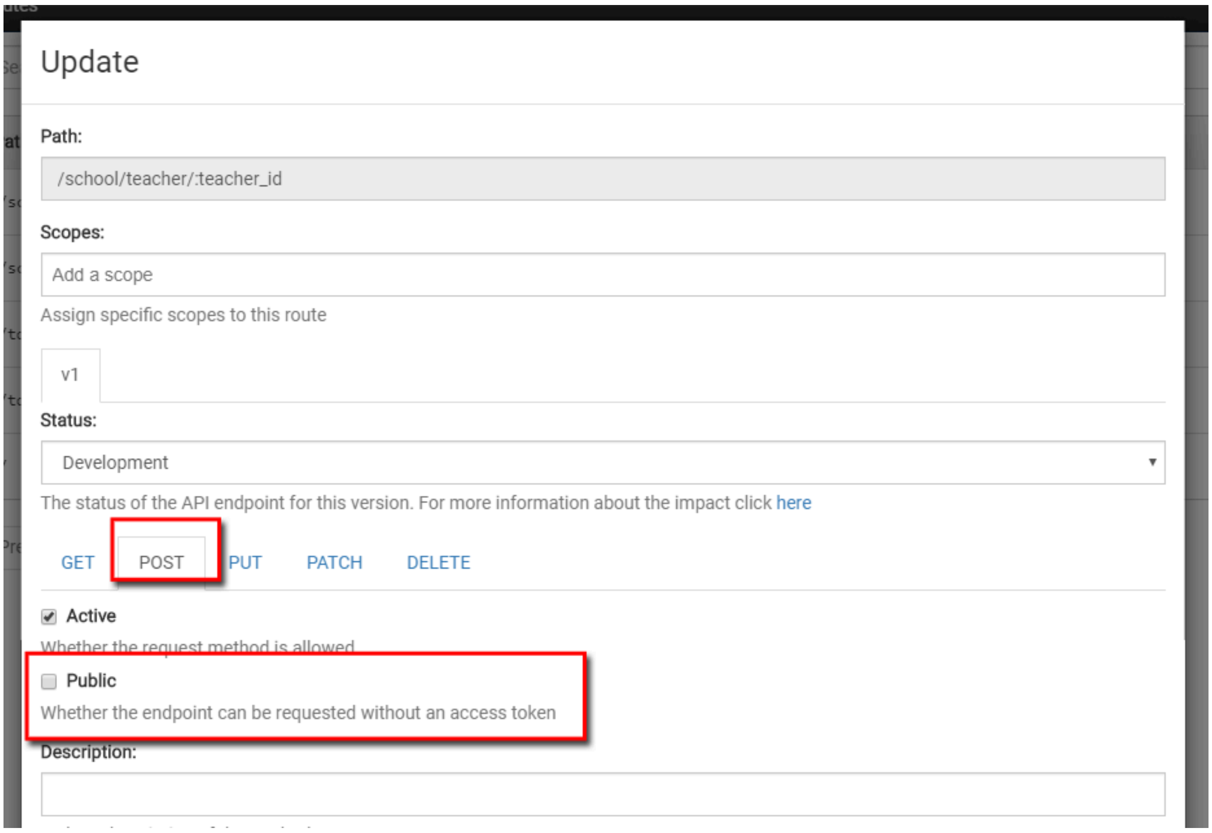
In the previous tutorial, we have created two API endpoints where both of them are available publicly. This means that anybody can access the APIs without having to provide any credentials.

We are going to make changes to the existing **/school/teacher/:teacher_id** (POST) API to be private instead of public.

From your Fusio Administration Panel, click Route from the left menu and click on the ⚙ gear icon that represents the **/school/teacher/:teacher_id** API.

Click to make the POST tab panel active. Then click to uncheck the **Public** option. Click **Save** at the bottom of the form.

What this means is that this API is now private. Only users with the assigned privilege will be able to use this API.


Update

Path: /school/teacher/:teacher_id

Scopes: Add a scope
Assign specific scopes to this route

v1

Status: Development
The status of the API endpoint for this version. For more information about the impact click here

☑ Active
Whether the request method is allowed

☐ Public
Whether the endpoint can be requested without an access token

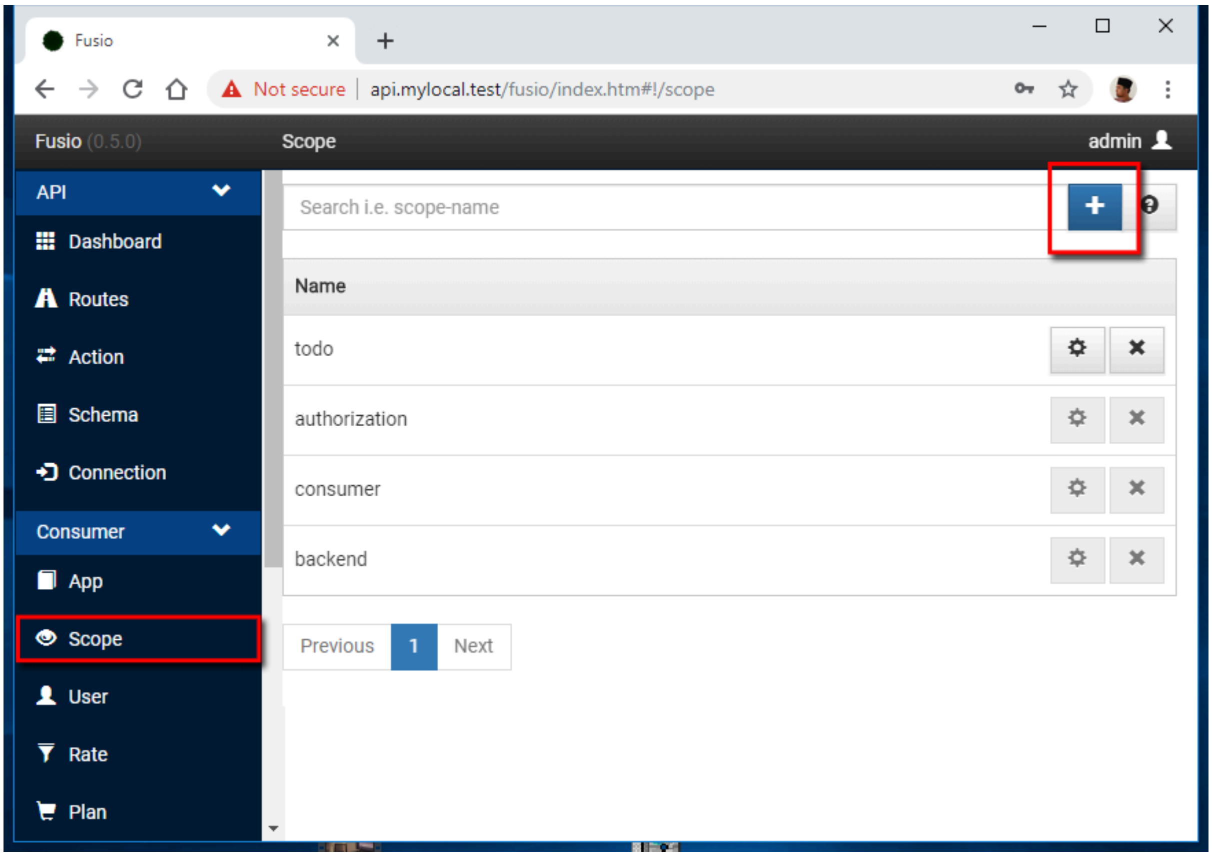
Description:

When done with this API, you can also do this with the other School API, **/school/teacher**.

Create Scope

A Scope allows you to group API endpoints. With this, you can later assign Scope(s) to a user. Only users with the Scope privileges can access the designated APIs.

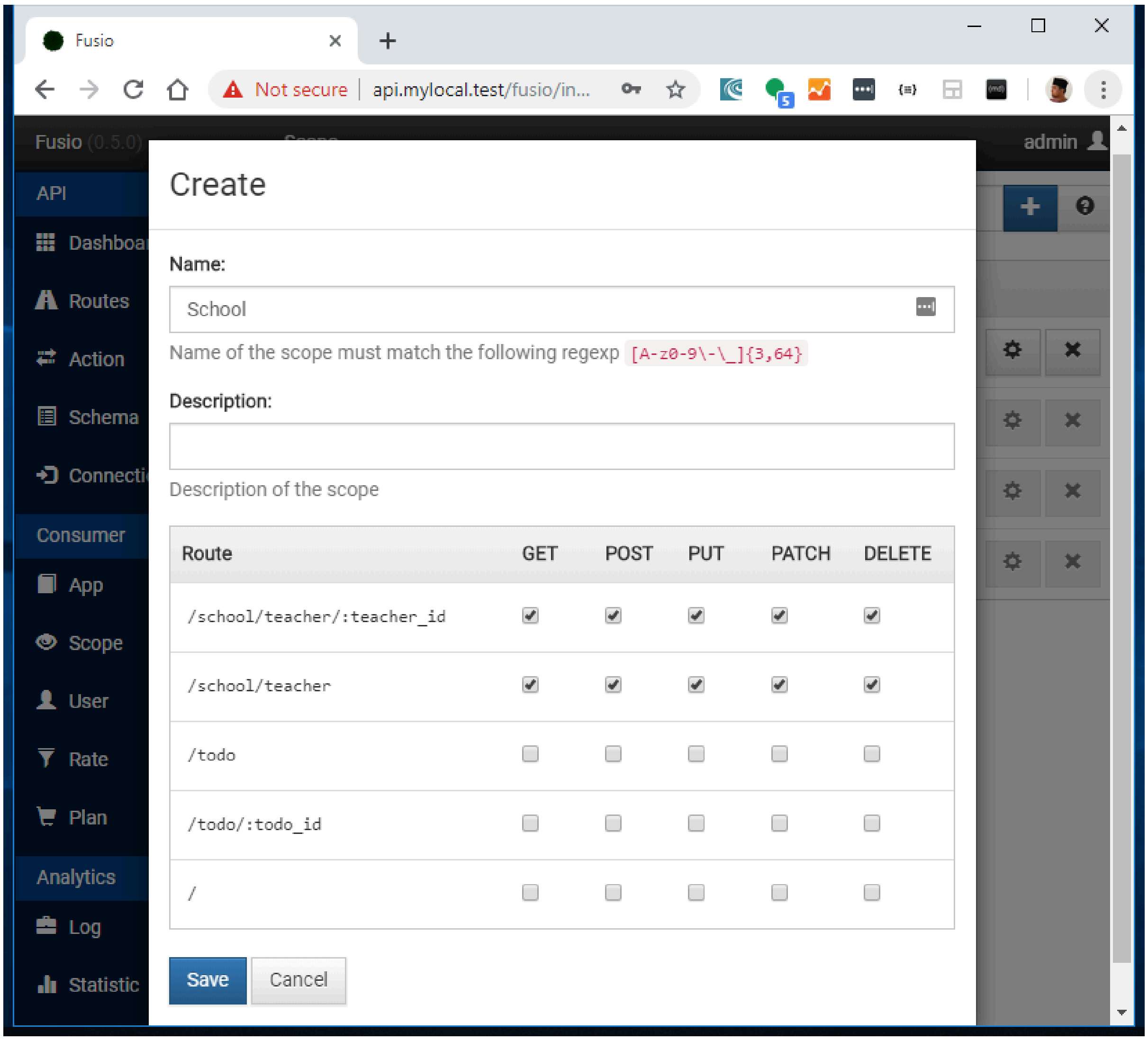
To begin creating a Scope, click on **Scope** from the left menu. Then click on the blue Add New button with the plus (+) icon.



We will group all the school related APIs into the School scope. So we will name this Scope as School. Make sure you have the following checkboxes selected.

Name	School				
	GET	**POST**	**PUT**	**PATCH**	**DELETE**
/school/teacher	☐	☐	☐	☐	☐
/school/teacher/:teacher_id	☐	☐	☐	☐	☐

Click **Save** when done.



Create

Name:

School

Name of the scope must match the following regexp `[A-z0-9\-\_]{3,64}`

Description:

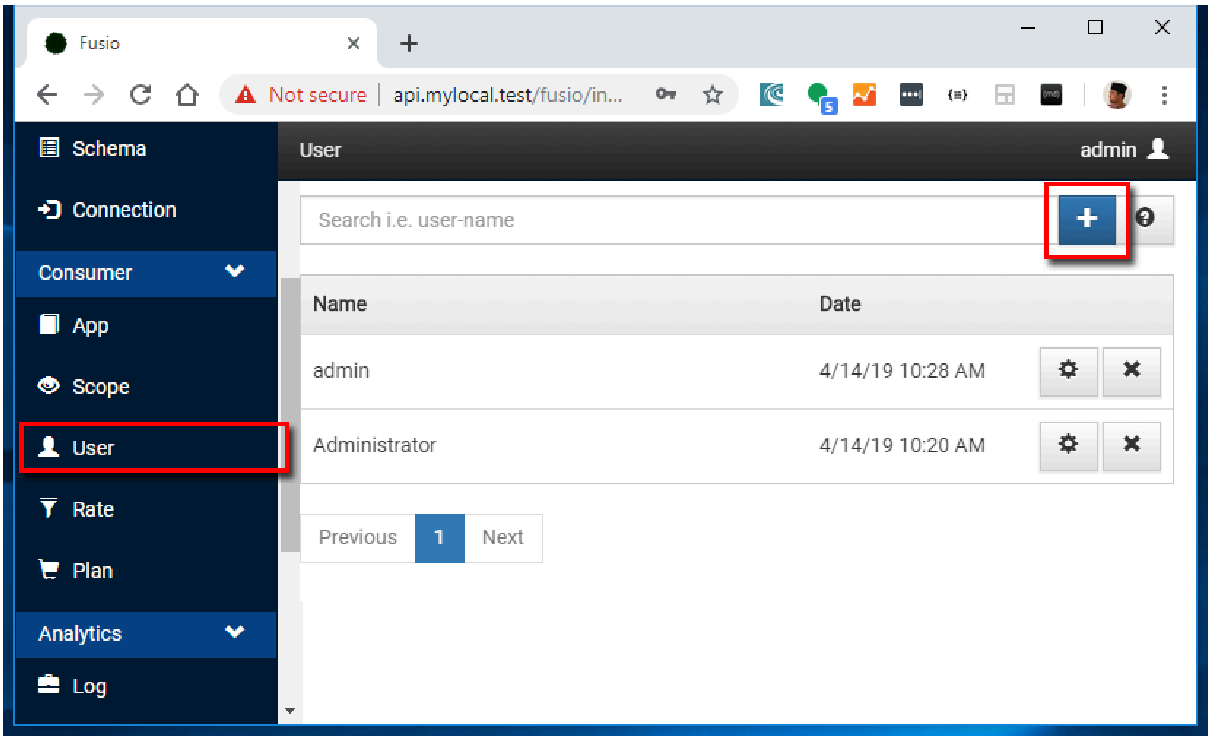
Description of the scope

Route	GET	POST	PUT	PATCH	DELETE
/school/teacher/:teacher_id	✓	✓	✓	✓	✓
/school/teacher	✓	✓	✓	✓	✓
/todo	☐	☐	☐	☐	☐
/todo/:todo_id	☐	☐	☐	☐	☐
/	☐	☐	☐	☐	☐

Save Cancel

Create User

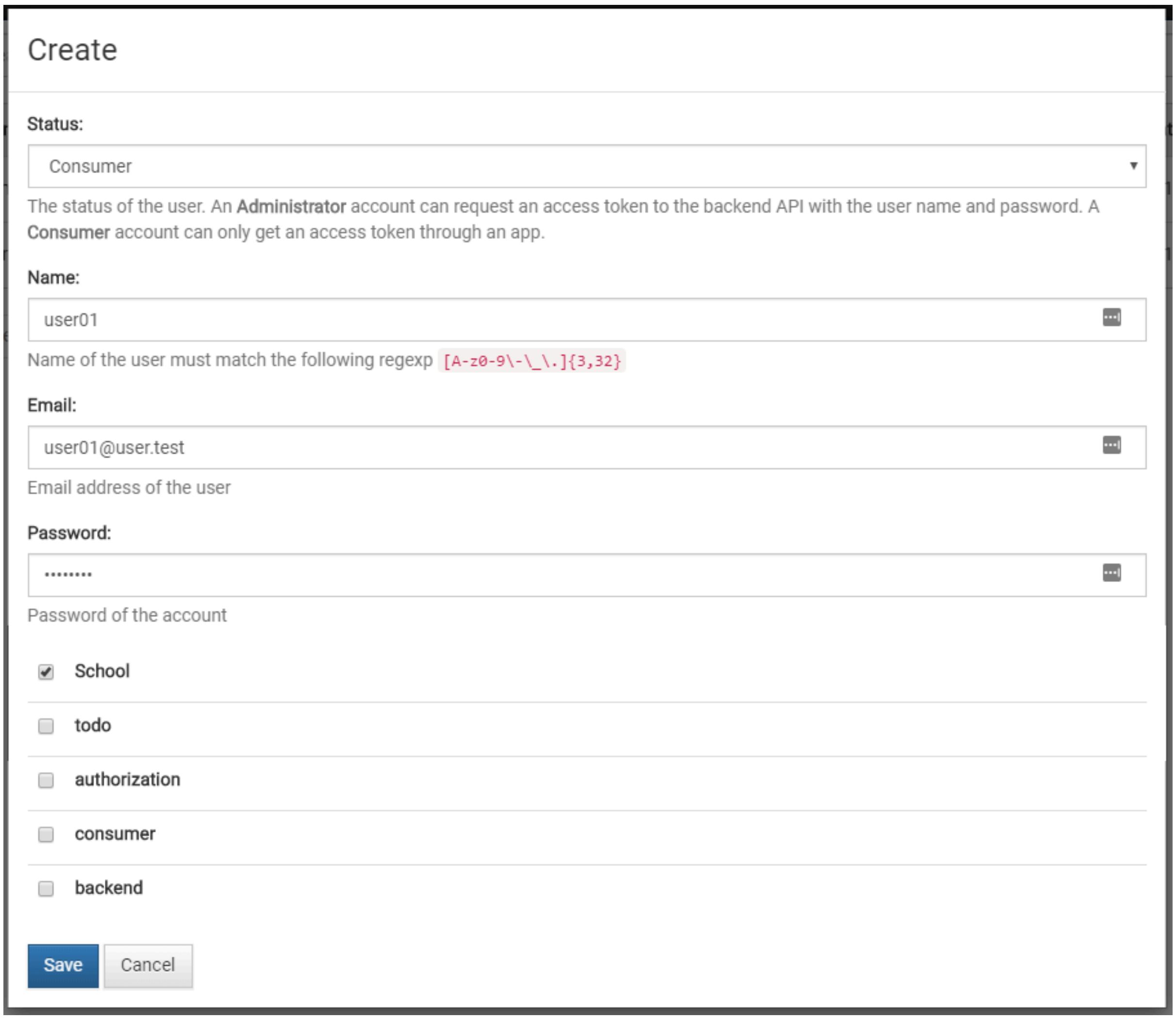
To create a User, click on **User** from the left menu. Then click on the blue Add New button with the plus (+) icon.



Name	Date
admin	4/14/19 10:28 AM
Administrator	4/14/19 10:20 AM

Now enter the details for this user as below. For the sake of this tutorial, please follow the details as specified below.

Status	Consumer
Name	user01
Email	user01@user.test
Password	user1234
School	(checked)
todo	(checked)


Create

Status:

Consumer

The status of the user. An **Administrator** account can request an access token to the backend API with the user name and password. A **Consumer** account can only get an access token through an app.

Name:

user01

Name of the user must match the following regexp `[A-z0-9\-\_\.]{3,32}`

Email:

user01@user.test

Email address of the user

Password:

••••••••

Password of the account

- [x] School
- [ ] todo
- [ ] authorization
- [ ] consumer
- [ ] backend

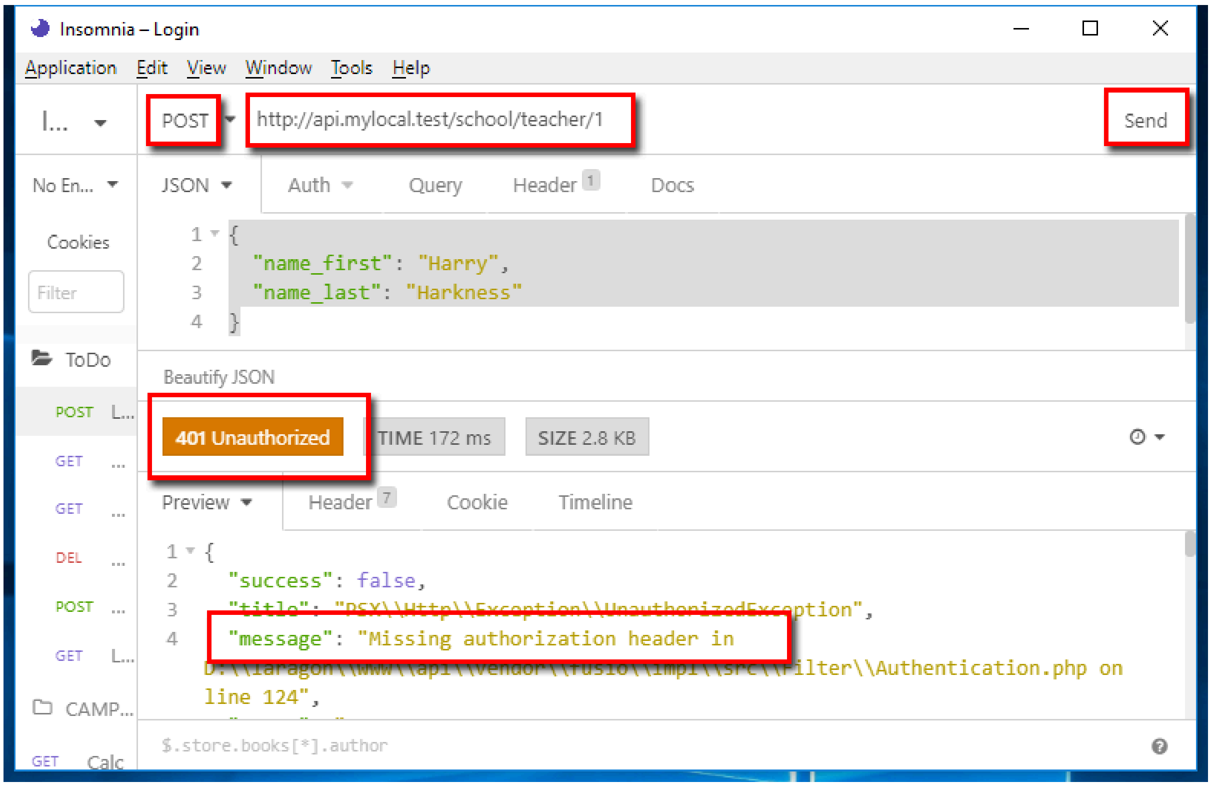
By now we should have made the API private, created a **Scope** and created a **User**.

Accessing a Private API without Credentials

Let's try the API now without any credentials. Launch Insomnia and access the API as specified below:

Method	POST
URL	http://api.mylocal.test/school/teacher/1
Body Format	JSON
Body Content	`{ "name_first": "Harry", "name_last": "Harkness" }`

When ready, click on **Send**.


Insomnia – Login
Application Edit View Window Tools Help
POST
http://api.mylocal.test/school/teacher/1
Send
JSON Auth Query Header 1 Docs
Beautify JSON
401 Unauthorized
TIME 172 ms
SIZE 2.8 KB
Preview Header 7 Cookie Timeline

```
{
  "name_first": "Harry",
  "name_last": "Harkness"
}
```

```
{
  "success": false,
  "title": "PSX\\Http\\Exception\\UnauthorizedException",
  "message": "Missing authorization header in D:\\laragon\\www\\api\\vendor\\fusio\\impl\\src\\Filter\\Authentication.php on line 124",
```

If you look at the response now, you will see that you get a 401 Error. And with this, there is a **message** that starts with **“Missing authorization header in…”**

This means that you are trying to access a private API without sending your credentials. In the next section, we will access this API again the correct way.

How to Access Private API

Before we access the private API, please understand how this works.

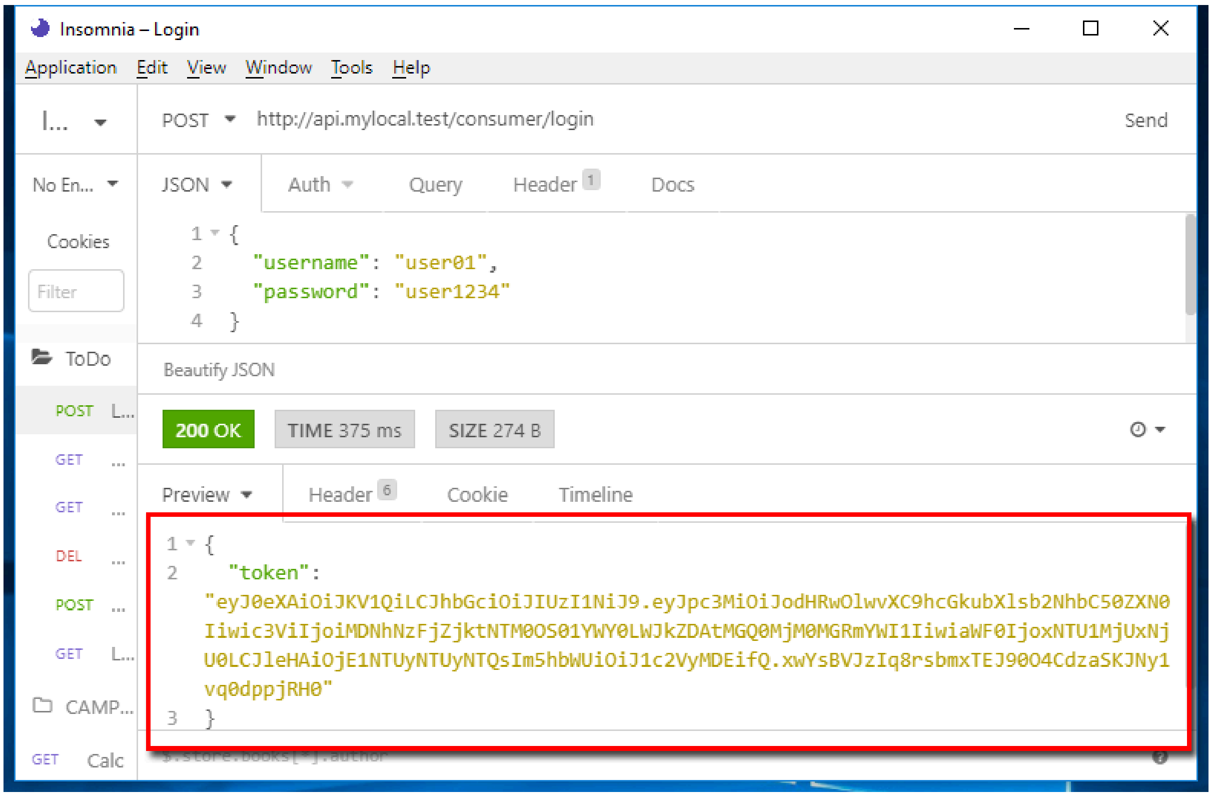
1. Send and API request to **/consumer/login** with the user’s username and password. The /consumer/login endpoint is a built-in endpoint by Fusio.
2. If the username and password is correct, you will get a unique hash string, which is the JSON Web Token.
3. Since the user has been assigned the scope for this API, we can try to access the API again with the JWT.

To get the JWT, use the API details below to send a request with Insomnia.

Method	POST
URL	http://api.mylocal.test**/consumer/login**

Body Format	JSON
Body Content	`{ "username": "user01", "password": "user1234" }`

Looking at the response, you can see that Fusio returned JWT with a JSON data. Select and copy this hash string token.



http://api.mylocal.test/consumer/login

```
{
  "username": "user01",
  "password": "user1234"
}
```

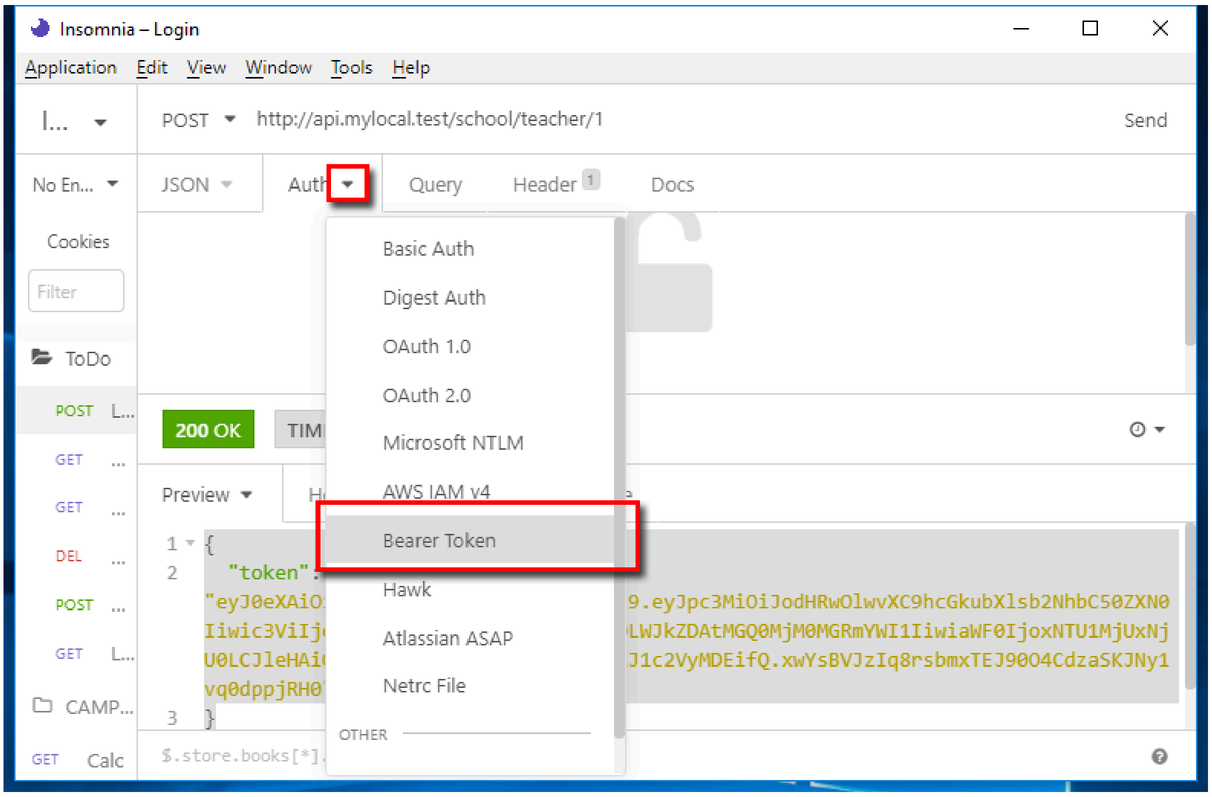
```
{
  "token":
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkubXlsb2NhbC50ZXN0Iiwic3ViIjoiMDNhNzFjZjktNTM0OS01YWY0LWJkZDAtMGQ0MjM0MGRmYWI1IiwiaWF0IjoxNTU1MjUxNjU0LCJleHAiOjE1NTUyNTUyNTQsIm5hbWUiOiJ1c2VyMDEifQ.xwYsBVJzIq8rsbmxTEJ9004CdzaSKJNy1vq0dppjRH0"
}
```

Now that we have the JWT, we can try the previous API once more.

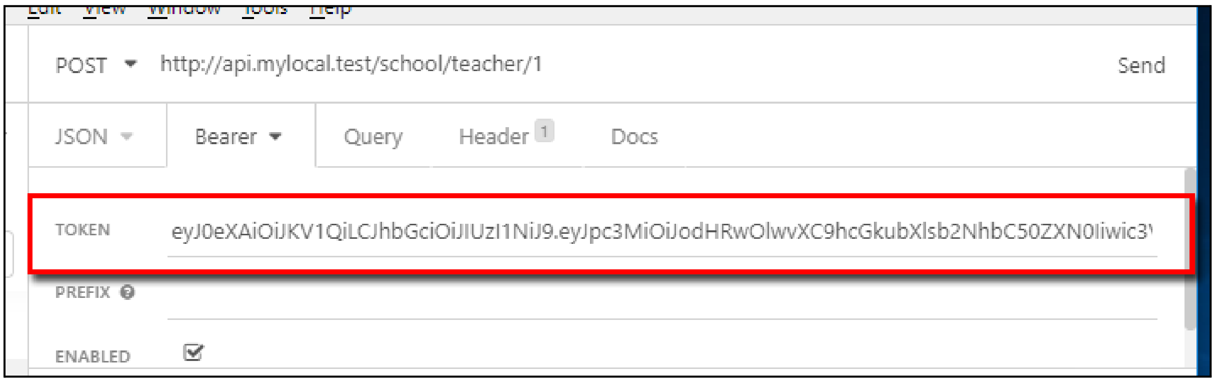
Method	POST
URL	http://api.mylocal.test/school/teacher/1
Body Format	JSON
Body Content	`{ "name_first": "Harry", "name_last": "Harkness" }`
Auth	Bearer Token

Token (example)	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkubXlsb2NhbC50ZXN0Iiwic3ViIjoiMDNhNzFjZjktNTM0OS01YWY0LWJkZDAtMGQ0MjM0MGRmYWI1IiwiaWF0IjoxNTU1MjUxNjU0LCJleHAiOjE1NTUyNTUyNTQsIm5hbWUiOiJ1c2VyMDEifQ.xwYsBVJzIq8rsbmxTEJ9004CdzaSKJNy1vq0dppjRH0

To enter your JWT, click on the **Auth** tab, and on the little triangle to have options appear. Select **Bearer Token**.



Enter the token you got from the previous **/consumer/login** API request in the **Token** field.

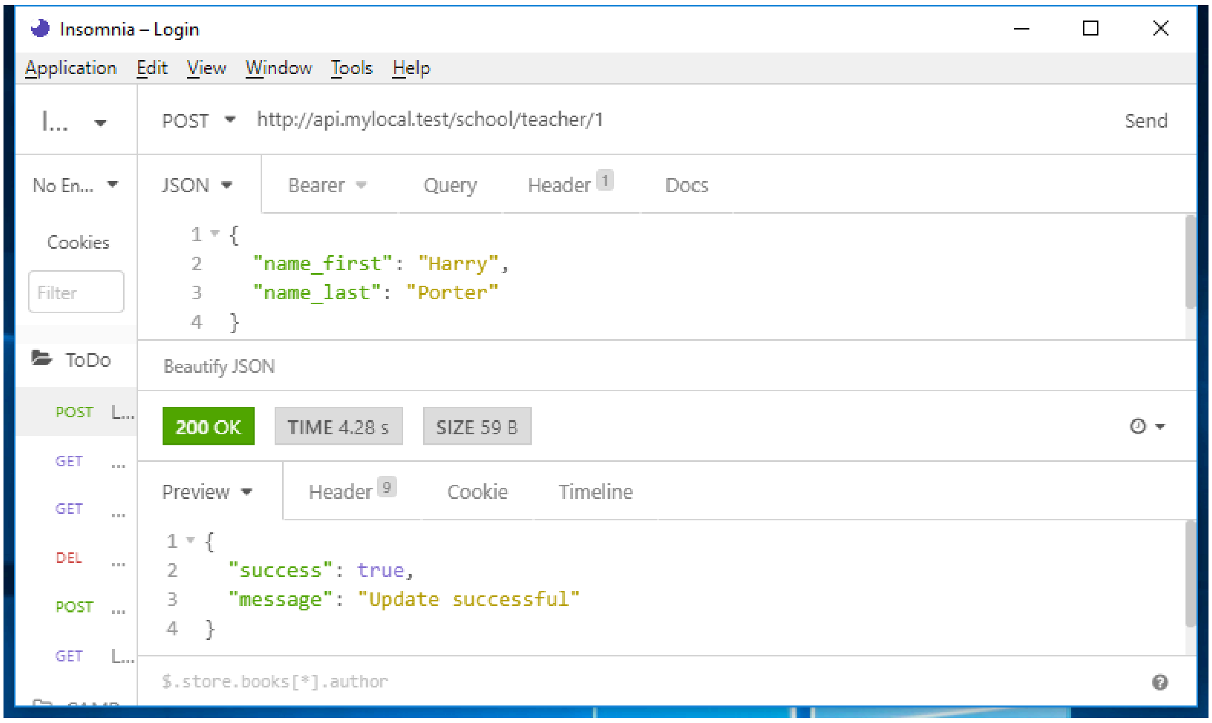


TOKEN eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9hcGkubXlsb2NhbC50ZXN0Iiwic3

PREFIX

ENABLED ✓

When ready, click **Save**.



```
POST http://api.mylocal.test/school/teacher/1
```

```
{
  "name_first": "Harry",
  "name_last": "Porter"
}
```

```
{
  "success": true,
  "message": "Update successful"
}
```

Summary

In this tutorial, you should have learned:

- How to make an API endpoint private and requires authentication
- How to login to Fusio and generate your JSON Web Token (JWT)
- How to access a private API endpoint using your JWT
- How to use Insomnia to include your JWT
- How group API endpoints to Scopes
- How to assign Scopes to users
- How to control user’s access and privileges with Scope

Working with Schema

Fusio comes built in with Schema based of JSONSchema format (http://json-schema.org/).

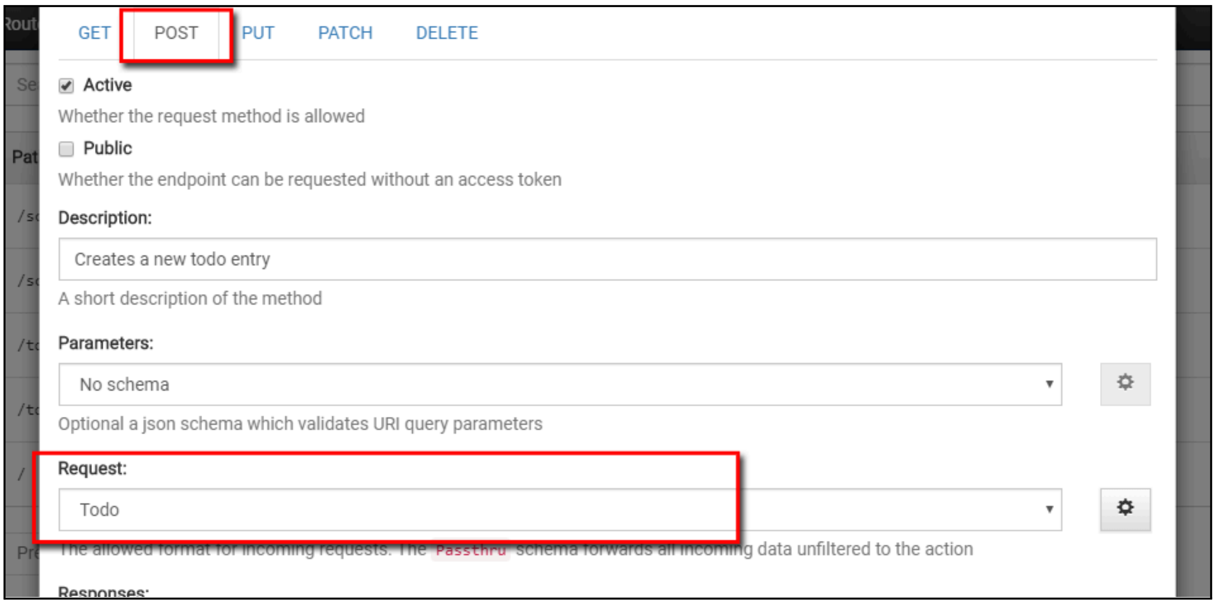
If you have gone through the previous tutorials and created some API endpoints, I am sure that you have encountered Schema. With Schema, you can define how data should be sent to your API endpoint. Schema can also act as a validation rule for your API endpoint. With Schema, it will also help document your API better with the built-in documentation tool.

Let's take a look at the To-Do Sample API and the schema that it has.

To-Do Schema

If we look at **Routes**, the **/todo** route and under the **POST** panel tab, you will find that for the **Request** field, it has **Todo** selected.

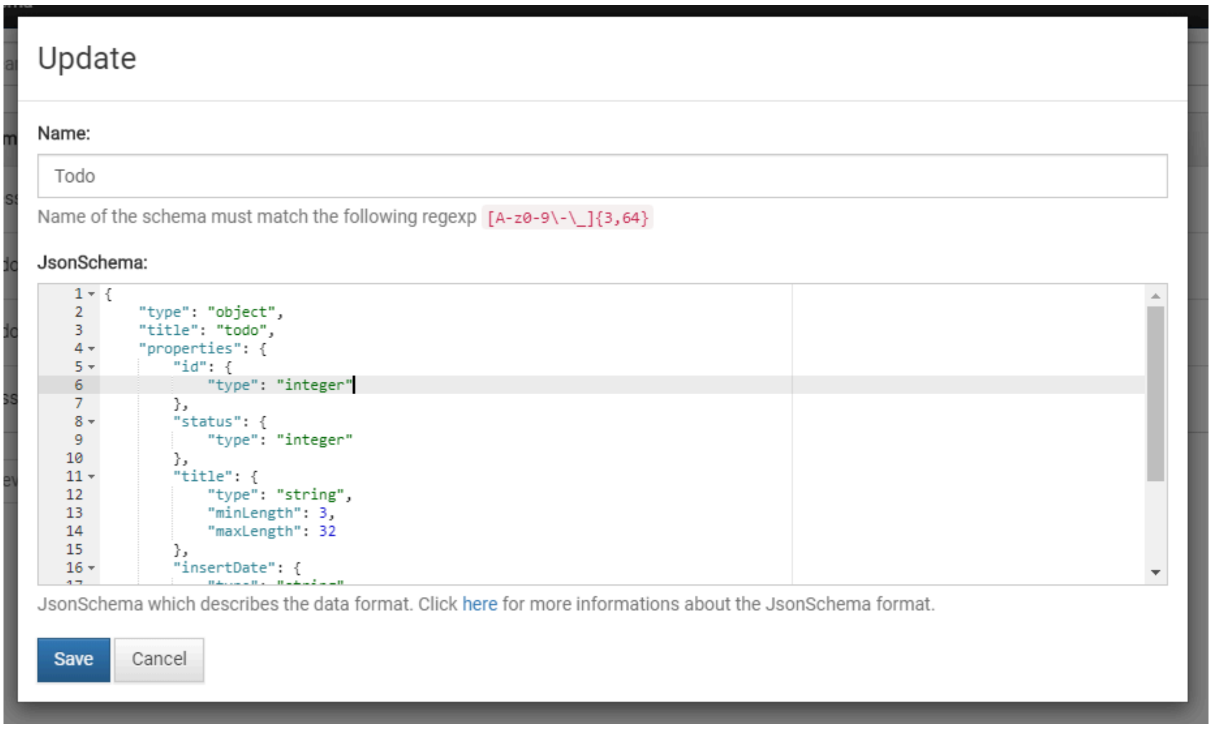
Routes (left menu) > /todo > POST



Active ✓
Whether the request method is allowed
Public
Whether the endpoint can be requested without an access token
Description:
Creates a new todo entry
A short description of the method
Parameters:
No schema
Optional a json schema which validates URI query parameters
Request:
Todo
The allowed format for incoming requests. The Passthru schema forwards all incoming data unfiltered to the action
Responses:

This API endpoint is meant for creating new To-Do item. Therefore, if you were to use this API, it expect that you will be sending it data. And you data must comply with the Todo Schema.

Let's take a look at how this Schema looks like. Head over to Schema from the left menu, then click on the ⚙ gear icon that represents the Todo schema.


Update
Name:
Todo
Name of the schema must match the following regexp [A-z0-9\-_]{3,64}
JsonSchema:
JsonSchema which describes the data format. Click here for more informations about the JsonSchema format.
Save
Cancel

```
{
    "type": "object",
    "title": "todo",
    "properties": {
        "id": {
            "type": "integer"
        },
        "status": {
            "type": "integer"
        },
        "title": {
            "type": "string",
            "minLength": 3,
            "maxLength": 32
        },
        "insertDate": {
```

The JsonSchema field has the following:

```
{
    "type": "object",
    "title": "todo",
    "properties": {
        "id": {
            "type": "integer"
        },
        "status": {
            "type": "integer"
        },
        "title": {
            "type": "string",
            "minLength": 3,
            "maxLength": 32
        },
        "insertDate": {
            "type": "string",
            "format": "date-time"
        }
    },
    "required": [
        "title"
    ]
}
```

Testing the Schema

From a glance, perhaps you can pickup that some of the rules for creating a new Todo item, is that the title is required, and that it must be from 3 to 32 characters long. To see how this works, why don't try to send a request to this API to create a new Todo item.

We will be using Insomnia to send an API request with the following details:

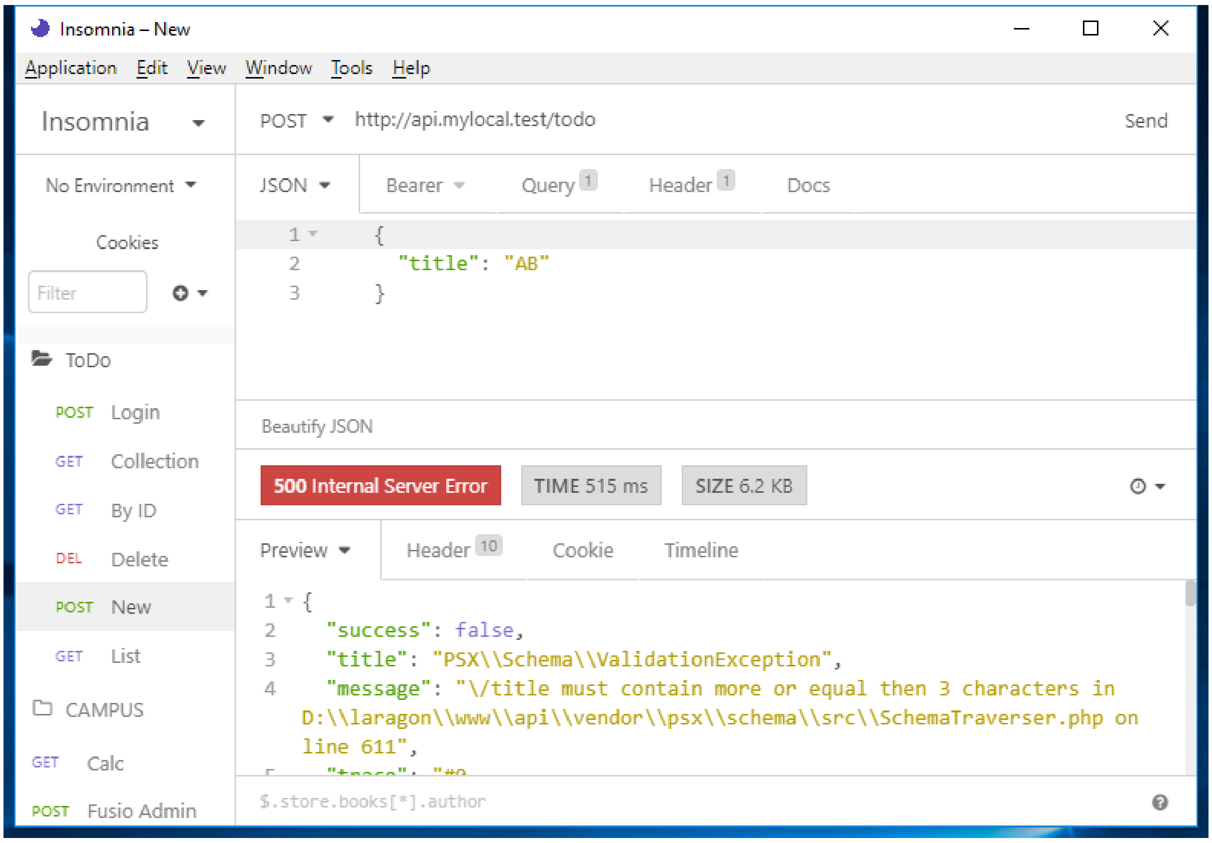
Path	http://api.mylocal.test/todo
Method	POST
Body Format	JSON

Body Content	`{`
	`  "title" : "AB"`
	`}`

Bearer Token	Make an API request to **/consumer/login** with your username and password to get a token

We may need to do a quick request to **/consumer/login** to get a token before sending a request to **/todo**, because this is a private API.

But once you have sent a request to /todo with the details ***with your authentication token***, you should be getting a response like in the image below.



```
POST http://api.mylocal.test/todo
```

```
{
    "title": "AB"
}
```

```
{
  "success": false,
  "title": "PSX\\Schema\\ValidationException",
  "message": "\/title must contain more or equal then 3 characters in D:\\laragon\\www\\api\\vendor\\psx\\schema\\src\\SchemaTraverser.php on line 611",
```

You are getting a 500 error, which the message “title must contain more or equal then 3 characters”. This is a message from Fusio itself after implementing the Schema rule.

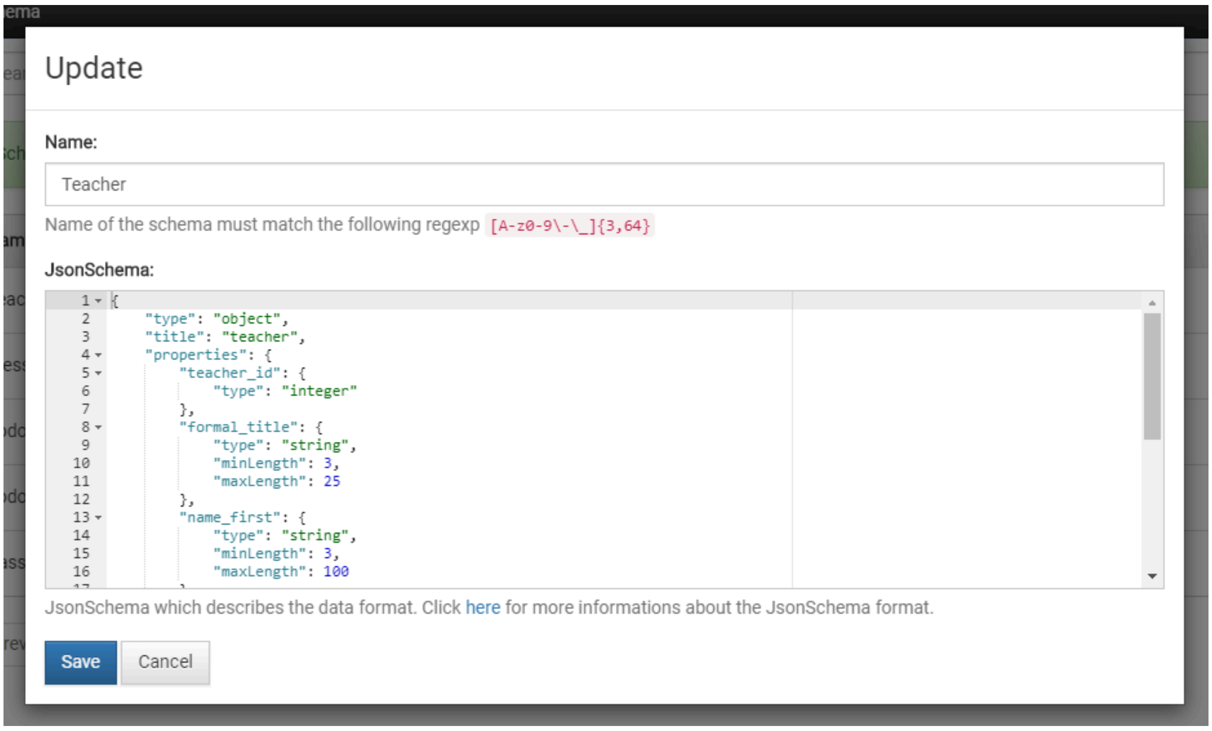
Creating Teacher Schema

We can create our own schema for the **teacher** API. Take a look at the Json Schema below:

```
{
    "type": "object",
    "title": "teacher",
    "properties": {
        "teacher_id": {
            "type": "integer"
        },
        "formal_title": {
            "type": "string",
            "minLength": 3,
            "maxLength": 25
        },
        "name_first": {
            "type": "string",
            "minLength": 3,
            "maxLength": 100
        },
        "name_last": {
            "type": "string",
            "minLength": 3,
            "maxLength": 100
        },
        "email_address": {
            "type": "string",
            "format": "email"
        }
    },
    "required": [
        "name_first",
        "name_last",
        "email_address"
    ]
}
```

Based on the JsonScheme above, let's create our own Schema in Fusio. From the left menu, click on **Schema** and then click on the blue Add New button with the plus icon (+).

Create your new Schema with the above JsonSchema and give it the name **Teacher**. Click **Save** when ready.



Update

Name:

Teacher

Name of the schema must match the following regexp `[A-z0-9\-\_]{3,64}`

JsonSchema:

```
{
    "type": "object",
    "title": "teacher",
    "properties": {
        "teacher_id": {
            "type": "integer"
        },
        "formal_title": {
            "type": "string",
            "minLength": 3,
            "maxLength": 25
        },
        "name_first": {
            "type": "string",
            "minLength": 3,
            "maxLength": 100
```

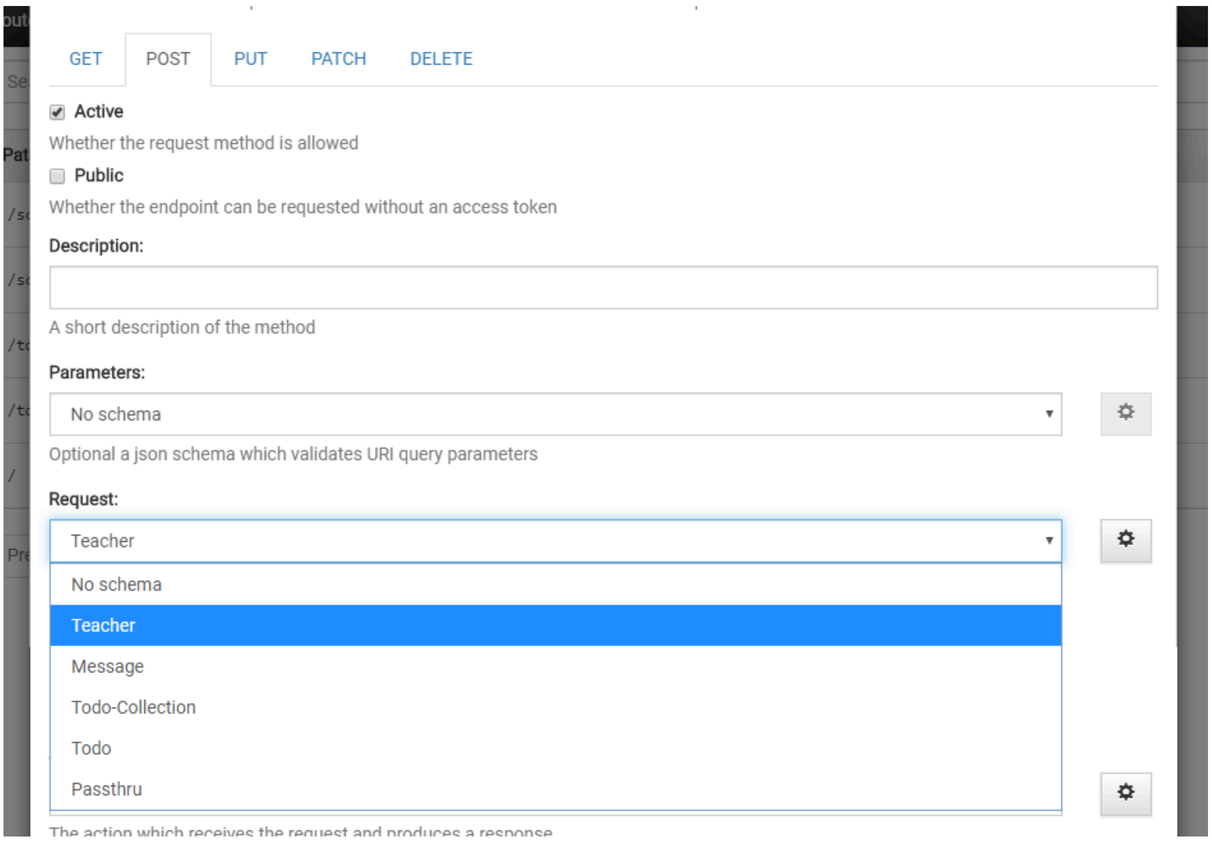
JsonSchema which describes the data format. Click here for more informations about the JsonSchema format.

Now we can start to use this in our /school/teacher/:teacher_id API. Click on **Routes** from the left menu then click on the ⚙ gear icon that represents **/school/teacher/:teacher_id**.

On the modal dialog box, make sure you click on the POST tab panel.

Lower down, you will see a select box for **Request**. It should have **Passthru** selected.

Click and change that to **Teacher**. Click on the **Save** button when ready.



☑ Active
Whether the request method is allowed
☐ Public
Whether the endpoint can be requested without an access token
Description:
A short description of the method
Parameters:
No schema
Optional a json schema which validates URI query parameters
Request:
Teacher

The action which receives the request and produces a response

Now we can try editing a teacher and see if our schema is taking affect.

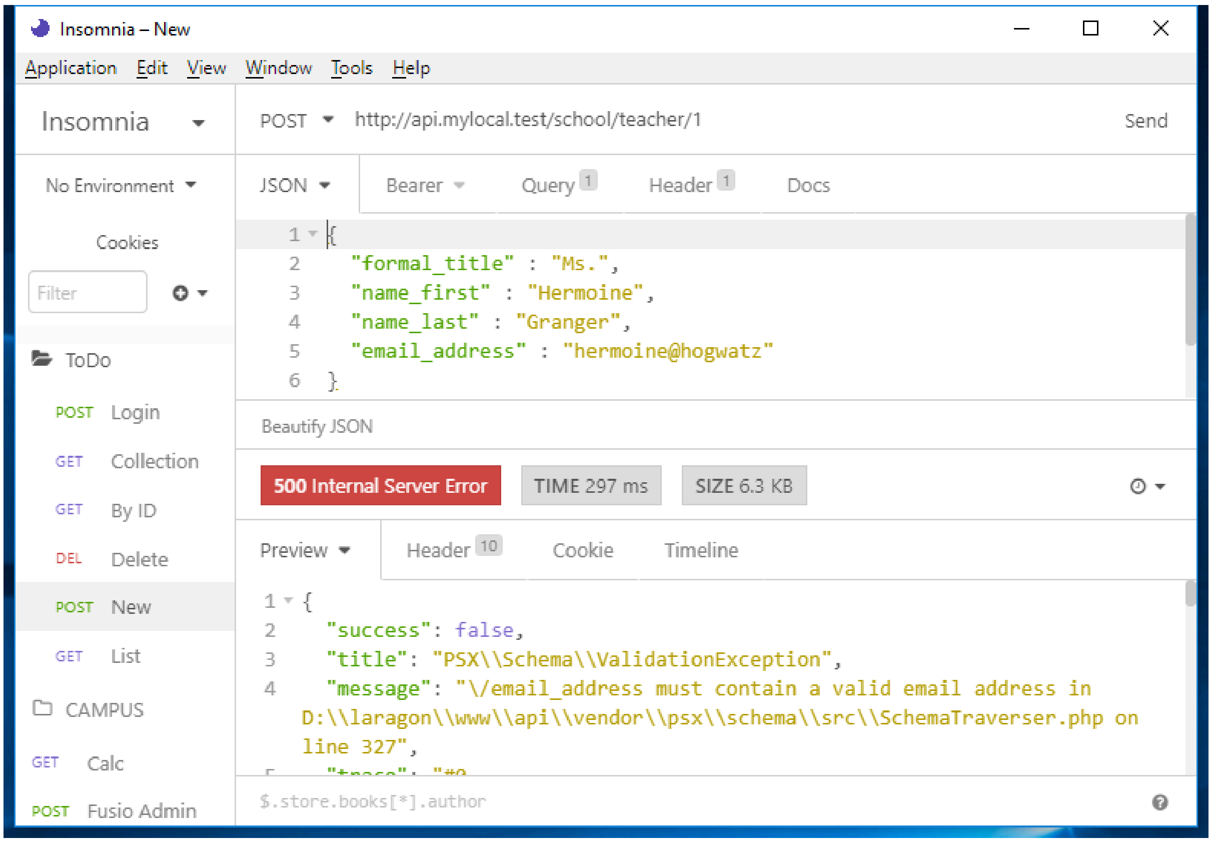
Launch Insomnia. If you don't have a valid token, make sure you make a request to **/consumer/login** to get one.

Path	http://api.mylocal.test/school/teacher/1
Method	POST
Body Format	JSON

Body Content	`{`
	`  "formal_title" : "Ms.",`
	`  "name_first" : "Hermoine",`
	`  "name_last" : "Granger",`
	`  "email_address" : "hermoine@hogwatz"`
	`}`

Bearer Token	Make an API request to **/consumer/login** with your username and password to get a token

Now you should be getting an error message that says, “email_address must contain a valid email address”.



POST http://api.mylocal.test/school/teacher/1

```
{
  "formal_title" : "Ms.",
  "name_first" : "Hermoine",
  "name_last" : "Granger",
  "email_address" : "hermoine@hogwatz"
}
```

```
{
  "success": false,
  "title": "PSX\\Schema\\ValidationException",
  "message": "\/email_address must contain a valid email address in D:\\laragon\\www\\api\\vendor\\psx\\schema\\src\\SchemaTraverser.php on line 327",
```

Make adjustments and see if you can make an update that is accepted.

Summary

In this tutorial, you should have learned:

- What is Schema
- How to create your own Schema
- How to use Schema in you Route

Building API with the SQL-Table

The SQL-Table in **Action** allows use to build API for insert, update, select and delete operation on specific table from our database Connection, without having to write any codes.

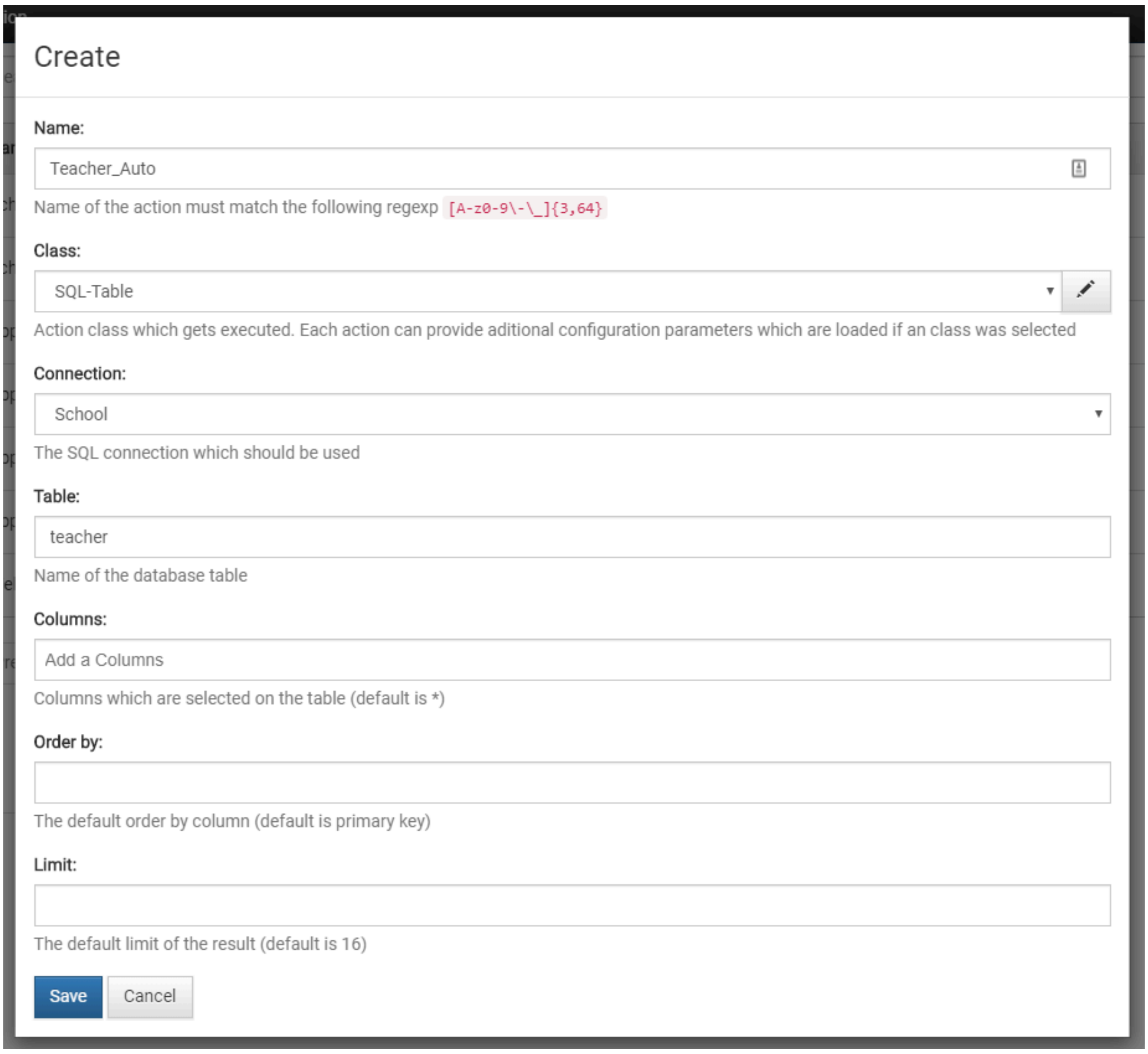
Create an Action with SQL-Table Class

To begin, we start by creating an Action. Click on Action from the left menu and then click on the blue Add New button with the plus icon (+).

In the Create modal dialog, enter the following details.

Name	Teacher_Auto
Class (select dropdown)	SQL-Table
Connection (select dropdown)	School
Table	teachers
Columns (leave blank)	
Order by (leave blank)	
Limit (leave blank)	

Click the **Save** button when ready.



Create

Name:
Teacher_Auto
Name of the action must match the following regexp `[A-z0-9\-\_]{3,64}`

Class:
SQL-Table
Action class which gets executed. Each action can provide aditional configuration parameters which are loaded if an class was selected

Connection:
School
The SQL connection which should be used

Table:
teacher
Name of the database table

Columns:
Add a Columns
Columns which are selected on the table (default is *)

Order by:

The default order by column (default is primary key)

Limit:

The default limit of the result (default is 16)

Create 5 Routes for CRUD Operations

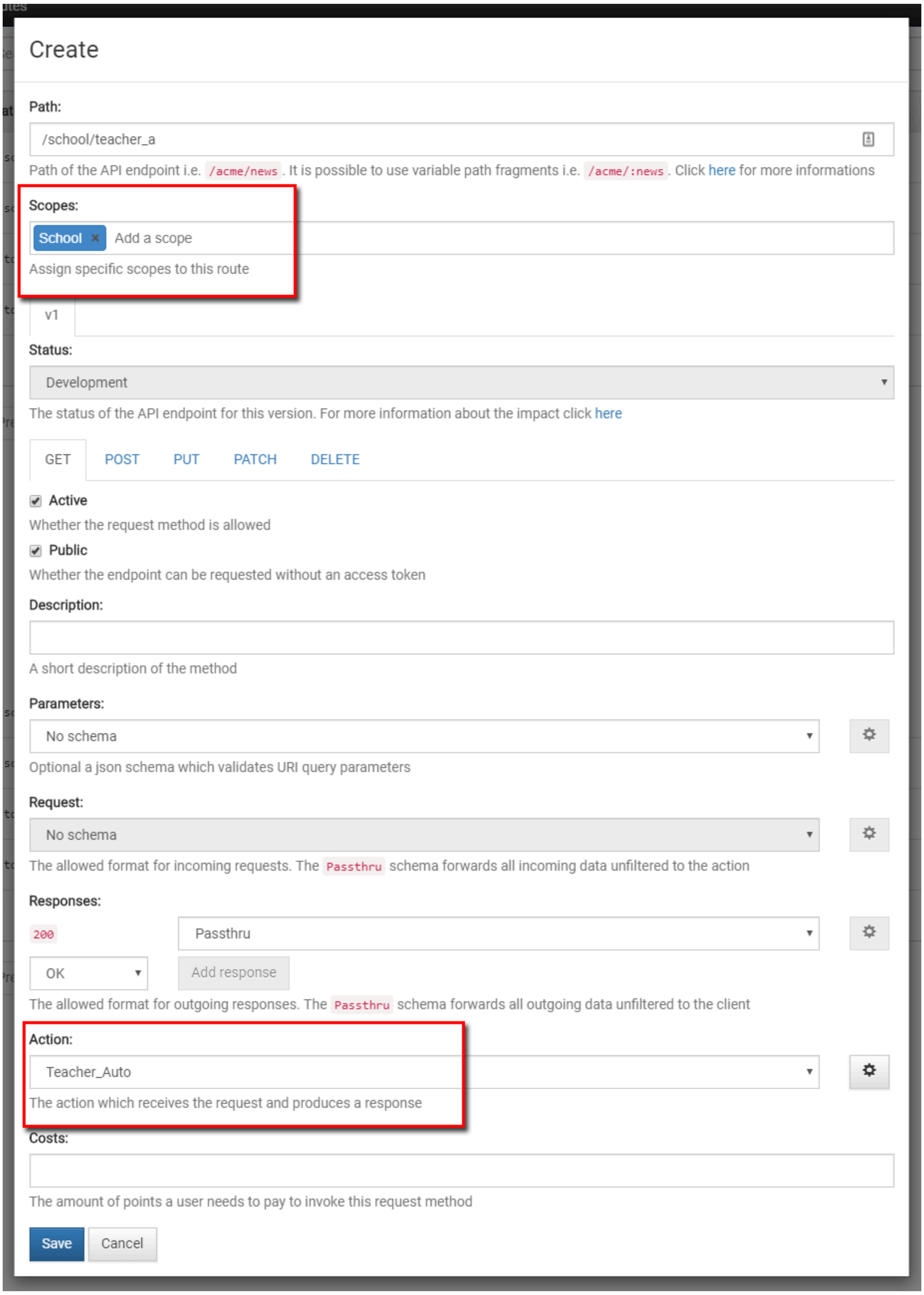
From here onwards, we can straight away create five (5) endpoints to represent the operations that we have to have.

Path	Method	Operation
`/school/teacher_a`	GET	Lists all records
`/school/teacher_a`	POST	Insert a new record
`/school/teacher_a/:id`	GET	Find a record
`/school/teacher_a/:id`	PUT	Edit a record
`/school/teacher_a/:id`	DELETE	Delete a record

To create these, we will be creating two Routes and activate the required methods. However, for all of these endpoints, the Action is the same -- Teacher_Auto.

You can also add a Scope to the Routes. I recommend adding all these to **School** scope. This way, all users that have this Scope will also be granted permission to this new API.

Those that require data submission like POST and PUT, be sure to set the **Request** field to **Teacher** or **Passthru**.



Create

Path:

/school/teacher_a

Path of the API endpoint i.e. `/acme/news`. It is possible to use variable path fragments i.e. `/acme/:news`. Click here for more informations

Scopes:

School × Add a scope

Assign specific scopes to this route

v1

Status:

Development

The status of the API endpoint for this version. For more information about the impact click here

GET POST PUT PATCH DELETE

☑ Active

Whether the request method is allowed

☑ Public

Whether the endpoint can be requested without an access token

Description:

A short description of the method

Parameters:

No schema

Optional a json schema which validates URI query parameters

Request:

No schema

The allowed format for incoming requests. The `Passthru` schema forwards all incoming data unfiltered to the action

Responses:

`200` Passthru

OK Add response

The allowed format for outgoing responses. The `Passthru` schema forwards all outgoing data unfiltered to the client

Action:

Teacher_Auto

The action which receives the request and produces a response

Costs:

The amount of points a user needs to pay to invoke this request method

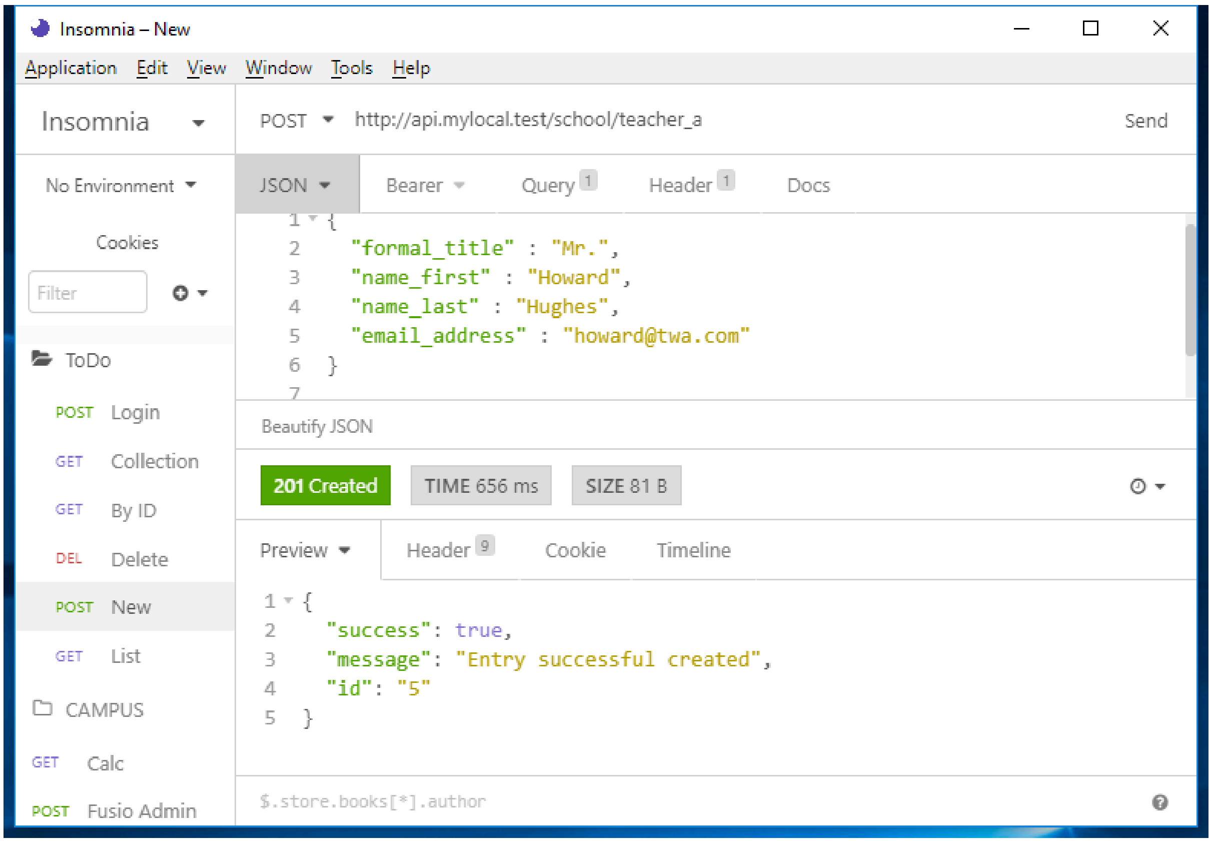
Save Cancel

Testing Your API

If you have created all the necessary Routes and endpoint, it is time to test this out. We will be creating a new record to test this API. Launch Insomnia and make a request to the API with the details below.

Path	http://api.mylocal.test/school/teacher_a
Method	POST
Body Format	JSON
Body Content	`{ "formal_title" : "Mr.", "name_first" : "Howard", "name_last" : "Hughes", "email_address" : "howard@twa.com" }`
Bearer Token	Make an API request to **/consumer/login** with your username and password to get a token

If all goes well, you should get a response like in the image below. Pay attention to the **"id"** parameter that is being sent back from the API.



POST http://api.mylocal.test/school/teacher_a

```
{
   "formal_title" : "Mr.",
   "name_first" : "Howard",
   "name_last" : "Hughes",
   "email_address" : "howard@twa.com"
}
```

201 Created TIME 656 ms SIZE 81 B

```
{
  "success": true,
  "message": "Entry successful created",
  "id": "5"
}
```

$.store.books[*].author

Try out the other API endpoints and see if they work.

Summary

In this tutorial, you should have learned

- How to use the SQL-Table to expose CRUD operations to a specific table
- How to create the Routes and API endpoint required for your API