MICRO-SERVICE CONFIGURATION WITH API SECURITY

DDS Development Environment Requirements

1. Create a Github Account
2. Install Webserver (Apache)
3. Install MySQL
4. Install Lumen (Setup Backend Environment (Laravel -> Lumen))
5. Install Vue.js for Front End
6. Create Project
    1. Create Connection to MySQL DB
    2. Create Model
    3. Create Controller
    4. Create View
7. Test your Project using Postman
8. Push your Folder to Github

Setup Lumen Laravel for Backend

A) Install Composer

1. https://getcomposer.org/
2. click Getting started
3. Select Globally
4. **For MAC OS**
    1. Download stable composer.phar
    2. Type mv composer.phar /usr/local/bin/composer
5. **For Windows**
    1. Select the executable installer after clicking **Getting started**
6. To test, goto terminal and type **composer -v**

B) Install Laravel Lumen by Creating a Project

https://lumen.laravel.com/docs/8.x

1. Goto the directory where you want to put your lumen project folder
2. type
   > **composer create-project --prefer-dist laravel/lumen ddsbe**
3. copy the project_folder(ddsbe) to the location of your Apache Document Root i.e. in apache **htdocs**
4. You can also run your project using this php command

> **php -S localhost:8000 -t public**

C) Open your IDE, example Visual Studio, then add your workspace folder (ddsbe)

D) Connect to MySQL DB, Create Model, Controller, View in Lumen

1. Generate App Key
    1. Goto http://www.unit-conversion.info/texttools/random-string-generator/
    2. copy the string to
        1.
2. Change the .env for MYSQL
    1. DB_CONNECTION=mysql
    2. DB_HOST=127.0.0.1
    3. DB_PORT=2206
    4. DB_DATABASE=ddsbe
    5. DB_USERNAME=root
    6. DB_PASSWORD=1234
3. Goto Folder **bootstrap -> app.php**
    1. uncomment the below lines to enable the main components of lumen

```
// $app->withFacades();
// $app->withEloquent();
```

3. Goto app folder
    1. Create you Model
        1. New File User.php

```
<?php

    namespace App;

    use Illuminate\Database\Eloquent\Model;
    class User extends Model{
        protected $table = 'tbluser';
       // column sa table
        protected $fillable = [
           'username', 'password'
        ];
    }
```

        2. To Put all yours Models inside Model Folders
            1. change your Model namespace to namespace App\Models;
            2. Also change the namespace of UserController.php controller > use App\Models\User;

    2. Create your Controller
        1. Goto **Http -> Controllers**
        2. Create File **UserController.php**

3. Type

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\User;

Class UserController extends Controller {
   private $request;

   public function __construct(Request $request){
      $this->request = $request;
   }

   public function getUsers(){
      $users = User::all();
      return response()->json($users, 200);
        }
     }

    /**
    * Return the list of users
    * @return Illuminate|Http|Response
    */
   public function index()
   {
      $users = User::all();
      return $this->successResponse($users);

   }

  public function add(Request $request ){
        $rules = [
           'username' => 'required|max:20',
           'password' => 'required|max:20',
           'gender' => 'required|in:Male,Female',
        ];

        $this->validate($request,$rules);

        $user = User::create($request->all());
```

```
        return $this->successResponse($user,
Response::HTTP_CREATED);
    }

    /**
     * Obtains and show one user
     * @return Illuminate|Http|Response
     */
    public function show($id)
    {

         $user = User::findOrFail($id);
         return $this->successResponse($user);

        // old code
        /*
        $user = User::where('userid', $id)->first();
        if($user){
            return $this->successResponse($user);
        }
        {
            return $this->errorResponse('User ID Does Not Exists',
Response::HTTP_NOT_FOUND);
        }
         */
    }

    /**
     * Update an existing author
     * @return Illuminate|Http|Response
     */
    public function update(Request $request,$id)
    {
        $rules = [
        'username' => 'max:20',
        'password' => 'max:20',
        'gender' => 'in:Male,Female',
        ];

        $this->validate($request, $rules);
        $user = User::findOrFail($id);

        $user->fill($request->all());
```

```
        // if no changes happen
        if ($user->isClean()) {
            return $this->errorResponse('At least one value must
change',        Response::HTTP_UNPROCESSABLE_ENTITY);
        }

        $user->save();
        return $this->successResponse($user);

        // old code
        /*
            $this->validate($request, $rules);
            //$user = User::findOrFail($id);
            $user = User::where('userid', $id)->first();
            if($user){
                $user->fill($request->all());

                // if no changes happen
                if ($user->isClean()) {
                    return $this->errorResponse('At least one value must
change', Response::HTTP_UNPROCESSABLE_ENTITY);
                }

                $user->save();
                return $this->successResponse($user);
            }
            {
                return $this->errorResponse('User ID Does Not Exists',
Response::HTTP_NOT_FOUND);
            }
        */
    }

     /**
     * Remove an existing user
     * @return Illuminate|Http|Response
     */
    public function delete($id)
    {

        $user = User::findOrFail($id);
        $user->delete();
```

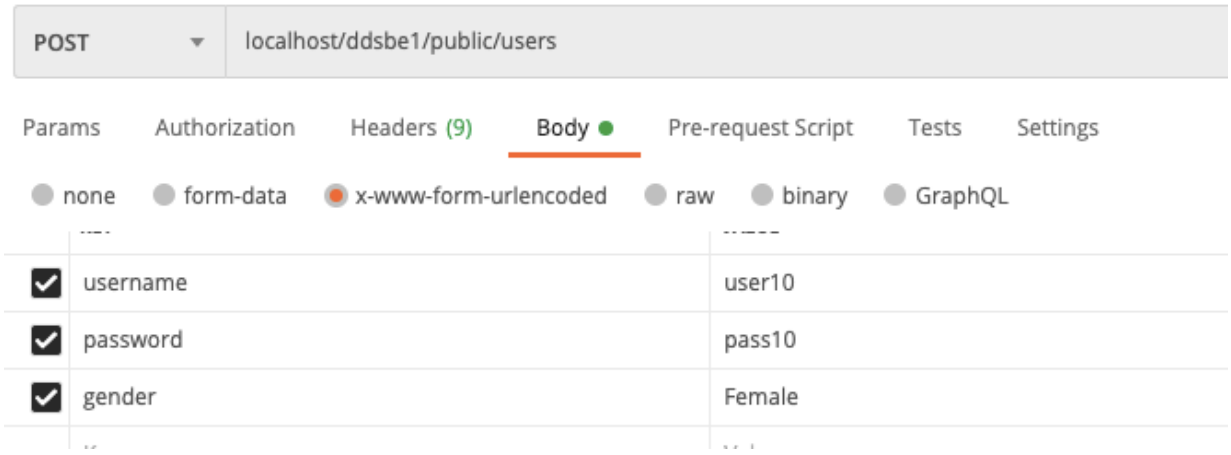
```
        return $this->successResponse($user);
        // old code
        /*
        $user = User::where('userid', $id)->first();
        if($user){
            $user->delete();
            return $this->successResponse($user);
        }
        {

            return $this->errorResponse('User ID Does Not Exists',
    Response::HTTP_NOT_FOUND);
        }
        */
    }
```

3. Create View (Routes (API Link))
    1. Goto **routes -> web.php**

```
<?php

/** @var \Laravel\Lumen\Routing\Router $router */

/*
|--------------------------------------------------------------------------
| Application Routes
|--------------------------------------------------------------------------
|
| Here is where you can register all of the routes for an application.
| It is a breeze. Simply tell Lumen the URIs it should respond to
| and give it the Closure to call when that URI is requested.
|
*/

$router->get('/', function () use ($router) {
    return $router->app->version();
});

// unsecure routes
$router->group(['prefix' => 'api'], function () use ($router) {
    $router->get('/users',['uses' => 'UserController@getUsers']);
```

```
});

// more simple routes
$router->get('/users',['uses' => 'UserController@getUsers']);
$router->get('/users', 'UserController@index');   // get all users records
$router->post('/users', 'UserController@add');  // create new user record
$router->get('/users/{id}', 'UserController@show'); // get user by id
$router->put('/users/{id}', 'UserController@update'); // update user record
$router->patch('/users/{id}', 'UserController@update'); // update user record
$router->delete('/users/{id}', 'UserController@delete'); // delete record
```

4. **To test our backend API that will return all tblusers record in json format**
    1. open your Postman
        1. **Test view all users API**
            1. select GET as type of request
            2. type link: localhost/**ddsbe**/public/api/users
            3. then click Send
        2. Test Create New User API
            1. select POST as type of request
            2. select Body
                1. Enter the Fieldname and Value



2. type localhost/**ddsbe**/public/api/users
3. Click Send

Zoom Video is posted in our USTEP account

E. Putting your project to github

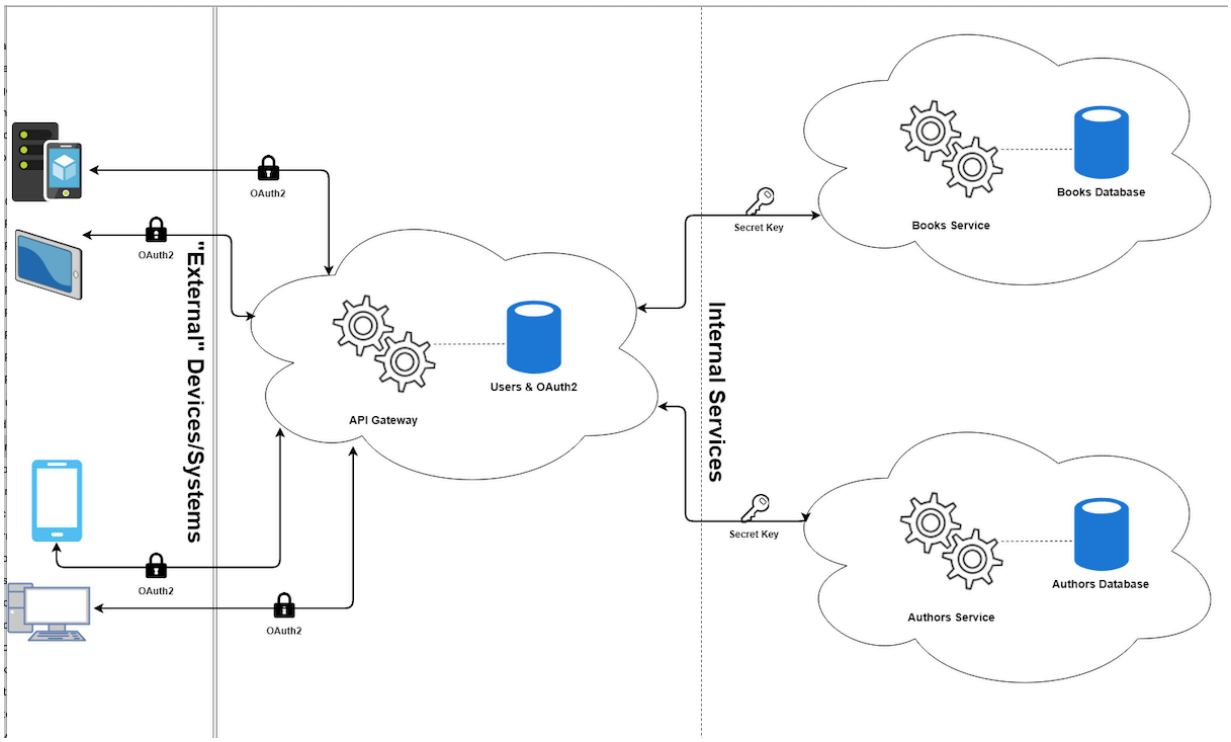
Option 1: By Putting your Folder to Github Repo

1. > create new repository in GitHub
2. > git init
3. > git remote remove origin —> (optional)
4. > git remote add origin https://github.com/cydarvs/crwn-clothing.git <— this should be your repo link
5. > git status
6. > git add -A
7. > git commit -m “First Push to Repository”
8. > git push origin master

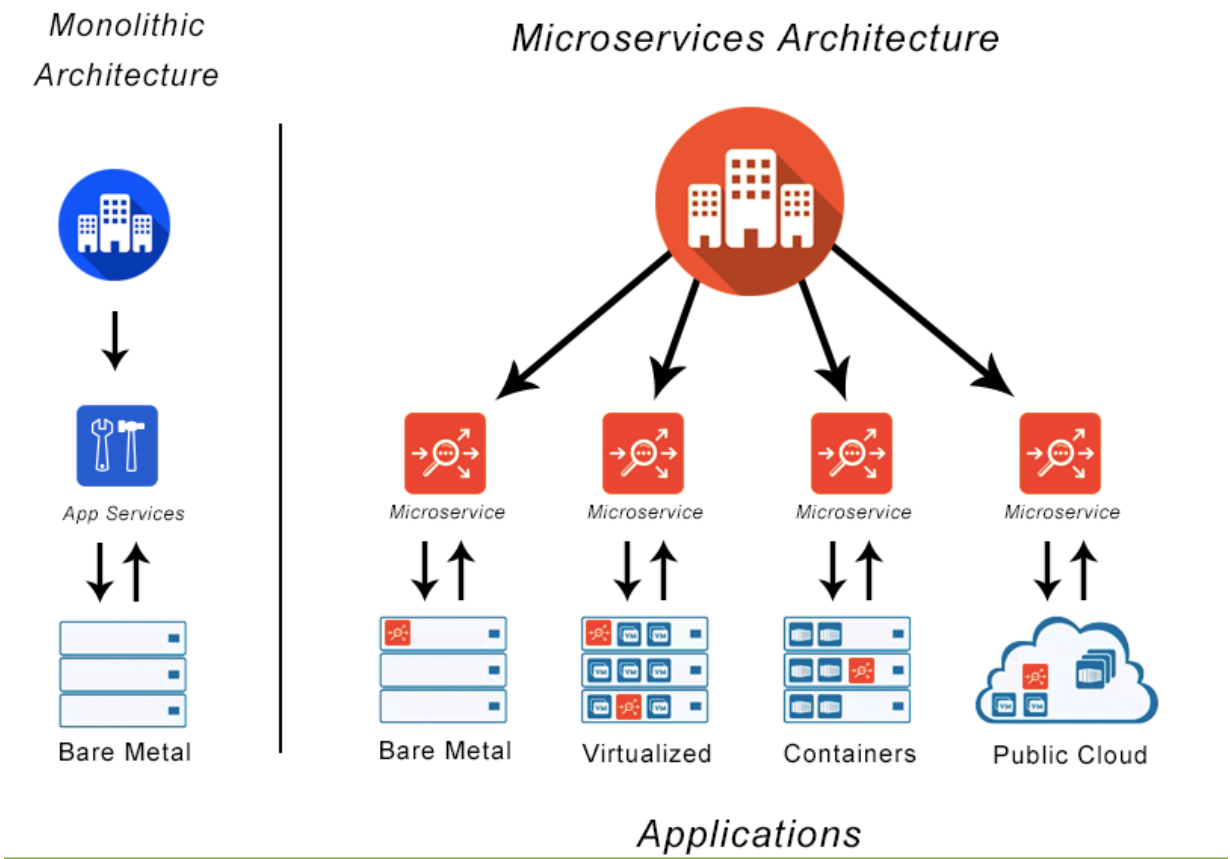
Option2: By Cloning

> git clone https://github.com/cydarvs/ddsbe.git
> cd to the directory folder
> make changes to directory
> git status <— know all changes
> git add -A
> git commit -m “First Push to Repository”
> git push origin master

F. Know the Lumen Architecture



G. Microservices Architecture



H. USING LUMENT TRAIT

What is a Trait

Traits are a handy way of reusing code in PHP, another way to help keep your code DRY (or "don't repeat yourself").
If you've used whats commonly referred to as Mixins in some languages and frameworks, you'll be probably be familiar with Traits.
It's great that we can write some code, reuse that again and again throughout our codebase more efficiently,
meanwhile – It's good practice and helps us maintain the codebase easier, so yay for Laravel Traits / PHP Traits!

Adding Traits in Lumen

1. Add Traits Folder inside app folder in Lumen
2. Create a filename ApiResponser.php
    1. **Type this code**

```
<?php

namespace App\Traits;

use Illuminate\Http\Response;

trait ApiResponser
{
    /**
     * Build success response
     * @param  string|array $data
     * @param  int $code
     * @return Illuminate|Http|JsonResponse
     */
    public function successResponse($data, $code = Response::HTTP_OK)
    {
        return response()->json(['data' => $data], $code);
    }

    /**
     * Build error responses
     * @param  string|array $message
     * @param  int $code
     * @return Illuminate|Http|JsonResponse
     */
    public function errorResponse($message, $code)
```

```
    {
        return response()->json(['error' => $message, 'code' => $code],
$code);
    }
}
```

3. Use the Traits (ApiResponser.php in our controller)
    1. Declare the namespace of ApiResponser
    2. add use ApiResponser inside the UserController Class definition
    3. See below

4. **Use the responser method**

```php
public function index()
{
    $users = User::all();
    return $this->successResponse($users);
    // old code
    /*
    // return $users; // <-- not standardized return of data
    // return $this->successResponse($users);
    // return response()->json($users, 200);
    */

}
```

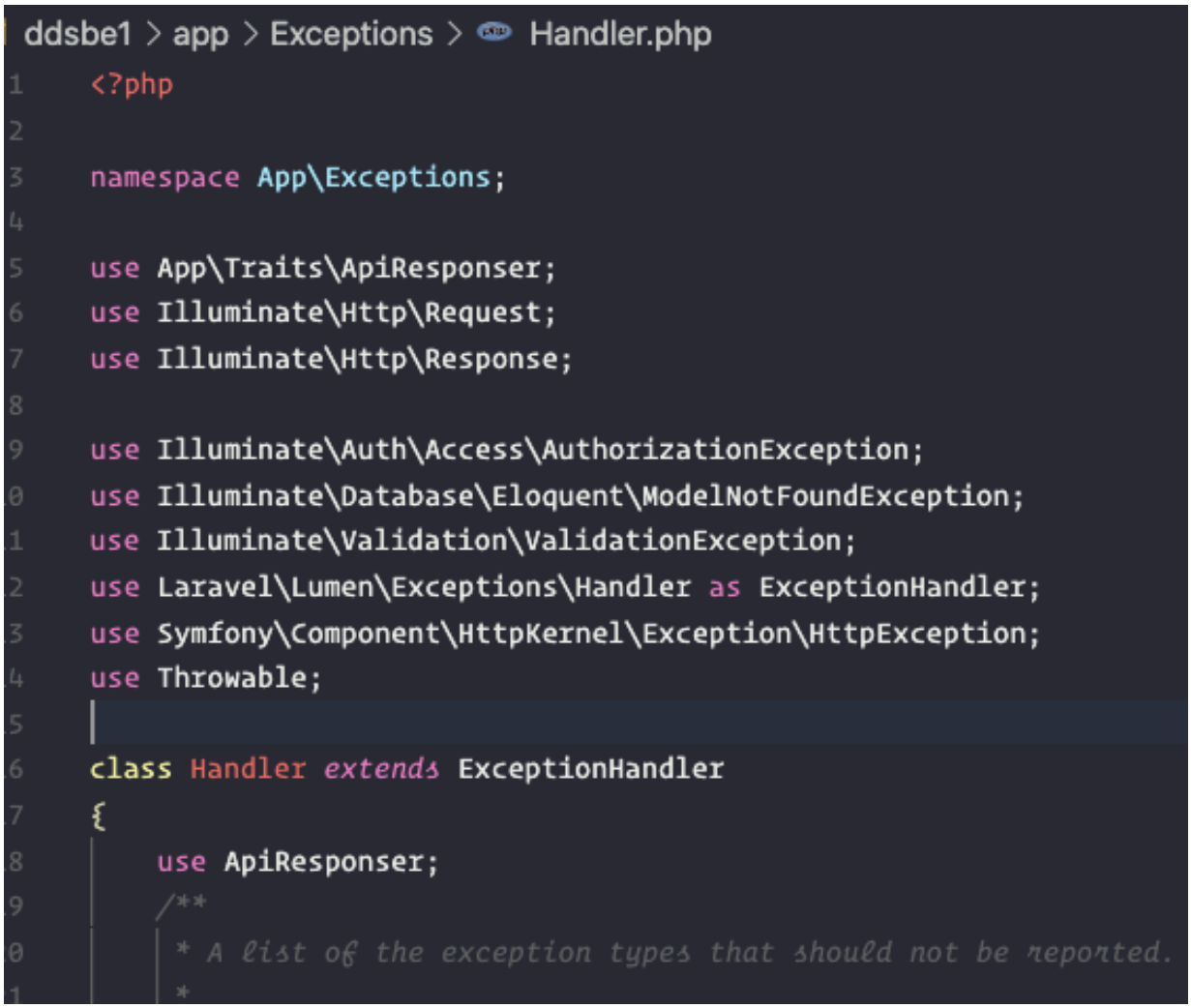
Using the successReponse

```php
public function add(Request $request ){
    $rules = [
        'username' => 'required|max:20',
        'password' => 'required|max:20',
        'gender' => 'required|in:Male,Female',
    ];

    $this->validate($request,$rules);

    $user = User::create($request->all());

    return $this->successResponse($user, Response::HTTP_CREATED);
}
```

H. CUSTOMIZING EXCEPTIONS HANDLING

1. Goto app -> Exceptions Folder
2. Open Handler.php
    1. Add this code in the definition
        > use App\Traits\ApiResponser;
        > use Illuminate\Http\Request;
        > use Illuminate\Http\Response;

2. Add this code inside the Handler Class
   > use ApiResponser;
3. See picture below



4. Edit the code function —> *public* function render($request, Throwable $exception)

```
/**
 * Render an exception into an HTTP response.
 *
 * @param  |Illuminate|Http|Request  $request
 * @param  |Throwable  $exception
 * @return |Illuminate|Http|Response||Illuminate|Http|JsonResponse
 *
 * @throws |Throwable
 */
public function render($request, Throwable $exception)
```

```
    {
        // http not found
        if ($exception instanceof HttpException) {
            $code = $exception->getStatusCode();
            $message = Response::$statusTexts[$code];

            return $this->errorResponse($message, $code);
        }
        // instance not found
        if ($exception instanceof ModelNotFoundException) {
            $model = strtolower(class_basename($exception->getModel()));

            return $this->errorResponse("Does not exist any instance of
{$model} with the given id",
                    Response::HTTP_NOT_FOUND);
        }

        // validation exception
        if ($exception instanceof ValidationException) {
            $errors = $exception->validator->errors()->getMessages();

            return $this->errorResponse($errors,
Response::HTTP_UNPROCESSABLE_ENTITY);
        }

        // access to forbidden
        if ($exception instanceof AuthorizationException) {
            return $this->errorResponse($exception->getMessage(),
Response::HTTP_FORBIDDEN);
        }

        // unauthorized access
        if ($exception instanceof AuthenticationException) {
            return $this->errorResponse($exception->getMessage(),
Response::HTTP_UNAUTHORIZED);
        }

        // if your are running in development environment
        if (env('APP_DEBUG', false)) {
            return parent::render($request, $exception);
        }

        return $this->errorResponse('Unexpected error. Try later',
```

```
            Response::HTTP_INTERNAL_SERVER_ERROR);
        }
```

5. **You can try to test again your API's using POSTMAN**

I. SETTING YOUR SECOND MICROSERVICE API (SITE 2)

1. clone your first USER API
2. Rename the folder to **DDSBE2** (for site2)
3. Goto to DDSBE2
4. Update the library requirements of **SITE2** of PHP using command > **composer install**
5. Change the Database Connection in your .env file
6. Change the APP_KEY in your .env file using STRING GENERATOR: http://www.unit-conversion.info/texttools/random-string-generator/
7. Test your SITE2 MICRO-SERVICE using POSTMAN and make all API (**CRUD**) routes are running as expected
8. Put your ddsbe2 folder to your Github Account
    1. > Create first a new repo for the site2
    2. > git init
    3. > git remote remove origin —> (optional)
    4. > git remote add origin https://github.com/cydarvs/ddsbe2.git <— this should be your repo link
    5. > git status

6. > git add -A
7. > git commit -m “First Push to Repository”
8. > git push origin master

J. SETTING YOUR API-GATEWAY MICROSERVICE

1. create a new lumen project for api-gateway
    1. > **composer create-project --prefer-dist laravel/lumen ddsgateway**
2. Put your api gateway folder ddsgateway to your GitHub account
    1. > Create first a new repo for the site2
    2. > git init
    3. > git remote remove origin —> (optional)
    4. > git remote add origin https://github.com/cydarvs/ddsgateway.git <— this should be your repo link
    5. > git status
    6. > git add -A
    7. > git commit -m “First Push to Repository”
    8. > git push origin master
3. Add the two controller, User1Controller of Site1 and User2Controller Site2 Micro-Service to your api gateway
    1. > You can copy and paste the controllers inside app -> Http -> Controllers
    2. > Clear all the codes inside the Controller Methods/Functions
4. Add trait ApiResponser to the api gateway
    1. change the code successResponse

```
public function successResponse($data, $code = Response::HTTP_OK)
    {
        // old code
        // return response()->json(['data' => $data, 'site' => 1], $code);
        // this code is changes since the message to return is already formatted by API responser of each site
        return response($data, $code)->header('Content-Type', 'application/json');
    }
```

    2. change the code of errorReponse
        1. *// error response generated by api gateway itself*

public function errorResponse($message, $code)
{
return response()->json(['error' => $message, 'code' => $code], $code);
}

3. add Error Message return by API gateway request
    1. Add errorMessage will return message generate by our API's
    public function errorMessage($message, $code)
    {
    return response($message, $code)->header('Content-Type', 'application/json');
    }

5. Add a php library that will consume all request from external user to our API Gateway
    1. Install Guzzle, **Guzzle is a PHP HTTP client that makes it easy to send HTTP requests and trivial to integrate with web services. (https://github.com/guzzle/guzzle)**
        1. > goto to your api gateway project folder
        2. > type **composer require guzzlehttp/guzzle**
        3. Create the configuration files of guzzle services
            1. create a folder name **config** inside your api folder
            2. create a file name **services.php** inside your config folder
                1. type this code (for now)
                2. <?php

                return [

                ];
                3. **Register you newly created config file**
                    1. Goto bootstap -> app.php
                    2. Add this line after $app -> withEloquent();
                        1. **> $app->configure('services');**
                4. Make our api have the capability to use the Guzzle by creating a Trait (for Code Reusability)
                    1. Goto **app** -> **Traits** Folder
                    2. Create a filename -> **ConsumesExternalService.php**
                    3. Type this code

```
<?php

namespace App\Traits;

// include the Guzzle Component Library
```

```
use GuzzleHttp\Client;

trait ConsumesExternalService
{
   /**
    * Send a request to any service
    * @return string
   */
   // note form params and headers are optional
   public function performRequest($method,
$requestUrl,$form_params =[],$headers =[])
   {
      // create a new client request
      $client = new Client([
         'base_uri' => $this->baseUri,
      ]);

      // perform the request (method, url, form
parameters, headers)
      $response = $client-
>request($method,$requestUrl,
['form_params' =>
         $form_params, 'headers' =>
$headers]);

      // return the response body contents
      return $response->getBody()->getContents();

   }

}
```

3. **Preparing the Lumen components to consume the internal services**
   1. Goto config -> services.php
   2. Edit the code

```
<?php

return [
   'users1' => [
     'base_uri' =>
```

```
env('USERS1_SERVICE_BASE_URL'
),
    ],

    'users2' => [
        'base_uri' =>
env('USERS2_SERVICE_BASE_URL'
),
    ],
];
```

3. **Goto your .env file to define the two base_uri**
    1. Add this code
        1. **USERS1_SERVICE_BASE_URL**=localhost:8000 <— this is your server port for Site1
        2. **USERS2_SERVICE_BASE_URL**=localhost:8001 <— this is your server port for Site2
4. **To represent the the two services we will create services**
    1. Goto **app** folder
    2. Create Folder -> **Services**
    3. Create a file **User1Service.php**
        1. Write this code

```
<?php

namespace App\Services;

use App\Traits\ConsumesExternalService;

class User1Service{

    use ConsumesExternalService;

    /**
     * The base uri to consume the User1 Service
     * @var string
     */
    public $baseUri;
```

```
    public function __construct()
    {
        $this->baseUri = config('services.users1.base_uri');
    }

}
```

4. **Copy the User1Service.php to Create User2.Service.php and change the classname and baseUri**
    1. Write this code for User2Service.php

```
<?php

namespace App\Services;

use App\Traits\ConsumesExternalService;

class User2Service{

    use ConsumesExternalService;

    /**
     * The base uri to consume the User2 Service
     * @var string
     */
    public $baseUri;

    public function __construct()
    {
        $this->baseUri = config('services.users2.base_uri');
    }
}
```

5. **Lets make our api gateway controller to be able to use the services we created to perform request to two microservices**

1. We will add the dependency injection of our two controller
2. For User1Controller
    1. Goto Http -> User1Controller
    2. **Add this code below use ApiResponser;**

```
/**
     * The service to consume the User1 Microservice
     * @var User1Service
     */
    public $user1Service;

    /**
     * Create a new controller instance
     * @return void
     */

    public function __construct(User1Service $user1Service){
        $this->user1Service = $user1Service;
    }
```

    3. Make sure that you add your service definition
        1. **> use App\Services\User1Service;**
    4. Do the step 1 to 3 in our User2Controller

4. Implementing the functions of the Gateway with Lumen

1. Get all the lists of users in Site1

1. Goto api gateway User1Controller -> Http -> User1Controller.php
2. Edit the method index()

```
<?php

namespace App\Http\Controllers;


use Illuminate\Http\Response; // Response Components
use App\Traits\ApiResponser;  // <-- use to standardized our code for api response
```

```
use Illuminate\Http\Request;  // <-- handling http request in lumen
use App\Services\User1Service; // user1 Services


Class User1Controller extends Controller {
   // use to add your Traits ApiResponser
   use ApiResponser;

   /**
    * The service to consume the User1 Microservice
    * @var User1Service
    */
   public $user1Service;

   /**
    * Create a new controller instance
    * @return void
    */
   public function __construct(User1Service $user1Service){
       $this->user1Service = $user1Service;
   }

   /**
    * Return the list of users
    * @return Illuminate|Http|Response
    */
   public function index()
   {
     //
      return $this->successResponse($this->user1Service->obtainUsers1());
   }
}
```

3. Let put functionality of our **obtainUsers1() method** found in our Services -> **User1Service.php**
    1. Our code as of now

```
<?php

namespace App\Services;
```

```
use App\Traits\ConsumesExternalService;

class User1Service{

    use ConsumesExternalService;

    /**
     * The base uri to consume the User1 Service
     * @var string
     */
    public $baseUri;

    public function __construct()
    {
        $this->baseUri =
config('services.users1.base_uri');
    }

    /**
     * Obtain the full list of Users from User1 Site
     * @return string
     */

    public function obtainUsers1()
    {
        return $this-
>performRequest('GET','/users');  <—
this code will call the GET
localhost:8000/users (our site1)
    }

}
```

4. Make sure you have this code in your api gateway routes **> $router->get('/users1', 'User1Controller@index');**
5. Execute a GET request to your api gateway in postman > **localhost:8002/users1**, it should display all users from your Site1

2. **Create new User to Site1**
    1. Our Api Gateway Route for Create New User > **$router-**

>post('/users1', 'User1Controller@add');

2. Our Code in User1Controller add method:
   ```
   public function add(Request $request ){
         return $this->successResponse($this->user1Service->createUser1($request->all(), Response::HTTP_CREATED));
      }
   ```
3. Our Code in User1Service createUser1 Service
   ```
   /**
    * Create one user using the User1 service
    * @return string
    */
   public function createUser1($data)
   {
      return $this->performRequest('POST', '/users', $data);
   }
   ```
4. Execute a **POST** request to your api gateway in postman > **localhost:8002/users1**, it should create a new user to Site1

3. **Get Specific User based on User ID to Site1**
   1. Our Api Gateway Route for Show User > **$router->get('/users1/{id}', 'User1Controller@show'); *// get user by id***
   2. Our Code in User1Controller **show method**
      ```
      /**
       * Obtains and show one user
       * @return Illuminate|Http|Response
       */
      public function show($id)
      {
         return $this->successResponse($this->user1Service->obtainUser1($id));
      }
      ```
   3. Our Code in User1Service obtainUser1 Service
      ```
      /**
       * Obtain one single user from the User1 service
       * @return string
      ```

```
     */
    public function obtainUser1($id)
    {
        return $this->performRequest('GET', "/users/
{$id}");
    }
```

4. **Update specific User based on User ID to Site1**
    1. Our Api Gateway Route for Update User > **$router->put('/users1/{id}', 'User1Controller@update'); *// update user record***
    2. Our Code in User1Controller **update method**

```
/**
     * Update an existing user
     * @return Illuminate|Http|Response
     */
    public function update(Request $request,$id)
    {
        return $this->successResponse($this->user1Service->editUser1($request->all(), $id));
    }
```

    3. Our Code in User1Service **editUser1** Service

```
/**
     * Update an instance of user1 using the User1 service
     * @return string
     */
    public function editUser1($data, $id)
    {
        return $this->performRequest('PUT', "/users/{$id}", $data);
    }
```

5. **Delete specific User based on User ID to Site1**
    1. Our Api Gateway Route for Delete User > **$router->delete('/users1/{id}', 'User1Controller@delete'); *// delete record***
    2. Our Code in User1Controller **delete method**

```
/**
     * Remove an existing user
```

```
     * @return Illuminate|Http|Response
     */
    public function delete($id)
    {
        return $this->successResponse($this->user1Service->deleteUser1($id));
    }
```

3. Our Code in User1Service **deleteUser1** Service

```
/**
     * Remove an existing user
     * @return Illuminate|Http|Response
     */
    public function delete($id)
    {
        return $this->performRequest('DELETE', "/users/{$id}");
    }
```

6. **TEST THE ROUTES OF OUR API GATEWAY SEVER localhost:8002 using POSTMAN**

6. IMPLEMENT A FOREIGN KEY REQUIREMENTS IN ADDING A USER

1. **Create a table in SITE1 i.e. tbluserjob**
    1. Lets say that the table tbluser will required the user job (**jobid**) reference from table name **tbluserjob**
2. **Create the Model in Site1 name**
    1. **Name of Model —> UserJob.php**
3. **Add jobid field in the User Model**

```
protected $fillable = [
        'username', 'password','gender','jobid'
    ];
```

4. **Create the UserJobController**

```
<?php

namespace App\Http\Controllers;

//use App|User;
use App\Models\UserJob;  // <-- your model is located inside Models Folder
use Illuminate\Http\Response; // Response Components
use App\Traits\ApiResponser;  // <-- use to standardized our code
```

```
for api response
use Illuminate\Http\Request;  // <-- handling http request in lumen
use DB; // <-- if your not using lumen eloquent you can use DB
component in lumen

Class UserJobController extends Controller {
   // use to add your Traits ApiResponser
   use ApiResponser;

   private $request;

   public function __construct(Request $request){
      $this->request = $request;
   }

   /**
    * Return the list of usersjob
    * @return Illuminate|Http|Response
    */
   public function index()
   {
      $usersjob = UserJob::all();
      return $this->successResponse($usersjob);

   }

   /**
    * Obtains and show one userjob
    * @return Illuminate|Http|Response
    */
   public function show($id)
   {
      $userjob = UserJob::findOrFail($id);
      return $this-
>successResponse($userjob);
   }
}
```

5. **Create the Routes for UserJobController**

```
// userjob routes
$router->get('/usersjob',
```

'UserJobController@index');
$router->get('/userjob/{id}',
'UserJobController@show'); ***// get user by id***

6. **Modify the Code of Site1 User Add to validate the existing of jobid before saving to tbluse**
    1. Import the Model
        1. Add this line after **namespace App\Http\Controllers;**
        —> **use App\Models\UserJob;**
    2. Modify the method add

```
public function add(Request $request ){
        $rules = [
            'username' => 'required|max:20',
            'password' => 'required|max:20',
            'gender' => 'required|in:Male,Female',
            'jobid' => 'required|numeric|min:1|not_in:0',
        ];

        $this->validate($request,$rules);

        // validate if Jobid is found in the table tbluserjob
        $userjob = UserJob::findOrFail($request->jobid);

        $user = User::create($request->all());

        return $this->successResponse($user, Response::HTTP_CREATED);
    }
```

7. **Modify the Code of Site1 Update` to validate the existing of jobid before saving to tbluse**

```
public function update(Request $request,$id)
```

```
    {
        $rules = [
        'username' => 'max:20',
        'password' => 'max:20',
        'gender' => 'in:Male,Female',
        'jobid' => 'required|numeric|min:1|not_in:0',
        ];

        $this->validate($request, $rules);

        // validate if Jobid is found in the table tbluserjob
        $userjob = UserJob::findOrFail($request->jobid);

        $user = User::findOrFail($id);

        $user->fill($request->all());

        // if no changes happen
        if ($user->isClean()) {
            return $this->errorResponse('At least one value must change', Response::HTTP_UNPROCESSABLE_ENTITY);
        }

        $user->save();
        return $this->successResponse($user);

    }
```

8. **Do the STEPS 1 TO 7 for site2 Microservice**
9. **Test your site1 and site2 Microservice using Postman**
    1. **Make sure to run the server**
        1. **Site1 —>** php -S localhost:8000 -t public
        2. **Site2 —>** php -S localhost:8001 -t public
        3. Gateway —>

7. Controlling errors obtained from services

1. Copy the file of your site1 Handler.php found in app -> Exceptions -> Handler.php
2. Add the below code in our render method

```
if ($exception instanceof ClientException) {
        $message = $exception->getResponse()->getBody();
        $code = $exception->getCode();

        return $this->errorMessage($message,200);
    }
```

8. Implementing the security layer of the microservices architecture with Lumen

1. Installing and enabling Lumen Passport components

1. Lumen Passport is a simple service provider that makes Laravel Passport work with Lumen
2. Lumen Passport is package that implement OAuth 2.0
3. OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. This specification and its extensions are being developed within the IETF OAuth Working Group.
4. The OAuth 2.0 authorization framework is a protocol that allows a user to grant a third-party web site or application access to the user's protected resources, without necessarily revealing their long-term credentials or even their identity.
5. What is OAuth 2.0 ?
    1. https://www.youtube.com/watch?v=CPbvxxslDTU
    2. https://www.youtube.com/watch?v=BNEoKexlmA4
    3. https://auth0.com/docs/protocols/protocol-oauth2
6. Goto to site: https://github.com/dusterio/lumen-passport to know the details about lumen passport
7. **Goto your apigateway root directory to require lumen passport**
    1. > cd **ddsgateway**
    2. > **composer require dusterio/lumen-passport**

```
  - Installing laravel/passport (v10.1.0): Downloading (100%)
  - Installing dusterio/lumen-passport (0.3.1): Downloading (100%)
paragonie/random_compat suggests installing ext-libsodium (Provides a modern crypto API that can be use
d to generate random bytes.)
phpseclib/phpseclib suggests installing ext-libsodium (SSH2/SFTP can make use of some algorithms provid
ed by the libsodium-php extension.)
phpseclib/phpseclib suggests installing ext-mcrypt (Install the Mcrypt extension in order to speed up a
 few other cryptographic operations.)
phpseclib/phpseclib suggests installing ext-gmp (Install the GMP (GNU Multiple Precision) extension in
order to speed up arbitrary precision integer arithmetic operations.)
lcobucci/jwt suggests installing lcobucci/clock (*)
Package fzaninotto/faker is abandoned, you should avoid using it. No replacement was suggested.
Writing lock file
Generating optimized autoload files
70 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

8. **Modify the bootstrap flow (bootstrap/app.php file)**
    1. **We need to enable both Laravel Passport provider and Lumen-specific provider:**
        1. Uncomment this line
            1. > $**app->register(App\Providers\AuthServiceProvider::class);**
        2. *Finally register two service providers - original one and Lumen adapter*

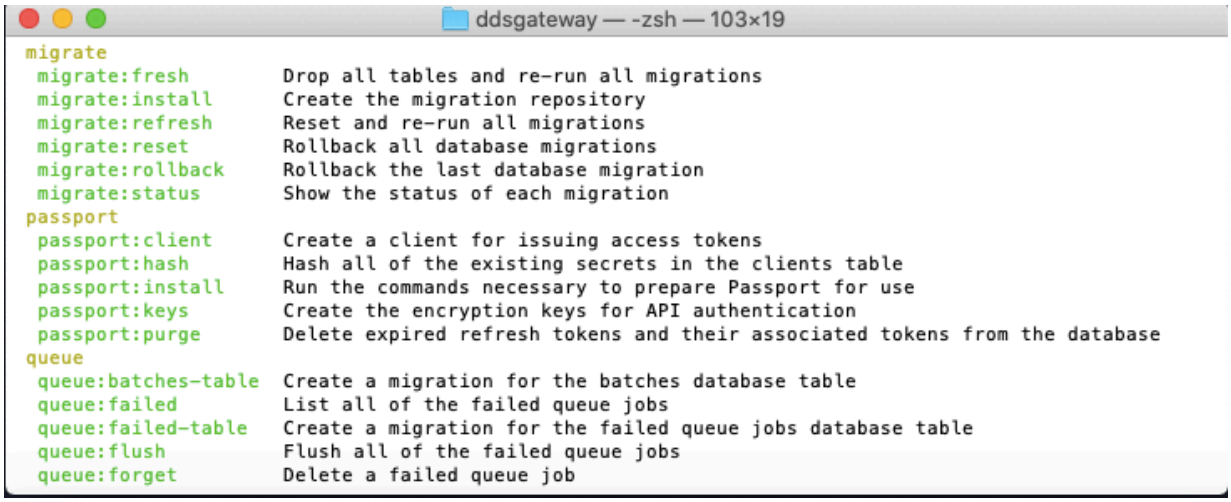
> **$app->register(Laravel\Passport\PassportServiceProvider::class);**
>
> **$app->register(Dusterio\LumenPassport\PassportServiceProvider::class);**

```
102
103   // edit for OAuth
104   $app->register(App\Providers\AuthServiceProvider::class);
105   $app->register(Laravel\Passport\PassportServiceProvider::class);
106   $app->register(Dusterio\LumenPassport\PassportServiceProvider::class);
107
108   /*
```

2. **Preparing and configuring Lumen to use lumen passport**
    1. Create a database for ddsgateway (in this example, i used **dbgateway** database name)

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=2206
DB_DATABASE=dbgateway
DB_USERNAME=root
DB_PASSWORD=1234
```

3. **Run the migrations to create the tables for Lumen**
    1. > php artisan



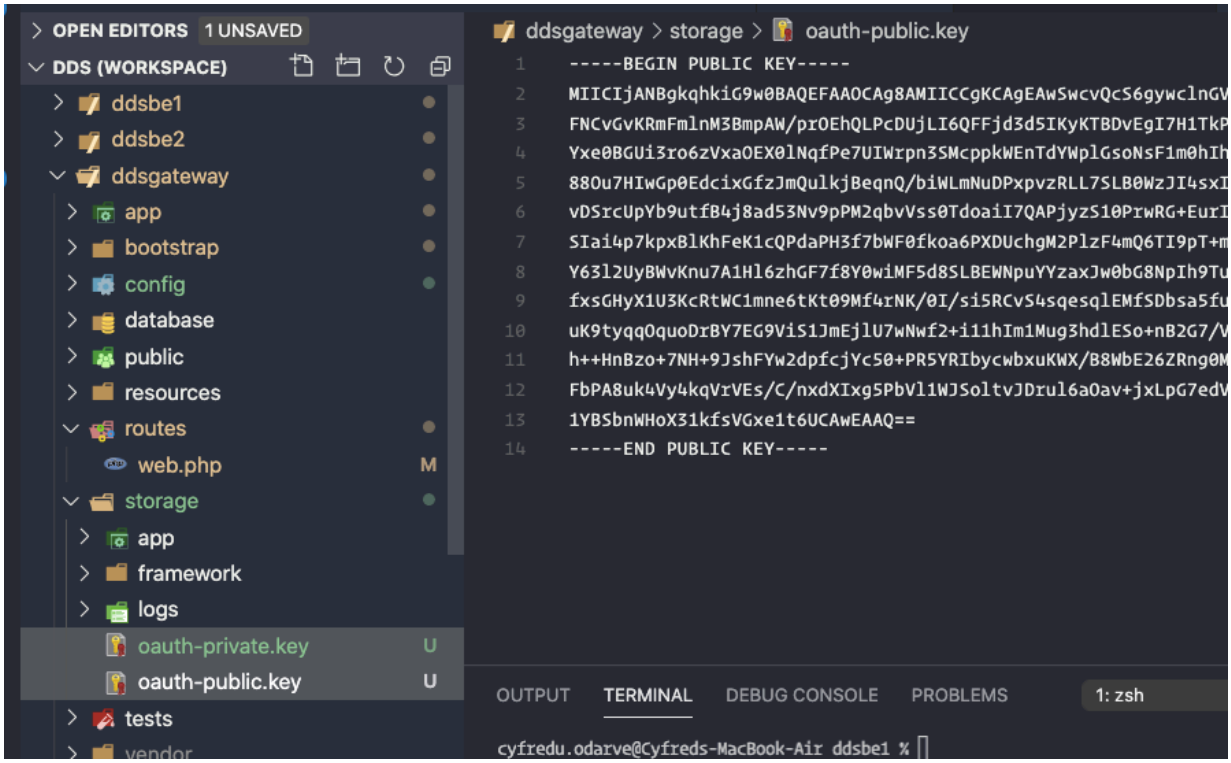
    2. cd to ddsapigateway root folder
    3. **Migrate and install Laravel Passport**
        1. Create new tables for Passport
            1. > **php artisan migrate**

```
cyfredu.odarve@Cyfreds-MacBook-Air ddsgateway % php artisan migrate
Migration table created successfully.
Migrating: 2016_06_01_000001_create_oauth_auth_codes_table
Migrated:  2016_06_01_000001_create_oauth_auth_codes_table (148.73ms)
Migrating: 2016_06_01_000002_create_oauth_access_tokens_table
Migrated:  2016_06_01_000002_create_oauth_access_tokens_table (98.99ms)
Migrating: 2016_06_01_000003_create_oauth_refresh_tokens_table
Migrated:  2016_06_01_000003_create_oauth_refresh_tokens_table (91.68ms)
Migrating: 2016_06_01_000004_create_oauth_clients_table
Migrated:  2016_06_01_000004_create_oauth_clients_table (64.25ms)
Migrating: 2016_06_01_000005_create_oauth_personal_access_clients_table
Migrated:  2016_06_01_000005_create_oauth_personal_access_clients_table (27.97ms)
```

        2. **Install encryption keys and other necessary stuff for Passport**
            1. > **php artisan passport:install**

```
cyfredu.odarve@Cyfreds-MacBook-Air ddsgateway % php artisan passport:install
Encryption keys generated successfully.
Personal access client created successfully.
Client ID: 1
Client secret: GKQ7lAgn0ePKnvYfwmjMxb6X1YKbJziBakldiP0E

In ChoiceQuestion.php line 36:
```

3. Notice in storage directory, **oauth-private.key and oauth-public** key are created



4. **Put the keys in the .gitignore directory**
    1. cd to root folder
    2. modify **.gitignore file**
    3. add this code
        1. > **storage/*.key**
5. **Register the routes and configuration to make sure to use lumen passport authentication of client in our API gateway microservices**
    1. **goto app -> Providers -> AuthServiceProvider.php**
    2. modify public function boot()
        1. **comment the below code**

```
// comment this code
/*
$this->app['auth']->viaRequest('api', function ($request) {
    if ($request->input('api_token')) {
        return User::where('api_token', $request->input('api_token'))->first()
    }
});*/
```

2. **Add this code**
    1. **LumenPassport::routes(*$this*->app->router);**
    2. Note
        1. make sure to import the LumenPassword use **Dusterio\LumenPassport\LumenPassport;**
3. **Register new configuration file coming from Lumen**
    1. Copy file from **vendor -> laravel -> lumen-framework -> config -> auth.php**
    2. Paste this file to **project folder -> config -> auth.php**
    3. Modify the auth.php file

        'guards' => [
        'api' => ['driver' => 'passport'],
        ],
        1. Replace api to **passport**
        2. **Next, load the config in bootstrap/app.php since Lumen doesn't load config files automatically:**
            1. **goto to bootstap -> app.php**
            2. **add this line**

                > **$app->configure('auth');**

```
/**
 * Registering config files
 */
$app->configure('services');
$app->configure('auth');
```

3. Protecting the Gateway routes with Lumen Passport

1. Enable **Middleware** in lumen to add authentication/ require access token in accessing the api gateway
    1. Goto bootstrap -> app.php
    2. **Uncomment this code**

```php
// Enable auth middleware
$app->routeMiddleware([
    'auth' => App\Http\Middleware\Authenticate::class,
    'client.credentials' => Laravel\Passport\Http\Middleware\CheckClientCredentials::class,
]);
```

2. Protect your web routes
    1. goto routes -> web.php
    2. edit the routes using below code

```php
// with authentication using the middleware
$router->group(['middleware' => 'client.credentials'],function() use ($router) {

    // API GATEWAY ROUTES FOR SITE1 users
    $router->get('/users1', 'User1Controller@index');
    $router->post('/users1', 'User1Controller@add');  // create new user record
    $router->get('/users1/{id}', 'User1Controller@show'); // get user by id
    $router->put('/users1/{id}', 'User1Controller@update'); // update user record
    $router->patch('/users1/{id}', 'User1Controller@update'); // update user record
    $router->delete('/users1/{id}', 'User1Controller@delete'); // delete record


    // API GATEWAY ROUTES FOR SITE2 users
    $router->get('/users2', 'User2Controller@index');
    $router->post('/users2', 'User2Controller@add');  // create new user record
    $router->get('/users2/{id}', 'User2Controller@show'); // get user by id
    $router->put('/users2/{id}', 'User2Controller@update'); // update user record
    $router->patch('/users2/{id}', 'User2Controller@update'); // update user record
    $router->delete('/users2/{id}', 'User2Controller@delete'); // delete record
});
```

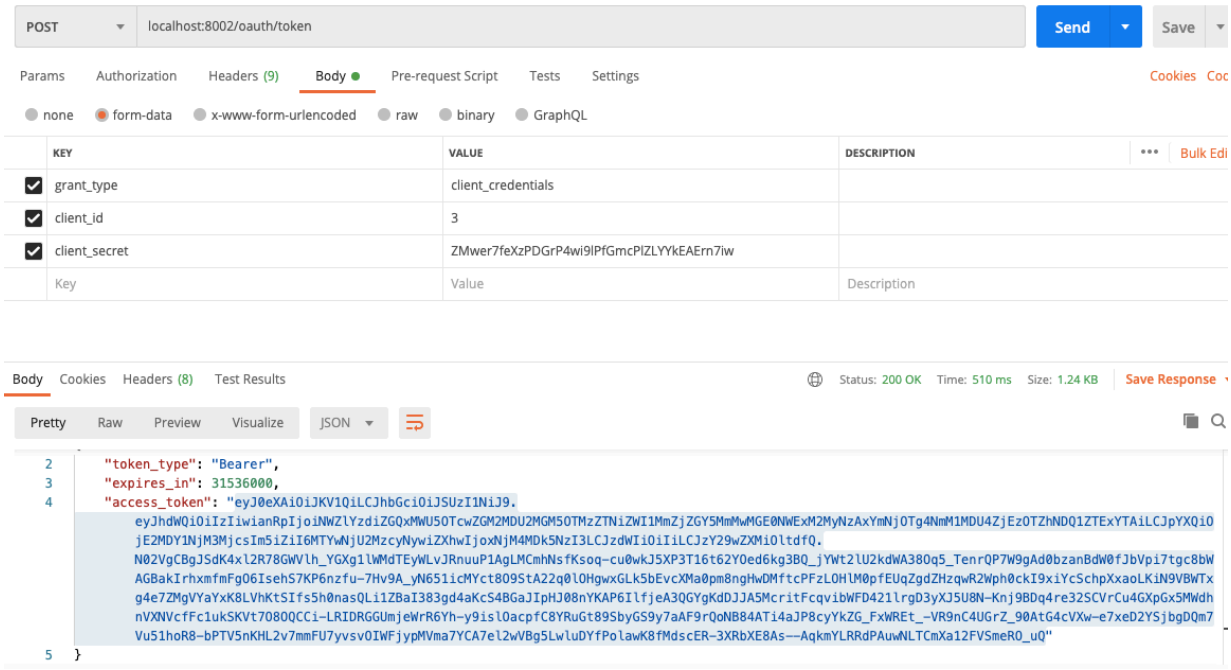
4. Obtaining and using access tokens for the Lumen API Gateway

1. Add access token in api request
    1. **Generate Access Token**

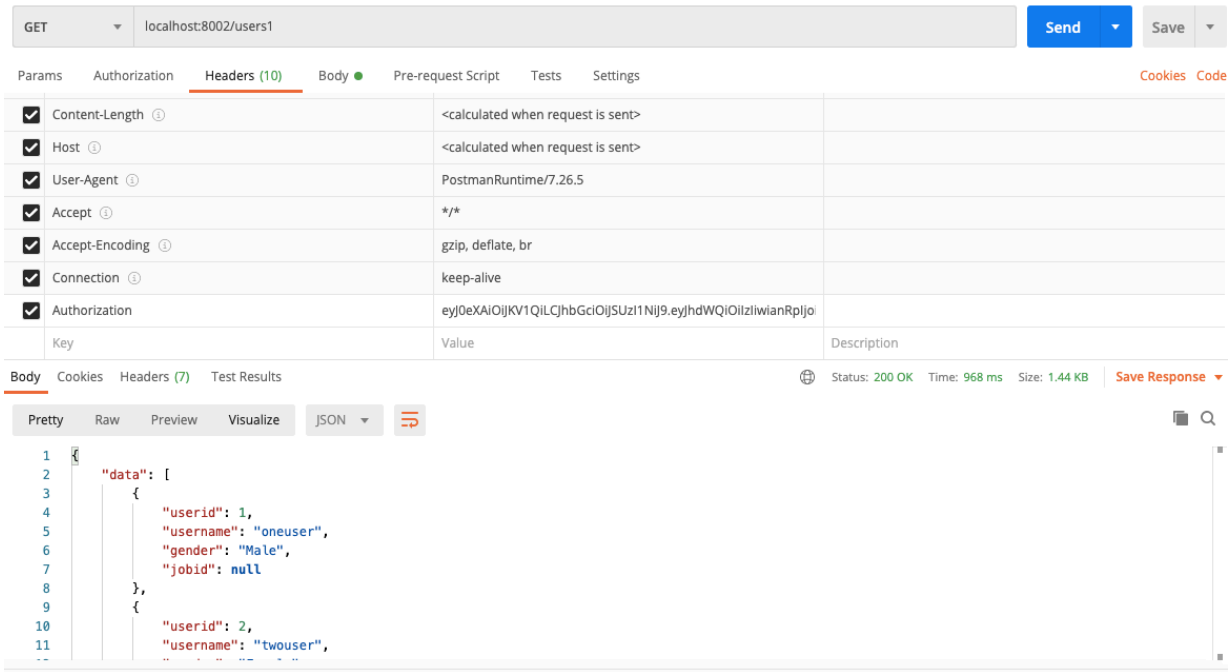
1. **Goto to your push man**
2. **Request {POST} localhost:8002/oauth/token**
3. In The Body add header
    1. **grant_type = client_credentials**
    2. **client_id = 3**
    3. **client_secret =** ZMwer7feXzPDGrP4wi9lPfGmcPlZLYYkEAErn7iw
        1. **Note you can create a client using the below command**
            1. > **php artisan passport:client**
            2. > In Which user ID should the client be assigned to?:
                1. Press Enter
            3. > In What should we name the client?:
                1. Type your client name i.e. MyClient
            4. > In where should we redirect the request after authorization
                1. Press Enter
            5. Copy the Generated ID and Key Secret
                1. Client ID: 3
                2. Client secret: ZMwer7feXzPDGrP4wi9lPfGmcPlZLYYkEAErn7iw

```
cyfredu.odarve@Cyfreds-MacBook-Air ddsgateway % php artisan passport:client

 Which user ID should the client be assigned to?:
 >

 What should we name the client?:
 > MyClient

 Where should we redirect the request after authorization? [http://localhost/auth/callback]:
 >

New client created successfully.
Client ID: 3
Client secret: ZMwer7feXzPDGrP4wi9lPfGmcPlZLYYkEAErn7iw
cyfredu.odarve@Cyfreds-MacBook-Air ddsgateway %
```

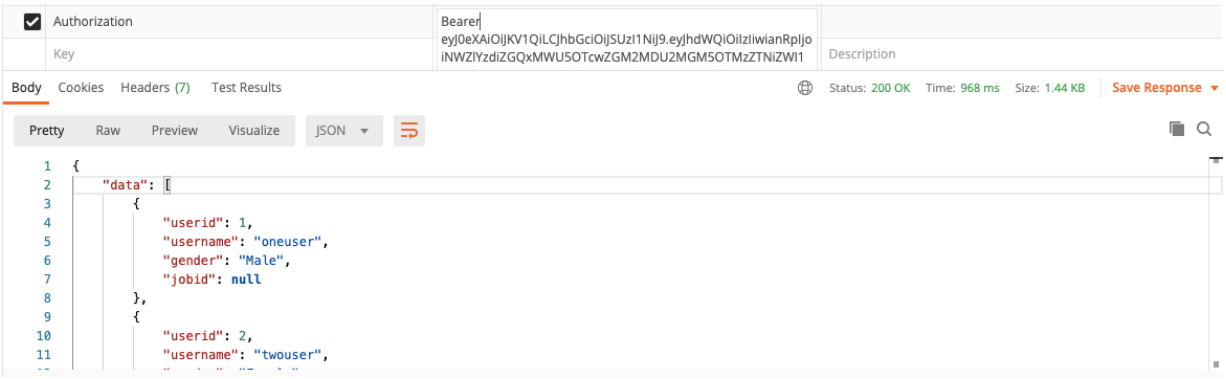
2. **Click Sent in Postman** (The results should be below)



5. Use the generated access token to request for lists of users to Site1
    1. In Postman Header, add this key and value
        1. **Authorization** = {The Generated Key Above}



2. Optionally the value of Authorization can include the token_type

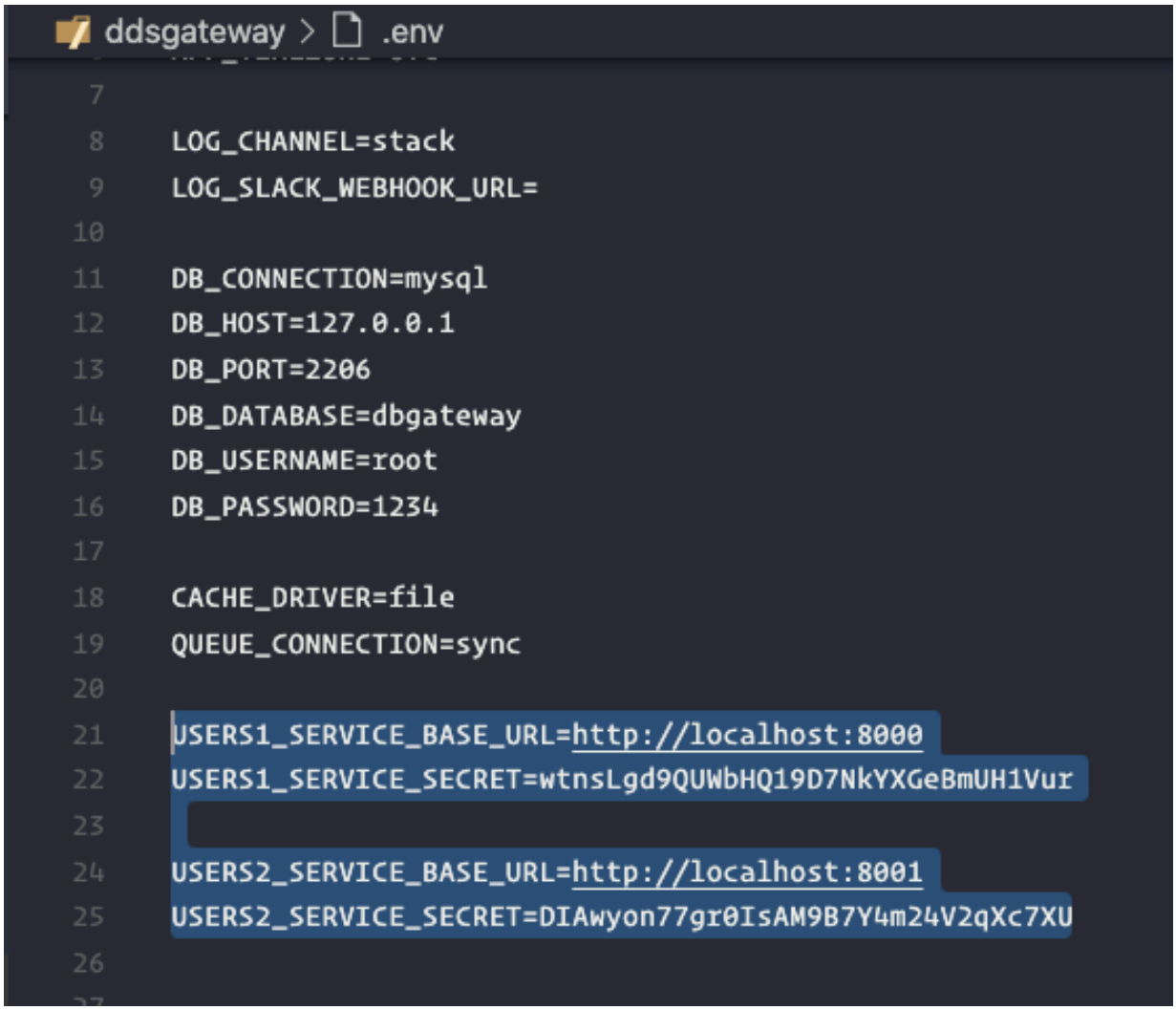


5. Preparing the API Gateway to authenticate its request

1. Goto app -> config -> services.php
2. Modify the code to add secret

```
<?php

return [
    'users1' => [
        'base_uri' => env('USERS1_SERVICE_BASE_URL'),
        'secret' => env('USERS1_SERVICE_SECRET'),
    ],

    'users2' => [
        'base_uri' => env('USERS2_SERVICE_BASE_URL'),
        'secret' => env('USERS2_SERVICE_SECRET'),
    ],
];
```

3. Goto .env of your apigateway and add the secret key of two sites
    1. Generate two key string using this site http://www.unit-conversion.info/texttools/random-string-generator/
    2. Put the generated key string in your .env



3. Lets modify our site services to include secret key in every request
    1. **SITE1** Goto app -> Services -> User1Service.php
        1. Add this code

```
/**
 * The secret to consume the User1 Service
 * @var string
 */
public $secret;


public function __construct()
{
    $this->baseUri = config('services.users1.base_uri');
    $this->secret = config('services.users1.secret');
}
```

2. **SITE2** Goto app -> Services -> User2Service.php
    1. Add this code

```
/**
 * The secret to consume the User2 Service
 * @var string
 */
public $secret;


public function __construct()
{
    $this->baseUri = config('services.users2.base_uri');
    $this->secret = config('services.users2.secret');
}
```

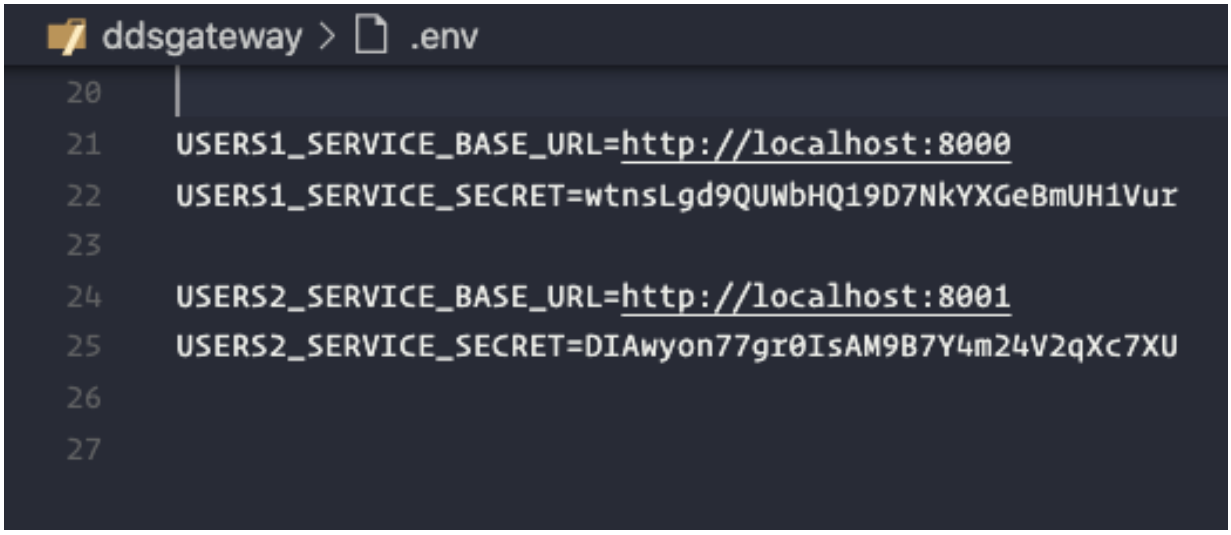
4. Lets modify our Traits -> **ConsumesExternalService** to include the **Authorization** value in submitting request to our **Internal API Sites**
    1. Goto app -> Traits -> ConsumesExternalService.php
        1. **Add the highlighted code below that will include a header name Authorization equal to the value found our .env file**

```
public function performRequest($method,$requestUrl,$form_params =[],$headers =[])
{
    // create a new client request
    $client = new Client([
        'base_uri' => $this->baseUri,
    ]);

    if(isset($this->secret)) {
        $headers['Authorization'] = $this->secret;
    }


    // perform the request (method,url,form parameters,headers)
    $response = $client->request($method,$requestUrl,['form_params' =>
        $form_params, 'headers' => $headers]);


    // return the response body contents
    return $response->getBody()->getContents();
```

6. **Authenticating direct access to the Lumen Site1 User microservice**
    1. **This time we are going to make sure that only our API**

GATEWAY is allowed to access our two Internal Microservices for security

2. Adding a middleware to our User Site 1 Micorservice to validate if the submitted key from gateway is match to our allowed secrets keys
    1. Goto our Site 1 User and modify **.env** file
    2. Enable to accept secret key to our Site1 microservice
        1. Add this code (secret keys are separated by comma)
            1. > ACCEPTED_SECRETS=**wtnsLgd9QUWbHQ19D7NkYXGeBmUH1Vur**,abcvdjadfk
        2. Note the secret key should be the same with value in our API gateway .env file



3. Lets create a middleware for **SITE 1 USER MICROSERVICE**
    1. Goto app -> Http -> Middleware
        1. Create a duplicate copy of ExampleMiddleware.php to a new filename AuthenticateAccess.php
        2. Below is the code of our Middleware name —> **AuthenticateAccess.php**

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Response;

class AuthenticateAccess
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        $validSecrets = explode(',',env('ACCEPTED_SECRETS'));

        if(in_array($request->header('Authorization'),$validSecrets)) {
            return $next($request);
        }

        abort(Response::HTTP_UNAUTHORIZED);
    }
}
```

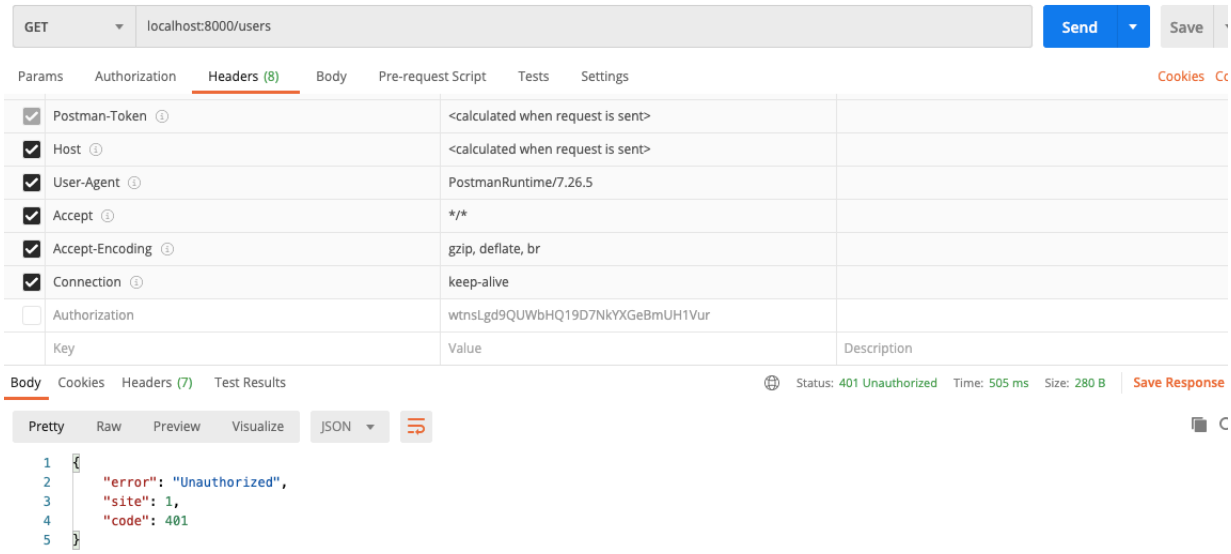
4. Register the **AuthenticateAccess.php middleware globally**
    1. Goto app -> bootstrap -> app.php
    2. Under Register Middleware (Uncomment the line below)

$app->middleware([

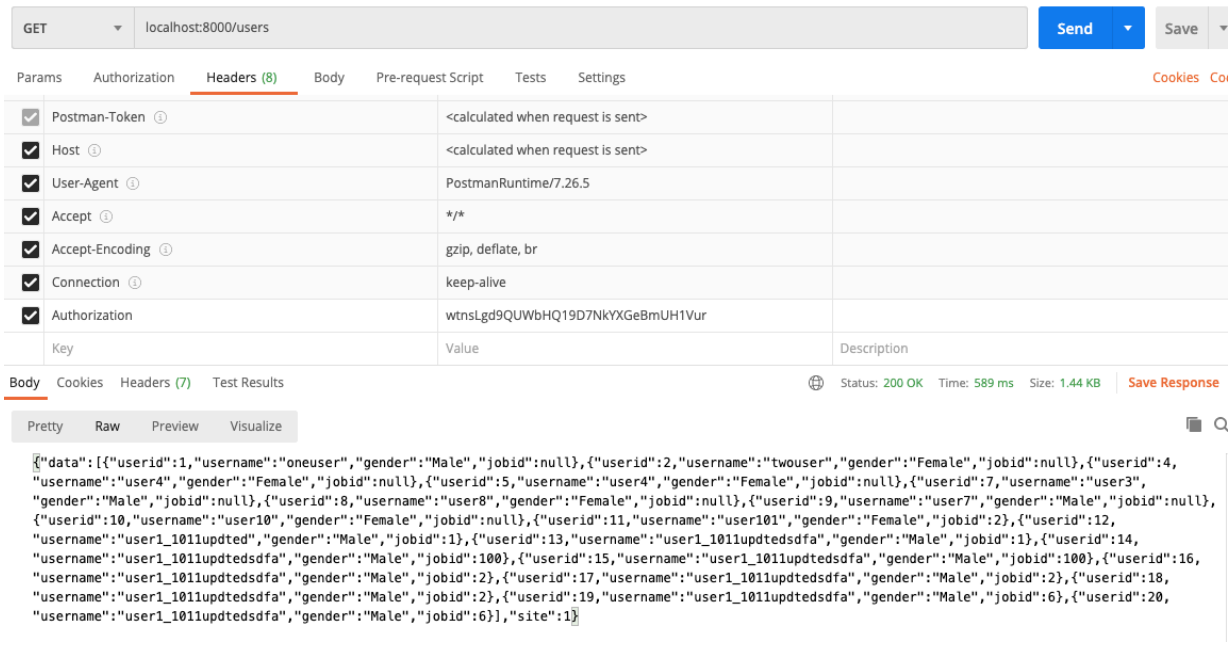
App\Http\Middleware\AuthenticateAccess::class,
]);

3. **Make sure that the name of Middleware is AuthenticateAccess::class**
4. Goto Postman, an Error will display if we will not

include authorization code in accessing directly our site1 microservice



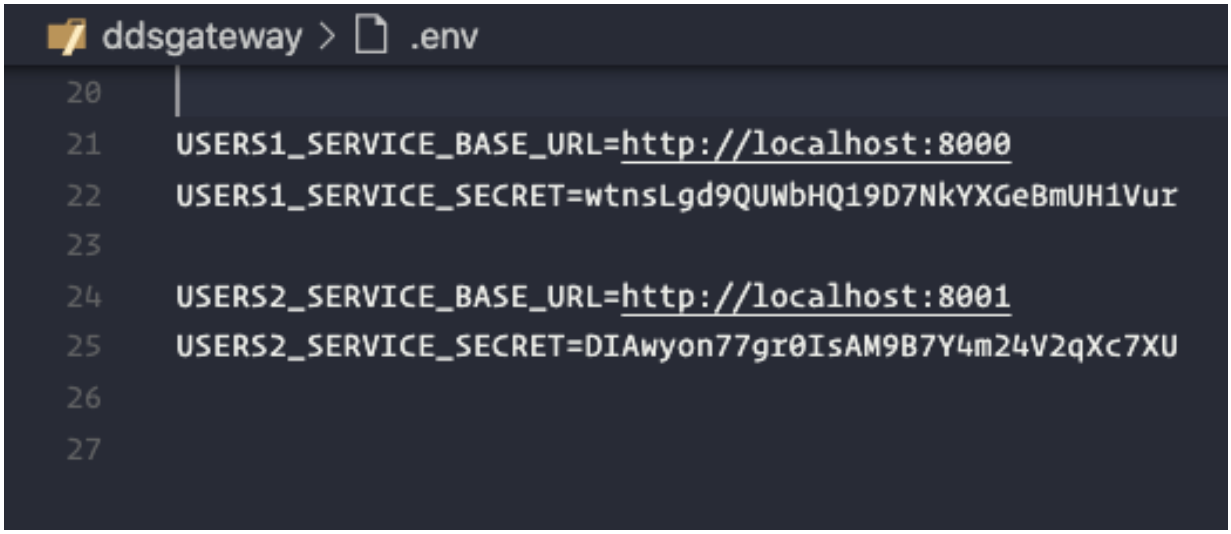
5. **But if we put the Authorization = key string value (it will be success)**



7. **Authenticating direct access to the Lumen Site2 User microservice**
    1. This time we are going to make sure that only our API GATEWAY is allowed to access our two Internal Microservices for security
    2. Adding a middleware to our User Site 2 Micorservice to validate if the

submitted key from gateway is match to our allowed secrets keys

1. Goto our Site 1 User and modify **.env** file
2. Enable to accept secret key to our Site1 microservice
    1. Add this code (secret keys are separated by comma)
        1. > ACCEPTED_SECRETS=**DIAwyon77gr0IsAM9B7Y4m24V2qXc7XU**
        2. Note the secret key should be the same with value in our API gateway .env file



3. Lets create a middleware for **SITE 2 USER MICROSERVICE**
    1. Goto app -> Http -> Middleware
        1. Create a duplicate copy of ExampleMiddleware.php to a new filename AuthenticateAccess.php
        2. Below is the code of our Middleware name —> **AuthenticateAccess.php**

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Response;

class AuthenticateAccess
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        $validSecrets = explode(',',env('ACCEPTED_SECRETS'));

        if(in_array($request->header('Authorization'),$validSecrets)) {
            return $next($request);
        }

        abort(Response::HTTP_UNAUTHORIZED);
    }
}
```

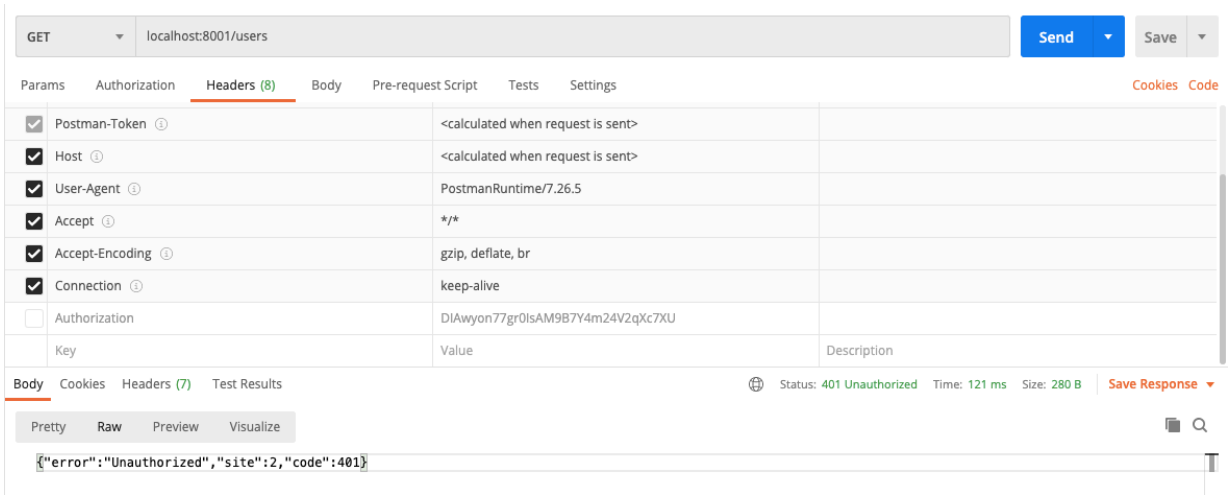
4. Register the **AuthenticateAccess.php middleware globally**
    1. Goto app -> bootstrap -> app.php
    2. Under Register Middleware (Uncomment the line below)

$app->middleware([

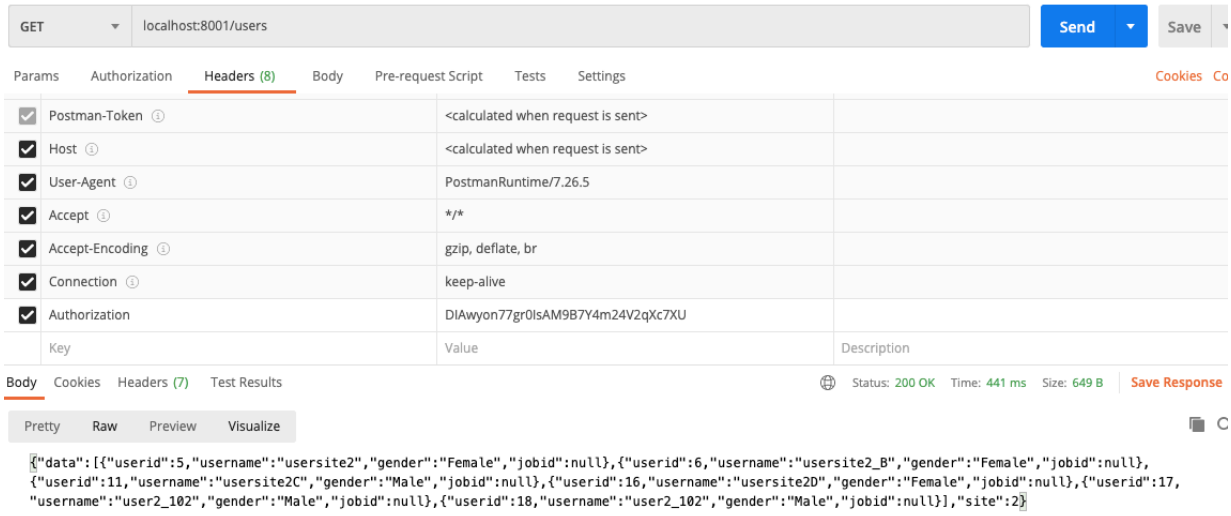
App\Http\Middleware\AuthenticateAccess::class,
]);

3. **Make sure that the name of Middleware is AuthenticateAccess::class**
4. Goto Postman, an Error will display if we will not

include authorization code in accessing directly our site2 microservice



5. **But if we put the Authorization = key string value (it will be success)**



YOU HAVE SUCCESSFULLY CREATED A MICROSERVICE

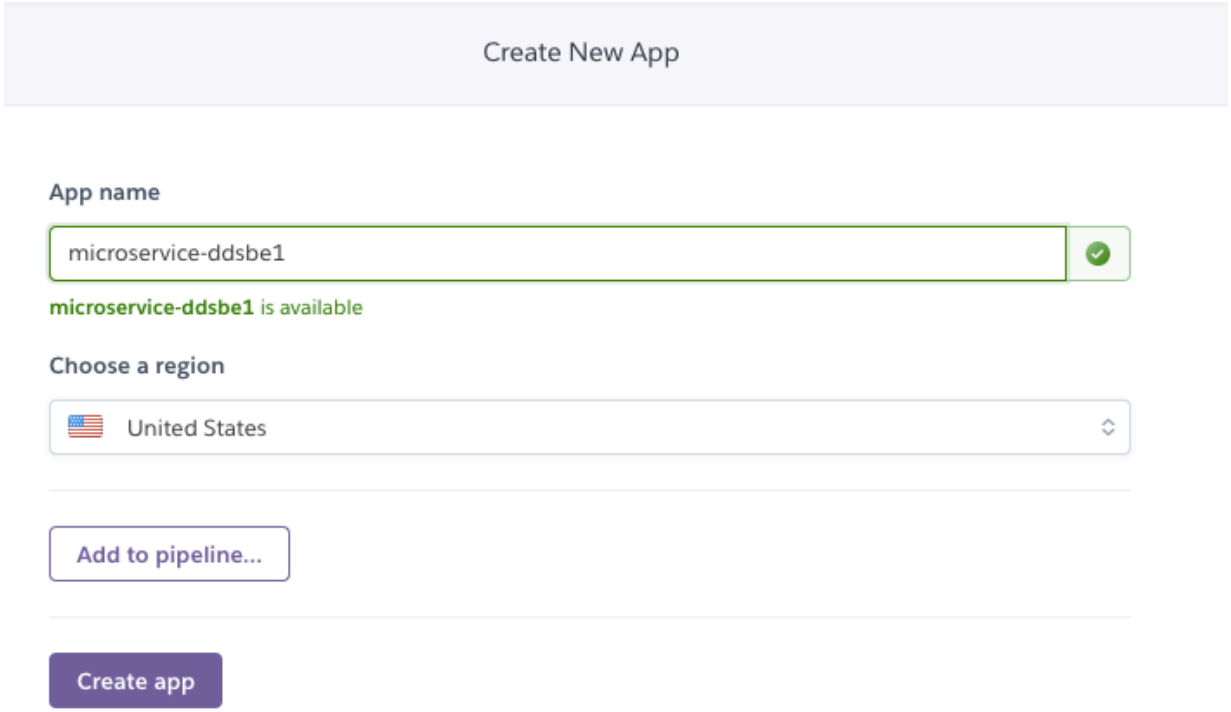
8. DEPLOYING YOUR MICROSERVICE TO HEROKU CLOUD SERVER

1. **Prerequisite**
    1. You completed all your Microservice requirements up to Authentication
    2. You have push all your Microservice source codes to your Github repository

2. **Create an account on Heroku using the link below:**
    1. https://www.heroku.com

3. **Download and Install Heroku CLI tools to allow you to write Heroku commands from your command line using the link below:**
    1. https://devcenter.heroku.com/articles/heroku-cli

4. **Deploying our site1 (ddsbe1) lumen project to heroku cloud server**
    1. Login to your heroku account https://www.heroku.com
    2. Click [Create new app] and named it, in my case I named it with **microservice-ddsbe1**



3. Goto to your site1 root directory —> cd to your **ddsbe1** folder
4. Make sure to commit all your changes to your git repository using the below commands:
    1. > git add -A
    2. > git commit -m “ddsbe1 Update as of May252021”
    3. > git push origin master —> this is to push your changes to your github repository
5. Login to your Heroku using the Heroku CLI by the executing the commands below:
    1. > goto your site1 folder —> cd to your **ddsbe1** folder
    2. > heroku login

```
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % heroku login
 ›   Warning: Our terms of service have changed: https://dashboard.heroku.com/terms-of-service
heroku: Press any key to open up the browser to login or q to exit:
```

3. Press any key to continue to open browser to login and message will display Logging in... done

```
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/3dddd983-7b61-4a59-aaa7-6179eb731395?requestor=SFMyNTY.g2gDbQAAAA4xMDMuNjIuMTUzLjIxNm4GABNJQ6R5AWIAAVGA.RtUTRBfesvkr4armAYSRtBPw1GaeMssQoPtVlwP8aBQ
Logging in... done
Logged in as cyfred.odarve@ustp.edu.ph
```

4. Since our lumen project have already an existing repositories, we will add only the Heroku Remote Repository link to our git config using the command below :
    1. > heroku git:remote -a **microservice-ddsbe1**
    2. Note: **microservice-ddsbe1** is the app name

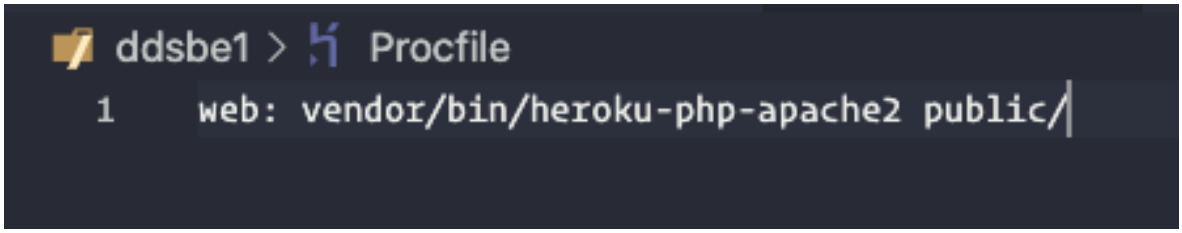
```
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % heroku git:remote -a microservice-ddsbe1
set git remote heroku to https://git.heroku.com/microservice-ddsbe1.git
```

3. To check if you successfully added the heroku remote repository to your Git config
    1. > git remote -v

```
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % git remote -v
heroku  https://git.heroku.com/microservice-ddsbe1.git (fetch)
heroku  https://git.heroku.com/microservice-ddsbe1.git (push)
origin  https://github.com/cydarvs/ddsbe.git (fetch)
origin  https://github.com/cydarvs/ddsbe.git (push)
```

5. Create **Procfile (no extension file)** inside your site1 Lumen project root directory (**ddsbe1 > Procfile**) Copy the below code to Procfile file and save:

web: vendor/bin/heroku-php-apache2 public/

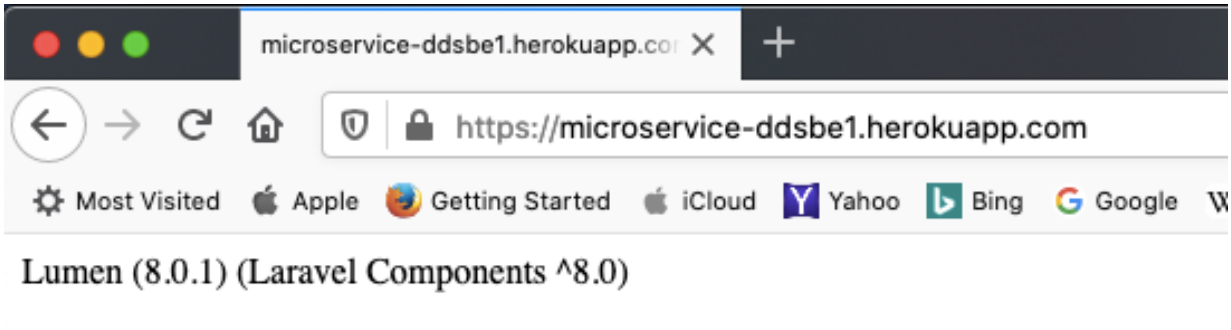


6. Push your site1 (**ddsbe1**) lumen project into Heroko
    1. > git add -A —> *(just to make sure that you have updated code committed to your local git)*
    2. > git commit -m "My latest ddsbe1 code" —> *(just to make sure that you have updated code committed to your local git>*
    3. > git push origin master —> pushing first to your github repo
    4. > **git push heroku master —> pushing to your heroku repo and heroku will install the required lumen project dependency libraries as shown below**

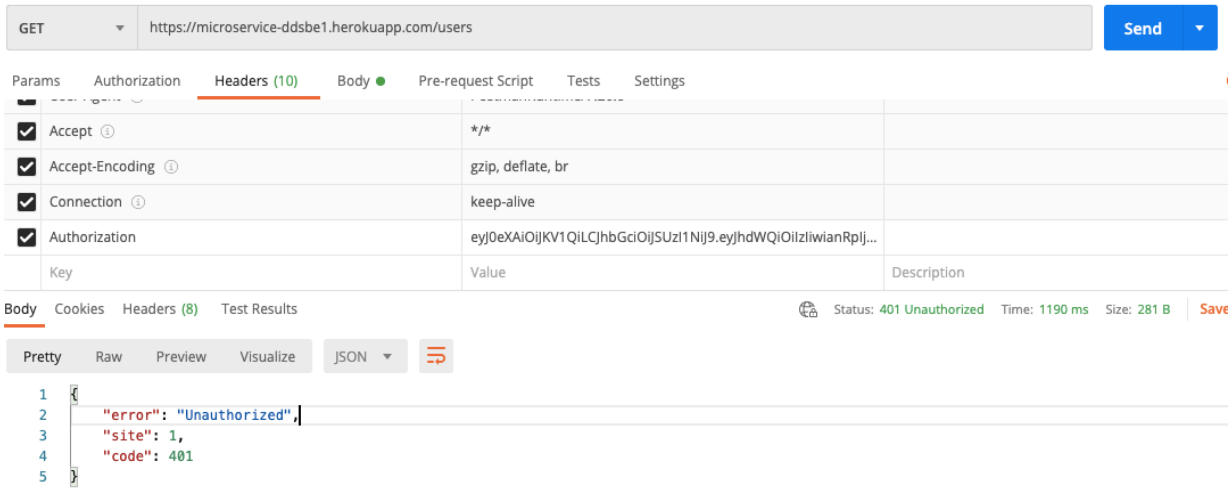
```
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % git push heroku master
Enumerating objects: 117, done.
Counting objects: 100% (117/117), done.
Delta compression using up to 4 threads
Compressing objects: 100% (97/97), done.
Writing objects: 100% (117/117), 40.40 KiB | 1.55 MiB/s, done.
Total 117 (delta 32), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-20 stack
remote: -----> Determining which buildpack to use for this app
remote: -----> PHP app detected
remote: -----> Bootstrapping...
remote: -----> Installing platform packages...
remote:        - php (7.4.19)
remote:        - ext-mbstring (bundled with php)
remote:        - apache (2.4.46)
remote:        - nginx (1.18.0)
remote:        - composer (1.10.22)
remote: -----> Installing dependencies...
remote:        Composer version 1.10.22 2021-04-27 13:10:45
remote:        Loading composer repositories with package information
remote:        Installing dependencies from lock file
remote:        Package operations: 72 installs, 0 updates, 0 removals
remote:          - Installing doctrine/inflector (2.0.3): Downloading (100%)
remote:          - Installing symfony/polyfill-php72 (v1.18.1): Downloading (100%)
remote:          - Installing paragonie/random_compat (v9.99.99): Downloading (100%)
remote:          - Installing symfony/polyfill-php70 (v1.18.1): Downloading (100%)
remote:          - Installing symfony/polyfill-intl-normalizer (v1.18.1): Downloading (100%)
remote:          - Installing symfony/polyfill-intl-idn (v1.18.1): Downloading (100%)
remote:          - Installing doctrine/lexer (1.2.1): Downloading (100%)
remote:          - Installing egulias/email-validator (2.1.22): Downloading (100%)
remote:          - Installing symfony/polyfill-php80 (v1.18.1): Downloading (100%)
remote:          - Installing symfony/polyfill-mbstring (v1.18.1): Downloading (100%)
remote:          - Installing symfony/deprecation-contracts (v2.2.0): Downloading (100%)
remote:          - Installing symfony/http-foundation (v5.1.7): Downloading (100%)
remote:          - Installing symfony/finder (v5.1.7): Downloading (100%)
```

```
remote:          - Installing illuminate/broadcasting (v8.9.0): Downloading (100%)
remote:          - Installing illuminate/auth (v8.9.0): Downloading (100%)
remote:          - Installing dragonmantank/cron-expression (3.0.1): Downloading (100%)
remote:          - Installing laravel/lumen-framework (v8.1.0): Downloading (100%)
remote:        Generating optimized autoload files
remote:        Deprecation Notice: Class App\Models\User located in ./app/Models/User1.php does not comply with psr-4 autoloading st
andard. It will not autoload anymore in Composer v2.0. in phar:///tmp/build_665ba007/.heroku/php/bin/composer/src/Composer/Autoload/
ClassMapGenerator.php:201
remote:        Warning: Ambiguous class resolution, "App\Models\User" was found in both "/tmp/build_665ba007/app/Models/User1.php" a
nd "/tmp/build_665ba007/app/Models/User.php", the first will be used.
remote:        37 packages you are using are looking for funding.
remote:        Use the `composer fund` command to find out more!
remote: -----> Preparing runtime environment...
remote: -----> Checking for additional extensions to install...
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 16.4M
remote: -----> Launching...
remote:        Released v3
remote:        https://microservice-ddsbe1.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/microservice-ddsbe1.git
 * [new branch]      master -> master
```

7. Notice that our site1(ddsbe1) was deployed to the link https://microservice-ddsbe1.herokuapp.com/
8. When you open the link it will display below



9. By this time you have deployed your site1 (**ddsbe1**) to Heroku but this will not yet function as expected because we did not yet setup our MySQL database
10. It will be display below in our postman an error response, if we try to test our Microservice API because database is not yet setup

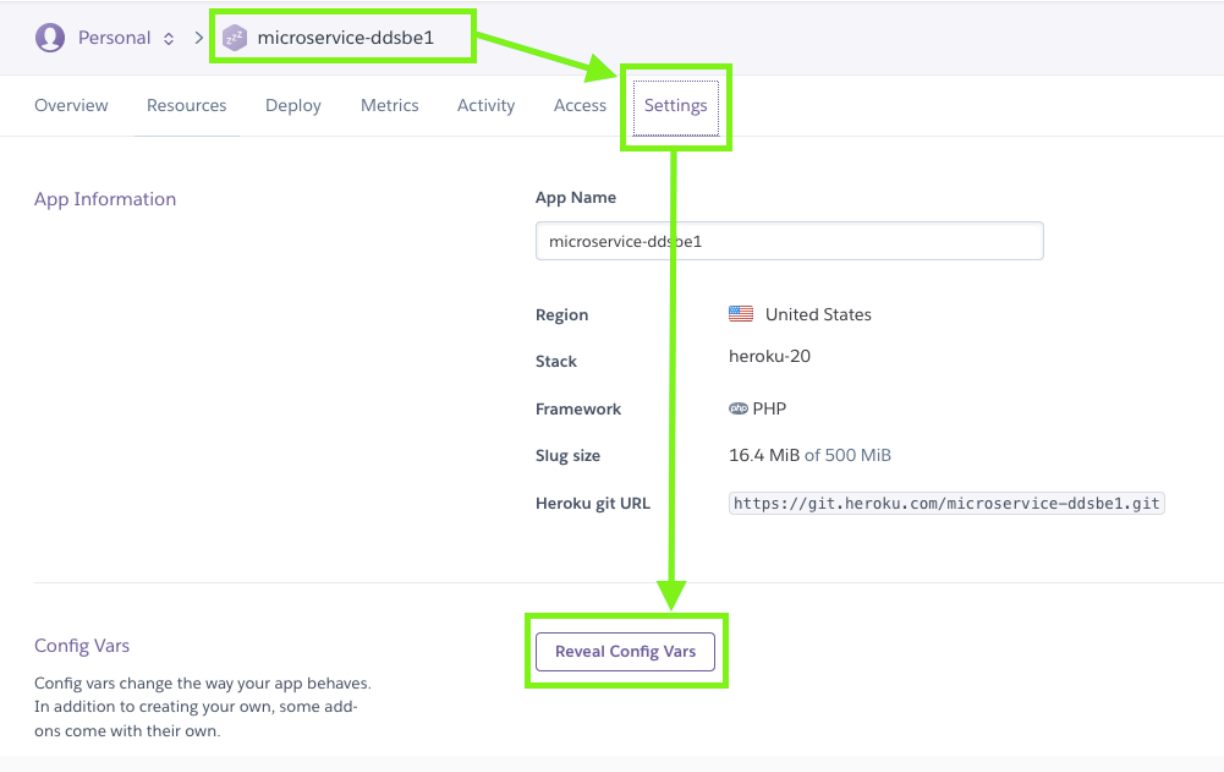


5. SETTING OUR LUMEN .env ENVIRONMENT VARIABLE CONFIG FILE TO HEROKU SERVER

1. We will be setting the below config to our Heroku Server

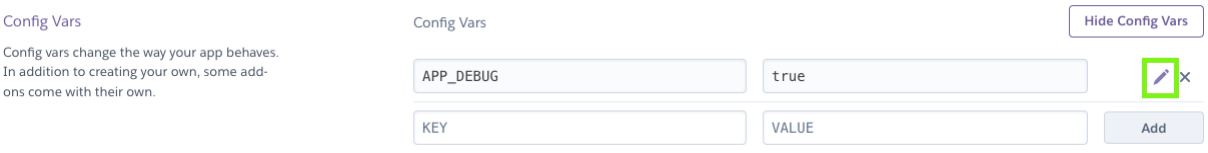
```
APP_NAME=Lumen
APP_ENV=local
APP_KEY=yZmu4sQDVQMBGEvU3abMnqVtb3b6QQAD
APP_DEBUG=true
APP_URL=http://localhost
APP_TIMEZONE=UTC

LOG_CHANNEL=stack
LOG_SLACK_WEBHOOK_URL=

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=2206
DB_DATABASE=dbsite1
DB_USERNAME=root
DB_PASSWORD=1234

CACHE_DRIVER=file
QUEUE_CONNECTION=sync

ACCEPTED_SECRETS=wtnsLgd9QUWbHQ19D7NkYXGeBmUH1Vur,abcvdjadfk
```

2. SETTING THE APP_NAME Config using the Heroku Portal
    1. Open your Heroku > Click your ddsbe1 app > Click Settings > Click Reveal Config Vars



2. After Clicking Reveal Config Vars, below will be displayed
3. Type the Key and Value, in our case Key=APP_DEBUG Value=true

4. If you want to edit value, click the Pencil Symbol Below



5. You can also add the lumen .env config using Heroku CLI
    1. Let adds the following Key and Value to our Heroku Server

1. APP_NAME = Laravel
2. APP_ENV = production
3. APP_URL = https://microservice-ddsbe1.herokuapp.com <— note this the App URL provided by Heroku when we push it
4. APP_KEY = yZmu4sQDVQMBGEvU3abMnqVtb3b6QQAD <— you can generate new app_key fro production
5. APP_TIMEZONE=UTC

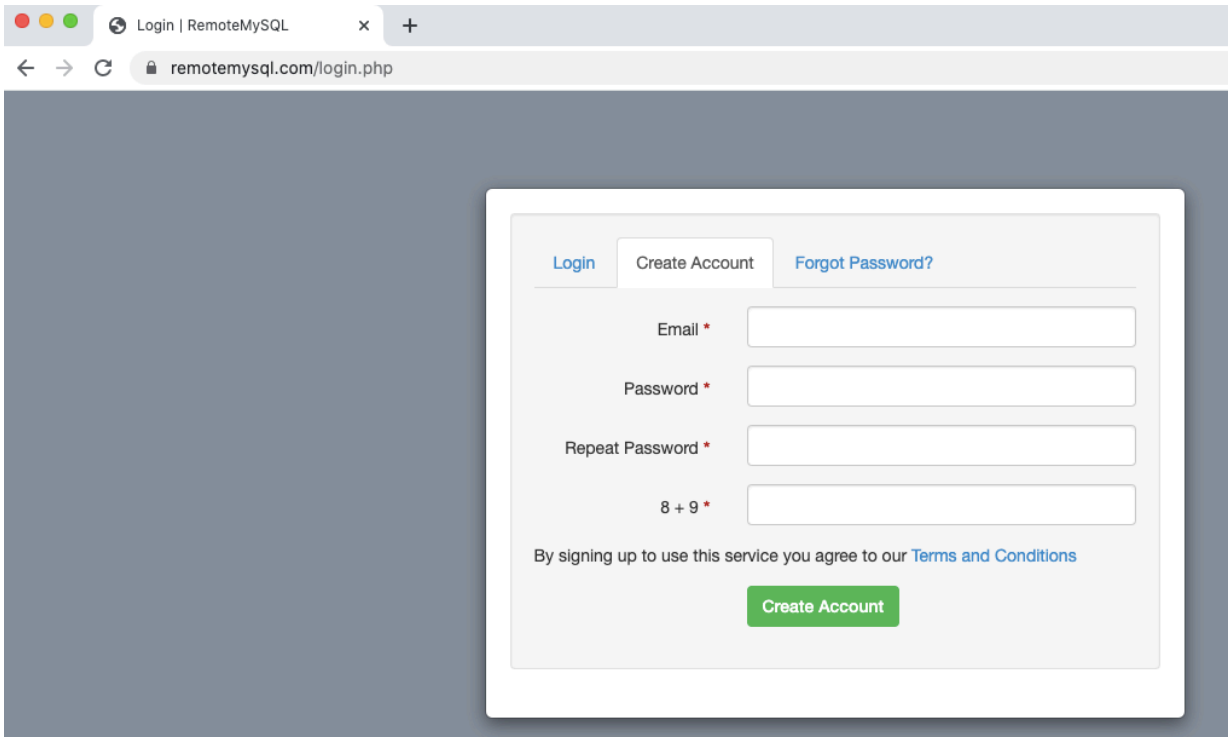
2. Execute the following command inside your site1 (ddsbe1) root directory
    1. > heroku config:add APP_NAME=Laravel
    2. > heroku config:add APP_ENV=production
    3. > heroku config:add APP_URL=https://microservice-ddsbe1.herokuapp.com
    4. > heroku config:add APP_KEY=yZmu4sQDVQMBGEvU3abMnqVtb3b6QQAD
    5. > heroku config:add APP_TIMEZONE=UTC

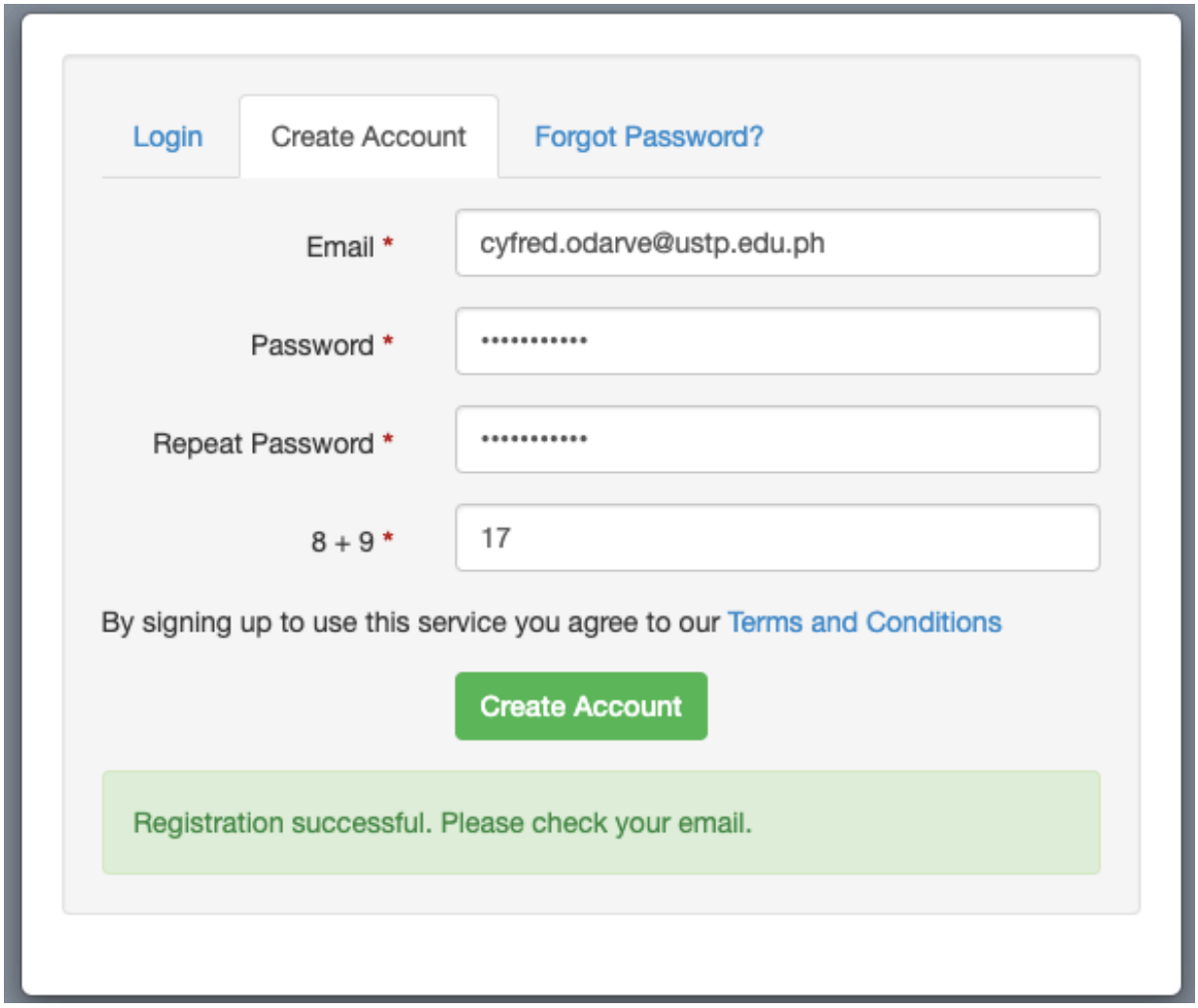
```
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % heroku config:add APP_NAME=Laravel
Setting APP_NAME and restarting ⬢microservice-ddsbe1... done, v5
APP_NAME: Laravel
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % heroku config:add APP_ENV=production
Setting APP_ENV and restarting ⬢microservice-ddsbe1... done, v6
APP_ENV: production
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % heroku config:add APP_URL=https://microservice-ddsbe1.herokuapp.com
Setting APP_URL and restarting ⬢microservice-ddsbe1... done, v7
APP_URL: https://microservice-ddsbe1.herokuapp.com
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 %
```

```
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % heroku config:add APP_KEY=yZmu4sQDVQMBGEvU3abMnqVtb3b6QQAD
Setting APP_KEY and restarting ⬢microservice-ddsbe1... done, v8
APP_KEY: yZmu4sQDVQMBGEvU3abMnqVtb3b6QQAD
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % nano .env
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 % heroku config:add APP_TIMEZONE=UTC
Setting APP_TIMEZONE and restarting ⬢microservice-ddsbe1... done, v9
APP_TIMEZONE: UTC
cyfredu.odarve@Cyfreds-MacBook-Air ddsbe1 %
```

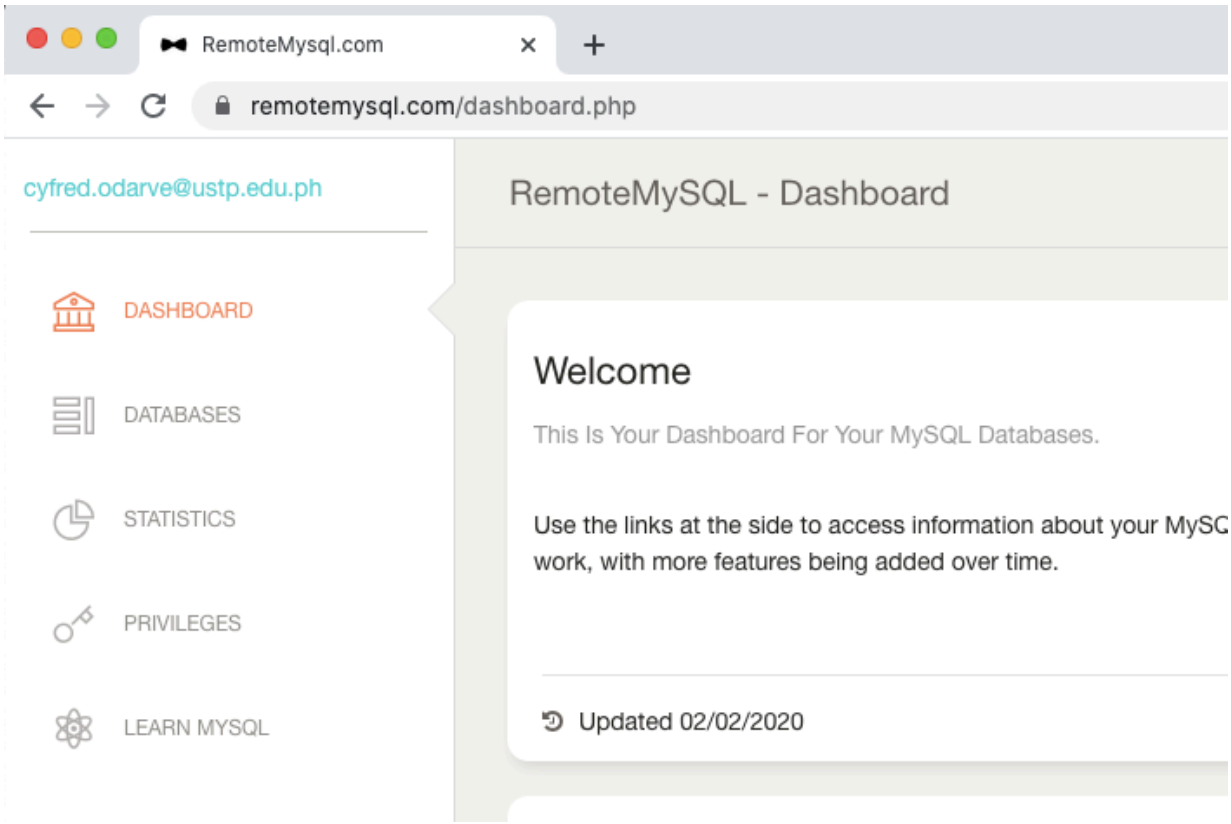
3. If we refresh our Heroku portal, the config should be added in the Heroku config Vars as shown below

4. SETTING THE MYSQL DATABASE CONFIG
    1. Before we can setup the mysql database, we need to create a remote database, Heroku has a free mysql database but requires Credit Card Information. You can add this by clicking > Resources > click Add-ons > click Find more add-ons and Search for Cleardb MySQL
    2. To overcome the requirement of Credit Card, we will use a free remote mysql server by creating an account at https://remotemysql.com/login.php
    3. Note: it is better to open this link using a chrome browser
    4. Create an account by completing the information below

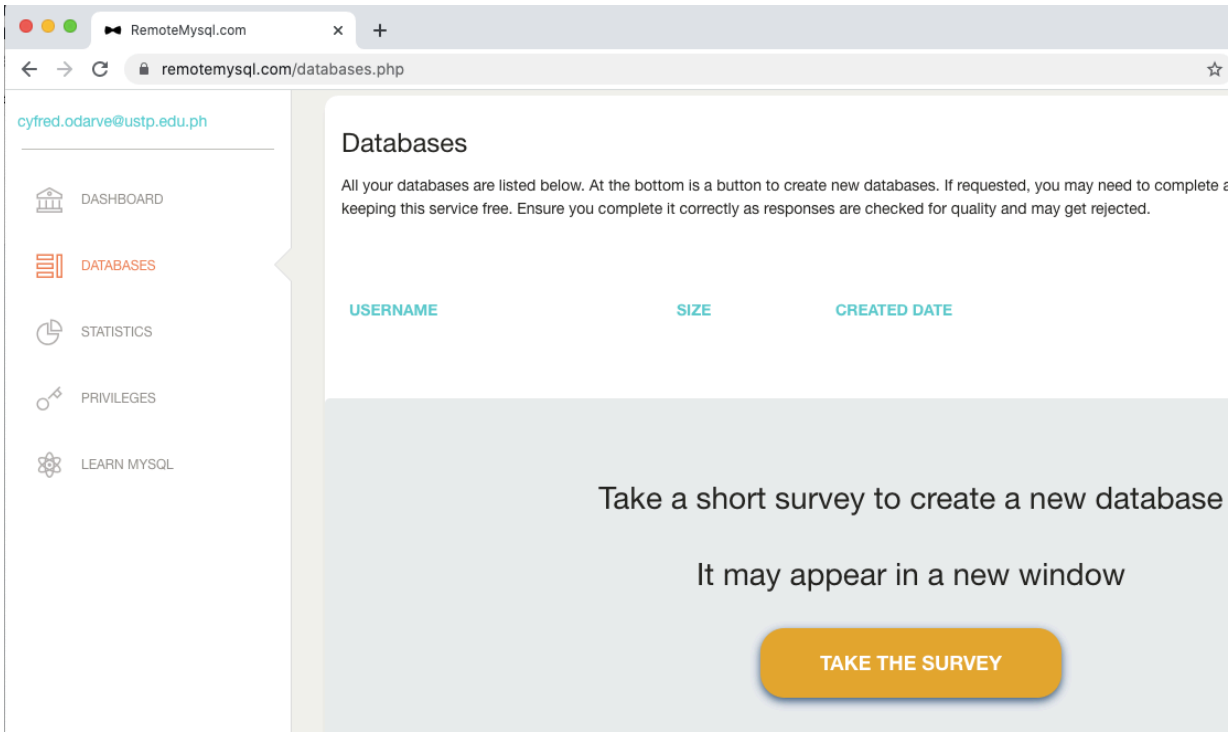

Login | RemoteMySQL
remotemysql.com/login.php
Login
Create Account
Forgot Password?
Email *
Password *
Repeat Password *
8 + 9 *
By signing up to use this service you agree to our Terms and Conditions
Create Account

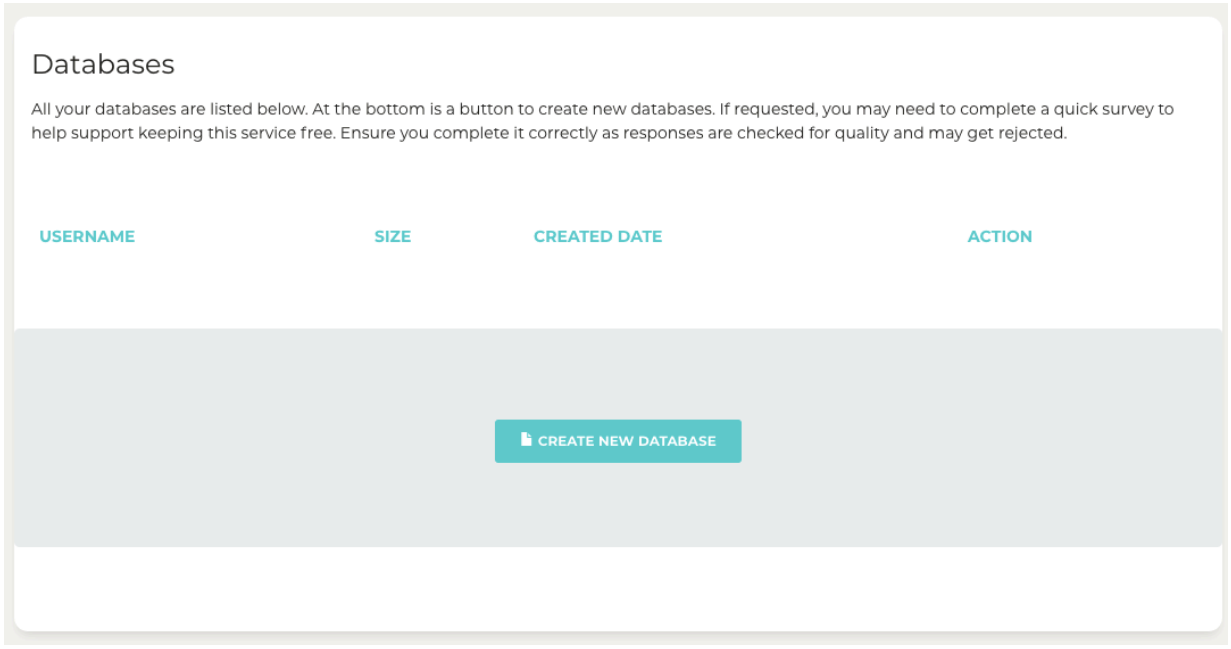


5. Open your registered email for verification and login then click DATABASES



5. After Click, it will require a survey for us to create a new database, for security purpose don't answer the correct personal information when doing the survey. Click DATABASES to refresh so that CREATE NEW DATABASE will appear.





7. CLICK Create New Database to create a free mysql database account as shown below

RemoteMySQL - Dashboard

Databases

All your databases are listed below. At the bottom is a button to create new databases. If requested, you may need to complete a quick survey to help support keeping this service free. Ensure you complete it correctly as responses are checked for quality and may get rejected.

Created!

You have successfully created a new database. The details are below.

Username: 1PpHofpOUL

Database name: 1PpHofpOUL

Password: MuYoOFwEPy

Server: remotemysql.com

Port: 3306

These are the username and password to log in to your database and phpMyAdmin

Make sure you keep your password secure. Ensure you keep back ups of your database in case it gets deleted.

USERNAME	SIZE	CREATED DATE	ACTION
1PpHofpOUL	0 MB	2021-05-26 08:33:12	≡ ACTION

Created!

You have successfully created a new database. The details are below.

Username: vG8B1JnIam

Database name: vG8B1JnIam

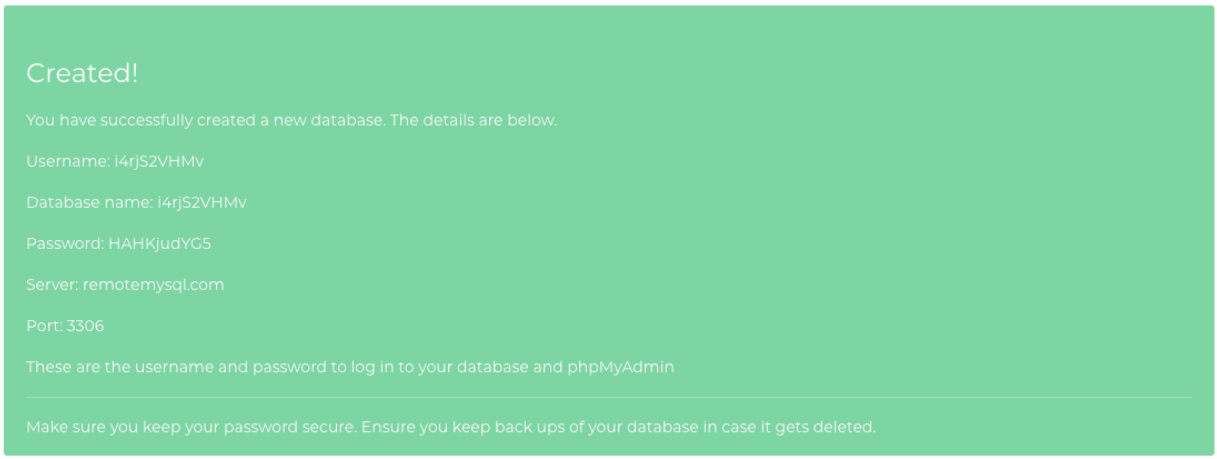
Password: yStsmo4aJt

Server: remotemysql.com

Port: 3306

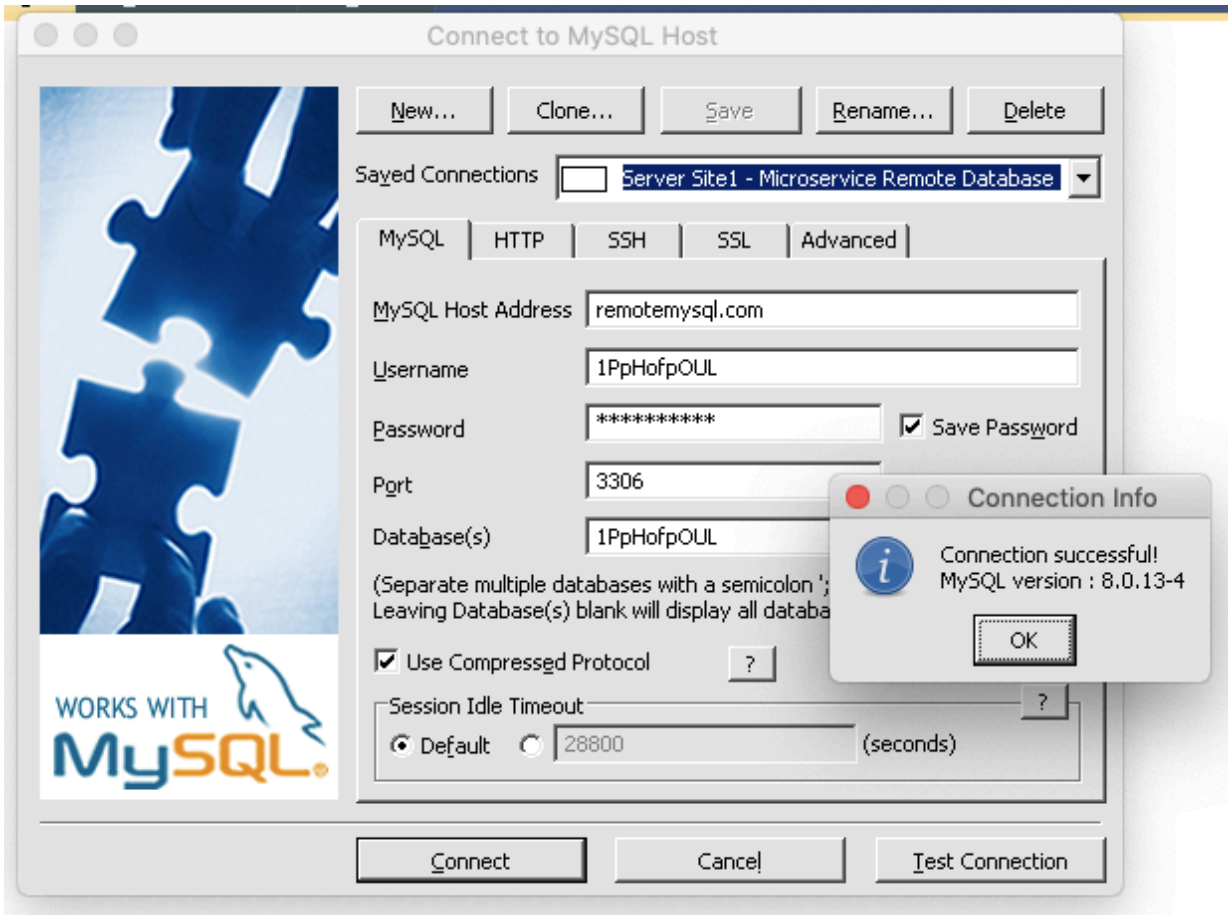
These are the username and password to log in to your database and phpMyAdmin

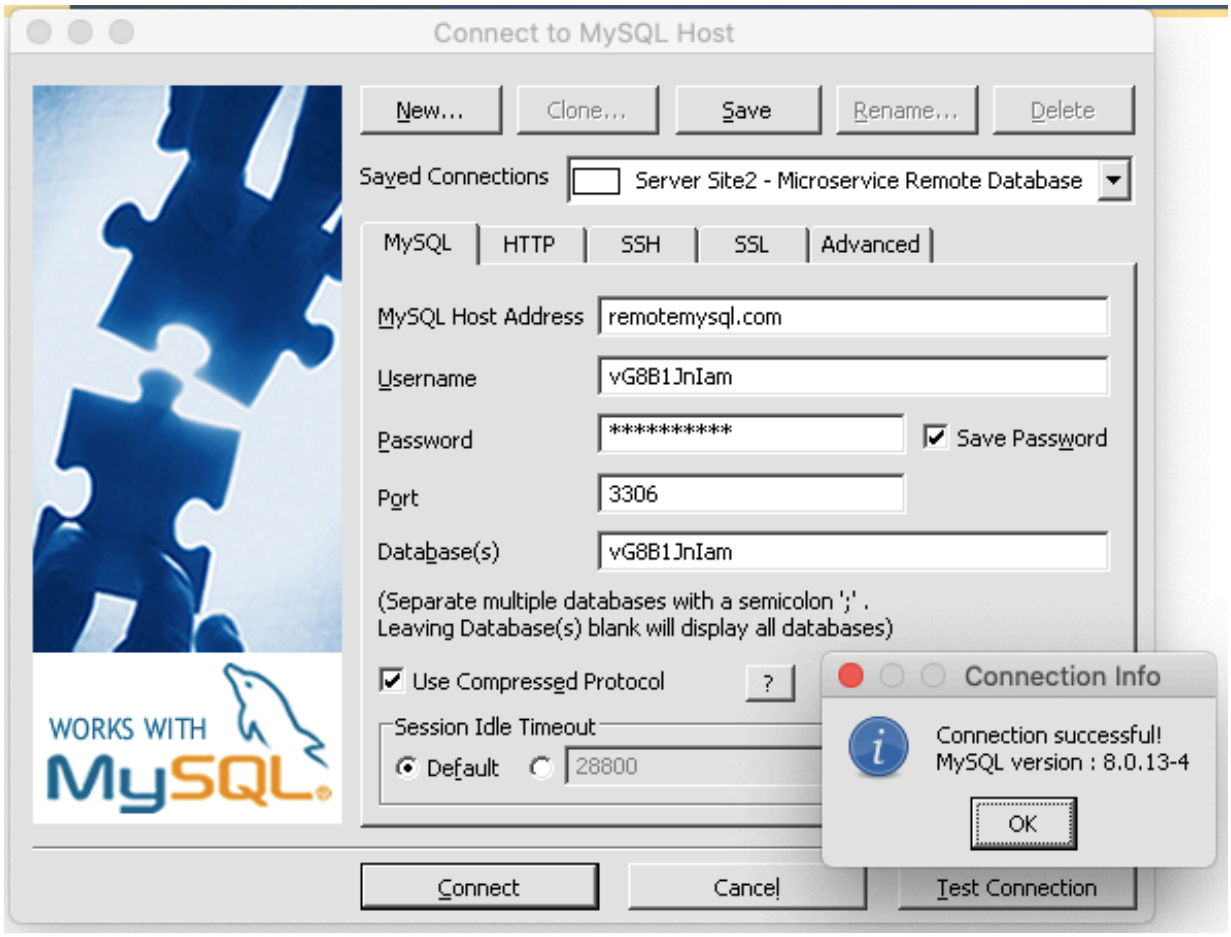
Make sure you keep your password secure. Ensure you keep back ups of your database in case it gets deleted.

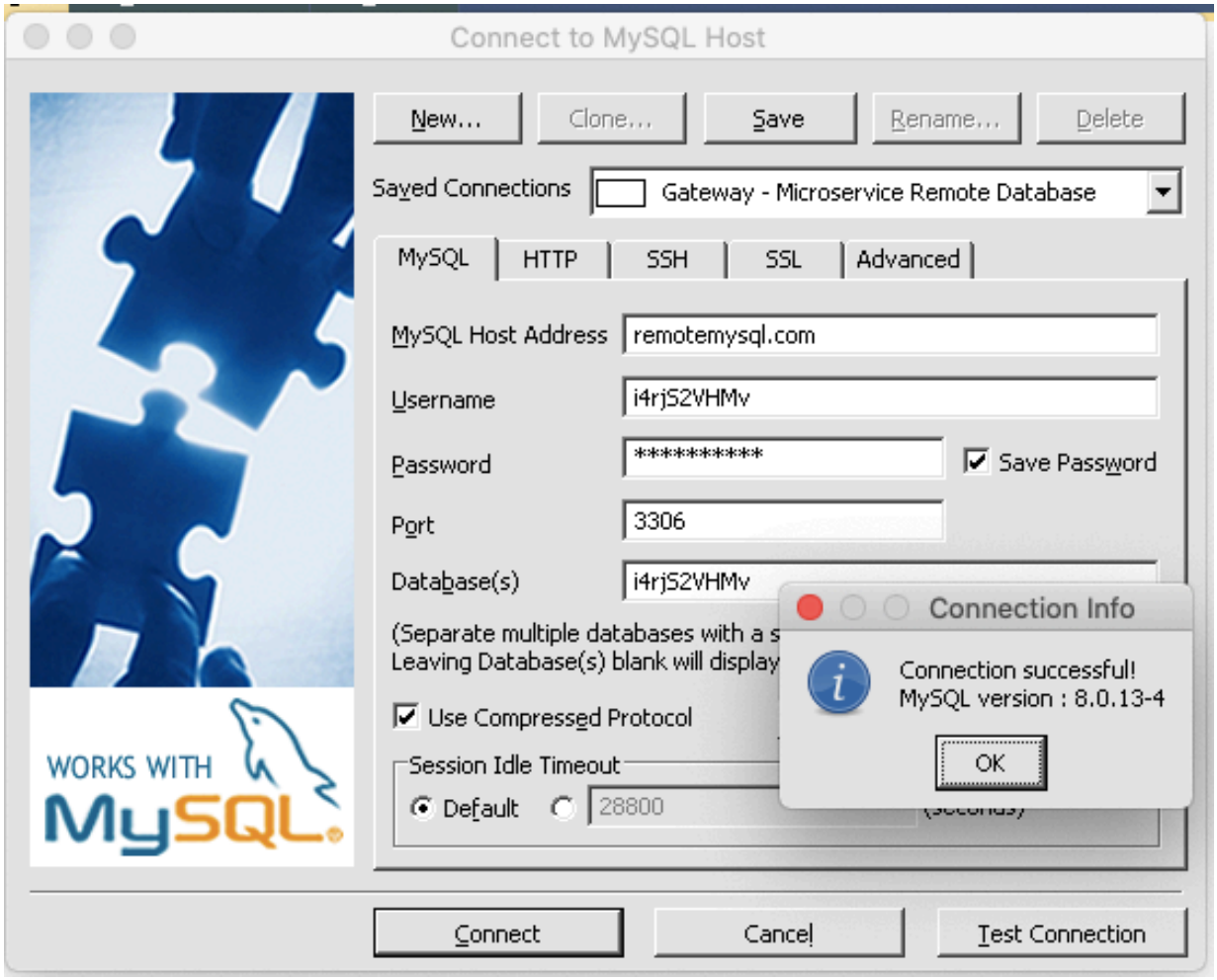


4. Take Note of your Remote MySQL Database Account. In my case :
    1. **For Site1 (dbsite1) database**
        1. username: **1PpHofpOUL**
        2. Database name: **1PpHofpOUL**
        3. Password: **MuYoOFwEPy**
        4. Server: remotemysql.com <— this is the MySQL Host Address
        5. Port: **3306**
    2. **For Site2 (dbsite2) database**
        1. username: **vG8B1Jnlam**
        2. Database name: **vG8B1Jnlam**
        3. Password: **yStsmo4aJt**
        4. Server: remotemysql.com <— this is the MySQL Host Address
        5. Port: **3306**
    3. **For Database Gateway (dbgateway) database**
        1. username: **i4rjS2VHMv**
        2. Database name: **i4rjS2VHMv**
        3. Password: **HAHKjudYG5**
        4. Server: remotemysql.com <— this is the MySQL Host Address
        5. Port: **3306**

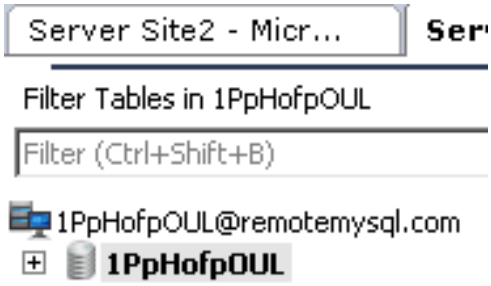
5. Try to open this database using your phpmyadmin, mysql workbench or in my case I am using an SQLYog
    1. Click New Connection
    2. Name the Connection in my case is :
        1. **Site1 = Server Site1 - Microservice Remote Database**
        2. **Site2 = Server Site2 - Microservice Remote Database**
        3. **Gateway = Gateway - Microservice Remote Database**
    3. **Click Test Connection,** a message box will appear below
    4. Click Connect to Connect to the Database

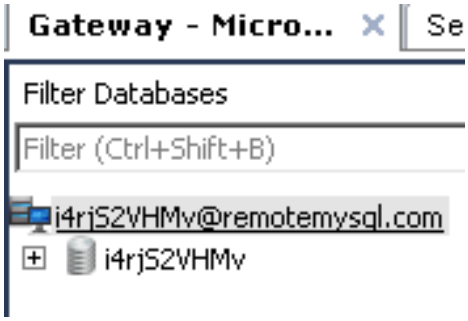



Connect to MySQL Host
New...
Clone...
Save
Rename...
Delete
Saved Connections
Server Site2 - Microservice Remote Database
MySQL
HTTP
SSH
SSL
Advanced
MySQL Host Address
remotemysql.com
Username
vG8B1JnIam
Password
****
Save Password
Port
3306
Database(s)
vG8B1JnIam
(Separate multiple databases with a semicolon ';' .
Leaving Database(s) blank will display all databases)
Use Compressed Protocol
?
Session Idle Timeout
Default
28800
WORKS WITH
MySQL
Connection Info
Connection successful!
MySQL version : 8.0.13-4
OK
Connect
Cancel
Test Connection



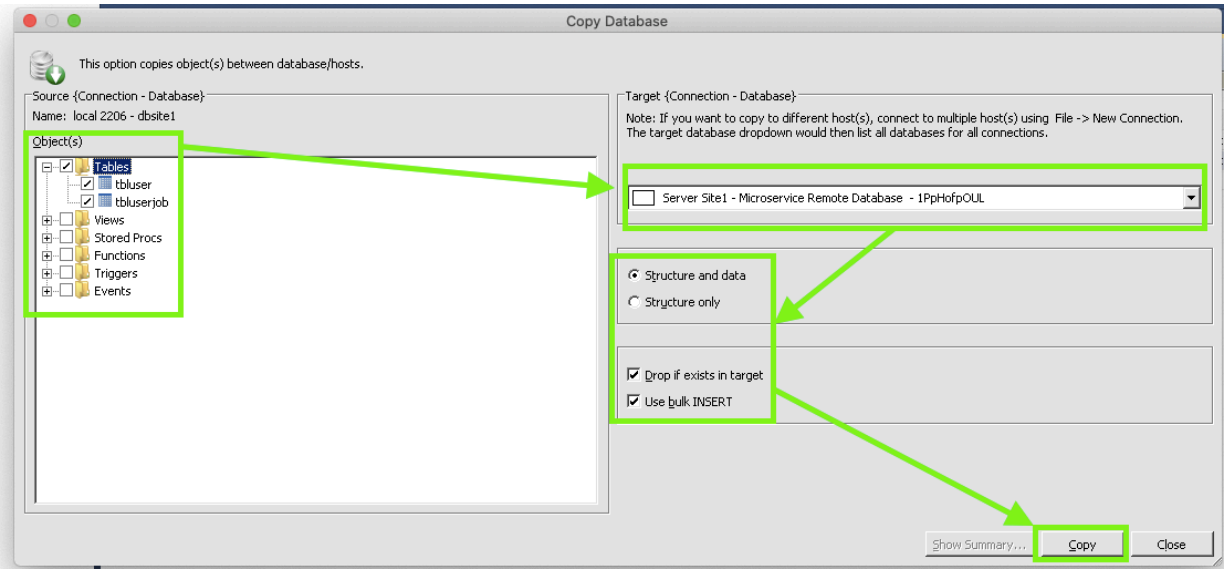
5. It will display below if Connected

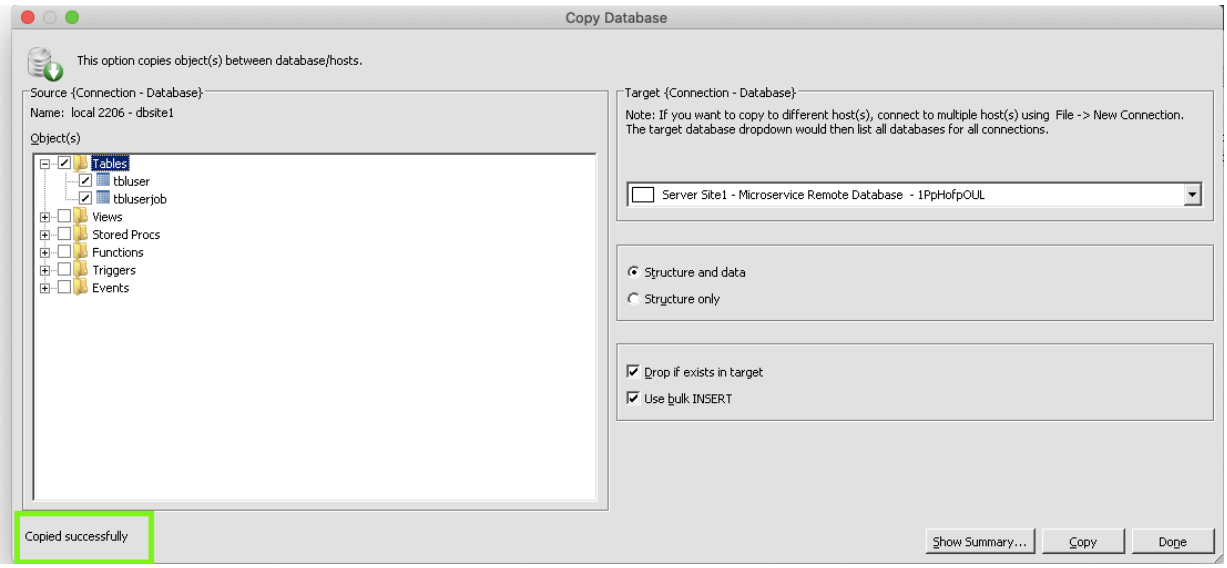




6. Transfer all you database tables from Local Database to your Remote.
    1. In my case I do this using SQLYog Copy Database to other Database Location
    2. You backup a .sql file of your local database and open the file with notepad++ and execute the .sql in your remote mysql database
    3. In my case SQLYog is shown below from Local ddsbe1 to Remote Database Site1
        1. Right click your local database, in my case it is **dbsite1** and click **Copy Database to Different Host and** a form will show below
        2. Select the objects to Transfer, in my case, I only select tables
        3. Select the Remote Database Server Connection, in my case it is the Server Site1

4. I specify to migrate to migrate the Structure and data
5. I specify to delete existing table is already exists in the destination database
6. Click Copy
7. I do the same for Database Site 2 and Gateway <— ***in preparation for Site2 and Gateway Deployment***

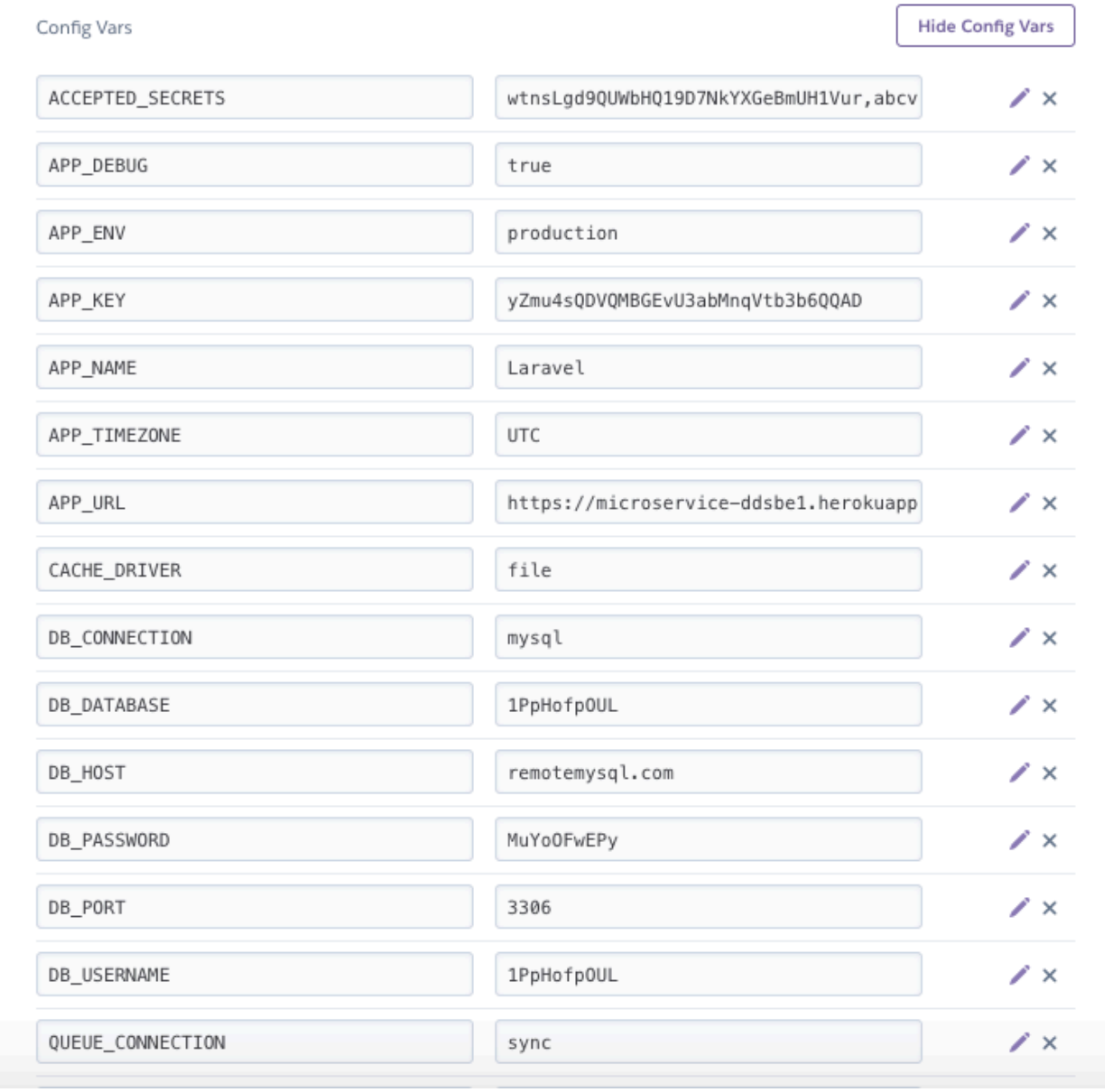




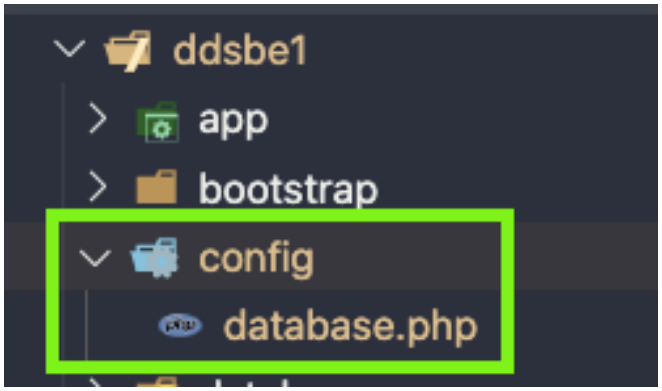
7. LETS CONFIGURE THE DATABASE SETUP and remaining config OF OUR LUMEN .env of

SITE1 using Heroku CLI

1. DB_CONNECTION=**mysql**
2. DB_HOST=remotemysql.com
3. DB_PORT=**3306**
4. DB_DATABASE=**1PpHofpOUL**
5. DB_USERNAME=**1PpHofpOUL**
6. DB_PASSWORD=**MuYoOFwEPy**
7. CACHE_DRIVER**=file**
8. QUEUE_CONNECTION**=sync**
9. ACCEPTED_SECRETS**=wtnsLgd9QUWbHQ19D7NkYXGeBmUH1Vur,abcvdjadfk,uyuyuy**



8. ADD the database configuration in our lumen site 1 project so that it will be used by Heroku
    1. Goto to your site1 folder root directory **ddsbe1**
    2. Create a folder name **config**
        1. Under config create a filename **database.php**



2. Copy and paste the following code to your database.php file <— I will attach the database.php for a more correct code

```
<?php

return [

    /*

|--------------------------------------------------------------------------
    | Default Database Connection Name

|--------------------------------------------------------------------------
    |
    | Here you may specify which of the database connections below you wish
    | to use as your default connection for all database work. Of course
    | you may use many connections at once using the Database library.
    |
    */
```

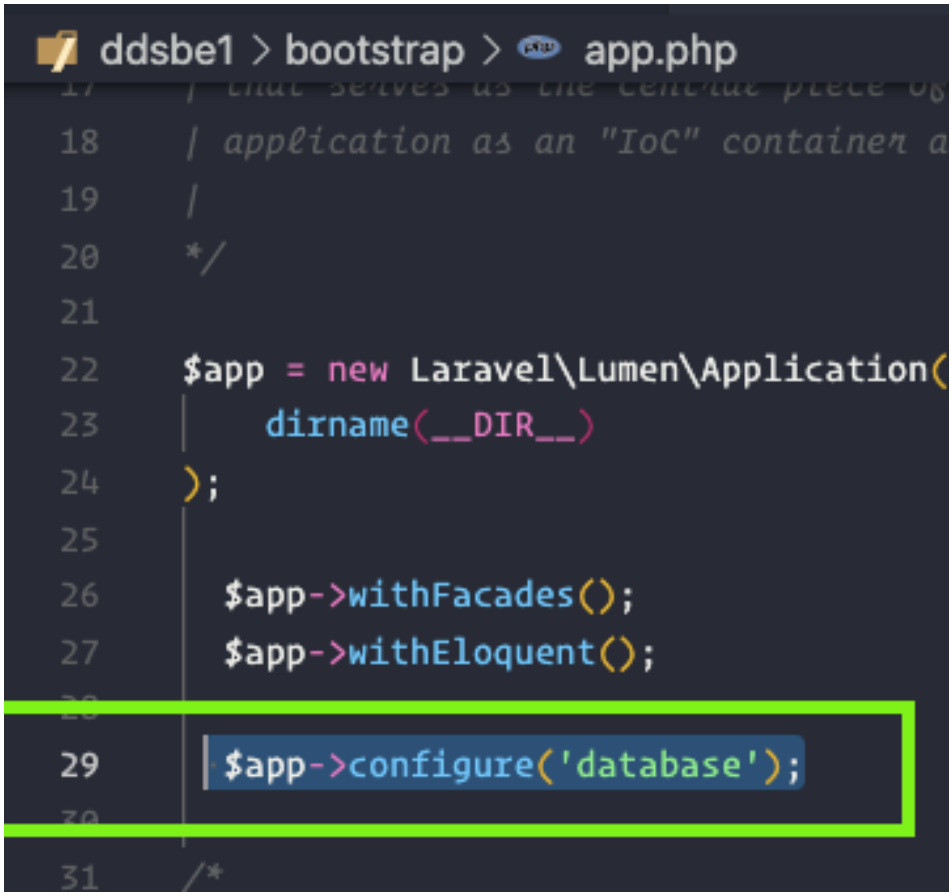
```
    'default' => env('DB_CONNECTION',
'mysql'),

    /*

|--------------------------------------------------------------------------
    | Database Connections

|--------------------------------------------------------------------------
    |
    | Here are each of the database
connections setup for your application.
    | Of course, examples of configuring
each database platform that is
    | supported by Laravel is shown below to
make development simple.
    |
    |
    | All database work in Laravel is done
through the PHP PDO facilities
    | so make sure you have the driver for
your particular database of
    | choice installed on your machine
before you begin development.
    |
    */

    'migrations' => 'migrations',
    'connections' => [

        'mysql' => [
```

```
            'driver'       => 'mysql',
            'host'         => env('DB_HOST', null),
            'port'         => env('DB_PORT', null),
            'database'     => env('DB_DATABASE',
null),
            'username'     =>
env('DB_USERNAME', null),
            'password'     =>
env('DB_PASSWORD', null),
            'unix_socket'   => env('DB_SOCKET',
''),
            'charset'      => env('DB_CHARSET',
'utf8'),
            'collation'    => env('DB_COLLATION',
'utf8_unicode_ci'),
            'prefix'       => env('DB_PREFIX', ''),
            // 'strict'       =>
env('DB_STRICT_MODE', false),
            // 'engine'       => env('DB_ENGINE',
null),
            // 'timezone'     =>
env('DB_TIMEZONE', '+00:00'),
        ],

        'sqlsrv' => [
            'driver'       => 'sqlsrv',
            'host'         => env('DB_HOST1', null),
            'port'         => env('DB_PORT1', null),
            'database'     =>
env('DB_DATABASE1', null),
            'username'     =>
env('DB_USERNAME1', null),
            'password'     =>
```

```
            env('DB_PASSWORD1', null),
                'charset'      => env('DB_CHARSET',
'utf8'),
                'prefix'       => env('DB_PREFIX', ''),
            ],

            'testing' => [
                'driver'       => 'sqlite',
                'database'     => ':memory:',
                'host'         =>
database_path('testing.sqlite'),
                'prefix'       => env('DB_PREFIX', ''),
            ],
        ],

    ];
```

3. REGISTER THIS NEW config in our app.php file in **ddsbe1 > bootstrap > app.php**

1. add this code in your app.php —> **$app->configure('database');**



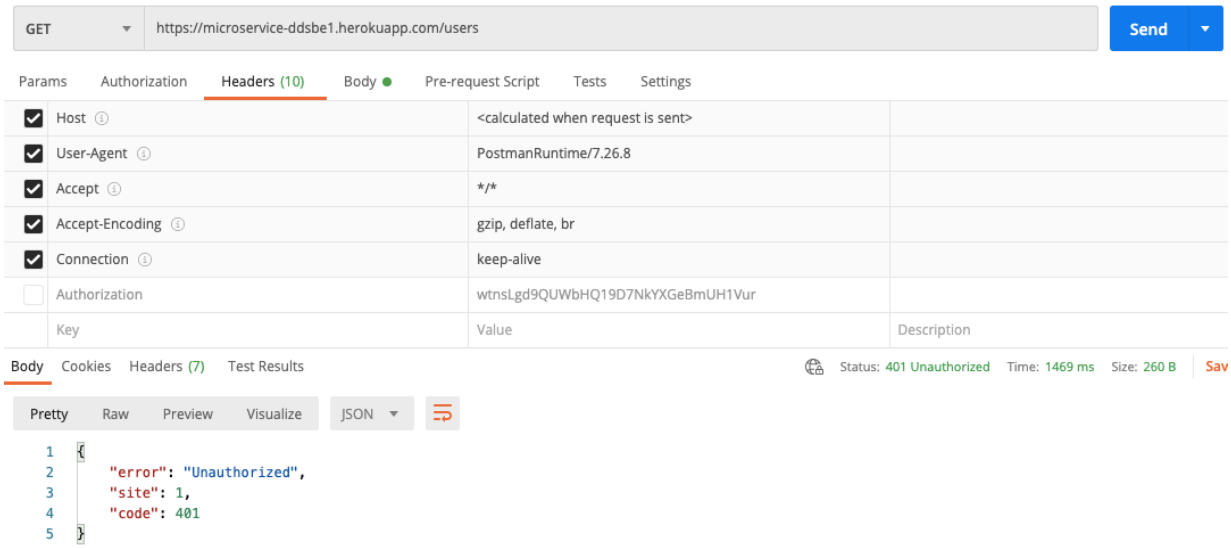
4. NOW PUSH ALL YOUR CHANGES TO YOUR GITHUB AND HEROKO REPOSITORY

1. > cd ddsbe1
2. > git add -A
3. > git commit -m “Added Config Database.php in App.php”
4. > git push origin master <— push first to Github
5. > git push heroku master <— push to Heroku
6. Restart Heroku Server by doing this command > **heroku ps:restart web -a microservice-ddsbe1**
7. Note: If you have any changes to your code, make sure to push it to heroku server before

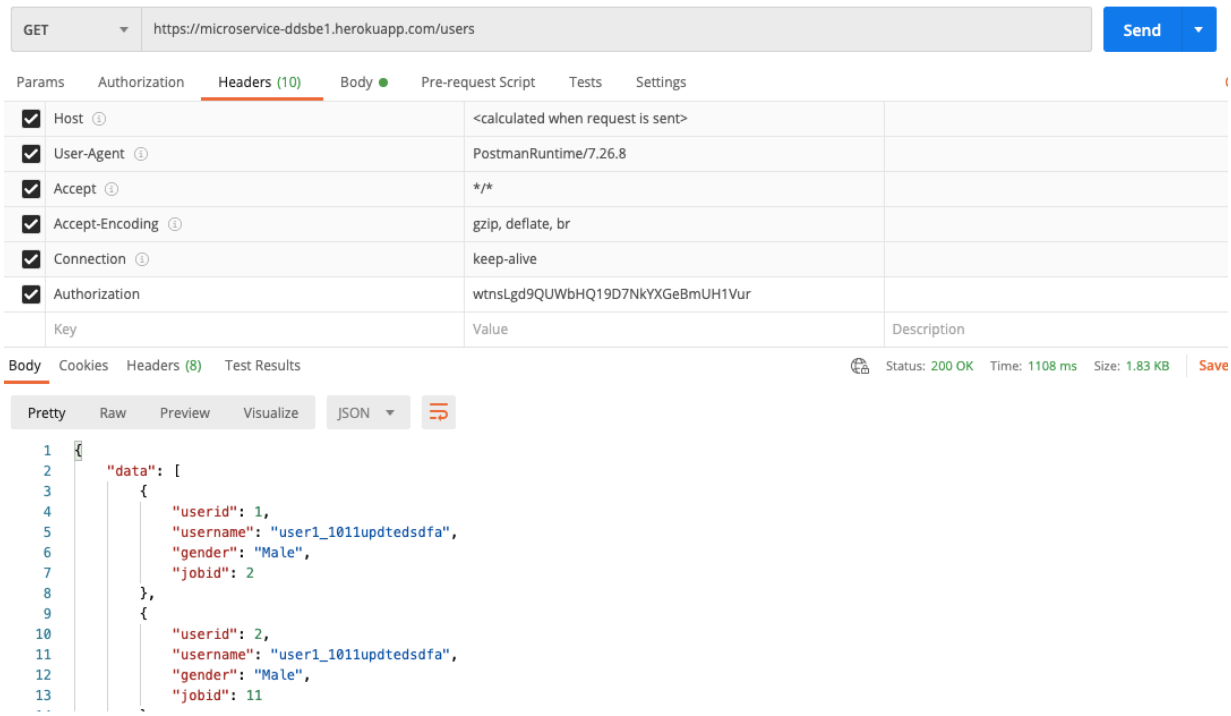
running

5. NOW LETS TEST OUR SITE 1 MICROSERVICE

1. TEST to get all users without authorization, there should be an error as shown below



2. TEST to get all users with authorization, there should be no error as shown below



CONGRATULATION YOU HAVE SUCCESSFULLY DEPLOYED YOUR MICROSERVICE TO LIVE HEROKU SERVER AND REMOTE MYSQL SERVER

NOW JUST DO THE SAME STEPS FOR YOUR MICROSERVICE SITE 2 AND MICROSERVICE GATEWAY

authorization key to connect to api gateway:

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiIzIiwianRpIjoiYzE5MzQ1NWUwMWRlZWQzOTM1NmIzOGE1M2QyYjQ1ODQ3ZTJhNmQ0YWIwODYxNDk2YjZlYWJhYzEyYmMyYmQzYmIwODVlZGQ1ODRmM2VjZjgiLCJpYXQiOjE2MjIwOTc3OTEsIm5iZiI6MTYyMjA5Nzc5MSwiZXhwIjoxNjUzNjMzNzkxLCJzdWIiOiIiLCJzY29wZXMiOltdfQ.aLEFOaQNf6z3yxIraTkxO0fbn3np9VPC0X6UpptdyXhAP0uYo4g5af6K3hwVDvnEs0iYvI-YhiZ1v-lDM_tdUbjv9xSV3Lb85wPW3yInzE1p6Wa97L1Z0RfyK87mLk3dAKdc4p2Z2bc6wAdpJyiVbDNWbHFZ6gPLRnNXJ4vr6gkb6Ic-zJ7bJKeaJYWzbCaUvxwNzx0ed0LnQBMFLA8TZkYelKqH0Eb013DgP8sXkB8Bo7-VRhBz5kj3L_qqYplNtOfqFIsAqdQ02S44t9UAJAscNY0iaglF2F6A6lMwkt19oXrCP539CgfbKF66K_ddGeDY35ji62GY5w0S9o-cRZDVUgdc3u6yIBrDuX9oRBtKXghORVuVgkDhmNX7Z9d7lD6dT-dMJIeSh0wCWs8eSnDDi2hs3nu3wiH-W1zxDIHKeWx0_ztTyW9-s3mI2O9mzjNz1xeZJ3j3NP1fCYAZcnwY-ik_NJaADAamNZ_Kjs8vqMg3Up78bDMkongWMyeMRyL77UT_VEzUfCH8XgU7CY5PaWFFZGhOmLowfTPm_8BnYx6pscuo4H8sWgCxh5jmyPiOKmO_0I83gNTHEelgiDbo8ED3-V5gARoNchli9nEZQIYCUmq-_zqNonKq10WdDFZl4Lzy83wGphpQ11VuoPnhRN8opkslJkyiitzbPyM