**Creating a new project**

Let’s start by creating a new Laravel project. Run the following command to create a new project.

composer create-project --prefer-dist laravel/laravel passport

**Install Package**

We need to install Laravel Passport package via the composer dependency manager. Run the following command to require the package.

composer require laravel/passport

**Adding Laravel Passport**

Laravel Passport requires some steps to set up properly.

**Service Provider**

I am using Laravel 6 which is the latest version of laravel right now which automatically registers package using package auto discovery. If you are using **laravel 5.4 or below**, you need to add Service Provider in the **config/app.php** file. So, open the file and add the Service Provider in the **providers** array.

|  |  |
| --- | --- |
| 1  2  3  4 | 'providers' => [      ....      Laravel\Passport\PassportServiceProvider::class,  ] |

### Migration and Installation

Setup database credentials in **.env** file. Laravel Passport comes up with migration for passport tables that are required to be in our database. Passport Migrations are used for storing tokens and client information. Run migration command to migrate schemas to your database.

php artisan migrate

Next, it is required to install passport using the command below. It will generate encryption keys required to generate secret access tokens.

php artisan passport:install

### 

### Passport Configure

In this step, we need to make changes in our Laravel application to complete passport configuration.

#### app/User.php

Add **Laravel\Passport\HasApiTokens** trait to your **User** model. It will provide few helper methods.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <?php    namespace App;    use Illuminate\Notifications\Notifiable;  use Illuminate\Foundation\Auth\User as Authenticatable;  use Laravel\Passport\HasApiTokens;    class User extends Authenticatable  {      use HasApiTokens, Notifiable;        /\*\*       \* The attributes that are mass assignable.       \*       \* @var array       \*/      protected $fillable = [          'name', 'email', 'password',      ];        /\*\*       \* The attributes that should be hidden for arrays.       \*       \* @var array       \*/      protected $hidden = [          'password', 'remember\_token',      ];  } |

#### AuthServiceProvider

Add Passport::routes method in the boot method of your **AuthServiceProvider**. It will generate necessary routes. This is how the **app/Providers/AuthServiceProvider.php** will look like after changes.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <?php    namespace App\Providers;    use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;  use Laravel\Passport\Passport;    class AuthServiceProvider extends ServiceProvider  {      /\*\*       \* The policy mappings for the application.       \*       \* @var array       \*/      protected $policies = [          'App\Model' => 'App\Policies\ModelPolicy',      ];        /\*\*       \* Register any authentication / authorization services.       \*       \* @return void       \*/      public function boot()      {          $this->registerPolicies();            Passport::routes();      }  } |

#### config/auth.php

In the **config/auth.php** file, set driver to the passport.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | return [      ....        'guards' => [          'web' => [              'driver' => 'session',              'provider' => 'users',          ],            'api' => [              'driver' => 'passport',              'provider' => 'users',          ],      ],        ....  ] |

## Create Route

Let’s create API routes. Add routes in the **routes/api.php** file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Route::post('login', 'PassportController@login');  Route::post('register', 'PassportController@register');    Route::middleware('auth:api')->group(function () {      Route::get('user', 'PassportController@details');        Route::resource('products', 'ProductController');  }); |

## Create Controller for Authentication

Let’s set up our logic for Authentication. Create Passport Controller by running the following command.

php artisan make:controller PassportController

Copy the contents below to **app/Http/Controllers/PassportController.php**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | <?php    namespace App\Http\Controllers;    use App\User;  use Illuminate\Http\Request;    class PassportController extends Controller  {      /\*\*       \* Handles Registration Request       \*       \* @param Request $request       \* @return \Illuminate\Http\JsonResponse       \*/      public function register(Request $request)      {          $this->validate($request, [              'name' => 'required|min:3',              'email' => 'required|email|unique:users',              'password' => 'required|min:6',          ]);            $user = User::create([              'name' => $request->name,              'email' => $request->email,              'password' => bcrypt($request->password)          ]);            $token = $user->createToken('TutsForWeb')->accessToken;            return response()->json(['token' => $token], 200);      }        /\*\*       \* Handles Login Request       \*       \* @param Request $request       \* @return \Illuminate\Http\JsonResponse       \*/      public function login(Request $request)      {          $credentials = [              'email' => $request->email,              'password' => $request->password          ];            if (auth()->attempt($credentials)) {              $token = auth()->user()->createToken('TutsForWeb')->accessToken;              return response()->json(['token' => $token], 200);          } else {              return response()->json(['error' => 'UnAuthorised'], 401);          }      }        /\*\*       \* Returns Authenticated User Details       \*       \* @return \Illuminate\Http\JsonResponse       \*/      public function details()      {          return response()->json(['user' => auth()->user()], 200);      }  } |

Let me explain what’s happening in the code above

In the **register** method, we validate the request data and then create the user. We then create the token using the **createToken** method and passing the name as an argument. Finally, we return token in a JSON response.

In the **login** method, we attempt to authenticate using the request details. Then, we return an appropriate response based on the success or failure of the attempt.

In the **details** method, we simply return the user eloquent model.

## Create Product CRUD

Let’s create a product CRUD. Run the following command to create Product Model, Migration and Controller.

php artisan make:model Product -mc

It will create a new database migration file **create\_products\_table.php** in **database/migrations** directory. Update the **up** method to the code below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public function up()  {      Schema::create('products', function (Blueprint $table) {          $table->increments('id');          $table->integer('user\_id');          $table->string('name');          $table->integer('price');          $table->timestamps();            $table->foreign('user\_id')              ->references('id')              ->on('users');      });  } |

Now, add a **fillable** property to the **Product** model. Open the **Product.php** file in the **app** directory.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php    namespace App;    use Illuminate\Database\Eloquent\Model;    class Product extends Model  {      protected $fillable = [          'name', 'price'      ];  } |

Now, run database migration.

php artisan migrate

Now, let’s add a product relationship method in the **app/User.php** file.

|  |  |
| --- | --- |
| 1  2  3  4 | public function products()  {      return $this->hasMany(Product::class);  } |

Open the **ProductController.php** file in **app/Http/Controllers** directory. Copy the contents below to the Product Controller.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | <?php    namespace App\Http\Controllers;    use App\Product;  use Illuminate\Http\Request;    class ProductController extends Controller  {      public function index()      {          $products = auth()->user()->products;            return response()->json([              'success' => true,              'data' => $products          ]);      }        public function show($id)      {          $product = auth()->user()->products()->find($id);            if (!$product) {              return response()->json([                  'success' => false,                  'message' => 'Product with id ' . $id . ' not found'              ], 400);          }            return response()->json([              'success' => true,              'data' => $product->toArray()          ], 400);      }        public function store(Request $request)      {          $this->validate($request, [              'name' => 'required',              'price' => 'required|integer'          ]);            $product = new Product();          $product->name = $request->name;          $product->price = $request->price;            if (auth()->user()->products()->save($product))              return response()->json([                  'success' => true,                  'data' => $product->toArray()              ]);          else              return response()->json([                  'success' => false,                  'message' => 'Product could not be added'              ], 500);      }        public function update(Request $request, $id)      {          $product = auth()->user()->products()->find($id);            if (!$product) {              return response()->json([                  'success' => false,                  'message' => 'Product with id ' . $id . ' not found'              ], 400);          }            $updated = $product->fill($request->all())->save();            if ($updated)              return response()->json([                  'success' => true              ]);          else              return response()->json([                  'success' => false,                  'message' => 'Product could not be updated'              ], 500);      }        public function destroy($id)      {          $product = auth()->user()->products()->find($id);            if (!$product) {              return response()->json([                  'success' => false,                  'message' => 'Product with id ' . $id . ' not found'              ], 400);          }            if ($product->delete()) {              return response()->json([                  'success' => true              ]);          } else {              return response()->json([                  'success' => false,                  'message' => 'Product could not be deleted'              ], 500);          }      }  } |

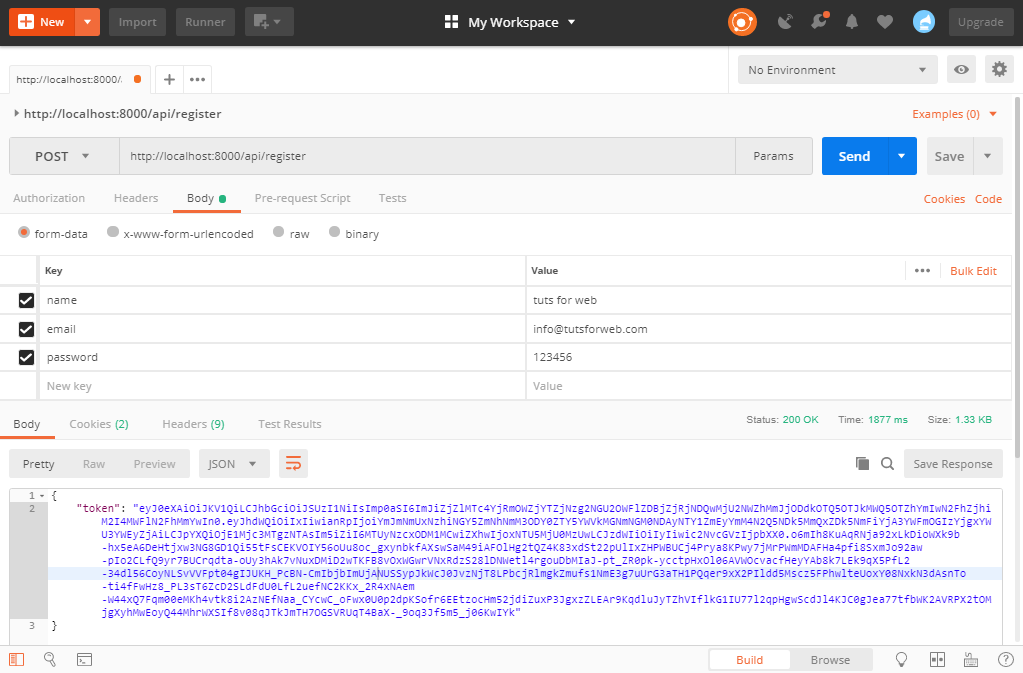
## Testing

Now, our logic is complete, let’s start testing. We will be testing it on PHP development server but you can use virtual host if you want. Run the following command to serve the application on the PHP development server.

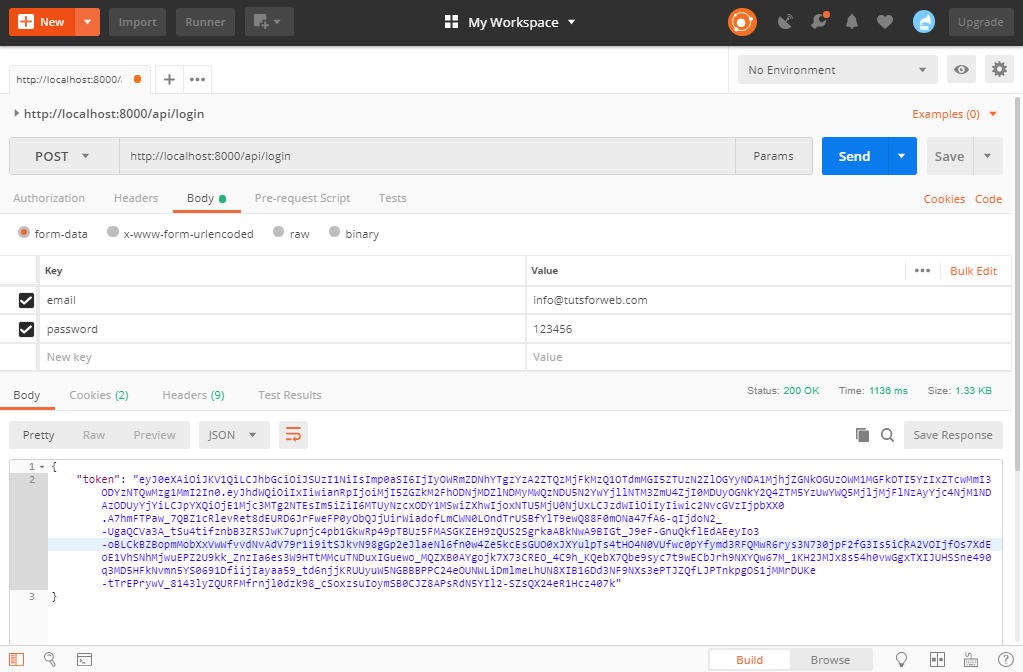
php artisan serve

Now, let’s test our API endpoints using an API testing tool like [Postman](https://www.getpostman.com/).

**Register API**



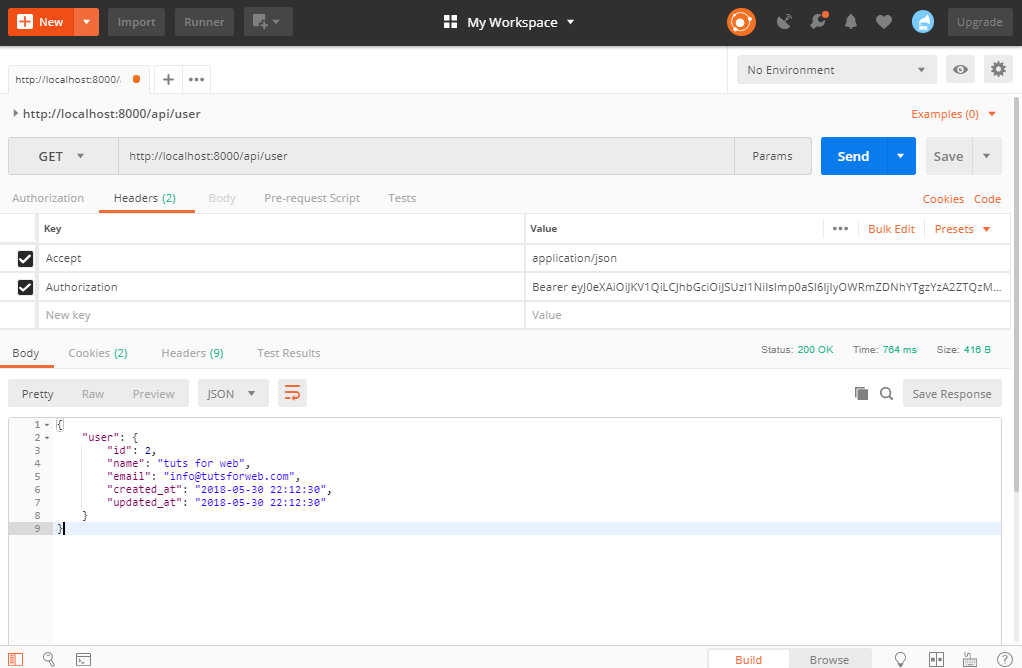
**Login API**



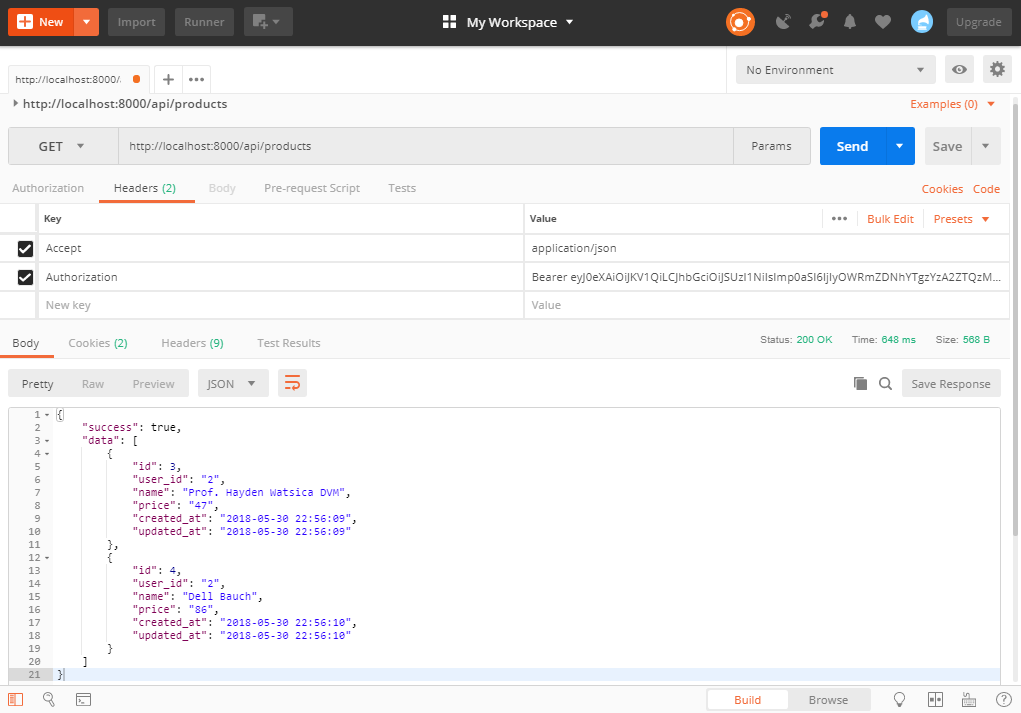
**Details API**

When testing Details API or any API that requires a user to be authenticated, you need to specify two headers. You must specify access token as a **Bearer** token in the **Authorization** header. Basically, you have to concatenate the access token that you received after login and registration with the **Bearer** followed by a space.

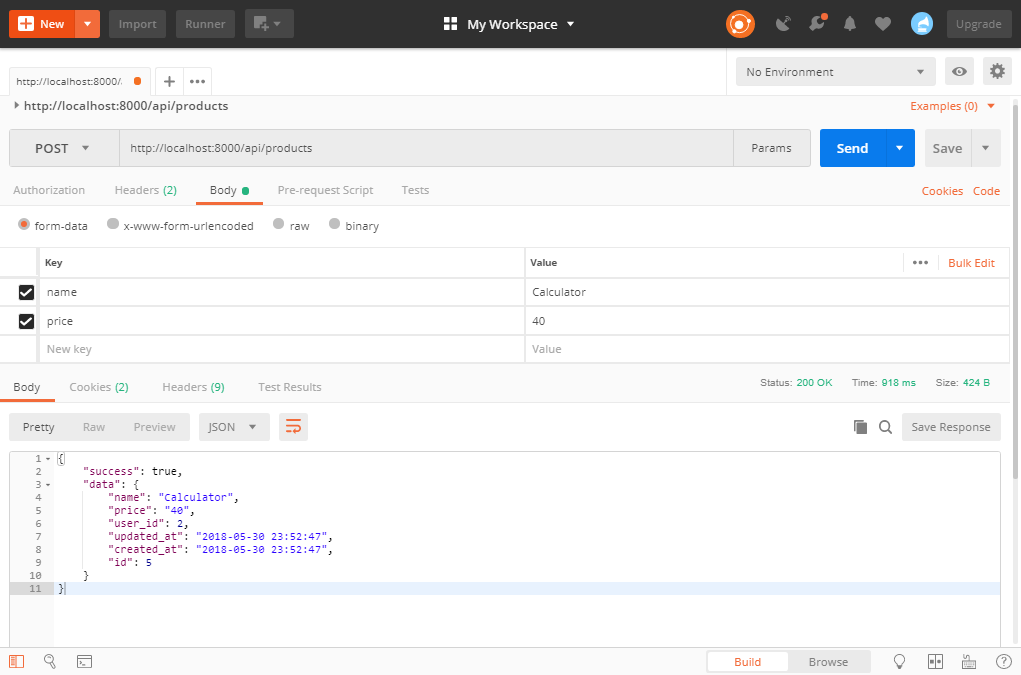
|  |  |
| --- | --- |
| 1  2  3  4 | 'headers' => [      'Accept' => 'application/json',      'Authorization' => 'Bearer '. $accessToken,  ] |



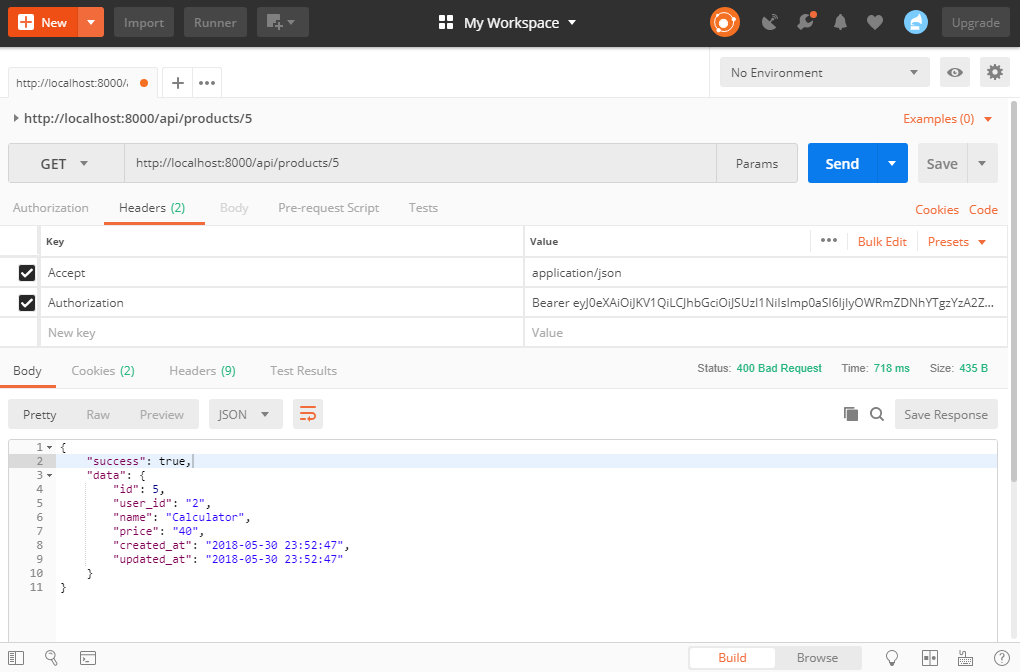
**Products Index API**



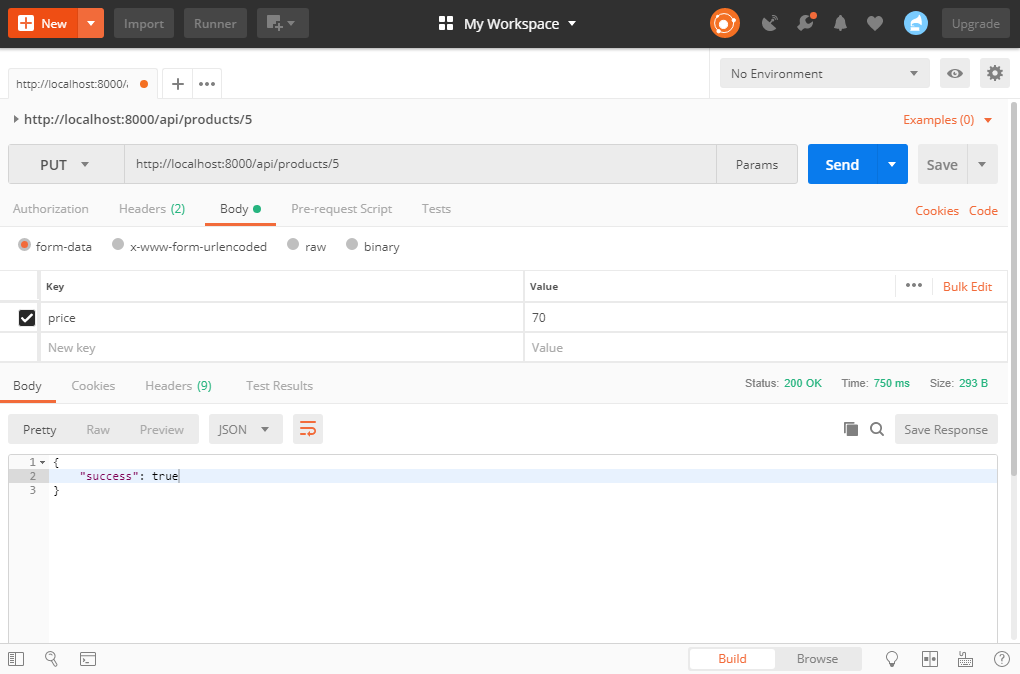
**Product Store API**



**Product Show API**



**Product Update API**



**Product Delete API**

