LARAVEL 8 - API REST

Assembler
School of Software Engineering

Assembler
School of Software Engineering

Contents

Assembler
School of Software Engineering

Contents

what are we going to code...

target

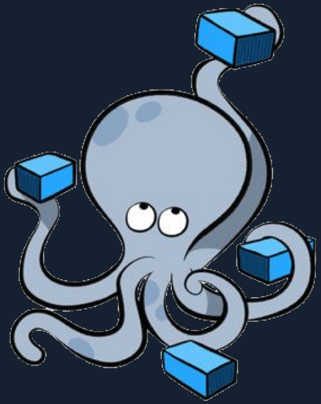
During today’s session

Today we are going to code and **API** for you a **properties** platform that contains information related to **Users,Properties and Favourites**.

Assembler
School of Software Engineering

All the services we need, tied together

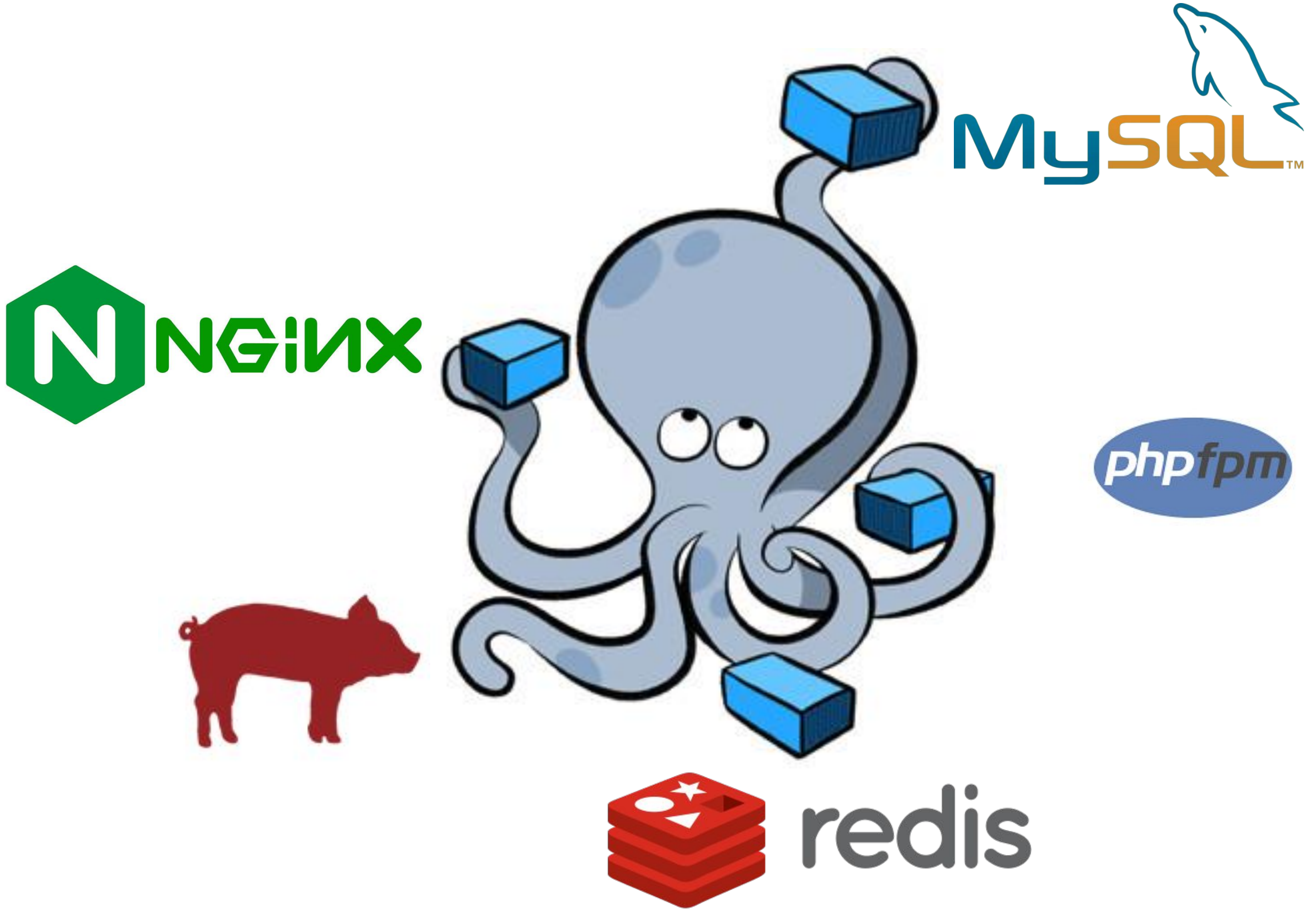
docker-compose & docker-build



Docker-compose

“Tool for defining and running multi-container Docker applications.With Compose, you use a YAML file to configure your application’s services. Then with a single commando, you create and start all the services from your configuration.”

https://docs.docker.com/compose/install/


MySQL™
NGINX
phpfpm
redis

```
// File that describes our Multi-Tier App
version: "3.7"
services:
    app:
        image: jperjim398/assemblerlaravel8:1.0
        …
    mysql:
        image: mysql:5.7.33
        …
    nginx:
        image: nginx:1.19.8-alpine
        …
    redis:
        image: redis:6.2.1-buster
        …
    mailhog:
        image: mailhog/mailhog:v1.0.1
        …
…
```

docker-compose.yml

File that describes all the services that compound our multi-tier application.

https://docs.docker.com/compose/compose-file/compose-file-v3/

Assembler
School of Software Engineering

Docker-compose commands

Some basic commands for

docker-compose --help

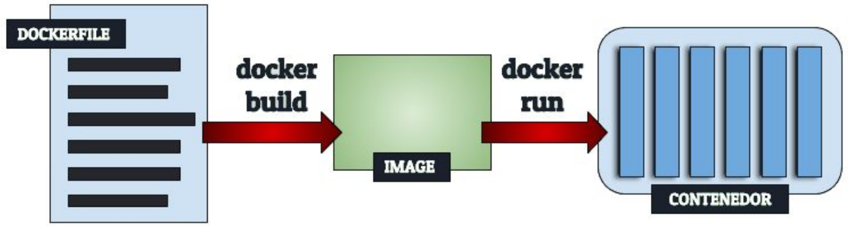
docker-compose command --help

```
// Create / start / build all the services
docker-compose up
// Start all the services
docker-compose start
// Remove all the services
docker-compose down
// Remove all the services (volumes included)
docker-compose down -v
// Stop all the services (volumes included)
docker-compose stop
//List all the containers related to the serives
docker-compose ps
//Execute command in a Service
docker-compose exec [-T] service_name command
```

Assembler
School of Software Engineering

Docker Builder

“Docker can build images....by reading instructions from a Dockerfile....that contains all the commands”a user could call on the command line to assemble an image...”



https://docs.docker.com/engine/reference/builder/

```
// Dockerfile
FROM php:8.0.3-fpm-buster
RUN docker-php-ext-install bcmath pdo_mysql
RUN apt-get update
RUN apt-get install -y git zip unzip iputils-ping nano
RUN pecl install xdebug
COPY --from=composer:latest /usr/bin/composer
/usr/bin/composer
EXPOSE 9000
```

My Own PHP-FPM image

We can build our own Docker images in a declarative way using Dockerfiles.

These built images can be pushed tou our DockerHub account using ***docker push.***

https://docs.docker.com/engine/reference/builder/

PHP Framework

What’s Laravel?

What’s Laravel?

“...is a **PHP Web Application Framework**...provides a structure and starting point for creating your application... providing powerful features such as thorough dependency injection, an expressive database abstraction layers, queues and scheduled jobs, unit a integration testing, and more...”

https://laravel.com/

Assembler
School of Software Engineering

Some concepts...

What is Laravel?

composer

Dependency Manager for PHP.

https://getcomposer.org/

migrations

“Migrations are like version control for your database, allowing your team to define and share the application’ database”

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

artisan

“CLI included with Laravel...provides a number of helpful commands that can assist you while building your application...”

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

Let’s start…

Setting Up the Environment

Setting Up the Environment

1. Clone Laravel 8.X repo.
2. Create volumes folders.
3. Create Nginx configuration file.
4. Give the right permissions to folders.
5. Run the docker-compose.yml

1. Clone Laravel Repo

We are going with the 8.X version...

https://github.com/laravel/laravel

```
>git clone --branch 8.x https://github.com/laravel/laravel.git src
```

2. Create volumes folder

For all the services whose data we want to persist.

> MySQL data folder
> Nginx data folder
> Redis data folder

3. Create Nginx configuration file

We need the Nginx container to connect with the PHP-FPM container to process requests to PHP files.

- **Port**
- **Default files**
- **Logs**
- **Root folder**
- **PHP-FPM configuration**

4. Give the right permissions to

Not needed for every OS (Linux for sure).

```
>[sudo] chown -R $USER:www-data bootstrap/cache
>[sudo] chown -R $USER:www-data storage
```

5. Run the docker-compose file

Everything running at last....

```
>docker-compose up [-d]
```

Everything I need

Laravel set up & Migrations

Assembler
School of Software Engineering

Laravel Set Up

1. Install dependencies with composer.
2. Create my project’s .env file.
3. Generate encryption key.
4. Clear cache.
5. Execute initial migrations.
6. Install and configure additional packages.

1. Installing dependencies with composer

composer is a PHP dependency manager

https://getcomposer.org/

```
>docker-compose exec -T app composer install
```

2. Create my Project’s .env file

This file contains configuration info for database, encryption, mail, log etc...

```
>docker-compose exec -T app cp .env.example .env
```

Assembler
School of Software Engineering

.env DATABASE CONFIGURATION

This configuration should match container’s configuration expressed in the docker-compose.yml file

```
//.env file → Database configuration section
DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=laravel8
DB_USERNAME=laravel8
DB_PASSWORD=123456
```

3. Generate Encryption Key

Generated key keeps app’s data safe.

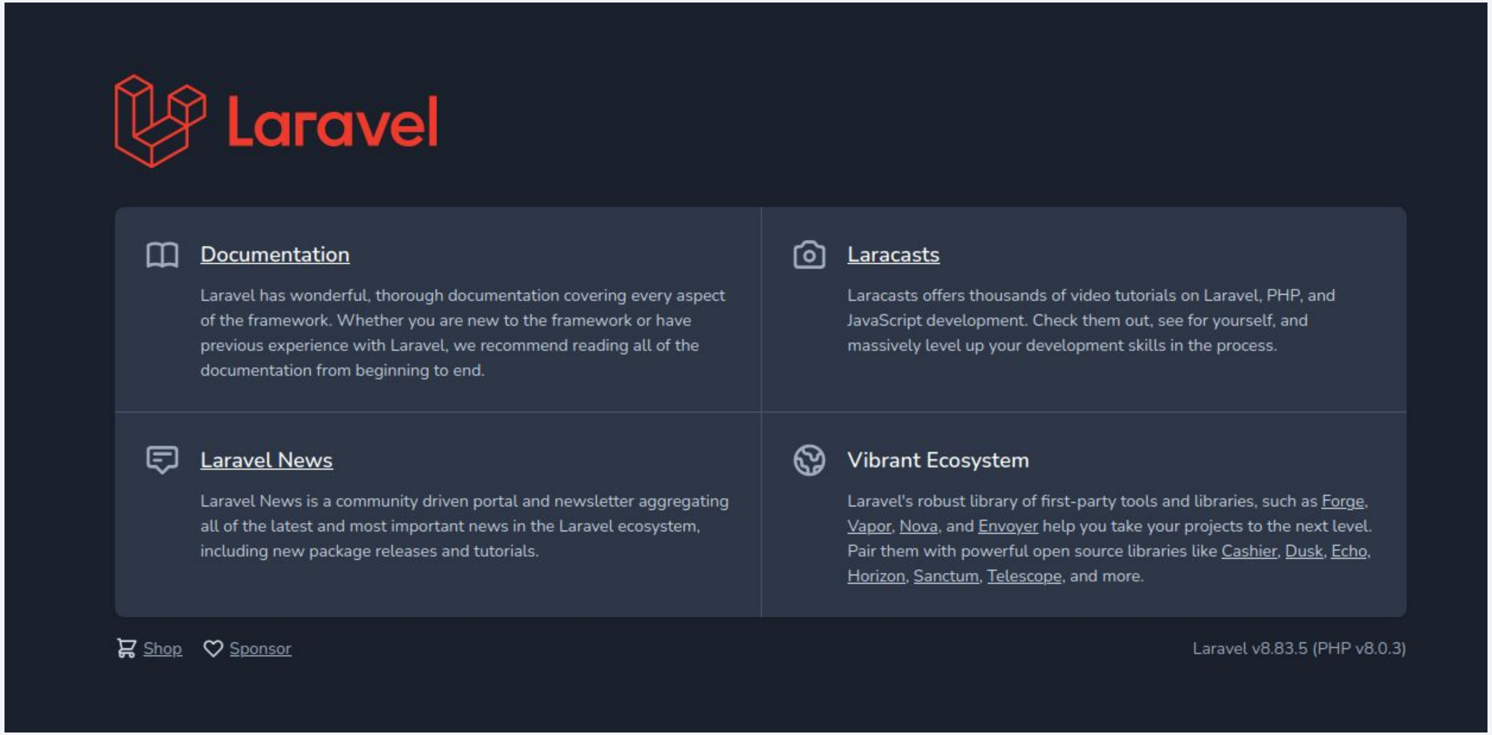
```
>docker-compose exec -T app php artisan key:generate
```

```
// .env file → APP configuration section
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:XXXXXXXXXXXXXX
APP_DEBUG=true
APP_URL=http://localhost
```

Laravel encryption Key

Docker cli docs can be found here:

https://docs.docker.com/engine/reference/commandline/cli/



Laravel

Documentation

Laravel has wonderful, thorough documentation covering every aspect of the framework. Whether you are new to the framework or have previous experience with Laravel, we recommend reading all of the documentation from beginning to end.

Laracasts

Laracasts offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.

Laravel News

Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.

Vibrant Ecosystem

Laravel's robust library of first-party tools and libraries, such as Forge, Vapor, Nova, and Envoyer help you take your projects to the next level. Pair them with powerful open source libraries like Cashier, Dusk, Echo, Horizon, Sanctum, Telescope, and more.

Shop Sponsor

Laravel v8.83.5 (PHP v8.0.3)

https://127.0.0.1:8100

4. Clear cache

Once we have modified our .env file it’s a good idea to clear our configuration cache

```
>docker-compose exec -T app php artisan config:clear
```

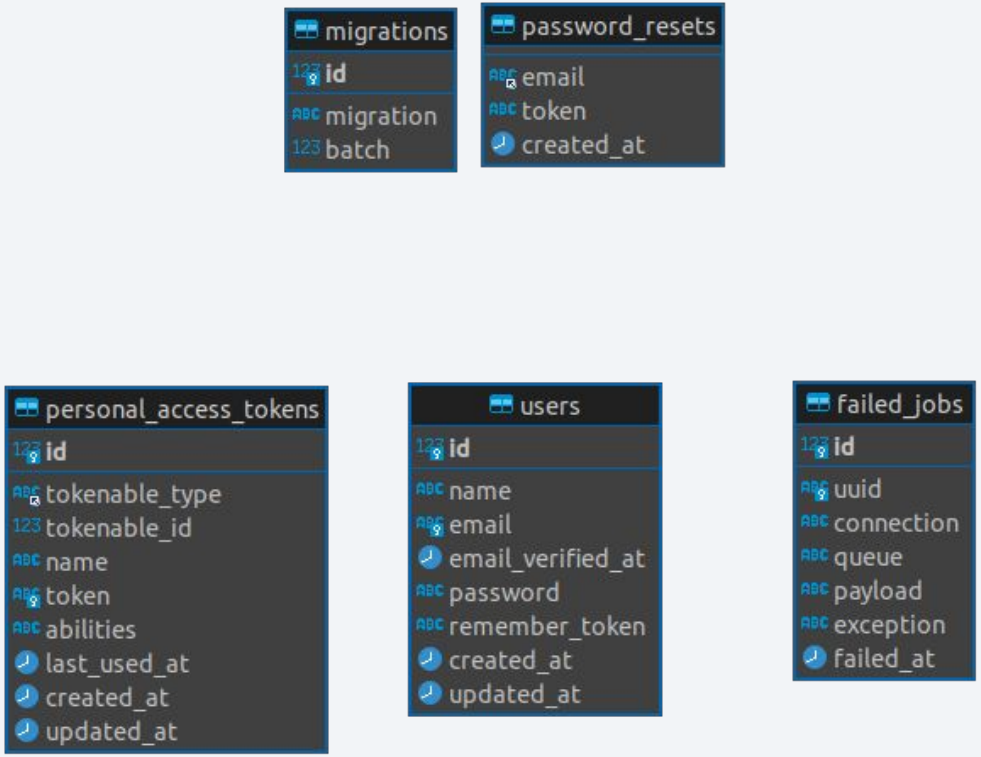
5. Execute initial database migrations (empty for now)-

“Migrations are like version control for your database, allowing your team to define and share the application’ database”

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

```
>docker-compose exec -T app php artisan migrate
```

Assembler
School of Software Engineering



6. Install and configure additional dependencies

Authenticate with Oauth Providers (socialite)

Featherweight authentication system for SPA (sanctum)

Caching solution (redis)

```
>docker-compose exec -T app composer require laravel/socialite
>docker-compose exec -T app composer require laravel/sanctum
>docker-compose exec -T app composer require predis/predis
```

more configurations..

Redis and MailHog configurations

Assembler
School of Software Engineering

Redis and MailHog configuration

1. Modify Laravel’s .env file.
2. Test Redis caching system adding testing routes.
3. Create email and template.
4. Test email send.
5. Execute initial migrations.
6. Install additional packages.

.env CONFIGURATION

We need to add/modify new configuration entries in the .env file

```
//.env configuration for Redis
CACHE_DRIVER=redis
…
REDIS_HOST=redis
…
REDIS_CLIENT=predis

//.env configuration for MailHog
MAIL_FROM_ADDRESS=someuser@gmail.com
```

Assembler
School of Software Engineering

```php
// Add Redis testing routes to Laravel web routes
use Illuminate\Support\Facades\Redis;

Route::get('/store', function() {

      Redis::set('foo', 'bar');

});

Route::get('/retrieve', function() {

      return Redis::get('foo');

});
```

Test Redis Caching System

Modify the src/routes/web.php file. This files contains all the routes related to our Laravel APP.

- /store to store some data using Redis
- /retrieve to get the previously stored data

Make sure to clear cache before restarting the multi-layer application.

Create email & template

1. Create the new Laravel Mail.
2. Set email template.
3. Create the template.
4. Create a new route to test email sending in Laravel.

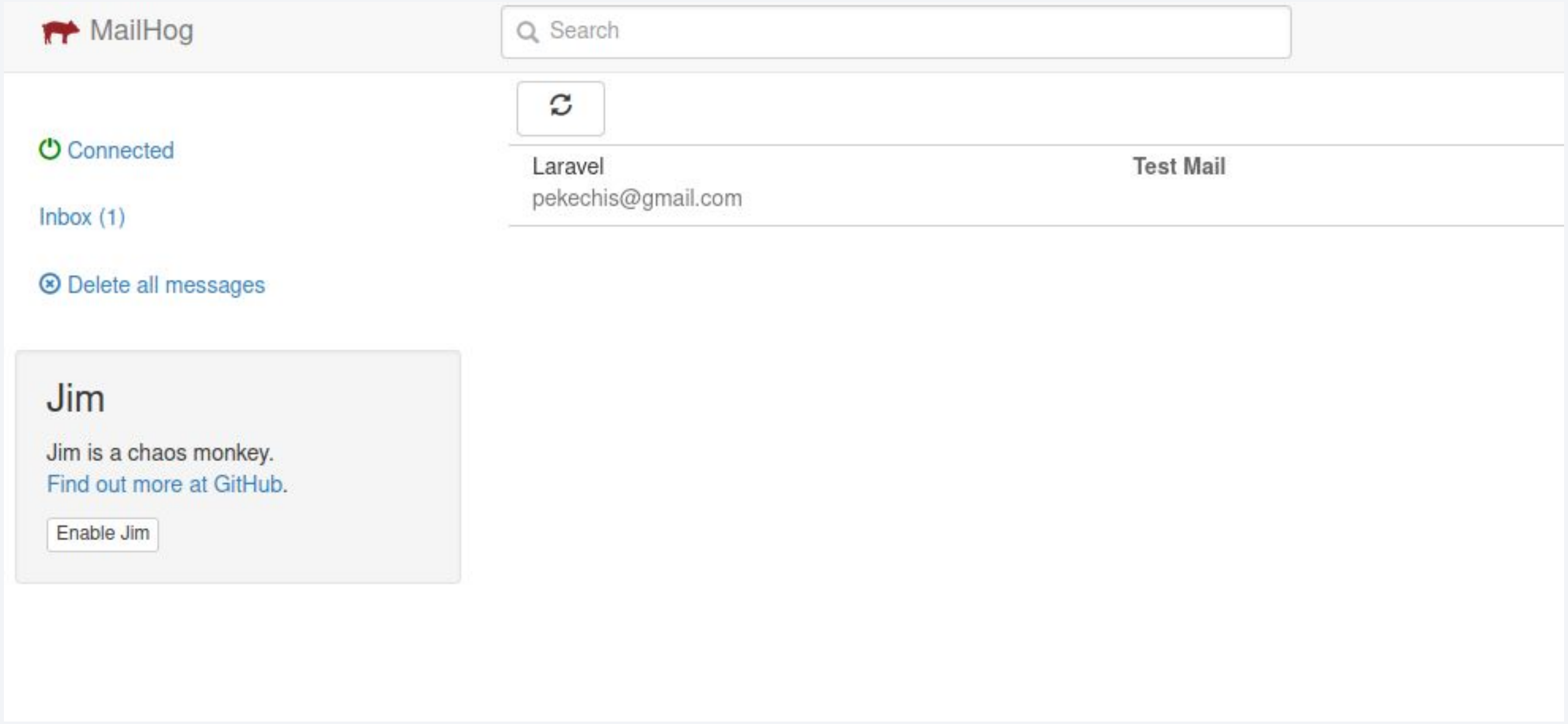
Make sure to clear cache before restarting the multi-layer application.

```
// Create a new laravel mail (create
app/Mail/TestMail.php
docker-compose exec -T app artisan make:mail TestMail

// Set email template in TestMail.php
public function build() {
      return $this->view('email.test');
}
// The template must be
// resources/views/email/test.blade.php

//New Route inside routes/wep.php
use App\Mail\TestMail;
use Illuminate\Support\Facades\Mail;
Route::get('/send-email', function() {
      Mail::to('pekechis@gmail.com')->send(new
TestMail);
});
```

Assembler
School of Software Engineering


MailHog

Search

Connected

Inbox (1)

Delete all messages

Jim

Jim is a chaos monkey.
Find out more at GitHub.

Enable Jim

Laravel
pekechis@gmail.com

Test Mail

Assembler
School of Software Engineering

Our API auth System

Laravel AUTH with Sanctum

Using Sanctum

1. Add Sanctum provider.
2. Register Sanctum Middleware for API.
3. Add Sanctum’s HasApiToken into the User Model.

1. Add Sanctum Provider

A Service Provider in Laravel simplifies the creation and configuration of some objects.

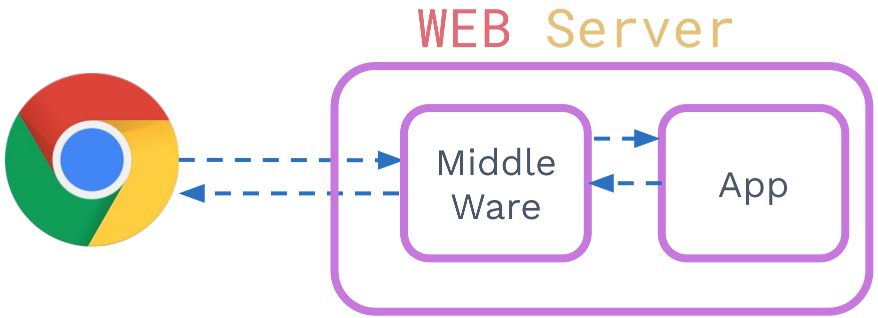
https://laravel.com/docs/8.x/providers

```
>docker-compose exec -T app php artisan vendor:publish
--provider="Laravel\Sanctum\SanctumServiceProvider"
```

Assembler
School of Software Engineering

2. Register Sanctum Middleware

A Middleware is an HTTP filter, transparent for the user, that usually is used for authentication purposes.



```
// Sanctum Middleware into the api array inside the
app/Http/Kernel.php
'api' => [
\Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAr
eStateful::class,
'throttle:api',
\Illuminate\Routing\Middleware\SubstituteBindings::class,
 ],
// Sanctum
use Laravel\Sanctum\HasApiTokens;
class User extends Authenticable {
   use HasApiTokens, HasFactory, Notifiable;
   protected $fillable = [
      'name',
      'email',
      'password',
   ];
   protected $hidden = [
      'password',
      'remember_token',
   ];
   protected $casts = [
      'email_verified_at' => 'datetime',
   ];
}
```

3. Add Sanctum’s API TOKEN

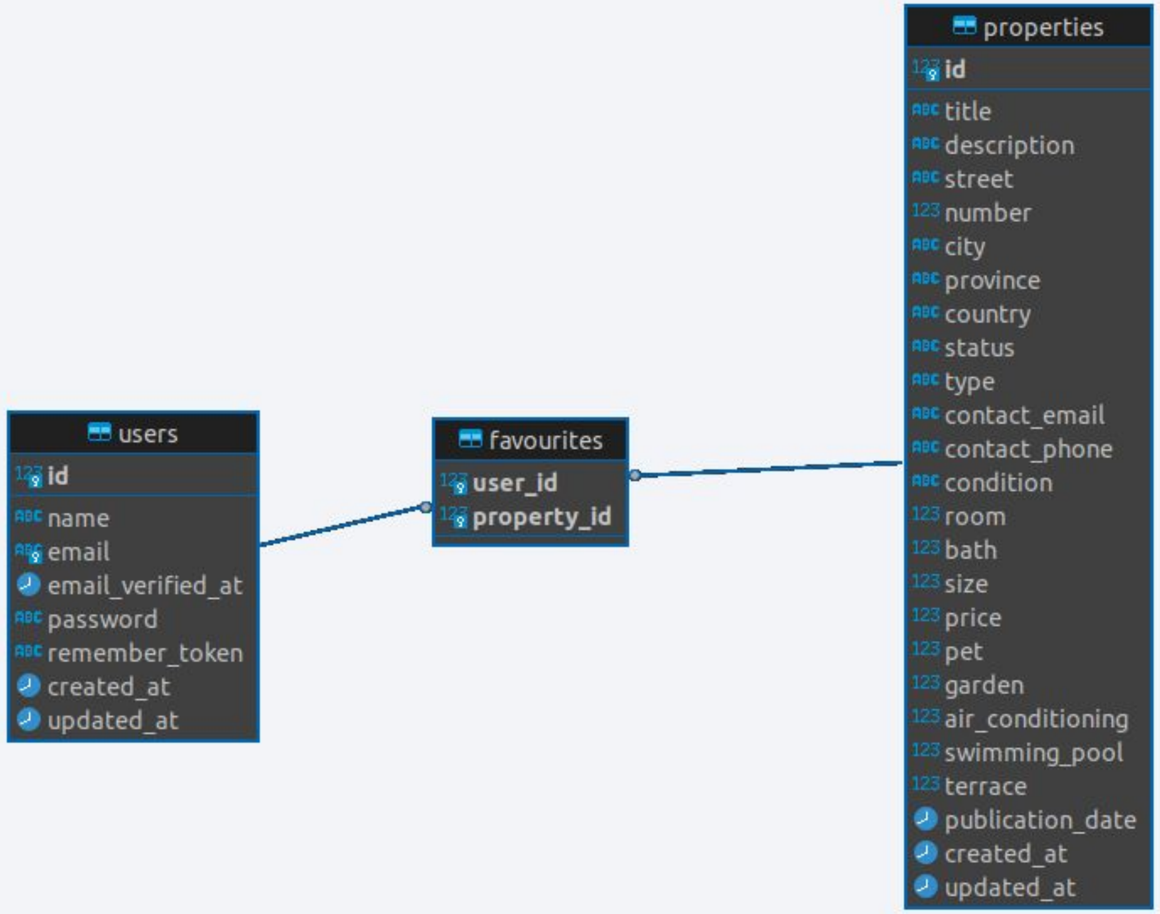
Check that the User Model has everything needed to use Sanctum.

Now, our project at last

Laravel Models,Controllers and API

Models and Controllers

1. Create Migration and Model.
2. Fill Migration and Model.
3. Create API Resources.
4. Create Controllers.
5. Create Rest API Routes.
6. Serve the App!!!!!


users
id
name
email
email_verified_at
password
remember_token
created_at
updated_at
favourites
user_id
property_id
properties
id
title
description
street
number
city
province
country
status
type
contact_email
contact_phone
condition
room
bath
size
price
pet
garden
air_conditioning
swimming_pool
terrace
publication_date
created_at
updated_at

1. Create Migration and Model

We need a migration for each one of the related tables on the DB.

We need a Model (ORM) for interacting with that table.

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

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

```
>docker-compose exec -T app php artisan make:model
Property --migration

//Creates a file inside /src/app/database/migrations
//Create a file inside /src/app/Model/
```

2.Fill Migration and Model

Migrations have two methods:

- up() ---> Create/Update the table in the database.
- down() ---> Reverse all changes done in up().

And run migrations once the migration file is created.

```
//Table creation
public function up() {
      Schema::create('properties', function (Blueprint
$table) {
            $table->id();
            $table->string('title');
            $table->text('description');
            $table->text('street');
            …..
      });
}

//Reverse the change (table drop)
public function down()   {
      Schema::dropIfExists('properties');
}
```

Assembler
School of Software Engineering

```
namespace App\Models;
use
Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Property extends Model
{
      use HasFactory;
      protected $fillable = [
      'title',
      'street',
      ….
      ];
      public function users() {
            //Relationships
      }
}
```

2. Fill Migration and Model

We need to create a class (Model) and tell Laravel which fields from the database are going to be retrieved.

It’s important to establish the relationships…..

2. Fill Migration and Model

“Migrations are like version control for your datase”...

Using ***php artisan migrate*** we can control this versioning operations.

On the right we have the most common operations.

You should read:

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

```
//RUN NEW migrations
docker-compose exec -T app php artisan migrate

//CHECK migrations status (something new?)
docker-compose exec -T app php artisan migrate:status

//UNDO LAST migration
docker-compose exec -T app php artisan migrate:rollback

//UNDO LAST N migrations
docker-compose exec -T app php artisan migrate:rollback
--step=N

//UNDO ALL Migrations
docker-compose exec -T app php artisan migrate:reset

//UNDO ALL and APPLY ALL
docker-compose exec -T app php artisan migrate:refresh
[--seed]
```

```
//Create the resource
docker-compose exec -T app php artisan make:resource
PropertyResource / UserResource

//Resource file /src/app/Http/Resources/
class Property extends JsonResource {

      public function toArray($request)
      {
            return [
                  'id' => $this->id,
                  'title' => $this->title,
                  'street' => $this->street,
                  'number' => $this->number,
                  …
            ];
      }
}
```

Assembler
School of Software Engineering

3. Create API Resources

“Resource classes allows to transform Models to JSON.

You may use toJson() methods but using resources allows more control and granularity.

i.e. : Depending on the use, including relationships….

https://laravel.com/docs/8.x/eloquent-resources

4. Create Controllers

- **BaseController:** With sendResponse() / sendError() generic operations.
- **AuthController:** Registration and SignIn.
- **Properties Controller:** CRUD operations for Property objects.

>File /src/app/Http/Controllers/API/BaseController.php
>File /src/app/Http/Controllers/API/AuthController.php
>File /src/app/Http/Controllers/API/PropertyController.php

It’s also possible to use artisan.
It’s also possible to use Resource Controllers
https://laravel.com/docs/8.x/controllers#resource-controllers

Assembler
School of Software Engineering

5. Create API Routes

API Routes are created inside the reoutes/api.php file and depending if they are PUBLIC or NOT should be guarded with SANCTUM Middleware.

Routes **MAP http requests to Controllers' functions.**

```
Route::post('login', [AuthController::class, 'signin']);
Route::post('register', [AuthController::class, 'signup']);

Route::get('properties',[PropertyController::class,'index']);
Route::get('properties/{id}',[PropertyController::class,'show']);

Route::middleware('auth:sanctum')->group(function() {

Route::post('properties',[PropertyController::class,'store']);
Route::delete('properties/{id}',[PropertyController::class,'destroy']);
Route::put('properties/{id}/users/{userid}',[PropertyController::class,'addUser']);
…..}
```

Let’s work and Test our API

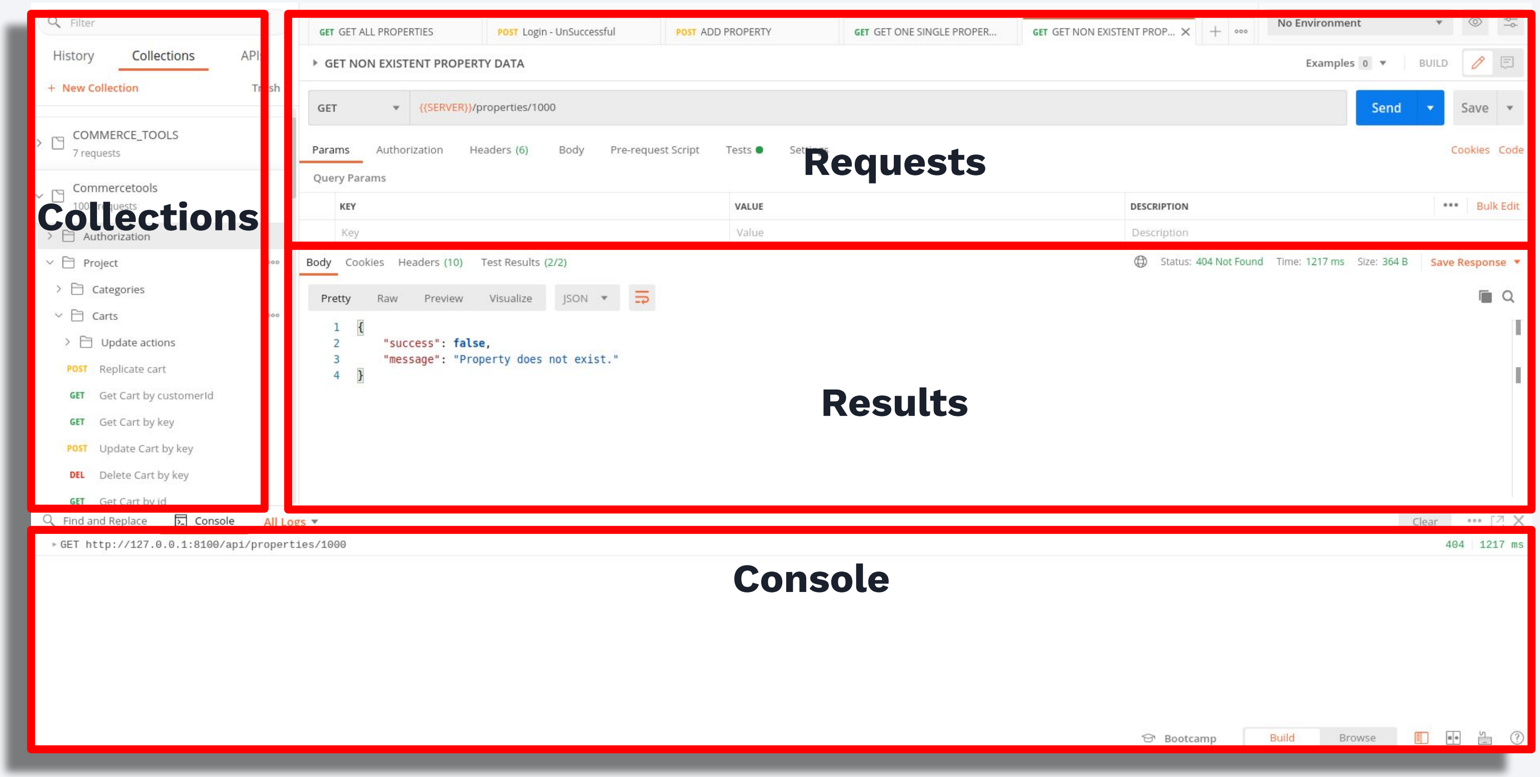
Using PostMan

How to use our API with POSTMAN

1. POSTMAN UI.
2. Creating POSTMAN Collections.
3. Creating and running HTTP Requests.
4. Collection’s vars.
5. Pre-Request Scripts and Postman Tests.
6. Running POSTMAN Collections from CLI (newman)

Assembler
School of Software Engineering

PostMan UI



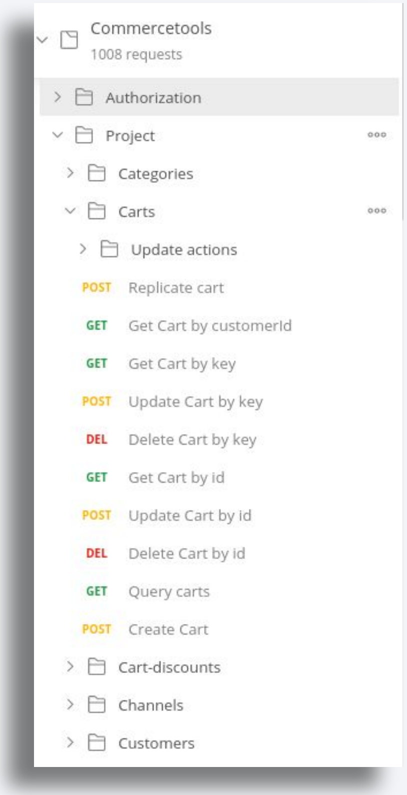
```
GET {{SERVER}}/properties/1000
```

```
{
    "success": false,
    "message": "Property does not exist."
}
```

```
GET http://127.0.0.1:8100/api/properties/1000                404   1217 ms
```

Assembler
School of Software Engineering

Creating PostMan Collections


Commercetools
1008 requests

Authorization

Project

Categories

Carts

Update actions

POST Replicate cart

GET Get Cart by customerId

GET Get Cart by key

POST Update Cart by key

DEL Delete Cart by key

GET Get Cart by id

POST Update Cart by id

DEL Delete Cart by id

GET Query carts

POST Create Cart

Cart-discounts

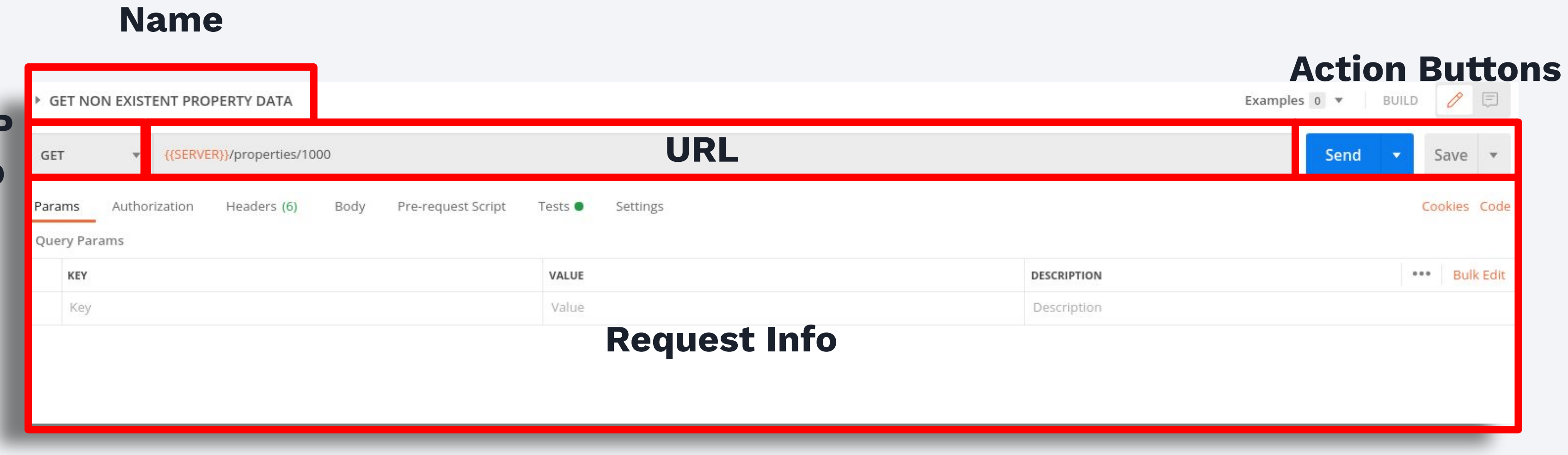
Channels

Customers

A **POSTMAN COLLECTION** are organized HTTP Requests that can be:

- **Executed** using Postman’s collections **Runner**.
- **Exported** to be shared with others.
- **Documented** automatically.

Creating HTTP Requests



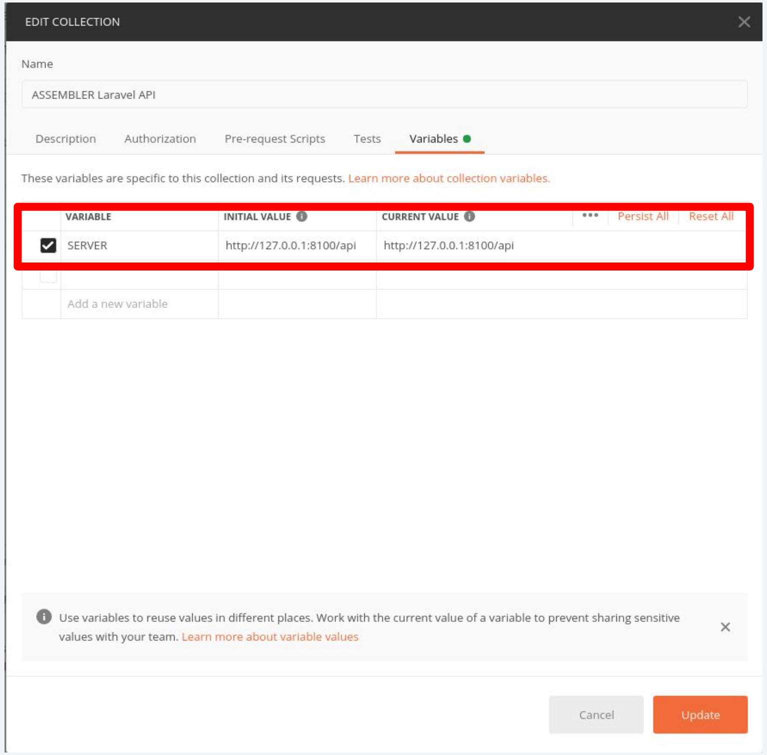
{{SERVER}}/properties/1000

Assembler
School of Software Engineering

Collection's Vars

Collection's VARS allows the user to store and reuse values inside a collection.

- Can be defined inside Collection's properties.
- Are accessed within the collection {{VAR_NAME}}
- Can be set inside Pre-Requests and Tests `pm.collectionVariables.set("VAR_NAME",data);`
- Can be retrieved inside Pre-Requests and Test `pm.collectionVariables.get("VAR_NAME");`


EDIT COLLECTION

Name

ASSEMBLER Laravel API

These variables are specific to this collection and its requests. Learn more about collection variables.

	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	SERVER	http://127.0.0.1:8100/api	http://127.0.0.1:8100/api
	Add a new variable		

Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. Learn more about variable values

Running Collections from Command Line

1. Installing Newman
2. Running exported Collection (json file)

1. **Installing Newman**

Newman is a command line collection runner for Postman.

https://www.npmjs.com/package/newman

- **Requires:** NPM(Node package manager) must be installed.

```
>[sudo] npm install -g newman
```

2. Running Exported Collections

If we want to integrate our Postman Requests into a CI/CD pipeline we can export them and run a full collection from CLI

```
>newman run assembler_collection.json
```

Testing using Code

API Testing

PHPUnit

Assembler
School of Software Engineering

Testing using Code
(just some examples)

1. Creating Tests.
2. Executing Tests.
3. Running Tests as Auth users.

Assembler
School of Software Engineering

```
//EACH TEST CLASS MUST EXTEND TestCase
class ExamplesTest extends TestCase
{

    //EACH FUNCTION IS A DIFFERENTS TEST
    public function exampleTest()
    {
        //TEST ACTIONS
        $response = $this->get('/');
        …..
        //TEST CHECKS
        $response->assertStatus(200);
    }
}
```

1. Create Tests

Laravel includes PHPUnit out of the box.

Tests must be located inside the tests folder.

- Unit folder contains unit testing for classes.
- Feature folder contains other kind of tests.
- Tests can be created using artisan

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

2 . Execute Tests

We can use artisan to run all the tests of our application

```
> docker-compose exec -T app php artisan test
```

3. Running Test as Auth Users

Laravel provides a bunch of useful functions for testing like:

- Getting random models.
- Act as registered / authenticated user.
- Many more.

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

```
public function testGetAllUsersWithPermission()
{
      $user = User::get()->random();
      $response =
$this->actingAs($user)->get('/api/users');
      $response->assertStatus(200);
}


public function testGetAllUsersWithoutPermission()
{
      $response = $this->get('/api/users');

      $response->assertStatus(500);
}
```

Going further...

Is there anything more?

- Seeding the database.
- Using SANCTUM alternative (Passport).
- Read Laravel DOCS (Trust me...it’s worthy).
- Artisan CheatSheet https://learninglaravel.net/cheatsheet/
- API Resource Naming.
- Resource Routes.
- POSTMAN Training.
- Integrate everything in my CI/CD workflow (Jenkins / GitHub Action etc...)
- Many more....

Assembler
School of Software Engineering

Questions?