# Apply Redis in PHP

# Installing

The same as last week. For code, checkout <https://gitlab.com/laravel100/tutorial-5>

# Applications of Redis in real-life

**PREREQUISITE: Prepare Laravel and check whether Laravel can save to Redis or not**

After installing Laravel, run php artisan serve to start

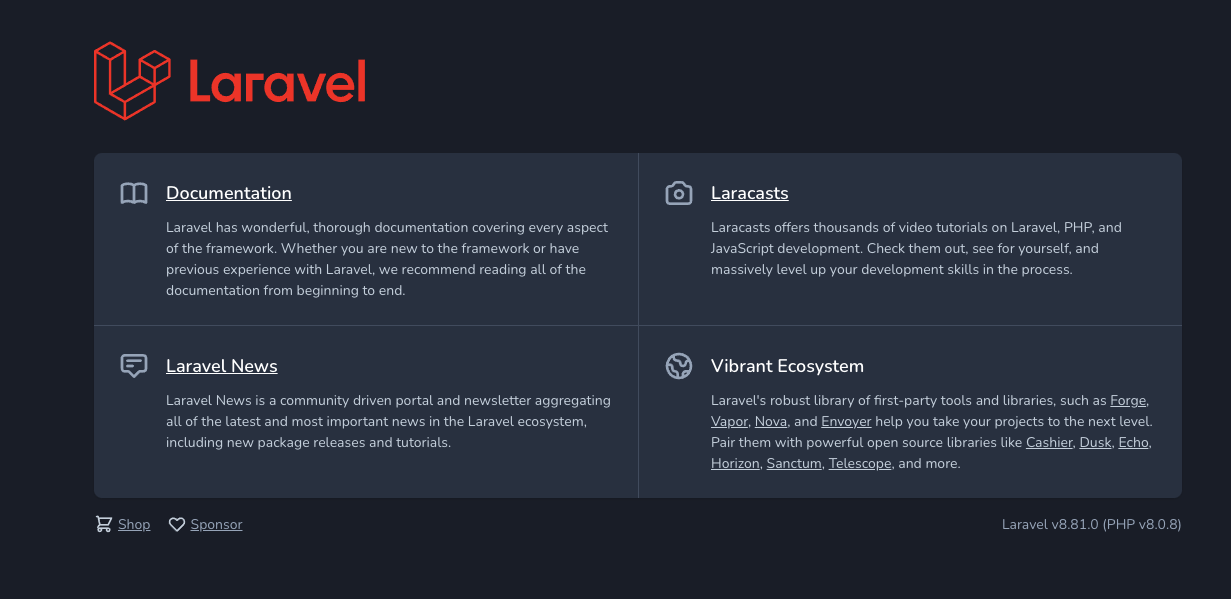


Figure 1: Domain setup

We need to setup Redis host in line 24

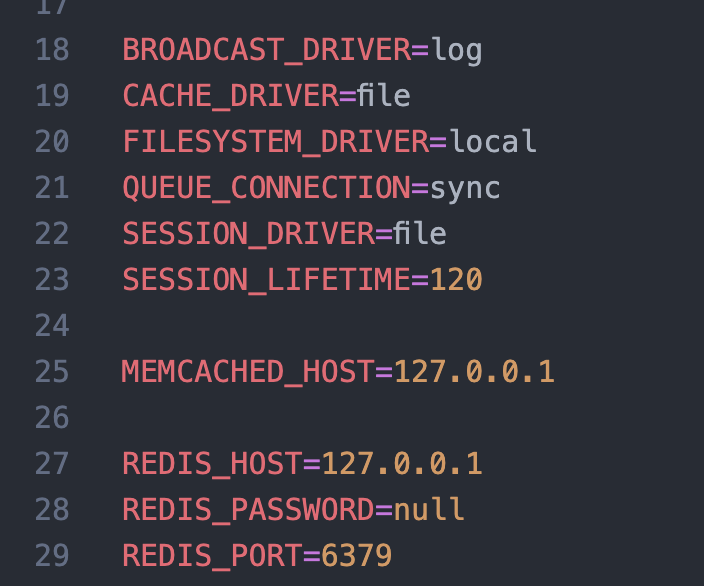
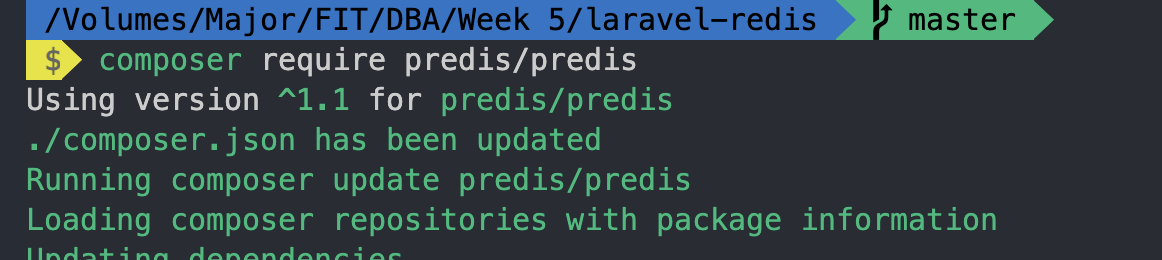


Figure 3: Set cache driver to Redis in line 24

The first use case when Redis comes into town is caching item. When user logs in to system successfully, we cache user information, since user information is rarely change.

Install package predis/predis to run Redis in Laravel by running this command in terminal



After installation, go to config/database.php and change REDIS\_CLIENT to predis

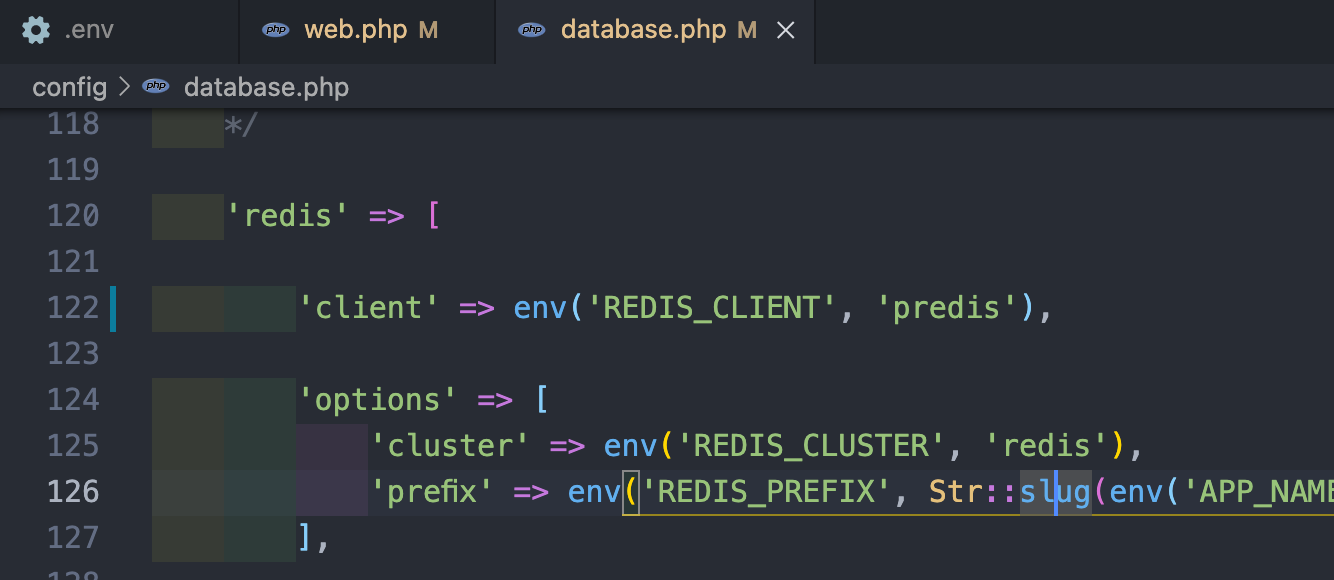


Figure 4: Change from phpredis to predis (Line 122)

Fields of users inside database includes

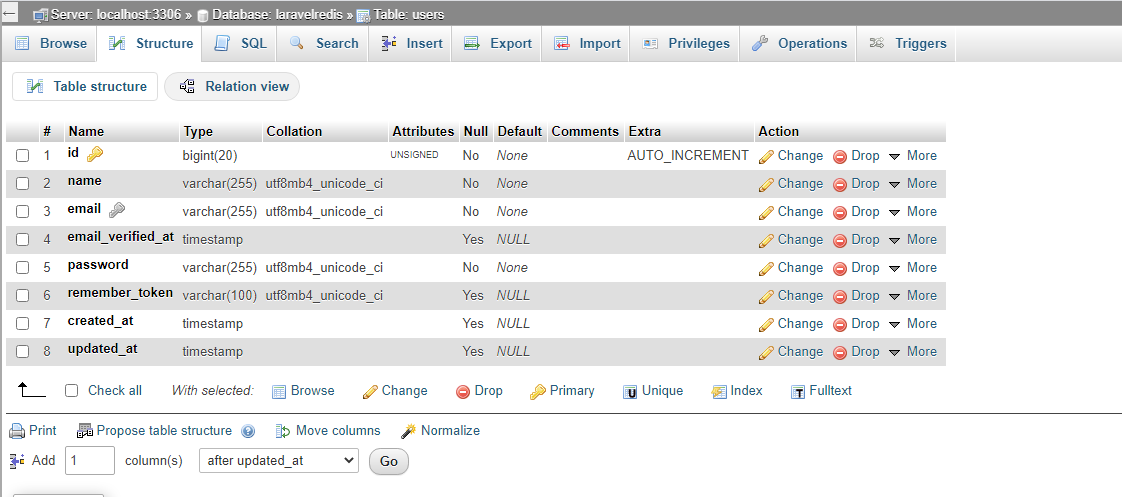


Figure 5: Users table

Giving Redis check by typing into routes/web.php

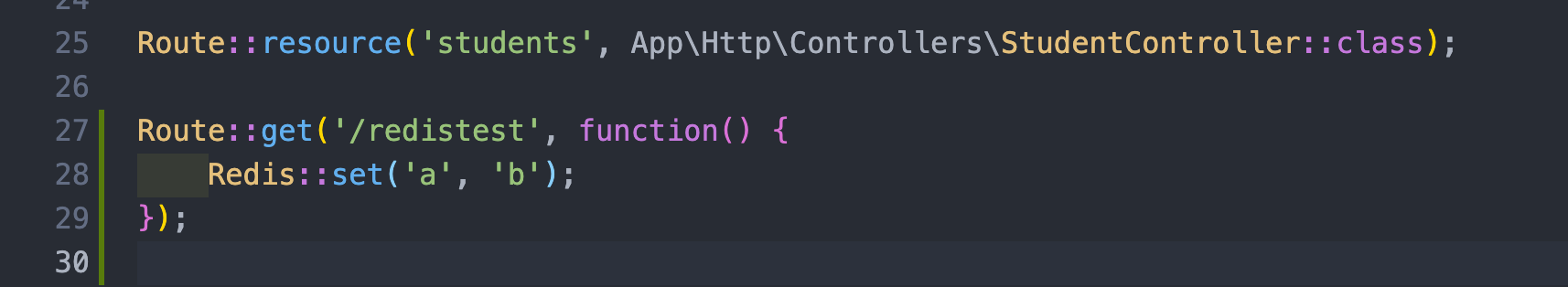


Figure 6: Redis check

Then go to url ‘/redistest’

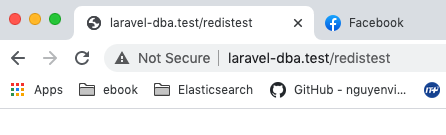


Figure 7: Redis test

Then check in redis-cli

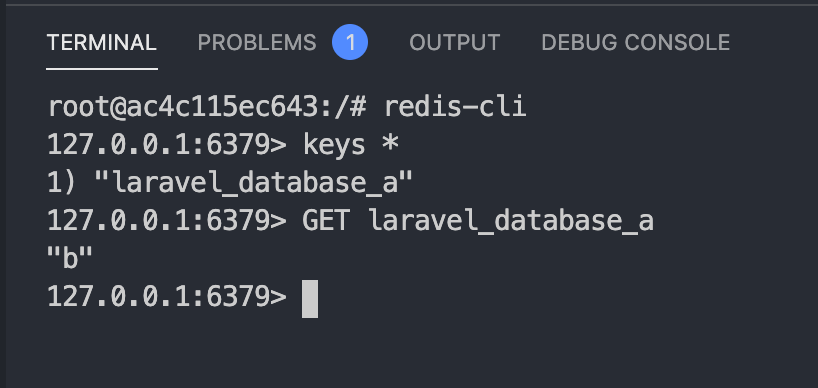


Figure 8:Check Redis-cli

By default: Laravel will add laravel\_database as prefix. This is used to differentiate between applications. You can change it to a name by changing cache\_prefix in config/database.php

(\*): If you want to delete all keys stored in Redis, run flushall in redis-cli

## USE CASE 1: Cache user information

Install Laravel Breeze as you do in 2 previous tutorials

Head to /login, you will see the following screen. This is login page of Laravel scaffolding

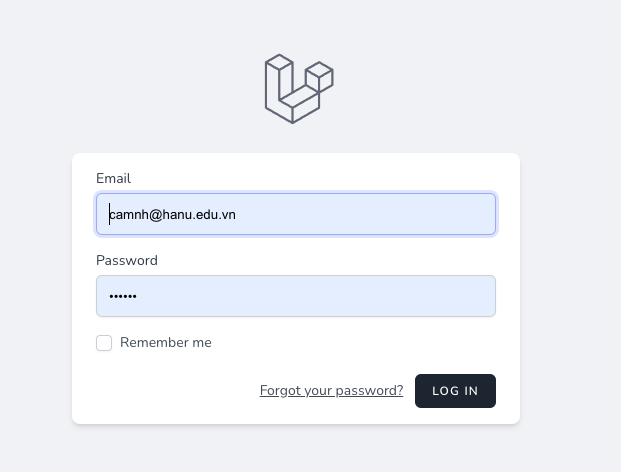


Figure 9: Login screen: /login

Since it’s the first time. User need to create an account; you need to experience the Registration and Login process in the interface by yourself.

However, the main thing is, after user logins to the system successfully. He can edit his own profile. So, our responsibility is to cache user\_profiles information to prevent request to database.

User profile info includes:

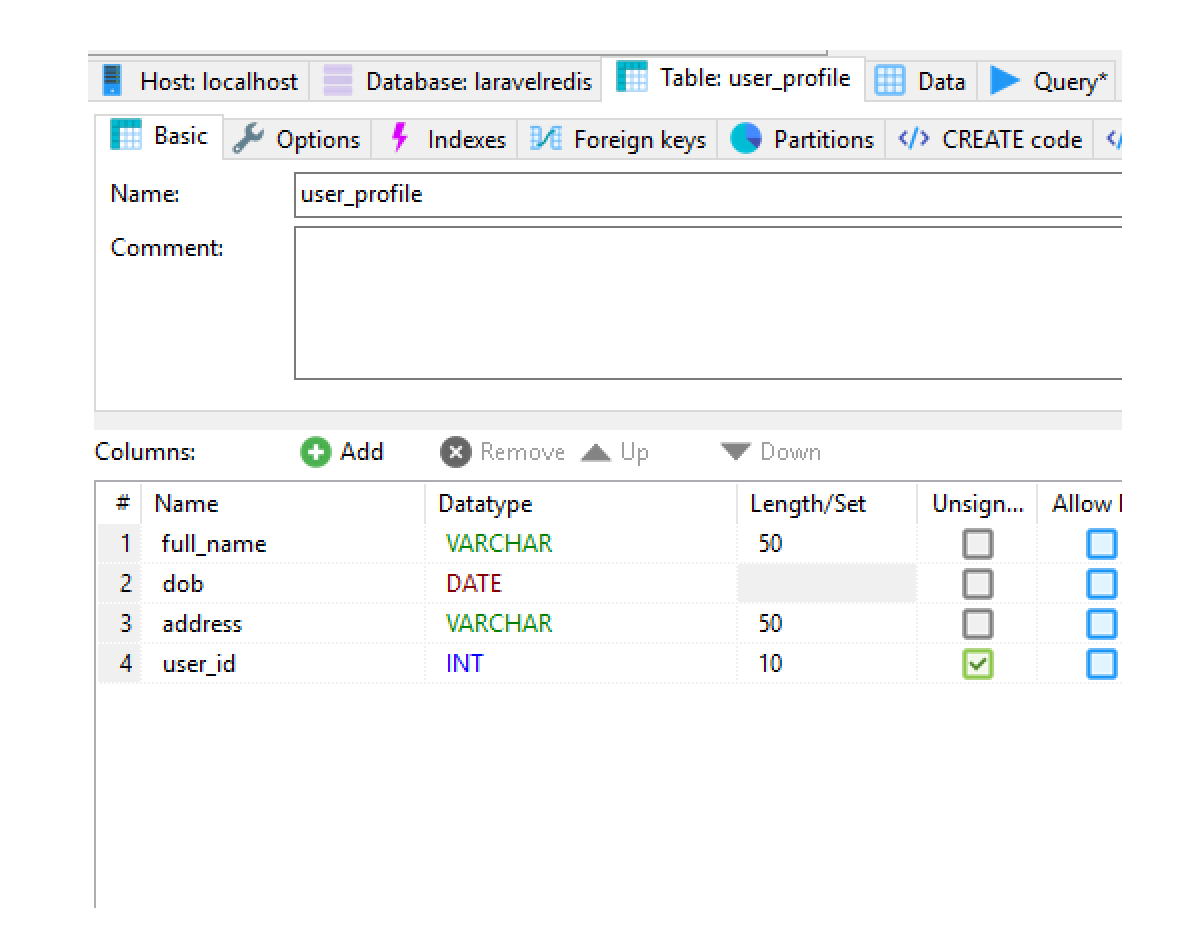


Figure 10: User\_profile table (remember to add plural form of table name be4 saving)

We need to create a new route to accept both get user\_profile and store user\_profile information

So, open routes/web.php

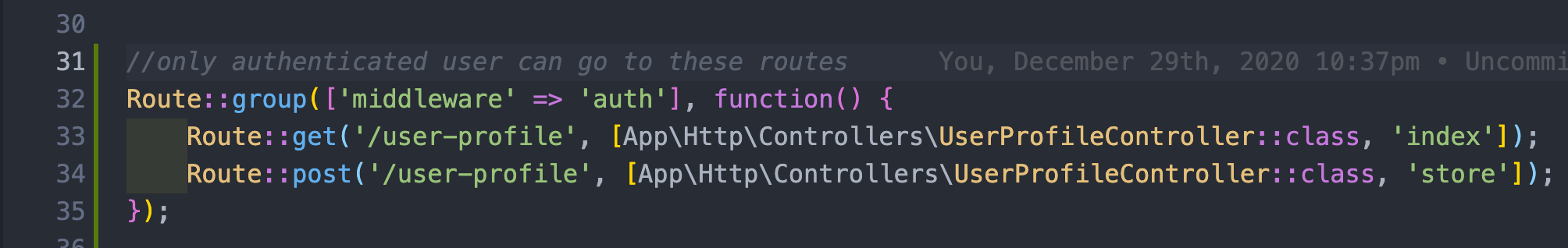


Figure 11: Create user\_profile routes

New routes are added. Line 32: Get user profile and in line 33: Update user profile. In order to access /user-profile to work, you need to login to page first

Create UserProfile model

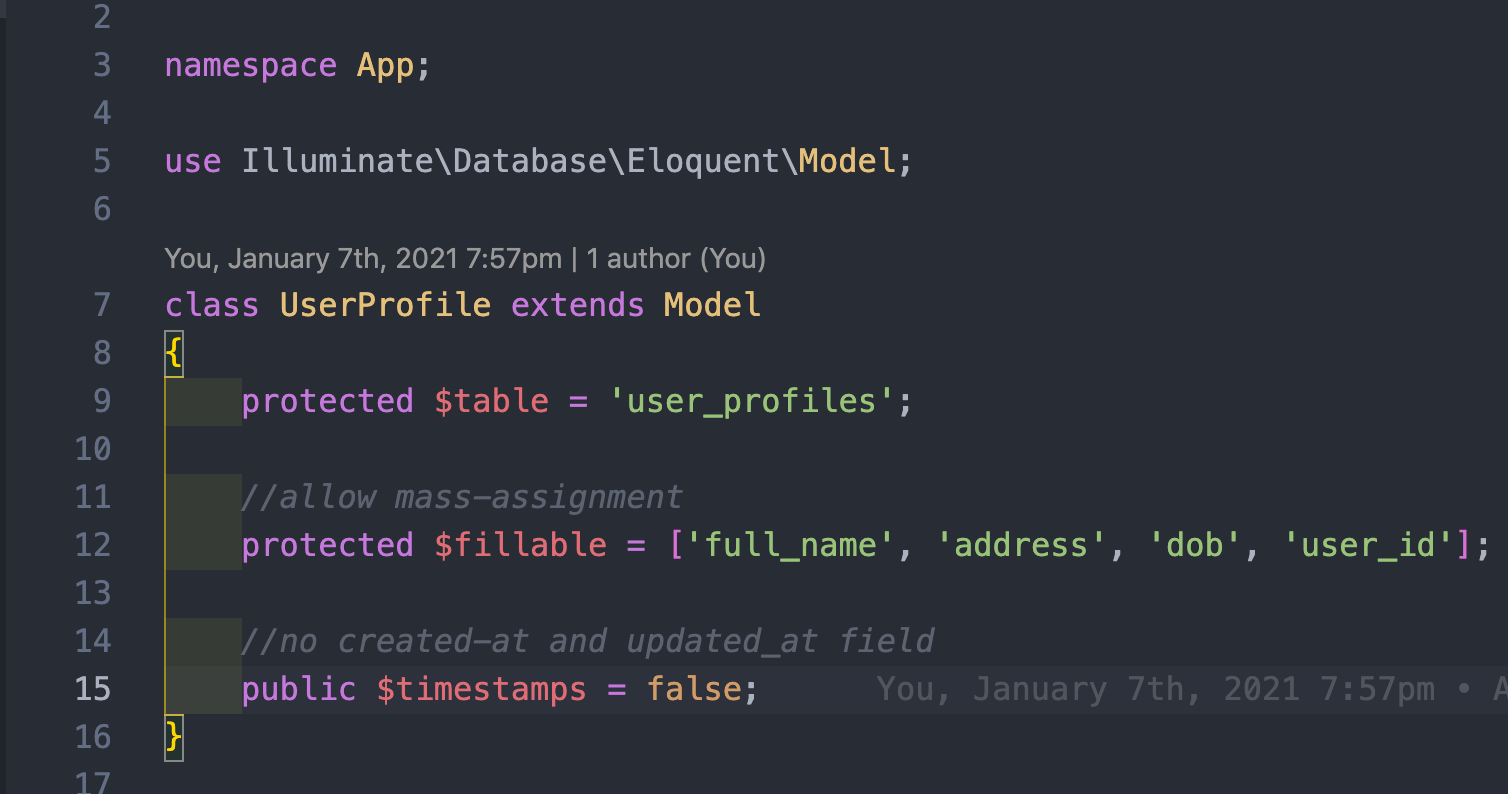


Figure 12: UserProfile model

Add a link to menu as follow:

<a class="block px-4 py-2 text-sm leading-5 text-gray-700 hover:bg-gray-100 focus:outline-none focus:bg-gray-100 transition duration-150 ease-in-out" href="/user-profile">User profile</a>

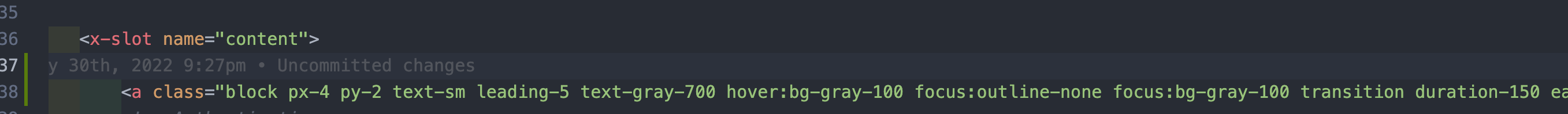


Figure 13: Edit profile menu

At the controller, add these lines, detail is noted in screen



Figure 14: UserProfileController@index for viewing profile

What serialize() method means is Get string representation of specific key of object instance



Figure 15: UserProfileController@store for updating profile

User profile index page is here

<https://gitlab.com/laravel100/tutorial-5/-/blob/master/resources/views/user-profile/index.blade.php>

Evidence? I use Laravel debug-bar to track query execution. If debugbar is not installed, install it by running

composer require barryvdh/laravel-debugbar --dev

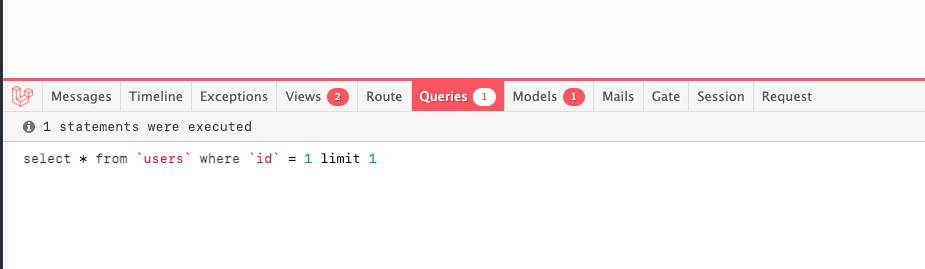


Figure 16: No user-profile query load

No database for user-profile hit. Thus, it’s faster.

The nature of this solution is

* User created account successfully in the system
* After created successfully, he goes to profile page and fill in basic information such as name, age.
* We will cache this information so that when user view this information again, take it from Redis instead of getting from database. If Redis key is not exist, we redirect user to a new view, where user enters information. After user fills information, we cache data using **String** datatype by serializing data.
* When user view information, take from Redis by unserializing key in Redis.
* When user changes information, update that hash key inside Redis, using Redis hmset command again

Now we check in Redis

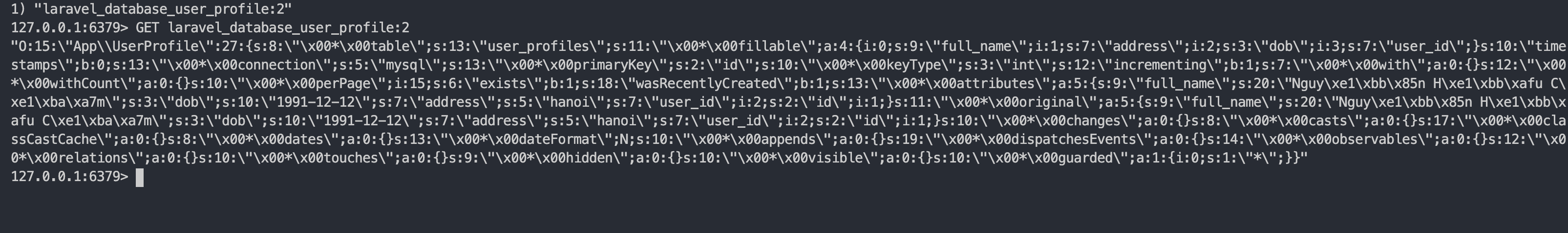


Figure 17: Redis check for user\_profile to store in String data type.

Reference for checking Predis methods: <https://github.com/phpredis/phpredis>

## EXCERCISE

Use this project

1. Write migration and full CRUD to create product table. Product table includes ID, name, description (to be simple).

After login successfully. Navigate to a new page where you can see a list of products. These products come inside database. You need to create a new migration for products table, or create table directly in database management tools and use Product model to map to this table.

1. Caching

Implement caching on each product itself when showing product information, assume that products don’t often change over time. Cache for 15 mins.

Hint: Use **STRING** datatype, serialize object by calling serialize($objects). After that, check debug bar whether products are loaded or not.

Read more at: <https://redis.io/topics/data-types#:~:text=Redis%20Lists%20are%20simply%20lists,new%20element%20on%20the%20tail>.

1. Your task is to implement function and create a page (a full set of routes + migrations + controller + view + model is needed) to view recent products that you have recently viewed (Sản phẩm gần đây)

Product data is in the first exercise. Recent product is calculated when you click on “View” of specific product in the interface. Recent products are alive in 30 minutes.

Sample interface is given below

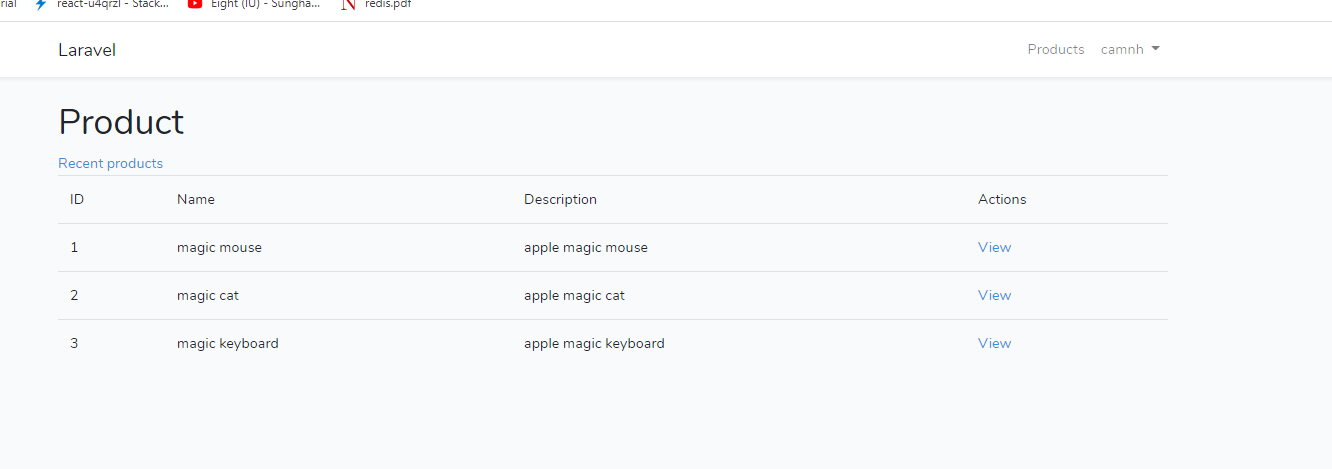


Figure 18: Sample UI to list products

When view recent products

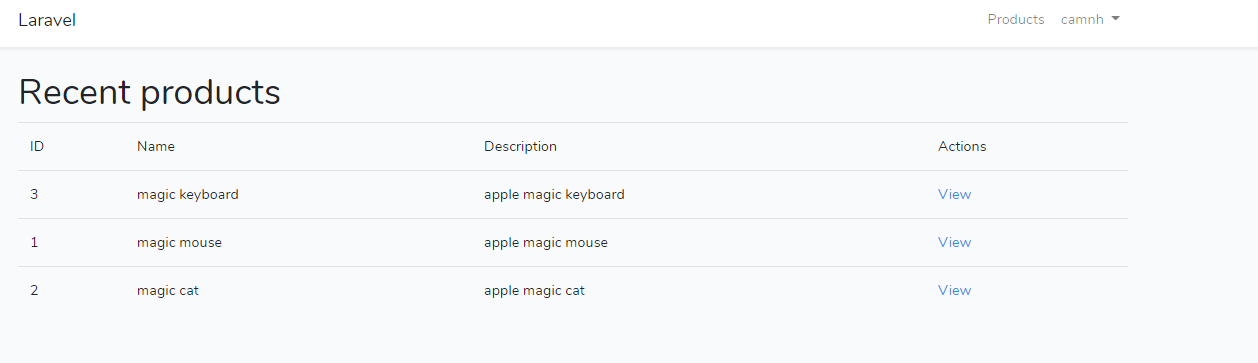


Figure 19: Sample recent products page

Hint: Using Redis sorted set (ZSET), store score along with object, score is timestamp that you clicked on product, retrieved by using Carbon::now()->timestamp (<https://stackoverflow.com/questions/32719972/how-to-get-current-timestamp-from-carbon-in-laravel-5>), key of product is identified by format for example users:1:recent\_products, value of ZSET key is the product\_id itself. Each product is stored in Redis using string datatype

For example:

* SET products:1 (serialized\_product\_info).
* ZADD users:1:recent\_products 1552296328 product:1