1.Caching Introduction

Cahe is layer between Controller and Storage

Memcached and Redis most important

It will make query for data much much faster



We see how it works in diagram at first we need to fetch data we can see whether its found in Cache or not if not than fetch from database

If its found in cache then we should return data

For that reasons we need to know slowest part of your application

2.Laravel-Debugger

For installing Laravel debugger installation **composer require barryvdh/laravel-debugbar --dev**

**In postsController we will use**

**'posts' => BlogPost::latest()->withCount('comment')->with('user')->get();**

**instead of**

'posts' => BlogPost::latest()->withCount('comment')->get();

The queries became 5 from 17

Then in delete.blade.php

**@auth**

**@if(!$post->trashed())**

**@can('delete',$post)**

**<form method="POST" action="{{ route('posts.destroy', [**

**'post'=>$post->id])}}">**

**@csrf**

**@method('DELETE')**

**<input type="submit" value="DELETE" class="btn btn-danger ">**

**</form>**

**@endcan**

**@endauth**

**@endif**

**In index.blade.php**

**{{-- //Code for explanation --}}**

**{{-- @if($post->trashed())**

**<a class="d-none" href="{{ route('posts.edit', [**

**'post'=>$post->id])}}">Edit</a>**

**@else**

**<a class="d-block" href="{{ route('posts.edit', [**

**'post'=>$post->id])}}">Edit</a>**

**@endcan**

**@endif**

**@endauth --}}**

**{{-- @cannot('delete',$post)**

**<p class="text-warning">You cannot delete this post</p>**

**@endcannot --}}**

**{{-- @endif --}}**

**{{-- //Refactored code --}}**

**@if(!$post->trashed())**

**<a class="" href="{{ route('posts.edit', [**

**'post'=>$post->id])}}">Edit</a>**

**>**

**@endcan**

**@endif**

**@endauth**

**3.Storing data in cache**

**At first we have to go to Cache.php**

**In PostsController.php**

**public function index(){**

**$most\_commented=Cache::remember('most\_commented', now()->addSeconds(30), function () {**

**return BlogPost::mostCommented()->take(5)->get();**

**});**

**$mostActive=Cache::remember('mostActive', now()->addSeconds(30), function () {**

**return User::WithMostBlogPosts()->take(5)->get();**

**});**

**$mostActiveLastMonth=Cache::remember('mostActiveLastMonth', now()->addSeconds(30), function () {**

**return User::withMostBlogPostsLastMonth()->take(5)->get();**

**});**

**return view(**

**'posts.index',**

**[**

**'posts' => BlogPost::latest()->withCount('comment')->with('user')->get(),**

**// 'most\_commented' => BlogPost::mostCommented()->take(5)->get(),**

**'most\_commented' =>$most\_commented,**

**'mostActive' =>$mostActive,**

**'mostActiveLastMonth' =>$mostActiveLastMonth**

**]**

**);**

**}**

**Removing From Cache**

**In Posts.Controller**

**public function show($id)**

**{**

**$blogPost=Cache::remember("blog-post-{$id}", 60, function () use($id) {**

**return BlogPost::with('comment')->findorFail($id);**

**}); return view('posts.show',[**

**'posts'=>$blogPost**

**]);**

**In BlogPost.php**

**static::updating(function (BlogPost $blogPost) {**

**Cache::forget("blog-post-{$blogPost->id}");**

**});**

**4.Cache Facade**

**If you run Cache::put it will run forever few artisan tinker functions here**

**5.Practical using cache as Storage**

We reduced queries earlier here we will reduce sql queries

Now we need to need to get list of users session id and last visited time

We will initialize the counter

Remove user from the list if its expired

Decrease counter

If user is on teh list we will increase the counter by 1



**6.Practical using Cache for Storage implementation**

In PostsController.php

$sessionId=session()->getId();

$counterKey="blog-post-{$id}-counter";

$usersKey="blog-post-{$id}-users";

$users=Cache::get($usersKey,[]);

$usersUpdate=[];

$difference=0;

$now=now();

foreach($users as $session=>$lastVisit){

if($now->diffInMinutes($lastVisit)>=1){

$difference--;

}else{

$usersUpdate[$session]=$lastVisit;

}

}

// if(!array\_key\_exists($sessionId,$users)

// ||$now->diffInMinutes($users[$sessionId])>=1)

if(!array\_key\_exists($sessionId, $users)

|| $now->diffInMinutes($users[$sessionId]) >= 1)

{

$difference++;

}

$usersUpdate[$sessionId]=$now;

Cache::forever($usersKey, $usersUpdate);

if(!Cache::has($counterKey)){

Cache::forever($counterKey,1);

}else{

Cache::increment($counterKey, $difference);

}

$counter=Cache::get($counterKey, $difference);

7.Using and Setting up Redis as cache storage

Redis and memcached are the fastest for data storage

Redis Cache create an account in .env file

REDIS\_HOST=redis-10680.c114.us-east-1-4.ec2.cloud.redislabs.com:10680

REDIS\_PASSWORD=19eqCY9CIzz9UFeJ4bU6fwZZ5s6qNnxh

REDIS\_PORT=6379

REDIS\_CACHE\_DB=0

Redis free version is not really fast

8.Cache tags

**9.Practical using cache tags**

**Cache::tags(['people','artists'])->put('Tommy','Hello',30);**