به نام خدا

پروژه ی پایانی درس مبانی بازیابی

سید محمد موسوی

شماره دانشجویی : 9613200074

مقدمه

هدف این پروژه طراحی یک موتور جستجو و تحلیل فاز های indexing و searching میباشد.

این پروژه به صورت spa(single page application) طراحی شده است.

این پروژه از سه قسمت اصلی تشکیل شده است :

1. کرال سایت
2. رسم نمودار heaps و zipfs
3. searching و indexing

جهت درک بهتر پروژه و این مقاله ، شما میبایست به زبان های python ، php(laravel) ، js(vue) و همچنین زبان های نشانه گذاری همانند html برای طراحی قالب و css برای استایل دهی مسلط باشید.

قسمت های اول و دوم پروژه با زبان پایتون و قسمت سوم آن با لاراول میکس که از فریمورک های php میباشد ، زده شده.

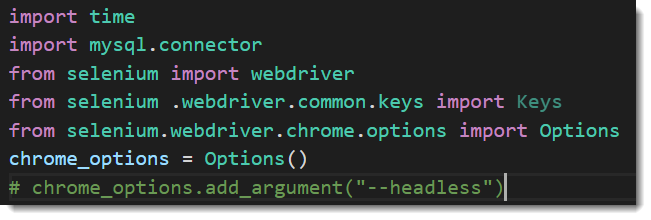
کرال سایتcrawling)):

این قسمت از پروژه با استفاده از کتابخانه ی سلنیوم در پایتون زده شده است.

جهت استفاده از این کتابخانه ابتدا باید دستور pip install selenium را در ترمینال وارد کرده تا کتابخانه نصب شود.

ما در این پروژه از mysql استفاده میکنیم که برای استفاده از آن در پایتون باید کتابخانه ی مورد نظر را با دستور pip install mysql.connector نصب کنیم.

پس از نصب کتابخانه های مورد نظر ، حال نوبت به import آنها میرسد.



همان طوری که در تصویر بالا مشخص است ، علاوه بر کتابخانه های ذکر شده ما از یک کتابخانه به اسم time که به طور پیش فرض در پایتون وجود دارد برای ایجاد وقفه یا همان delay استفاده میکنیم.

لازم به ذکر است در صورت عدم تمایل به تماشای مرورگر میتوانید حالت headless را که در تصویر کامنت شده است ، فعال کنید.

برای استفاده از کتابخانه ی سلنیوم شما نیاز به دانلود chromdriver دارید که متناسب با ورژن کروم خود باید آن را دانلود نمایید.

پس از دانلود ، ما از آن استفاده میکنیم.

C:\Users\Asus\AppData\Local\Temp\SNAGHTML205d8a44.PNG

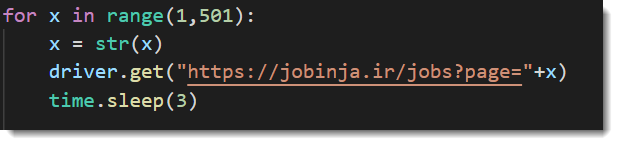
ما در این پروژه 10 هزار دیتا از سایت jobinja را کرال کردیم.

از آنجایی که در هر صفحه از این سایت 20 دیتا وجود دارد ، ما نیاز به کرال 500 صفحه داریم.

آدرس اصلی سایت، <https://jobinja.ir> میباشد که با توجه به صفحه بندی های موجود در سایت و آگهی های موجود به شکل <https://jobinja.ir/jobs?page=x> در می آید که در آن ، x شماره ی صفحه میباشد به عنوان مثال برای صفحه ی پنجم ، لینک به صورت زیر است :

[https://jobinja.ir/jobs?page=5](https://jobinja.ir/jobs?page=x)

ما با استفاده از حلقه ی for ، به تک تک آن 500 صفحه رفته و شروع به کرال آنها میکنیم.



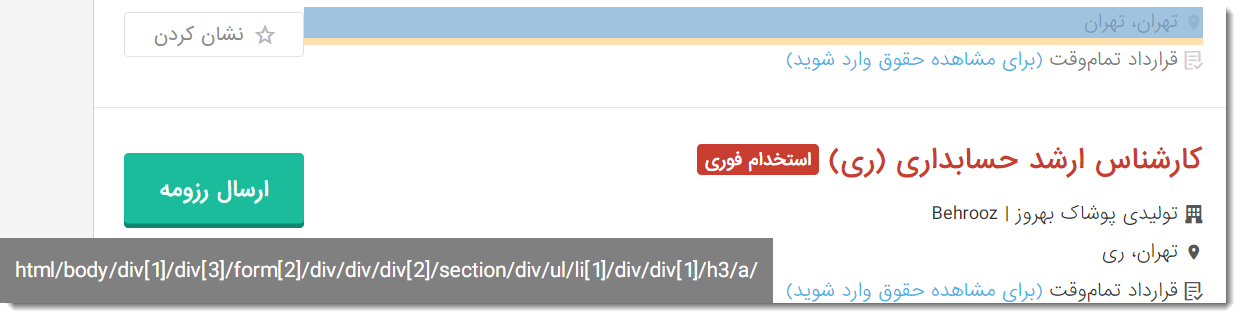
در سلنیوم برای انتخاب(select) آیتم مورد نظر راه های زیادی از جمله class , tag , xpath و ... وجود دارد که ما در این پروژه از xpath استفاده میکنیم.

برای به دست آوردن xpath ، روش های متنوعی وجود دارد که ما در ادامه دو مورد از این روش ها را معرفی میکنیم

روش اول : استفاده از chrome extensions

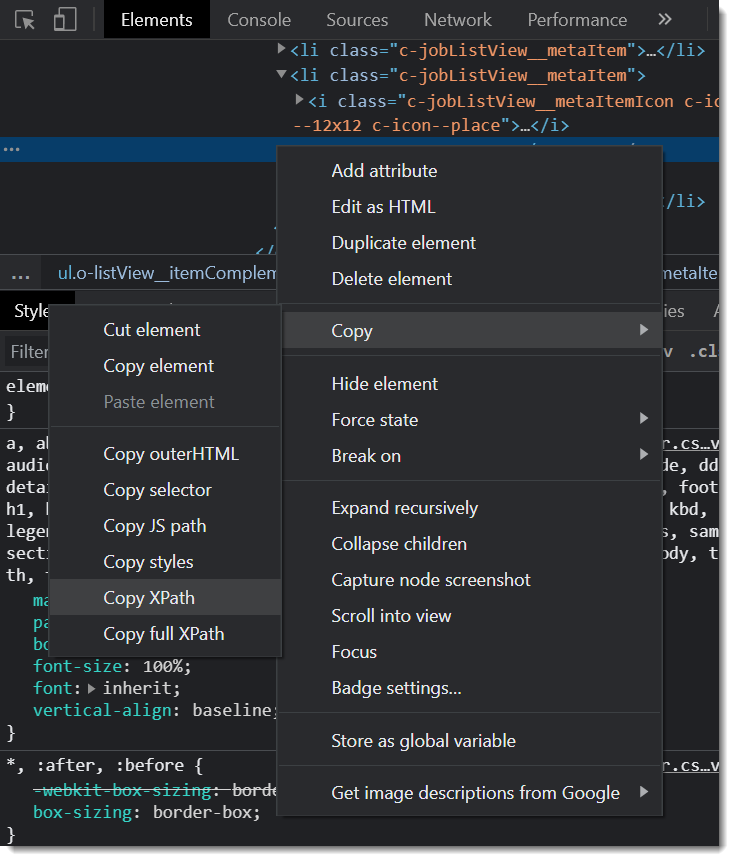
در این روش extension به نام xpath finder را دانلود کرده و از آن استفاده میکنیم.

نحوه ی استفاده به این شکل است که پس از فعال کردن extension ، روی آیتم مورد نظر کلیک کرده و xpath آن را به دست می آوریم.



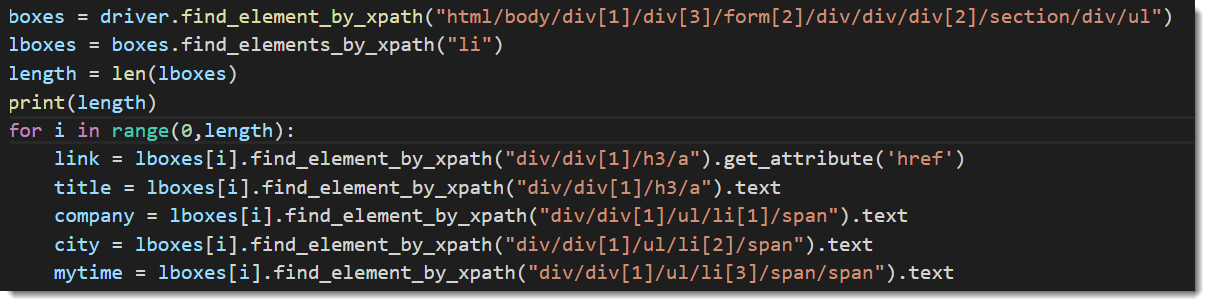
روش دوم : استفاده از inspect

در این روش روی سایت کلیک راست کرده و inspect میگیریم و در قسمت بالا و سمت چپ پنجره ی باز شده را کلیک کرده و پس از آن المنت مورد نظر را انتخاب کرده و xpath آن را کپی میکنیم.



همان طور که در html میدانیم ، هر ul در سایت ، چندین li به عنوان فرزند خود دارد که ما نیاز داریم که عنوان ، شهر ، شرکت ، زمان آگهی و لینک هرکدوم از آنها را به دست آوریم.

پس بار دیگر نیاز به حلقه ی for داریم.



همان طور که در تصویر مشخص است ، ما ul خود را با استفاده از xpath ای که دارد انتخاب کرده و در متغیر boxes میریزیم.

حال با استفاده از آن li های خود را سلکت کرده و در متغیر lboxes میریزیم و با استفاده از یک حلقه ، که نقطه ی شروع آن 0 ، و نقطه ی پایان آن طول آرایه ی lboxes است ، عنوان ، شهر ، شرکت ، زمان آگهی و لینک هرکدوم از آنها را به دست آوریم.

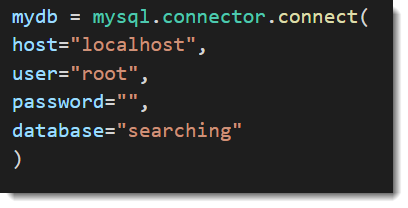
نکته ی قابل توجه این است که در پایان آنها باید .text بگذارید تا متن تگ را به شما نشان دهد و اگر تگ شما یک لینک یا همان تگ a است و دنبال لینک آن یا همان href آند هستید ، باید از متد .get\_attribute(‘href’) استفاده کنید.

پس از آن نوبت به ذخیره ی داده ها در mysql است.

به منظور این کار ابتدا باید یکی از نرم افزار های wampserver و یا xamp را داشته و یا دانلود کنید.

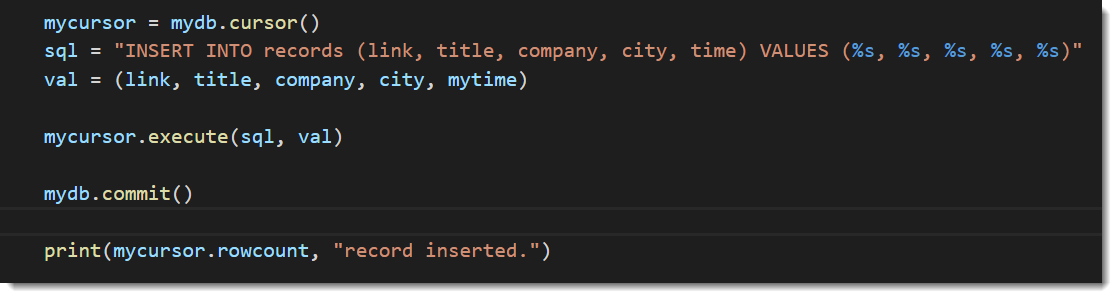
پس از دانلود و نصب ، شما باید یک دیتابیس ساخته و ستون های مورد نظر را به آن اضافه کنید.

حال نوبت به اتصال به دیتابیس میرسد.



مقادیر را با توجه به دیتابیس خود ، کامل کنید.

حال نوبت به insert داده ها میرسد که به این شکل آن را انجام میدهیم :



دقت کنید که در پایان کد حتما driver.quit() را قرار داده که منابع شمارا بیش از اندازه مصرف نکند

خب ، به پایان بخش کرال رسیدیم. در بخش بعدی الگوریتم های zipf و heaps را توضیح خواهیم داد.

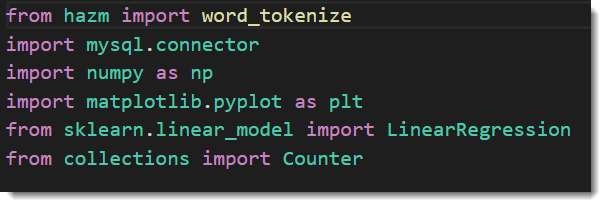
Heaps and zipfs law:

قانون heap به ما میگوید در عمل کرالینگ هرچه در محور تعداد توکن ها بیشتر جلو برویم ، سایز vocabulary یا در واقع همان تعداد term های داخل دیکشنری با شیب کمتری افزایش پیدا میکند. مثالش هم میتوان مردم و کشور هایشان را در نظر گرفت.

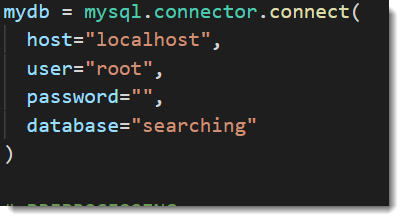
قانون zipf هم تعداد تکرار کلمات را درنظر میگیرد و طبق آن نمودار را رسم میکند.

خب بریم سراغ کدها .

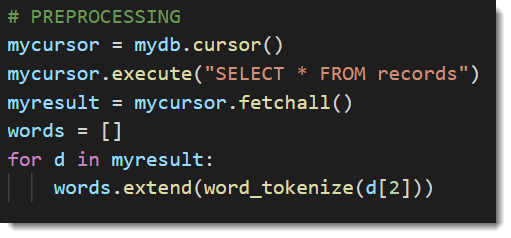
در خطوط ابتدایی کتابخانه ها ایمپورت میشوند که همانطور که گفته شد اگر آنهارا ندارید با دستور pip install میبایست آنهارا نصب کنید.



حال نوبت به اتصال به دیتابیس و mysql میرسدد که کدها مشابه کدهایی است که قبلا داشتیم.

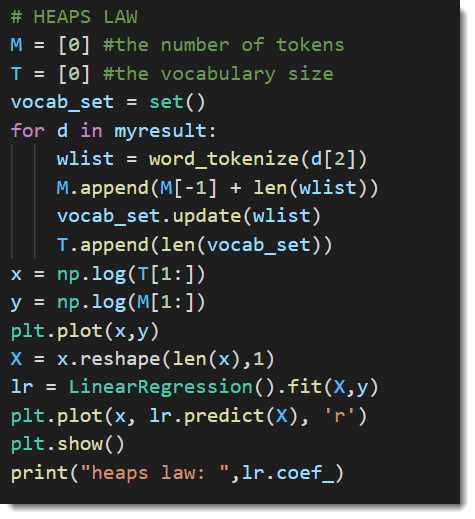


اکنون نوبت خواندن اطلاعات از دیتابیس است که با کوئری select انجام میدهیم. سپس آرایه ای با نام words ساخته و با استفاده از یک حلقه تمامی عنوان های توکنایز شده را داخل آرایه میریزیم.

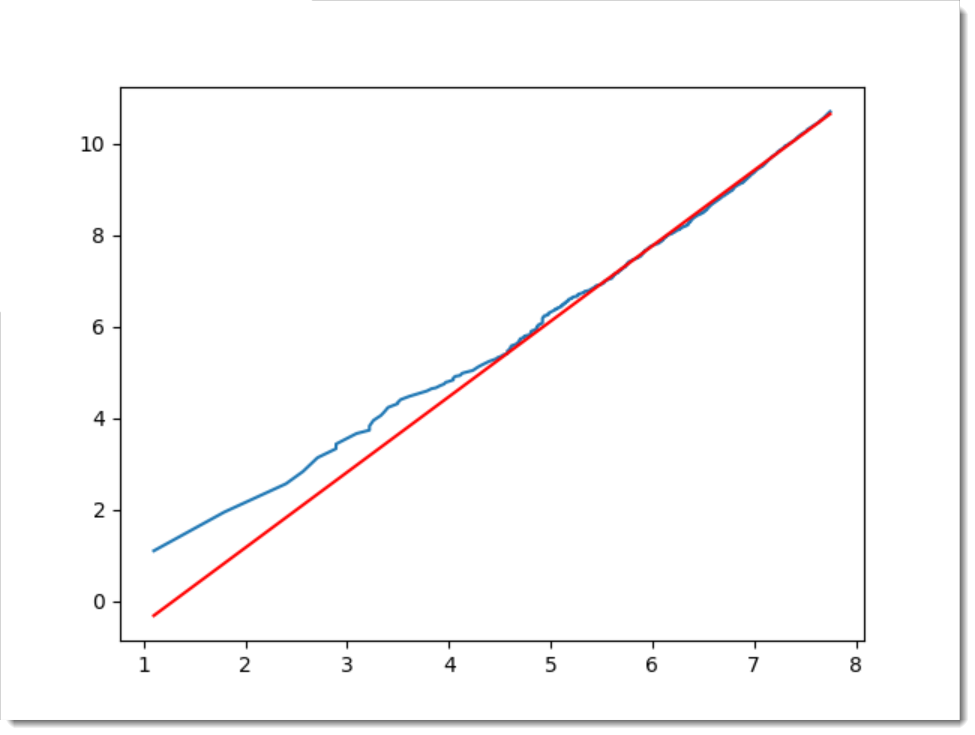


ما نیاز به دو آرایه داریم که یکی تعداد توکن ها و دیگری سایز vocabulary است.

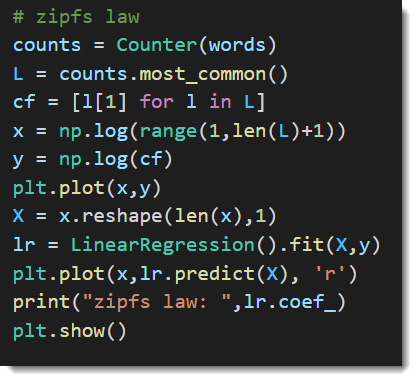
سپس با حلقه این دو را پر میکنیم و با استفاده از کتابخانه pyplot نمودار آنها و همچنین با استفاده از کتابخانه sklearn خط پیش بینی شده برای آنها را رسم میکنیم و سپس coef\_که همان شیب نمودار است را چاپ میکنیم.



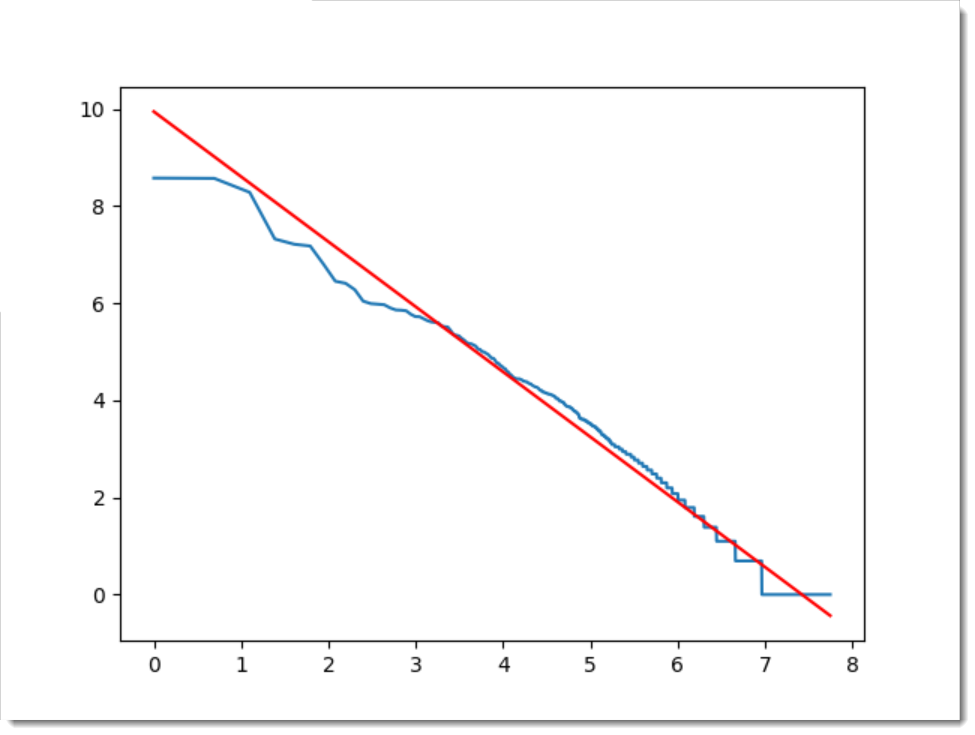
**خروجی کد شکل زیر خواهد بود:**



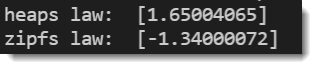
برای قانون zipf نیز با استفاده از کتابخانه count تعداد تکرار کلمات را حساب کرده و نمودار های آن را همانند قانون heap میکشیم



خروجی کد ، شکل زیر خواهد بود.



و شیب خط آنها :



سرچ و ایندکس :

برای این قسمت از فریمورک لاراول استفاده شده است.  
این فریمورک برای سرچ به طور پیش فرض از کتابخانه algolia استفاده میکند ولی از آنجایی که ایران تحریم است ما از elasticsearch استفاده کردیم.

برای اینکه در لاراول کار کند ما از یکی از لینک های گیت هاب کمک گرفته و مرحله به مرحله پیش میرویم.

در ابتدا باید laravel scout را در لاراول نصب کنیم.

composer require laravel/scout

و پس از آن الستیک سرچ را با استفاده از داکر نصب میکنیم.

سپس نوبت به نصب پکیج گیت های میرسه.

composer require babenkoivan/scout-elasticsearch-driver

اگر ورژن لاراول شما از 5.4 کمتر باشد ، این دو خط را باید در مسیر config/app.php اضافه کنید.

'providers' => [

Laravel\Scout\ScoutServiceProvider::class,

ScoutElastic\ScoutElasticServiceProvider::class,

]

برای پیکربندی تنظیمات نیز دو دستور زیر را مینویسیم.

php artisan vendor:publish --provider="Laravel\Scout\ScoutServiceProvider"

php artisan vendor:publish --provider="ScoutElastic\ScoutElasticServiceProvider"

حال نوبت به پیکربندی index میرسد.

php artisan make:index-configurator MyIndexConfigurator

پس از زدن دستور بالا یک فایل با نام MyIndexConfigurator ساخته می شود که درون آن کدهای زیر را در آن وارد میکنیم.

*<?php  
  
namespace* App;  
  
*use* ScoutElastic\IndexConfigurator;  
*use* ScoutElastic\Migratable;  
  
*class* MyIndexConfigurator *extends* IndexConfigurator  
{  
 *use* Migratable;  
 *// It's not obligatory to determine name. By default it'll be a snaked class name without `IndexConfigurator` part.  
 protected* $name = 'my\_index';  
  
 *// You can specify any settings you want, for example, analyzers.  
 protected* $settings = [  
 "analysis" => [  
 "char\_filter" => [  
 "zero\_width\_spaces" => [  
 "type" => "mapping",  
 "mappings" => ["\\u200C=>\\u0020"]  
 ]  
 ],  
 "filter" => [  
 "persian\_stop" => [  
 "type" => "stop",  
 "stopwords" => "\_persian\_"  
 ]  
 ],  
 "analyzer" => [  
 "rebuilt\_persian" => [  
 "tokenizer" => "standard",  
 "char\_filter" => ["zero\_width\_spaces"],  
 "filter" => [  
 "lowercase",  
 "decimal\_digit",  
 "arabic\_normalization",  
 "persian\_normalization",  
 "persian\_stop"  
 ]  
 ]  
 ]  
 ]  
 ];  
}

در ابتدا یک اسم دلخواه برایش میگذاریم و سپس در قسمت settings آن prsian analyzer که در سایت الستیک سرچ برای زبان فارسی وجود دارد را قرار میدهیم.

حال نوبت به آن رسیده که یک مدل searchable ایجاد کنیم.

php artisan make:searchable-model MyModel --index-configurator=MyIndexConfigurator

پس از زدن دستور بالا ما یک مدل searchable با نام Record در پوشه ی app داریم که محتویات آن به شکل زیر است :

*<?php  
  
namespace* App;  
  
*use* Illuminate\Database\Eloquent\Model;  
*use* ScoutElastic\Searchable;  
  
*class* Record *extends* Model  
{  
 *use* Searchable;  
 *protected* $table="records";  
 *public* $timestamps=*false*;  
  
 */\*\*  
 \** ***@var*** *string  
 \*/  
 protected* $indexConfigurator = MyIndexConfigurator::*class*;  
  
 */\*\*  
 \** ***@var*** *array  
 \*/  
 protected* $searchRules = [  
 *//* ];  
  
 */\*\*  
 \** ***@var*** *array  
 \*/  
 protected* $mapping = [  
 'properties' => [  
 'suggest' => [  
 'type' => 'completion',  
 ],  
 'link' => [  
 'type' => 'keyword',  
 ],  
 'title' => [  
 'type' => 'text',  
 ],  
 'company' => [  
 'type' => 'text',  
 ],  
 'city' => [  
 'type' => 'text',  
 ],  
 'time' => [  
 'type' => 'text',  
 ],  
 ]  
 ];  
}

که در ابتدای آن نام جدول ذکر شده و در ادامه و در قسمت $mapping ستون ها و نوع آنها آمده.

پس از آن دو دستور زیر را زده تا دیتا ها به الستیک سرچ ارسال شوند:

php artisan elastic:update-mapping "App\Record"

php artisan scout:import "App\Record"

حال به سراغ کنترلر میرویم. ابتدا یک کنترل با دستور php artisan make:controller SearchController میسازیم.

در خطوط ابتدایی ، ما مدل searchable خود را فراخوانی کرده تا از آن استفاده کنیم.

در خطوط 14 و 15 ما کوئری کاربر به همراه شماره صفحه را میگیریم.

این اطلاعات از طریق vue از فرانت به سمت کنترلر فرستاده میشود.

دیتا ها از فرانت به app.js فرستاده شده و از آنجا بدون رفرش صفحه به کنترلر فرستاده میشوند.

در پوشه ی ما app.js یک سری دیتا را وارد میکنیم و پس از آن تعدادی تابع داریم که در ادامه به توضیحات آن میپردازیم.

تابع submit :

در این تابع 2 ورودی را گرفته و به کنترلر پاس میدهیم .

در خطوط ابتدایی آن کد های مربوط به paginate قرار داده شده.

برای paginate ما یک آرایه داریم که اعداد 1 تا 9 را شامل میشود.

ما میخواهیم وقتی کاربر صفحات بیشتر از 5 را کلیک میکند ، همانند گوگل این آرایه آپدیت شده و بازه ی y=x-4 تا y+10 را نشان دهد.

به عنوان مثال اگر کاربر روی صفحه ی 6 کلیک کرد ، باید بازه ی صفحات ما از 2 تا 10 باشد.و اگر کلیک کاربر کمتر و مساوی 5 بود همان 1 تا 9 را نشان دهد.

submit: *function*(*pagenumber*) {  
 *this*.mypagenumber = *pagenumber*;  
 *if* (*pagenumber*>5){  
 *this*.pagearrays = [];  
 *var* y = *pagenumber* - 4;  
 *var* z = y + 10;  
 *for*(y;y<z;y++){  
 *this*.pagearrays.push(y);  
 }  
 }  
 *else* {  
 *this*.pagearrays = [];  
 *var* y = 1;  
 *for*(y;y<10;y++){  
 *this*.pagearrays.push(y);  
 }  
 }

در قسمت بعدی برای این که کاربر متوجه شود در کدام صفحه است ، بک گراند مربوط به اون آیتم را باید تغییر دهیم.

به منظور این کار کلاس active را از تمامی آیتم ها گرفته و پس از آن فقط به آن آیتمی که کاربر روی آن کلیک کرده کلاس active را میدهیم.

*var* a = document.getElementsByClassName('options');  
*for* (*var* i = 0; i < a.length; i++) {  
 a[i].classList.remove('active')  
}  
setTimeout(*function*(){document.getElementById(*pagenumber*).classList.add('active');},1000);

پس از آن آرایه ی پیشنهادات برای کاربر یا همان sarray را خالی میکنیم. زیرا کاربر سرچ خود را انجام داده است.

در خط بعدی با استفاده از axios دو مقدار کوئری کاربر و شماره ی صفحه را به بک اند میفرستیم تا در آنجا با استفاده از این مقادیر اطلاعات مورد نیاز را از بک اند دریافت کنیم.

*this*.sarrays = [];  
axios.post('http://127.0.0.1:8000/records', {  
 mysearch: *this*.mysearch,  
 pagenumber: *pagenumber*})

پس از که این اطلاعات به کنترلر فرستاده شد ، در آنجا این اطلاعات را گرفته و به کمک مدل searchable که ساختیم داده ها را از دیتابیس دریافت میکنیم.

و قسمت های مختلف داده ها را ( یعنی title , link , city ,company , time) را به همراه تعداد رکورد برگردانده شده ($count) برمیگردانیم.

این اطلاعات در app.js و به کمک response.data قابل دریافت است.

تصویر کدها در صفحه ی بعد =>

$records = Record::*search*("$mysearch")->paginate();  
$count = Record::*search*("$mysearch")->count();  
$titles = *array*();  
$links = *array*();  
$citys = *array*();  
$companys = *array*();  
$times = *array*();  
 *foreach* ($records *as* $record){  
 *array\_push*($titles,$record['title']);  
 *array\_push*($links,$record['link']);  
 *array\_push*($citys,$record['city']);  
 *array\_push*($companys,$record['company']);  
 *array\_push*($times,$record['time']);  
 }  
 *return* [$titles , $links , $citys , $companys , $times, $count, $mysearch];

در app.js ، ما داده هایی که از کنترلر برگردانده شده را بررسی میکنیم.  
اگر طول مثلا title آنها صفر باشد یعنی کوئری کاربر ، نتیجه ی قابل قبولی نداشته است و باید این را به کاربر اعلام کنیم.

حال 2 حالت پیش می آید که در هر 2 حالت نباید کاربر کوئری خالی ارسال کرده باشد که اگر این گونه باشد ، نباید تغییری در فرانت ایجاد شود.

حالت اول اینکه کاربر قبل این سرچ ، سرچ دیگری انجام داده و نتیجه ی مثبت گرفته است. در این حالت input ما به بالای صفحه رفته است و مقدار متغیر executed برابر true می باشد که در این حالت ما هیچ انیمیشنی نداریم و فقط به کاربر میگوییم "به نظر میرسد نتیجه ی خوبی مطابق با جستجوی شما وجود ندارد" و یک متد راهنمایی که در گوگل هم نشان داده میشود.

در حالت دوم کاربر قبل از این سرچ ، سرچ دیگری انجام نداده و input ما در همان حالت اولیه است که در این حالت انیمیشن داریم و سرچ باکس ما بسته شده و ذره بین به شکل ضربدر در می آید ( این کار را با افزودن یک کلاس انجام میدهیم) ، سپس این ضربدر اندکی به بالا رفته و زیر آن متنی با عنوان "نتیجه ای یافت نشد" نشان داده میشود.

حالت اول :

*if* (*this*.mysearch != "" && *this*.executed == *true*) {  
 *this*.items = [];  
 *this*.items.push({'title': "به‌نظر می‌رسد نتیجه خوبی مطابق با جستجوی شما وجود ندارد"});  
 *this*.items[0]['company'] = "نکته: سعی کنید از کلمه‌هایی استفاده کنید که ممکن است در صفحه‌ای که جستجو می‌کنید وجود داشته باشد. مثلاً به‌جای «دستور پخت کیک» از «نحوه درست کردن کیک» استفاده کنید.";  
}

حالت دوم:

*if*(*this*.mysearch != "" && *this*.executed == *false*){  
$("#pq").fadeOut();  
$('link[*href*="/css/notfound.css"]')[0].disabled = *false*;  
setTimeout(*function*(){ document.getElementById("search").style.top = "-50px" }, 1500);  
setTimeout(*function*(){ $("#find").fadeIn(700) }, 2000);  
setTimeout(*function*(){ $("#find").fadeOut(700) }, 2500);  
setTimeout(*function*(){ document.getElementById("search").style.top = "45px" }, 3500);  
setTimeout(*function*(){ $('link[*href*="/css/notfound.css"]')[0].disabled=*true*; }, 4800);  
setTimeout(*function*(){ $("#pq").fadeIn(1000); }, 4800);  
}

حال اگر کوئری کاربر نتیجه داشت ، باز هم همانند مورد قبلی 2 حالت پیش می آید که بازهم در هر 2 حالت نباید کوئری کاربر خالی باشد و در یک حالت انیمیشن داریم(حالتی که سرچ باکس پایین و در حالت اولیه است) و در حالت دیگر انیمیشن نداریم (همان حالتی که سرچ باکس بالاست و کاربر قبل از این کوئری یک بار کوئری زده و جواب گرفته)

تصاویر در صفحه ی بعد =>

*else* {  
 *if* (*this*.mysearch != "" && *this*.executed == *false*) {  
 *this*.executed = *true*;  
 $("#pq").fadeOut();  
 $('link[*href*="/css/found.css"]')[0].disabled = *false*;  
 setTimeout(*function* () {  
 document.getElementById("search").style.bottom = "85%";  
 }, 1500);  
 setTimeout(*function* () {  
 document.getElementById("input").style.bottom = "85%"  
 }, 1700);  
 setTimeout(*function* () {  
 $('link[*href*="/css/found.css"]')[0].disabled = *true*;  
 }, 2700);  
 setTimeout(*function* () {  
 $("#results").fadeIn(1000);  
 }, 3000);  
 *for* (*var* i = 0; i < length; i++) {  
 *this*.items.push({'title': *response*.data[0][i]});  
 *this*.items[i]['link'] = *response*.data[1][i];  
 *this*.items[i]['city'] = *response*.data[2][i];  
 *this*.items[i]['company'] = *response*.data[3][i];  
 *this*.items[i]['time'] = *response*.data[4][i];  
 }  
 }  
 *if* (*this*.mysearch != "" && *this*.executed == *true*) {  
 *this*.items = [];  
 *for* (*var* i = 0; i < length; i++) {  
 *this*.items.push({'title': *response*.data[0][i]});  
 *this*.items[i]['link'] = *response*.data[1][i];  
 *this*.items[i]['city'] = *response*.data[2][i];  
 *this*.items[i]['company'] = *response*.data[3][i];  
 *this*.items[i]['time'] = *response*.data[4][i];  
 }  
 }  
}

خب 2 تابع دیگر در app.js داریم که در ادامه به آنها میپردازیم .

ادامه ی مطلب را در صفحه ی بعد دنبال کنید =>

تابع suggest در واقع عمل پیشنهاد به کاربر را انجام میدهد.

به عنوان مثال وقتی کاربر حرف "ب" را تایپ میکند ، کلماتی که این حرف را دارند را به طور لحظه ای به کاربر نشان میدهد.

برای این کار ما در کد های فرانت خود باید @keyup را قرار دهیم.

<input *type*="text" *placeholder*="عبارت را جستجو کنید" *id*="input" *v-model*="mysearch" *autocomplete*="off" *@keyup*="suggest()">

سپس به کمک axios و متد post به کنترلر ارسال میکنیم.

axios.post('http://127.0.0.1:8000/suggest', {  
 suggest: document.getElementById("input").value  
})

در کنترلر اطلاعات را دریافت کرده و پردازش را انجام میدهیم.

این کار را به کمک متد prefix الستیک سرچ انجام میدهیم . و در اینجا ما چون بیشتر از 4 تا پیشنهاد نمیخوایم به کاربر نشون بدیم، 4 پیشنهاد را در آرایه ریخته و پس آن حلقه را break میکنیم و آرایه را return میکنیم.

*public function* suggest(Request *$request*)  
{  
 $suggest\_array = *array*();  
 $suggest = *$request*->suggest;  
 $data = [  
 'body' => [  
 'query' => [  
 "prefix" => [ "title" => "$suggest" ],  
 ]  
 ]  
 ];  
 $client = ClientBuilder::*create*()->build();  
 $response = $client->search($data);  
 $x = 0;  
 $response\_hits = $response["hits"]["hits"];  
 *foreach* ($response\_hits *as* $response\_hit){  
 *array\_push*($suggest\_array,$response\_hit["\_source"]["title"]);  
 *if*($x == 4){  
 *break*;  
 }  
 $x++;  
 }  
 *return* $suggest\_array;  
}

پس از return کردن ، در app.js ، response را دریافت میکنیم.

ما در این پروژه برای حالت دوم(یعنی زمانی که کاربر قبلا سرچ موفق داشته و سرچ باکس در بالای صفحه قرار دارد) به دلیل حفظ ظاهر پروژه و زیبایی کار ، suggest قرار ندادیم و تابع sarray که پیش تر بیان شد که برای پیشنهاد است را خالی میکنیم.

اما در حالت اولیه suggest داریم ) همان else در کد زیر)

.then(*response* => {  
 *var* input = document.getElementById("input").value;  
 *if* (input == "" || *this*.executed == *true*){  
 *this*.sarrays = []  
 }  
 *else* {  
 *this*.sarrays = *response*.data;  
 }  
})

و تابع آخر برای زمانی است که کاربر روی suggest کلیک میکند که در این حالت باید مقدار input تغییر کرده و آرایه ی پیشنهادات ما (sarray) خالی شود.

close:*function* (*text*) {  
 $('#input').val(*text*);  
 *this*.sarrays = [];  
 *this*.mysearch = *text*}

به پایان کار رسیدیم .  
امیدوارم لذت برده باشید  
در پناه حق.