


সূচিপত্র

ভূমিকা	0
ইন্সটলেশন	1
বেসিক রাউটিং	2
ভিউ	3
ব্রড টেমপ্লেট	4
কন্ট্রোলার	5
মাইগ্রেশন	6
মডেল	7
মধ্যবর্তী প্রোজেক্ট - ১	8
পরিবেশ তৈরি	8.1
লারাভেল ইন্সটল ও প্রোজেক্ট শুরু	8.2
সিড ও ইনডেক্স পেইজ	8.3
লিঙ্ক ও সিঙ্গেল পেইজ	8.4
মিডলওয়্যার	9
মধ্যবর্তী প্রোজেক্ট - ২	10
এডভান্স রাউটিং	11
বেটার লারাভেল ইনভায়রনমেন্ট সেটআপ	12
ইনভায়রনমেন্ট ডিটেকশন	12.1
কনফিগারেশন ম্যানেজমেন্ট ফর মাল্টিপল ইনভায়রনমেন্ট	12.2
স্ট্রাকচারিং প্রজেক্ট কোডস	12.3
সেটিং আপ ম্যানড্রীল ফর ইমেইল ট্রান্সপোর্ট	12.4
সেটিং আপ কনফাইড ফর অথেনটিকেশন	12.5
সেটিং আপ কিউ উইথ beanstalkd and supervisord	12.6
প্যাকেজ ডেভেলপমেন্ট	13
ইভেন্ট ব্রডকাস্টিং	14
চলবে	15

লারাভেল পিএইচপি ফ্রেমওয়ার্ক

 Like  Share 10K people like this. [Sign Up](#) to see what your friends like.

স্বয়ংক্রিয় কন্ট্রিবিউটরের তালিকা
(প্রথম ৫ জন)

[\[050\] Robert Biswas](#)
[\[040\] Sohel Amin](#)
[\[037\] Porimol Chandro](#)
[\[017\] Nuhil Mehdy](#)
[\[008\] Prasenjit Kumar Nag](#)

ভূমিকা

লারাভেল একটি ওপেন সোর্স পিএইচপি ফ্রেমওয়ার্ক। ২০১১ সালে [টেইলর অটওয়েল](#) প্রথম লারাভেল ডেভেলপ করেন। এটি এখন শুধু টেইলর-এর প্রডাক্ট নয়, এটি এখন এক বিশাল প্রোগ্রামার কমিউনিটির প্রডাক্ট। সাধারণ ব্লগ, কনটেন্ট ম্যানেজমেন্ট সিস্টেম, ইকমার্স সাইট, বড় ধরনের বিজনেস এপ্লিকেশন, সামাজিক ওয়েবসাইট কিংবা মোবাইল এপ্লিকেশনের জন্য JSON নির্ভর এপ্লিকেশন সহ সব কিছুই লারাভেল ব্যবহার করে সার্ভারের সাথে ডেভেলপ করা সম্ভব।

লারাভেল একটি পূর্ণাঙ্গ ফ্রেমওয়ার্ক। এর গঠন বা syntax খুবই সহজ ও পরিষ্কার। ক্লাস ও ফাংশন গুলো সুন্দর সাজানো গোছানো। কখনও আপনি ডকুমেন্টেশন দেখে অনুমান করে নিতে পারবেন পরবর্তী ফাংশন কী হবে। নিজের সক্রিয়তা বজায় রেখে নতুন সূত্র বা পদ্ধতি সংযোগ করার ব্যবস্থাতো থাকছেই। লারাভেল আপনাকে অনেক স্বাধীনতা দিচ্ছে। লারাভেলের ভার্সন আপগ্রেড বা আপনার লেখা কোডের ভার্সন আলাদা ভাবে রাখতে পারছেন।

ওপেন সোর্স

এই বইটি মূলত স্বেচ্ছাশ্রমে লেখা এবং বইটি সম্পূর্ণ ওপেন সোর্স। এখানে তাই আপনিও অবদান রাখতে পারেন লেখক হিসেবে। আপনার কন্ট্রিবিউশন গৃহীত হলে অবদানকারীদের তালিকায় আপনার নাম স্বয়ংক্রিয়ভাবে যুক্ত হয়ে যাবে।

এটি মূলত একটি [গিটহাব রিপোজিটরি](#) যেখানে এই বইয়ের আর্টিকেল গুলো মার্কডাউন ফরম্যাটে লেখা হচ্ছে। রিপোজিটরিটি ফর্ক করে পুল রিকুয়েস্ট পাঠানোর মাধ্যমে আপনারাও অবদান রাখতে পারেন। বিস্তারিত দেখতে পারেন এই ভিডিওতে [Video](#)

বর্তমানে বইটির কন্টেন্ট বিভিন্ন কন্ট্রিবিউটর এবং নানা রকম সোর্স থেকে সংগৃহীত এবং সংকলিত।

 Like 74  Share

gitter [join chat](#)



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

ইন্টলেশনঃ

লারাভেল ফ্রেমওয়ার্ককে ইন্টল করতে হলে **LAMP/LEMP** এনভায়রনমেন্ট আগে থেকেই প্রস্তুত থাকতে হবে। অর্থাৎ **PHP, MySQL, Apache/nginx** ইন্টল করা থাকতে হবে আর এগুলো আমরা উইন্ডোজ মেশিন হলে **wamp/xampp** ব্যবহার করে করতে পারি অপরদিকে ইউনিক্স মেশিন হলে **LAMP/LEMP** স্ট্যাক সেটআপ করে করতে পারি।

আরেকটি অপরিহার্য বিষয় হল যে লারাভেল ইন্টল আর এর ডিপেন্ডেন্সি ম্যানেজ করার জন্য **Composer** অবশ্যই ইন্টল করা থাকতে হবে। আপনার মেশিনে যদি **Composer** ইন্টল না থাকে তাহলে নিচের পদ্ধতি অনুসরণ করতে পারেন।

উইন্ডোজ মেশিন হলে [এই লিংক](#) থেকে কম্পোজার ইন্টলারটি নামিয়ে নিয়ে খুব সহজেই ইন্টল করে নিতে পারেন।

আবার লিনাক্স কিংবা ইউনিক্স মেশিন হলে টার্মিনালে নিচের কমান্ডটি লিখতে হবেঃ

```
curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin --f
```

আমরা যেহেতু লারাভেলের নতুন ভার্সন ব্যবহার করব সেহেতু পিএইচপির নতুন ভার্সনের সাথে সাথে কিছু এক্সটেনশন থাকতে হবে নিচে সেগুলো নিচে উল্লেখ করা হলঃ

- PHP >= 5.6.4
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

কম্পোজার সহ সব কিছু ইন্টল আর কনফিগার করা হয়ে গেলে নিচের মত করে লারাভেল ইন্টল করুন।

টার্মিনাল কিংবা কমান্ড প্রম্পট ওপেন করুন। এবার টার্মিনাল হতে আপনার ওয়ার্কিং ডিরেক্টরিতে নেভিগেট করুন। এরপর নিচের মত করে কমান্ড লিখুনঃ

```
composer create-project laravel/laravel your-project-name --prefer-dist
```

এখানে **your-project-name** এর জায়গায় আপনার প্রজেক্টের নাম দিতে হবে। ধরুন আমাদের ক্ষেত্রে **howtocode** নাম দিলাম তাহলে আমাদেরকে প্রথম থেকে নিচের মত করে কমান্ড লিখতে হবেঃ

```
cd /var/www
composer create-project laravel/laravel howtocode --prefer-dist
cd howtocode
```

বিঃদ্রঃ **Composer** কমান্ডটি উইন্ডোজ মেশিনে রান করার সময় **github** এর এক্সেস চাইতে পারে সেই ক্ষেত্রে আপনার এক্সেসটি ব্যবহার করবেন।

এবার ইন্সটল করা হয়ে গেলে আমরা লারাভেলের আর্টিসান কমান্ড দিয়ে লারাভেল রান করতে পারি।

```
php artisan serve
```

আর <http://localhost:8000> লিংক দিয়ে প্রজেক্ট দেখতে পারব। আর্টিসান কমান্ডটি না ব্যবহার করতে চাইলে <http://localhost/howtocode/public/> লিংক দিয়ে অ্যাক্সেস করতে পারব।

অন্যদিকে আমরা লারাভেল ইন্সটলার ব্যবহার করেও খুব সহজেই লারাভেল ইন্সটল করে নিতে পারি নিচের মত করেঃ

ইন্সটলারটিকে গ্লোবালী ইন্সটল করার জন্যঃ

```
composer global require "laravel/installer=~1.1"
```

এবার কম্পোজারের `~/.composer/vendor/bin` ডিরেক্টরিকে **PATH** এ যুক্ত করতে হবে এর জন্য ইউনিক্স মেশিনের কমান্ড নিচে দেয়া হলঃ

```
export PATH="~/.composer/vendor/bin:$PATH"
```

এবার ইন্সটলার দিয়ে লারাভেল প্রজেক্ট তৈরি করার জন্যঃ

```
laravel new howtocode
```

পরিশেষে **Pretty URL** এর জন্য নিচের কনফিগার ব্যবহার করবেন।

Apache/htaccess এর জন্যঃ

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
```

nginx এর জন্যঃ

```
location / {  
    try_files $uri $uri/ /index.php?$query_string;  
}
```

লারাভেল ৫.৩ ফাইল বিন্যাস

লারাভেল ইন্সটল শেষ, আমাদের প্রোজেক্ট ডিরেক্টরি তে তো অনেক কিছু! বেশীর ভাগই লারাভেল এর নিজের ব্যবহারের জন্য কিন্তু বাকিটা আমাদের জন্য, যেখানে আমরা নতুন ফাইল তৈরি করবো, ডিফল্ট ফাইল এডিট করবো - আমাদের এপ্লিকেশন বানাতে। আসুন বেসিক ধারণা নিয়ে নেই।

এখানে সাধারণ ভাবে ৯টি ডিরেক্টরি ও কিছু ফাইল পাই।

১. app ২. bootstrap ৩. config ৪. database ৫. public ৬. resources ৭. routes ৮. storage ৯. tests
১০. vendor

app

এই ডিরেক্টরিতে সব এপ্লিকেশন লজিক থাকে, যেগুলো আমরাই বানাবো। যেমনঃ মডেল, কন্ট্রোলার, রাউট

bootstrap

এটা লারাভেল এর কিছু ফাইল থাকে যা দিয়ে ফ্রেমওয়ার্ক তাকে এক সাথে কাজ করতে সাহায্য করে, এখানে cache ফাইল গুলোও থাকে।

config

এই ডিরেক্টরির ফাইল গুলো ব্যবহার করে আমাদের অ্যাপ্লিকেশন কনফিগার করি, যেমন ডাটাবেস, মেইল, সেশন ইত্যাদি।

database

এখানে সব ডাটাবেস মাইগ্রেশন, সীড থাকে। এমনকি SQLite database ফাইলটাও এখানে থাকে।

public

সব ধরনের আসেট এখানে রাখে, যেমন CSS, JS, fonts, ছবি ইত্যাদি।

resources

আমাদের ফ্রন্ট এন্ড ফাইল যাকে ভিউ বলি সেগুলো এখানেই রাখব। তা ছাড়া LESS, SASS, CoffeeScript ও ভায়ার ফাইলগুলোও এখানে থাকে।

routes

এখানে সব রাউটের ডিফাইনেশন গুলো থাকে। যেমনঃ web.php, api.php, ও console.php

storage

লারাভেল ব্যবহার করে, সব কম্পাইলড টেমপ্লেট, সেশন, cache ফাইল রাখার জন্য।

tests

টেস্ট ফাইল গুলো থাকে, যেমনঃ PHPUnit ফাইল।

vendor

সব Composer dependency ফাইলগুলো থাকে।

আসলে লারাভেল নিয়ে কাজ করতে থাকলে এগুলো এমনভাবেই পরিচিত হয়ে যাবে।

তবুও আরও জানতে চাইলে [Laravel Application structure](#) দেখুন।

পরবর্তী চ্যাপ্টারে বেসিক রাউটিং নিয়ে আলোচনা করা হবে।

বেসিক রাউটিং

সফটওয়্যার আর্কিটেকচারে রাউট হচ্ছে মৌলিক উপাদান(Basic Component). রাউট URL(URL = Uniform Resource Locator)

থেকে অনুরোধ গ্রহণ করে এবং এপ্লিকেশনকে রিসোর্সের জন্য নির্দেশনা প্রদান করে। লারাভেলের রাউট সমূহ একই সাথে সুবিন্যস্ত করে রাখার জন্য **routes/web.php** তে লিখা হয়।

এর সবচেয়ে বড় সুবিধা হচ্ছে আপনি এক যায়গা থেকেই সমস্ত রাউটকে নিয়ন্ত্রণ করতে পারবেন অর্থাৎ পরবর্তিতে রাউট সম্পর্কিত যেকোনো ধরনের পরিবর্তন এখান থেকেই করতে পারবেন।

উদাহরণ হিসেবে একটি সাধারণ রাউট তুলে ধরা হল।

```
Route::get('/', function()  
{  
    return view('hello');  
});
```

উপরোক্ত উদাহরণে Route class এর get() মেথডকে কল করা হয়েছে। get() মেথডটি দুইটি আর্গুমেন্ট গ্রহণ করে। প্রথম আর্গুমেন্ট হিসেবে পাথ (URL Path) এবং দ্বিতীয় আর্গুমেন্ট হিসেবে ক্লোজার(Anonymous Function) যেমনঃ উপরোক্ত উদাহরণে প্রথম আর্গুমেন্ট হিসেবে ফরয়ার্ড স্ল্যাশ / ব্যবহার করা হয়েছে যার দ্বারা রুট ডোমেইনকে নির্দেশ করা হচ্ছে। দ্বিতীয় আর্গুমেন্ট হিসেবে ক্লোজার ব্যবহার করা হয়েছে যার দ্বারা একশন সম্পাদিত হচ্ছে। অর্থাৎ উক্ত রাউটটি **resources/views/hello.php** ফাইলকে রিটার্ন করবে।

কি বিষয়টু একটু ঘোলা মনে হচ্ছে? সমস্যা নাই চলুন আমরা আর একটি উদাহরণের মাধ্যমে স্বচ্ছ ধারণা লাভ করি।

```
Route::get('books', function()  
{  
    return 'Hey, I am laravel!';  
});
```

এখন যদি আপনি ব্রাউজার এর Address Bar এ **/books** (<http://localhost/your-project-name/books>) লিখে হিট করেন, তাহলে দেখতে পাবেন Hey, I am laravel!

অর্থাৎ get() মেথডটিতে প্রথম আর্গুমেন্ট হিসেবে books যা পাথ নির্দেশ করে এবং দ্বিতীয় আর্গুমেন্ট ক্লোজার যা দ্বারা কাঙ্ক্ষিত একশন সম্পাদিত হচ্ছে।

লারাভেলে রাউট মূলত দুই ভাবে লেখা যায়।

১) resource ডিফাইন করে রাউটারের উদাহরণঃ


```
Route::resource('users', 'UserController');
```

২) controller ডিফাইন করে রাউটারের উদাহরণঃ

```
Route::controller('users', 'UserController');
```

এ ক্ষেত্রে Controller class টি হবে।

```
class UserController extends BaseController {
    public function getIndex()
    {
        return 'I am restful index';
    }
}
```

নোট : এই দুই পদ্ধতি যথাযথভাবে কাজ করবে যদি আপনার Controller এবং Method সমূহ RESTful হয়।
RESTful Controller সম্পর্কে এখান থেকে [বিস্তারিত](#) পড়ুন

অনুশীলনঃ

কোন কিছু শেখার সময় হাতে কলমে করতে পারলে শেখাটা ভাল হয় তাই আমরা চেষ্টা করবো কিছু অনুশীলন করতে।

দ্বিতীয় অধ্যায়ে আমরা শিখেছিলাম লারাভেল ইন্সটল করতে, আসুন blog.app নামে একটি লারাভেল অ্যাপ বানাই ও বেসিক রাউটিং অনুশীলন করি, পরবর্তীতে আমরা আরও নতুন নতুন জিনিস শিখবো ও এটাকে উন্নত করব।

আমাদের নিজস্ব এনভায়রনমেন্ট(htdocs, www etc.) এ যাই ও টার্মিনালে নিচের কমান্ড দেই

```
composer create-project laravel/laravel blog.app
```

এই কমান্ড টি আমাদের ওয়ার্কিং ফোল্ডারে blog.app নামে একটি directory বানিয়ে তার ভিতরে লারাভেল এর যাবতীয় ফাইল ইন্টারনেট থেকে নামিয়ে নিবে।

আশাকরি ব্রাউজারে আপনি সাইটটি দেখতে পারছেন।

আসুন নতুন রাউট বানাই ៖

আমরা যদি **routes/web.php** ফাইলটি খুলি তাহলে এটা দেখবো

```
Route::get('/', function () {
    return view('welcome');
});
```

আসুন নতুন কিছু রুট বানাই এটাকে পরিবর্তন করে। এই মুহূর্তে আমরা সৌন্দর্যের কথা না ভেবে শুধু কার্যকরীটা দেখি।

```
Route::get('/', function () {
    return 'This is our home page.';
});

Route::get('/about', function () {
    return 'This is our about page.';
});

Route::get('/contact', function () {
    return 'This is our contact page.';
});

//Lets make some group route
Route::group(['prefix' => 'admin'], function () {
    Route::get('/', function () {
        return 'This is our Admin Dashboard';
    });
    // this link: blog.app/admin/

    Route::get('/user-list', function () {
        return 'This is our Admin Dashboard user list page';
    });
    // this link: blog.app/admin/user-list

    Route::get('/create-blog', function () {
        return 'This is our Admin Dashboard create-blog page';
    });
    //// this link: blog.app/admin/create-blog
});
```

এখানে নতুন হলো রুট গ্রুপ করা। 'prefix' ব্যবহার করে গ্রুপ তৈরি করেছি। আশা করি বুজতে পারছেন। এখন একটু কঠিন মনে হলেও পরবর্তী অধ্যায় করতে করতে সব আয়ত্তে এসে যাবে।

পরবর্তী চ্যাপ্টারে ভিউ নিয়ে আলোচনা করা হবে।

ভিউ

ভিউ পার্টের মধ্যে আপনার অ্যাপ্লিকেশনের সাধারণত HTML কন্টেন্ট গুলো থাকে, এবং কন্ট্রোলার ও মডেল(বিজনেস লজিক) সেপারেট করার এটি একটি ভাল পদ্ধতি। ভিউস ফাইল গুলো **resources/views** ডিরেক্টরিতে থাকে। নিচে একটা খুব সাধারণ উদাহরন দেয়া হলঃ

```
<!-- View stored in resources/views/welcome.php -->
<html>
  <body>
    <h1>Hello, <?php echo $name; ?> Welcome to the laravel world!</h1>
  </body>
</html>
```

ভিউ ফাইলটি ব্রাউজারে দেখতে চাইলে নিচের ন্যায় কোডটি route এ লিখতে হবে।

```
Route::get('/', function()
{
    return view('welcome', ['name' => 'Sohel Amin']);
});
```

ভিউস ডিরেক্টরির মধ্যে নেস্টেট সাব-ডিরেক্টরি থাকতে পারে। উদাহরন হিসেবে বলা যায় আপনি যদি ভিউ ফাইলটিকে এইভাবে রাখেনঃ **resources/views/admin/profile.php** তাহলে আপনাকে নিম্নের ন্যায় ভিউ হেল্পার মেথডটিকে কল করতে হবেঃ

```
return view('admin.profile', $data);
```

এখন আমি আপনাদেরকে কিভাবে ভিউ ফাইলের মধ্যে ডাটা পাস করতে হয় সেটা সম্পর্কে ধারণা দিব। আপনারা হয়ত খেয়াল করেছেন ভিউ মেথডটি কল করার সময় আমি দ্বিতীয় প্যারামিটার সহ দেখিয়েছি।

```
return view('admin.profile', $data);
```

এখানে দ্বিতীয় প্যারামিটার **\$data** দিয়ে অ্যারে পাস করা হয়েছে। এইক্ষেত্রে ভিউ ফাইলের মধ্যে আপনি ডাটা অ্যাকসেস করতে চাইলে **\$data** এর পরিবর্তে আপনাকে **\$data** অ্যারে এর ইনডেক্স কে কল করতে হবে। কারন কন্ট্রোলার অথবা রাউট থেকে ডাটা পাস করার সময় অ্যারে সয়ংক্রিয় ভাবে ভিউতে গিয়ে ভেরিয়েবল এ রূপান্তরিত হয়। আপনি যেকোনো ভিউ ফাইলকে চাইলে ভেরিয়েবেলের মধ্যে রাখতে পারেন এবং সেটি অন্য কোন ভিউ ফাইলে ইকো (echo) করতে পারেন। যেমন ধরুন আপনি কন্সট্যান্ট ফর্ম অথবা অন্য কিছু ছোট আকারের কোড একটা ভিউ ফাইলের মধ্যে রেখে দিয়েছেন আর অন্য ভিউ ফাইলে এইটা দেখাইতে চাচ্ছেন তাহলে আপনাকে নিচের মত করে লিখতে হবে।

```
$view = view('welcome', $data);
```

ভিউ ফাইলে অ্যারে পাস করা ছাড়াও আমরা লারাভেলের প্রচলিত নিয়মে ডাটা পাস করতে পারি আর এইজন্য নিচের ন্যায় লিখতে হবে।

```
// Using conventional approach  
$view = view('welcome')->with('name', 'Sohel Amin');
```

এটা ছাড়াও আমরা ম্যাজিক মেথড ব্যবহার করতে পারি যেটা আমাদেরকে আরেক রকম ভিন্ন স্বাদ দিবে।

```
// Using Magic Methods  
$view = view('welcome')->withName('Sohel Amin');
```

ধরুন এখন আপনি নাম এর পরিবর্তে ইমেইল পাস করবেন তাহলে আপনাকে নিচের মত করে লিখতে হবে।

```
$view = view('welcome')->withEmail('sohelamin@example.com');
```

আপনি চাইলে নাম আর ইমেইল একসাথে লিখতে পারেন।

```
return view('welcome')->withName('Sohel Amin')->withEmail('sohelamin@example.com');
```

এখন আপনি ভিউ ফাইলে যেভাবে ডাটা অ্যাকসেস করবেন সেটা নিচে দেখান হল।

```
<!-- View stored in resources/views/welcome.php -->  
<?php  
    var_dump($name);  
    var_dump($email);  
?>
```

এছাড়াও আপনি আপনার ডাটা কিংবা অ্যারে সব ভিউ ফাইলের সাথে শেয়ার করতে পারবেন এইভাবে।

```
view()->share('data', [1, 2, 3]);
```

অনুশীলনঃ

ভিউ সম্পর্কে তো কিছু ধারণা হলো, আসুন [পূর্ববর্তী অনুশীলন](#) এর সাথে আজ আমরা ভিউ যোগ করি। আমরা **resources/views** ফোল্ডারটি খুললে `welcome.blade.php` নামে একটি ভিউ দেখতে পাবো। মজার এই নামটির ৩টি অংশঃ প্রথমটি ভিউ এর নাম, দ্বিতীয় অংশঃ `.blade` যার মানে এই ফাইলে ব্লেড টেমপ্লেট থাকলে সেটা PHP এবং HTML এ রেন্ডার হবে(ব্লেড টেমপ্লেটিং সম্পর্কে আমরা পরবর্তী চ্যাপ্টারএ জানবো), তৃতীয় অংশঃ `.php`,

হা হা আপনি এটার মানে জানেন। এই আনুশীলনে আমরা HTML, CSS ও Twitter Bootstrap ব্যবহার করবো, যেহেতু আপনি লারাভেল শিখছেন সেহেতু আমি ধরে নিচ্ছি আপনি এগুলো জানেন তাই এই অংশ গুলর কোনও ব্যাখ্যা আমি দিয়ে আপনার সময় নষ্ট করব না।

আমাদের এপটির নাম ছিল blog.app, এবার blog.app/resources/views এর ভিতর `home.blade.php` নামে একটি ফাইল তৈরি করি। যার কন্টেন্ট নিম্নরূপ

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>HowtoCode By Laravel</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/boot
</head>
<body>
<div class="container">
  <div class="row">
    <div class="col-md-12 text-center">
      <h1>HowToCodeBD: Laravel 5.2.24</h1>
    </div>
  </div>
</div>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></scrip
</body>
</html>
```

এবার `app/Http/routes.php` ফাইলটি খুলি ও নিম্নরূপ পরিবর্তন করি

```
Route::group(['middleware' => ['web']], function () {

    Route::get('/', function () {
        return view('home');
    });

});
```

লারাভেল জানে কোথায় ভিউ ফাইল থাকে আর কি ফাইল এক্সটেনশন, তাই শুধু ভিউ এর নামটা দিলেই কাজ হয়।

এবার আপনি `home.blade.php` এর মত করে `about.blade.php` এবং `contact.blade.php` দুইটা ফাইল তৈরি করেন একই পথ এ। নতুন ভিউ দুটার জন্য `routes.php` তে কিছু কোড যোগ করিঃ

```
Route::get('/about', function () {
    return view('about');
});

Route::get('/contact', function () {
    return view('contact');
});
```

এভাবে যদি সব ভিউ একই জায়গায় থাকে তাহলে বেশ সমস্যা হয়ে যায়, আসুন না আমরা একই টাইপের ভিউ গুলোকে এক একটি ফোল্ডার এর ভিতর রাখি। ধরি আমাদের এডমিন এরিয়ার জন্য কিছু ভিউ আছে, তাই `resources/views` এর মধ্যে `admin` নামে ফোল্ডার বানাই। এবং এই ফোল্ডার এর মধ্যে `dashboard.blade.php` এবং `users.blade.php` নামে ভিউ বানাই। হয়ে গেল তো? এবার এগুলোর জন্য `Routes.php` তে রাউট বানাই

```
Route::get('/admin', function () {
    return view('admin.dashboard');
});

Route::get('/admin/users', function () {
    return view('admin.users');
});
```

`admin` ফোল্ডার এর জন্য `admin.{view-name}`, কিন্তু যদি `admin` ফোল্ডার এর ভিতর আরও একটি ফোল্ডার থাকত এবং তার ভিতর কিছু ভিউ থাকতো? ঠিক ধরেছেন! এমন হতো - `admin.another-folder.{view-name}` মনে আছে গত অনুশীলন এ Route Group বানিয়েছিলাম? দেখি অ্যাডমিন ভিউ এর রুট গুলোকে সেটার মধ্যে নিলে কেমন হয়।

```
//Lets make some group route
Route::group(['prefix' => 'admin'], function () {
    Route::get('/', function () {
        return view('admin.dashboard');
    });
    // this link: blog.app/admin

    Route::get('/users', function () {
        return view('admin.users');
    });
    // this link: blog.app/admin/users
});
```

আমরা কিন্তু এই দুটা অনুশীলনএ বেসিক রাউট ও বেসিক ভিউ সম্পর্কে জানলাম।

পরবর্তী চ্যাপ্টারে ব্লেড টেমপ্লেটিং নিয়ে আলোচনা করা হবে যেটা আপনার ভিউ ফাইলকে আরও সহজভাবে উপস্থাপনে সহযোগিতা করবে।

ব্লেড টেমপ্লেটিং

লারাভেল এর সাথে ব্লেড টেমপ্লেট ইঞ্জিন বিল্ড ইন ভাবে এসেছে। আর এটা ব্যবহার করা খুবই সহজ। ব্লেড টেমপ্লেটে কোড লেখার জন্য ভিউ ফাইলের এক্সটেনশন হতে হবে **.blade.php** ধরুন আপনার ভিউ ফাইলটির নাম **about** তাহলে ব্লেড এর ফরমেটে এটি হবে **about.blade.php**। এখন ব্লেড এর নিয়ম অনুসারে ভিউ পেজ গুলো সাজানোর জন্য একটা মাস্টার পেজ বানাতে হবে যেটাকে এক্সটেন্ড করে অন্য ভিউ গুলো বানাতে হবে। তাহলে আমরা যদি **about** এর জন্য একটি ভিউ বানাই ব্লেড এর নিয়ম অনুযায়ী, প্রথমে **route** এ **about** ডিফাইন করতে হবে তারপর মাস্টার পেজ বানাইতে হবে তারপর **about** পেজটি বানাইতে হবে। নিচে ধারাবাহিক ভাবে কোড গুলো দেখানো হল।

Route এর জন্য।

```
Route::get('/about', function(){
    return view('about');
});
```

মাস্টার পেজের জন্য।

```
<!-- Stored in resources/views/layouts/master.blade.php -->

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Laravel Project</title>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/
    </head>
    <body>
        @yield('content')
        @yield('footer')
    </body>
</html>
```

About পেজের জন্য।


```
<!-- Stored in resources/views/about.blade.php -->

@extends('master')

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10">
            <p>Your contents goes here</p>
        </div>
    </div>
</div>
@endsection

@section('footer')
    <script src="{{ URL::asset('assets/js/myscript.js') }}"></script>
@stop
```

এখন ধরুন আপনাকে বিশেষ ক্ষেত্রে কনটেন্ট সেকশন অথবা অন্য কোন সেকশনের জন্য ডিফল্ট কনটেন্ট/ডাটা রাখতে হচ্ছে তাহলে আপনাকে **yield** মেথডটিতে দ্বিতীয় প্যারামিটার পাস করে এটা করতে হবে।

```
@yield('section', 'Default Content')
```

ব্লেডে ডাটা **echo** করার জন্য ডাবল কারলি ব্রাকেট ব্যবহার করতে হয়।

```
Hello, {{ $name }}.

The current UNIX timestamp is {{ time() }}.
```

এইক্ষেত্রে ডেরিয়েবল ডিফাইন করার জন্য ব্লেডের কমেণ্টস ব্যবহার করতে হবে নিচের মত করে।

```
@php
    $name = 'Sohel Amin';
@endphp

Hello, {{ $name }}
```

এবার ব্লেডে কন্ট্রোল স্ট্রাকচার সহ অন্যান্য ফিচার গুলো ধারাবাহিক ভাবে দেখানো হল।

If Statements এর জন্য নিচের পদ্ধতি অনুসরণ করুন।

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif

@unless (Auth::check())
    You are not signed in.
@endunless
```

Loops এর জন্য নিচের পদ্ধতি অনুসরণ করুন।

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

একটা ভিউ ফাইলে অন্য ভিউ ফাইল ইন্ক্লুড করার জন্য নিচের মত করে লিখতে হবে।

```
@include('view.name')
```

আর আপনি চাইলে ভিউ ফাইল ইন্ক্লুড করার সময়ে ডাটা পাস করতে পারেন এইভাবে।

```
@include('view.name', ['some' => 'data'])
```

কমেন্টস লিখতে চাইলে।

```
{{-- This comment will not be in the rendered HTML --}}
```

ব্লেড টেমপ্লেটিং: HTML ও Forms

লারাভেল এর ব্রড টেমপ্লেট ব্যবহার করে HTML ও Forms তৈরি করতে আমাদের একটি "Third party package" প্রয়োজন হবে। আসুন জেনে নেই কি ভাবে আমরা এই সুবিধা আমাদের প্রোজেক্ট এ আনতে পারি।

প্রোজেক্ট ডিরেক্টরি এর মূল পাথে `composer.json` নামে একটি ফাইল আছে যেটা আমাদের প্রোজেক্ট এর প্রয়োজনীয় প্যাকেজ গুলার তালিকা রাখে। ফাইলটির প্রথম কিছু অংশ অনেকটা এরকমঃ

```
{
  "name": "laravel/laravel",
  "description": "The Laravel Framework.",
  "keywords": ["framework", "laravel"],
  "license": "MIT",
  "type": "project",
  "require": {
    "php": ">=5.6.4",
    "laravel/framework": "5.3.*"
  },
  .
  .
  .
  .
```

ফাইলে "require" অংশে আমাদের প্রয়োজনীয় প্যাকেজটি যোগ করি, তাহলে এটা দেখতে হবে অনেকটাঃ

```
{
  "name": "laravel/laravel",
  "description": "The Laravel Framework.",
  "keywords": ["framework", "laravel"],
  "license": "MIT",
  "type": "project",
  "require": {
    "php": ">=5.6.4",
    "laravel/framework": "5.3.*",
    "laravelcollective/html": "5.3.*"
  },
  .
  .
  .
  .
```

এবার "Composer" ব্যবহার করে এটাকে আমাদের প্রোজেক্ট এ যোগ করি নিচের command টি Terminal এ লিখে

```
composer update
```

এটা আমাদের সব প্যাকেজ গুলোকে update, প্রয়োজনে add করে নিবে।

এবার `config/app.php` ফাইলটি খুলি, যেটা সব "provider" ও এর 'alias' এর মধ্যে রাখে এবং অ্যাপ run করার সময় autoloader এর মাধ্যমে এগুলোকে load করে।

```
Collective\Html\HtmlServiceProvider::class,
```

লাইনটি provider এরে এর শেষে যোগ করি।

```
'providers' => [
    // ... previous providers classes

    Collective\Html\HtmlServiceProvider::class,

],
```

শেষ কাজ aliases এরে তে দুইটি class এর alias যোগ করা

```
'aliases' => [
    // ... previous aliases
    'Form' => Collective\Html\FormFacade::class,
    'Html' => Collective\Html\HtmlFacade::class,
],
```

আমরা প্রস্তুত ব্রেড টেমপ্লেট ব্যবহার করে HTML ও Forms তৈরি করত!!!!

উদাহরণ দেখার আগে একটি বিষয় জেনে নেই

```
{{ $data }}
```

escaped value দেখাবে

```
{!! $data !!}
```

unescaped value দেখাবে

একটু উদাহরণ দেই

```
$data = '<strong> How to Code BD </strong>'
{{ $data }}
```

দেখাবে: How to Code BD

```
$data = '<strong> How to Code BD </strong>'
{!! $data !!}
```

দেখাবে: **How to Code BD**

এইবার ফর্ম হেল্পার ব্যবহার করে ফর্ম **echo** করার জন্য নিচের মত করে লিখতে হবে।

```
{!! Form::open(['url' => '#']) !!}

{!! Form::close() !!}
```

এবং HTML আউটপুট হবে

```
<form accept-charset="UTF-8" action="#" method="POST">
  <input type="hidden" value="hbyt7fA7Mw09iT8V54z2V5u8j0mFFJJckSs7XI9G" name="_token">

</form>
```

এখানে 'POST' মেথড দিবে যদি অন্য কোন মেথড আমরা উল্লেখ না করি। সাথে একটি `html <input type="hidden" value="hbyt7fA7Mw09iT8V54z2V5u8j0mFFJJckSs7XI9G" name="_token">` যা আমাদের সিকুইরিটি বাড়াবে।

এবার, নানা ধরনের `input element` গুলো আমাদের ফর্মে যোগ করিঃ

```
<div id="simple-form">
  <h3>User Profile</h3>
  <hr/>
  {!! Form::open(array('url' => '#')) !!}

  {!! Form::label('user_name', 'Name:') !!}
  {!! Form::text('user_name') !!}
  <br>
  {!! Form::label('email', 'Email:') !!}
  {!! Form::email('email', $value = null) !!}
  <br>
  {!! Form::label('gender', 'Gender:') !!}
  {!! Form::radio('gender', 'Male', true) !!} Male
  {!! Form::radio('gender', 'Female', false) !!} Female
  {!! Form::radio('gender', 'Unisex', false) !!} Unisex
  <br>
  {!! Form::label('role', 'Role:') !!}
  {!! Form::select('role', array(
    'admin' => 'Administrator',
    'author' => 'Author',
    'subscriber' => 'Subscriber',
    'registered' => 'registered'

  )) !!}
  <br>
  {!! Form::label('about', 'About Me') !!}<br>
  {!! Form::textarea('about') !!}
  <br><br>
  {!! Form::submit('Update') !!}
  {!! Form::close() !!}
</div>
```

ব্রাউজার দেখাবেঃ

User Profile

Name:

Email:

Gender: ☒ Male ☐ Female ☐ Unisex

Role: 

About Me

HTML আউটপুট হবেঃ

```
<div id="simple-form">
  <h3>User Profile</h3>
  <hr>
  <form accept-charset="UTF-8" action="#" method="POST"><input type="hidden" value=
    <label for="user_name">Name:</label>
    <input type="text" id="user_name" name="user_name">
    <br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email">
    <br>
    <label for="gender">Gender:</label>
    <input type="radio" id="gender" value="Male" name="gender" checked="checked">
    <input type="radio" id="gender" value="Female" name="gender"> Female
    <input type="radio" id="gender" value="Unisex" name="gender"> Unisex
    <br>
    <label for="role">Role:</label>
    <select name="role" id="role">
      <option value="admin">Administrator</option>
      <option value="author">Author</option>
      <option value="subscriber">Subscriber</option>
      <option value="registered">registered</option>
    </select>
    <br>
    <label for="about">About Me</label><br>
    <textarea id="about" rows="10" cols="50" name="about"></textarea>
    <br><br>
    <input type="submit" value="Update">
  </form>
</div>
```

কিন্তু আমরা এই ফর্ম এর সাথে CSS দিবা, আমাদের id, class ও নানা ধরনের attribute ও যোগ করতে হবে।
পূর্ববর্তী অনুশীলনে আমরা Twitter Bootstrap দিয়েছিলাম, এবার এই অনুশীলনে

- ভিউ তে blade ব্যবহার করে ভিউ আরও ব্যবহার যোগ্য করবো
- Form এর জন্য নতুন Template বানাবো, রাউট সেট করবো
- লারাভেল blade ও Twitter Bootstrap মিলিয়ে একটি Form তৈরি করবো

অনুশীলনঃ

গত অধ্যায় এ আমরা ভিউ এবং এই অধ্যায় এ blade template দেখলাম। আসুন একটি master.blade.php ফাইল বানাই `resources/views` এর ভিতর।


```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>@yield('title')</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/boot
</head>
<body>
  <div class="container">
    @yield('content')
  </div>

  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></s
</body>
</html>
```

@yield('title') ও @yield('content') এর মানে title ও content নামে চাইল্ড টেমপ্লেটে কোনও section থাকলে এই অংশে দেখাবে। এটা আমরা একটি "Parent template" বানালাম।

এবার একটি "Child Template" বানাই উপরের blade template ব্যবহার করে যে Form টি বানিয়েছি সেটা নিয়ে।

resources/views এর ভিতর pages নামে একটি ডাইরেক্টরি ও তার ভিতর form.blade.php নামে ফাইল তৈরি করি। এবং নিচের code গুলো লিখি।

```
@extends('master')

@section('title', 'Forms')

@section('content')
    <div class="row">
        <div class="col-md-6 col-md-offset-3">
            <h3>User Profile</h3>
            <hr/>
            {!! Form::open(array('url' => '#')) !!}

            {!! Form::label('user_name', 'Name:') !!}
            {!! Form::text('user_name') !!}
            <br>
            {!! Form::label('email', 'Email:') !!}
            {!! Form::email('email', $value = null) !!}
            <br>
            {!! Form::label('gender', 'Gender:') !!}
            {!! Form::radio('gender', 'Male', true) !!} Male
            {!! Form::radio('gender', 'Female', false) !!} Female
            {!! Form::radio('gender', 'Unisex', false) !!} Unisex
            <br>
            {!! Form::label('role', 'Role:') !!}
            {!! Form::select('role', array(
                'admin' => 'Administrator',
                'author' => 'Author',
                'subscriber' => 'Subscriber',
                'registered' => 'registered'

            )) !!}
            <br>
            {!! Form::label('about', 'About Me') !!}<br>
            {!! Form::textarea('about') !!}
            <br><br>
            {!! Form::submit('Update') !!}
            {!! Form::close() !!}
        </div>
    </div>
@endsection
```

তিনটি জিনিস খেয়াল করার আছেঃ

`@extends('master')` মানে এই ফাইলটি `views` এর ভিতর যে `master.blade.php` আছে সেটি ব্যবহার করবে।
`@section('title', 'Forms')` এটি `section`, প্রথম প্যারামিটারটি `section` এর নাম ও দ্বিতীয় প্যারামিটারটি `section` এর `value`. `Value`টি যদি এমন একটি স্ট্রিং হয় তো এভাবেই লেখা যায়। কিন্তু দেখেন `content` নামের `section`টি শুধু প্রথম প্যারামিটারটি ব্যবহার করে নামটি দিয়েছে পরে এর বিশাল `value` দিয়েছে এবং অবশেষে `@endsection` লিখে `section`টি শেষ করেছে।

ফাইল তো হল, এদের জন্য রাউট তৈরি করি। `routes/web.php` ফাইলটি খুলে নিচের মত রাউট বানাই। আগের অনুশীলন গুলো করে থাকলে এটা আপনার জন্য খুব সহজ।

```
Route::get('/form', function () {
    return view('pages.forms');
});
```

এখানে pages.forms লেখার কারণ pages ডাইরেক্টরি এর মধ্যে forms.blade.php ফাইলটি আছে। আর লারাভেল তো জানেই ডিউগুলো কোথায় থাকে।

আসুন এবার Twitter Bootstrap এর সাথে আমাদের HTML Blade মিল্লাপ করি।

আপনার forms.blade.php ফাইলটি আপডেট করুনঃ

```
<div class="col-md-6 col-md-offset-3">
    <h3>User Profile</h3>
    <hr/>
    {!! Form::open(array('url' => '#', 'class'=>'form-horizontal')) !!}
    <div class="form-group">
        {!! Form::label('user_name', 'Name:') !!}
        {!! Form::text('user_name', '', array('class' =>'form-control', 'placeholder' => 'Name')) !!}
    </div>
    <div class="form-group">
        {!! Form::label('email', 'Email:') !!}
        {!! Form::email('email', $value = null, array('class' =>'form-control', 'placeholder' => 'Email')) !!}
    </div>
    <div class="form-group">
        {!! Form::label('gender', 'Gender:') !!}
        {!! Form::radio('gender', 'Male', true) !!} Male
        {!! Form::radio('gender', 'Female', false) !!} Female
        {!! Form::radio('gender', 'Unisex', false) !!} Unisex
    </div>
    <div class="form-group">
        {!! Form::label('role', 'Role:') !!}
        {!! Form::select('role', array(
            'admin' => 'Administrator',
            'author' => 'Author',
            'subscriber' => 'Subscriber',
            'registered' => 'registered'
        ), null, ['class' => 'form-control']) !!}
    </div>
    <div class="form-group">
        {!! Form::label('about', 'About Me') !!}<br>
        {!! Form::textarea('about', '', array('class'=>'form-control', 'rows' => '4')) !!}
    </div>
    {!! Form::submit('Update Profile', array('class' => 'btn btn-primary')) !!}
    {!! Form::close() !!}
</div>
```

ব্রাউজার দেখাবেঃ

User Profile

Name:

User Name

Email:

Email Address

Gender: ☐ Male ☐ Female ☒ Unisex

Role:

Administrator

About Me

Update Profile

HTML আউটপুট হবেঃ

```
<form class="form-horizontal" accept-charset="UTF-8" action="#" method="POST"><input type
  <div class="form-group">
    <label for="user_name">Name:</label>
    <input type="text" id="user_name" value="" name="user_name" placeholder="User Nam
  </div>
  <div class="form-group">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" placeholder="Email Address" class="fo
  </div>
  <div class="form-group">
    <label for="gender">Gender:</label>
    <input type="radio" id="gender" value="Male" name="gender" checked="checked"> Mal
    <input type="radio" id="gender" value="Female" name="gender"> Female
    <input type="radio" id="gender" value="Unisex" name="gender"> Unisex
  </div>
  <div class="form-group">
    <label for="role">Role:</label>
    <select name="role" id="role" class="form-control"><option value="admin">Administ
  </div>
  <div class="form-group">
    <label for="about">About Me</label><br>
    <textarea id="about" cols="50" name="about" rows="4" class="form-control"></text
  </div>
  <input type="submit" value="Update Profile" class="btn btn-primary">
</form>
```

কি লারাভেল Blade templating মজার না?

আজকে এই পর্যন্ত, এর পরের চ্যাপ্টারে কন্ট্রোলার সম্পর্কে আলোচনা করা হবে।

কন্ট্রোলার

কন্ট্রোলার হচ্ছে মডেল ও ভিউ এর মধ্যবর্তী একটা অংশ, এটি **HTTP Request** গুলো সার্ভ করে থাকে। যদিও আমরা এই কাজটি লারাভেলে কন্ট্রোলার ছাড়া রাউট ডিফাইন করে ভিউ ও মডেলের মধ্যে যোগাযোগ করতে পারি। আলাদাভাবে কন্ট্রোলার ব্যবহার করা হচ্ছে একটি আদর্শ পদ্ধতি। **routes/web.php** ফাইলের মধ্যে সকল প্রকার লজিক গুলো ডিক্লেয়ার না করে একটা পৃথক ফাইলের মধ্যে রাখা যায়, যেটি একটি সিঙ্গেল ক্লাসের মধ্যে সীমাবদ্ধ থাকবে। আর যখন ওই কন্ট্রোলার ক্লাসের কোন ফাংশন অথবা কোন মেথডকে **HTTP Request** এর মাধ্যমে কল করার দরকার হবে তখন সেটা রাউটের মধ্যে ডিফাইন করে দিতে হবে। লারাভেলে কন্ট্রোলার ফাইল গুলো **app/Http/Controllers** এই ডিরেক্টরিতে রাখতে হয়।

বেসিক কন্ট্রোলারঃ

এখন ইউজার এর জন্য একটা বেসিক কন্ট্রোলার তৈরি সম্পর্কে আমি আপনাদেরকে ধারণা দিব। ধরুন কন্ট্রোলারটিতে ২টি মেথড রাখব নিচে সেটা দেয়া হল।

```
<?php namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

class UserController extends Controller {

    public function index() {
        return "You've arrived at User Controller.";
    }

    public function showProfile($id) {
        $data = [
            'name' => 'Sohel Amin',
            'email' => 'sohelamin@example.com'
        ];
        return view('user.profile', $data);
    }
}
```

এখানে মূল রাউটের জন্য **index()** আর ইউজার প্রোফাইল দেখার জন্য **showProfile()** নামে ২টি মেথড ব্যবহার করা হয়েছে। কন্ট্রোলার ডিক্লেয়ার করার সময় মনে রাখতে হবে যেন সঠিক ভাবে কন্ট্রোলার এর নেমস্পেস ডিফাইন করা হয়। এইক্ষেত্রে এইখানে **App\Http\Controllers** নেমস্পেস ব্যবহার করা হয়েছে। কন্ট্রোলার এর নেমস্পেস ডিফাইন করা খুবই সহজ কারণ কন্ট্রোলার এর ডিরেক্টরি লোকেশন আর নেমস্পেস একই। এরপর কন্ট্রোলার এর নাম দিতে হয়। ইউজার কন্ট্রোলার এর জন্য দেয়া হয়েছে **UserController** মনে রাখবেন কন্ট্রোলার এর নাম আর কন্ট্রোলার এর ফাইল এর নাম একই হতে হবে। যেমনঃ **UserController.php** এখন ব্রাউজারে কন্ট্রোলারটিকে

একসেস করতে চাইলে নিচের মত করে রাউট ডিফাইন করতে হবে। `Route::get('user', 'UserController@index');` উপরের লাইনের মাধ্যমে **index** মেথডটিকে কল করা হয়েছে আর এর জন্য ব্রাউজারে দিতে হবে <http://localhost:8000/user/> এরকম এড্রেস।

```
Route::get('user/{id}', 'UserController@showProfile');
```

উপরের লাইনের মাধ্যমে **showProfile** মেথডটিকে কল করা হয়েছে আর এর জন্য ব্রাউজারে দিতে হবে <http://localhost:8000/user/1> এরকম এড্রেস। এখানে **1** এর জায়গায় আপনার কাস্টম **id** ব্যবহার করতে হবে। আর **id** অনুযায়ী ডাটাবেস থেকে ডাটা আনতে হবে মডেল এর সাহায্যে। এই উদাহরণে আমি আপনাদেরকে শুধু একটা অ্যারে এর মাধ্যমে দেখিয়েছি। আপনারা পরে মডেল ব্যবহার করে এটা করবেন।

কন্ট্রোলার আর নেমস্পেসঃ

অ্যাপ্লিকেশন ডেভেলপ করার সময় কিছু কন্ট্রোলারকে পৃথক কোন ফোল্ডার কিংবা ডিরেক্টরিতে রাখতে হয়। আর এইক্ষেত্রে আমরা কাস্টম নেমস্পেস ব্যবহার করে তা করতে পারি। ধরুন লারাভেলের কন্ট্রোলার ডিরেক্টরিতে **Photos** নামে একটি সাব-ডিরেক্টরি তৈরি করব আর ওই সাব-ডিরেক্টরিতে **AdminController** রাখব তাহলে কন্ট্রোলার ফাইলটি নিচের মত হবে।

```
<?php namespace App\Http\Controllers\Photos;

use App\Http\Controllers\Controller;

class AdminController extends Controller {

    public function index() {
        return "You've arrived at Admin Controller.";
    }

}
```

আর রাউট ফাইলে নিচের মত করে ডিফাইন করতে হবে।

```
Route::get('photos/admin', 'Photos\AdminController@index');
```

আর এখানে **AdminController** এর সামনে শুধু **Photos** নেমস্পেস উল্লেখ করা হয়েছে।

ইমপ্লিসিট কন্ট্রোলারসঃ

লারাভেলে একটা সিঙ্গেল কন্ট্রোলার এর সকল মেথডকে একটা সিঙ্গেল রাউটের মধ্যে ডিফাইন করা যায়। ধরুন আমরা ইউজার কন্ট্রোলার এর মেথডগুলো নিচের মত করে রাখব।

```
<?php namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

class UserController extends Controller {

    public function getIndex() {
        //
    }

    public function postProfile() {
        //
    }

    public function anyLogin() {
        //
    }

}
```

তাহলে এর জন্য রাউটে নিচের মত করে ডিফাইন করতে হবে।

```
Route::controller('users', 'UserController');
```

এখানে মেথডের সামনে প্রিফিক্স হিসেবে **get**, **post** আর **any** ব্যবহার করা হয়েছে। আর এই প্রিফিক্স ব্যবহার করার ফলে স্বয়ংক্রিয় ভাবে রাউট সেই মেথডগুলোকে **HTTP Verb** হিসেবে ডিফাইন করে নিবে। এখানে **any** এর মাধ্যমে **HTTP** এর সকল **Verb** রেজিস্টার হবে।

আপনারা রেজিস্টারকৃত রাউটগুলো খুব সহজেই কমান্ড প্রমোট অথবা টার্মিনালে দেখতে পারেন এই কমান্ডটির মাধ্যমে।

```
php artisan route:list
```

বেস্টফুল রিসোর্স কন্ট্রোলারঃ

আপনারা যদি বেস্টফুল কন্ট্রোলার তৈরি করতে চান তাহলে রিসোর্স কন্ট্রোলার এর মাধ্যমে এটা করতে পারেন যেটি আপনাদেরকে অনেক সময় বাঁচিয়ে দিবে।

Artisan এর মাধ্যমে আপনারা খুব সহজেই বেস্টফুল কন্ট্রোলার তৈরি করতে পারেন। ধরুন এখন আমরা **Photo** কন্ট্রোলার তৈরি করব তাহলে আমাদেরকে নিচের মত করে টার্মিনালে লিখতে হবে।

```
php artisan make:controller PhotoController
```


এবার রাউটে নিচের মত লিখতে হবে।

```
Route::resource('photo', 'PhotoController');
```

যার ফলে কন্ট্রোলার এ আগেই ডিফাইনকৃত কিছু মেথড পাওয়া যাবে। এখন উক্ত কন্ট্রোলারে আপনার প্রয়োজনীয় লজিক গুলো ডিফাইন করতে হবে।

রিসোর্স কন্ট্রোলারটি নিচের মত করে অ্যাকশন হ্যান্ডেল করবে।

Verb	Path	Action	Route Name
GET	/photo	index	photo.index
GET	/photo/create	create	photo.create
POST	/photo	store	photo.store
GET	/photo/{photo}	show	photo.show
GET	/photo/{photo}/edit	edit	photo.edit
PUT/PATCH	/photo/{photo}	update	photo.update
DELETE	/photo/{photo}	destroy	photo.destroy

এখন আপনি আপনার প্রয়োজনে রিসোর্স রাউট এর অ্যাকশন কাস্টমাইজ করে নিতে পারেন। ধরুন আপনি **index** আর **show** অ্যাকশন/মেথড রাখতে চাচ্ছেন তাহলে রাউটে নিচের মত করে লিখতে হবে।

```
Route::resource('photo', 'PhotoController',
    ['only' => ['index', 'show']]);
```

এবার আপনি বিশেষ কিছু অ্যাকশন বাদ দিবেন তাহলে নিচের মত করে তা করতে পারেন।

```
Route::resource('photo', 'PhotoController',
    ['except' => ['create', 'store', 'update', 'destroy']]);
```

রিসোর্স কন্ট্রোলারে বাই ডিফল্ট কিছু নাম থাকে মেথড কিংবা অ্যাকশন এ আপনি চাইলে সহজেই তা পরিবর্তন করতে পারেন রাউটের মাধ্যমে। এইক্ষেত্রে কন্ট্রোলার ক্লাসের মধ্যে কোন পরিবর্তন করতে হবে না।

```
Route::resource('photo', 'PhotoController',
    ['names' => ['create' => 'photo.build']]);
```

এবার আপনি যদি নেক্ট রিসোর্স কন্ট্রোলার ব্যবহার করতে চান অর্থাৎ **Url** এরকম হবে

photos/{photos}/comments/{comments}

তাহলে আপনি সেটা রাউটে নিচের মত করে লিখতে পারেন।

```
Route::resource('photos.comments', 'PhotoCommentController');
```

এইক্ষেত্রে আগের ডিফাইন করা রাউটটিকেও রাখতে হবে।

এবার ধরুন আপনাকে অতিরিক্ত কিছু মেথড আপনার কন্ট্রোলারে রাখতে হচ্ছে এবং আপনি তা রাউটেও ডিফাইন করতে চাচ্ছেন তাহলে আপনাকে নিচের মত করে লিখতে হবে।

```
Route::get('photos/popular', 'PhotoController@method');  
Route::resource('photos', 'PhotoController');
```

এইক্ষেত্রে অবশ্যই মনে রাখবেন রাউটে রিসোর্স কন্ট্রোলার ডিফাইন করার আগেই ওই মেথডটি রাউটে ডিফাইন করতে হবে।

কন্ট্রোলার মিডলওয়্যারঃ

মিডলওয়্যারকে কন্ট্রোলারের রাউটে ডিফাইন করা যায়। ধরুন আমরা ইউজার অ্যাথেন্টিকেশনের জন্য **auth** মিডলওয়্যারটি ব্যবহার করব তাহলে রাউটে নিচের মত করে লিখতে হবে।

```
Route::get('profile', [  
    'middleware' => 'auth',  
    'uses' => 'UserController@showProfile'  
]);
```

এবার কন্ট্রোলারের কম্পট্রাক্টরে ***\$this->middleware('auth');*** এই লাইনটি যোগ করতে হবে। নিচে কন্ট্রোলার এর কোডটি কিছু উদাহরণ সহ দেয়া হল।

```
<?php namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

class UserController extends Controller {

    /**
     * Instantiate a new UserController instance.
     */
    public function __construct()
    {
        // auth middleware will affect on all method in this controller
        $this->middleware('auth');
        // log middleware will only affect for fooAction & barAction
        $this->middleware('log', ['only' => ['fooAction', 'barAction']]);
        // subscribed middleware will affect on all method exclude fooAction & barAction
        $this->middleware('subscribed', ['except' => ['fooAction', 'barAction']]);
    }

}
```

অনুশীলনঃ

আজ জানলাম controller এবং বুঝলাম routes.php থেকে সরাসরি view তে না গিয়ে controller ব্যবহার করা উচিত। আসুন আমাদের আগের অনুশীলন গুলি পরিবর্তন করে routes.php থেকে সরাসরি view তে না গিয়ে controller ব্যবহার করি।

লারাভেল এর দারুণ এক ফিচার Artisan CLI. ভবিষ্যৎ অনুশীলনে আমরা এটার প্রচুর ব্যবহার দেখবো। আর কথা না বাড়িয়ে নিচের command টি run করাই।

```
php artisan make:controller PagesController
```

Controller created successfully. এই বার্তা আমাদের Terminal এ আসলেই বুজব লারাভেল আমাদের হয়ে একটি ফাইল তৈরি করেছে app/Http/Controllers পাথে, আমাদের দেওয়া নাম অনুসারে PagesController.php এবং আমাদের প্রয়োজনীয় কিছু code লিখেছে নিচের মতো।

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class PagesController extends Controller
{
    //
}
```

এবার এই class PagesController ভিতর আমাদের প্রয়োজন মতো কিছু method লিখে ফেলি।

```
class PagesController extends Controller
{
    public function home(){
        return view('home');
    }

    public function about(){
        return view('about');
    }

    public function forms(){
        return view('pages.forms');
    }
}
```

এবার routes.php কেও update করে ফেলিঃ

```
Route::group(['middleware' => ['web']], function () {

    Route::get('/', 'PagesController@home');

    Route::get('/about', 'PagesController@about');

    Route::get('/form', 'PagesController@forms');

});
```

এবার ব্রাউজারে একটু চেক করে নিন। যদি `NotFoundHttpException in RouteCollection.php` এই এররটি দেয় তাহলে দেখুন আপনার view গুলা আছে কিনা এবং আপনি সঠিক ভাবে address bar এ address টি লিখেছেন কিনা। আগের অনুশীলন গুলি করা থাকলে সমস্যা হবার কথা নয়।

আজকে এই পর্যন্তই। এর পরের চ্যাপ্টারে মাইগ্রেশন নিয়ে আলোচনা করা হবে।

মাইগ্রেশন

লারাভেল মাইগ্রেশন(কিছু ফাইল এর মধ্যে কিছু ক্লাস)এর মাধ্যমে জানে আমরা ডাটাবেজ এ কি কি করছি। অর্থাৎ যখন আমরা টেবিল বানাই, কোনও ফিল্ড এর নাম পালটাই সবই আমরা এই মাইগ্রেশন এর ভিতর লিপিবদ্ধ করি।

বই এর ভাষায় বলতে গেলে মাইগ্রেশন(Migration) হলো আমাদের এপ এর ডাটাবেজ(Database) এর স্কিমা(Schema).

সুবিধাঃ

কোড লেখার সময় আমরা নানা রকম ভাশর্ন কন্ট্রোল ব্যবহার করি, যেমন git. এর সুবিধা আমরা সবাই জানি যে, দলের সবাই আমাদের কাজের পরিবর্তন দেখছে, প্রয়োজনে যেকোনো কাজ আমরা undo করে আগের যেকোনো পর্যায়ে সহজে জেতে পারি। লারাভেল এর মাইগ্রেশন ব্যবহার করে কোড এর মতো ডাটাবেজ এও এই সুবিধা নিতে পারি। অর্থাৎ মাইগ্রেশন হলো অনেকটা ডাটাবেজ এর ভাশর্ন কন্ট্রোল সিস্টেম। শুধু তাই না, লারাভেল কে `migrate` করতে বললে লারাভেল আমাদের তৈরি `migration` এর উপর ভিত্তি করে আমাদের ডাটাবেজ এ সব টেবিল, সব ফিল্ড, সব পরিবর্তন-পরিবর্ধন এবং আরও যা যা থাকবে সব ঠিক ঠিক তৈরি করে দিবে। ধরুন, একটি ডেভেলপার দল git ব্যবহার করে একটি প্রোজেক্ট করছে। হঠাৎ একজনের মনে হলো user টেবিল এ নতুন একটি ফিল্ড লাগবে যার নাম active এবং boolean টাইপ। এবং তিনি migration ব্যবহার করে টেবিলের পরিবর্তন টা করলেন ও git এ পাঠিয়ে দিলেন। অন্যান্য সদস্যরা git merge করলেন এবং দেখলেন যে নতুন `migration` আছে এবং উনি লারাভেলকে বলেন `migrate` কর, জাদুর মতো তার ডাটাবেজও আপডেট হয়ে গেল। আগের মতো আর `newdbstructure.sql` মত ফাইল গুলো আর চালা চালি করা লাগবে না।

মাইগ্রেশন নিয়ে আরও বিস্তারিত আলোচনা করবো কিন্তু তার আগে আসেন আমরা আমাদের লারাভেল প্রোজেক্ট এর সাথে একটি ডাটাবেজ এর সংযোগ দেই।

সাধারণ ভাবে আমরা সবাইই কম বেশি mySQL ব্যবহার করি তাই আমার উদাহরনে আমি mySQLই ব্যবহার করলাম।

আসুন যার যার এনভায়রনমেন্ট আনুসারে mydb নামে একটি ডাটাবেজ বানাই। ধরে নেই,

Database Host: 127.0.0.1

Database Name: mydb

User: root

Password: secret

এই বার আমাদের প্রোজেক্ট ফোল্ডার এর রুটে `.env` নামে ফাইলটা খুলে নিম্ন লিখিত অংশের মতো পরিবর্তন আনি।

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=mydb
DB_USERNAME=root
DB_PASSWORD=secret
```

আপনি নিশ্চয় বুজতে পারছেন এগুলো constant variable, কিন্তু কোথায় এগুলার ব্যবহার হয়??

মনে আছে আমরা ইন্সটলেশন অধ্যায়ে লারাভেল ৫.৩ ফাইল বিন্যাস দেখেছিলাম?

হুম.. তাহলে আসুন `/config/database.php` ফাইলটি খুলি।

বাহ সব পরিষ্কার তাই না? তাও একটি লাইন এর মানে দেখিঃ

```
'database' => env('DB_DATABASE', 'forge'),
```

`.env` ফাইল এর ভেতর যদি `DB_DATABASE` নামের ভারিয়ারবল এ কোনও value সেট করি তাহলে database হবে ওই value টি নয়তো database এর value হবে `forge`।

তাহলে আমাদের এপ এর সাথে ডাটাবেজ চলে আসলো!!!

এবার আসুন আমরা আবার মাইগ্রেশনে ফিরে আসি।

মাইগ্রেশনের ফাইল গুলি থাকে `/database/migrations` ফোল্ডার এ। ফোল্ডারটি খুললে আমরা নিচের ফাইলটির মতো একটি(আসলে লারাভেল আমাদের জন্য দুটি ফাইল আগেই তৈরি করে রাখে) দেখতে পাবো।

```
2014_10_12_000000_create_users_table.php
```

সংখ্যা গুলো দিয়ে ফাইলটি কবে ও কখন তৈরি করা হয়েছে সেটা দেখায়, পরের অংশে মাইগ্রেশনটিতে কি করা হয়েছে সেটা থাকে। নামটি পরলেই আমরা বুজতে পারিঃ এটা দিয়ে users নামে একটি টেবিল তৈরি করা হয়েছে। এবং এটি একটি PHP ফাইল, হা হা হা।

আসুন ফাইলটি খুলি ও একটু বুঝে নেই কারণ কিছু দিনের মধ্যেই আমরা এরকম অনেক ফাইল বানাবো।

```
<?php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('users');
    }
}
```

প্রথমত Blueprint ও Migration নামের দুটি ক্লাসকে ব্যবহার করা হবে। Migration নামের ক্লাস কে এক্সটেন্ড করে CreateUsersTable নামের একটি ক্লাস লেখা হয়েছে।

এই ক্লাসে আমরা দুটি মেথড দেখতে পাই, একটি up অন্যটি down। **up** মেথড দিয়ে আমরা কোনও একটি কাজ করবো যেমন এখানে টেবিল তৈরি করা হয়েছে। এই মেথড ব্যবহার করে আমরা কোন নতুন ফিল্ড একটি টেবিলে যোগ করতে পারি, কখনো নাম পরিবর্তন করতে পারি বা অন্য কোনও পরিবর্তনও করতে পারি। আর **down** মেথড এ আমরা up মেথডে যে পরিবর্তন টা করেছি সেটা কিভাবে undo করা যায় সেই প্রসেসটা বলে দিবে। এখানে একটি টেবিল বানানো হয়েছিল তাই down মেথডে এটাকে উধাও করে দেবার প্রসেসটা বলা হয়েছে।

up মেথড এর ভেতর লারাভেল এর স্কিমা(Schema) ফ্যাসাদ(Facade) ব্যবহার করে টেবিলটি বানানো হয়েছে।

চলুন না আমরাও একটি টেবিল এর স্কিমা(**Schema**) বানিয়ে ফেলিঃ

ভয় পেলেন নাকি যে আবার কত গুলো কোড লেখা লাগবে? ভয় নেই, লারাভেল আমাদের জন্য শক্তিশালী একটি CLI(Command Line Interface) দিয়েছে। আসুন ব্যবহার করি। Terminal এ আমাদের প্রজেক্ট এর ভেতর ঢুকে নিচের কমান্ড টি run করাইঃ

```
php artisan make:migration create_posts_table --create=posts
```

নিশ্চয় দেখবেন

```
Created Migration: 2016_04_18_201310_create_posts_table
```


তবে প্রথমে দিন ও সময় টি আপনাত হবে।

এবার আপনার Code Editor এ এই `/database/migrations` ডিরেক্টরি খুলুন ও নতুন তৈরি করা migration টি দেখুন(প্রয়োজনে একবার path টি refresh করে নিন)

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('posts');
    }
}
```

আমরা আগেই সাধারণ একটি মাইগ্রেশনের ব্যাখ্যা পড়েছি, তাই সরাসরি up মেথড এ চলে আসি।

up মেথড এর ভিতরের অংশটুকু কে যদি আমি বাংলা ভাষায় বলার চেষ্টা করি তাহলেঃ Schema ফ্যাসাদ, টেবিল তৈরি(create) করো যার নাম posts এবং টেবিলটির গঠনটি(Blueprint) এরকম যে এর ভিতর একটি id ফিল্ড থাকবে যেটা auto increment হবে এবং timestamps() এর মাধ্যমে আরও দুটি ফিল্ড বানাও created_at ও updated_at নামে যা দিন ও সময় সংরক্ষণ করবে।

সাধারনত একটি টেবিলে একটি ID লাগে এবং ওই রেকর্ডটি কবে তৈরি ও পরিবর্তন হয়েছে সেইটাও জানতে হয়, তাই লারাভেল আগেই আমাদের জন্য লিখে দিয়েছে। কিন্তু এগুলো আমাদের প্রয়োজন না হলে মুছে ফেলতে পারি। যাক, আমাদের কাল্পনিক posts টেবিলটায় আরও কিছু ফিল্ড/কলাম(Column) যোগ করিঃ

```
Schema::create('posts', function (Blueprint $table) {
    $table->increments('id');
    $table->string('title', 255);
    $table->text('content');
    $table->string('slug')->nullable();
    $table->tinyInteger('status')->default(1);
    $table->timestamps();
});
```

বুজতেই পারছেন আমি title নামের string টাইপের ২৫৫ অক্ষরের মধ্যে সীমাবদ্ধ একটি column যোগ করেছি। এছাড়া content, slug ও status নামের আরও তিনটি column যোগ করেছি। আশা করি এগুলার টাইপটি বুজতে পেরেছেন। আমি এই কলামগুলো তৈরি করার জন্য যে মেথড গুলো ব্যবহার করেছি সেগুলোকে বলে কলাম মেথড(Column methods)।

আপনাদের সুবিধার্থে নিচে কলাম মেথড গুলোর লিস্টটি লারাভেল ৫.৪ ডকুমেন্টেশন থেকে কপি করে আনলাম।

Command	Description
<code>\$table->bigIncrements('id');</code>	Incrementing ID (primary key) using a "UNSIGNED BIG INTEGER" equivalent.
<code>\$table->bigInteger('votes');</code>	BIGINT equivalent for the database.
<code>\$table->binary('data');</code>	BLOB equivalent for the database.
<code>\$table->boolean('confirmed');</code>	BOOLEAN equivalent for the database.
<code>\$table->char('name', 4);</code>	CHAR equivalent with a length.
<code>\$table->date('created_at');</code>	DATE equivalent for the database.
<code>\$table->dateTime('created_at');</code>	DATETIME equivalent for the database.
<code>\$table->dateTimeTz('created_at');</code>	DATETIME (with timezone) equivalent for the database.
<code>\$table->decimal('amount', 5, 2);</code>	DECIMAL equivalent with a precision and scale.
<code>\$table->double('column', 15, 8);</code>	DOUBLE equivalent with precision, 15 digits in total and 8 after the decimal point.
<code>\$table->enum('choices', ['foo', 'bar']);</code>	ENUM equivalent for the database.
<code>\$table->float('amount', 8, 2);</code>	FLOAT equivalent for the database, 8 digits in total and 2 after the decimal point.
<code>\$table->increments('id');</code>	Incrementing ID (primary key) using a "UNSIGNED INTEGER" equivalent.
<code>\$table->integer('votes');</code>	INTEGER equivalent for the database.
<code>\$table->ipAddress('visitor');</code>	IP address equivalent for the database.
<code>\$table->json('options');</code>	JSON equivalent for the database.

<code>\$table->jsonb('options');</code>	JSONB equivalent for the database.
<code>\$table->longText('description');</code>	LONGTEXT equivalent for the database.
<code>\$table->macAddress('device');</code>	MAC address equivalent for the database.
<code>\$table->mediumIncrements('id');</code>	Incrementing ID (primary key) using a "UNSIGNED MEDIUM INTEGER" equivalent.
<code>\$table->mediumInteger('numbers');</code>	MEDIUMINT equivalent for the database.
<code>\$table->mediumText('description');</code>	MEDIUMTEXT equivalent for the database.
<code>\$table->morphs('taggable');</code>	Adds unsigned INTEGER <code>taggable_id</code> and STRING <code>taggable_type</code> .
<code>\$table->nullableMorphs('taggable');</code>	Nullable versions of the <code>morphs()</code> columns.
<code>\$table->nullableTimestamps();</code>	Nullable versions of the <code>timestamps()</code> columns.
<code>\$table->rememberToken();</code>	Adds <code>remember_token</code> as VARCHAR(100) NULL.
<code>\$table->smallIncrements('id');</code>	Incrementing ID (primary key) using a "UNSIGNED SMALL INTEGER" equivalent.
<code>\$table->smallInteger('votes');</code>	SMALLINT equivalent for the database.
<code>\$table->softDeletes();</code>	Adds nullable <code>deleted_at</code> column for soft deletes.
<code>\$table->string('email');</code>	VARCHAR equivalent column.
<code>\$table->string('name', 100);</code>	VARCHAR equivalent with a length.
<code>\$table->text('description');</code>	TEXT equivalent for the database.
<code>\$table->time('sunrise');</code>	TIME equivalent for the database.
<code>\$table->timeTz('sunrise');</code>	TIME (with timezone) equivalent for the database.
<code>\$table->tinyInteger('numbers');</code>	TINYINT equivalent for the database.
<code>\$table->timestamp('added_on');</code>	TIMESTAMP equivalent for the database.
<code>\$table->timestampTz('added_on');</code>	TIMESTAMP (with timezone) equivalent for the database.
<code>\$table->timestamps();</code>	Adds nullable <code>created_at</code> and <code>updated_at</code> columns.
<code>\$table->timestampsTz();</code>	Adds nullable <code>created_at</code> and <code>updated_at</code> (with timezone) columns.
<code>\$table->unsignedBigInteger('votes');</code>	Unsigned BIGINT equivalent for the database.
<code>\$table->unsignedInteger('votes');</code>	Unsigned INT equivalent for the database.
<code>\$table->unsignedMediumInteger('votes');</code>	Unsigned MEDIUMINT equivalent for the database.

<code>\$table->unsignedSmallInteger('votes');</code>	Unsigned SMALLINT equivalent for the database.
<code>\$table->unsignedTinyInteger('votes');</code>	Unsigned TINYINT equivalent for the database.
<code>\$table->uuid('id');</code>	UUID equivalent for the database.

আসুন দেখি না করে আমাদের বানানো এবং আগে থাকা(যদি থাকে) মাইগ্রেশনগুলোকে দিয়ে ডাটাবেজে টেবিলগুলি বানিয়ে ফেলি। Terminal এ নিচের কমান্ডটি রান করাইঃ

```
php artisan migrate
```

এবং বলছে যে মাইগ্রেশন হয়ে গেছে। দেখি না করে আপনার ডাটাবেজটি দেখুন। আপনার মাইগ্রেশনগুলোর টেবিল ছাড়াও আরও একটি টেবিল আছে যার নাম migrations. এটি লারাভেল নিজে ব্যবহার করে।

সর্বশেষ মাইগ্রেশনটি **undo** করতেঃ

```
php artisan migrate:rollback
```

আপনার কোনও মাইগ্রেশনে যদি রিনেম অথবা ড্রপ থাকে তাহলে উপরের command টি error দিবে। এই জন্য, doctrine/dbal নামক একটি প্যাকেজ আপনার প্রোজেক্ট এ যোগ করতে হবে। আমরা কিন্তু [ব্রেড টেমপ্লেটিং](#) অধ্যায়ে নতুন প্যাকেজ যোগ করেছিলাম। এই প্যাকেজটি দুই ভাবে যোগ করতে পারেন

প্রথমতঃ Terminal এ শুধু নিচের কমান্ডটি রান করুনঃ

```
composer require doctrine/dbal
```

দ্বিতীয়তঃ composer.json ফাইলটি খুলুন ও require অংশের শেষে `"doctrine/dbal": "^2.5"` যোগ করুন। তাহলে অনেকটা নিচের মতো দেখাবেঃ

```
"require": {
    "php": ">=5.5.9",
    "laravel/framework": "5.2.*",
    "laravelcollective/html": "5.2.*",
    "doctrine/dbal": "^2.5"
},
```

এবং নিচের কমান্ডটি রান করুনঃ

```
composer update
```

এটা আমাদের সব প্যাকেজ গুলোকে update, প্রয়োজনে add করে নিবে।

কখন আমাদের **rollback** করার প্রয়োজন হতে পারে?

যেকোনো সময়। কিন্তু আসুন একটি পরিস্থিতি এর কথা চিন্তা করি, মনে করি আমরা এই উপরের মাইগ্রেশনটাই মাইগ্রেট করার পর মনে হলোঃ আহাঃ একটি column নামের বানান ভুল হয়েছে বা আরও একটি ফিল্ড লাগতো তখন আবার নতুন মাইগ্রেশন না লিখে rollback করি, মাইগ্রেশনটাকে প্রয়োজন মতো লিখে নেই এবং আবার মাইগ্রেট কমান্ড রান করাই। কিন্তু প্রোজেক্টটি যদি লাইভ সার্ভারে থাকে তখন হয়ত আমরা কাজটি অন্য ভাবে করবো।

সব টেবিল গুলো মুছে ফেলতে

কাজের সময় এমন হয় যে অনেক আজে বাজে ডাটা ডাটাবেজ ভরে ফেলে বা অন্য কোনও কারণেও মনে হতে পারে সব টেবিল গুলি মুছে ফেলি। তখন নিচের কমান্ডটি রান করাই

```
php artisan migrate:reset
```

এখন পুরা ডাটাবেজ ক্লিয়ার হয়ে যাবে এবং আবার যদি মাইগ্রেট কমান্ড রান করাই তাহলে একে বারে ফ্রেশ একটি ডাটাবেজ পাবো।

একবারে সব টেবিল মুছে ফেলে নতুন করে তৈরি করতে

এই কাজটিই তো আমরা একটু আগে দুই ধাপে করলাম!! হ্যাঁ, আসুন একটি কমান্ডেই কাজটি সেরে ফেলি।

```
php artisan migrate:refresh
```

কমান্ডটি প্রথমে সব মুছে ফেলে আবার তৈরি করে দিবে - ডাটাবেজটি refresh হয়ে যাবে।

আজকে এই পর্যন্তই। এর পরের চ্যাপ্টারে মডেল নিয়ে আলোচনা করা হবে।

মডেল

যদিও অধ্যায়টির নাম মডেল(**Model**) দিয়েছি, এটা আসলে এলোকোয়েন্ট মডেল(**Eloquent Model**) কিন্তু এটাকে শুধু এলোকোয়েন্ট না বলে এলোকোয়েন্ট ওআরএম মডেল(**Eloquent ORM Model**) বললে সঠিক ভাবে ইন্ডিকেট করা হয়। আর যখনই একবার এটাকে চিনে যাব তখন থেকে আমাদের লারাবেল বন্ধু মহলে শুধুই মডেল(**Model**) বলে ডাকবো। অনেকটা "মাসনুন ভাই" বা "হাসিন ভাই" এর মতো, একবার চিনে গেলে আর পুরা নামটা বলা লাগে না।

আসুন ব্যাপারগুলোকে একটু ভেঙ্গে ভেঙ্গে বুঝে নেইঃ

ওআরএম(ORM)

পুরোটা হলো Object Relational Mapping, যা এক ধরনের কায়দা ব্যবহার করে অবজেক্টের মধ্যে রিলেশন তৈরি করে, এই অবজেক্ট গুলো মূলত ডাটাবেজ এর অবজেক্ট।

এলোকোয়েন্ট(Eloquent)

এলোকোয়েন্ট(Eloquent) হলো একটি ওআরএম(ORM) এর নাম, যা একটি একটিভ রেকর্ড(Active Record) এর প্রয়োগ(Implementations)। যেটা লারাবেল এর জন্যই তৈরি করা হয়েছে। এটা অন্যান্য ওআরএম থেকে বেশ শক্তিশালী ও বুদ্ধিমান। ডাটাবেজ নিয়ে কাজ করার সময় আমরা এর নানান কারিশমার সাথে পরিচিত হবো।

মডেল(Model)

মডেল হলো একধরনের ক্লাস যার প্রতিটি অবজেক্ট এক একটি টেবিলের এক একটি রো বা রেকর্ড কে রিপ্রেজেন্ট করে। মনে করি আমাদের একটি টেবিল আছে যার নাম users. এবং যদি এর জন্য একটি মডেল বানাই তার নাম দিবো User. যা users টেবিল এর প্রতিটি রেকর্ড কে রিপ্রেজেন্ট করবে। এখানে একটি মজার নিয়ম আমরা ফলো করবো, টেবিল এর নাম বহুবচন(plural) ও মডেল এর নাম একবচন(singular)। তখন আমরা যে মডেলটি বানাবো, এলোকোয়েন্ট ঠিকই তার টেবিলটি ডাটাবেজ থেকে খুঁজে ম্যাপ করে নিবে। যদি এই নিয়মের অন্যথা হয় তখন মডেল বানানোর সময় টেবিলটির নামটি বলে দিতে হবে।

আমরা এলোকোয়েন্ট ওআরএম ব্যবহার করে টেবিলে Create, Edit, Delete, Select এবং আরও অনেক কিছুই করতে পারবো কোনও SQL statement না লিখেই। আপনি কি রিলেশনাল ডাটাবেজ এর কথা ভাবছেন? হ্যাঁ, সেটাও আমরা এই সিস্টেম এর মধ্যেই করে ফেলবো!!

করতে করতে শেখা

গত অধ্যায়ে আমরা একটি মাইগ্রেশন ও সেটাকে মাইগ্রেট করে একটি টেবিল বানিয়েছিলাম। আসুন আজ আবারও মাইগ্রেশন নিয়ে একটু অনুশীলন করে নেই।

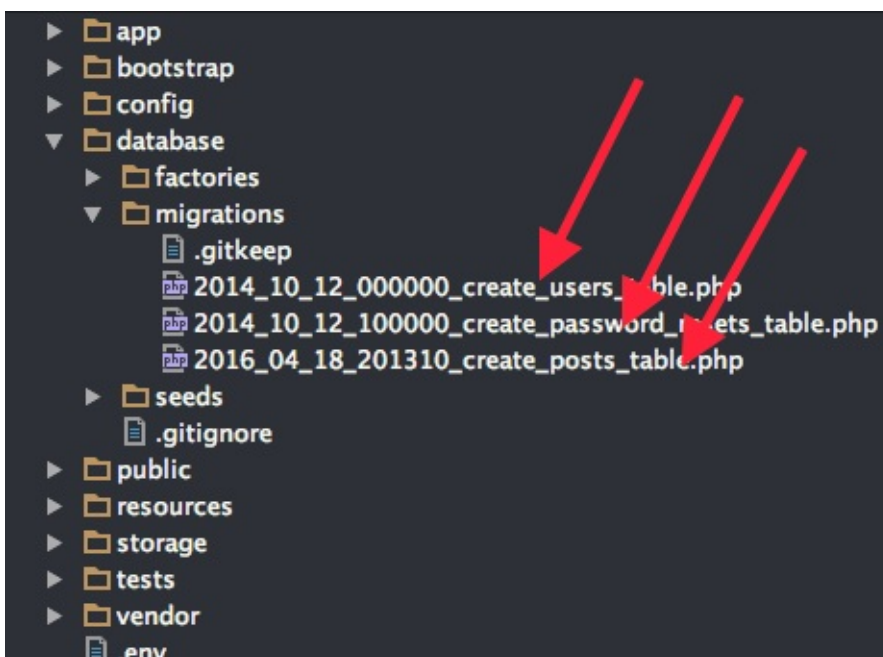
প্রস্তুতিঃ

Terminal থেকে আমাদের প্রোজেক্ট ফোল্ডারে ঢুকে নিচের কমান্ডটি রান করাইঃ

```
php artisan migrate:reset
```

বলুনতো কি হলো? মনে না আসলে [মাইগ্রেশন](#) অধ্যায়টি আরেকবার দেখে আসুন please.

/database/migrations ডাইরেক্টরি খুলে সব মাইগ্রেশন ফাইলগুলি মুছে ফেলি।



আপনার প্রজেক্টে কমবেশি ফাইল থাকতেই পারে দরকার না হলে সেগুলোও মুছে ফেলতে পারেন।

এবার নতুন করে posts নামের টেবিলের জন্য একটি মাইগ্রেশন তৈরি করি লারাভেলের আর্টিসান কমান্ড এর মাধ্যমে

```
php artisan make:migration 'create_posts_table' --create=posts
```

মাইগ্রেশনটি খুলুন ও লক্ষ্য করুন আপনার ক্লাসের নামটা লারাভেল কি সুন্দর ভাবে লিখেছে। এবার প্রয়োজনীয় অংশ আপডেট করে নিচের মতো বানাইঃ

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePostsTable extends Migration
{
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title', 255);
            $table->text('content');
            $table->tinyInteger('status')->default(0);
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('posts');
    }
}
```

এবার মাইগ্রেট করি

```
php artisan migrate
```

তাহলে নিচের মতো একটি টেবিল পেলাম

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action			
<input type="checkbox"/>	1 id	int(10)		UNSIGNED	No	None	AUTO_INCREMENT	Change	Drop	Primary	Unique Index More
<input type="checkbox"/>	2 title	varchar(255)	utf8_unicode_ci		No	None		Change	Drop	Primary	Unique Index More
<input type="checkbox"/>	3 content	text	utf8_unicode_ci		No	None		Change	Drop	Primary	Unique Index More
<input type="checkbox"/>	4 status	tinyint(4)			No	1		Change	Drop	Primary	Unique Index More
<input type="checkbox"/>	5 created_at	timestamp			Yes	NULL		Change	Drop	Primary	Unique Index More
<input type="checkbox"/>	6 updated_at	timestamp			Yes	NULL		Change	Drop	Primary	Unique Index More

মডেল তৈরিঃ

এখানে টেবিল এর নাম posts তাহলে মডেলের নাম হবে Post । আরটিসান কমান্ড এর মাধ্যমে কাজটি সেবে ফেলিঃ

```
php artisan make:model Post
```

মডেল গুলো সাধারণ ভাবে app ডিরেক্টরির রুটেই থাকে, মডেলটি খুললে আমরা এরকম পাবোঃ


```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    //
}
```

এই Post ক্লাসটির মধ্যে আমরা নানা রকম property ও method এর মাধ্যমে মডেলটিকে define করবো।

কাস্টম টেবিল এর নাম

এমন যদি হয় যে আপনার টেবিলটির নাম বহুবচন-একবচন নিয়মের বাইরে বা যেকোনো কারণেই আপনি আপনার মতো করে মডেলের নাম ও টেবিলের নাম ঠিক করলেন। তখন নিচের মতো করে করতে হবে।

```
class Post extends Model
{
    protected $table = 'custom_posts_table';
}
```

তাহলে ইলোকয়েন্ট এই Post মডেলের সাথে custom_posts_table এর সংযোগ করে নিবে।

টেবিল এর কাস্টম প্রাইমারি-কি(primary key)

সাধারণ ভাবে ইলোকয়েন্ট একটি টেবিল এর প্রাইমারি-কি(primary key) id নামক কলামটাকেই ধরে নেয়, তাই মাইগ্রেশন তৈরি করার সময় primary key এর নাম অন্য রকম দিলে অবশ্যই মডেলে নিচের মতো জানিয়ে দিতে হবে।

```
class Post extends Model
{
    protected $primaryKey = 'post_id';
}
```

টেবিল এ **timestamps** না চাইলে

মাইগ্রেশিওন তৈরি করার সময় আমরা দেখেছি যে id ও timestamps(created_at ও updated_at দুটি কলাম) বানিয়ে দেয় এবং এগুলো মডেলেও আশা করে। তাই এটা না চাইলে, নিচের মতো করে জানাতে হবে।

```
class Post extends Model
{
    public $timestamps = false;
}
```

Mass-Assignment Vulnerability

জিনিসটা কি জানার আগে আমরা একটা পরিস্থিতি কল্পনা করি। মনে করি, আমাদের একটি form আছে যেখান থেকে লেখকরা পোস্ট তৈরি করে সাবমিট করবে, এবং একটি মেথড এর সাহায্যে এক লাইনেই আমরা ডাটাবেজে সেভ করে নিব। পরে অ্যাডমিন পোস্টটি চেক করে, status 1 করে দিলেই পোস্টটি সাইট এ দেখাবে।

কিন্তু কোনও চালাক ডেভেলপার পোস্ট তৈরি করে সাবমিট করার সময় ইমপেক্ট এলিমেন্ট করে পোস্ট রিকুয়েস্ট এর সাথে status=1 পাঠিয়ে দিলো আর সেজন্য সাথে সাথেই পোস্টটি লাইভ হয়ে যাবে - তাতে যাচ্ছে তাই জাইই থাকুক না কেন। কি এটা একটা দুর্বলতা নয়?

আর এই দুর্বলতাকেই বলে ম্যাস অ্যাসাইনমেন্ট ভলনারাবিলিটি।

এই দুর্বলতা কাটাতে লারাভেল মডেলের জন্য দুটি প্রপার্টি দিয়েছেঃ fillable ও guarded। এদের যে কোনও একটিকে আমাদের মডেলে ব্যবহার করলেই এই দুর্বলতা কাটাতে পারবো, নিচের কোড টি দেখুন, আমাদের posts টেবিলের জন্য লেখা।

```
class Post extends Model
{
    protected $guarded = ['status'];
    protected $fillable = ['title', 'content'];
}
```

এখন কেউ ওই চালাকি করলেই লারাভেল "ম্যাস অ্যাসাইনমেন্ট এক্সসেপসন" নামক এরর দিবে।

Tinker নিয়ে কিছু মজা!

টিঙ্কার হলো লারাভেল এর একটি CLI, যেটা দিয়ে আমরা আমাদের এপ্লিকেশন এর সাথে ইন্টারাক্ট করতে পারবো। আসুন tinker দিয়ে আমাদের তৈরি মডেলটাকে একটু টেস্ট করার সাথে সাথে কিছু জিনিস শিখে নেই। আমাদের Terminal এ নিচের আরটিসান কমান্ডটি দেই

```
php artisan tinker
```



```
robert lara-book $ php artisan tinker
Psy Shell v0.7.2 (PHP 5.5.31 - cli) by Justin Hileman
>>>
```

দেখতে এরকমই। আপনাদের সুবিধার্থে বেশ কিছু স্ক্রিনশট দিবো। এবার ওখানে `exit` লিখুন, দেখবেন আপনাকে goodbye জানিয়েছে। আবার tinker শেল এ ফিরে আসুন কারণ আমরা এখন আমাদের মডেল নিয়ে এখানে কিছু শিখব।

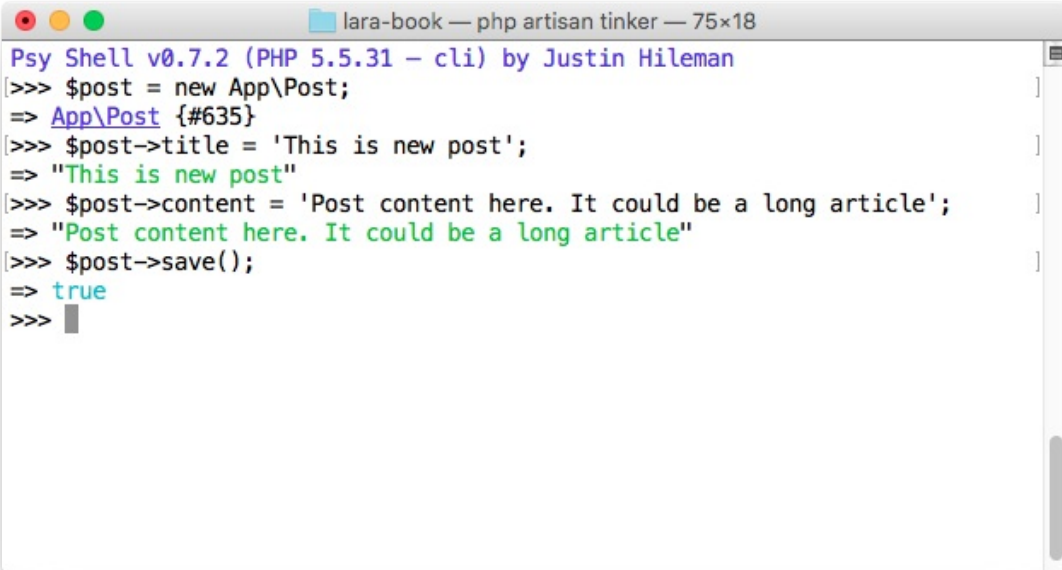
এবার আমাদের App নেম স্পেস এর Post মডেলের একটি অবজেক্ট তৈরি করি এভাবেঃ

```
$post = new App\Post;
```

তারপর নিচের মতো title ও content এর ভেলু দিয়ে অবজেক্ট টির save method কে কল করি।

```
$post->title = 'This is new post';
$post->content = 'Post content here. It could be a long article';
$post->save();
```

tinker এ এমন দেখাবে



```

Psy Shell v0.7.2 (PHP 5.5.31 - cli) by Justin Hileman
[>>> $post = new App\Post;
=> App\Post {#635}
[>>> $post->title = 'This is new post';
=> "This is new post"
[>>> $post->content = 'Post content here. It could be a long article';
=> "Post content here. It could be a long article"
[>>> $post->save();
=> true
[>>>
  
```

করেছেন? এবার ডাটাবেজটা একটু খুলে দেখুন।

	id	title	content	status	created_at	updated_at
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	1	This is new post	Post content here. It could be a long article	1	2016-04-22 04:23:18	2016-04-22 04:23:18

আসুন এবার একটি array পাস করে একবারেই আরেকটি পোস্ট বানাই।

```

$post = App\Post::create(['title'=>'2nd awesome post', 'content'=>'Mind blowing content'])
  
```

এখন যদি আপনি ম্যাস অ্যাসাইনমেন্ট ভলনারাবিলিটি ঠিক করার জন্য আপনার মডেল ঠিক না করেন তবে নিশ্চয় এরর পাবেন।

এবার সব পোস্ট গুলো দেখার জন্য নিচের কমান্ডটি দিন।

```
App\Post::all()->toArray();
```

এবার id দিয়ে একটি পোস্ট খুঁজে বের করি

```
$post = App\Post::find(2);
```

উপরের কমান্ড গুলো Terminal এ এমন

```
lara-book — php artisan tinker — 96x37
>>> $post = App\Post::create(['title'=>'2nd awesome post', 'content'=>'Mind blowing content']);
=> App\Post {#650
  title: "2nd awesome post",
  content: "Mind blowing content",
  updated_at: "2016-04-22 05:01:07",
  created_at: "2016-04-22 05:01:07",
  id: 2,
}
>>> App\Post::all()->toArray();
=> [
  [
    "id" => 1,
    "title" => "This is new post",
    "content" => "Post content here. It could be a long article",
    "status" => 0,
    "created_at" => "2016-04-22 04:41:34",
    "updated_at" => "2016-04-22 04:41:34",
  ],
  [
    "id" => 2,
    "title" => "2nd awesome post",
    "content" => "Mind blowing content",
    "status" => 0,
    "created_at" => "2016-04-22 05:01:07",
    "updated_at" => "2016-04-22 05:01:07",
  ],
]
>>> $post = App\Post::find(2);
=> App\Post {#635
  id: 2,
  title: "2nd awesome post",
  content: "Mind blowing content",
  status: 0,
  created_at: "2016-04-22 05:01:07",
  updated_at: "2016-04-22 05:01:07",
}
>>>
```

টিক্কার নিয়ে বেশ মজা হলো এরপর আমরা আমাদের আগের অধ্যায়ের জ্ঞান ব্যবহার করে ফর্ম থেকেই পোস্ট তৈরি করবো।

পরবর্তী অধ্যায় "মধ্যবর্তী প্রজেক্ট - ১" এ আমরা মডেল এর ব্যবহার দেখবো।

মধ্যবর্তী প্রোজেক্ট - ১

পূর্বশর্ত

আপনাকে অবশ্যই আগের অধ্যায় গুলো সম্পর্কে ধারণা থাকতে হবে।

বিষয়

আমাদের প্রধান বিষয় হবে গত অধ্যায়ে যে মডেল শিখলাম তার ব্যবহার। দ্বিতীয় বিষয় হবে আগে যে অধ্যায় গুলো শেষ করেছি সেগুলোকে একসাথে করে সবার মধ্যে সংযোগ স্থাপন করানো।

পরিকল্পনা

একটি খুঁবিই সাধারণ ব্লগ। টুইটার বুটস্টারপ ব্যবহার করবো। শুরুতে, আমরা লারাডেল এনডায়রনমেন্ট তৈরি করবো, লারাডেল ইন্সটল করবো। যাতে একজন নতুন ব্যবহারকারি সহজ বোধ করে।

ব্লড টেমপ্লেট ও ডিউঃ

ইনডেক্স পেইজঃ যেখানে সব পোস্ট গুলোর লিস্ট থাকবে এক্সসারপ্ট সহ। পোস্ট এর টাইটেল এ মূল পোস্ট এর লিঙ্ক থাকবে।

একক পোস্ট পেইজঃ যেখানে একটি পোস্ট এর টাইটেল ও পুরো কন্টেন্ট দেখাবে।

এডমিন ইনডেক্স পেইজঃঃ যেখানে সব পোস্ট গুলোর লিস্ট থাকবে এডিট/ডিলিট/লাইভ করার জন্য।

এডমিন নতুন পোস্ট পেইজঃ নতুন পোস্ট তৈরি করার পেইজ।

এডমিন পোস্ট এডিট পেইজঃ এডিট করার জন্য।

মেনুঃ টুইটার বুটস্টারপ এর সাধারণ মেনু হবে।

কন্ট্রোলার ও মডেলঃ

কন্ট্রোলার এর মাধ্যমে আমরা ডিউ তে ডাটা পাঠাবো ও মডেল ব্যবহার করে CRUD করবো।

মাইগ্রেশনঃ

একটি টেবিল এর জন্য মাইগ্রেশন তৈরি, টেবিল তৈরি করতে হবে।

এলোকোয়েন্টঃ

এলোকোয়েন্ট এর ব্যবহার।

রিকোয়েন্ট ও ইনপুট ফ্যাসাদঃ

এটি নতুন এবং প্রোজেক্ট করার সময়ই শিখে নিবো।

উল্লেখ্য

এই প্রোজেক্টটি শুধুমাত্র শেখার জন্য, মূলত এই বই এর পূর্ববর্তী অধ্যায় গুলোকে অনুশীলন ও আলাদা আলাদা বিষয় গুলো কিভাবে এক সাথে কাজ করে তা জানার জন্য।

পরিবেশ তৈরি

যারা লারাভেল এর প্রতি আগ্রহী তারা অবশ্যই PHP জানেন এবং কিছুটা হলেও কাজ করেছেন। তাই আশা করছি সকলেই LAMP(Linux, Apache, mySQL ও PHP) এর সাথে পরিচিত। তাই সাধারণ কিছু আলোচনা ছাড়া তেমন বিস্তারিত কিছু করবো না।

লারাভেল হোমস্টিড

লারাভেল নিজে তার ডেভেলপমেন্ট পরিবেশ হিসাবে লারাভেল হোমস্টিড কেই সবচেয়ে বেশি পছন্দ করে। তাই নিচে কিছু লিঙ্ক দিলাম হয়তো কাজে লাগবে।

[লারাভেল এর নিজস্ব ডকুমেন্টেশন](#)

[নুরুজ্জামান মিলন এর হোমস্টিড বিষয়ে বাংলা আর্টিকেল](#)

[নাহিদ বিন আজহারের এর হোমস্টিড বিষয়ে বাংলা আর্টিকেল](#)

[হাসিন হায়দার এর ভ্যাগর্যান্ট পরিচিতি - Youtube চ্যানেল](#)

LAMP Stack, MAMP, XAMP, WAMP

আপনি লিনাক্স বা ম্যাক এ সরাসরি LAMP Stack ইন্সটল করে নিতে পারেন অথবা আপনার OS অনুসারে MAMP, XAMP, WAMP বা অন্য কোনও কিছু ব্যবহার করতে পারেন যতক্ষণ পর্যন্ত নিচের প্রয়োজনীয় লিস্ট এর সব থাকে।

- PHP >= 5.5.9
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension

[লিঙ্কঃ বাংলায় পিএইচপি কোর্সের ইনস্টলেশন অধ্যায়](#)

ভারচুয়াল হোস্ট

আমাদের প্রোজেক্ট আনুশিলন করার সময় অবশ্যই একটি ভারচুয়াল হোস্ট তৈরি করে নিন, যেমন project.one যার ডকুমেন্ট রুট এর শেষ অংশ `project.one/public`। আশা করি সমস্যা হবে না যদি হয় তবে নিচের **Comment** এ জানান, আমরা এই অংশটি সংযোজিত করব।

লারাভেল ইন্সটল

প্রস্তুতি

আসুন নিচের কমান্ডটির মাধ্যমে আমাদের প্রোজেক্ট project.one ইন্সটল করি।

```
composer create-project laravel/laravel project.one
```

ইন্সটল হবার পর ভার্সন চেক করলাম, আপনি নিশ্চয় আরও আপডেটএড ভার্সন পাবেন।

```
[robert project.one $ php artisan -v  
Laravel Framework version 5.2.30]
```

ভার্চুয়াল হস্ট কনফিগার করার জন্য আমার host(OS X path: /private/etc/hosts) ফাইলে নিচের অংশ

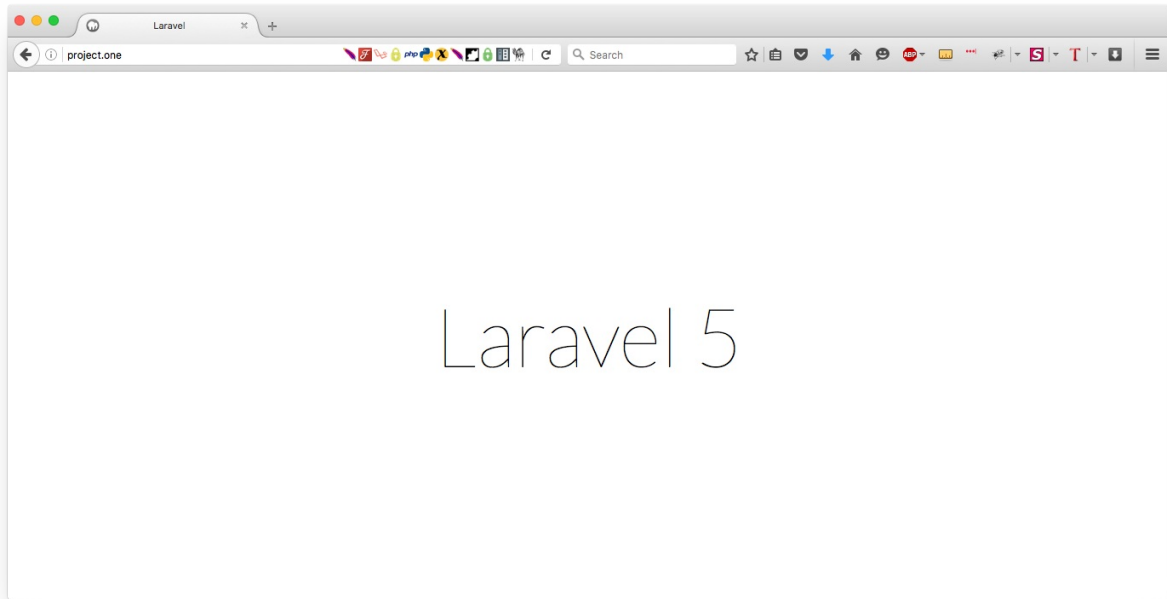
```
#Project 1 - howtocode.com.bd  
127.0.0.1 project.one
```

ও httpd-vhosts.conf(OS X path: /Applications/MAMP/conf/apache/extra/httpd-vhosts.conf) ফাইলে নিচের অংশ

```
<VirtualHost *:80>  
    DocumentRoot "/Applications/MAMP/htdocs/project.one/public"  
    ServerName project.one  
</VirtualHost>
```

আপনি নিশ্চয় বুজতে পারছেন এই প্রোজেক্টটি করার সময় আমি MAMP ব্যবহার করছি।

এবার এড্রেস বার এ `http://project.one` দিয়ে দেখি



এই প্রোজেক্টে আমরা ব্লেড টেমপ্লেট দিয়ে Form তৈরি করবো তাই আসুন `laravelcollective/html` যোগ করে নেই আমাদের প্রোজেক্ট এ। প্রয়োজনে [ব্লেড টেমপ্লেটিংঃ HTML ও Forms](#) অংশটি আবেকবার দেখে নিন।

এবার একটি mySQL ডাটাবেজ তৈরি করে ফেলুন। ধরি,

Database Name: blog

server: 127.0.0.1 মানে আপনার লোকাল হোস্ট

User: root

Pass: root

তাহলে এবার `.env` ফাইলটি খুলুন ও নিচের মতো পরিবর্তন করুন।

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=blog
DB_USERNAME=root
DB_PASSWORD=root
```

বিশেষ করে ইউজার এর নাম ও পাসওয়ার্ড আপনার মতো করে দিন। এই কন্সট্যান্ট গুলো ব্যবহার হয়েছে `config/database.php` ফাইলে। লক্ষ্য করুন, আপনি যদি `.env` ফাইলে 'DB_CONNECTION' কন্সট্যান্টটি সেট না করে থাকেন তাহলে সাধারণ ভাবে লারাভেল mySQL ডাটাবেজকেই সেট করে নিবে।

এবার নিচের কমান্ডটি দেই

```
composer require doctrine/dbal
```

এ ব্যাপারে [মাইগ্রেশন](#) অধ্যায়ে আলোচনা হয়েছে।

মাইগ্রেশন ও post টেবিল তৈরি

টার্মিনালে আমাদের প্রোজেক্ট ডাইরেক্টরি তে থেকে নিচের কমান্ডটি দেইঃ

```
php artisan make:migration create_posts_table --create=posts
```

বিষয়ে [মাইগ্রেশন](#) অধ্যায়ে বিস্তারিত আলোচনা হয়েছে।

এবার `project.one/database/migrations` ভিতরের `xxx_xx_xx_XXXXXX_create_posts_table.php` ফাইলটি খুলি ও নিচের মতো পরিবর্তন করি।

```
class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title', 255);
            $table->text('content');
            $table->tinyInteger('status')->default(0);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('posts');
    }
}
```

এবার `project.one/database/migrations` ভিতরে দেখুন আরও দুটি মাইগ্রেশন ফাইল আছে, সেগুলো ডিলিট করে দিন কারণ এখুনি আমাদের ওগুলো লাগবে না। এবার নিচের কমান্ডটি দেইঃ

```
php artisan migrate
```

তাহলে আমাদের প্রজেক্টের জন্য প্রয়োজনীয় টেবিলটি তৈরি হয়ে গেল।

তাহলে আজ আমাদের প্রোজেক্ট শুরু হল।

কি ভাবে সোর্স কোড পাওয়া যাবে?

গিটহাব এর [project.one](https://github.com/project-one) রিপজিটোরিটি ফোর্ক করুন এবং আপনার সিস্টেমে ক্লোন করুন।

এই অধ্যায়ের সোর্স কোড পেতে

```
git checkout a522ab7ba
```

সর্বশেষ কমিট পর্যন্ত পেতে আবার নিচের কমান্ডটি দিন

```
git checkout master
```

সিড(Seed) ও ইনডেক্স পেইজ

প্রস্তুতি

যেহেতু এই প্রোজেক্টটি একটি blog এর মতো হবে তাই হোম বা ইনডেক্স পেইজ এ আমাদের পোস্ট গুলোর লিস্ট থাকাই স্বাভাবিক।

posts টেবিল এর জন্য Post মডেল তৈরিঃ

মডেল অধ্যায়ে আমার এই বিষয় নিয়ে বেশ আলোচনা করেছিলাম, চাইলে একবার ঘুরে আসতে পারেন।

আসুন নিচের কমান্ডটি দেইঃ

```
php artisan make:model Post
```

project.one/app/Post.php ফাইলটি আমরা পাবো, সাধারণ নিয়ম অনুসারে Post ক্লাসটি, গত অধ্যায়ে migrate করা posts টেবিলটাকেই গ্রহণ করবে। আসুন টেবিলটি দেখে নেইঃ

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action				
<input type="checkbox"/>	1 id	int(10)		UNSIGNED	No	None	AUTO_INCREMENT	Change	Drop	Primary	Unique	Index More
<input type="checkbox"/>	2 title	varchar(255)	utf8_unicode_ci		No	None		Change	Drop	Primary	Unique	Index More
<input type="checkbox"/>	3 content	text	utf8_unicode_ci		No	None		Change	Drop	Primary	Unique	Index More
<input type="checkbox"/>	4 status	tinyint(4)			No	0		Change	Drop	Primary	Unique	Index More
<input type="checkbox"/>	5 created_at	timestamp			Yes	NULL		Change	Drop	Primary	Unique	Index More
<input type="checkbox"/>	6 updated_at	timestamp			Yes	NULL		Change	Drop	Primary	Unique	Index More

এবার Post.php ফাইলটি খুলি

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    //
}
```

এবং নিচের মতো পরিবর্তন করি

```
class Post extends Model
{
    protected $guarded = ['status'];
    protected $fillable = ['title', 'content'];
}
```

অংশটি বোঝার জন্য [এইখানে দেখুন](#) ।

posts টেবিল এ কিছু রেকর্ড ইন্সার্ট করি ডেভেলপমেন্ট এর সুবিধার জন্য

আমরা আগেই [Tinker](#) ব্যবহার করে কিভাবে টেবিলে ডাটা ইন্সার্ট করা যায় তা দেখেছিলাম । আজ দেখবো Seeding ।

সিডিং(Seeding)

লারাভেল ডাটাবেজে টেস্ট ডাটা ইন্সার্ট করার জন্য সাধারণ একটি method যোগ করেছে, যার নাম সিডিং(Seeding) । সব সিড ক্লাস `database/seeds` ডাইরেক্টরিতে থাকে । সিড ক্লাস এর নাম আপনি যেকোনো কিছুই দিতে পারেন, কিন্তু কিছু রীতি মেনে নাম দেয়াই ভালো, যেমন আমাদের posts টেবিলের সিডার ক্লাস এর নাম হতে পারে `PostsTableSeeder` ।

আসুন তাহলে আমাদের সিডার বানিয়ে ফেলি নিচের কমান্ড দিয়েঃ

```
php artisan make:seeder PostsTableSeeder
```

আমরা নিশ্চয় `project.one/database/seeds/PostsTableSeeder.php` ফাইলটি পাবো ।

```
<?php

use Illuminate\Database\Seeder;

class PostsTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //
    }
}
```

আমাদের ফাইলটাকে পরিবর্তন করার আগে আসুন `project.one/database/seeds/DatabaseSeeder.php` ফাইল এ একটু পরিবর্তন আনি নিচের মতো।

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(PostsTableSeeder::class);
    }
}
```

এবার `project.one/database/seeds/PostsTableSeeder.php` ফাইলটাকে পরিবর্তন করি

```
public function run()
{
    DB::table('posts')->insert([
        'title' => 'Seeding Title',
        'content' => 'Seeding content'
    ]);
}
```

এবার নিচের কমান্ডটি দিন ও ডাটাবেজে টেবিলটির পরিবর্তন লক্ষ্য করুন।

```
php artisan db:seed
```

সতর্কতাঃ আপনি যদি নিচের মতো এরর দেখেন

```
[ReflectionException]
Class PostsTableSeeder does not exist
```

তাহলে নিচের কমান্ডটি দিন।

```
composer dump-autoload
```

আবার `db:seed` কমান্ডটি দিন ও ডাটাবেজে টেবিলটির পরিবর্তন লক্ষ্য করুন।

কিন্তু **Tinker** বা উপরের মতো করে টেস্ট ডাটা ডাটাবেজে ঢুকানো বড়ই কষ্টকর তাই লারাভেল টেস্ট ডাটার কারখানা বানিয়ে দিয়েছে, আসুন তার সাথে পরিচিত হই।

মডেল ফ্যাক্টরি(model factory) এর ব্যবহারঃ

`project.one/database/factories/ModelFactory.php` ফাইলটিতে নিচের মতো করে একটি ফ্যাক্টরি ডিফাইন করি।

```
$factory->define(App\Post::class, function (Faker\Generator $faker) {
    return [
        'title' => $faker->sentence(mt_rand(3, 10)),
        'content' => join("\n\n", $faker->paragraphs(mt_rand(3, 6))),
        'created_at' => $faker->dateTimeBetween('-1 month', '+3 days'),
    ];
});
```

`project.one/database/seeds/DatabaseSeeder.php` ফাইলটি আগের মতোই থাকুক।

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(PostsTableSeeder::class);
    }
}
```

এবার `project.one/database/seeds/PostsTableSeeder.php` ফাইলটাকে পরিবর্তন করি

```
<?php

use Illuminate\Database\Seeder;

class PostsTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        App\Post::truncate(); // delete all previous rows

        factory(App\Post::class, 10)->create(); // Create 10 rows
    }
}
```

এবার নিচের কমান্ডটি দিন ও ডাটাবেজে টেবিলটির পরিবর্তন লক্ষ্য করুন।

```
php artisan db:seed
```

আপনি এভাবে মডেল ফ্যাক্টরি ও ফেকার এর সাহায্যে সহজেই ইউজার, পাসওয়ার্ড, ঠিকানা, মানুষের নাম এবং আরও অনেক কিছুই তৈরি করতে পারেন।

ইনডেক্স পেইজ তৈরি

আমরা [ব্লড টেমপ্লেটিং](#) অধ্যায়ে দেখেছি কিভাবে ভিউ ও টেমপ্লেটিং করতে হয়। তাই সময় নষ্ট না করে নিচের মতো করে `project.one/resources/views` এর মধ্যে `master.blade.php` ফাইলটি তৈরি করি।

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>@yield('title')</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/boot
    {!! Html::style('css/style.css') !!}
</head>
<body>
@include('partials.navbar')
<div class="container">

  <div class="row">
    @yield('content')
  </div>
</div>

<script src="{!! asset('js/jquery-1.12.2.min.js') !!}"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></scrip
</body>
</html>
```

৬ নম্বর লাইনে CDN থেকে টুইটার বুটস্ট্রাপ লিঙ্ক করা হয়েছে। আসুন নিজেদের স্টাইল এর জন্য project.one/public/css পাথে style.css নামে একটি ফাইল তৈরি করি, যেটা ৭ নম্বর লাইনে লিঙ্ক করা হয়েছে। কিছু CSS যোগ করি এই ফাইলেঃ

```
.post-info span {
  background-color: #888;
  color: #fff;
  padding: 4px;
}
```

এবার project.one/public ফোল্ডারে js নামে আরেকটি ফোল্ডার করে jquery-1.12.2.min.js ডাউনলোড করে রাখি, যেটা ১৬ নং লাইনে লিঙ্ক করা হয়েছে। তার পরের লাইন টি আশা করি বুজতে পারছেন।

১০ নং লাইনে দেখুন navbar.blade.php নামে একটি ফাইল ইনক্লুড করা হয়েছে আসুন project.one/resources/views এর মধ্যে partials নামে একটি ফোল্ডার বানিয়ে ফাইলটি নিচের মতো বানিয়ে ফেলি।

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <!-- Brand and toggle get grouped for better mobile display -->
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed" data-toggle="collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="/">laravel.howtocode.com.bd @yield('action')</a>
    </div>

    <!-- Navbar Right -->
    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
      <ul class="nav navbar-nav navbar-right">
        <li class="active"><a href="/">Home</a></li>
        <li><a href="/about">About</a></li>
        <li class="dropdown">
          <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="butt
          <ul class="dropdown-menu" role="menu">
            <li><a href="/admin/posts/new">New Post</a></li>
            <li><a href="/admin/posts">Edit</a> </li>
          </ul>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

এটা টুইটার বুটস্ট্রাপের সাধারণ navbar, যেখানে লিঙ্ক গুলো তৈরি করেছি, পরে পেইজ গুলোও বানাবো।

এবার আসুন routes.php ফাইলে আমাদের হোমে রুট বানাই। নিচের মতো করে(এটা লারাভেল 5.2.31 ভার্সন)

```
<?php

/*
|-----
| Application Routes
|-----
|
| Here is where you can register all of the routes for an application.
| It's a breeze. Simply tell Laravel the URIs it should respond to
| and give it the controller to call when that URI is requested.
|
|*/

Route::get('/', 'PostsController@index');
```

এবার নিচের কমান্ড দিয়ে PostsController টি বানিয়ে ফেলি।

```
php artisan make:controller PostsController
```

ও নিচের মতো পরিবর্তন করি।

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

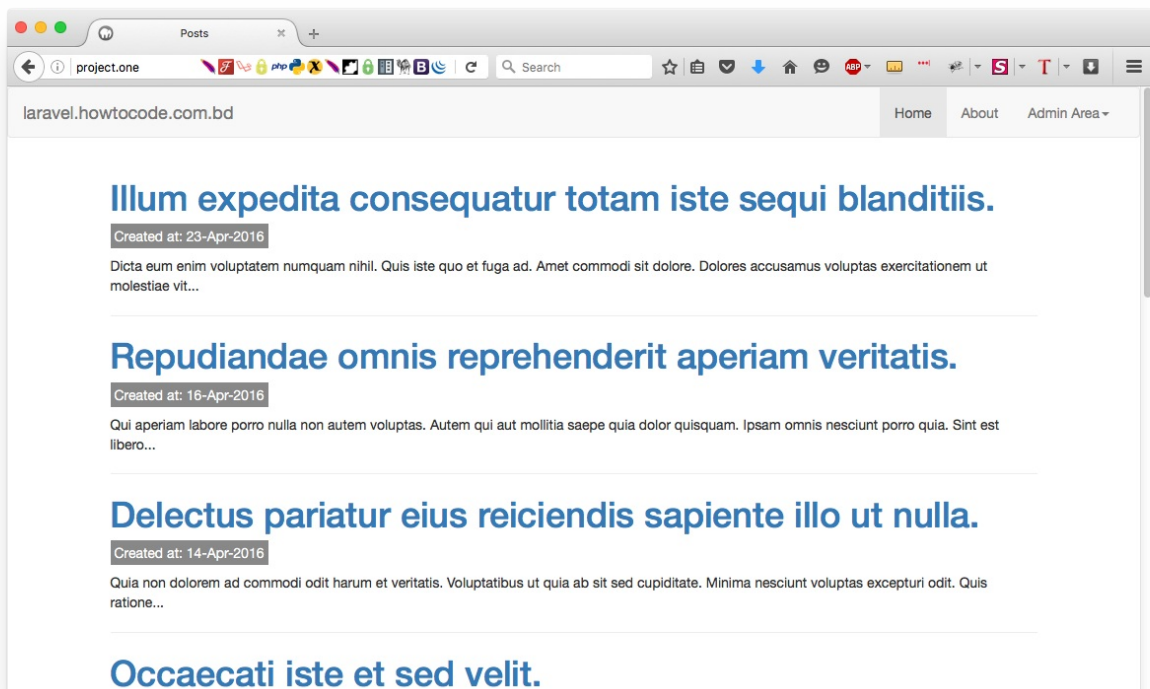
use App\Http\Requests;
use App\Post;

class PostsController extends Controller
{
    public function index(){
        $posts = Post::all();
        return view('posts.index', compact('posts'));
    }
}
```

এবার ভিউ তৈরি project.one/resources/views এর ভিতর posts নামে ফোল্ডার করে index.blade.php নামে নিচের মতো একটি ফাইল বানাই।

```
@extends('master')
@section('title', 'Posts')
@section('content')
    <div class="col-md-12">
        @if (count($posts))
            @foreach($posts as $post)
                <article>
                    <a href="#"><h1>{!! $post->title !!}</h1></a>
                    <div>
                        <p class="post-info"><span>Created at: {!! $post->created_at->format('d-m-Y H:i:s') !!}</span>
                    </div>
                    <div class="post-excerpt">
                        {!! str_limit($post->content, 150) !!}
                    </div>
                </article>
            <hr>
        @endforeach
    @else
        <h1>Hi, you've landed in our First Project!</h1>
        <h2>Sorry, We don't have any post right now.</h2>
    @endif
</div>
@endsection
```

এবার আমরা ব্রাউজারে আমাদের প্রোজেক্ট টি চেক করি, আশা করি নিচের মত দেখতে হবে, না হলে কमेंটে জানান দ্যা করে।



কি ভাবে সোর্স কোড পাওয়া যাবে?

গিটহাব এর [project.one](#) রিপজিটোরিটি ফোর্ক করুন এবং আপনার সিস্টেমে ক্লোন করুন।

এই অধ্যায়ের সোর্স কোড পেতে

```
git checkout f384e15
```

সর্বশেষ কমিট পর্যন্ত পেতে আবার নিচের কমান্ডটি দিন

```
git checkout master
```

লিঙ্ক ও সিঙ্গেল পেইজ

গত পর্বে আমরা আমাদের ইনডেক্স পেইজ বানিয়েছিলাম, আজ পোস্ট এ ক্লিক করে পুরা পোস্টটি দেখতে নতুন রাউট, কন্ট্রোলার ও ভিউ বানাবো।

আসুন প্রথমে আমাদের রুট বানিয়ে ফেলি routes.php ফাইলে।

```
Route::get('post/{id?}', 'PostsController@show');
```

এবার project.one/app/Http/Controllers/PostsController.php এর ডিভার show মেথডটি বানাই।

```
public function show($id){
    $post = Post::find($id);
    if($post == null) return redirect(action('PostsController@index'));
    return view('posts.single', compact('post'));
}
```

এবার project.one/resources/views/posts ডাইরেক্টরি তে single.blade.php নামে নিচের মতো ভিউ বানাই।

```
@extends('master')
@section('title', '|| Post')
@section('content')
    <div class="col-md-12">

        <article>
            <h1>{!! $post->title !!}</h1>
            <div>
                <p class="post-info"><span>Created at: {!! $post->created_at->format('d-M-Y') !!}</span>
            </div>
            <div class="single-post-content">
                {!! $post->content !!}
            </div>
        </article>
    </div>
@endsection
```

ইনডেক্স পেইজ এর লিঙ্ক এ ক্লিক করলে যেন এই ভিউ তে আসে সেজন্য পোস্ট এর লিঙ্কটা

project.one/resources/views/posts/index.blade.php ফাইলে পরিবর্তন করে দিয়ে আসি, পরিবর্তনের পর ফাইলটি হবে এরকমঃ


```

@extends('master')
@section('title', 'Posts')
@section('content')
    <div class="col-md-12">
        @if (count($posts))
            @foreach($posts as $post)
                <article>
                    <a href="{!! action('PostsController@show', $post->id) !!}"><h1>{!! $
                    <div>
                        <p class="post-info"><span>Created at: {!! $post->created_at->forma
                    </div>
                    <div class="post-excerpt">
                        {!! str_limit($post->content, 150) !!}
                    </div>
                </article>
                <hr>
            @endforeach
        @else
            <h1>Hi, you've landed in our First Project!</h1>
            <h2>Sorry, We don't have any post right now.</h2>
        @endif
    </div>
@endsection

```

কি ভাবে সোর্স কোড পাওয়া যাবে?

গিটহাব এর [project.one](#) রিপজিটোরিটি ফোর্ক করুন এবং আপনার সিস্টেমে ক্লোন করুন।

এই অধ্যায়ের সোর্স কোড পেতে

```
git checkout dcf3ab9
```

সর্বশেষ কমিট পর্যন্ত পেতে আবার নিচের কমান্ডটি দিন

```
git checkout master
```

কথাও বুঝতে সমস্যা হোলে নিচে কमेंট করুন, প্লিজ।

মিডলওয়্যার

এডভান্স রাউটিং

রাউট প্যারামিটার

আপনি চাইলে রাউটে বিভিন্ন প্যারামিটার পাঠাতে পারেন। মনে করুন আপনি user এর id প্যারামিটার URL থেকে নিতে চাচ্ছেন, সেক্ষেত্রে নিচের মত করে রাউট লিখতে পারেন।

```
Route::get('user/{id}', function ($id) {  
    return 'User ' . $id;  
});
```

এখানে দেখুন get এর প্রথম আর্গুমেন্ট এ কার্লি ব্র্যাকেট/দ্বিতীয় বন্ধনী এর ভেতরে id লেখা হয়েছে এবং দ্বিতীয় আর্গুমেন্ট এ ক্লোজারের মধ্যে যে ভ্যারিয়েবল/আর্গুমেন্ট পাস করা হয়েছে তা User এর সাথে যুক্ত করে রিটার্ন করা হয়েছে।

যদি আপনি ক্লোজার ব্যবহার না করে কন্ট্রোলার ব্যবহার করে থাকেন তবে আপনার কন্ট্রোলারের মেথডে নিচের মত করে আর্গুমেন্ট যোগ করুন।

```
// controller method  
public function show($id)  
{  
    return "User " . $id;  
}  
  
// route  
Route::get('/user/{id}', 'SomeController@show');
```

এই জাতীয় রাউট এর ক্ষেত্রে প্যারামিটার ব্যধ্যতামূলক। যদি প্যারামিটার ব্যধ্যতামূলক না করতে চান তাহলে get এর প্রথম আর্গুমেন্ট এ কার্লি ব্র্যাকেট/দ্বিতীয় বন্ধনী এর ভেতরে id লেখা হয়েছে তার শেষে এটি ? যুক্ত করুন। এবং দ্বিতীয় আর্গুমেন্ট এ ক্লোজারের মধ্যে যে ভ্যারিয়েবল/আর্গুমেন্ট পাস করা হয়েছে তার জন্য ডিফল্ট ভ্যালু ডিফাইন করে দিন।

```
Route::get('user/{id?}', function ($id=1) {  
    return 'User ' . $id;  
});
```

```
// controller method
public function show($id=1)
{
    return "User ".$id;
}

// route
Route::get('/user/{id?}', 'SomeController@show');
```

রেগুলার এক্সপ্রেসন

আপনি চাইলে রেগুলার এক্সপ্রেসন দিয়ে রাউট লিখতে পারেন। Route ক্লাসের where মেথড ব্যবহার করে প্যারামিটার এর সাথে রেগুলার এক্সপ্রেসন যুক্ত করে দিতে পারেন।

```
Route::get('user/{name}', function ($name) {
    return $name;
})
->where('name', '[A-Za-z]+');

Route::get('user/{name}/post/{id}', function ($name,$id) {
    return $name."'s post ".$id;
})
->where(['name'=> '[A-Za-z]+', 'id'=> '[0-9]+']);
```

বেটার লারাভেল ইনডায়রনমেন্ট

(বেস্ট প্রাকটিস ফলো করে কিভাবে আমরা একটি লারাভেল অ্যাপ্লিকেশন শুরু করতে পারি তা নিয়ে এই অধ্যায় এ আলোচনা করব)

টপিক লিস্ট

- ইনডায়রনমেন্ট ডিটেকশন
- কনফিগারেশন ম্যানেজমেন্ট ফর মাল্টিপল ইনডায়রনমেন্ট
- স্ট্রাকচারিং প্রজেক্ট কোডস
- সেটিং ম্যানড্রীল ফর ইমেইল ট্রান্সপোর্ট
- সেটিং আপ কনফাইড ফর অথেনটিকেশন
- সেটিং আপ কিউ

ইনভায়রনমেন্ট ডিটেকশন

এই চ্যাপ্টার এ আমরা ইনভায়রনমেন্ট নিয়ে বিস্তারিত আলোচনা করব।

ইনভায়রনমেন্ট সেটআপ কেন জরুরী?

বড় কোন অ্যাপ্লিকেশন এর বিভিন্ন স্টেজ থাকে। এবং প্রত্যেক স্টেজ এর জন্য সার্ভার ইন্সটলেশন অথবা ওয়েব হেড থাকে। উদাহরন হিসাবে আমরা একটা আইডিয়াল কেস চিত্রা করি।

ধরা যাক আমাদের অ্যাপ্লিকেশন এর গিট রিপো তে তিনটা ব্রাঞ্চ আছে যথাক্রমে `dev`, `staging` এবং `master` (which is also maybe the main production branch) এবং তিনটা ব্রাঞ্চ এর জন্য তিনটা ওয়েব ইন্সটলেশন আছে যথাক্রমে `dev.application.com`, `staging.application.com` এবং `application.com` এবং গিট হুক ব্যবহার করে আমরা সহজেই অটো ডিপ্লয়মেন্ট সেট করতে পারি যেন ডেভলপাররা গিট রিপো তে নতুন কোড কমিট করার সাথে সাথেই আমরা corresponding ইন্সটলেশন এ গিয়ে ইফেক্ট দেখে চেক করতে পারি।

ওর ম্যাবি প্রজেক্ট এর QA টিম ঐ ফিচারটা টেস্ট করা শুরু করে দিতে পারে। এখন কোন ফিচার প্রথমে `dev` এ যায়, সেখানে অন্য ডেভলপাররা অথবা QA টিম চেক করে গ্রিন সিগনাল দিলে, সেই ফিচার ম্যানেজার/বিজনিজ ওনার কে দেখানোর জন্য `staging` এ পুশ করা হয়। এবং সবাই ফাইনালি এপ্রুভ করলে `production` এ এন্ড ইউজার দের জন্য সেই ফিচারটা ওপেন করা হয়।

এখন আমাদের codebase তো একটা কিন্তু আমাদের তিন ধরনের কনফিগারেশন দরকার। কারণ `dev`, `staging` এবং `production` এর আলাদা আলাদা ডেটাবেজ থাকবে, `dev` এ দেখা গেল আমরা সুবিধার জন্য `sqlite` ব্যবহার করলাম, বাট স্টেজিং এ মাই সিকিউএল লাগবে। প্রোড এ অন্য কোন DBMS। কিংবা ডেভ এ আমরা ফাইল বেইসড ক্যাশিং ব্যবহার করলাম কিন্তু সার্ভার এ `memcached` অথবা `redis` লাগবে। এইসব এক্সটার্নাল সার্ভিস ব্যবহার করার জন্য আমাদের কোন php প্যাকেজ বা লাইব্রেরী ব্যবহার করতে হবে। বেশির ভাগ ক্ষেত্রেই এইসব প্যাকেজ এর কনফিগ ফাইল থাকে যেখানে ভিভিন্ন প্যারামিটার সেট অথবা দরকার মত বদলানো যায়।

কিংবা ধরুন আপনারা দুইজন একটা প্রোজেক্ট এ কাজ করতেন কিন্তু আপনাদের ডেটাবেজ এর কনফিগ ফাইল একটা। এখন কি হবে প্রত্যেক বার গিট থেকে কোড আপডেট করার পর আপনাকে ঐ ফাইল চেঞ্জ করতে হবে যেন অ্যাপ্লিকেশন আপনার লোকাল ডেটাবেজ এ কানেক্ট করতে পারে।

আরও একটা কমন কেইস হল, সেন্ডিং ইমেইল ফ্রম লোকাল মেশিন। সো আমরা চাই লোকাল ইনভায়রনমেন্ট এ মেইল ফাংশান কল করলে সেইটা শুধু লগ ফাইলে মেইল এর কন্টেন্ট লগ করবে যেহতু লোকাল মেশিন থেকে মেইল পাঠানো সহজ না, কিন্তু সার্ভার এ যেহতু মেইল সেন্ড করতে কোন সমস্যা নাই সেখানে যেন ঠিকমত মেইল সেন্ড হয়।

এইরকম অবস্থায় আমাদের যা দরকার তা হল মাল্টিপল সেট অফ কনফিগ ফাইলস যা কিছু কন্ডিশন এর উপর বেইজ করে অটোম্যাটিকালি লোড হবে এবং আমরা গিট এ কোড পুশ করা মাত্রই অন্য কোন কিছু চেঞ্জ করা ছাড়াই আমাদের কমিট করা নতুন ফিচার কারেসপন্ডিং ওয়েব হেড এ কাজ করবে।

সো ইনভায়রনমেন্ট ম্যানেজমেন্ট হল আমাদের অ্যাপ্লিকেশন কে কোন সময় কি করতে হবে তা ডিসাইড করতে সাহায্য করা।

লারাভেল কিভাবে ইনভায়রনমেন্ট ডিটেক্ট করে ?

প্রত্যেকটি লারাভেল অ্যাপ্লিকেশন যখন রান করে তখন বুটস্ট্র্যাপিং এর প্রথম দিকে লারাভেল ইনভায়রনমেন্ট ডিটেক্ট করে। এবং এর পর ওই রিকুয়েস্ট সাইকেল এর পরবর্তী অনেক কাজে ওই ইনভায়রনমেন্ট এর উপর নির্ভর করে বিভিন্ন ডিসিশান নেয়।

আমি এই সিরিজ এ লারাভেল এর 4.2.* ভার্সন নিয়ে কথা বলব। আমার জানি লারাভেল এর সব রিকুয়েস্ট `public/index.php` এর মাধ্যমে প্রসেস হয়। এই প্যাটার্ন `Front Controller Pattern` নামে পরিচিত। লারাভেল সহ সব `MVC framework` ই এই প্যাটার্ন ফলো করে। আচ্ছা আমরা লারাভেল রিকুয়েস্ট সাইকেল নিয়ে অন্য কোন পোস্ট এ কথা বলব।

লারাভেল ডিফল্ট ইন্সটলেশন এ `bootstrap/start.php` ফাইল এ ইনভায়রনমেন্ট ডিটেকশন হয় যা `public/index.php` ফাইল এ `autoload` এর পরই লোড করা হয়। সো বলা যায় ইনভায়রনমেন্ট ডিটেকশন লারাভেল রিকুয়েস্ট লাইফ সাইকেল এর একদম প্রথম দিকেই ঘটে, কারণ ইনভায়রনমেন্ট এর উপর নির্ভর করে লারাভেল কনফিগ ফাইল লোড করে।

```
$env = $app->detectEnvironment(array(  
  
    'local' => array('homestead'),  
  
));
```

বাই ডিফল্ট `local` ইনভায়রনমেন্ট এর জন্য হোস্টনেম `homestead` থাকে কারণ যারা `homestead` ব্যবহার করে তাদের যেন কিছু পরিবর্তন করতে না হয়। এইরকম অনেক ক্ষেত্রেই লারাভেল বিভিন্ন কনভেনশন ব্যবহার করে। এই কাজটার জন্য একটা ডেভিকেটেড টার্ম আছে `Convention Over Configuration` যেই কাজটা মেবি রেইলস প্রথমে শুরু করেছিল। এইটার উদ্দেশ্য হল প্রচলিত আলিখিত কিছু নিয়ম ফলো করলে কাজ অনেক সহজ হয়ে যাবে। কাইন্ড অফ দ্যা ফ্রেমওয়ার্ক উইল থিংক এহেড ফর ইয় ইন ম্যানি কেজেস টু ম্যাইক ইউর লাইফ ইজিয়ার।

লারাভেল ইনভায়রনমেন্ট ডিটেক্ট করার জন্য `hostname` ব্যবহার করে। বাট লাইক ম্যানি আদার থিংস, ইউ ক্যান ইউজ ইউর ওন ওয়ে টু।

নতুন ইনভায়রনমেন্ট যোগ করা

ইনভায়রনমেন্ট যোগ করার সহজ উপায় হল `hostname` ব্যবহার করা। লিনাক্স অথবা ম্যাক এ টার্মিনাল এ `hostname` কমান্ড ব্যবহার করে হোস্টনেম বের করা যায়। আমরা নিচের মত করে আরও ইনভায়রনমেন্ট যোগ করতে পারি।


```
$env = $app->detectEnvironment(array(

    'local' => array('homestead'),
    'staging' => array('staging-host-name'),
    'prod' => array('prod-host-name')

));
```

এইভাবে ইনভায়রনমেন্ট যোগ করলে লারাভেল হোস্টনেম চেক করে ঠিকমত ইনভায়রনমেন্ট ডিটেক্ট ও সেট করবে। কিন্তু আমাদের staging আর prod যদি একই সার্ভার এ থাকে তাহলে hostname ব্যবহার করে তাদের আলাদাভাবে ডিটেক্ট করা সম্ভব না।

সো আরও সুন্দরভাবে ইনভায়রনমেন্ট যোগ করার জন্য আমরা \$app->detectEnvironment() মেথড এ এরে এর পরিবর্তে পরিবর্তে closure ব্যবহার করে আমাদের সুবিধামত ইনভায়রনমেন্ট ডিটেকশন কোড যোগ করতে পারি।

```
$env = $app->detectEnvironment(function(){

    return getenv('LARAVEL_ENV') ?: 'local';

});
```

উপরের ফাংশনটি প্রথমে getenv() ফাংশান এর মাধ্যমে LARAVEL_ENV নামে কোন key ইনভায়রনমেন্ট ভ্যারিয়েবল এ আছে কিনা এবং যদি পায় তাহলে তার ভ্যালু রিটার্ন করে এবং ঐ ভ্যালুই ইনভায়রনমেন্ট হিসাবে সেট করে। আর যদি ঐ key Exist না করে তাহলে local রিটার্ন করে, অর্থাৎ ঐ key না থাকলে local ইনভায়রনমেন্ট সেট হয়।

আমরা কিভাবে ইনভায়রনমেন্ট ভ্যারিয়েবল সেট করতে পারি?

এপাচি তে ইনভায়রনমেন্ট ভ্যারিয়েবল সেট করা খুব সহজ। vhost অথবা htaccess ফাইল এ নিচের মত করে ইনভায়রনমেন্ট ভ্যারিয়েবল সেট করতে পারি।

```
SetEnv LARAVEL_ENV dev
```

অথবা আপনি যদি nginx ব্যবহার করেন তাহলে আপনাকে php-fpm, or php-cgi এর মাধ্যমে ইনভায়রনমেন্ট ভ্যারিয়েবল সেট করতে হবে কারণ nginx, apache এর মত নিজে php প্রসেস নিজে ম্যানেজ করেনা।

```
php-fpm

...
env[LARAVEL_ENV] = dev
...
php-cgi

location / {
    ...
    fastcgi_param LARAVEL_ENV dev;
    ...
}
```

আমরা পরের পোষ্ট এ লারাভেল কিভাবে মাল্টিপল কনফিগ ফাইল ইনভায়রনমেন্ট এর উপর বেইজ করে লোড করে তা দেখব।

কনফিগারেশন ম্যানেজমেন্ট ফর মাল্টিপল ইনভায়রনমেন্ট

আগের আধ্যায় এ আমরা লারাভেল কিভাবে ইনভায়রনমেন্ট ডিটেক্ট করে তা দেখেছি। এই আধ্যায় এ আমরা ইনভায়রনমেন্ট এর উপর নির্ভর করে কিভাবে কনফিগারেশন লোড হয় এবং আমরা কিভাবে ভিন্ন ভিন্ন ইনভায়রনমেন্ট এর জন্য ভিন্ন ভিন্ন কনফিগারেশন ব্যবহার করতে পারি তা দেখব।

আপনি হয়ত ইতিমধ্যেই জানেন লারাভেল `app/config` ফোল্ডার থেকে সব কনফিগ ফাইল লোড করে।

ধরা যাক আপনার ইনভায়রনমেন্ট হিসাবে `local` সেট করা আছে। এখন লারাভেল যখন কোন কনফিগ ফাইল লোড করতে যাবে তখন সবার আগে `app/config/local/` ডিরেক্টরি তে খুঁজবে। যদি ওই ডিরেক্টরি তে না পায় তাহলে `app/config/` থেকে লোড করবে।

সো লারাভেল প্রথমে `app/config/ENVIRONMENT_NAME/` ডিরেক্টরি তে কনফিগ ফাইল খুঁজে। যদি না পায় তাহলে রুট কনফিগ ডিরেক্টরি তে যে ডিফল্ট ফাইল থাকে এটা লোড করে। আপনার ইনভায়রনমেন্ট এর নামে কনফিগ ডিরেক্টরিতে একটি নতুন ফোল্ডার তৈরি করুন।

আপনি শুধু যে কনফিগ গুলো চেঞ্জ করতে চান ঐ ফাইল গুলোর ডিফল্ট ভার্শন আপনার ইনভায়রনমেন্ট এর কনফিগ ডিরেক্টরিতে কপি করে প্রয়োজনীয় অংশ চেঞ্জ করতে পারেন।

যেমন আপনি যদি ডেটাবেজ এর কানেকশন ইনফো চেঞ্জ করতে চান তাহলে তাহলে `app/config/database.php` ফাইলটি `app/config/local/` ডিরেক্টরিতে কপি করুন। তারপর আপনার কানেকশন ইনফো চেঞ্জ করতে পারেন।

এখন ধরুন আপনারা কয়েকজন একটি প্রজেক্ট এ কাজ করছেন তাহলে আপনি যদি পূর্বের টিউটোরিয়ালের মত করে ইনভায়রনমেন্ট সেটআপ করে থাকেন। তাহলে আপনাদের সবার ইনভায়রনমেন্ট `local` হিসাবে সেট হবে। কিন্তু আপনাদের ডাটাবেজ এর কানেকশন ইনফো ভিন্ন হতে পারে।

এই কেইজ টা হ্যান্ডেল করার জন্য লারাভেল এ সুন্দর একটা উপায় আছে। প্রথমে নিচের মত করে

`app/local/database.php` ফাইলটি পরিবর্তন করুন।

```
<?php
return array(
    'connections' => array(

        'mysql' => array(
            'driver'      => 'mysql',
            'host'        => getenv('DB_HOST'),
            'database'    => getenv('DB_NAME'),
            'username'    => getenv('DB_USERNAME'),
            'password'    => getenv('DB_PASSWORD'),
            'charset'     => 'utf8',
            'collation'   => 'utf8_unicode_ci',
            'prefix'      => '',
        )
    )
);
```

আপনি যদি ইতিমধ্যেই না জানেন, পিএইচপির `genenv()` ফাংশনটি ইনভায়রনমেন্ট ড্যারিয়েবল এর ড্যালু পাওয়ার জন্য ব্যবহার করা হয়।

এখন লোকাল ইনভায়রনমেন্ট এ আপনি কিভাবে সহজে ইনভায়রনমেন্ট ড্যারিয়েবল সেট করবেন। আপনি জেনে খুশি হবেন যে লারাবেল এর অথর এইটা নিয়ে ভেবেছে এবং এর জন্য সহজ একটি ব্যবস্থা রেখেছে।

লারাবেল সিস্টেম এ সহজে ইনভায়রনমেন্ট ড্যারিয়েবল যোগ করার জন্য আপনি `.env.`

`{ENVIRONMENT_NAME}.php` নামে একটি ফাইল আপনার প্রজেক্ট এর রুট ডিরেক্টরিতে যোগ করতে পারেন।

লারাবেল বুটস্ট্রাপ এর সময় ঐ ফাইল থেকে ড্যালু গুলো রিড করে ইনভায়রনমেন্ট গ্লোবাল ড্যারিয়েবল এ যোগ করে দিবে।

সো লোকাল এর জন্য আপনাকে `env.local.php` যোগ করতে হবে। `env.local.php` ফাইলটিতে নিচের মত করে ড্যালু এড করতে পারেন।

```
<?php

return [
    'DB_HOST' => 'localhost',
    'DB_NAME' => 'laravel_database_name',
    'DB_USERNAME' => 'db_username',
    'DB_PASSWORD' => 'db_password'
];
```

আপনি ড্যারিয়েবল এর নাম গুলো দেখে বুঝতে পারবেন আমরা এই নাম এর ড্যারিয়েবল ডাটাবেজ কনফিগ এ ব্যবহার করেছি। এই ফাইল আপনি গিট ইগনোর ফাইল এ এড করে আপনাদের ডেটাবেজ এর কনফিগ আলাদা রাখতে পারেন। তাহলে সবাই যার যার নিজের ডাটাবেজ এ কানেক্ট করতে পারবেন কোন ফাইল পরিবর্তন না করেই।

এখানে বলে রাখা ভাল, আপনি চাইলে লাইভ সাইট এও এইভাবে ডাটাবেজ কনফিগ লোড করতে পারেন। কিন্তু ভাল প্রাকটিস হল কোন ফাইল এ সেন্সিটিভ ইনফো না রাখা। সিস্টেমে ইনভায়রনমেন্ট ড্যারিয়েবল হিসাবে রাখা অনেক বেশি নিরাপদ।

আরেকটি বিষয় আলোচনা করে আমরা এই অধ্যায় শেষ করব।

অনেক প্যাকেজ এর জন্য আমাদের `app/config/app.php` ফাইল এ `Service Provider` এড করতে হয়।

এখন অনেক প্যাকেজ আছে যা আপনি শুধু লোকালি ব্যবহার করবেন লাইভ এ দরকার নাই। সেক্ষেত্রে আপনি শুধু `app/config/app.php` ফাইলে নিচের মত করে কনফিগ এপেন্ড করতে পারেন।

```
<?php

return array(

    'debug' => true,

    'providers' => append_config([
        'Way\Generators\GeneratorsServiceProvider',

    ])

);
```

তাহলে এই জেনারেটর প্যাকেজটি শুধু আপনার লোকাল ইনডায়রনমেন্ট এই কাজ করবে।

লারাভেল কনফিগ ক্যাস্কেডিং করে লোড করে। অর্থাৎ লারাভেল এর যখন কোন ভ্যালু দরকার হয় তা প্রথমে `app/config/ENVIRONMENT_NAME/` ফোল্ডার এর মধ্যে খুঁজে এবং ঐখানে না পেলে `app/config/` ফোল্ডার থেকে লোড করে।

বেইজ কনফিগ ফোল্ডার থেকে আপনি যে ফাইলটি পরিবর্তন করতে চান ঐ ফাইলটি আপনার ইনডায়রনমেন্ট ফোল্ডার এ কপি করে প্রয়োজনীয় অংশটুকু পরিবর্তন করে কাজ করতে পারেন।

লারাভেল প্যাকেজ ডেভেলপমেন্ট

প্যাকেজঃ

পিএইচপি কিংবা কোন প্রোগ্রামিং ল্যাংগুয়েজে ব্যবহৃত প্যাকেজ বলতে উক্ত ল্যাংগুয়েজে তৈরি লাইব্রেরীকে বোঝান হয়ে থাকে সেটা বিভিন্ন প্রজেক্টে বারবার ব্যবহার উপযোগী। পিএইচপিতে ব্যবহারের ক্ষেত্র অনুযায়ী প্যাকেজ সাধারণত দুই ধরনের হয়ে থাকে একটি হল স্ট্যান্ডঅ্যালন আরেকটি হল কোন নির্দিষ্ট ফ্রেমওয়ার্ক কিংবা স্ট্যাক ভিত্তিক।

লারাভেল প্যাকেজঃ

লারাভেলের প্যাকেজ পিএইচপির স্ট্যান্ডঅ্যালন প্যাকেজ এর মতই শুধু পার্থক্য হল এটি লারাভেলের ব্যবহার উপযোগী আর লারাভেল কেন্দ্রিক করেই তৈরি করা হয়।

একটি লারাভেল প্যাকেজ এর মধ্যে নিচের রিসোর্স গুলো থাকতে পারেঃ

1. Service Provider.
2. Configurations.
3. Migrations.
4. Seeds.
5. Views.
6. Routes.
7. Translations.
8. Assets etc.

বিঃদ্রঃ একটি লারাভেল প্যাকেজের মধ্যে **Service Provider** থাকতেই হবে।

লারাভেলের প্যাকেজ তৈরি এর আগে পিএইচপিতে কিভাবে প্যাকেজ তৈরি করতে হয় সেই সম্পর্কে ধারণা থাকলে ভাল হয়। আর এই জন্য এই কয়লারপেটটি দেখে নিতে পারেন।

আমরা লারাভেলের প্যাকেজ “illuminate/workbench” এই প্যাকেজটি ব্যবহার করেও প্যাকেজ জেনারেট করতে পারি কিন্তু সমস্যা হল এটি লারাভেল ভার্সন ৪.২ এর পর থেকে বাদ দেয়া হয়েছে। তাছাড়াও এটিতে আর সাপোর্ট দেয়া হয়না, আর ক্লাস অ্যাটোলোডিং এর জন্য PSR-0 ব্যবহার করা হয়েছে যেটি ডেপ্ৰিকেটেড হয়েছে কিছুদিনের মধ্যে রিমুভ করা হবে। কাজেই আমরা আমাদের প্যাকেজ তৈরি করতে “illuminate/workbench” প্যাকেজটি ব্যবহার করবনা আর **PSR-4** স্ট্যান্ডার্ড অনুসরণ করব।

প্যাকেজ তৈরিকরণ প্রক্রিয়াঃ

লারাভেল কিংবা পিএইচপির প্রজেক্টে কম্পোজার প্যাকেজের জন্য **vendor** নামে একটি বাই ডিফল্ড ডিরেক্টরি থাকে। আপনার ভেন্ডর ও প্যাকেজ ডিরেক্টরি তৈরি করার জন্য উক্ত **vendor** ডিরেক্টরিতে ঢুকতে হবে। এবার ধরুন আপনার ভেন্ডর ডিরেক্টরির নাম দিলাম **my-vendor** আর এর ভিতরে প্যাকেজ ডিরেক্টরির জন্য **my-package** নাম দিলাম। মনে রাখবেন ডিরেক্টরি এর নাম সব সময় যেন ছোট হাতের লেখা হয়। এবার প্যাকেজের ইনফরমেশন এবং কনফিগারেশনের জন্য নিচের মত করে **composer.json** ফাইল তৈরি করতে হবে।

```
{
    "name": "my-vendor/my-package",
    "license": "MIT",
    "description": "Description about your package",
    "keywords": [],
    "authors": [
        {
            "name": "Your Name",
            "email": "email@example.com"
        }
    ],
    "require": {
        "php": ">=5.5.9",
        "illuminate/support": "5.1.*"
    },
    "autoload": {
        "psr-4": {
            "MyVendor\\MyPackage\\": "src/"
        }
    }
}
```

এখানে **"name": "my-vendor/my-package"** এই সেকশনে আপনার ভেন্ডর এবং প্যাকেজ ডিরেক্টরির নাম দিতে হবে।

নিচের সেকশনে আপনার প্যাকেজের নেমস্পেস, সাব-নেমস্পেস দিতে হবে। আর **src** ডিরেক্টরির মধ্যে আমরা সব ফাইল রাখব এইজন্য **"src/"** দেয়া হয়েছে।

এবার একটি সার্ভিস প্রোভাইডার তৈরি করা শিখব। নিচে **"MyPackageServiceProvider.php"** নামে একটি স্যাম্পল প্রোভাইডার দেখানো হলঃ

```
<?php

namespace MyVendor\MyPackage;

use Illuminate\Support\ServiceProvider;

class MyPackageServiceProvider extends ServiceProvider
{
    /**
     * Indicates if loading of the provider is deferred.
     */
}
```

```

* @var bool
*/
protected $defer = false;

/**
 * Perform post-registration booting of services.
 *
 * @return void
 */
public function boot()
{
    // Loading routes
    if (!$this->app->routesAreCached()) {
        require __DIR__ . '/routes.php';
    }

    // Publishing configs
    $this->publishes([
        __DIR__ . '/config/my-package.php' => config_path('my-package.php'),
    ]);

    // Publishing views
    $this->publishes([
        __DIR__ . '/views' => base_path('resources/views'),
    ]);

    // Loading translations
    $this->loadTranslationsFrom(__DIR__ . '/translations', 'my-package');

    // Publishing public assets
    $this->publishes([
        __DIR__ . '/assets' => public_path('my-vendor/my-package'),
    ], 'public');

    // Publishing migrations
    $this->publishes([
        __DIR__ . '/migrations' => database_path('/migrations'),
    ], 'migrations');

    // Publishing seeds
    $this->publishes([
        __DIR__ . '/seeds' => database_path('/seeds'),
    ], 'migrations');
}

/**
 * Register bindings in the container.
 *
 * @return void
 */
public function register()

```

```
{
    $this->app->bind( 'MyPackageClass', 'MyVendor\MyPackage\MyPackageClass' );
}

}
```

উপরের সার্ভিস প্রোভাইডার ক্লাসে নেমস্পেস **namespace MyVendor\MyPackage;** দেয়া হয়েছে। আপনার প্যাকেজ তৈরি করার সময় আপনার মত করে এটি দিবেন।

এখানে **Illuminate\Support\ServiceProviders** ব্যবহার করা হয়েছে আর এইটাকে এক্সটেন্ড করে প্যাকেজের সার্ভিস প্রোভাইডার ডিক্লেয়ার করা হয়েছে। এই লাইনে `protected $defer = false;` ডেফার মোড ফলস করা হয়েছে। ডেফার সার্ভিস প্রোভাইডার বলতে সার্ভিস প্রভাইডারটি সার্ভিস কন্টেইনারে রেজিস্টার হবে কিন্তু প্রয়োজন না হওয়া পর্যন্ত অ্যাপ্লিকেশনে লোড হবে না।

এবার দেখতে পাচ্ছেন **boot()** মেথডটিতে কিছু রাউট, কনফিগারেশন, ডাটাবেস মাইগ্রেশন, সিডিং, ডিউস ফাইল আরও কিছু অ্যাসেটস লোড কিংবা পাবলিশের জন্য ডিক্লেয়ার করা হয়েছে। এসব রিসোর্স আপনার প্যাকেজে প্রয়োজন হলে স্যাম্পল কোডের মত করে ডিক্লেয়ার করবেন আর তার সাথে ডিরেক্টরি তৈরি করে ওই ডিরেক্টরির মধ্যে ফাইল গুলো রাখবেন। তবে প্যাকেজের মধ্যে এই রিসোর্স গুলো থাকা অপরিহার্য না। এখানে পাবলিশ বলতে রিসোর্স গুলো প্যাকেজ থেকে অ্যাপ্লিকেশন এর ভিতরে কাংখিত ডিরেক্টরিতে হস্তান্তর করা বুঝান হয়েছে।

register() মেথডটির মধ্যে সাধারণত প্যাকেজের ক্লাস ডিপেন্ডেন্সি গুলো লারাভেলের সার্ভিস কন্টেইনারে বাইন্ড করার জন্য ডিক্লেয়ার করা হয়। নিচের লাইনে দেখলে বুঝতে পারবেন আমি **MyPackageClass** নামে একটি ক্লাস বাইন্ড করেছি।

```
$this->app->bind( 'MyPackageClass', 'MyVendor\MyPackage\MyPackageClass' );
```

MyPackageClass এর জন্য নিচের কোড লিখেছি।

```
<?php

namespace MyVendor\MyPackage;

class MyPackageClass
{
    public static function sayHi()
    {
        return 'Hello World!';
    }
}
```

প্যাকেজটি ব্যবহারের নিয়মঃ স্যাম্পল প্যাকেজটি আপনার অ্যাপ্লিকেশনে ব্যবহার করতে চাইলে আমার [এই বয়লারপ্লেটটি](#) দেখতে কিংবা ব্যবহার করতে পারেন। যদিও ব্যবহারের নিয়ম বয়লারপ্লেটটিতে দেয়া আছে তারপরও নিচের মত করে কনফিগার করতে পারেন। আপনার লারাভেল প্রজেক্টের মেইন `composer.json` ফাইলে নিচের কোড যোগ করতে হবেঃ

```
"psr-4": {
    "App\\": "app/",
    "MyVendor\\MyPackage\\": "vendor/my-vendor/my-package/src"
}
```

ক্লাস অটোলোডিং এর জন্য কম্পোজার ফাইলে প্যাকেজটিকে চিনিয়ে দেয়া হয়েছে তবে মনে রাখবেন যখন আপনি কম্পোজার প্যাকেজ এর জন্য "packagist.org" আপনার প্যাকেজটি সাবমিট কিংবা পাবলিশ করবেন তখন উপরের লাইন লেখার কোন দরকার নাই। এবার সার্ভিস প্রোভাইডারটি "config/app.php" ফাইলে নিচের মত করে যোগ করতে হবে।

```
'providers' => [
    ...

    MyVendor\MyPackage\MyPackageServiceProvider::class,
],
```

এবার নিচের কমান্ড রান করেনঃ

```
composer dump-autoload
composer update
```

আর প্যাকেজের রিসোর্স গুলো পাবলিশ করার জন্যঃ

```
php artisan vendor:publish
```

এবার প্যাকেজের **MyPackageClass** টি অ্যাপ্লিকেশনের মধ্যে ব্যবহারের জন্য নিচের মত করে লিখতে হবে।

```
$myPackage = $this->app['MyPackageClass'];
echo $myPackage::sayHi();
```

অথবা নিচের মত করেও সরাসরি ব্যবহার করা যাবে।

```
echo \MyVendor\MyPackage\MyPackageClass::sayHi();
```

লারাভেলের প্যাকেজ ডেভেলপমেন্ট সম্পর্কে বিস্তারিত জানতে [এই লিঙ্ক](#) অনুসরণ করবেন। লারাভেলের সার্ভিস প্রোভাইডার সম্পর্কে বিস্তারিত জানতে [এই লিঙ্ক](#) অনুসরণ করবেন। সার্ভিস প্রোভাইডার সম্পর্কে বিস্তারিত জানতে [এই লিঙ্ক](#) অনুসরণ করবেন।

এছাড়াও আপনারা আমার ডেভেলপ করা কিছু প্যাকেজ দেখে আইডিয়া নিতে পারেন।

<https://github.com/appzcoder/>

লারাভেলে ইভেন্ট ব্রডকাস্টিংঃ

ইভেন্ট হল কোন এক মূল এ্যাক্টিভিটি কিংবা ফাংশন কল করার সময় অন্য কোন বিশেষ টাস্ক কিংবা এ্যাক্টিভিটি কার্যকর করা। ধরুন ইউজার রেজিস্ট্রেশনের সময় ইউজারকে স্বাগতম ইমেইল পাঠাতে হবে সেটা ইভেন্ট ফায়ার আপ করে করা যায়। আর এই জন্য উক্ত ইভেন্টের জন্য একটা লিসেনার সেট করতে হবে যেটি ব্যকগ্রাউন্ডে ওই কাজটি করে দিবে।

লারাভেলে স্বাভাবিক ভাবেই ইভেন্ট আর লিসেনার সেট করে এই ধরনের কাজ করা যায়। এমনকি ইভেন্ট কিউইং করারও সিস্টেম আছে।

এবার মূল বিষয় হচ্ছে ইভেন্ট ফায়ার করা এবং তার সাথে সাথে সেটা ব্রডকাস্টিং করা যাতে করে সকল ইউজার কিংবা ক্লায়েন্ট উক্ত ইভেন্ট কে বুজতে পারে। আর সেটা নোটিফিকেশনের মাধ্যমে ইউজারদের কাছে পাঠানো হয়।

পিএইচপি কিংবা লারাভেলে **Pusher** লাইব্রেরী ব্যবহার করে রিয়ালটাইম ইভেন্ট ব্রডকাস্ট করা যায় কিন্তু **Pusher** পেইড সার্ভিস এছাড়াও এর **HTTP Latency** অনেক বেশি অর্থাৎ রিকুয়েস্ট রেসপন্সে অনেক সময় যার কারণে আমরা **Redis** ব্যবহার করব যেটি **NodeJS**, **Socket.io** এর সাথে কাজ করে আর অনেক দ্রুত কাজ করে।

প্রক্রিয়াঃ

প্রথমে আমাদেরকে ইভেন্ট আর লিসেনার তৈরি করে নিতে হবে। ইভেন্ট আর লিসেনার যথাক্রমে **app/Events** ও **app/Listeners** ডিরেক্টরিতে থাকে। যদিও ইভেন্ট ও লিসেনার আলাদা আলাদা করে আর্টিসান কমান্ডের মাধ্যমে জেনারেট করা সম্ভব কিন্তু আমরা এটি খুব সহজেই **EventServiceProvider** এ ডিফাইন করে আর্টিসান কমান্ড দিয়ে জেনারেট করব।

এবার ধরুন আমাদের ক্ষেত্রে আমরা **UserRegisteredEvent** নামে ইভেন্ট রাখব তাহলে নিচের মত করে কোডটি **app/Providers/EventServiceProvider.php** ফাইলে লিখতে হবেঃ

```
protected $listen = [  
    'App\Events\UserRegisteredEvent' => [  
        'App\Listeners\UserRegisteredEventListener',  
    ],  
];
```

এরপর আর্টিসান কমান্ড **php artisan event:generate** রান করতে হবে। যার ফলে ইভেন্ট আর লিসেনার স্বয়ংক্রিয় ভাবে প্রয়োজন মারফিক জেনারেট হবে।

ইভেন্ট আর লিসেনার তৈরি করার পর কিছু পরিবর্তন করার ফলে নিচের মত হবে।

ইভেন্টঃ


```
<?php

namespace App\Events;

use App\Events\Event;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;

class UserRegisteredEvent extends Event implements ShouldBroadcast
{
    public $user;

    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct($user)
    {
        $this->user = $user;
    }

    /**
     * Get the channels the event should be broadcast on.
     *
     * @return array
     */
    public function broadcastOn()
    {
        return ['test-channel'];
    }
}
```

উপরের কোড এ খেয়াল করলে দেখবেন ইভেন্ট ক্লাসে আমরা **ShouldBroadcast** ইন্টারফেস ইমপ্লিমেন্ট করেছি যেটা ইভেন্ট ব্রডকাস্টের জন্য করতে হয়। আবার ক্লাসের কন্সট্রাক্টর ফাংশনের প্যারামিটারে **\$user** ভেরিয়েবল পাস করতেছি যেটা পাবলিক ভেরিয়েলে এসাইন থাকবে এবং পরবর্তীতে লিসেনার থেকে একসেস করা যাবে। এবার একবারে নিচের দিকে খেয়াল করলে দেখবেন **broadcastOn()** মেথড/ফাংশনটি নিচের মত করে ডিফাইন করা হয়েছে।

```
public function broadcastOn()
{
    return ['test-channel'];
}
```

উক্ত ফাংশনে রেডিসের একটা চ্যানেল দিতে হবে আর আমাদের ক্ষেত্রে **test-channel** নাম দিয়েছি। এটা যেকোনো নামে হতে পারে তবে যেহেতু এটি ইভেন্ট ব্রডকাস্টের জন্য সাবস্কাইব করা হয় আর ক্লায়েন্ট-ইন্ডে একই চ্যানেল উল্লেখ করে লিসেন করতে হয় কাজেই একটি অর্থপূর্ণ নাম দেয়াই ভাল।

ইভেন্ট লিসেনারঃ

```
<?php

namespace App\Listeners;

use App\Events\UserRegisteredEvent;

class UserRegisteredEventListener
{
    /**
     * Create the event listener.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Handle the event.
     *
     * @param UserRegisteredEvent $event
     * @return void
     */
    public function handle(UserRegisteredEvent $event)
    {
        var_dump("User: " . $event->user);
    }
}
```

এখানে **handle()** ফাংশনে আমাদের ইভেন্ট ক্লাসটি ইনজেক্ট করা হয়েছে আর **\$event->user** প্রোপার্টি একসেস করা হচ্ছে। এই প্রোপার্টি ইচ্ছেমত ইমপ্লিমেন্ট করা যায় আপাতত ডাম্প করে দেখানো হয়েছে।

এবার রাউট এ আমরা নিচের মত করে ডিফাইন করব।

```
Route::get('broadcast', function () {
    event(new App\Events\UserRegisteredEvent('Sohel Amin'));

    return 'Event has been fired!';
});

Route::get('listen', function () {
    return view('events');
});
```

এখানে দুইটি রাউট ডিফাইন করা হয়েছে। **broadcast** রাউটের মাধ্যমে ইভেন্ট ফায়ার করার জন্য ইভেন্ট কল করা হয়েছে আর প্যারামিটারে নাম দেয়া হয়েছে উদাহরন হিসেবে যেটি লিসেনার এ **\$event->user** হিসেবে একসেস করা যাবে। আর **listen** রাউটে একটা ভিউ ফাইল লোড করা হয়েছে যেটিতে নিচের মত কোড আছে।

```
<!DOCTYPE html>
<html>
  <head>
    <title>Laravel Event Broadcasting</title>
    <link href="//fonts.googleapis.com/css?family=Lato:100" rel="stylesheet" type="text/css">
  </head>
  <body>

    <h2>User's List</h2>
    <ul id="user-list">
    </ul>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.3.6/socket.io.min.js">
    </script>
    var socket = io('http://localhost:3000');

    socket.on("test-channel:App\\Events\\UserRegisteredEvent", function(message){
      console.log(message);

      // Appending user to user's list
      var ul = document.getElementById("user-list");
      var li = document.createElement("li");
      li.appendChild(document.createTextNode(message.user));
      ul.appendChild(li);

    });
  </script>
</body>
</html>
```

এখানে আমরা **socket.io** ব্যবহার করেছি কাজেই ভিউ ফাইলে **socket.io** এর জেএস ফাইল ইনক্লুড করেছি এবং চ্যানেল আর ইভেন্ট এর নাম নেমস্পেস সহ ডিফাইন করেছি।

এবার ব্রডকাস্টিং এর কনফিগারেশন ফাইলে ডিফল্ট ড্রাইভার হিসেবে রেডিস ডিফাইন করব নিচের মত করে **config/broadcasting.php** এই ফাইলে।

```
'default' => env('BROADCAST_DRIVER', 'redis'),
```

এর সাথে সাথে রেডিস এর পিএইচপি লাইব্রেরী কম্পোজারে অ্যাড করে নিতে হবে। আর এই জন্য নিচের কমান্ড লিখতে হবে।

```
composer require predis/predis
```

এখন আমাদের এই কাজে **NodeJS**, **Redis** আর **Socket.io** লাগবে কাজেই এগুলো ইন্সটল করে নিতে হবে।

NodeJS ইন্সটল করা [এই লিংক](#) দেখে নিতে পারেন।

Redis Server ইন্সটল করা [এই লিংক](#) থেকে দেখে নিতে পারেন।

এবার **socket.io** জন্য একটা স্ক্রিপ্ট নিচের মত করে লিখতে হবে যেটি **NodeJS** এ করা আর **ExpressJS** এও লেখা সম্ভব। আর এটি অ্যাপ্লিকেশনের রুট ডিরেক্টরিতে রাখতে হবে।

```
var app = require('http').createServer(handler);
var io = require('socket.io')(app);

var Redis = require('ioredis');
var redis = new Redis();

app.listen(3000, function() {
  console.log('Server is running!');
});

function handler(req, res) {
  res.writeHead(200);
  res.end('');
}

io.on('connection', function(socket) {
  //
});

redis.psubscribe('*', function(err, count) {
  //
});

redis.on('pmessage', function(subscribed, channel, message) {
  message = JSON.parse(message);
  io.emit(channel + ':' + message.event, message.data);
});
```

স্ক্রিপ্টে খেয়াল করলে দেখবেন নোডের কিছু মডিউল কিংবা প্যাকেজ যেমনঃ **socket.io**, **ioredis** ব্যবহার করা হয়েছে কাজেই আমাদেরকে **npm** প্যাকেজ ম্যানেজারটি ইন্সটল আর কনফিগার করার জন্য প্রজেক্ট ডিরেক্টরি থেকে টার্মিনালে নিচের কমান্ড রান করাতে হবে।

```
npm install
npm install ioredis
npm install socket.io
```

এবার মোটামুটি সকল কাজ শেষ ইভেন্ট ব্রডকাস্ট করার জন্য প্রজেক্টটি রান করাতে হবে এইক্ষেত্রে নিচের ধাপ অনুসরণ করতে হবে।

প্রজেক্ট/অ্যাপ্লিকেশন রান করার জন্য প্রজেক্ট ডিরেক্টরি থেকে টার্মিনালে:

```
php artisan serve
```

রেডিস সার্ভার রান করার জন্য অন্য টার্মিনালে:

```
redis-server
```

নোডের **socket.js** ফাইলটি রান করার জন্য প্রজেক্ট ডিরেক্টরি থেকে টার্মিনালে:

```
node socket.js
```

পরিশেষে ইভেন্ট ব্রডকাস্ট করার জন্য <http://localhost:8000/broadcast/> এই লিংকে হিট করতে হবে আর ইভেন্ট লিসেন কিংবা রিয়ালটাইম নোটিফিকেশন দেখতে <http://localhost:8000/listen/> এই লিংকে হিট করতে হবে আর এটি একাধিক ব্রাউজার থেকে দেখতে পারেন।

সোর্স কোডটি [এই লিংক](#) থেকে দেখে নিতে পারেন। <https://github.com/sohelamin/laravel-event-broadcast>