Before we begin

1. Ensure you have PHP 7.1.3 installed with these extensions:

OpenSSL	Mbstring	Tokenizer
XML	Ctype	JSON
PDO	SQLite	PDOSqlite

(use `php -m` to confirm)

2. Ensure you have *Composer* installed
3. Grab https://github.com/19ft/phpyorks

Using Laravel for Rapid Development

Rob Allen ~~& Gary Hockin~~

April 2018

Today's plan

- Getting started with Laravel
- Creating pages
- Databases
- Authentication
- Lunch!

What is Laravel?

Laravel

Elegant applications delivered at warp speed.

- Created by Taylor Otwell
- Full stack framework
- Focussed on rapid development
- MVC structure
- CLI tooling with `artisan`

What do you get?

- Routing
- Database ORM
- Caching
- Cookies
- Events
- etc!

- DI Container
- Authentication
- Console
- Encryption
- Forms

Why Laravel?

- Everything in the box
- Opinionated
- Convention over configuration
- `artisan` CLI tool
- Ecosystem
- Great community
- Laracasts

Ecosystem

- Cashier - subscription billing
- Dusk - browser tests
- Echo - broadcasting over sockets
- Envoy - task runner
- Horizon - Redis queue integration & dashboard
- Passport - API authentication
- Scout - Full-text search
- Socialite - external OAuth authentication
- Envoyer - Deployment (paid)
- Spark - SAAS scaffolding (paid)

Running Laravel apps

- `artisan serve` - use built-in PHP server
- Homestead - local Vagrant setup
- Forge - server manager and deployment system

Getting started

Installation

1. Install Laravel Installer:

```
composer global require "laravel/installer"
```

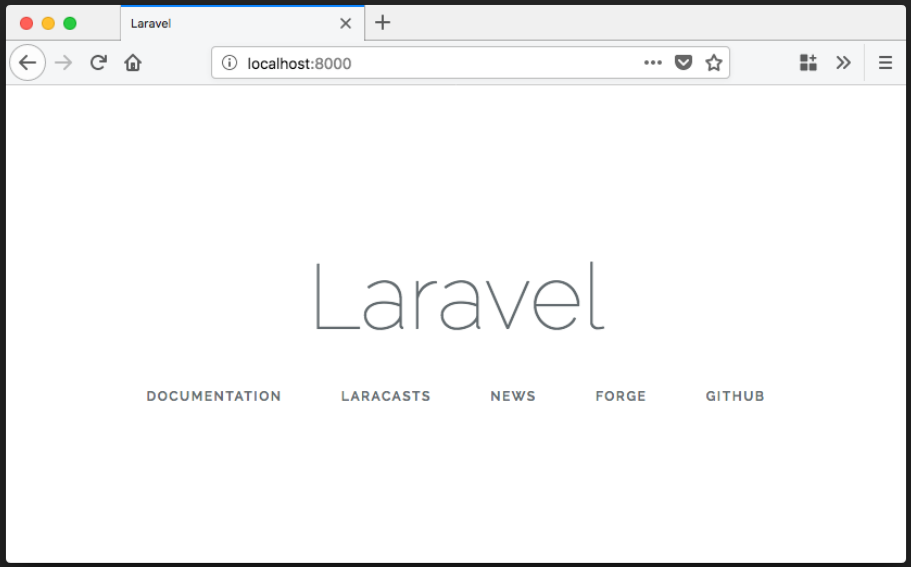
2. Create a new project:

```
laravel new my-project-name
```

3. Run using local webserver:

```
cd my-project-name && php artisan serve
```

First run


Laravel
localhost:8000
Laravel
DOCUMENTATION
LARACASTS
NEWS
FORGE
GITHUB

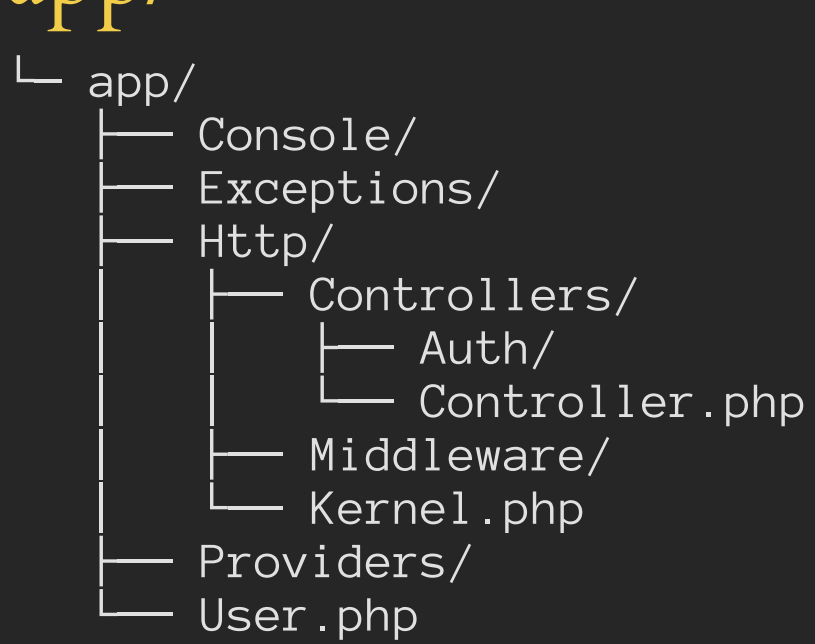
Coding time!

So how does Laravel work?!

Directory structure

```
.
├── app/
├── bootstrap/
├── config/
├── database/
├── public/
│   ├── index.php
├── resources/
│   ├── assets/
│   ├── lang/
│   └── views/
├── routes/
├── storage/
├── tests/
├── artisan
├── .env
├── composer.json
├── composer.lock
└── package.json
```

app/


app/
Console/
Exceptions/
Http/
Controllers/
Auth/
Controller.php
Middleware/
Kernel.php
Providers/
User.php

Dispatch process

1. Bootstrap
2. Dispatch
    a. Run middleware
    b. Execute router
    c. Invoke controller
    d. Render view

Configuration

`config/` holds a set of PHP files; each returns an array.

```
// config/app.php
<?php
return [
    'name' => env('APP_NAME', 'Laravel'),
    'env' => env('APP_ENV', 'production'),
    'debug' => env('APP_DEBUG', false),
    'url' => env('APP_URL', 'http://localhost'),
    'timezone' => 'UTC',
    'locale' => 'en',
    // etc...
];
```

.env file

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:Ohv+3NM3HHHpkCuG3VFK8WzSvY1AfAR4A7PlkvI=
APP_DEBUG=true
APP_URL=http://localhost
# etc...
```

Access using `config()` function:

```
$appName = config('app.name');
```

Routing

A route is defined in `routes/web.php` & consists of:

- HTTP method
- URI
- An action (closure or class method)

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

Router methods

```
Route::get('/hello', $callable);
Route::post('/hello', $callable);
Route::put('/hello', $callable);
Route::patch('/hello', $callable);
Route::delete('/hello', $callable);
Route::options('/hello', $callable);

Route::any('/hello', $callable);
Route::match(['get', 'post'], '/hello', $callable);

Route::redirect('/bonjour', '/hello', 301);
Route::view('/hello', 'route-name');
```

URI pattern

- Literal string match
```
Route::get('/hello', function () {…});
```
- Parameters are wrapped in { }
```
$app->get('/hello/{name}', function ($name) {…});
```
- Optional parameters end with ?
```
$app->get('/hello/{name?}', function ($name='Rob') {…
```
- Constrain parameters via Regex
```
$app->get('/hello/{name}', …)
  ->where('name', '[A-Za-z]+');
```

Routes can be named

Assign a name to a route so you can generate URLs from it

```
$app->get('/hello/{name}', …)->name('hello');


// Generating URLs...
$url = route('hello', ['name' => 'Rob']);

// Generating Redirects...
return redirect()->route('hello', ['name' => 'Rob']);
```

Controllers & actions

In addition to closures, you can specify an action method in a controller:

```
$app->get('/hello/{name}', 'UserController@profile');
```

`UserController`:
The `\App\Http\Controllers\UserController` class

`profile`:
The `profile()` method in `UserController`

Actions

- Receive route parameters
- Can optionally receive Request object
- Manage business logic operations
- Return a `Response` object or a `View` object

Actions

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

class UserController extends Controller
{
  public function profile($name)
  {
    // grab $user from the model layer
    return view('user/profile', ['user' => $user]);
  }
}
```

Blade templates

- Call `view()` to render template:
- Store templates in `resources/views` directory
- Use the `.blade.php` file extension

Action template

```
@extends('layout')

@section('content')
  <h1>User Profile</h1>
  <ul>
    @foreach ($user->hobbies as $hobby)
      <li>{{ $hobby }}</li>
    @endforeach
  </ul>
@endsection
```

Print variables

```
@extends('layout')

@section('content')
  <h1>User Profile</h1>
  <ul>
    @foreach ($user->hobbies as $hobby)
      <li>{{ $hobby }}</li>
    @endforeach
  </ul>
@endsection
```

Control statements

```
@extends('layout')

@section('content')
  <h1>User Profile</h1>
  <ul>
    @foreach ($user->hobbies as $hobby)
      <li>{{ $hobby }}</li>
    @endforeach
  </ul>
@endsection
```

Template inheritance

```
@extends('layout')

@section('content')
  <h1>User Profile</h1>
  <ul>
    @foreach ($user->hobbies as $hobby)
      <li>{{ $hobby }}</li>
    @endforeach
  </ul>
@endsection
```

Template inheritance

- For cohesive look and feel
    - includes default CSS, JS & structural HTML
- Build a base skeleton
- Define *sections* for children to override
- Each child chooses which template to inherit from

Base layout

```
<!-- resources/views/layouts/app.blade.php: -->
<html>
  <head>
        <link rel="stylesheet" href="style.css" />
        <title>@yield('title', 'Site name')</title>
  </head>
  <body>
    @yield('content')
  </body>
</html>
```

Let's build an application!

Simple Todo List

Our todo list will display a list of items to be done.

Item properties:

- *title*: text of what's to be done
- *doby*: date when it must be done by
- *completed*: has it been done?

The first page will list all our todos...

Coding time!

Databases

Eloquent

- Simple to use Active Record implementation
- Each table has a Model
- Conventions:
  - `User` model is connected to the `users` table
  - Each table has a primary key call `id`
  - `created_at` and `updated_at` columns exist on your table

Model

```
$ php artisan make:model User
Model created successfully.
```

Creates `App\User`:

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
}
```

Fetching data

Use `all()` to quickly grab all records:

```
$users = User::all();
foreach($users as $user) {
  echo $user->name;
}
```

or use the fluent interface to build your query:

```
$users = User::where('deleted', 0)
  ->orderBy('name', 'desc')
  ->take(20)
  ->get();
```

Fetching a single record

Use `find()` or `first()`:

```
$user = User::find(1);
// or
$rob = User::where('username', 'akrabat')->first();
```

Inserting/updating

Create a new Model instance and call `save()` on it:

```
$user = new User();
$user->username = 'geeh';
$user->save();
```

Calling `save()` on a retrieved model will update the database:

```
$user = User::where('username', 'geeh')->first();
$user->name = 'Gary';
$user->save();
```

Timesavers

Fetch a user or instantiate if it doesn't exist:

```
$user = User::firstOrNew(['username' => 'geeh']);
```

Update or create a new record if it doesn't exist:

```
$user = User::updateorCreate(
  ['username' => 'geeh'],     // where clause
  ['name' => 'Gary']          // data to update
);
```

Deleting

Find a Model and call `delete()` on it:

```
$user = User::where('username', 'geeh')->first();
$user->delete();
```

Or, call `destroy()` if you know the primary key:

```
User::destroy(2);
```

Or, `delete()` on a query result:

```
User::where('active', 0)->delete();
```

Soft delete

Create a `deleted_at` column in your table and Eloquent will *automatically* exclude these records from your queries:

```
class User extends Model
{
  use SoftDeletes;

  protected $dates = ['deleted_at'];
}
```

Relationships

Relationships are defined as *methods* on your model

Eloquent supports:

- One to one
- One to many
- Many to many

One to one relationships

A user has an address. Links via `addresses:user_id`.

```
class User extends Model
{
  public function address() {
    return $this->hasOne('App\Address');
  }
}

// usage:
$address = User::find(1)->address;
```

One to one relationships

Reverse:

```
class Address extends Model
{
  public function user() {
    return $this->belongsTo('App\User');
  }
}

// usage:
$user = Address::find(1)->user;
```

One to many relationships

A user has many talks. Links via `talks:user_id`.

```
class User extends Model
{
  public function talks() {
    return $this->hasMany('App\Talk');
  }
}

// usage:
$talks = User::find(1)->talks;
foreach ($talks as $talk) { /* … */ }
```

One to many relationships

Reverse:

```
class Talk extends Model
{
  public function user() {
    $this->belongsTo('App\User');
  }
}

// usage:
$user = Address::find(1)->user;
```

Many to many relationships

A user attends many events. Links via `events_users`.

```
class User extends Model
{
  public function events() {
    return $this->belongsToMany('App\Event');
  }
}

// usage:
$events = $user->events()->orderBy('event_date')->get();
foreach ($events as $event) { /* … */ }
```

Many to many relationships

Reverse is the same as definition for `User`.

```
class Event extends Model
{
  public function users() {
    return $this->belongsToMany('App\User');
  }
}

// usage:
$users = $e->users()->where('name', 'like', 'R%')->get();
foreach ($users as $user) { /* … */ }
```

Migrations

Store schema changes in code

Create a migration:

```
$ php artisan make:migration create_users_table
Created Migration: 2018_03_31_143419_create_users_table
```

Run migrations:

```
$ php artisan migrate
```

CreateUsersTable

up(): The changes you want to make

```
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users',
          function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }
```

CreateUsersTable

down(): Reverse your changes

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

Schema building

```
Schema::create('users', function (Blueprint $table) {
  $table->increments('id');
  $table->timestamps();
  $table->string('username', 50);
  $table->string('name', 100);
  $table->date('date_of_birth')->nullable();
  $table->boolean('active')->default(true);

  $table->unique('username');
});
```

Seeding

Seed you database with test data

Create a seeder:

```
$ php artisan make:seeder UsersTableSeeder
```

Run seeds:

```
$ php artisan db:seed
```

UsersTableSeeder

```
class UsersTableSeeder extends Seeder
{
  public function run()
  {
      DB::table('users')->insert([
        ['username' => 'akrabat', 'name' => 'Rob'],
        ['username' => 'geeh', 'name' => 'Gary'],
      ]);
  }
}
```

Coding time!

Form handling

Forms

- Write your form in HTML
- Validate in a separate action controller
- Supports Post-Redirect-Get pattern out of the box

Set up routes

One route for the form, one for processing it:

```
// routes/web.php:
Route::get('/user/edit/{id}', 'UserController@edit')
    ->name('user-edit');

Route::post('/user/doedit/{id}', 'UserController@doEdit')
    ->name('user-doedit');
```

A form

On the `/profile/edit/{id}` page:

```
<h1>Update your profile</h1>

<form action="{{ route('user-doedit', ['id' => $id]) }}"
  method="POST">
    <label for="name">Name</label>
    <input name="name"
      value="{{ old('name', $user->name) }}">

    <button>Update</button>
</form>
```

Validate request data

```
class UserController extends Controller
{
  public function doEdit(Request $request) {
    $data = $request->validate([
      'name' => 'required|max:100',
    ]);

    // The data is valid - save it

    return redirect()->route('user-profile');
  }
}
```

$request->validate()

- Validates the POSTed data against the rules array
- Returns the data on success
- Redirects back to the *previous* URL on failure (our form)

Rules:

- Format:

```
'fieldname' => 'rules|separated|by|pipes'
```

- There's lots of validation rules; Look them up!

Display errors

- The `$errors` variable is available in your template
- Use `$errors->any()` to test if you have any errors
- Access errors:
  - Collection: `$errors->all()`
  - Individual: `$errors->get('name')`

Coding time!

Authentication

Authentication

- Provided for you out-of-the-box:
  - `App\User` model
  - Controllers for registration, login, forgot password & reset password
  - View templates in `resources/views/auth`
  - Database migrations for users & password_resets tables
- Enable:

```
$ php artisan make:auth
$ php artisan migrate
```

Operations

Do we have a logged in user?

```
$isLoggedIn = Auth::check();
```

Retrieve the logged in user:

```
$user = Auth::user();
// or in a controller action:
$user = $request->user();
```

Protect routes:

Add the `auth` middleware to a route to restrict to logged in users

Coding time!

That's it!

Resources

- https://github.com/akrabat/laravel-bookshelf
- https://laravel.com
- https://laracasts.com
- https://laravel-news.com
- *Laravel: Up and Running* by Matt Stauffer

Thank you!

https://joind.in/talk/94c85

Rob Allen ~ @akrabat | Gary Hockin ~ @geeh