Introduction to Routing, Controllers, and more

April 1, 2015 6:59 PM

- php -S localhost:8888 -t public
  - Runs local server to run php files

- In app/http/ the routes.php file contains all of the ways the app should react when you access a route
- Route::get('/', 'WelcomeController@index');
  - This line says that at the root folder ('/') we want to look for the controller called WelcomeController (this is located in the App/Http/Controllers/ folder)
  - The @index states that we want to call the function index that resides in the WelcomeController class (which is its own file)

- A controller can return an html page, this is often done using the view() function
  - return view('pages/contact');
  - View() will look for the file called pages/contact in the App/resources/views/ folder
    - View will always by default search in the views folder
    - 'contact' is any file that starts with that word, we don't have to specify the suffix
    - e.g. contact.blade.php can be called two ways:
      - View('pages/contact.blade.php) is the same as view('pages/contact)

Passing Data to Views

April 1, 2015 7:09 PM

- In app/vendor/.env the line: "APP_ENV=LOCAL" shows error messages
  - Remember to not do this when in production
  - However it is okay to do so in development because we want to debug easily
- By default the file is called ".env.example" so that it doesn't take affect

- php artisan make:controller PagesController --plain
  - This creates a php file in the controller folder with the name PagesController
    - The --plain flag makes it so that default methods are not included

- Want views to be as dumb as possible
  - Give them all the data they need beforehand in order to create a page
- Laravel's blade engine gives us a nice syntax that compiles down to php
  - Escaped data: {{ $name }}
    - This echoes it out
  - Un-escaped data: {!! $name !!}
- Very important to escape the data
  - Prevents running html/javascript/etc..

```
//how to make arr in PHP quickly:
/*
$var1 = 'mario';
$var2 = 'luigi';
$arr = compact('var1', 'var2');
return view('pages.about', $arr);
*/
```

- Two ways of passing data to view:
  - Return view('pages.about')->with($var);
  - Return view('pages.about', $arr_of_data
- To pass more than one variable into a view use an array

Blade 101

April 2, 2015 10:07 AM

- In an html view the syntax @yield('section_name') lets you insert html from another view without needing to add all the html skeleton code
  - In another view you can insert html into the section in the layout page (the php/html file that contains the @yield() function)
  - Do that by typing:
    @extends('skeleton page name')
    - (before the '.') i.e., not app.blade.php, just 'app'

    @section('section_name')

- This searches the php/html page with the name 'skeleton page name' and looks for the section 'section_name' and inserts the html below the two @extends() and @section() functions

- @stop is a command that stops inserting the html code into the section stated previously
  - This allows us to inject html in to the skeleton in multiple areas in one vew

- Blade also allows us to use conditionals
  - @if (php comparison)
    Do this
    @else
    Do this instead
    @endif

- Also the command @unless which is just like saying if not
  - @foreach
  - @forelse
    - If you have some content for each one do this
    - Otherwise if you have none then do that
    - Useful for filtering through a collection and checking for an empty collection
      - @foreach
      - @endforeach

- Remember to always check if array is empty before printing HTML because we don't want to create an element that isn't being used
  - This causes the page to be cluttered

Environments and Configuration

April 4, 2015 6:50 PM

- Environment variables are stored in the .env file in the project root folder
- The /config/ folder contains php files for all sorts of configuration
    - Database configuration file is in here as well
- To set it up you have to specify, host, database, username, and password
    - Don't hardcode this in!
- Reference an environment variable instead:
    - Env('DB_PASSWORD', '')
        - Second string is a backup that is used if environment variable is not found
- Environment file is always ignored by git so don't have to worry about it being public if you publish the code
- When you want to choose a database to use update the default section in the /config/database.php file

Migrations

April 4, 2015 8:13 PM

- Database migration is one of laravel's most powerful features
  - Kind of like version control for your database
- Represent instructions to the database as a PHP class
  - Located in /database/migrations
- When you need to make a change to the schema you just rollback the migration, make the change and then re-run the migration
- To migrate the database run this in a shell:
  - php artisan migrate
- Never touch the migrations table
  - Laravel uses that to test which migrations to run, what needs to be rolled back and etc…
- Check the database (sqlite) by running this in a shell:
  - Sqlite3 <path to database file>
  - e.g. sqlite3 storage/database.sqlite
  - This launches the sqlite shell
    - .tables shows the tables
    - .schema shows the schema
- To roll back: php artisan migrate:rollback
  - This is like doing an undo
  - Now we can fix the schema
  - After fixing run: php artisan migrate

- To create a table:
  - php artisan make:migration <php_file_name> --create="<table_name>"
  - php artisan make:migration create_articles_table --create="articles"

- To make a change to an existing table you need to create a new table that extends it:
  - php artisan make:migration <php_file_name> --table="<table_name_that_is_being_extended>"
  - php artisan make:migration add_excerpt_to_articles_table --table="articles"
- Remember to drop the extra columns in the down() method
  - Need DBAL to call the dropColumn() method
  - Run this to install it: composer require doctrine/dbal
- Then just migrate it to make the changes

Eloquent

April 4, 2015 10:17 PM

- Eloquent == active record implementation
- One class that represents a single row from the associated database table
- E.g. an articles table would have an eloquent model called article
    - Users table ⇒ eloquent model would be called user
- php make:model <eloquent model name>
- Creates a php file in /apps/ folder with the name <eloquent model name>
    - This class will extend the class Model
        - This lets us use functions like save() and update()
- php artisan tinker
    - Kind of ssh's into a new shell lets us work with the laravel codebase
    - $article = new App\Article;
        - Create a new eloquent model
    - Can save this by typing: $article->save();
        - Can check if it worked by typing: App\Article::all();
- Carbon\Carbon::now();
    - Creates a timestamp of the current time
- $article = App\Article::find(1);
    - Returns an article whose id is 1
- $article = App\Article::where('body', 'Lorem ipsum')->get();
    - Gets a collection of articles where the body field equals 'Lorem ipsum'

- To allow fields to be mass fillable you need to declare a protected variable in the eloquent model php file with an array of fields that are allowed to be mass fillable
    - protected $fillable = ['title','body','published_at'];
- Now we can do this:
    - $article = App\Article::create(['title' => 'New Article', 'body'=>'New body', 'published_at'=> Carbon\Carbon::now()]);
        - This quickly creates an article that is saved instantly
        - We don't want fields like id or isUserAdmin to be mass fillable because then the user can manually access and change those fields
- $article->update(['body' => 'UPDATED AGAIN']);
    - Quickly change a field and save it into the database

Basic MVC Workflow

April 5, 2015 11:57 AM

- When just returning a collection from an eloquent model Laravel will automatically return it in JSON format

- Route::get('articles/{id}', 'ArticlesController@show');
  - The 'id' will be passed to the show function as a variable

- In the /.env file set debug to false so exceptions don't show up for user
  - Don't want users to see errors/exceptions

- 2 ways of making a URL on the fly:
  - <a href="{{action('ArticlesController@show', [$article->id])}}">
  - <a href="{{url('articles/', $article->id)}}">

Forms

April 5, 2015 1:54 PM

- Composer require illuminate/html
  - Pulls in a form builder
- Request::all();
  - Returns all the inputted fields in a form that is either POST or GET
- {!! Form::label('title', 'Title:') !!}
  - Sets the title attribute in a form to 'Title:'
- {!! Form::text('title', null, ['class' => 'form-control']) !!}
  - Gives the text box of id title a default value of null and sets its class to 'form-control'
- $article = Article::findOrFail($id);
  - If article is not found throw exception

Dates, Mutators and Scopes

April 5, 2015 7:24 PM

- Mutators lets us manipulate data before inserting into a database or after it is retrieved from one
- Scope is something that returns a subset of a query
- This makes dates a Carbon instance:
    - protected $dates = ['published_at'];

Forms Requests & Controller Validation

April 5, 2015 8:37 PM

- Form requests lets the user make some kind of request for your application
  - e.g. making a new article, logging in, signing up
  - PHP artisan make:request <php_file_name>
- In the controller class if the function type hints a form request then Laravel will automatically check the validation in the request php file created earlier

```
return [
        'title' => 'required|min:3',
        'body'   =>      'required',
        'published_at' => 'required|date' //has to be date type
    ];
```

  - Title has to be minimum 3 characters
  - Published_at has to be of type date
- If a validation fails then the function that type-hinted does not run

View Partials and Form Reuse

April 8, 2015 2:01 PM

- php artisan route:list
  - Lists all the routes specified in routes.php in the shell
- Route::resource('object_name', 'ObjectController');
  - Route::resource('articles', 'ArticlesController');
  - Automatically generates all the routes based on the methods defined in the corresponding controller php file

- {!! Form::model($article, ['method' => 'PATCH', 'action' => ['ArticlesController@update', $article->id]]) !!}
  - Model() binds an eloquent model to a form
  - Method is going to be PATCH since we are updating an eloquent model
  - Can either use action() or url(), it doesn't matter

- Laravel offers method injection
  - Can type hint a Request object in a function and laravel will instantiate it and pass it into the function

- Remember that type-hinting the request we can use a custom request class to enforce validation

- Partial Views:
  - Just include a php file to show some html code
  - @include('articles.form', ['submitButtonText' => 'Update Article'])
  - @include('blade.php file', ['variable name' => 'value for variable'])
  - @include('blade.php file')

Eloquent Relationships

April 8, 2015 3:43 PM

- Eloquent relationship:
  - A user can create many objects
    - A single object belongs to a user
  - e.g. one to many, many to many, one to one

- Laravel comes with a User.php class straight out of the box
  - In the user class use the hasMany function to define what the user has a lot of
    - public function articles() {
      return $this->hasMany('App\Article');
      }

- In the object class (e.g. article) define who the object belongs to
  - public function user() {
    return $this->belongsTo('App\User');
    }
  - Have to declare in the migration (i.e. database schema) to show which user the article belongs to

- Note: the function names (e.g. user and articles) is up to you to decide

- Keep in mind that articles() returns a hasMany return value
  - $user->articles()->get()->toArray() is the same as:
    - $user->articles->toArray()

- The belongsTo() and hasMany() relationship between the two eloquent models by having a similar key (e.g. foreign key user_id)

Easy Auth

April 11, 2015 12:57 PM

- In AuthController the trait AuthenticatesAndRegistersUsers allows you to autmatically create routes by doing:
  - Public function <get/post><Name>
  - e.g. public function getFoo creates a route that responds to <base uri>/foo
    - e.g. if we do
      Route::controllers([
      'auth' => 'Auth\AuthController',
      'password' => 'Auth\PasswordController'
      ]);
    - Then the base URI is 'auth' so it would get auth/Foo

- \Auth::user();
  - Gets authenticated user, if no user logged in returns null

- When doing Auth::user()->articles()->save($article) laravel automatically sets the user ID to the authenticated user id

- Remember to not do Auth::user()->articles;
  - This returns a collection
  - Auth::user()->articles(); lets us to continue chaining

Ogre Are Like Middleware

April 11, 2015 7:49 PM

- In the app/Requests/Kernel.php file the $middleware array will run for every single request
  - The $routeMiddleware is used when we want to attach middleware only to specific results

- Declare which middle ware to use in the constructor of the class
  - e.g. $this->middleware('auth', ['only' => 'create']);
    - Can also do $this->middleware('auth', ['except' => 'create'])

- Can also specify which middleware to use in the routes.php file
  - Route::get('about', ['middleware' => 'auth', uses => 'PagesController@about'])

- Php artisan down puts the application into maintenance mode
  - Basically shuts down the site which lets us update it
  - Then run php artisan up to re run the application

- Middleware performs like a decorator, handles requests

- Php artisan make:middleware <name>
  - Makes a middleware

Midterm Review

April 11, 2015 9:31 PM

- In the eloquent model files the php files directly in the app folder) the fillable array shows which variables are allowed to be mass assigned
    - This means things that the user can changed manually by them
- $dates in the eloquent model allows us to add additional columns that are allowed to be Carbon instances
    - Allows us to do chaining like: $article->published_at->format('Y-m-d');

- Query scopes allows us to assign a name to some kind of arbitrary where clause
    - Makes code look more clean
    - Naming convention: scope<Name>($query)
        - e.g. scopePublished($query)

- Mutators and accessors tell Laravel to do something behind the scenes if we set some kind of property
    - Naming convention: set<Name of property>Attribute
        - e.g. setPublishedAtAttribute()
    - Allows us to do a normal assignment but Laravel 'intercepts' and runs the mutator instead
        - e.g. $this->attributes['password'] = mcrypt($password);
        - Will get run when it intercepts $user->password = 'foobar';

Relationships:

- Can declare a belongsTo or hasMany relationship (others can be done as well)
    - Can do $user->articles or $article->user

Middleware:

- Middleware is like an onion
- Gives us a way to create any number of classes
- Receives a quest
    - Passes it along the chain of middlewares being used for a specific route
    - Or rejects it and redirects the user (or can do other things)
- Good for protecting some pages and making the pages private

Controllers:

- A controller is mostly responsible for receiving some kind of request and delegating as needed
    - Gets the request done and/or returns a response

Form Requests:

- Validates a form automatically by type-hinting a specific request (e.g. ArticleRequest)
    - Before the method is run it first validates the request

Blade:

- {!! Php code !!} runs the php code (unescaped)
- {{ php code }} prints the variable of php code (escaped)
- @yield('<section name>')
    - Lets other blade.php files inject code into that position
- @extends('<app_name>')
- @section('yield_section_name')
    - Inject this block until the @stop tag into any file with an @yield of that name
- @include('<php file name>')
    - Lets us use partial views so we can include them

    - Can also pass some values into the partial view

Environments:

- Specifies what kind of settings/values we want to use in the configuration files
    - Debugging stack trace is enabled here
- Included in the .gitignore file

Database Migrations:

- Simple way to define schema for database tables and build it
- Migrations also for modifying something after it has been deployed
    - Add another table, remember to drop it onDrop