Web Application Framework: Laravel

IF3110 – Web-based Application Development
School of Electical Engineering and Informatics
Institut Teknologi Bandung

Software Framework

- An abstraction in which common code providing generic functionality can be ***selectively overridden*** or specialized by user code providing specific functionality.
- A special case of software libraries in that they are ***reusable abstractions of code*** wrapped in a well-defined Application programming interface (API).

Framework vs Library

1. Inversion of Control
    - Program's flow of control is dictated by the framework, not the caller.
2. Extensibility
    - User can extend through a selective overriding or add specific functionality.
3. Non-modifiable Framework Code
    - Users are supposed to extend the framework, not modify the framework code.
4. Default Behaviour

Web Application Framework

- A set of tools to help designing web application development:
  - Front-end components
    - Form handling, authentication, templating, etc.
  - Back-end components
    - Routing, database ORM, etc.
  - Managing services, resources, APIs

Web Framework in Focus: Laravel

- PHP-based web framework
- Follows MVC architectural pattern
  - Model-View-Controller
- Released first time in 2011
- Licensed as free-open source software under MIT License
  - Non-copyleft, GPL compatible
- Documentation: https://laravel.com/docs/9.x

Laravel Project Skeleton: Root dir.

- app/
- bootstrap/
- config/
- database/
- public/
- resources/
- routes/
- storage/
- tests/
- vendor/

Laravel Project Skeleton: app dir.

- Broadcasting/
- Console/
- Events/
- Exceptions/
- Http/
- Jobs/
- Listeners/
- Mail/
- Models/
- Notifications/
- Policies/
- Providers/
- Rules/

Root Directory (1)

- app/ → Core code of the application, almost all of the classes will be here.
- bootstrap/ → Contains *app.php* file which bootstraps the framework.
  - Also has cache directory that contains framework-generated files for optimization. *File modification here is not necessary*.
- config/ → Contains application's configuration files.
- database/ → Contains database migration, model factories, and seeds.
  - Can also be used to hold SQLite database
- public/ → Contains *index.php*, entry point for all requests entering the application.
  - Also houses web assets (images, JavaScript, CSS)

Root Directory (2)

- resources/ → Contains the ***views*** components.
  - Also contains uncompiled CSS or JavaScript, and language files.
- routes/ → Contains all of the route definition.
  - *web.php* → Session state, CSRF protection, cookie encryption. Majority of the routes.
  - *api.php* → Stateless routes, requests entering through these routes are supposedly authenticated via tokens and no access to session state.
  - *console.php* → Closure based console commands.
  - *channels.php* → Event broadcasting channels.

Root Directory (3)

- storage/ → Contains logs, compiled Blade templates, file based sessions, file caches, and other files generated by the framework.
  - sub-directories: app (application generated files), framework (framework generated files and caches), logs (application log files)
- tests/ → Contains automated tests. Can be run using phpunit or php vendor/bin/phpunit
- vendor/ → Contains Composer dependencies.

App Directory

- Where majority of the application is housed.
- Namespaced under *app* and autoloaded by Composer
- Can be generated using make Artisan command

```
spl_autoload_register(function ($class) {
    $path = "classes/" . $class . ".php";
    if (file_exists($path)) {
        require $path;
    }
});
```

Some sub-directories

- Http/ → Contains the controllers, middleware, and form requests. All logic to handle entering requests are placed here.
- Console/ → Contains all of the custom Artisan (CLI) commands for the application. Generated using make:command
- Models/ → Contains all of the Eloquent model classes for the ORM

Routing

- Basic Routing: URI and closure (anonymous function)

```
use Illuminate\Support\Facades\Route;

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

Route Files

- All routes are defined in the route files, located in routes/ directory.
- The files are automatically loaded by App\Providers\RouteServiceProvider.
- The file routes/web.php defines routes for the web interface.
- The routes in routes/api.php are stateless.

Router Methods

- `Route::get($uri, $callback);`
- `Route::post($uri, $callback);`
- `Route::put($uri, $callback);`
- `Route::patch($uri, $callback);`
- `Route::delete($uri, $callback);`
- `Route::options($uri, $callback);`

CSRF Protection

- Any HTML forms pointing to POST, PUT, PATCH, DELETE that are defined in web.php should include a CSRF token field

```
<form method="POST" action="/profile">
    @csrf
    ...
</form>
```

Redirect Routes

- Simple redirect
  - `Route::redirect('/here', '/there');`
- Redirect with status code
  - `Route::redirect ('/here', '/there', 301);`
- permanentRedirect automatically return 301 status code
  - `Route::permanentRedirect('/here', '/there');`

View Routes

- A route may also returns a *view*

```
Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

- Restricted keywords for *view* routes parameter
  - view, data, status, headers

Route Parameters

- Required Parameters

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


Route::get('/posts/{post}/comments/{comment}', function ($postId, $commentId) {
    //
});
```

- Optional Parameters

```
Route::get('/user/{name?}', function ($name = null) {
    return $name;
});


Route::get('/user/{name?}', function ($name = 'John') {
    return $name;
});
```

Route Groups

- Enables attributes sharing across a large number of routes without having to define the attributes on individual route.
  - Subdomain Routing
  - Prefix

Subdomain Routing

- Subdomains may be assigned route parameters just like route URIs
- The subdomain may be specified by calling the domain method before defining the group

```
Route::domain('{account}.example.com')->group(function () {
    Route::get('user/{id}', function ($account, $id) {
        //
    });
});
```

Route Prefixes

- The prefix method may be used to prefix each route in the group of a certain URI

```
Route::prefix('admin')->group(function () {
    Route::get('/users', function () {
        // Matches The "/admin/users" URL
    });
});
```

Views

- Views provide a convenient way to place all of the HTML in separate files.
  - *Instead of returning entire HTML documents strings directly from routes.*
- Views separates controller/application logic and presentation logic.
- Stored in resources/views/ directory
- Created using *Blade* template

View Examples

- View component

```
<!-- View stored in resources/views/greeting.blade.php -->

<html>
    <body>
        <h1>Hello, {{ $name }}</h1>
    </body>
</html>
```

- Associating a view to a route

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

View Parameters

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

- First argument: name of the view file (greeting.blade.php) in resources/views/ directory
- Second argument: array of data that should be made available to the view.

Nested View Directory

- Views may also be nested within subdirectories of the resources/views directory.
- Accessing resources/views/admin/profile.blade.php

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

Blade

- Templating engine included in Laravel
- Allows using plain PHP code inside the template
- Compiled into PHP code and cached until modified

Blade: Display (1)

- Simple display

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

- JSON rendering

```
<script>
    var app = @json($array);

   <!-- var app = <?php echo json_encode($array)?>; -->

    var app = @json($array, JSON_PRETTY_PRINT);
</script>
```

Blade: Display (2)

- Display without render

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

@@json()
```

- Multiline using @verbatim

```
@verbatim
    <div class="container">
        Hello, {{ name }}.
    </div>
@endverbatim
```

Blade Directives: Conditional

- Common conditional

```
@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

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

- @isset and @empty

```
@isset($records)
    // $records is defined and is
not null...
@endisset

@empty($records)
    // $records is "empty"...
@endempty
```

Blade Directives: Loops

- For loop

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

- Foreach loop

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

- For-else

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

- While

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

Model: Eloquent ORM

- An Object-Relational Mapper that facilitate database interaction.
- Each database table is mapped into corresponding Model
- Models are kept in app\Models directory
- Every Models extend Illuminate\Database\Eloquent\Model class.
- A Model can be generated using make:model Artisan command
  - php artisan make:model Flight
- A Model can be generated to accommodate database migration
  - php artisan make:model Flight --migration

Model example (1)

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model{

    //

}

?>
```

Model example (2)

```
<?php
class Flight extends Model{
        ...
        protected $table = ‘my_flights’;
        protected $primaryKey = ‘flight_id’;
        protected $keyType = ‘string’;
        public $incrementing = false;
        public $timestamps = false;
}
?>
```

Model example (3)

```
<?php
class Flight extends Model{
      ...
      protected $connection = ‘sqlite’;
      protected $attributes = [
            ‘delayed’ => false,
      ];
}
?>
```

Model retrieval/query (1)

```
<?php
  use App\Models\Flight;
  foreach (Flight::all() as $flight){
    echo $flight->name;
  }
?>
```

Model retrieval/query (2)

```
<?php

  ...

  $flight = Flight::where('active',1)

                  ->orderBy('name')

                  ->take(10)

                  ->get();


  $flight->refresh();

?>
```

Model aggregate retrieval

```
<?php

   ...

   $count = Flight::where('active', 1)->count();

   $max = Flight::where('active', 1)->max('price');

?>
```

Model retrieval with creation

```
<?php
  ...

  $flight = Flight::firstOrCreate(
    ['name' => 'London to Paris'],
    ['delayed' => 1, 'arrival_time' => '11:30’]
  )

  $flight = Flight::firstOrNew(
    ['name' => Tokyo to Sydney'],
    ['delayed' => 1, 'arrival_time' => '11:30’]
  )
?>
```

Record Insertion

```
<?php
  ...

  $flight = new Flight;
  $flight->name = ‘Jakarta to Singapore’;
  $flight->save();

//alternatively

  $flight = Flight::create([
    ‘name’ => ‘Warsaw to Budapest’,
  ]);
//require specifying properties like ‘guarded’ or ‘fillable’

?>
```

Record Updates

```
<?php
  ...

  $flight = Flight::find(1);
  $flight->name = ‘Sankt-Peterburg to Novosibirsk’;
  $flight->save();

//mass update

  Flight::where(active,1)
          ->where(‘destination’, ‘Seoul’)
          ->update([‘delayed’, => 1]);

?>
```

Record Deletion

```
<?php
   ...

   $flight = Flight::find(1);
   $flight->delete();

//delete using query

   Flight::where(active,0)->delete();

?>
```

Relationship (1-to-1)

```php
<?php

  namespace App\Models;

  use Illuminate\Database\Eloquent\Model;

  class User extends Model{
    /**
    * Get the phone associated with the user.
    */

    public function phone(){
      return $this->hasOne(Phone::class);
    }
  }
?>
```

Relationship (1-to-1 inverse)

```php
<?php

  namespace App\Models;

  use Illuminate\Database\Eloquent\Model;

  class Phone extends Model{
    /**
    * Get the user that own the phone.
    */

    public function user(){
      return $this->belongsTo(User::class);
    }
  }
?>
```

Relationship (1-to-1 query)

```
<?php
   $phone = User::find(1)->phone;
?>
```

Relationship (1-to-many)

```php
<?php

  namespace App\Models;

  use Illuminate\Database\Eloquent\Model;

  class Post extends Model{
    /**
    * Get the comments of a blog post.
    */

    public function comments(){
      return $this->hasMany(Comment::class);
    }
  }
?>
```

Relationship (1-to-many inverse)

```php
<?php

  namespace App\Models;

  use Illuminate\Database\Eloquent\Model;

  class Comment extends Model{
    /**
    * Get the post that owns the comment.
    */

    public function post(){
      return $this->belongsTo(Post::class);
    }
  }
?>
```

Relationship (1-to-many query)

```
<?php

  user App\Models\Post;
  user App\Models\Comment;

  $comments = Post::find(1)->comments;
  foreach($comments as $comment){
    //
  }

  $comment = Post::find(1)->comments()
                          ->where('title', 'foo')
                          ->first();

  $comment = Comment::find(1);
  echo $comment->post->title;

?>
```

Relationship (Many-to-Many) [Table]

- Case example: user-role
  - a user can have many roles, a role can represent many users
- Table structure
  - users (id: integer, name: string)
  - roles (id: integer, name: string)
  - role_user (user_id: integer, role_id: integer)

Relationship (Many-to-many) [Model] {1}

```php
<?php

  namespace App\Models;

  use Illuminate\Database\Eloquent\Model;

  class User extends Model{
    /**
    * Roles of a user.
    */

    public function roles(){
      return $this->belongsToMany(Role::class);
    }
  }
?>
```

Relationship (Many-to-many) [Model] {2}

```php
<?php

  namespace App\Models;

  use Illuminate\Database\Eloquent\Model;

  class Role extends Model{
    /**
    * Users of a role.
    */

    public function users(){
      return $this->belongsToMany(User::class);
    }
  }
?>
```

Relationship (Many-to-many query)

```php
<?php
  use App\Models\User;

  $user = User::find(1);

  foreach ($user->roles as $role){
    //
  }

  $roles = User::find(1)->roles()->orderBy('name')->get();
?>
```

Controller

- Organizing behavior for request handling logic
  - *Instead of repeatedly defining it as an anonymous function*
- Stored in app/Http/Controllers/ directory
- Extends App/Http/Controllers/Controller
- Generated using Artisan command
  - php artisan make:controller UserController
- A controller action might be particularly complex, so it might be convenient to dedicate an entire controller class to a single action
  - Single Action Controllers

Controller example (1)

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;

class UserController extends Controller
{
    /**
     * Show the profile for a given user.
     *
     * @param  int  $id
     * @return \Illuminate\View\View
     */

    public function show($id)
    {
        return view('user.profile', [
            'user' => User::findOrFail($id)
        ]);
    }
}
```

Controller example (2)

```
...

use App\Http\Controllers\UserController;

Route::get('/user/{id}', [UserController::class, 'show']);

...
```

Single Action Controller (1)

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;

class ProvisionServer extends Controller
{
    /**
     * Provision a new web server.
     *
     * @return \Illuminate\Http\Response
     */

    public function __invoke()
    {
        // ...
    }
}
```

Single Action Controller (2)

```
<?php

use App\Http\Controllers\ProvisionServer;

Route::post('/server', ProvisionServer::class);
```

Review

- How to utilize web application framework in developing both front-end and back-end components.
- Some matters worth attention:
  - Routing
  - View (and Templating)
  - Model (and Database ORM)
  - Controller

To be further explored…

- How to adjust the environment configuration
- How to connect to the database and adjust some configuration
- Resource Controller
- Middleware
- Finer control on request, response, session, validation, error handling, and log management
- Utilizing ready-to-use packages
- How to install these things and manage their dependencies