

RAPPORT DE STAGE (31/05 AU 04/06)

SOMMAIRE

Préambule.....	2
Lundi 31 mai 2021.....	2
mardi 01 juin 2021	5
mercredi 02 juin 2021	8
jeudi 03 Juin 2021	9
vendredi 04 juin 2021	11

PREAMBULE

Avant toute chose, il est nécessaire d'indiquer que, pour des raisons de confidentialité, certains travaux ne pourront pas être illustrés (indiqués par un *) ou expliqués dans le détail.

LUNDI 31 MAI 2021

Activité du matin : Le maître de stage a validé l'implémentation des vues achevées en lien avec les méthodes de leur contrôleur CRUD (create, show, update, delete). Celle-ci-dessous par exemple permet de créer un nouvel avocat :

A droite, nous pouvons voir le rendu dans le navigateur (la mise en forme n'étant ici pas la priorité).

Il est à noter ici qu'il a fallu d'abord créer un contact, un état civil et un utilisateur avant de pouvoir créer un nouvel avocat, sans quoi cela résulte d'une violation de contrainte d'intégrité référentielle.



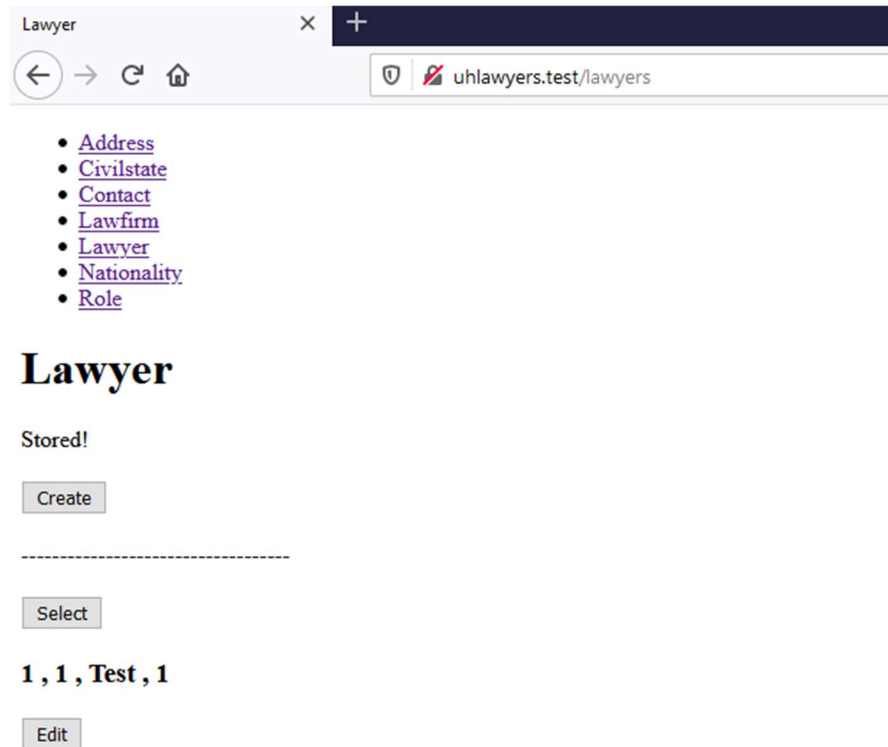
- [Address](#)
- [Civilstate](#)
- [Contact](#)
- [Lawfirm](#)
- [Lawyer](#)
- [Nationality](#)
- [Role](#)

Create a new lawyer

Contact_id	<input type="text" value="1"/>
Civilstate_id	<input type="text" value="1"/>
Barreau	<input type="text" value="Test"/>
User_id	<input type="text" value="1"/>
<input type="button" value="Create"/>	

Une fois le bouton « Create » cliqué, nous sommes renvoyés comme prévu sur la page d'index qui nous affiche la confirmation que l'avocat a bien été créé dans la base de données.

Nous pouvons aussi le voir s'afficher en dessous.



Activité de l'après-midi : La tâche suivante une fois la revue terminée est de s'initier au concept des tests unitaires. Un point particulier sur lequel le maître de stage a insisté est le fait d'être totalement autonome en ne s'aidant que de la documentation, pour nous mettre dans une situation professionnelle où l'on devrait apprendre par soi-même.

J'ai donc suivi des tutoriels^{1 2} qui m'ont permis de mieux comprendre le fonctionnement des tests unitaires et qui m'ont initié à PHPUnit et le composant Mockery.

Construire un test

Les trois étapes d'un test

Pour construire un test on procède généralement en trois étapes :

1. on initialise les données,
2. on agit sur ces données,
3. on vérifie que le résultat est conforme à notre attente.

Comme tout ça est un peu abstrait prenons un exemple. Remplacez le code de la méthode **testBasicTest** (peu importe dans quel dossier) par celui-ci :

```
public function testBasicTest()
{
    $data = [10, 20, 30];
    $result = array_sum($data);
    $this->assertEquals(60, $result);
}
```

¹ <https://laravel.sillo.org/cours-laravel-8-les-tests/>

² <https://laravel.com/docs/8.x/testing>

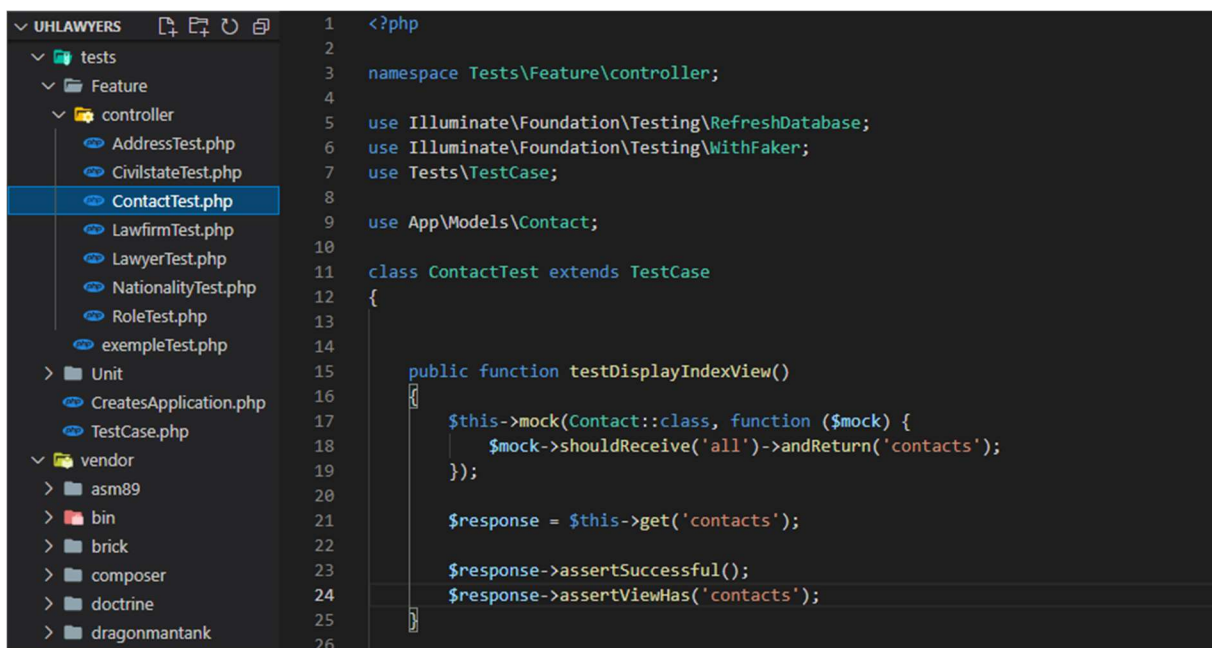
```
<?php
namespace Tests\Future;
use Tests\TestCase;
use App\Services\Livre;

class WelcomeControllerTest extends TestCase
{
    public function testIndex()
    {
        // Création Mock
        $this->mock(Livre::class, function ($mock) {
            $mock->shouldReceive('getTitle')->andReturn('Titre');
        });

        // Action
        $response = $this->get('welcome');

        // Assertions
        $response->assertSuccessful();
        $response->assertViewHas('titre', 'Titre');
    }
}
```

Je me suis inspiré de cette partie du tutoriel afin de réaliser mes tests unitaires de testDisplayIndexView pour chacun de mes fichiers test.



MARDI 01 JUIN 2021

Activité de la journée : Continuation de l'implémentation des tests unitaires. Pour nous aiguiller dans la bonne direction, le maître de stage nous a rapidement montré comment étaient écrits certains de ses tests unitaires.

L'utilisation du mocking n'était pas nécessaire non plus. J'ai donc réécrit mes tests unitaires déjà implémenté sans l'utiliser :

```

1  <?php
2
3  namespace Tests\Feature\controller;
4
5  use Illuminate\Foundation\Testing\RefreshDatabase;
6
7  use Illuminate\Foundation\Testing\WithFaker;
8  use Tests\TestCase;
9
10 use App\Models\Nationality;
11
12 class NationalityTest extends TestCase
13 {
14
15     public function testDisplayIndexView()
16     {
17
18         $response = $this->get(route('nationalities'));
19
20
21         $response->assertSuccessful();
22         $response->assertViewHas('nationalities');
23     }
24

```

J'ai aussi implémenté les autres tests :

```

46 public function testDisplayCreateView()
47 {
48     $response = $this->get(route('lawyers.create'));
49
50     $response->assertOk();
51     $response->assertViewIs('lawyers.create');
52 }
53 public function testStoreValidLawyer()
54 {
55     $data = [
56         'contact_id' => $this->contact->id,
57         'civilstate_id' => $this->civilstate->id,
58         'barreau' => 'test',
59         'user_id' => $this->user->id
60     ];
61
62     $response = $this->postJson(route('lawyers.store'), $data);
63
64     $response->assertRedirect(route('lawyers'));
65     $this->assertDatabaseHas('lawyers', $data);
66 }
67
68 public function testDisplayEdit()
69 {
70     $response = $this->get(route('lawyers.edit', ['id' => $this->lawyer->id]));
71
72     $response->assertOk();
73     $response->assertViewIs('lawyers.edit');
74     $response->assertViewHas('lawyer');
75 }
76
77 public function testDisplayShow()
78 {
79     $response = $this->get(route('lawyers.show', ['id' => $this->lawyer->id]));
80
81     //Assertions
82     $response->assertOk();
83     $response->assertViewIs('lawyers.show');
84     $response->assertViewHas('lawyer');
85 }
86
87
88
89
90
91

```

```

92 public function testUpdateValidLawyer()
93 {
94     $data = [
95         'contact_id' => $this->contact->id,
96         'civilstate_id' => $this->civilstate->id,
97         'barreau' => 'testchange',
98         'user_id' => $this->user->id
99     ];
100
101     $response = $this->postJson(route('lawyers.update', ['id' => $this->lawyer->id]), $data);
102
103     $response->assertRedirect(route('lawyers'));
104
105     $this->assertDatabaseHas('lawyers', $data);
106 }
107
108 public function testDeleteLawyer()
109 {
110     $data = [
111         'contact_id' => $this->contact->id,
112         'civilstate_id' => $this->civilstate->id,
113         'barreau' => $this->lawyer->barreau,
114         'user_id' => $this->user->id
115     ];
116
117     $response = $this->postJson(route('lawyers.destroy', ['id' => $this->lawyer->id]), $data);
118
119     $response->assertRedirect(route('lawyers'));
120
121     $this->assertDatabaseMissing('lawyers', $data);
122 }
123

```

Lors de cette implémentation de code j'ai rencontré plusieurs difficultés :

- Une erreur d'authentification avec le message « Expected status code 200 but received 419 » lié au middleware et à la protection csrf dans mes vues qui vérifient si l'utilisateur de l'application est bien authentifié. J'ai pu trouver la solution à l'aide de stackoverflow³ en ajoutant au début de mes fichiers test un WithoutMiddleware qui désactive ce processus d'authentification :

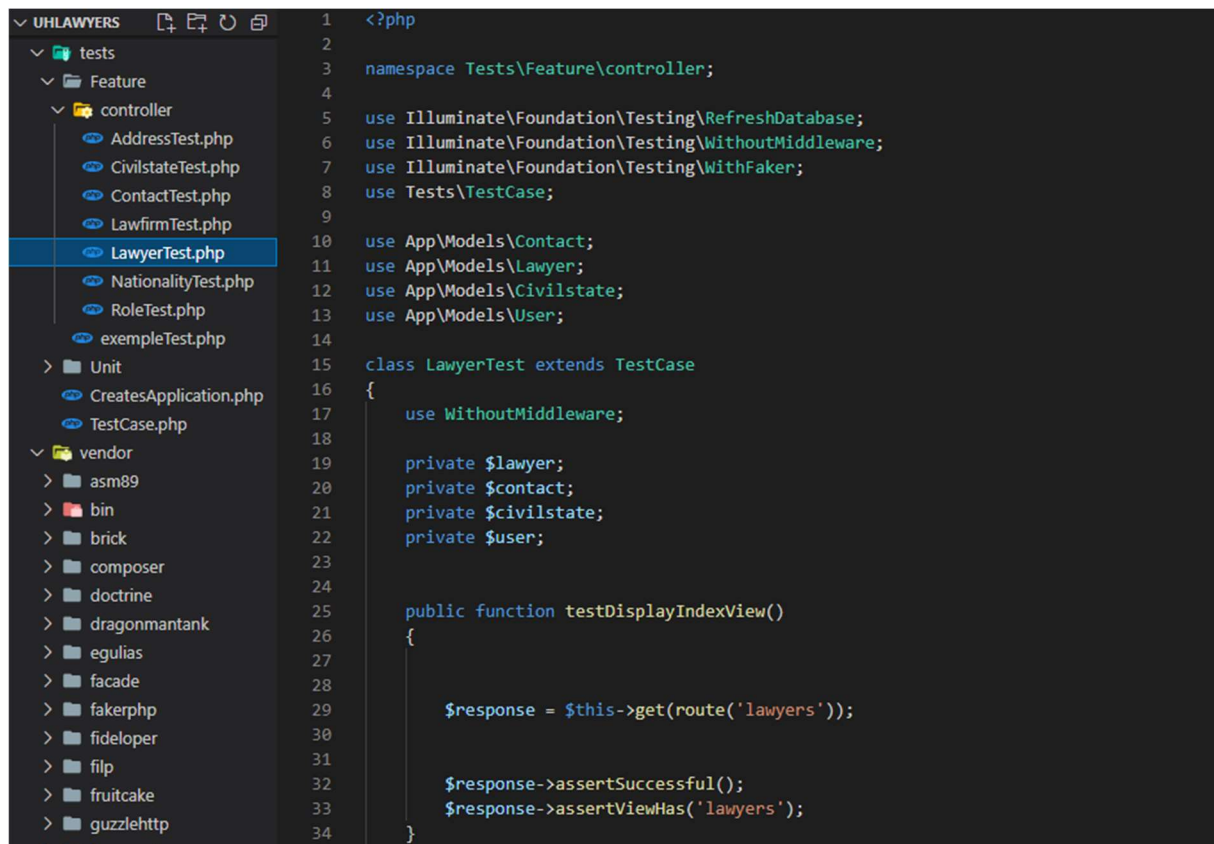
```

1 <?php
2
3 namespace Tests\Feature\controller;
4
5 use Illuminate\Foundation\Testing\RefreshDatabase;
6 use Illuminate\Foundation\Testing\WithoutMiddleware;
7 use Illuminate\Foundation\Testing\WithFaker;
8 use Tests\TestCase;
9
10 use App\Models\Lawyer;
11
12 class LawyerTest extends TestCase
13 {
14
15     use WithoutMiddleware;
16
17     public function testDisplayIndexView()
18     {
19
20
21         $response = $this->get(route('lawyers'));
22
23
24         $response->assertSuccessful();
25         $response->assertViewHas('lawyers');
26     }
27

```

³ <https://stackoverflow.com/questions/46325790/phpunit-expected-status-code-200-but-received-419-with-laravel/46326721>

- Une autre erreur, qui contenait « lawyer not defined », venait du fait que je n'avais pas initialisé mes variables objets. Dans l'exemple de LawyerTest.php il manquait donc ceci :



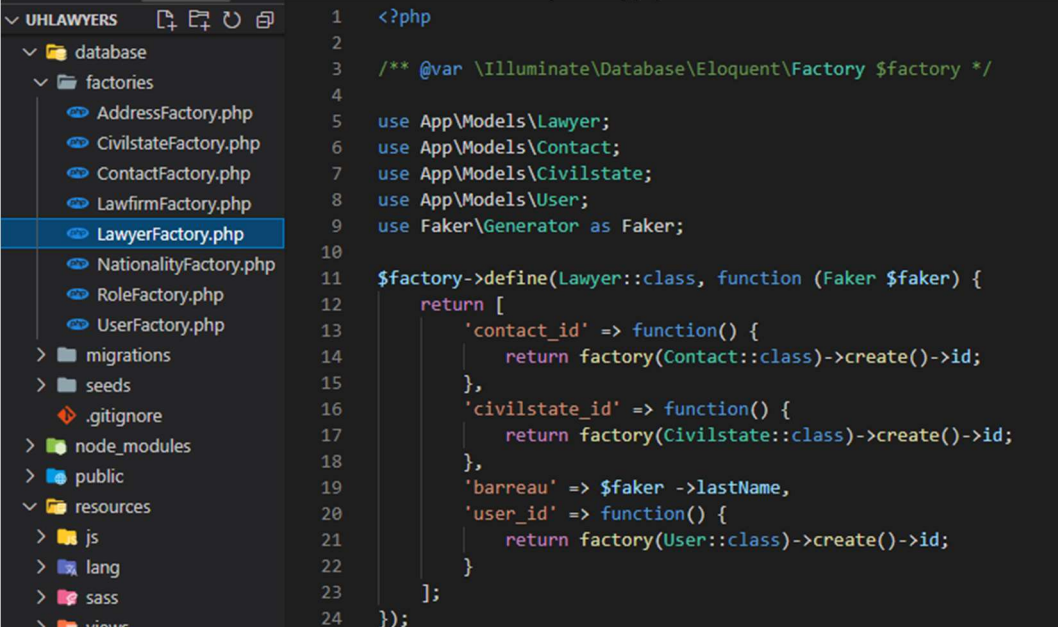
```
1 <?php
2
3 namespace Tests\Feature\controller;
4
5 use Illuminate\Foundation\Testing\RefreshDatabase;
6 use Illuminate\Foundation\Testing\WithoutMiddleware;
7 use Illuminate\Foundation\Testing\WithFaker;
8 use Tests\TestCase;
9
10 use App\Models\Contact;
11 use App\Models\Lawyer;
12 use App\Models\Civilstate;
13 use App\Models\User;
14
15 class LawyerTest extends TestCase
16 {
17     use WithoutMiddleware;
18
19     private $lawyer;
20     private $contact;
21     private $civilstate;
22     private $user;
23
24
25     public function testDisplayIndexView()
26     {
27
28
29         $response = $this->get(route('lawyers'));
30
31
32         $response->assertSuccessful();
33         $response->assertViewHas('lawyers');
34     }
```

- Une dernière erreur affichait « Trying to get property 'id' of non-object ». Ce n'est que le lendemain que j'ai pu trouver la solution.

MERCREDI 02 JUIN 2021

Activité de la journée : Finalisation des tests unitaires

En regardant la documentation Laravel pour les tests unitaires et certains forums, la solution a mon problème m'est apparue comme étant de créer des factories. En effet, mes objets étaient null puisqu'ils ne contenaient aucune donnée.

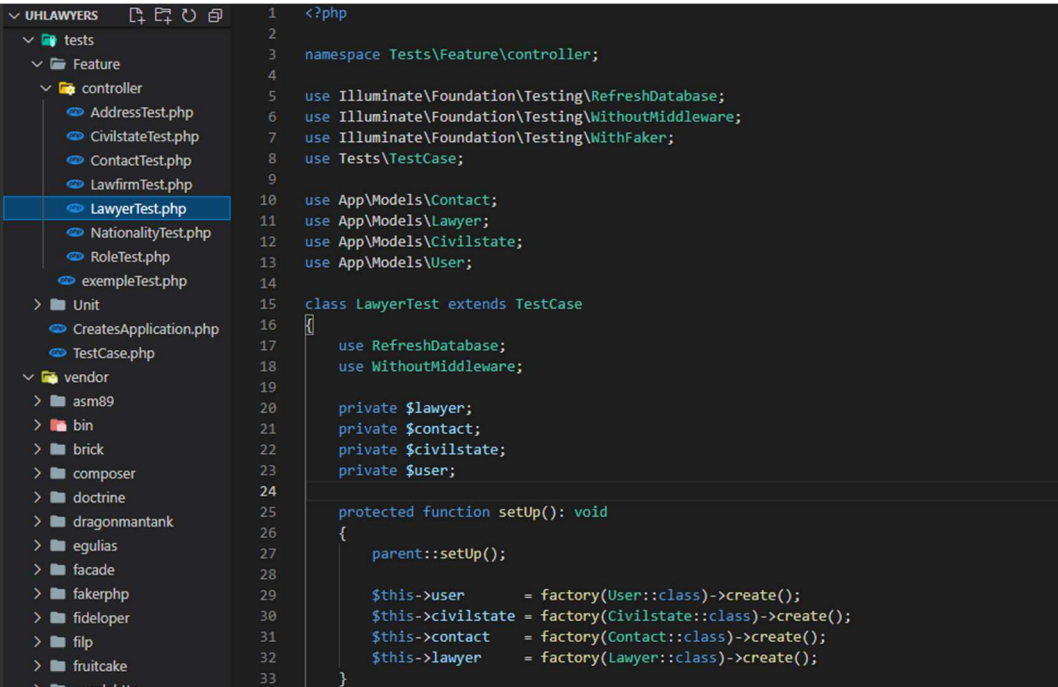


```

1  <?php
2
3  /** @var \Illuminate\Database\Eloquent\Factory $factory */
4
5  use App\Models\Lawyer;
6  use App\Models\Contact;
7  use App\Models\Civilstate;
8  use App\Models\User;
9  use Faker\Generator as Faker;
10
11 $factory->define(Lawyer::class, function (Faker $faker) {
12     return [
13         'contact_id' => function() {
14             return factory(Contact::class)->create()->id;
15         },
16         'civilstate_id' => function() {
17             return factory(Civilstate::class)->create()->id;
18         },
19         'barreau' => $faker->lastName,
20         'user_id' => function() {
21             return factory(User::class)->create()->id;
22         }
23     ];
24 });

```

Et dans le fichier LawyerTest.php :



```

1  <?php
2
3  namespace Tests\Feature\controller;
4
5  use Illuminate\Foundation\Testing\RefreshDatabase;
6  use Illuminate\Foundation\Testing\WithoutMiddleware;
7  use Illuminate\Foundation\Testing\WithFaker;
8  use Tests\TestCase;
9
10 use App\Models\Contact;
11 use App\Models\Lawyer;
12 use App\Models\Civilstate;
13 use App\Models\User;
14
15 class LawyerTest extends TestCase
16 {
17     use RefreshDatabase;
18     use WithoutMiddleware;
19
20     private $lawyer;
21     private $contact;
22     private $civilstate;
23     private $user;
24
25     protected function setUp(): void
26     {
27         parent::setUp();
28
29         $this->user = factory(User::class)->create();
30         $this->civilstate = factory(Civilstate::class)->create();
31         $this->contact = factory(Contact::class)->create();
32         $this->lawyer = factory(Lawyer::class)->create();
33     }
34 }

```

Un « use RefreshDatabase ; » a été ajouté pour que la base de données reparte de zéro, évitant ainsi des problèmes avec le assertDatabaseMissing() dans le testDeleteLawyer (en cas de doublon à cause des factories une donnée sera supprimée mais le doublon sera toujours là, ce qui fait échouer le test).

JEUDI 03 JUIN 2021**Activité de la journée :**

- Revue des tests unitaires
- Initiation à ReactJS en autonomie.

Pour commencer, je me suis rendu sur le site de ReactJS⁴ pour consulter la documentation.

The image shows two screenshots. The top screenshot is a browser window displaying the ReactJS tutorial page at <https://fr.reactjs.org/tutorial/tutorial.html>. The page title is "Black Lives Matter. Soutenez la Equal Justice Initiative." and it features a navigation bar with links to React, Docs, Tutorial, Blog, and Communauté. The main content area is titled "Qu'est-ce que React ?" and explains that React is a declarative JavaScript library for building user interfaces. It includes a code example for a `ShoppingList` component and a list of topics to be covered in the tutorial, such as "Qu'est-ce que React ?", "Examiner le code de départ", "Passer des données via les props", "Réaliser un composant interactif", "Outils de développement", "Finaliser le jeu", "Faire remonter l'état", "Pourquoi l'immuabilité est importante", "Fonctions composants", "Jouer tour à tour", "Déclarer un vainqueur", "Ajouter du voyage dans le temps", "Stocker un historique des mouvements", "Faire remonter l'état, encore", "Afficher les mouvements passés", "Choisir une key", "Implémenter le voyage dans le temps", and "Pour finir".

The bottom screenshot is a code editor showing the implementation of a Tic Tac Toe game. The code is written in JavaScript and uses the `React` library. It defines a `Square` component and a `Board` component. The `Square` component is a simple button that takes props and renders the value. The `Board` component is a class that extends `React.Component` and renders a 3x3 grid of squares. The code is as follows:

```
function Square(props) {  
  return (  
    <button className="square" onClick={props.onClick}>  
      {props.value}  
    </button>  
  );  
}  
  
class Board extends React.Component {  
  renderSquare(i) {  
    return (  
      <Square  
        value={this.props.squares[i]}  
        onClick={() => this.props.onClick(i)}  
      />  
    );  
  }  
}  
  
render() {  
  return (  
    <div>  
      <div className="board-row">  
        {this.renderSquare(0)}  
        {this.renderSquare(1)}  
        {this.renderSquare(2)}  
      </div>  
      <div className="board-row">  
        {this.renderSquare(3)}  
        {this.renderSquare(4)}  
        {this.renderSquare(5)}  
      </div>  
    </div>  
  );  
}
```

⁴ <https://fr.reactjs.org/tutorial/tutorial.html>

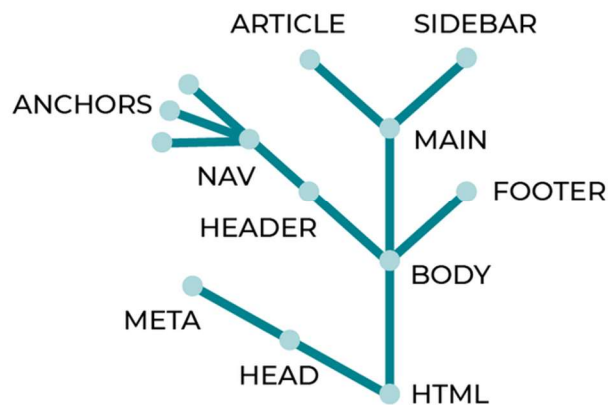
J'ai aussi revu certains concepts javascript comme le DOM⁵ :

Qu'est-ce que le DOM ?

Le DOM, qui signifie **Document Object Model** (c'est-à-dire "modèle d'objet de document", en français), est une **interface de programmation** qui est une représentation du HTML d'une page web et qui permet d'accéder aux éléments de cette page web et de les modifier avec le langage JavaScript.

Il faut voir le DOM comme un **arbre** où chaque élément peut avoir zéro ou plusieurs enfants, qui peuvent avoir eux-mêmes zéro ou plusieurs enfants, qui peuvent avoir zéro ou plusieurs... enfin je pense que vous comprenez le principe. 😊

Dans le DOM, on commence toujours par un élément racine qui est le point de départ du document : la balise `<html>`. Celle-ci a pour enfants les balises `<head>` et `<body>` qui ont donc un parent commun : la balise `<html>` ! Vous trouverez ensuite le contenu de votre page dans la balise `<body>` sous forme de liens, boutons, blocs, etc.



Le DOM est comme un arbre

⁵ <https://openclassrooms.com/fr/courses/5543061-ecrivez-du-javascript-pour-le-web/5543068-comprenez-ce-quest-le-dom>

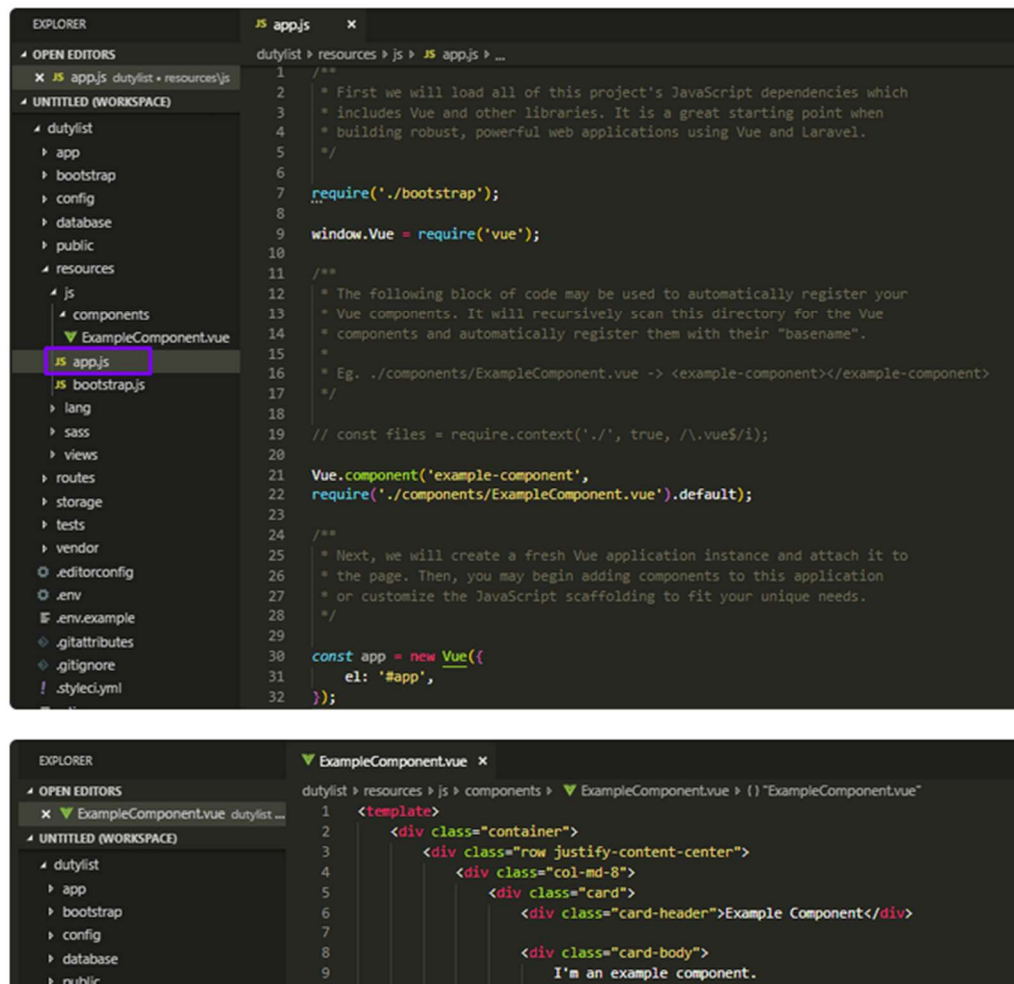
VENDREDI 04 JUIN 2021

Activité de la journée : Continuation de l'initiation à ReactJS en autonomie et implémentation de celui-ci dans les vues.

Pour cette tâche j'ai à ma disposition la documentation du site ReactJS mais aussi de nombreux tutoriels qui expliquent comment l'utiliser avec le framework Laravel⁶.

Using React in a Laravel

By default, Laravel applications contain an `ExampleComponent.vue` Vue component located in the `resources/js/components` directory. The `ExampleComponent.vue` file is an example of a single file Vue component which defines its JavaScript and HTML template in the same file. Single file components provide a very convenient approach to building JavaScript driven applications. The example component is registered in your `app.js` file:



Exemple du tutoriel

⁶ <https://dev.to/lvtdeveloper/using-react-in-a-laravel-application-8fp>