Experimentando entre el *frontend* y el *backend* con Laravel Livewire

Versión 1.0

Jorge I. Meza

jorge.meza@ucaldas.edu.co
jimezam@autonoma.edu.co

Tabla de contenidos

1. Preparación del proyecto base

Instalación y configuración de Laravel

Ubicarse el directorio donde se almacenará el proyecto.

```
$ cd ~/code
```

Crear un proyecto nuevo en Laravel llamado `ucaldas2020`.

```
$ composer create-project --prefer-dist laravel/laravel ucaldas2020
```

Ubicarse en el proyecto recién creado.

```
$ cd ucaldas2020
```

Instalar el módulo de autenticación (básico).

```
$ composer require laravel/ui
```

Compilar y publicar los recursos necesarios.

```
$ npm install && npm run dev
```

Generar el *scaffoling* de autenticación (*login*/*logout*, crear cuentas de usuario y recordar contraseñas).

```
$ php artisan ui bootstrap --auth
```

Editar el archivo de configuración con las variables de ambiente (`.env`) y realizar los siguientes ajustes.

```
APP_NAME="Jornadas de Ingenieria XVIII - Universidad de Caldas"

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=ucaldas2020
DB_USERNAME=ucaldas2020
DB_PASSWORD=secret
```

Creación de la base de datos

Conectarse a la base de datos utilizando el usuario administrador (`root`).

```
$ sudo mysql -u root
```

Ejecutar los siguientes comandos SQL.

```
CREATE DATABASE ucaldas2020;
CREATE USER ucaldas2020@localhost IDENTIFIED BY 'secret';
GRANT ALL PRIVILEGES ON ucaldas2020.* TO ucaldas2020@localhost;
FLUSH PRIVILEGES;
EXIT;
```

Creación de un usuario por defecto

Crear un nuevo *seeder* para el usuario por defecto.

```
$ php artisan make:seeder UserSeeder
```

Agregar a `database/seeders/UserSeeder.php` los siguientes `use`.

```
use Illuminate\Support\Facades\Hash;
use App\Models\User;
```

Y reemplazar la implementación del método `run` como se muestra a continuación.

```
    public function run()
    {
        User::create([
            'name' => 'Usuario de desarrollo',
            'email' => 'ingeniero@gmail.com',
            'password' => Hash::make('hola123'),
        ]);
    }
```

Activar el seeder recién creado agregando el siguiente fragmento de código al final del método `run` de `database/seeders/DatabaseSeeder.php`.

```
        $this->call([
            UserSeeder::class,
        ]);
```

Crear las entidades de la base de datos

```
$ php artisan migrate:fresh --seed
```

2. Ejemplo clásico: contador con vistas

En el archivo `routes/web.php` agregar las siguientes rutas `get` y `post` para el manejo del contador clásico.

```
use Illuminate\Http\Request;

// Mostrar la página con el contador
Route::get('/counter-classic/{count?}', function ($count = 0) {
    return view('counter-classic', compact('count'));
});

// Procesar el clic sobre uno de los botones
Route::post('/counter-classic', function (Request $request) {
    $count = $request->input('count');

    $mode = $request->has('increase') ? 1 : -1;

    $count += $mode;

    return redirect(url('counter-classic', [$count]));
});
```

Crear el archivo `resources/views/counter-classic.blade.php` con el siguiente contenido.

```
@extends('layouts.app')

@section('content')
<div class="container">
    <h1>Contador</h1>

    <div style="text-align: center">
        <form action="{{ url('counter-classic') }}" method="post">
            @csrf
            <button type="submit" name="increase">+</button>

            <input type="hidden" name="count" value="{{ $count }}">
            <h1>{{ $count }}</h1>

            <button type="submit" name="decrease">-</button>
        </form>
    </div>
```

```
</div>
@endsection
```

Para probar este ejemplo, acceda a la ruta `/counter-classic` desde un navegador web. Por ejemplo, `http://127.0.0.1:8000/counter-classic` si utiliza `artisan serve`.

3. Instalación del *framework* Laravel Livewire

Referencia https://laravel-livewire.com/docs/2.x/installation.

Instalar los paquetes asociados al *framework*.

```
$ composer require livewire/livewire
```

Cargar los *scripts* de Livewire en la vista principal. Para hacer esto, editar el archivo `resources/views/layouts/app.blade.php` y agregar los siguientes elementos.

1. Agregar al final de `<head>`

   ```
   @livewireStyles
   ```

2. Agregar al final de `<body>`

   ```
   @livewireScripts
   ```

Publicar el archivo de configuración de Livewire.

```
$ php artisan livewire:publish
```

Publicar los *assets* del *frontend* de Livewire.

```
$ php artisan livewire:publish --assets
```

Editar `composer.json` y agregar el siguiente elemento en la sección `"post-autoload-dump"` dentro de `"scripts"`.

```
    "@php artisan vendor:publish --force --tag=livewire:assets --
ansi"
```

4. Ejemplo Livewire: contador

Referencia https://laravel-livewire.com/docs/2.x/quickstart.

Crear una página para el ejemplo

En el archivo `routes/web.php` agregar la siguiente ruta.

```
Route::get('/counter-livewire', function () {
    return view('counter-livewire');
});
```

Crear el archivo `resources/views/counter-livewire.blade.php` con el siguiente contenido.

```
@extends('layouts.app')

@section('content')
<div class="container">
    <h1>Contador</h1>

    <livewire:counter>
</div>
@endsection
```

Crear el componente *Counter*

```
$ php artisan make:livewire counter
```

Cada componente se encapsula entre dos archivos:

1. Lógica de negocio

   `app/Http/Livewire/Counter.php`

2. Lógica de presentación

   `resources/views/livewire/counter.blade.php`

Establecer la lógica de negocio *inicial* del componente

Modificar el archivo `app/Http/Livewire/Counter.php` con el siguiente contenido.

```
class Counter extends Component
{
    public $count = 0;

    public function render()
    {
        return view('livewire.counter');
    }
}
```

Establecer la presentación *inicial* del componente

Modificar el archivo `resources/views/livewire/counter.blade.php` con el siguiente contenido.

```
<div style="text-align: center">
    <button>+</button>
    <h1>{{ $count }}</h1>
    <button>-</button>
</div>
```

Hacer funcionar los botones del contador

En `resources/views/livewire/counter.blade.php` actualizar los botones de la siguiente manera, incluyendo qué método del componente va a manejar su evento de clic.

```
<div style="text-align: center">
    <button wire:click="increment">+</button>
    <h1>{{ $count }}</h1>
    <button wire:click="decrement">-</button>
</div>
```

En `app/Http/Livewire/Counter.php` establecer la implementación de los métodos `increment` y `decrement` de la siguiente manera.

```
    public function increment()
```

```
    {
        $this->count++;
    }

    public function decrement()
    {
        $this->count--;
    }
```

Para probar este ejemplo, acceda a la ruta `/counter-livewire` desde un navegador web. Por ejemplo, `http://127.0.0.1:8000/counter-livewire` si utiliza `artisan serve`.

5. Ejemplo Livewire: cosas por hacer

Preparación previa

Entidades de persistencia

Crear la entidad para almacenar las tareas (`Task`), incluyendo modelo, migración y *factory*.

```
$ php artisan make:model Task -mf
```

Actualizar la migración ubicada en `database/migrations/2020_10_13_150750_create_tasks_table.php` con el siguiente contenido para el método `up`.

```
    public function up()
    {
        Schema::create('tasks', function (Blueprint $table) {
            $table->id();
            $table->string('description');
            $table->enum('state', ['opened', 'closed'])-
>default('opened');
            $table->timestamps();
        });
    }
```

Agregar en `app/Models/Task.php` la definición de los atributos editables en masa de la tarea como se muestra a continuación.

```
    protected $fillable = [
        'description',
        'state',
    ];
```

En `database/factories/TaskFactory.php` modificar la definición del *factory* de tareas de la siguiente manera.

```
    public function definition()
    {
        return [
            'description' => $this->faker->sentence(6, true),
            'state' => 'opened'
        ];
```

```
    }
```

En `database/seeders/DatabaseSeeder.php` agregar la creación de 10 tareas genéricas a través del *factory*, agregando el siguiente código al final de la implementación del método `run`.

```
        \App\Models\Task::factory(10)->create();
```

Correr las migraciones y las semillas para crear la nueva tabla e insertar las tareas de ejemplo.

```
$ php artisan migrate:fresh --seed
```

Crear una página para el ejemplo

En el archivo `routes/web.php` agregar la siguiente ruta.

```
Route::get('/todo-livewire', function () {
    return view('todo-livewire');
});
```

Crear el archivo `resources/views/todo-livewire.blade.php` con el siguiente contenido.

```
@extends('layouts.app')

@section('content')
<div class="container">
    <h1>Cosas por hacer</h1>

    <livewire:todo>
</div>
@endsection
```

Crear el componente

```
$ php artisan make:livewire todo
```

Este componente se encapsula entre los siguientes archivos:

1. Lógica de negocio: `app/Http/Livewire/Todo.php`

2. Lógica de presentación: `resources/views/livewire/todo.blade.php`

Mostrar las tareas

En `app/Http/Livewire/Todo.php` enviar la lista de tareas a la vista del componente a través del método `render` de la siguiente manera.

```
    public function render()
    {
        return view('livewire.todo', [
            'tasks' => Task::all()
        ]);
    }
```

Para poder utilizar la clase Task en el componente, se debe agregar la siguiente sentencia `use` al inicio del archivo.

```
use App\Models\Task;
```

En la vista del componente, `resources/views/livewire/todo.blade.php`, reemplazar el contenido del `<div>` con el siguiente.

```
    <ul>
        @foreach($tasks as $task)
            <li>
                {{ $task->description }}
            </li>
        @endforeach
    </ul>
```

Para probar este ejemplo, acceda a la ruta `/todo-livewire` desde un navegador web. Por ejemplo, http://127.0.0.1:8000/todo-livewire si utiliza `artisan serve`.

Agregar nuevas tareas

Para tener un lugar donde escribir la descripción de la nueva tarea, agregar el siguiente código en `resources/views/livewire/todo.blade.php`, después de la lista de tareas.

```
    <p><strong>> {{ $name }}</strong></p>
    <input wire:model="name" type="text" size="55">
```

En `app/Http/Livewire/Todo.php` definir el atributo `name` para guardar la descripción de la nueva tarea que digita el usuario en el `frontend` (sincronizados).

```
    public $name;
```

Para probar este ejemplo, acceda a la ruta `/todo-livewire` desde un navegador web. Por ejemplo, http://127.0.0.1:8000/todo-livewire si utiliza `artisan serve`.

Modificar `resources/views/livewire/todo.blade.php` para incluir un botón que permita agregar nuevas tareas.

```
    <button wire:click="addTask">Agregar</button>
```

Agregar el método `addTask` al componente en `app/Http/Livewire/Todo.php` con la siguiente implementación.

```
    public function addTask()
    {
        Task::create([
            'description' => $this->name
        ]);

        $this->name = "";
    }
```

Modificar `resources/views/livewire/todo.blade.php` para permitir que se agreguen las tareas con sólo presionar `ENTER`.

```
    <input wire:model="name" wire:keydown.enter="addTask" type="text"
size="55">
```

Para probar este ejemplo, acceda a la ruta `/todo-livewire` desde un navegador web. Por ejemplo, http://127.0.0.1:8000/todo-livewire si utiliza `artisan serve`.

Remover tareas

En `resources/views/livewire/todo.blade.php` para remover todas las tareas, agregar el siguiente botón después del botón de *agregar*.

```
  <button wire:click="removeAllTasks">Reiniciar</button>
```

Para eliminar las tareas individuales, agregar el siguiente botón después de la descripción de cada tarea (dentro del `foreach`).

```
                <button wire:click="removeTask({{ $task->id }})"
```

```
                        class="btn btn-sm
btn-outline-danger">x</button>
```

En `app/Http/Livewire/Todo.php` implementar los métodos `removeAllTasks` y `removeTask` de acuerdo con la siguiente implementación.

```
    public function removeAllTasks()
    {
        Task::truncate();
    }

    public function removeTask($id)
    {
        Task::destroy($id);
    }

    // public function removeTask(Task $task)
    // {
    //     $task->delete();
    // }
```

Para probar este ejemplo, acceda a la ruta `/todo-livewire` desde un navegador web. Por ejemplo, http://127.0.0.1:8000/todo-livewire si utiliza `artisan serve`.

Mensajes informativos

Como última modificación de este laboratorio, se van a agregar mensajes informativos que notifican al usuario la realización exitosa de tareas: agregar una tarea, remover una tarea o remover todas las tareas.

Para esto, en `resources/views/livewire/todo.blade.php` agregar el siguiente bloque de código al inicio donde se desee mostrar la notificación.

```
    @if(session()->has('information'))
    <div class="alert alert-info" role="alert">
        {{ session('information') }}
    </div>
    @endif
```

En `app/Http/Livewire/Todo.php` agregar los siguientes registros de mensajes *flash* en cada uno de los métodos correspondientes.

1. `addTask()`
   ```
   session()->flash('information', '¡Tarea agregada!');
   ```

2. `removeAllTasks()`
   ```
   session()->flash('information', '¡Lista reiniciada!');
   ```

3. `removeTask($id)`
   ```
   session()->flash('information', '¡Tarea removida!');
   ```

Para probar este ejemplo, acceda a la ruta `/todo-livewire` desde un navegador web. Por ejemplo, http://127.0.0.1:8000/todo-livewire si utiliza `artisan serve`.

Mejoras propuestas al ejemplo

Como ejercicio para practicar, el lector podría agregar las siguientes funcionalidades al ejemplo de *cosas por hacer*.

1. Permitir cerrar tareas.
2. Listar aparte las tareas pendientes (abiertas) de las ya finalizadas (cerradas).
3. Permitir que cada usuario maneje sus tareas de manera independiente.
4. Impedir que se gestionen tareas sin haberse autenticado previamente.

Recursos

- *Framework* Laravel Livewire
  https://laravel-livewire.com/
  - Documentación
    https://laravel-livewire.com/docs
  - *Screencasts*
    https://laravel-livewire.com/screencasts/installation
- *Framework* Laravel
  https://laravel.com/
  - Documentación
    https://laravel.com/docs/8.x

El código fuente de los ejemplos incluidos en este laboratorio puede consultarse en el siguiente repositorio.

- https://github.com/jimezam/UCaldas-Jornadas-2020