End-to-End Testing with Laravel Dusk:

* Create a Laravel application separately and configured Laravel Dusk.
* Write E2E tests for the following scenarios:
  * Create Product: Test the creation of a new product, including validation errors.
  * Read Products: Test viewing all products and a single product.
  * Update Product: Test updating a product's details, including validation errors.
  * Delete Product: Test deleting a product
* Ensure each test verifies that the database state changes as expected

End-to-end (E2E) testing with Laravel Dusk allows you to automate browser tests to ensure that your application behaves as expected. Below is a guide on how to set up Laravel Dusk for your Laravel application and write E2E tests for typical CRUD operations on a product.

Step 1: Set Up Your Laravel Application

Create a New Laravel Application:

1. If you haven't already, create a new Laravel application:
2. bash

```
composer create-project --prefer-dist laravel/laravel laravel-dusk-app
```

3. Navigate to the new Laravel project directory:
4. bash

```
cd laravel-dusk-app
```

5.

Install Laravel Dusk:

6. Install Laravel Dusk via Composer:
7. bash

```
composer require --dev laravel/dusk
```

8. After installing Dusk, run the install command:
9. bash

```
php artisan dusk:install
```

10.

Database Configuration:

11. Ensure your `.env` file is set up to connect to a test database.

1. Set Up Your Environment:
   Set up your database connection in the `.env` file:
2. plaintext

```
DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=laravel

DB_USERNAME=root

DB_PASSWORD=
```

3.

Step 2: Create the Product Model and Migration

1. Generate the Model and Migration:
   Generate a new model and migration for the Product:
2. bash

```
php artisan make:model Product -m
```

3.
4. Define the Product Schema:
   Open the generated migration file in `database/migrations/xxxx_xx_xx_create_products_table.php` and define the table structure:
5. php

```
public function up()

{

    Schema::create('products', function (Blueprint $table) {

        $table->id();

        $table->string('name');
```

```
            $table->decimal('price', 8, 2);

            $table->text('description')->nullable();

            $table->timestamps();

        });

    }
```

6.
7. Run the Migration:
   Run the migration to create the products table:
8. bash

```
php artisan migrate
```

9.

Step 3: Create the Controller with CRUD Methods

1. Generate a Controller:
   Generate a controller for the product with resource methods:
2. bash

```
php artisan make:controller ProductController --resource
```

3.
4. Define CRUD Operations:
   Open `app/Http/Controllers/ProductController.php` and define methods for each CRUD operation:
5. php

```
namespace App\Http\Controllers;


use App\Models\Product;

use Illuminate\Http\Request;


class ProductController extends Controller
```

```
{
    public function index()
    {
        $products = Product::all();
        return view('products.index', compact('products'));
    }

    public function create()
    {
        return view('products.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|string|max:255',
            'price' => 'required|numeric|min:0',
            'description' => 'nullable|string|max:500',
        ]);

        Product::create($request->all());

        return redirect()->route('products.index')->with('success', 'Product created successfully.');
    }
```

```php
public function show(Product $product)
{
    return view('products.show', compact('product'));
}

public function edit(Product $product)
{
    return view('products.edit', compact('product'));
}

public function update(Request $request, Product $product)
{
    $request->validate([
        'name' => 'required|string|max:255',
        'price' => 'required|numeric|min:0',
        'description' => 'nullable|string|max:500',
    ]);

    $product->update($request->all());

    return redirect()->route('products.index')->with('success', 'Product updated successfully.');
}
```

```
    public function destroy(Product $product)
    {
        $product->delete();

        return redirect()->route('products.index')->with('success', 'Product deleted successfully.');
    }
}
```

6.

Step 4: Create Views for Each CRUD Operation

Create the necessary Blade templates (`resources/views/products`) for listing, creating, showing, updating, and deleting products:

1. Index View (`index.blade.php`):
2. html

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

@section('content')
<div>
    <h2>Product List</h2>
    <a href="{{ route('products.create') }}">Create New Product</a>
    <ul>
        @foreach ($products as $product)
            <li>
                {{ $product->name }} - ${{ $product->price }}
                <a href="{{ route('products.show', $product) }}">View</a>
                <a href="{{ route('products.edit', $product) }}">Edit</a>
```

```
            <form action="{{ route('products.destroy', $product) }}" method="POST" style="display:
inline;">
                @csrf
                @method('DELETE')
                <button type="submit">Delete</button>
            </form>
        </li>
    @endforeach
  </ul>
</div>
@endsection
```

3.
4. Create View (`create.blade.php`):
5. html

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

@section('content')
<div>
  <h2>Create Product</h2>
  <form action="{{ route('products.store') }}" method="POST">
    @csrf
    <label for="name">Name:</label>
    <input type="text" name="name" required>

    <label for="price">Price:</label>
    <input type="text" name="price" required>
```

```
        <label for="description">Description:</label>

        <textarea name="description"></textarea>


        <button type="submit">Save</button>

    </form>

</div>

@endsection
```

6.
7. Edit View (`edit.blade.php`):
8. html

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


@section('content')

<div>

    <h2>Edit Product</h2>

    <form action="{{ route('products.update', $product) }}" method="POST">

        @csrf

        @method('PUT')

        <label for="name">Name:</label>

        <input type="text" name="name" value="{{ $product->name }}" required>


        <label for="price">Price:</label>

        <input type="text" name="price" value="{{ $product->price }}" required>


        <label for="description">Description:</label>
```

```
        <textarea name="description">{{ $product->description }}</textarea>

        <button type="submit">Update</button>
    </form>
</div>

@endsection
```

9.
10. Show View (`show.blade.php`):
11. html

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

@section('content')

<div>
    <h2>Product Details</h2>
    <p>Name: {{ $product->name }}</p>
    <p>Price: ${{ $product->price }}</p>
    <p>Description: {{ $product->description }}</p>
    <a href="{{ route('products.index') }}">Back to List</a>
</div>

@endsection
```

12.

Step 5: Set Up Routes

1. Define Routes in `web.php`:
   Add the following routes to `routes/web.php`:
2. php

```
use App\Http\Controllers\ProductController;
```

```
Route::resource('products', ProductController::class);
```

3.

Step 6: Implement Laravel Dusk for Testing

1. Install Laravel Dusk:
2. bash

```
composer require --dev laravel/dusk

php artisan dusk:install
```

3.
4. Write Dusk Tests:
   Create a test file:
5. bash

```
php artisan dusk:make ProductTest
```

6. Implement the test cases similar to those written in the earlier response for full CRUD operations.
7. Run Dusk Tests:
   Execute your Dusk tests:
8. bash

```
php artisan dusk
```

9.

This setup will give you a fully functional CRUD application for products within your Laravel application and enable you to perform end-to-end testing using Laravel Dusk to verify the correct behavior of your application. Adjust the view and controller logic to meet any additional requirement as needed for your specific use case.

Step 2: Write E2E Tests with Laravel Dusk

1. Create Product Test

Create a test class:

bash
```
php artisan dusk:make ProductTest
```

Open the `tests/Browser/ProductTest.php` and implement the test methods:

php
```php
namespace Tests\Browser;

use Laravel\Dusk\Browser;
use Tests\DuskTestCase;
use App\Models\Product;

class ProductTest extends DuskTestCase
{
    public function testCreateProduct()
    {
        $this->browse(function (Browser $browser) {
            $browser->visit('/products/create')
                    ->type('name', 'Test Product')
                    ->type('price', '100')
                    ->type('description', 'A test product description')
                    ->press('Save')
                    ->assertSee('Product created successfully.');

            $this->assertDatabaseHas('products', [
                'name' => 'Test Product',
            ]);
        });
    }

    public function testReadProducts()
    {
        $product = Product::factory()->create();

        $this->browse(function (Browser $browser) use ($product) {
            $browser->visit('/products')
                    ->assertSee($product->name)
                    ->clickLink($product->name)
```

```
                ->assertPathIs('/products/' . $product->id);
        });
    }

    public function testUpdateProduct()
    {
        $product = Product::factory()->create([
            'name' => 'Old Product',
        ]);

        $this->browse(function (Browser $browser) use ($product) {
            $browser->visit('/products/' . $product->id . '/edit')
                    ->type('name', 'Updated Product')
                    ->press('Update')
                    ->assertSee('Product updated successfully.');

            $this->assertDatabaseHas('products', [
                'id' => $product->id,
                'name' => 'Updated Product',
            ]);
        });
    }

    public function testDeleteProduct()
    {
        $product = Product::factory()->create();

        $this->browse(function (Browser $browser) use ($product) {
            $browser->visit('/products')
                    ->press('@delete-button-' . $product->id)
                    ->assertSee('Product deleted successfully.');

            $this->assertDatabaseMissing('products', [
                'id' => $product->id,
            ]);
        });
    }
}
```

Key Considerations

- Database Assertions: Each test uses database assertions (`assertDatabaseHas` or `assertDatabaseMissing`) to verify that the database state is as expected after the action.
- Dusk Selectors: Ensure that the buttons and links you've referenced in your tests (like `press('@delete-button-'.$product->id)`) correspond to elements in your views.
- Separate Environment: Laravel Dusk runs tests in a separate environment. Ensure your `phpunit.dusk.xml` is configured and your test database is set up properly.

Step 3: Run Your Dusk Tests

To execute your Dusk tests, run:

bash
php artisan dusk

Summary

This setup and the provided tests will help ensure that your product's CRUD functionality works as expected. Adjust the selectors and paths as needed to fit your specific project setup. Implement additional tests to cover more scenarios and validations as required by your application logic.

Update ChromeDriver:

- Run the following command to update ChromeDriver:
- bash
- php artisan dusk:chrome-driver

Running a Specific Test Method:

To run a specific method in a test class using Laravel Dusk, you can use PHPUnit's filtering capability. Here's how you can run your specific test method:

Navigate to your application's root directory in the terminal and run the following command, replacing `ProductTest` with your actual test class name if it's different:

```
php artisan dusk tests/Browser/ProductTest.php --filter testCreateProduct
```
This specific command will run only the `testCreateProduct` method within the `ProductTest` class.

```
php artisan dusk tests/Browser/CreateProductTest
.php --filter testCreateProduct
```

####

The error message you are encountering — `SessionNotCreatedException` — indicates that there is a mismatch between the version of ChromeDriver and the version of Google Chrome that is currently installed on your system. Specifically, your ChromeDriver supports Chrome version 128, while your installed Chrome version is 123.0.6312.105.

Steps to Resolve the Issue

Update Google Chrome:

1. Since you're using ChromeDriver version 128, it would be best to update Google Chrome to the latest version that is compatible with ChromeDriver 128.
   To update Google Chrome, you can follow these commands:
2. bash

```
sudo apt update
sudo apt --only-upgrade install google-chrome-stable
```

3. After the update, verify the version of Google Chrome:
4. bash

```
google-chrome --version
```

5.

Downgrade ChromeDriver (As an Alternative):

6. If for some reason you cannot update Google Chrome, you can also downgrade ChromeDriver to match your installed version of Chrome:
   To downgrade ChromeDriver, follow these steps:
   - First, stop any running instances of ChromeDriver.
   - Then, download the version of ChromeDriver that is compatible with your current Chrome version (123). You can find the versions here: ChromeDriver Downloads.
7. For example:

8. bash

```
wget https://chromedriver.storage.googleapis.com/123.0.1234.56/chromedriver_linux64.zip
unzip chromedriver_linux64.zip
sudo mv chromedriver /usr/local/bin/chromedriver
sudo chmod +x /usr/local/bin/chromedriver
```

9.

Re-run Dusk Tests:

10. After updating Google Chrome or downgrading ChromeDriver, you should be able to execute your Dusk tests again:
11. bash

```
php artisan dusk tests/Browser/ProductTest.php --filter testCreateProduct
```

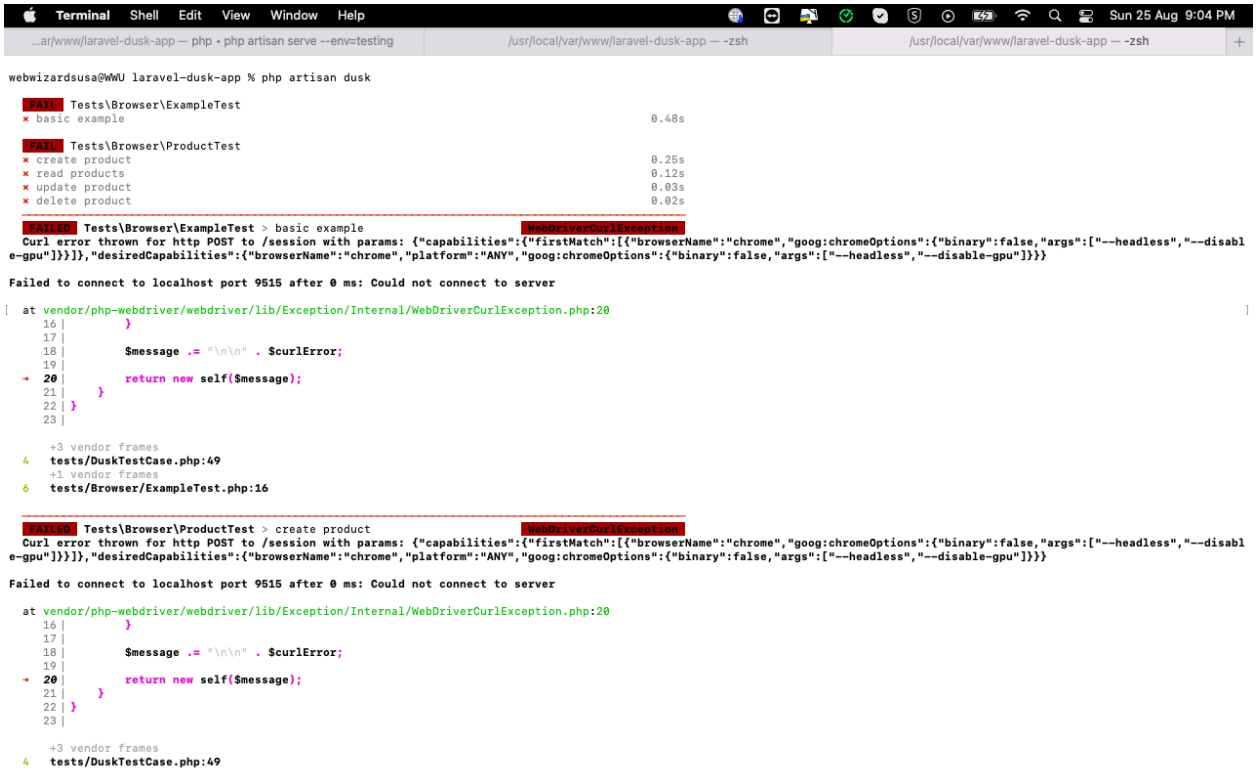
12.

Summary

- Updating your Chrome browser is the preferred solution, as it will ensure you have the latest security patches and features.
- Downgrading ChromeDriver is a workaround in case you cannot upgrade the browser.
- Ensure that after making any changes, you confirm the versions of Chrome and ChromeDriver are compatible before proceeding with your tests.

Let me know how it goes!

Php artisan serve –env=testng



webwizardsusa@WWU laravel-dusk-app % php artisan dusk

FAIL Tests\Browser\ExampleTest

× basic example 0.48s

FAIL Tests\Browser\ProductTest

× create product 0.25s

× read products 0.12s

× update product 0.03s

× delete product 0.02s

FAILED **Tests\Browser\ExampleTest** > basic example WebDriverCurlException

Curl error thrown for http POST to /session with params:
{"capabilities":{"firstMatch":[{"browserName":"chrome","goog:chromeOptions":{"binary":false,"args":["--headless","--disable-gpu"]}}]},"desiredCapabilities":{"browserName":"chrome","platform":"ANY","goog:chromeOptions":{"binary":false,"args":["--headless","--disable-gpu"]}}}

Failed to connect to localhost port 9515 after 0 ms: Could not connect to server

at vendor/php-webdriver/webdriver/lib/Exception/Internal/WebDriverCurlException.php:20

```
16 |     }
17 |
18 |     $message .= "\n\n" . $curlError;
19 |
20 |     return new self($message);
21 |   }
22 | }
23 |
```

+3 vendor frames

4 **tests/DuskTestCase.php:49**
+1 vendor frames
6 **tests/Browser/ExampleTest.php:16**

FAILED **Tests\Browser\ProductTest** > create product **WebDriverCurlException**

Curl error thrown for http POST to /session with params:
{"capabilities":{"firstMatch":[{"browserName":"chrome","goog:chromeOptions":{"binary":false,"args":["--headless","--disable-gpu"]}}]},"desiredCapabilities":{"browserName":"chrome","platform":"ANY","goog:chromeOptions":{"binary":false,"args":["--headless","--disable-gpu"]}}}
Failed to connect to localhost port 9515 after 0 ms: Could not connect to server

at vendor/php-webdriver/webdriver/lib/Exception/Internal/WebDriverCurlException.php:20

```
16 |     }
17 |
18 |     $message .= "\n\n" . $curlError;
19 |
➜ 20 |     return new self($message);
21 |   }
22 | }
23 |
```

+3 vendor frames
4 **tests/DuskTestCase.php:49**
+1 vendor frames
6 **tests/Browser/ProductTest.php:13**

FAILED **Tests\Browser\ProductTest** > read products **BadMethodCallException**

Call to undefined method App\Models\Product::factory()

at vendor/laravel/framework/src/Illuminate/Support/Traits/ForwardsCalls.php:67

```
63 |   * @throws \BadMethodCallException
64 |   */
65 |  protected static function throwBadMethodCallException($method)
66 |  {
➜ 67 |     throw new BadMethodCallException(sprintf(
68 |       'Call to undefined method %s::%s()', static::class, $method
69 |     ));
70 |  }
71 | }
```

+4 vendor frames
5 **tests/Browser/ProductTest.php:29**

FAILED **Tests\Browser\ProductTest** > update product **BadMethodCallException**

Call to undefined method App\Models\Product::factory()

at vendor/laravel/framework/src/Illuminate/Support/Traits/ForwardsCalls.php:67

```
63 |   * @throws \BadMethodCallException
64 |   */
65 |  protected static function throwBadMethodCallException($method)
66 |  {
➜ 67 |     throw new BadMethodCallException(sprintf(
68 |       'Call to undefined method %s::%s()', static::class, $method
69 |     ));
```

```
  70 |     }
  71 | }
   +4 vendor frames
5  tests/Browser/ProductTest.php:41
```

```
  FAILED  Tests\Browser\ProductTest > delete product                BadMethodCallException

  Call to undefined method App\Models\Product::factory()

  at vendor/laravel/framework/src/Illuminate/Support/Traits/ForwardsCalls.php:67
     63 |     * @throws \BadMethodCallException
     64 |     */
     65 |    protected static function throwBadMethodCallException($method)
     66 |    {
  ➜  67 |        throw new BadMethodCallException(sprintf(
     68 |            'Call to undefined method %s::%s()', static::class, $method
     69 |        ));
     70 |    }
     71 | }
      +4 vendor frames
  5  tests/Browser/ProductTest.php:60

  Tests:    5 failed (0 assertions)
  Duration: 1.14s

webwizardsusa@WWU laravel-dusk-app %
```