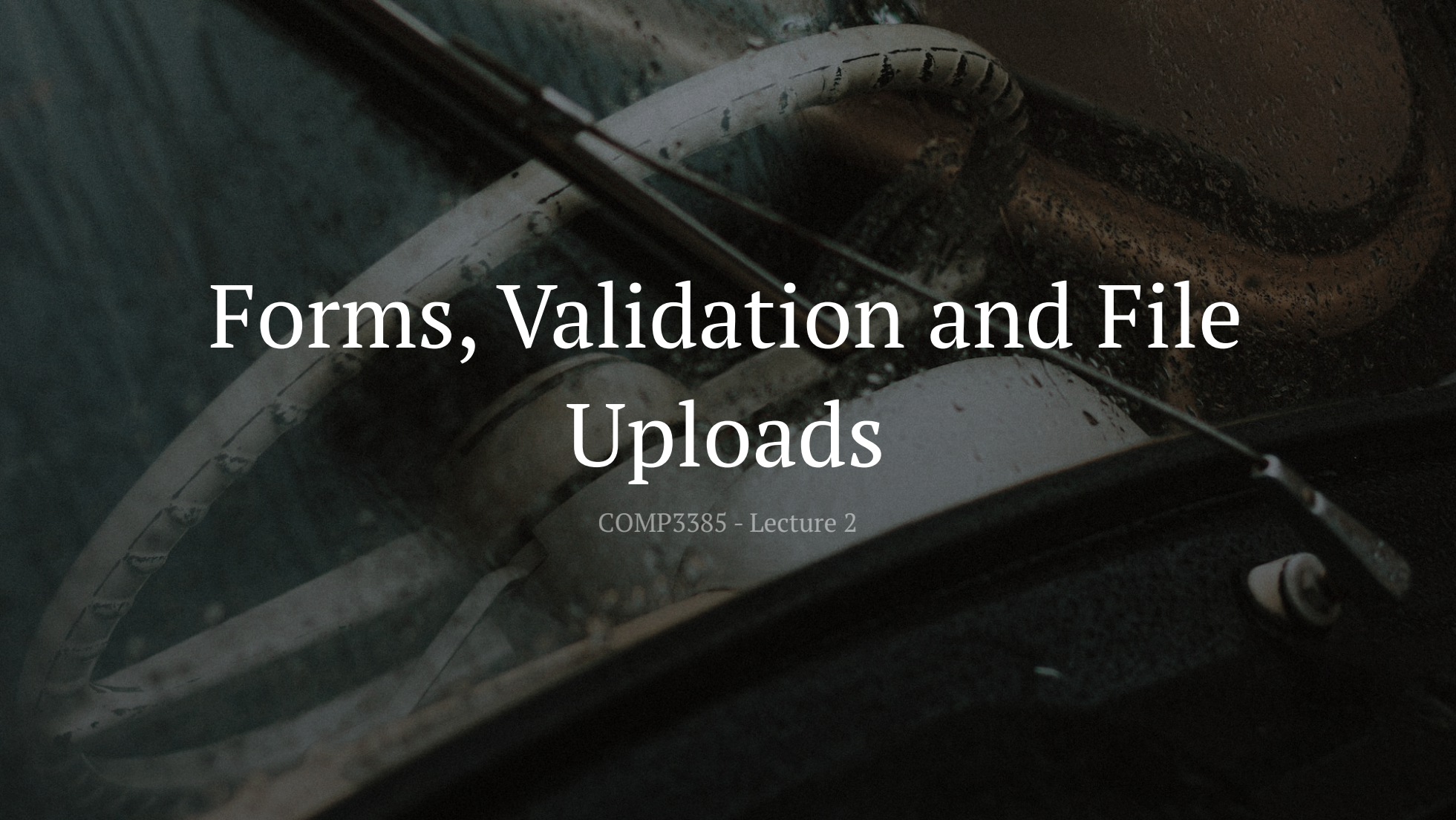

Forms, Validation and File Uploads

COMP3385 - Lecture 2

Forms

HTML Forms are one of the main points of interaction between a user and a web site or application. They allow users to send data to the web server to be processed.

```
<form action="/contact" method="post">
    <div>
        <label for="name">Full Name:</label>
        <input type="text" id="name" name="full_name" />
    </div>
    <div>
        <label for="mail">E-mail:</label>
        <input type="email" id="mail" name="email" />
    </div>
    <div>
        <label for="msg">Message:</label>
        <textarea id="msg" name="message"></textarea>
    </div>
    <button type="submit">Send Message</button>
</form>
```

Getting Form Data

To access form data in Laravel, we need an instance of the current HTTP request via dependency injection, you should type-hint the `Illuminate\Http\Request` class on your route closure or controller method.

You can then use the `input` method to retrieve the user input from the form.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;

class ContactController extends Controller
{
    /**
     * Send an email from your contact form.
     */
    public function send(Request $request): RedirectResponse
    {
        $name = $request->input('full_name');
        $email = $request->input('email');
        $message = $request->input('message');

        // Send message...

        return redirect('/');
    }
}

```

You can also pass a default value as the 2nd argument to the `input` method. If the value from the form is not present, the default value will be used instead.

```
$name = $request->input('full_name', 'Serena Williams');
```

And if you need to retrieve a value from the query string of a URL, then you can use the `query` method

```
$name = $request->query('id', '1');
```

Form Validation

Laravel comes bundled with a wide range of validation rules that you can use to check input values submitted by your users in your application.

Some of those built-in validation rules are:

- Alpha and Alpha Numeric
- Integer, Digits, Digits Between, Decimal
- String
- Date
- Email
- URL
- UUID
- JSON
- IP Addresses and MAC addresses
- Min and Max Digits
- Required
- Starts with and Ends with
- Regular Expressions
- and many more

```
public function send(Request $request): RedirectResponse
{
    $validated = $request->validate([
        'full_name' => 'required|max:100',
        'email' => 'required|email'
        'message' => 'required',
    ]);

    // Send Email

    return redirect('/');
}

```

To display your error messages in your `.blade.php` templates you could do the following:

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

Or next to a particular input field you could do the following to display an individual error message:

```
<input id="full-name"
    type="text"
    name="full_name"
    class="@error('full_name') is-invalid @enderror">

@error('full_name')
    <div class="alert alert-danger">{{ $message }}</div>
@enderror
```

Whenever a validation error occurs, Laravel automatically redirects you back to the form itself. If you need to access the old value or repopulate the form with the values initially entered by the user you can use the `old` method.

```
$full_name = $request->old('full_name');
```

or in a Blade template:

```
<input type="text" name="full_name" value="{{ old('full_name') }}">
```

If you find that you have more complex validation scenarios, you can opt to create a "form request". These are essentially custom request classes that has their own validation or authorization logic.

You can create a Form Request class using the `make:request` Artisan CLI command

```
php artisan make:request ContactRequest
```

The request class will look something like this:

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class ContactRequest extends FormRequest
{
    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
     */
    public function rules(): array
    {
        return [
            'full_name' => 'required|max:100',
            'email' => 'required|email',
            'message' => 'required',
        ];
    }
}
```

And in your controller you would update it to be:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Requests\ContactRequest;
use Illuminate\Http\RedirectResponse;

class ContactController extends Controller
{
    public function send(ContactRequest $request): RedirectResponse
    {
        // Retrieve the validated input data...
        $validated = $request->validated();

        // Send Email

        return redirect('/');
    }
}

```

CSRF Protection

As you know, Cross-site request forgeries (CSRF) are a type of security vulnerability in web applications whereby unauthorized commands are performed on behalf of an authenticated user who is currently logged in.

By using either the `@csrf` Blade directive or the `{{ csrf_token() }}` helper function you can add CSRF protection to your forms.

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

    <!-- Equivalent to... -->
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
</form>
```

Uploading Files

Files are a special case with HTML forms. They are binary data—or considered as such—where all other data is text data. Because HTTP is a text protocol, there are special requirements to handle binary data.

You need three things for File Uploads to work

- The `method` attribute must be set to POST because file content can't be put in a URL parameter.
- The value of the `enctype` attribute must be `multipart/form-data`. This is because the ifle will be split into multiple parts (chunks). One for each file, plus one for the text of the form body that may be sent with them.
- You also need an input field of the `type="file"`.

```
<form method="post" action="/upload-photo" enctype="multipart/form-data">
    <input type="file" name="photo" />
    <button>Send the file</button>
</form>
```

Typical File Upload Process

- Get File (via a form)
- Save File (to a location on the filesystem)
- Serve File (via a method in the application)

Now how do we get access to the file related data?

You can retrieve uploaded files from the `Request` isntance using the `file` method.

```
$file = $request->file('photo');
// or
$file = $request->photo;
```

When you need to get the file's path and extension, you can use the `path` and `extension` methods.

```
$path = $request->photo->path();
// and
$extension = $request->photo->extension();
```

If you need to get the file's Original name and extension, you can use the `getClientOriginalName` and `getClientOriginalExtension` methods.

```
$filename = $request->photo->getClientOriginalName();
// and
$extension = $request->photo->getClientOriginalExtension();
```

Then to store your uploaded files, you can use the `store` method. The first parameter is for the path where the file should be stored.

```
$request->photo->store('images')
```

Note: By default a unique ID will be generated to serve as the filename adn the file will be stored in your `storage/app` folder.

If you would like to specify your own filename then you can use the `storeAs` method.

```
$request->photo->storeAs('images', 'my-photo.jpg');
```

Securing File Uploads

Some things you may want to consider to help secure your application when doing file uploads are:

- Require a login before accessing the file upload form
- Limit the types of files a user is allowed to upload (possibly by it's extension or mime type e.g. .gif, .jpeg, .png)
- Check/Limit the file size allowed to be uploaded
- You can also modify the filename when you save it to the file system. Also ensure you escape or remove any special characters (e.g. "/" that could reprsent a file path. "../../../home/username/.bashrc")
- You can also save your files somewhere outside of your web server document root.

To do validation on a file upload, you could do the following:

```php
namespace App\Http\Controllers;

// ...other imports
use Illuminate\Validation\Rules\File;

class ContactController extends Controller
{
    public function send(Request $request): RedirectResponse
    {
        $validated = $request->validate([
            'photo' => [
                'required',
                File::types(['png', 'jpg', 'webp'])
                    ->min(1024)
                    ->max(12 * 1024),
            ]
        ]);

        // Save Attachment
        $request->photo->storeAs('images', 'test.jpg');

        return redirect("/");
    }
}

```

Resources

- HTTP Requests - https://laravel.com/docs/10.x/requests
- CSRF - https://laravel.com/docs/10.x/csrf
- Laravel Form Validation - https://laravel.com/docs/10.x/validation
- List of Laravel Validation Rules - https://laravel.com/docs/10.x/validation#available-validation-rules