1. SQL injection: if you use Eloquent queries these will keep you safe. But you will be vulnerable if you use DB::raw() queries as these can open you up to injection.
2. CSRF: Laravel takes care of this with CSRF tokens that it checks on each POST request so make sure you use them, essentially this protects you from someone changing the nature of the request, i.e from POST to GET.
3. XSS: First sanitise user input. Variables are not escaped using the blade syntax {!! !!}, which resolves to <?= e($foo) ?> inside your HTML code, whereas {{ }} escapes the data.

You should favor the double-brace syntax ({{ $value }}) in your Blade templates, and only use the {!! $value !!} syntax, where you're certain the data is safe to display in its raw format.

1. For exchanging private informations use HTTPs
2. Changing default errors
3. A malicious user could alter the form on the client side and add a new input to it:

<input name="is\_admin" value="1" />

Then, when the form is submitted, we attempt to create a new model using the following code:

Cat::create(Request::all())  
  
Thanks to the $fillable array, which defines a white list of fields that can be filled through mass assignment, this method call will throw a mass assignment exception.

It is also possible to do the opposite and define a blacklist with the $guarded property. However, this option can be potentially dangerous since you might forget to update it when adding new fields to the model