CWEB280 -wk4

# Form Validation

There is always the possibility the user may input data that is not compatible with the web application requirements.

Last year you learn about client-side validation which has the major benefit of making the user aware of issues before send the request to the server. There is a major problem with client-side validation - it can be completely bypassed by a savvy user.

Server-side validation can not be bypassed by a user, but it adds more work for the server. With any web application it is important to do BOTH client side and server-side validation. Most issues can be caught by the client browser, but the server add that’s extra security to the web application.

There are several validator packages but in this course we will use **express-validator**

Learn More About Express-Validator: <https://express-validator.github.io/docs/>

## Express Validator Modules

The Express validator package has several modules which we can import and use as needed the modules include:

check([field, message])

**body**([fields, message]) - Same as check([fields, message]), but only checking req.body.

cookie([fields, message]) - Same as check([fields, message]), but only checking req.cookies.

header([fields, message]) - Same as check([fields, message]), but only checking req.headers.

param([fields, message]) - Same as check([fields, message]), but only checking req.params.

**query**([fields, message]) - Same as check([fields, message]), but only checking req.query.

We will be using the body, query and the validtionResult modules in the following examples

Learn more about the check api : <https://express-validator.github.io/docs/check-api.html> )

## validationResult

* takes in a request object and will search through it for any data that does not meet the validation rules.
* . formatWith (formatter) - allows developers to customize the format of the errors object
* .mapped() – allows developers to use the field name as the key for the error information objects

Learn More about Validation Result: <https://express-validator.github.io/docs/validation-result-api.html>

## Validation Chain

Defining the validation uses a chain of validation function one after another

Example validation chain

***body***('pwd').isLength({min: 8, max: 25}).withMessage('The password must be 8 to 25 characters')  
 .bail().isStrongPassword({minSymbols: 0}).withMessage('Password must contain lowercase, uppercase, and numbers'),

When building the validations there are some helpful methods that can determine of a validation should be performed or ignored

.if(condition) uses the condition to determine whether to continue validating this chain.

<https://express-validator.github.io/docs/validation-chain-api.html#ifcondition>

.bail() Stops running validations if any of the previous ones have failed.

<https://express-validator.github.io/docs/validation-chain-api.html#bail>

Learn more about the validation chain: <https://express-validator.github.io/docs/validation-chain-api.html>

## Sanitizers

Clean the input before or after performing a validation. Some common sanitizers include:

* trim(input [, chars]) – removes extraneous white space before and after text. If you trim before checking that a field is notEmpty. User can enter data that is all spaces that would pass validation, but the trim ensures spaces get removed resulting in an empty field.
* normalizeEmail(email [, options]) – does cleaning like ensuring there is an @ symbol and makes all characters lowercase

Learn More about sanitizers: <https://github.com/validatorjs/validator.js#sanitizers>

# Use Express- Validator

In the terminal run

npm i express-validator

**\routes\examples.js – add the code in violet near the top of the file**

// add packages that will handle the file uploads  
// https://www.npmjs.com/package/multer#usage  
const multer = require('multer');  
// IMPORTANT: ensure you created the destination folder before using it below  
const upload = multer({// multer settings  
 dest: 'public/uploads/',  
 fileFilter: (req, file, callback) => {  
 // check mime type starts with 'image/'  
 if (file.mimetype.startsWith('image/')) {  
 callback(null, true); // file mime type is allowed  
 } else {  
 // file not allowed, so return callback with new error message  
 return callback(new ***Error***('Only images are allowed'));  
 }  
 },  
 limits: {  
 // limit file size to 2MB -> 2\*1024\*1024  
 fileSize: 2\*1024\*1024,  
 },  
});  
  
// require package for file io  
const ***fs*** = require('fs');  
  
// require validator packages  
const {***body***, ***query***, ***validationResult***} = require('express-validator');  
  
// custom error formatter to help traverse the fields and error messages  
const onlyMsgErrorFormatter = ({location, msg, param, value, nestedErrors}) => {  
 return msg; // only return the message of the error  
};

Lets add validators to the Form Post and run the validator to get any error messages

Learn More About Validator: <https://github.com/validatorjs/validator.js#validators>

Learn More about Sanitizers: <https://github.com/validatorjs/validator.js#sanitizers>

**\routes\examples.js – add the code in violet to the form post handler**

*/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* FORM POST  
 \*/*// POST submit form data to path : http://localhost:3000/examples/form/  
***router***.post('/form', [ // validation  
 ***body***('agreed').equals('yes').withMessage('You must agree to the terms and conditions'),  
 ***body***('email').trim().notEmpty().withMessage('Email is required').bail()  
 .normalizeEmail().isEmail().withMessage('Email must be in a valid format'),  
 ***body***('pwd').isLength({min: 8, max: 25}).withMessage('The password must be 8 to 25 characters')  
 .bail().isStrongPassword({minSymbols: 0}).withMessage('Password must contain lowercase, uppercase, and numbers'),  
 ***body***('phone').if(***body***('phone').notEmpty())  
 .trim().isMobilePhone('en-US').withMessage('Phone must be a Canadian phone number. Example: (000) 000-0000'),  
],  
function(req, res, next) {  
 // check the req.body values for validation errors / violations  
 const violations = ***validationResult***(req);  
 // OPTIONAL: inspect the violations in the terminal  
 ***console***.log('Violations:');  
 ***console***.log(violations);  
  
 // Format the error messages to be easier to traverse in handlebars  
 const errorMessages = violations.formatWith(onlyMsgErrorFormatter).mapped();  
 // OPTIONAL: inspect the mapped errors in the terminal  
 ***console***.log('Error Messages:');  
 ***console***.log(errorMessages);  
  
 res.render('form-example', {  
 title: 'POST - Simple Form Example',  
 // check to see if checkbox is checked <input name="agreed" value="yes" />  
 isSubmitted: req.body.agreed === 'yes',  
 // example <input name="someName"> we use req.body.someName  
 // to access the value  
 submittedEmail: req.body.email, // <input name="email" /> .:req.body.email  
 submittedPassword: req.body.pwd, // <input name="pwd" /> .:req.body.pwd  
 submittedAgreed: req.body.agreed, // <input name="agreed"> .:req.body.agreed  
 submittedPhone: req.body.phone,

err: errorMessages,  
 });  
});

We are going to use the bootstrap form validation styles to display error messages

Learn More Bootstrap validation for server-side validation: <https://getbootstrap.com/docs/5.0/forms/validation/#server-side>

**\view\form-example.hbs – add the code in violet**

<form action="/examples/form/" method="post" novalidate >  
 <h1>{{ title }}</h1>  
  
 <div class="mb-2">  
 <label for="email" class="form-label">Email:</label>  
 <!--IMPORTANT the input tag MUST have a name attribute to post to the server ie. name="email"-->  
 <input type="email" class="form-control {{#if err.email}}is-invalid{{/if}}"  
 placeholder="Enter email" name="email" id="email" value="{{submittedEmail}}" required/>  
 <div class="invalid-feedback">{{err.email}}</div>  
 </div>  
 <div class="mb-2">  
 <label for="pwd" class="form-label">Password:</label>  
 <input type="password" class="form-control {{#if err.pwd}}is-invalid{{/if}}"  
 placeholder="Enter password" name="pwd" id="pwd" required/>  
 <div class="invalid-feedback">{{err.pwd}}</div>  
 </div>  
  
 <div class="form-check mb-2">  
 <input class="form-check-input {{#if err.agreed}}is-invalid{{/if}}"  
 type="checkbox" id="agreed" name="agreed" value="yes" {{#if isSubmitted}}checked{{/if}}/>  
 <label for="agreed" class="form-check-label">I agree to the terms and conditions.</label>  
 <div class="invalid-feedback">{{err.agreed}}</div>  
 </div>  
  
 <div class="mb-2">  
 <label for="phone" class="form-label">Phone:</label>  
 <input type="tel" class="form-control {{#if err.phone}}is-invalid{{/if}}"  
 placeholder="Enter phone" name="phone" id="phone" value="{{submittedPhone}}" required/>  
 <div class="invalid-feedback">{{err.phone}}</div>  
 </div>  
  
  
 <button type="submit" class="btn btn-primary">Submit</button>  
</form>  
{{#if isSubmitted}}  
<div class="card p-2">  
 <h2>Submitted values</h2>  
 <p>Email: {{submittedEmail}}</p>  
 <p>Password: {{submittedPassword}}</p>  
 <p>Agreed: {{submittedAgreed}}</p>  
 <p>Phone: {{submittedPhone}}</p>   
</div>  
{{/if}}

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/examples/form/>

EXERCISE:   
What is the structure of the violations?  
What is the structure of the errorMessages (applied a custom formatter to violations) ?

Now lets add some validators to the request query string. Notice that the query string has many if functions to see if the field exists before validating. This prevent all the errors from displaying when there is no query string

**\routes\examples.js – add the code in violet to the form GET handler**

*/\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* FORM GET  
 \*/*// GET content for path: http://localhost:3000/examples/form/  
***router***.get('/form', [ // validation  
 ***query***('agreed').if(***query***('agreed').exists()) // validate if agreed exists  
 .equals('yes').withMessage('You must agree to the terms and conditions'),  
 ***query***('email').if(***query***('email').exists()).trim() // validate if email exists and remove white space  
 .isEmail().withMessage('Email must be in a valid format'),  
 ***query***('pwd').if(***query***('pwd').exists()) // validate if pwd exists  
 .isLength({min: 8, max: 25}).withMessage('The password must be 8 to 25 characters')  
 .bail().isStrongPassword({minSymbols: 0}).withMessage('Password must contain lowercase, uppercase, and numbers'),  
 ***query***('phone').if(***query***('phone').exists()) // validate if phone exists  
 .trim().isMobilePhone('en-US').withMessage('Phone must be a Canadian phone number. Example: (000) 000-0000'),  
],  
function(req, res, next) {  
 // check the req.query values for validation errors / violations  
 const violations = ***validationResult***(req);  
 // OPTIONAL: inspect the violations in the terminal  
 ***console***.log('Violations:');  
 ***console***.log(violations);  
  
 // Format the error messages to be easier to traverse in handlebars  
 const errorMessages = violations.formatWith(onlyMsgErrorFormatter).mapped();  
 // OPTIONAL: inspect the mapped errors in the terminal  
 ***console***.log('Error Messages:');  
 ***console***.log(errorMessages);  
  
 res.render('form-example', {  
 title: 'GET - Simple Form Example',  
 isSubmitted: req.query.agreed === 'yes', // check to see if the agreed url param is 'yes - http://localhost:3000/examples/form/?agreed=yes  
 // example localhost/examples/form/?someName=someValue  
 // then we use req.query.someName access the value of the URL parameter  
 submittedEmail: req.query.email, // ./form/?email=t%40t.ca.: req.query.email  
 submittedPassword: req.query.pwd, // ./form/?pwd=pa$$word .: req.query.pwd  
 submittedAgreed: req.query.agreed, // ./form/?agree=yes .: req.query.agreed  
 submittedPhone: req.query.phone, // ./form/?agree=yes .: req.query.agreed  
 err: errorMessages,  
 });  
});

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/examples/form/?pwd=12345&email=no-an-email>

EXERCISE:   
Add an invalid Value for the agreed field in query string?  
What happens when you remove the query string (Delete starting from the ? to the end of the URL)

# Custom Validators and Custom If Conditions

Ter are often situations when developers have to check the values of other fields on possible checked the database to validation a user input. A common example of this having a confirm password field and both it and the password field need to match

To add a custom field simple pass in an anonymous function the custom () function

Example:

***body***('file2').custom((value, {req}) => {  
 // check that file2 exists and is at least 1KB  
 if (req.files.file2 && req.files.file2[0].size<1024) {  
 throw new ***Error***('Uploaded file must be at least 1KB in size');  
 }  
 // check if title is specified when no file is uploaded  
 if (!req.files.file2 && req.body.title2.trim().length) {  
 throw new ***Error***('File is required when specifying a title');  
 }  
 return true; // Indicates the success of this synchronous custom validator  
}),

Learn more about Custom Validator: <https://express-validator.github.io/docs/custom-validators-sanitizers.html#custom-validator>

You can use a similar anonymous function in the if()

Example:

***body***('desc2').trim().if((value, {req}) => !req.files.file2)  
 .isEmpty().withMessage('Description requires a file to be uploaded'),

# Use Custom Validators on File Uploads

The validation we are trying to enforce requires us to check the values of other fields.

Our plan is to enforce the following validation:

* Title is Required if a file is uploaded
* File is required if a Title is input
* Description is Optional but will display an error if the desc is input but no file is uploaded

**\routes\examples.js – add the code in violet to the Upload POST handler**

*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* UPLOAD POST  
 \*/*// POST submit form data to path : http://localhost:3000/examples/upload/  
// BEST PRACTICE: specify the only fields the app app will accept  
// https://www.npmjs.com/package/multer#usage  
***router***.post('/upload', upload.fields([// multer fields  
 {name: 'file1', maxCount: 1},  
 {name: 'file2', maxCount: 1},  
 {name: 'pictures', maxCount: 3}, // part of Exercise solution  
]),  
[// validation  
 ***body***('title1').trim().if((value, {req}) => req.files.file1)  
 .notEmpty().withMessage('Title is required when uploading a file'),  
 ***body***('desc1').trim().if((value, {req}) => !req.files.file1)  
 .isEmpty().withMessage('Description requires a file to be uploaded'),  
 ***body***('file1').custom((value, {req}) => {  
 // check that file1 exists and is at leat 1KB  
 if (req.files.file1 && req.files.file1[0].size<1024) {  
 throw new ***Error***('Uploaded file must be at least 1KB in size');  
 }  
 // check if title is specified when no file is uploaded  
 if (!req.files.file1 && req.body.title1.trim().length) {  
 throw new ***Error***('File is required when specifying a title');  
 }  
 return true; // Indicates the success of this synchronous custom validator  
 }),  
  
  
 ***body***('title2').trim().if((value, {req}) => req.files.file2)  
 .notEmpty().withMessage('Title is required when uploading a file'),  
 ***body***('desc2').trim().if((value, {req}) => !req.files.file2)  
 .isEmpty().withMessage('Description requires a file to be uploaded'),  
 ***body***('file2').custom((value, {req}) => {  
 // check that file2 exists and is at leat 2KB  
 if (req.files.file2 && req.files.file2[0].size<1024) {  
 throw new ***Error***('Uploaded file must be at least 1KB in size');  
 }  
 // check if title is specified when no file is uploaded  
 if (!req.files.file2 && req.body.title2.trim().length) {  
 throw new ***Error***('File is required when specifying a title');  
 }  
 return true; // Indicates the success of this synchronous custom validator  
 }),  
],  
(req, res, next) => {  
 // output file array info to console to see what is available  
 ***console***.log('uploaded files:\n');  
 ***console***.log(req.files);  
  
 // check the req.query values for validation errors / violations  
 const violations = ***validationResult***(req);  
 // OPTIONAL: inspect the violations in the terminal  
 ***console***.log('Violations:');  
 ***console***.log(violations);  
  
 const errorMessages = violations.formatWith(onlyMsgErrorFormatter).mapped();  
  
 // loop through the files object and then the nested arrays  
 for (const [field, fileArray] of ***Object***.entries(req.files)) {  
 // OPTIONAL: inspect the objects in the terminal  
 ***console***.log('Field: ' + field + '\n');  
 ***console***.log(fileArray);  
 // loop through the files in the array  
 for (const tempFile of fileArray) {  
 // OPTIONAL: again inspect the files in the terminal  
 ***console***.log('Temp File:\n');  
 ***console***.log(tempFile);  
  
 /\* Now that we have validation - we need to ensure that any fields that correspond to file fields  
 ie( file1, title1, file2, title2, pictures) do not have errors/violations. If a field does have an  
 error then we should remove/delete the corresponding uploaded file from the server  
 if the field does not have an error then move the files as normal to the images folder  
 \*/  
 // if the is an error message for the file's fieldname  
 if (tempFile.fieldname in errorMessages  
 ) {  
 // Delete temporary uploaded file if there is an error in the filed name  
 ***fs***.unlink( tempFile.path, (err)=>{  
 if (err) throw err;  
 ***console***.log('File removed at ' + tempFile.path);  
 });  
 } else {  
 // call the move file function to move the file to public/images folder  
 moveFile(tempFile, \_\_dirname + '/../public/images/');  
 }  
 }  
 }  
  
 // declare variables to store the uploaded file information  
 let file1;  
 let file2;  
 let pictures; // part of exercise 6 solution  
 // check to see if the corresponding files were uploaded  
 // otherwise use new object  
 file1 = req.files['file1'] ? req.files['file1'][0] : {originalname: 'not uploaded'};  
 file2 = req.files['file2'] ? req.files['file2'][0] : {originalname: 'not uploaded'};

// if the pictures field has an error then do not list the uploaded picture files  
 pictures= ('pictures' in errorMessages)? [] : req.files.pictures;  
  
 res.render('upload-files', {  
 title: 'POST - Upload Form Example',  
 isSubmitted: true, // check to see if the file title is filled in />  
 file1Title: req.body.title1,  
 file2Title: req.body.title2,  
 file1Description: req.body.desc1,  
 file2Description: req.body.desc2,  
 file1Info: file1,  
 file2Info: file2,  
 pictures: pictures, // part of exercise 6 solution  
 err: errorMessages,  
 });  
});