**Why should we use validation ?**

We typically have a lot of forms through which a user interacts with our application. In the end when the form is submitted, then a request with that form data is sent to our backend. On our backend, we typically interact with a database or file to write the incoming data to them. This can be dangerous when we don’t have any validation on those data. For example if a user provides something different from an email while login, we allow that and the user can pass in any malicious data.

**How to validate input?**

We have got couple of places where we can add validations. For example-

* Validate on the client side with the help of Javascript which can greatly enhance the user experience. But the Javascript code that runs in the browser - the user can see that code and change that code and can even disable Javascript.
* Validate on the server side with the help of Nodejs. This ensures that we are working with a valid data on the server. For some database engines there is also built in validation that we can turn on. Mongodb provides this. But if we have good server side validation then this is not even required.

**Set up and Basic Validation**

To add validations we will use a 3rd party package - express-validator.

**npm install --save express-validator**

Express validator is made up of sub packages of validators.

We can use next gen Javascript destructuring to extract only the required validators from the express validator as below

// routes/auth.js

const expValidator = require('express-validator'); // gets all validators

const { check } = require('express-validator/ check'); // destructuring gets only check

router.post('/signup', check('email').isEmail(), authController.postSignup);

// controllers/auth.js

const { validationResult } = require('express-validator/check');

exports.postSignup = (req, res, next) => {

  const email = req.body.email;

  const password = req.body.password;

  const confirmPassword = req.body.confirmPassword;

  const errors = validationResult(req); ----->

  if (!errors.isEmpty()) { ----->

    console.log(errors.array()); ----->

    return res.status(422).render('auth/signup', { ----->

      path: '/signup', ----->

      pageTitle: 'Signup', ----->

      errorMessage: errors.array()[0].msg ----->

    }); ----->

  } ----->

  User.findOne({ email: email })

    .then(userDoc => {

      if (userDoc) {

        req.flash(

          'error',

          'E-Mail exists already, please pick a different one.'

        );

        return res.redirect('/signup');

      }

      return bcrypt

        .hash(password, 12)

        .then(hashedPassword => {

          const user = new User({

            email: email,

            password: hashedPassword,

            cart: { items: [] }

          });

          return user.save();

        })

        .then(result => {

          res.redirect('/login');

          return transporter.sendMail({

            to: email,

            from: 'shop@node-complete.com',

            subject: 'Signup succeeded!',

            html: '<h1>You successfully signed up!</h1>'

          });

        })

        .catch(err => {

          console.log(err);

        });

    })

    .catch(err => {

      console.log(err);

    });

};

To send specific message we do the following -

// routes/auth.js

router.post('/signup',

    check('email').isEmail().withMessage('Please provide a valid email'),

    authController.postSignup);

**Docs-**

<https://express-validator.github.io/docs/>

<https://github.com/validatorjs/validator.js>

**Custom Validators**

router.post('/signup',

    check('email')

        .isEmail()

        .withMessage('Please provide a valid email')

        .custom((value, {req}) => { ----->

            if(value == 'test@test.com') { ----->

                throw new Error('This email address is forbidden') ----->

            } ----->

            return true; ----->

        }),

    authController.postSignup);

This way we can chain any number of validators.

router.post('/signup',

    check('email')

     ..

    }),

    body('password',  ----->

         'Password must only contain numbers and text and should be atleast 5 characters long') ----->

         .isLength({min: 5}) ----->

         .isAlphanumeric(), ----->

    authController.postSignup);

**Checking for Field Equality**

router.post('/signup',

    check('email')

        .isEmail()

        .withMessage('Please provide a valid email')

        .custom((value, {req}) => {

            if(value == 'test@test.com') {

                throw new Error('This email address is forbidden')

            }

            return true;

        }),

    body('password',

         'Password must only contain numbers and text and should be atleast 5 characters long')

         .isLength({min: 5})

         .isAlphanumeric(),

    body('confirmPassword') ----->

        .custom((value, {req}) => { ----->

            if(value !== req.body.password) { ----->

                throw new Error('Password & Confirm Password should match') ----->

            } ----->

return true; ----->

}),

    authController.postSignup);

**Adding Async Validation**

router.post('/signup',

    check('email')

        .isEmail()

        .withMessage('Please provide a valid email')

        .custom((value, {req}) => {

            // if(value == 'test@test.com') {

            //     throw new Error('This email address is forbidden')

            // }

            // return true;

            return User.findOne({email: value}) ----->

                .then(user => { ----->

                    if(user) { ----->

                        return Promise.reject('Email already exists.'); ----->

                    }

                })

        }),

    body('password',

         ..

    body('confirmPassword')

        ..

    }),

    authController.postSignup);

**Keeping User Input**

In order to improve the UX we can populate the form with the data that the user has already provided when some error occurs. We can do so by returning the oldInputs object from the controller function in the res we are sending back and doing changes in the views as well as

return res.status(422).render('auth/signup', {

      path: '/signup',

      pageTitle: 'Signup',

      errorMessage: errors.array(),

      oldInput: {

        email: email,

        password: password,

        confirmPassword: confirmPassword

      }

    });

**Adding Conditional CSS**

We do have our errors array already and we can return that as well as

const errors = validationResult(req);

return res.status(422).render('auth/signup', {

  path: '/signup',

  pageTitle: 'Signup',

  errorMessage: errors.array()[0],

  oldInput: {

    email: email,

    password: password,

    confirmPassword: confirmPassword

  },

  validationErrors: errors.array(); ----->

});

Also in getSignUp controller function return validationError as [].

In the template-

<input class="<% validationErrors.find(e => e.param === 'email') ? 'invalid': '' %>" ----->

        type="email"

        name="email"

        id="email">

**Sanitizing Data**

Cleaning data that we received. we need to make sure that data that we get is not only validated but also sanitized so that all the data is stored in a uniform way in the database.

router.post('/signup',

    check('email')

        .isEmail()

        .withMessage('Please provide a valid email')

        .normalizeEmail(), ----->

    body('password',

         'Password must only contain numbers and text and should be atleast 5 characters long')

         .isLength({min: 5})

         .isAlphanumeric()

         .normalizeEmail()

         .trim(), ----->

body('confirmPassword')

.trim() ----->

        .custom((value, {req}) => {

            if(value !== req.body.password) {

                throw new Error('Password & Confirm Password should match')

            }

            return true;

    }),

    authController.postSignup);

**Useful Resources**

Express-Validator Docs: [https://express-validator.github.io/docs/](https://express-validator.github.io/docs/" \t "https://www.udemy.com/course/nodejs-the-complete-guide/learn/lecture/_blank)

Validator.js (which is used behind the scenes) Docs: [https://github.com/chriso/validator.js](https://github.com/chriso/validator.js" \t "https://www.udemy.com/course/nodejs-the-complete-guide/learn/lecture/_blank)