Sử dụng Express-Validator

npm install --save express-validator

Sử dụng

const { body, validationResult } = require('express-validator');

app.post('/user', [

  body('username').isEmail(),

  body('password').isLength({ min: 5 })

], (req, res) => {

  const errors = validationResult(req);

  if (!errors.isEmpty()) {

    return res.status(400).json({ errors: errors.array() });

  }

  User.create({

    username: req.body.username,

    password: req.body.password

  }).then(user => res.json(user));

});

Khi mà có lỗi thì ta sẽ nhận được object Errors như này :

{

"errors": [{

"location": "body",

"msg": "Invalid value",

"param": "username"

}]

}

Validation middlewares

Những method check validation :

* check([field, message])
* body([fields, message])
* cookie([fields, message])
* header([fields, message])
* param([fields, message])
* query([fields, message])
* checkSchema(schema)

## One Of

chỉ cần 1 điều kiện đúng là đủ

* oneOf(validationChains[, message])

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

app.post('/start-freelancing', oneOf([

check('programming\_language').isIn(['javascript', 'java', 'php']),

check('design\_tools').isIn(['canva', 'photoshop', 'gimp'])

]), someRouteHandler);

Kết hợp OR và AND

app.post('/protected/route', oneOf([

[

check('username').exists(),

check('password').exists()

],

check('access\_token').exists()

]), someRouteHandler);

Check Combination

Kết hợp check nhiều nơi

* buildCheckFunction(locations)

**const** { buildCheckFunction } = require('express-validator');

**const** checkBodyAndQuery = buildCheckFunction(['body', 'query']);

app.put('/update-product', [

// id must be either in req.body or req.query, and must be an UUID

checkBodyAndQuery('id').isUUID()

], productUpdateHandler)

args : body, cookies, headers, params or query

Check vs body,cookie v.v

Check là bao gồm tất cả : body, cookies, headers, params, query

Còn lại nếu chỉ muốn check phần cụ thể thì chỉ import phần đó

# Checking APIs

### .bail()

### .custom(validator)

### .exists(options)

### .if(condition)

### .isArray(options)

### .isString()

### .not()

### .notEmpty()

### .optional(options)

### .run(req[, options])

### .withMessage(message)

Xem thêm ở đây : [validation.js](https://github.com/validatorjs/validator.js#validators)

|  |  |
| --- | --- |
| equals(str, comparison) | isURL(str [, options]) |
| isBoolean(str) |  |
| isDataURI(str) |  |
| isDate(input [, options]) |  |
| isEmail(str [, options]) |  |
| isInt(str [, options]) |  |
| isLength(str [, options]) |  |

## Bail

dùng để dùng validation ngay khi điều kiện chỉ định bị sai

app.post('/', [

check('username')

.isEmail()

.bail()

// If username is not an email, checkBlacklistedDomain will never run

.custom(checkBlacklistedDomain)

.bail()

.custom(checkEmailExists);

]);

Dùng trong trường hợp sẽ có ~ validation truy vấn DB hoặc gọi API để check thì mình k cho nó chạy cái đó

## Custom

.custom((value, { req, location, path }) => {})

Để fail validation : mình có thể return Promise.reject hoặc throw Error

Nếu mình validation async task => return Promisemes

**co**nst { body } = require('express-validator');

app.post('/user', body('email').custom(value => {

return User.findUserByEmail(value).then(user => {

if (user) {

return Promise.reject('E-mail already in use');

}

// hoặc

**throw** **new** Error('Password confirmation does not match password');

});

}), (req, res) => {

// Handle the request

});

## Exist

.exists({

checkNull: true/false,

checkFalsy: true/false

})

* checkNull: if true, fields with null values will not exist
* checkFalsy: if true, fields with falsy values (eg "", 0, false, null) will also not exist

## If

xác định xem có continue validation

.if((value, { req, path, location }) => {})

body('oldPassword')

// if the new password is provided...

.if((value, { req }) => req.body.newPassword)

// OR

.if(body('newPassword').exists())

// ...then the old password must be too...

.not().empty()

// ...and they must not be equal.

.custom((value, { req }) => value !== req.body.newPassword)

## isArray

.isArray({

Min: arr length

Max: arr length

})

## .isString() : is value String ?

## .not()

check('weekday').not().isIn(['sunday', 'saturday'])

## .notEmpty() : string with length > 0 ?

check('username').notEmpty()

.optional({

nullable: true/false,

checkFalsy: true/false (falsy = "", 0, false, null)

})

### nếu field là undefined thì sẽ được bỏ qua

# Custom Error Message

Validator Level

[

    check("password")

        .isLength({ min: 5 }).withMessage("must be at least 5 chars long")

        .matches(/\d/).withMessage("must contain a number"),

];

Custom Level

[

    check("email").custom((value) => {

        //...

        return Promise.reject("E-mail already in use");

    }),

    check("password").custom((value, { req }) => {

        //...

        throw new Error("Password confirmation is incorrect");

    }),

];

Field Level

[

    check("password", "The password must be 5+ chars long and contain a number")

        .not()

        .isIn(["123", "password", "god"])

        .withMessage("Do not use a common word as the password")

        .isLength({ min: 5 })

        .matches(/\d/),

];

Ở những field không sử dụng withMessage cụ thể 🡪 sẽ sử dụng Field Level message (“The password must be 5+ chars….”)

## Dynamic Message

// check(field, withMessage) and .withMessage() work the same

check("something")

    .isInt()

    .withMessage((value, { req, location, path }) => {

        return req.translate("validation.message.path", { value, location, path });

    }),

check("somethingElse", (value, { req, location, path }) => {

    return req.translate("validation.message.path", { value, location, path });

}),

// oneOf is special though - it only receives the req object for now

oneOf([someValidation, anotherValidation], ({ req }) => {

    return req.translate("validation.multiple\_failures");

});

# Schema Validation

const { checkSchema } = require("express-validator");

app.put(

    "/user/:id/password",

    checkSchema({

        id: {

            // If omitted, all request locations will be checked

            in: ["params", "query"], // can be : body, cookies, headers, params or query.

            errorMessage: "ID is wrong",

            isInt: true,

            // Sanitizers can go here as well

            toInt: true,

        },

        myCustomField: {

            // Custom validators

            custom: {

                options: (value, { req, location, path }) => {

                    return value + req.body.foo + location + path;

                },

            },

            // and sanitizers

            customSanitizer: {

                options: (value, { req, location, path }) => {

                    let sanitizedValue;

                    if (req.body.foo && location && path) {

                        sanitizedValue = parseInt(value);

                    } else {

                        sanitizedValue = 0;

                    }

                    return sanitizedValue;

                },

            },

        },

        password: {

            isLength: {

                errorMessage: "Password should be at least 7 chars long",

                // Multiple options would be expressed as an array

                options: { min: 7 },

            },

        },

        firstName: {

            isUppercase: {

                // To negate a validator

                negated: true,

            },

            rtrim: {

                // Options as an array

                options: [[" ", "-"]],

            },

        },

        // Wildcards/dots for nested fields work as well

        "addresses.\*.postalCode": {

            // Make this field optional when undefined or null

            optional: { options: { nullable: true } },

            isPostalCode: true,

        },

    }),

    (req, res, next) => {

        // handle the request as usual

    }

);

# Sanitizers

Convert data sau khi đã validate xong

* normalizeEmail()
* trim()
* toInt()
* .toArray()
* .toLowerCase()
* .toUpperCase()
* .toDate()
* .toBoolean()
* .escape()
* .customSanitizer((value, { req, location, path }) => {})

Tìm hiểu thêm : [validator.js#sanitizers](https://github.com/validatorjs/validator.js#sanitizers)

## Custom

const { param } = require('express-validator');

app.post('/object/:id', param('id').customSanitizer(value => {

return ObjectId(value);

}), (req, res) => {

// Handle the request

});

## Middlewares

* sanitize(fields)
  + req.body
  + req.cookies
  + req.params
  + req.query
  + \* req.headers is not supported at the moment.
* sanitizeBody(fields)
* sanitizeCookie(fields)
* sanitizeParam(fields)
* sanitizeQuery(fields)
* buildSanitizeFunction(**locations**)
  + **locations**: body, cookies, params or query

const { buildSanitizeFunction } = require('express-validator');

const sanitizeBodyAndQuery = buildSanitizeFunction(['body', 'query']);

app.put('/update-product', [

// id being either in req.body or req.query will be converted to int

sanitizeBodyAndQuery('id').toInt()

], productUpdateHandler)

# Advanced (ngoài lề)

Parallel & Sequential

// parallel processing

const validate = validations => {

return async (req, res, next) => {

await Promise.all(validations.map(validation => validation.run(req)));

const errors = validationResult(req);

if (errors.isEmpty()) {

return next();

}

res.status(400).json({ errors: errors.array() });

};

};

// sequential processing, stops running validations chain if the previous one have failed.

const validate = validations => {

return async (req, res, next) => {

for (let validation of validations) {

const result = await validation.run(req);

if (result.errors.length) break;

}

const errors = validationResult(req);

if (errors.isEmpty()) {

return next();

}

res.status(400).json({ errors: errors.array() });

}

};

Usage

app.post('/api/create-user', validate([

body('email').isEmail(),

body('password').isLength({ min: 6 })

]), async (req, res, next) => {

// request is guaranteed to not have any validation errors.

const user = await User.create({ ... });

});