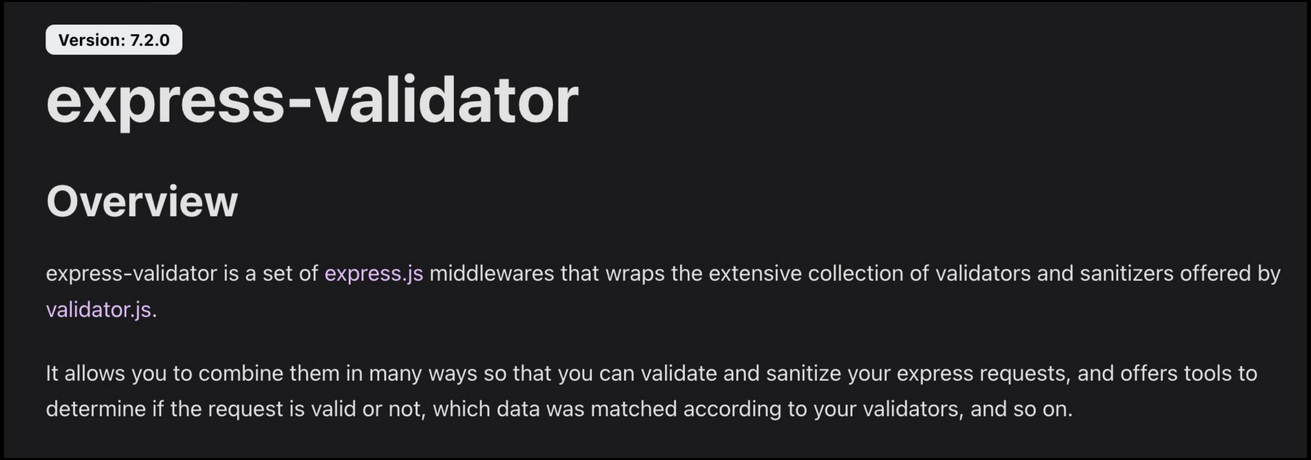
19.6 Using Express Validator

1. Add handling for POST /signup in auth controller and router.
2. Install Express Validator.
3. Use the email and password validations in the post handler.
4. Change the signup.ejs to show the errors. And accept the old values.



Version: 7.2.0

express-validator

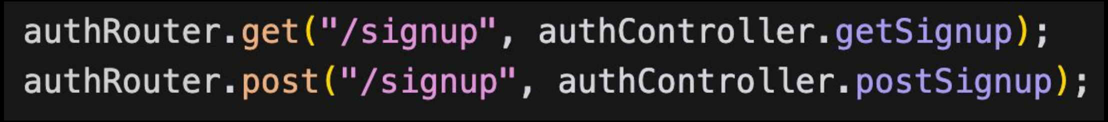
Overview

express-validator is a set of express.js middlewares that wraps the extensive collection of validators and sanitizers offered by validator.js.

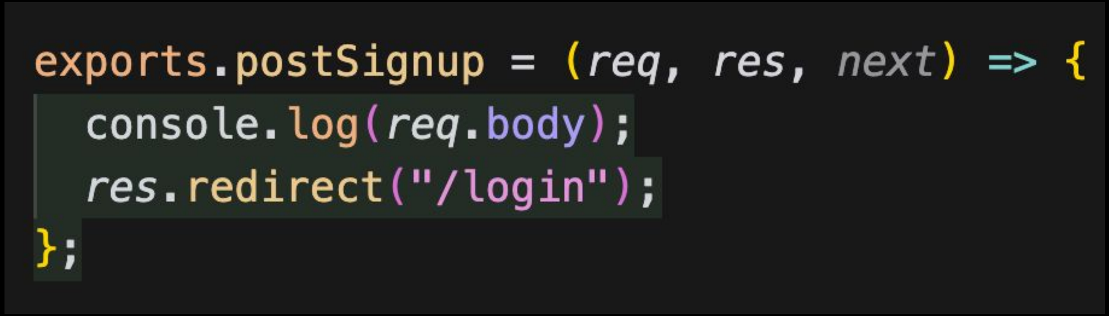
It allows you to combine them in many ways so that you can validate and sanitize your express requests, and offers tools to determine if the request is valid or not, which data was matched according to your validators, and so on.

19.6 Using Express Validator

1.

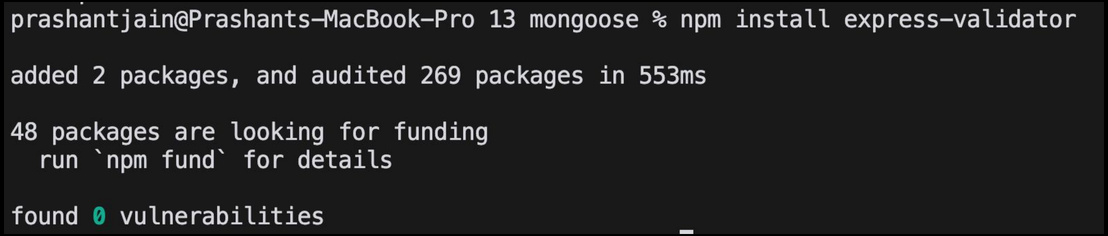


```
authRouter.get("/signup", authController.getSignup);
authRouter.post("/signup", authController.postSignup);
```



```
exports.postSignup = (req, res, next) => {
  console.log(req.body);
  res.redirect("/login");
};
```

2.



```
prashantjain@Prashants-MacBook-Pro 13 mongoose % npm install express-validator

added 2 packages, and audited 269 packages in 553ms

48 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

19.6 Using Express Validator

3.

```javascript
exports.postSignup = [
  // First Name validation
  check('firstName')
    .notEmpty()
    .withMessage('First name is required')
    .trim()
    .isLength({ min: 2 })
    .withMessage('First name must be at least 2 characters long')
    .matches(/^[a-zA-Z\s]+$/)
    .withMessage('First name can only contain letters'),

  // Last Name validation
  check('lastName')
    .notEmpty()
    .withMessage('Last name is required')
    .trim()
    .isLength({ min: 2 })
    .withMessage('Last name must be at least 2 characters long')
    .matches(/^[a-zA-Z\s]+$/)
    .withMessage('Last name can only contain letters'),

  // Email validation
  check('email')
    .isEmail()
    .withMessage('Please enter a valid email')
    .normalizeEmail(),
```

```javascript
// Password validation
check('password')
  .isLength({ min: 8 })
  .withMessage('Password must be at least 8 characters long')
  .matches(/[a-z]/)
  .withMessage('Password must contain at least one lowercase letter')
  .matches(/[A-Z]/)
  .withMessage('Password must contain at least one uppercase letter')
  .matches(/[!@#$%^&*(),.?":{}|<>]/)
  .withMessage('Password must contain at least one special character')
  .trim(),

// Confirm password validation
check('confirm_password')
  .trim()
  .custom((value, { req }) => {
    if (value !== req.body.password) {
      throw new Error('Passwords do not match');
    }
    return true;
  }),

// User Type validation
check('userType')
  .notEmpty()
  .withMessage('User type is required')
  .isIn(['guest', 'host'])
  .withMessage('Invalid user type'),

// Terms Accepted validation
check('termsAccepted')
  .notEmpty()
  .withMessage('You must accept the terms and conditions')
  .custom((value) => {
    if (value !== 'on') {
      throw new Error('You must accept the terms and conditions');
    }
    return true;
  }),
```

19.6 Using Express Validator

3.
```
// Final handler middleware
(req, res, next) => {
  const { firstName, lastName, email, password, userType } = req.body;
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(422).render('auth/signup', {
      pageTitle: 'Sign Up',
      isLoggedIn: false,
      errorMessages: errors.array().map(error => error.msg),
      oldInput: {
        firstName,
        lastName,
        email,
        password,
        userType
      }
    });
  }

  res.redirect('/login');
}
```

19.6 Using Express Validator

4.
```
<% if (typeof errorMessages !== 'undefined' && errorMessages && errorMessages.length > 0) { %>
  <div class="bg-red-100 border border-red-400 text-red-700 px-4 py-3 rounded relative mb-4"
  role="alert">
    <ul class="list-disc list-inside space-y-1">
      <% errorMessages.forEach(error => { %>
        <li><%= error %></li>
      <% }); %>
    </ul>
  </div>
<% } %>

<input
  type="text"
  name="firstName"
  placeholder="First Name"
  value="<%= typeof oldInput !== 'undefined' ? oldInput.firstName : '' %>"
  class="w-full px-4 py-2 mb-4 border border-gray-300 rounded-lg focus:outline-none
  focus:border-blue-500"
/>
```

19.7 Adding User Model

1. Define a new User Model with following fields:
   a. firstName & lastName (required)
   b. email (required, unique)
   c. Password (required)
   d. userType (possible values ‘guest’, ‘host’)
2. In the POST signup handler, create a new user with the fields from request and redirect to login after saving the user.

19.7 Adding User Model

1.

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true
  },
  lastName: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  userType: {
    type: String,
    required: true,
    enum: ['guest', 'host']
  }
});

module.exports = mongoose.model('User', userSchema);
```

19.7 Adding User Model

2.

```
const user = new User({
  firstName: firstName,
  lastName: lastName,
  email: email,
  password: password,
  userType: userType
});

user.save()
  .then(() => {
    res.redirect('/login');
})
.catch(err => {
    console.log("Error while creating user: ", err);
});
```