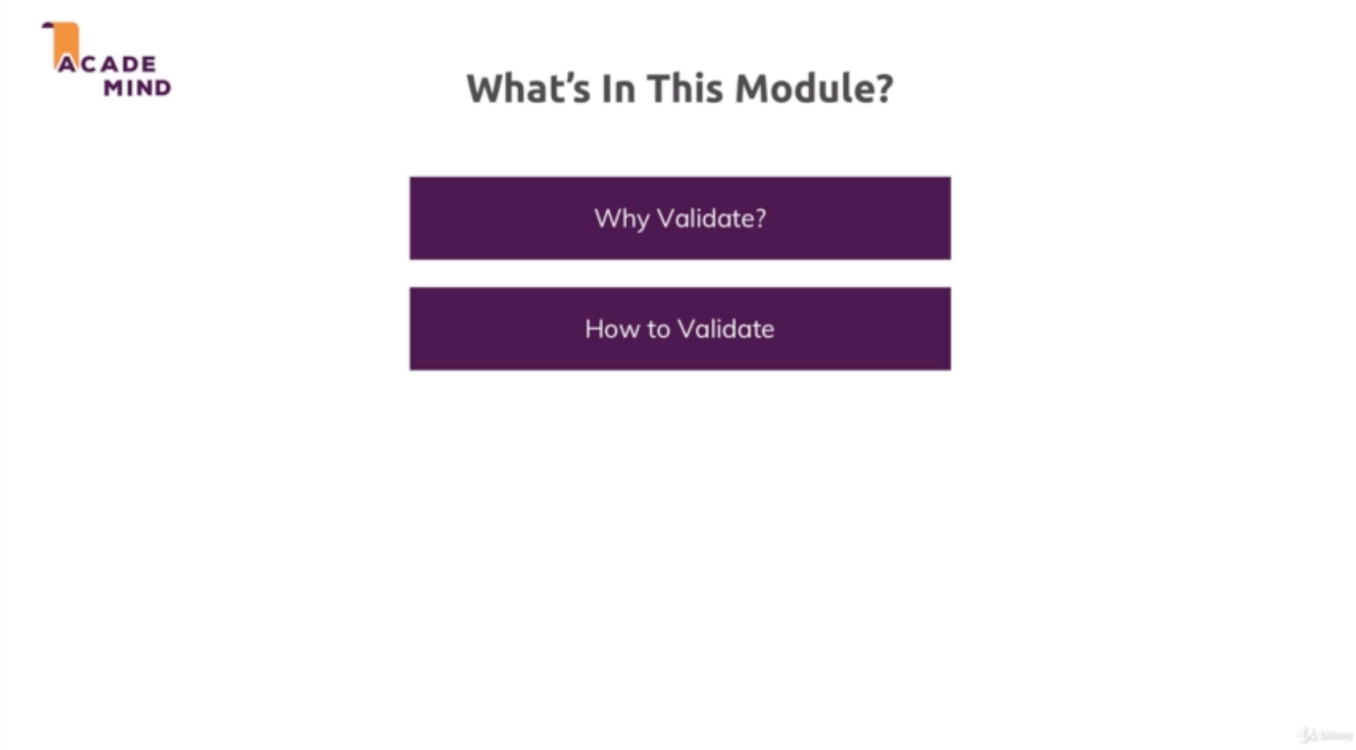
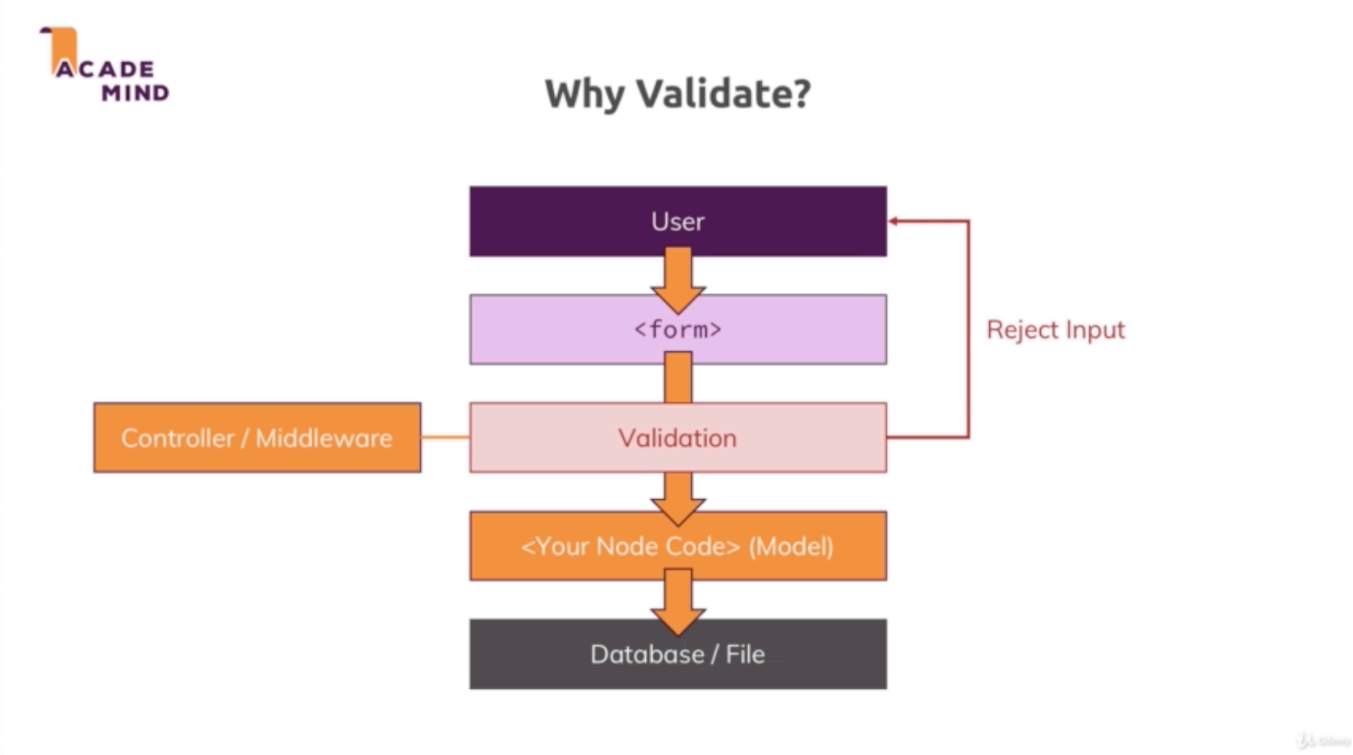
18. Understanding Validation

* Chapter 284: Module Introduction

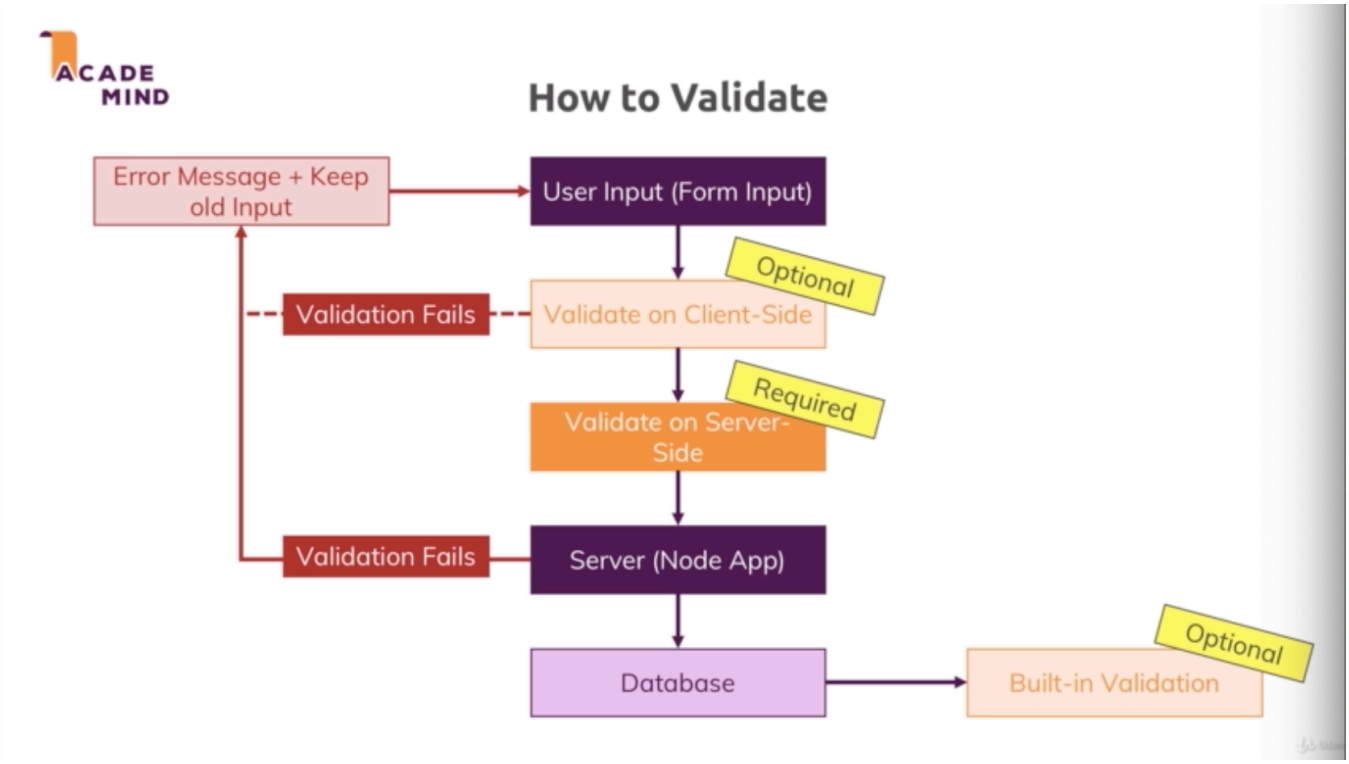


* Chapter 285: Why Should We Use Validation?



- in our example project for example, we have a form for signing up, signing in and we got one for adding products.
- when this forms is submitted with a POST request as we controlled it in our form, then a Request is sent to our backend.
- if a user in our current application would try to login with something that is not a valid email address, we would allow that. we are not preventing the user from entering something incorrect.
- the same is true for adding a product, we don't care about what the user enters and this is what i wanna chagne in this module.
- this validation can then either succeed and allow the data to be written to the database or allow it to be handled by the rest of our node code or we reject the input and then return some information to the user prompting the user to correct the error.

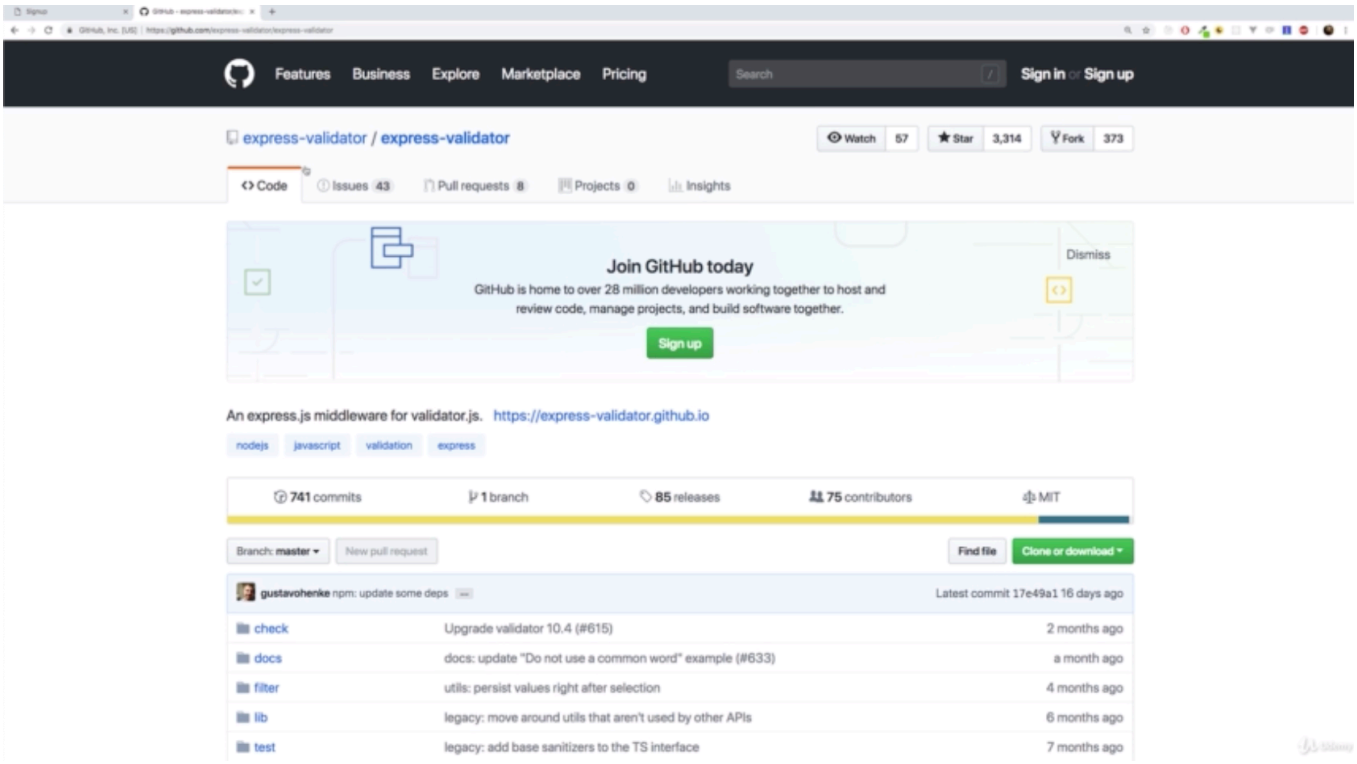
* Chapter 286: How To Validate Input?

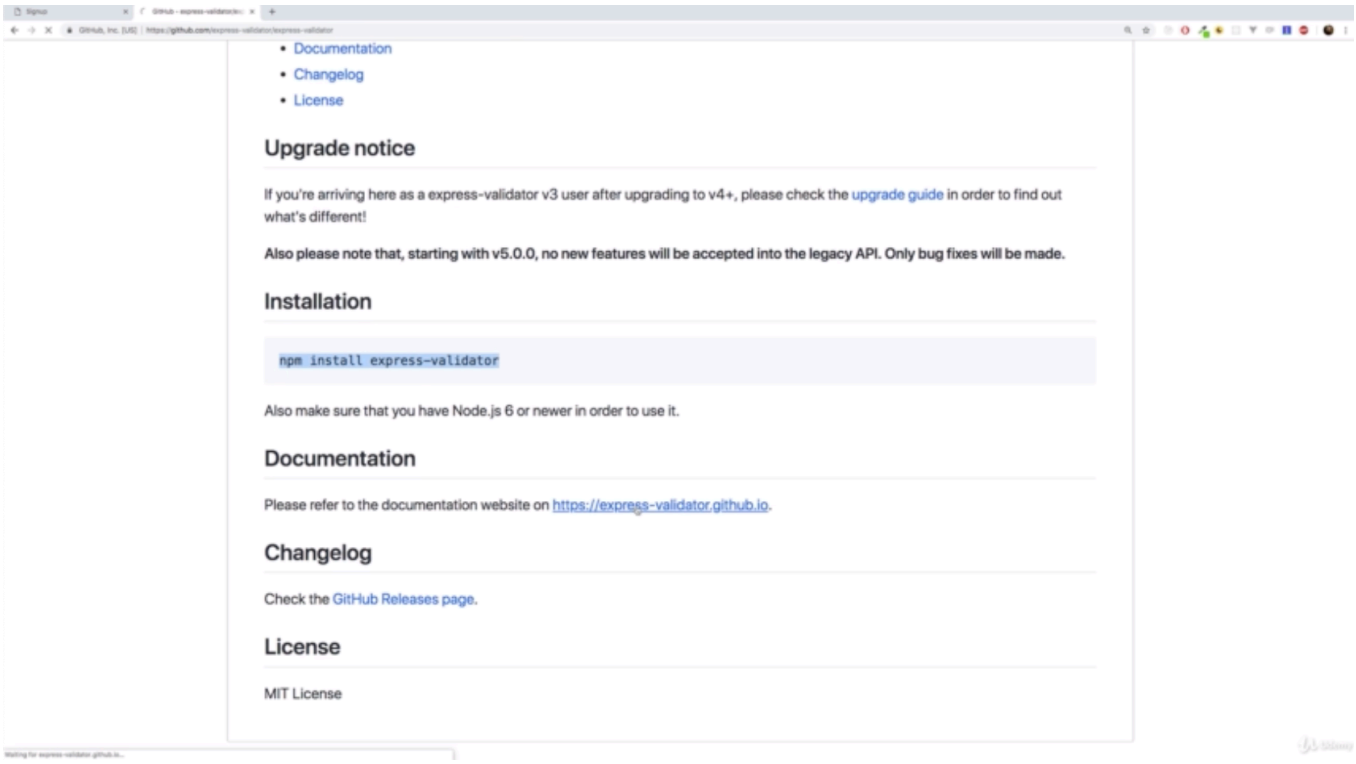


- since we use client side javascript, so javascript code that runs in the browser, the user can see that code, the user could change that code and the user can disable javascript. so this is not the protection that secures you against incorrect data being sent to your server. this is not secure solution.
- for some database engines and for most database engines like MongoDB, there is also built-in validation which you can turn on. it's optional because this can be a last resort but if you have a good server side validation in place as you should have. but this might not be required because there is not really a scenario where invalid data could reach your database. because you filter it out in that server side validation already.
- you should validate on the server side at all means but no matter what you choose, in the end, 'Server(Node App)' can also fail and then you should always return an error message-helpful error message and never reload the page but always keep the data the user already inserted because that is a horrible user experience which we know that you enter something incorrect and you get back and fill out from start again.

* Chapter 287: Setup & Basic Validation

1. update
- ./routes/auth.js
- ./controllers/auth.js
- ./views/auth/signup.ejs





- Documentation
- Changelog
- License

Upgrade notice

If you're arriving here as a express-validator v3 user after upgrading to v4+, please check the upgrade guide in order to find out what's different!

Also please note that, starting with v5.0.0, no new features will be accepted into the legacy API. Only bug fixes will be made.

Installation

```
npm install express-validator
```

Also make sure that you have Node.js 6 or newer in order to use it.

Documentation

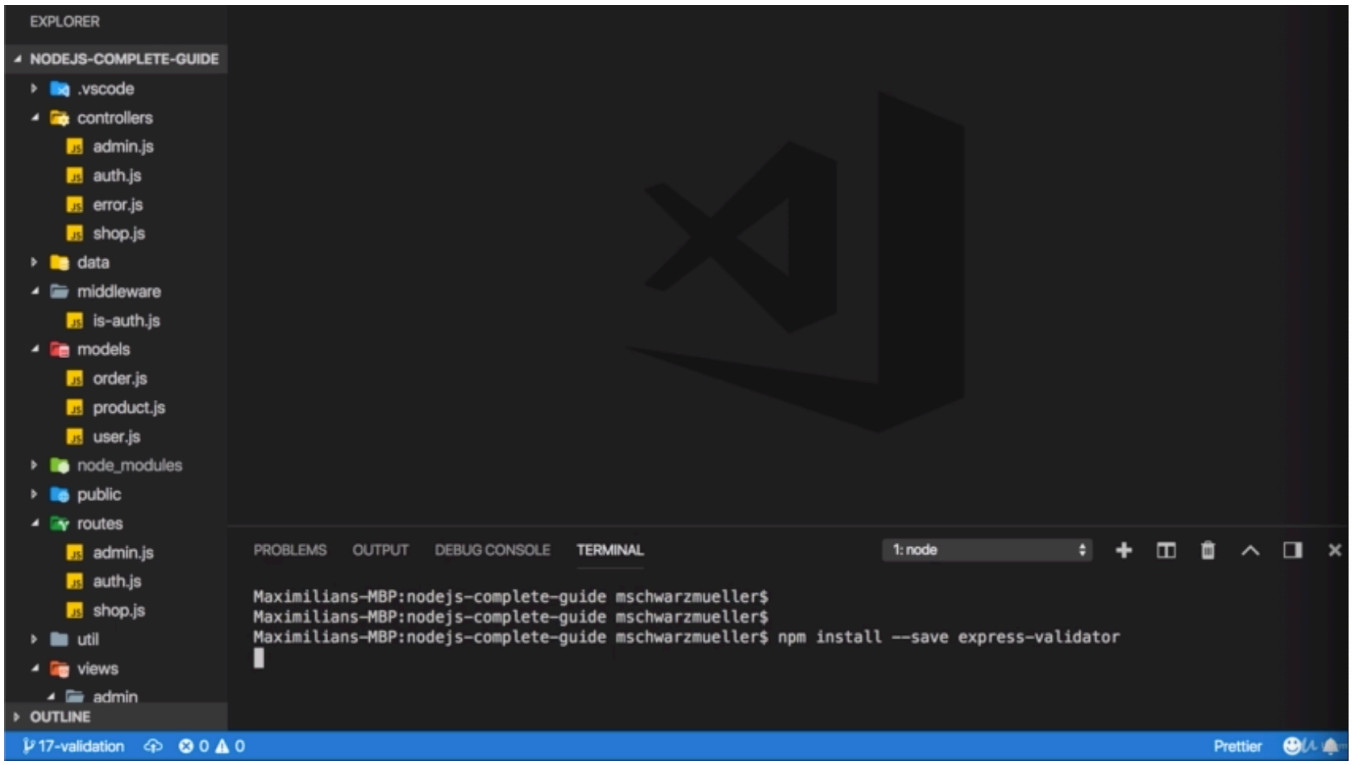
Please refer to the documentation website on https://express-validator.github.io.

Changelog

Check the GitHub Releases page.

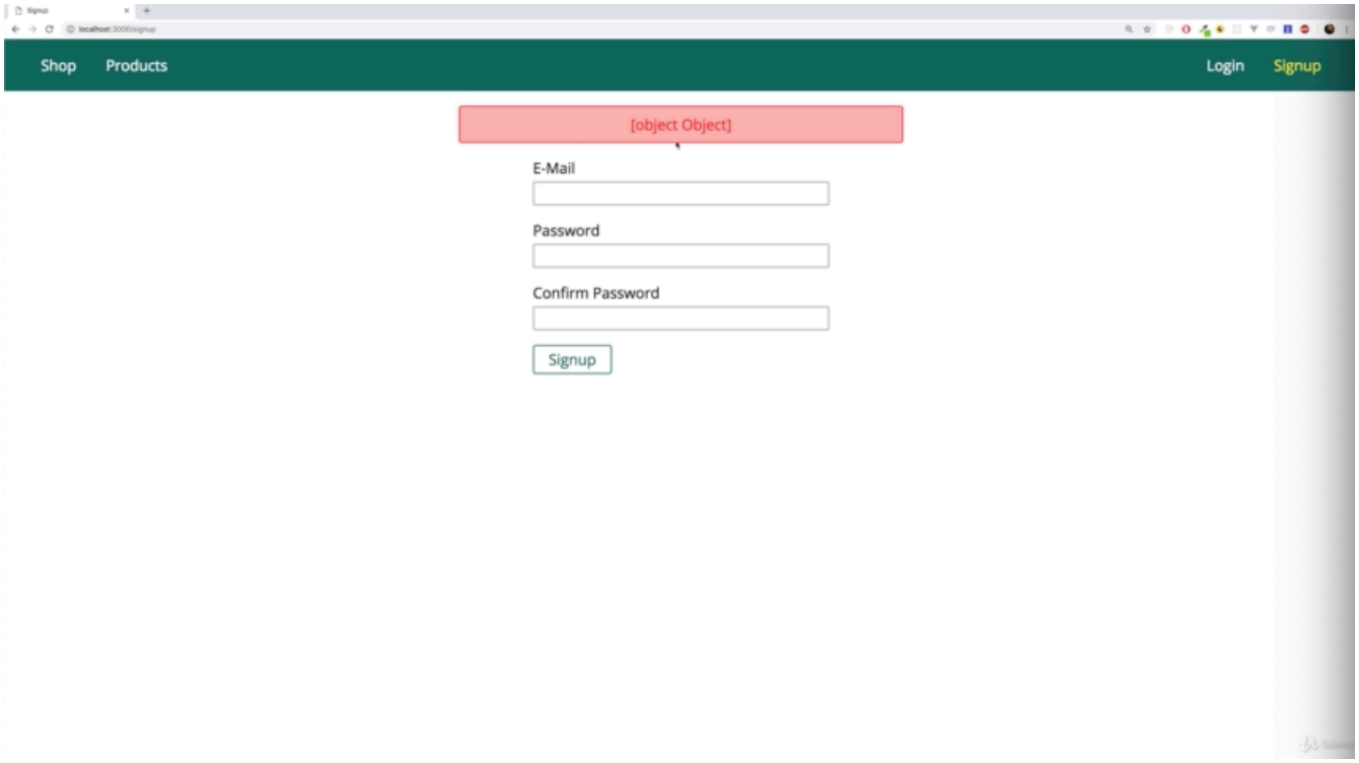
License

MIT License

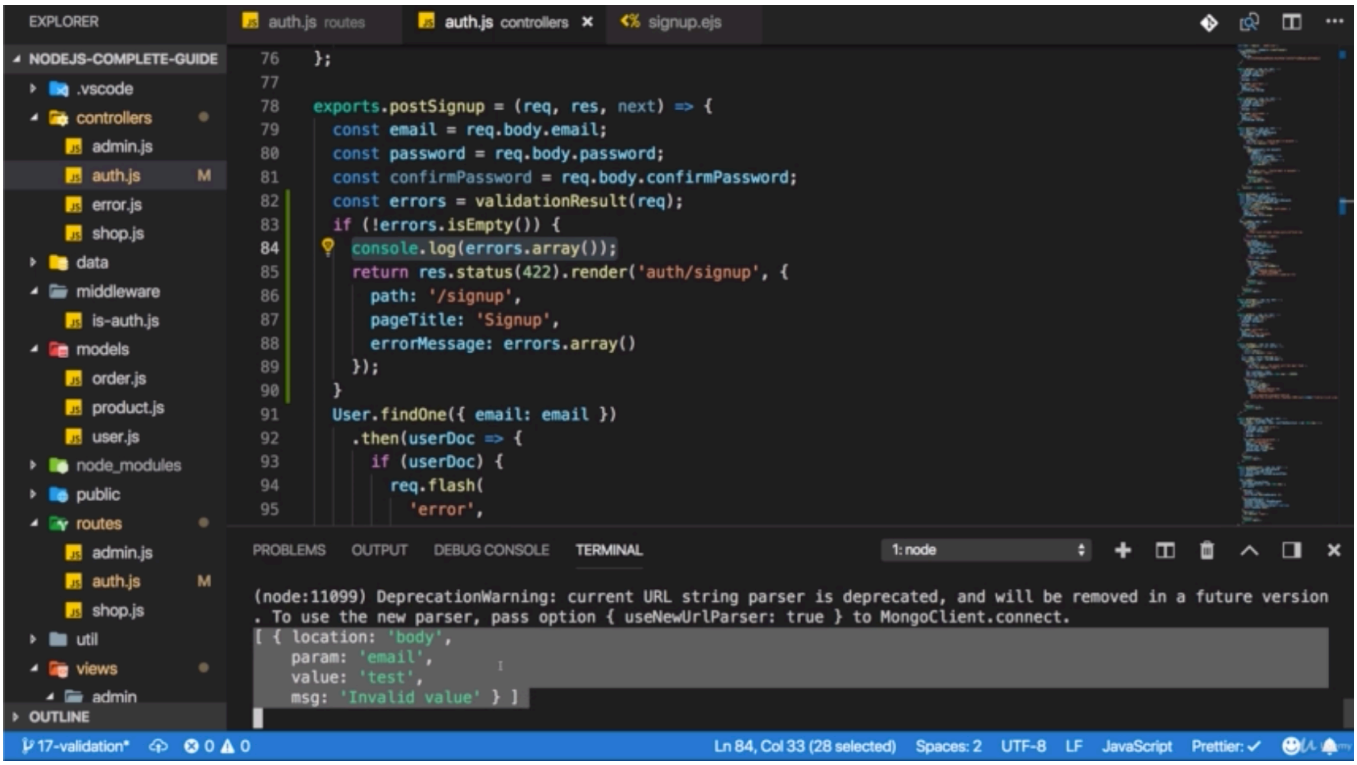


- the package we will be using is called 'express-validator'

————————————————————



- '[object Object]' is because we don't get a message, we get an array of errors



- but if we check our server side console, we see that this console log where i log an array of the errors that was gives me that array

```
// ./routes/auth.js

/**typically you wanna validate on your post or non-get routes
 * because you wanna validate whenever the use sends data
 * and that is not the case for our get routes for example.
 */

 /**you can use a feature called 'destructuring'
  * where we use curly braces on the left side of the equal sign
  * then you add some property names
  * which you wanna pull out of the object that 'express-validator/check' give you
  */
const express = require('express');
const { check } = require('express-validator/check')

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);

/**field name in 'check('')'
 * so this tell this middleware that the express-validator
 * that i'm insterested in confirming that e-mail value and so on.
 *
 * 'isEmail()' is a built-in methods in express-validator
 * which looks for that field in the body, query parameters, headers, cookies
 * and if finds that field and then checks if that is valid email address.
```

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

router.post('/logout', authController.postLogout);

router.get('/reset', authController.getReset);

router.post('/reset', authController.postReset);

router.get('/reset/:token', authController.getNewPassword);

router.post('/new-password', authController.postNewPassword);

module.exports = router;
```

```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');
/**'validationResult' will be a function that allows us to gather all the errors
 * prior validation middleware
 */
const { validationResult } = require('express-validator/check')

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
      api_key:
      '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'
    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
```

```
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.');
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
          }
          req.flash('error', 'Invalid email or password.');
          res.redirect('/login');
        })
        .catch(err => {
          console.log(err);
          res.redirect('/login');
        });
    })
    .catch(err => console.log(err));
};

exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  const confirmPassword = req.body.confirmPassword;
  /**in the request,
   * this express-validator middleware
   * which we added in ./routes/auth.js file will have added errors that can be retrieved
   * in ./routes/auth.js, collecting errors
   * and this 'validationResult(req)' will go through that errors obejct
   * managed by that middleware on the request.
   * and will then collect them all in this errors constant.
   */
  const errors = validationResult(req)
  if(!errors.isEmpty()){
    console.log(errors.array())
    /**422 means a common status code for indicating that validation failed
```

```
    * it will still send a response just with this different status code
    */
    return res.status(422).render('auth/signup', {
       path: '/signup',
       pageTitle: 'Signup',
       errorMessage: errors.array()
     });
  }
  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);
    });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
```

```
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
    User.findOne({ email: req.body.email })
      .then(user => {
        if (!user) {
          req.flash('error', 'No account with that email found.');
          return res.redirect('/reset');
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/');
        transporter.sendMail({
          to: req.body.email,
          from: 'shop@node-complete.com',
          subject: 'Password reset',
          html: `
            <p>You requested a password reset</p>
            <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
new password.</p>
          `
        });
      })
      .catch(err => {
        console.log(err);
      });
  });
};

exports.getNewPassword = (req, res, next) => {
  const token = req.params.token;
  User.findOne({
    resetToken: token,
    resetTokenExpiration: { $gt: Date.now()}
  })
    .then(user => {
      let message = req.flash('error');
      if (message.length > 0) {
        message = message[0];
```

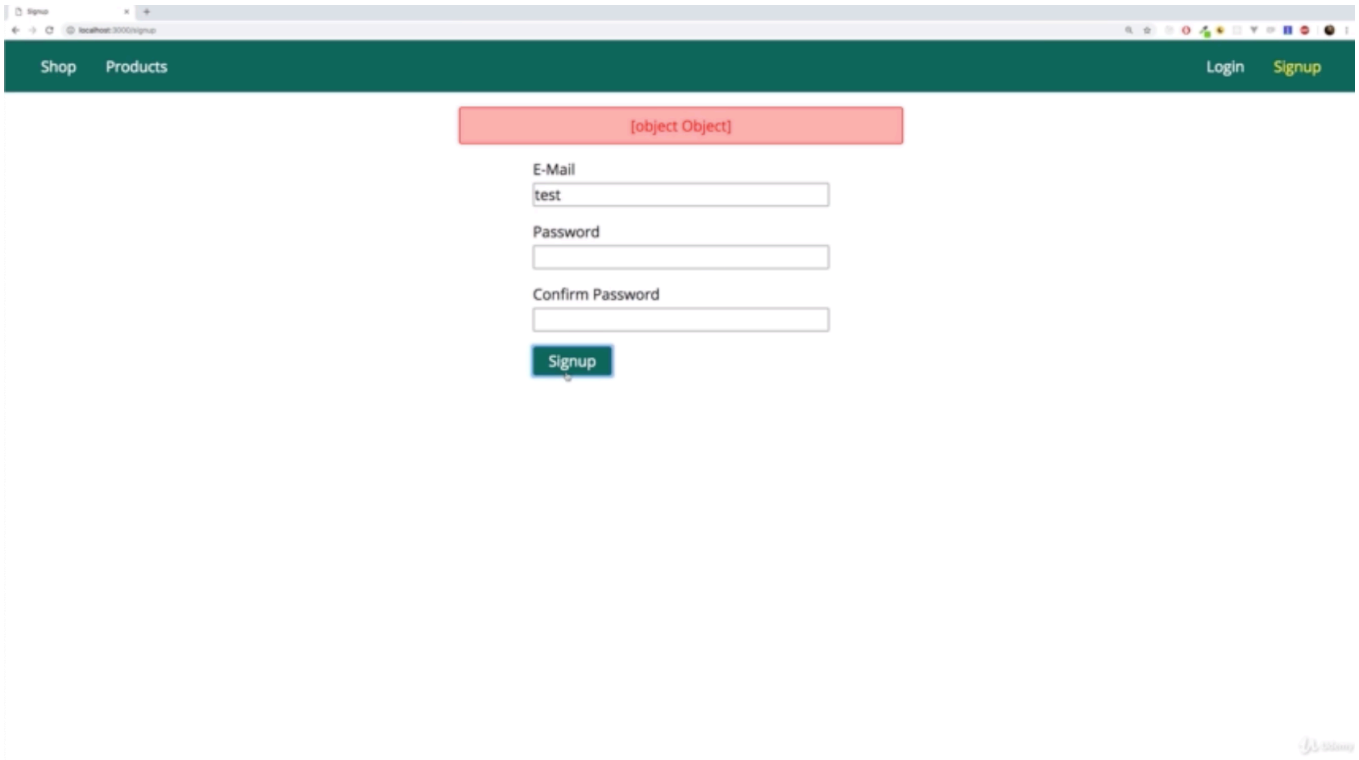
```
    } else {
      message = null;
    }
    res.render('auth/new-password', {
      path: '/new-password',
      pageTitle: 'New Password',
      errorMessage: message,
      userId: user._id.toString(),
      passwordToken: token
    });
  })
  .catch(err => {
    console.log(err);
  });
};

exports.postNewPassword = (req, res, next) => {
  /**i still wanna have that token
   * because otherwise people could start entering random tokens
   * and still reach that page and then maybe change users on the back
   * and by entering random userIds and that hidden input field as well.
   * so i wanna have that token.
   */
  const newPassword = req.body.password
  const userId = req.body.userId
  const passwordToken = req.body.passwordToken
  let resetUser

  User.findOne({
    resetToken: passwordToken,
    resetTokenExpiration: {$gt: Date.now()},
    _id: userId
  })
    .then(user => {
      resetUser = user
      return bcrypt.hash(newPassword, 12)
    })
    .then(hashedPassword => {
      resetUser.resetToken = undefined
      resetUser.resetTokenExpiration = undefined
      return resetUser.save()
    })
    .then(result => {
      res.redirect('/login')
    })
    .catch(err => {
      console.log(err)
    })

}
```

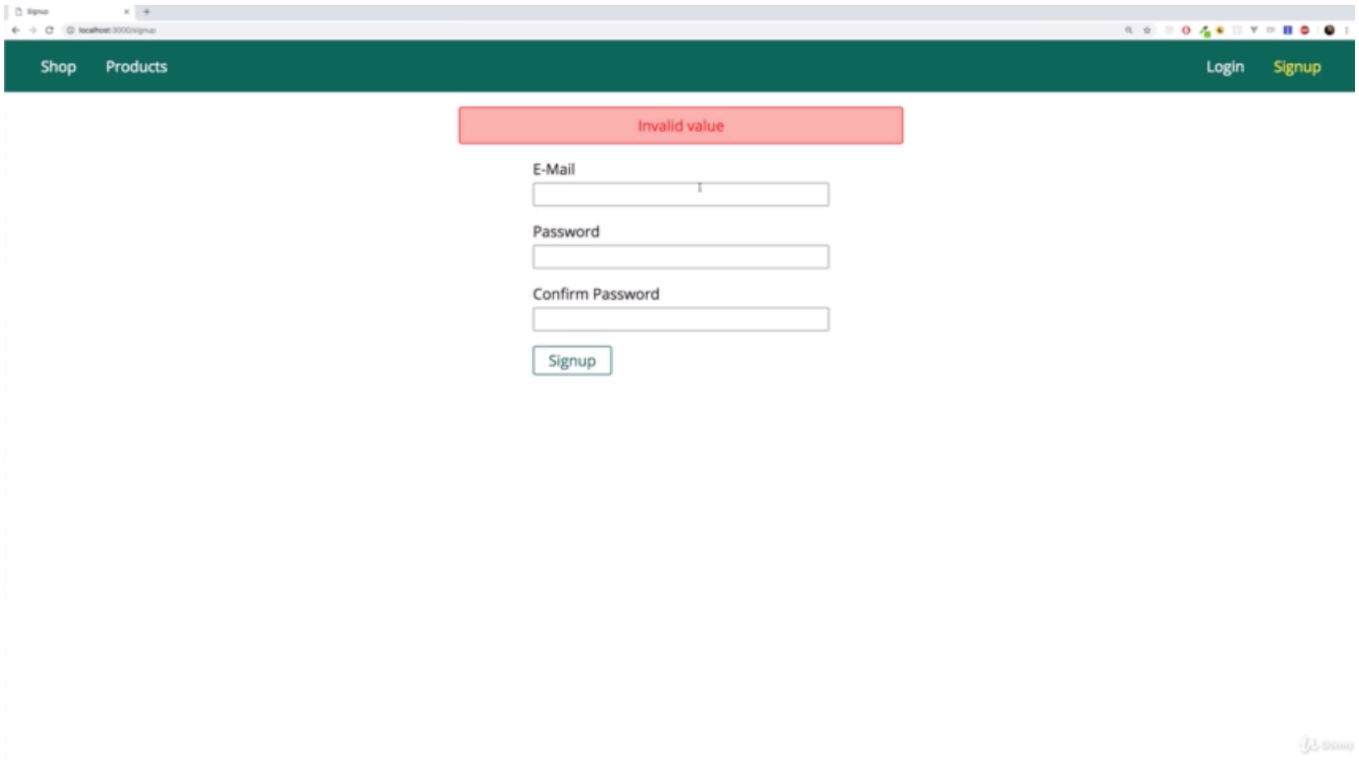
```
<!--./views/auth/signup.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="stylesheet" href="/css/auth.css">
```

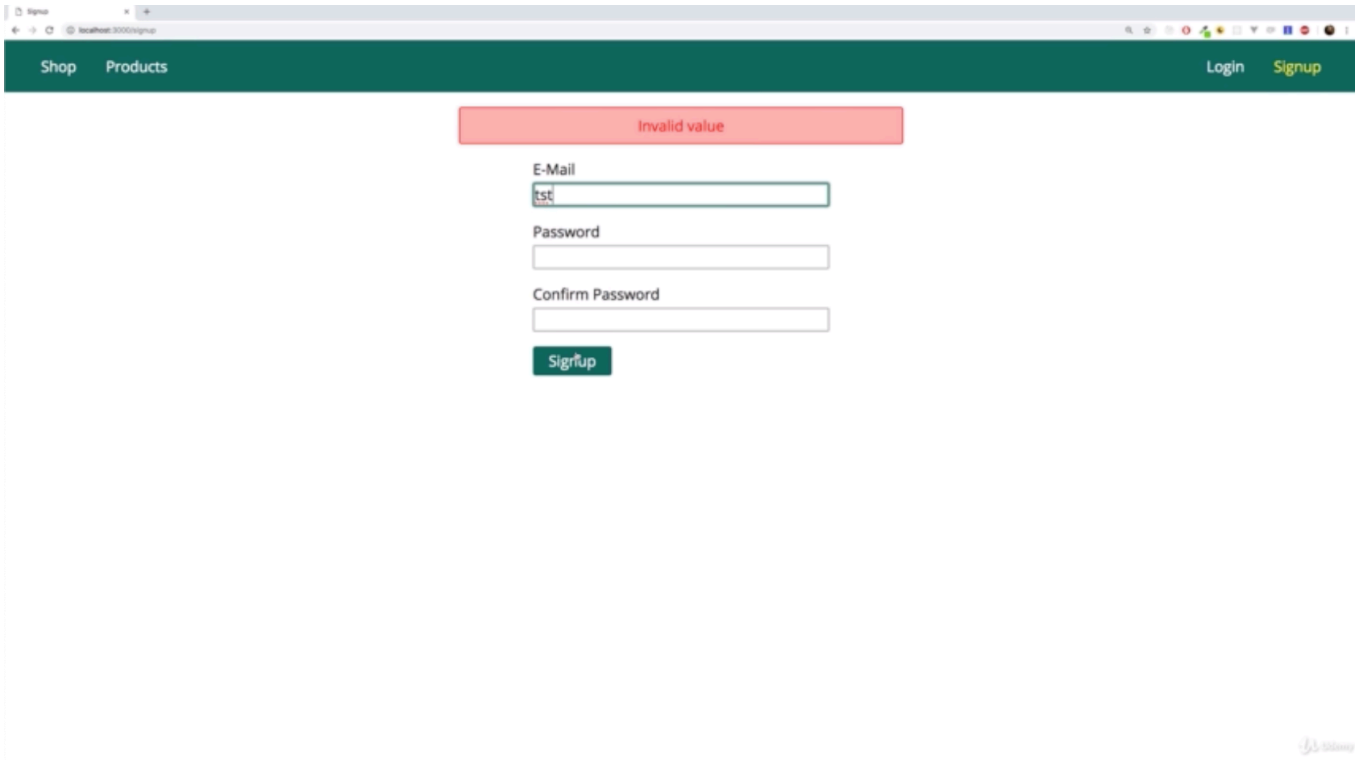
```
</head>

<body>
   <%- include('../includes/navigation.ejs') %>

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <!--you can add 'noValidate' to the overall form,
        validate to disable this check.-->
        <form class="login-form" action="/signup" method="POST" novalidate>
            <div class="form-control">
                <label for="email">E-Mail</label>
                <input type="email" name="email" id="email">
            </div>
            <div class="form-control">
                <label for="password">Password</label>
                <input type="password" name="password" id="password">
            </div>
            <div class="form-control">
                <label for="confirmPassword">Confirm Password</label>
                <input type="password" name="confirmPassword" id="confirmPassword">
            </div>
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Signup</button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

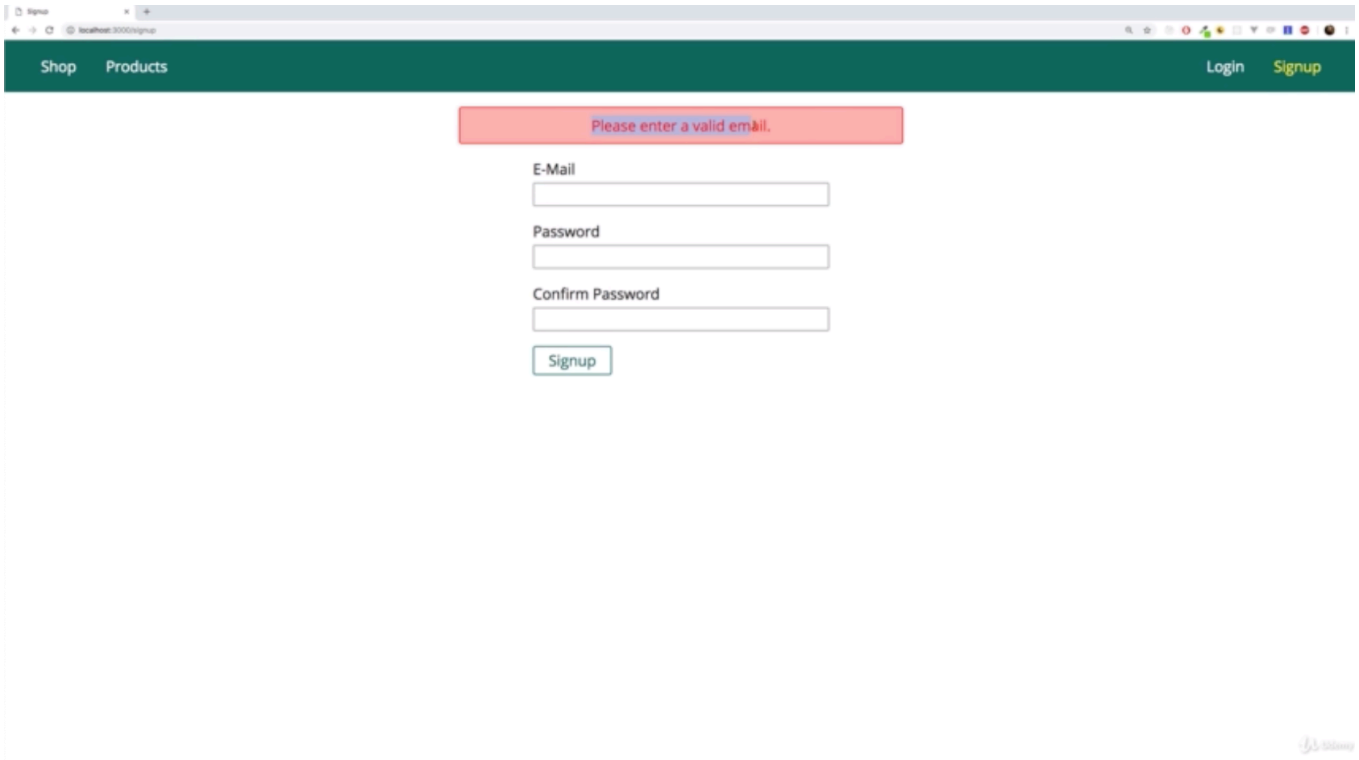
* Chapter 288: Using Validation Error Messages

1. update
- ./controllers/auth.js
- ./routes/auth.js


Shop
Products
Login
Signup
[object Object]
E-Mail
test
Password
Confirm Password
Signup


Shop
Products
Login
Signup
Invalid value
E-Mail
Password
Confirm Password
Signup






```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');
const { validationResult } = require('express-validator/check')

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
```

```
      api_key:
      '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'
    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.');
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
          }
          req.flash('error', 'Invalid email or password.');
          res.redirect('/login');
```

```
      })
      .catch(err => {
        console.log(err);
        res.redirect('/login');
      });
    })
    .catch(err => console.log(err));
};

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);
```

```
    });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
    User.findOne({ email: req.body.email })
      .then(user => {
        if (!user) {
          req.flash('error', 'No account with that email found.');
          return res.redirect('/reset');
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/');
        transporter.sendMail({
          to: req.body.email,
          from: 'shop@node-complete.com',
          subject: 'Password reset',
          html: `
            <p>You requested a password reset</p>
            <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
new password.</p>
          `
        });
      })
      .catch(err => {
        console.log(err);
```

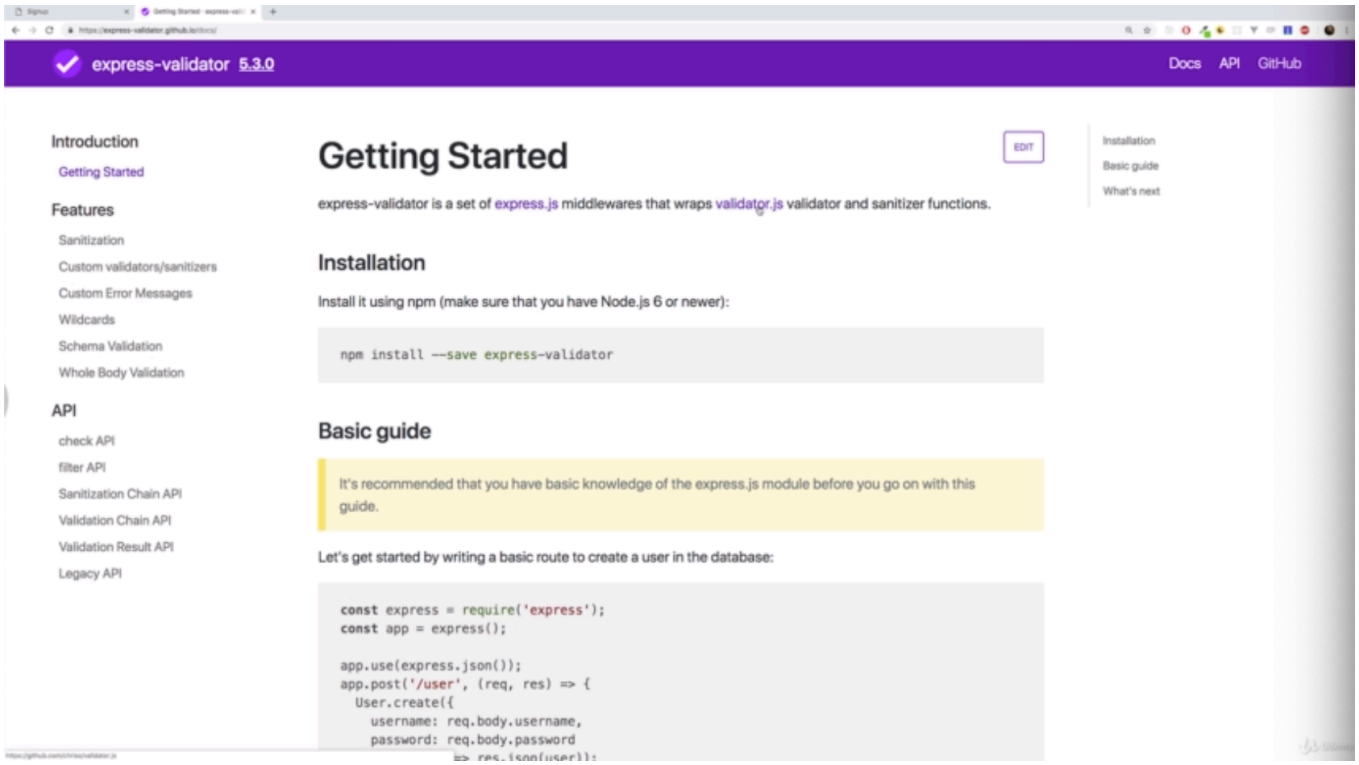
```
      });
  });
};

exports.getNewPassword = (req, res, next) => {
  const token = req.params.token;
  User.findOne({
    resetToken: token,
    resetTokenExpiration: { $gt: Date.now()}
  })
    .then(user => {
      let message = req.flash('error');
      if (message.length > 0) {
        message = message[0];
      } else {
        message = null;
      }
      res.render('auth/new-password', {
        path: '/new-password',
        pageTitle: 'New Password',
        errorMessage: message,
        userId: user._id.toString(),
        passwordToken: token
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.postNewPassword = (req, res, next) => {
  const newPassword = req.body.password
  const userId = req.body.userId
  const passwordToken = req.body.passwordToken
  let resetUser

  User.findOne({
    resetToken: passwordToken,
    resetTokenExpiration: {$gt: Date.now()},
    _id: userId
  })
    .then(user => {
      resetUser = user
      return bcrypt.hash(newPassword, 12)
    })
    .then(hashedPassword => {
      resetUser.resetToken = undefined
      resetUser.resetTokenExpiration = undefined
      return resetUser.save()
    })
    .then(result => {
      res.redirect('/login')
    })
    .catch(err => {
      console.log(err)
    })
```

```

}

```

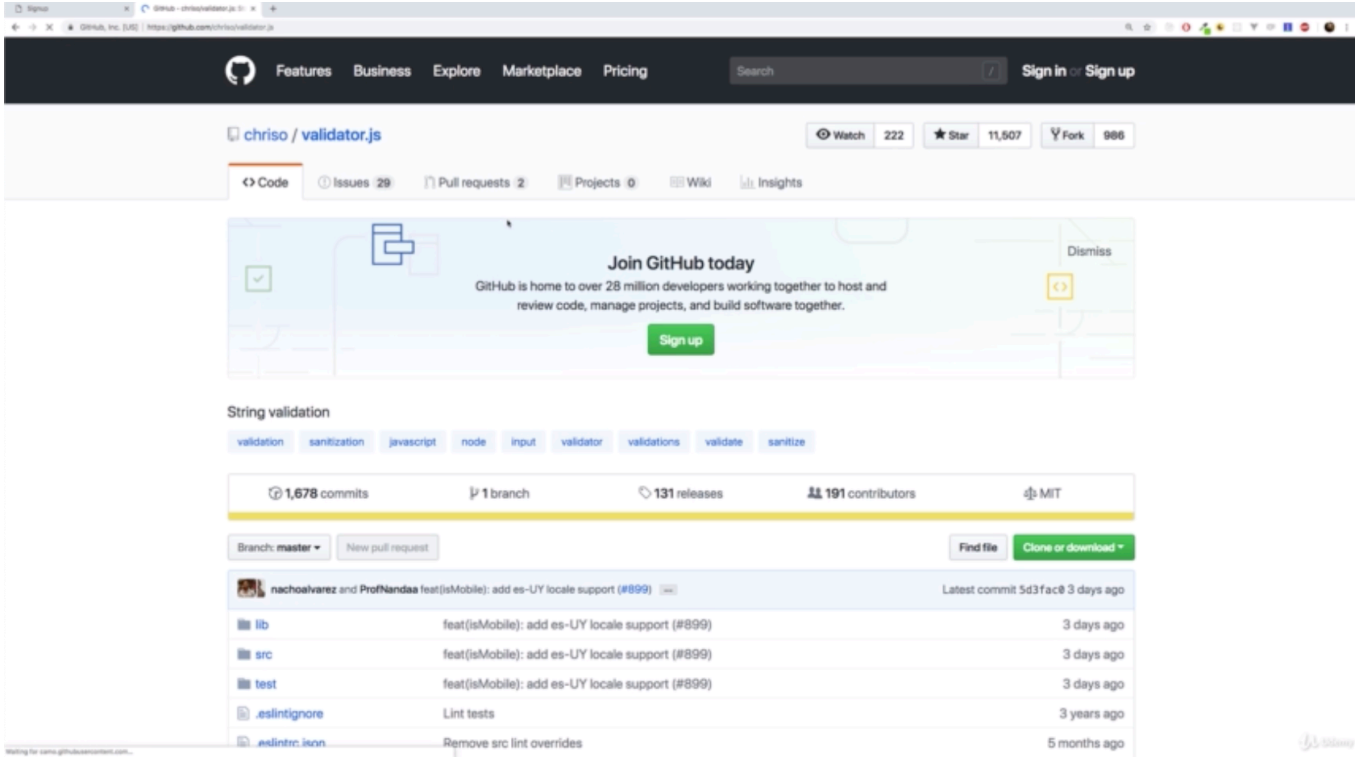
```
// ./routes/auth.js

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

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);
/**you could add multiple ones
 * but with that message,
 * we will always refer to the validation method right in front of it.
 */
router.post('/signup', check('email').isEmail().withMessage('Please enter a valid email'),
authController.postSignup);

router.post('/logout', authController.postLogout);

router.get('/reset', authController.getReset);

router.post('/reset', authController.postReset);

router.get('/reset/:token', authController.getNewPassword);

router.post('/new-password', authController.postNewPassword);

module.exports = router;
```

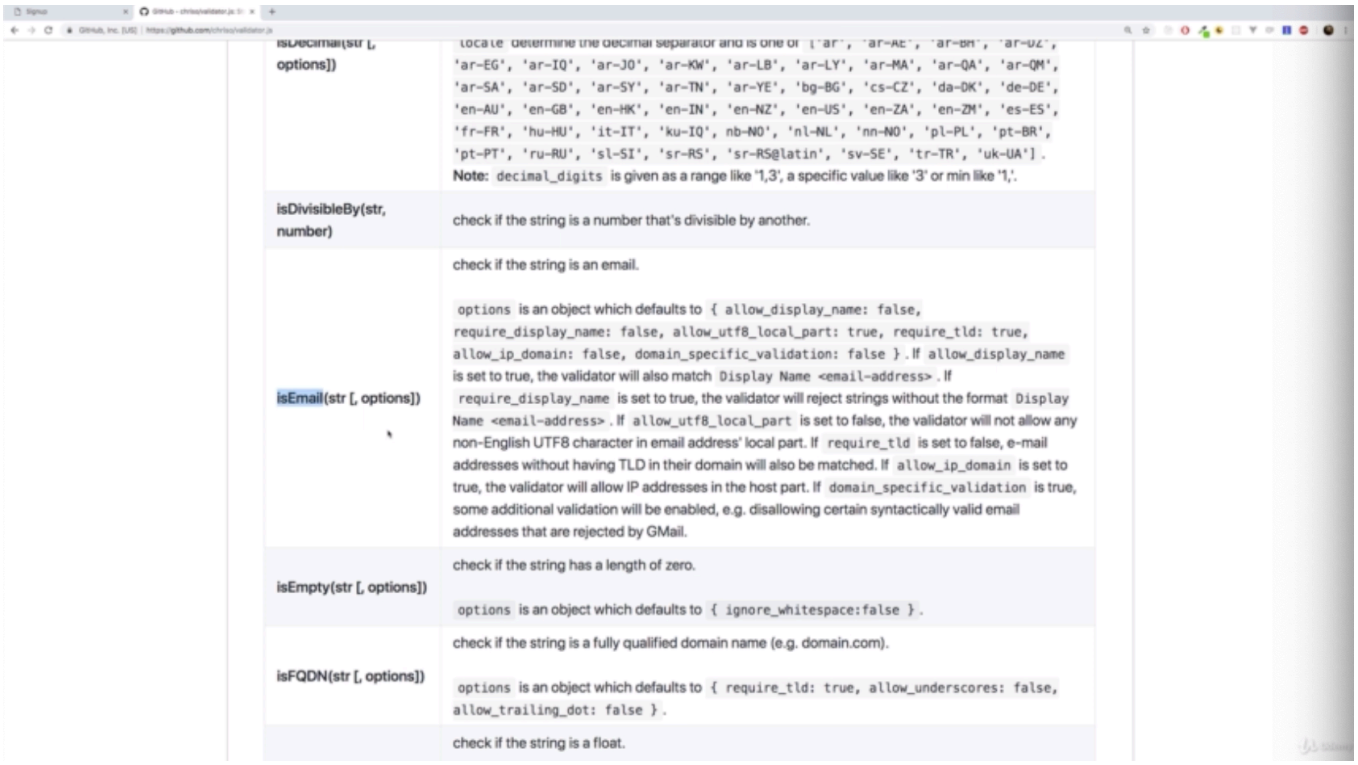
* Chapter 289: Built-in & Custom Validators

1. update
- ./routes/auth.js

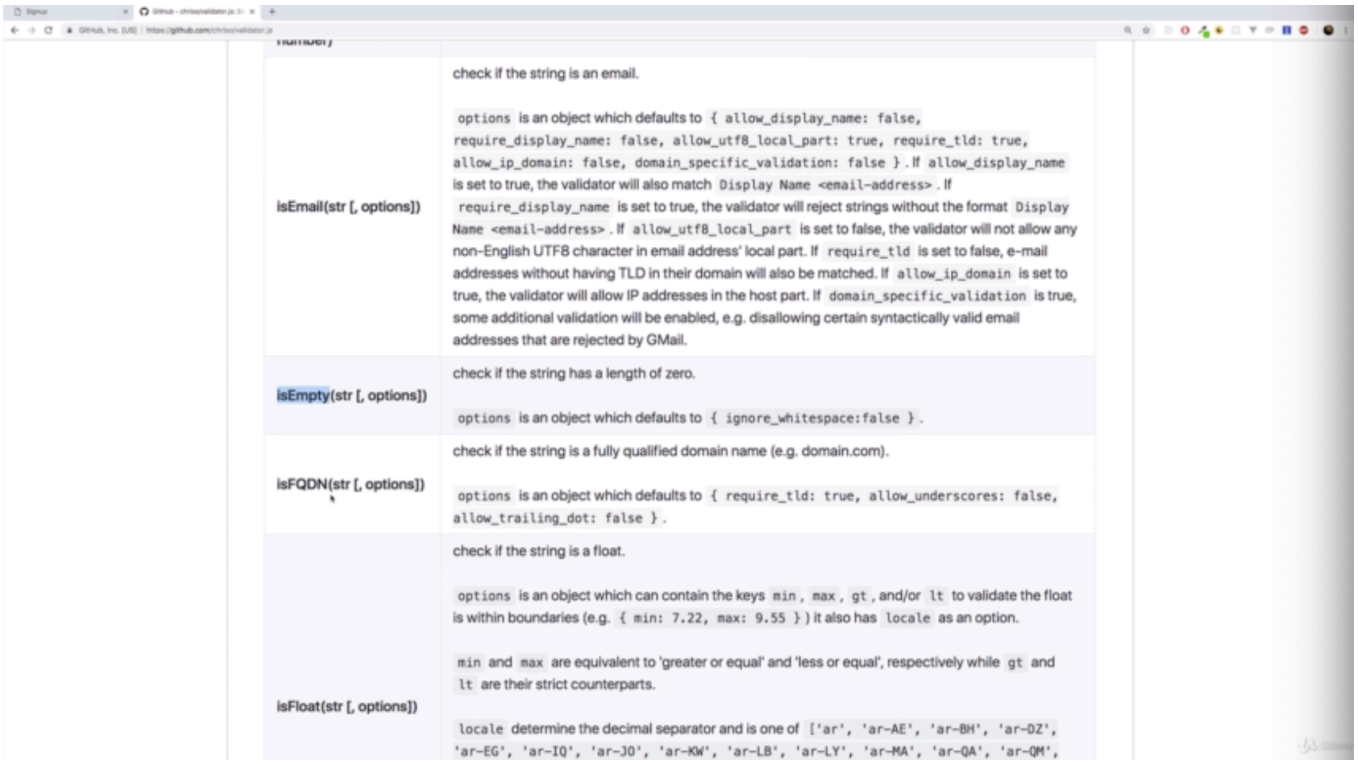


- 'validator.js' is another package that was implicitly installed with express-validator.



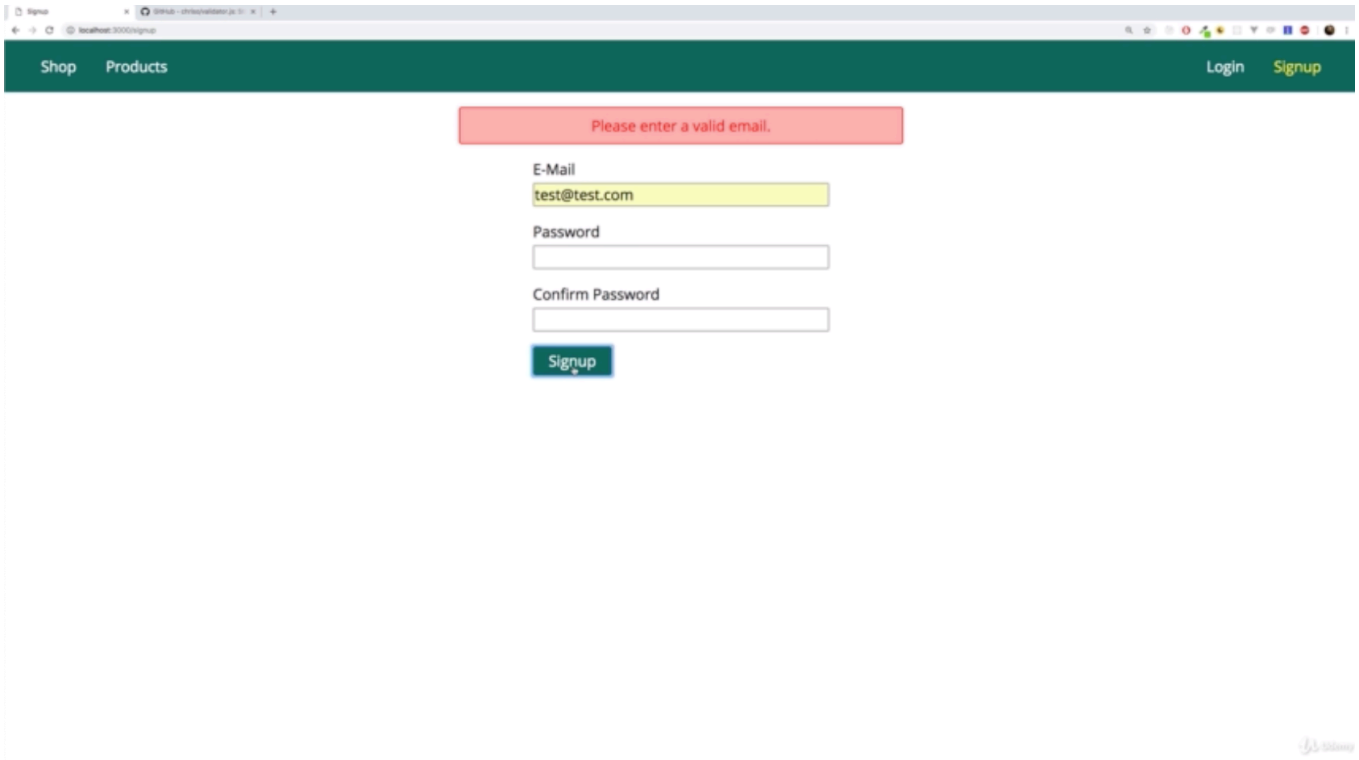


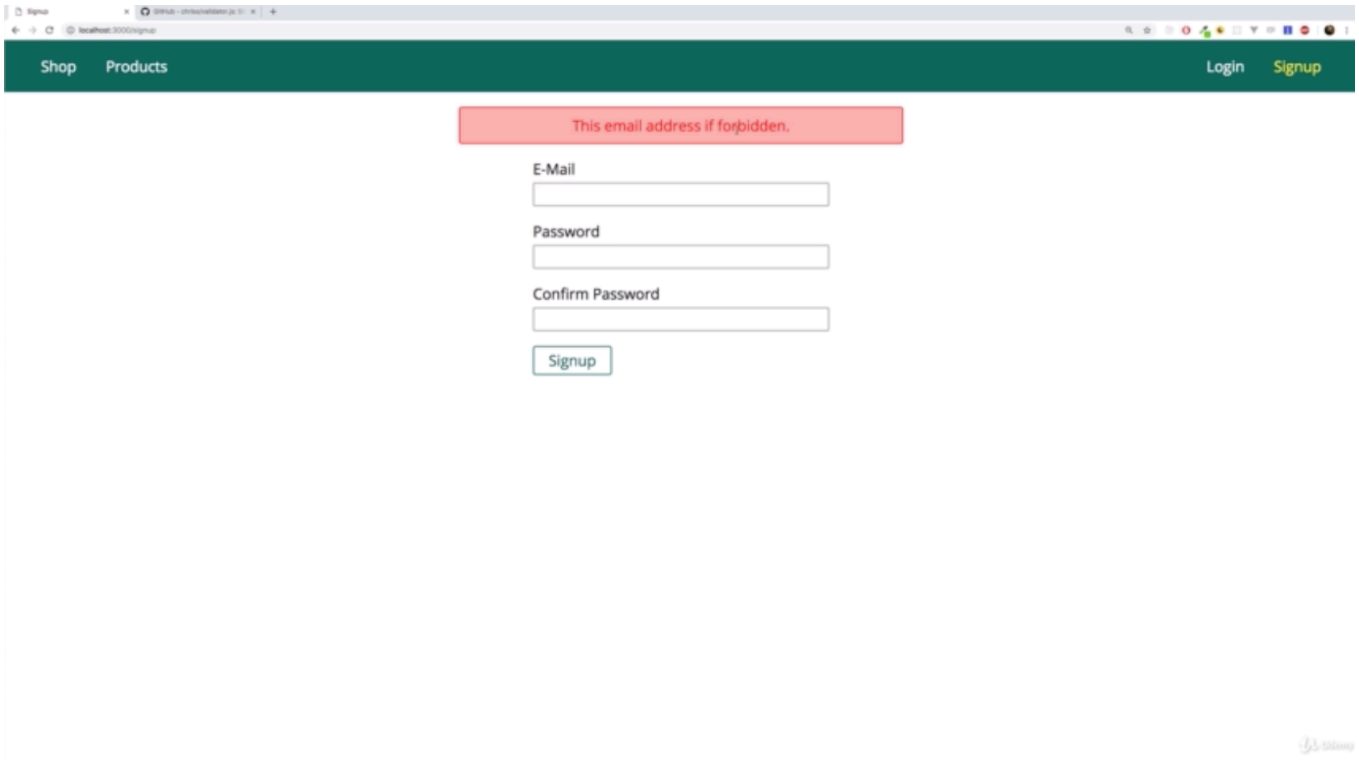
isDecimal(str [, options])	locale determine the decimal separator and is one of ['ar', 'ar-AE', 'ar-BH', 'ar-DZ', 'ar-EG', 'ar-IQ', 'ar-JO', 'ar-KW', 'ar-LB', 'ar-LY', 'ar-MA', 'ar-QA', 'ar-QM', 'ar-SA', 'ar-SD', 'ar-SY', 'ar-TN', 'ar-YE', 'bg-BG', 'cs-CZ', 'da-DK', 'de-DE', 'en-AU', 'en-GB', 'en-HK', 'en-IN', 'en-NZ', 'en-US', 'en-ZA', 'en-ZM', 'es-ES', 'fr-FR', 'hu-HU', 'it-IT', 'ku-IQ', nb-NO', 'nl-NL', 'nn-NO', 'pl-PL', 'pt-BR', 'pt-PT', 'ru-RU', 'sl-SI', 'sr-RS', 'sr-RS@latin', 'sv-SE', 'tr-TR', 'uk-UA'] . Note: decimal_digits is given as a range like '1,3', a specific value like '3' or min like '1,'.
isDivisibleBy(str, number)	check if the string is a number that's divisible by another.
isEmail(str [, options])	check if the string is an email. options is an object which defaults to { allow_display_name: false, require_display_name: false, allow_utf8_local_part: true, require_tld: true, allow_ip_domain: false, domain_specific_validation: false } . If allow_display_name is set to true, the validator will also match Display Name <email-address> . If require_display_name is set to true, the validator will reject strings without the format Display Name <email-address> . If allow_utf8_local_part is set to false, the validator will not allow any non-English UTF8 character in email address' local part. If require_tld is set to false, e-mail addresses without having TLD in their domain will also be matched. If allow_ip_domain is set to true, the validator will allow IP addresses in the host part. If domain_specific_validation is true, some additional validation will be enabled, e.g. disallowing certain syntactically valid email addresses that are rejected by GMail.
isEmpty(str [, options])	check if the string has a length of zero. options is an object which defaults to { ignore_whitespace:false } .
isFQDN(str [, options])	check if the string is a fully qualified domain name (e.g. domain.com). options is an object which defaults to { require_tld: true, allow_underscores: false, allow_trailing_dot: false } .
	check if the string is a float.

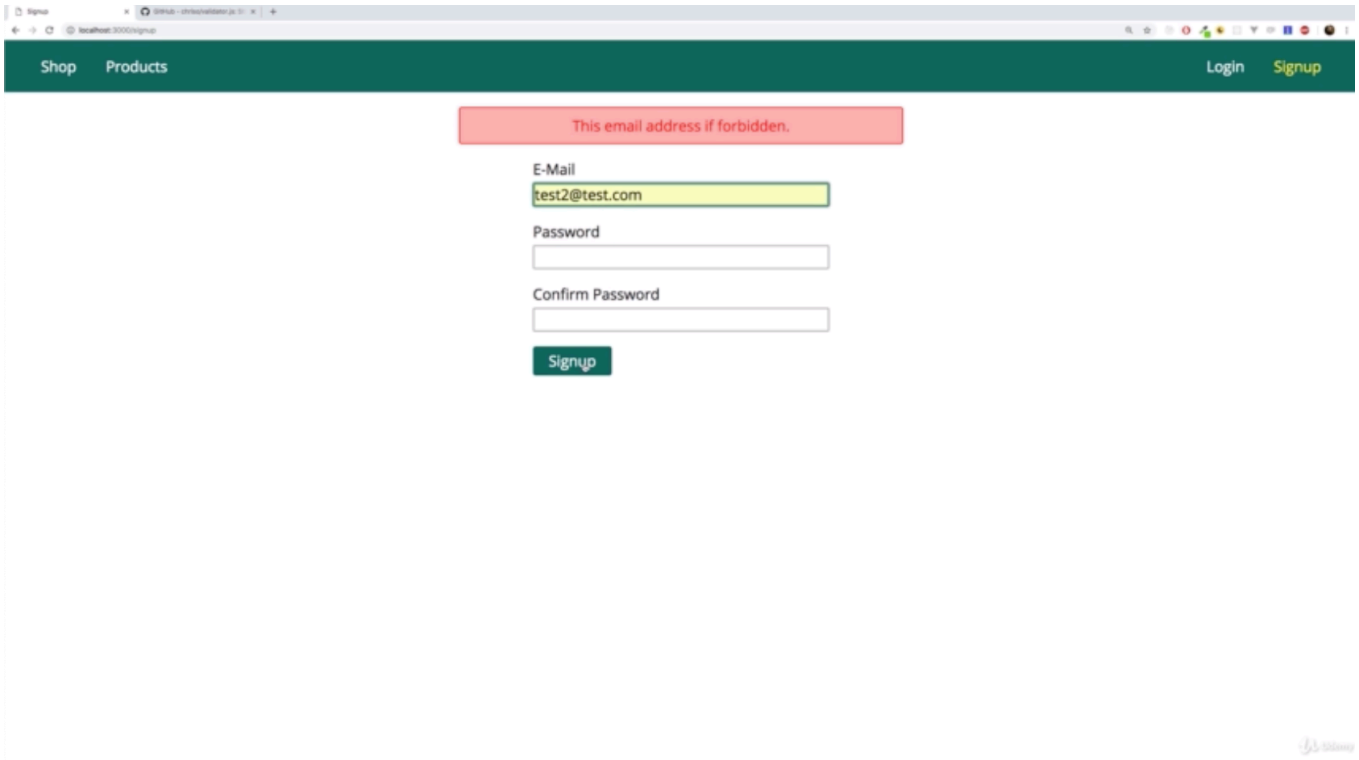


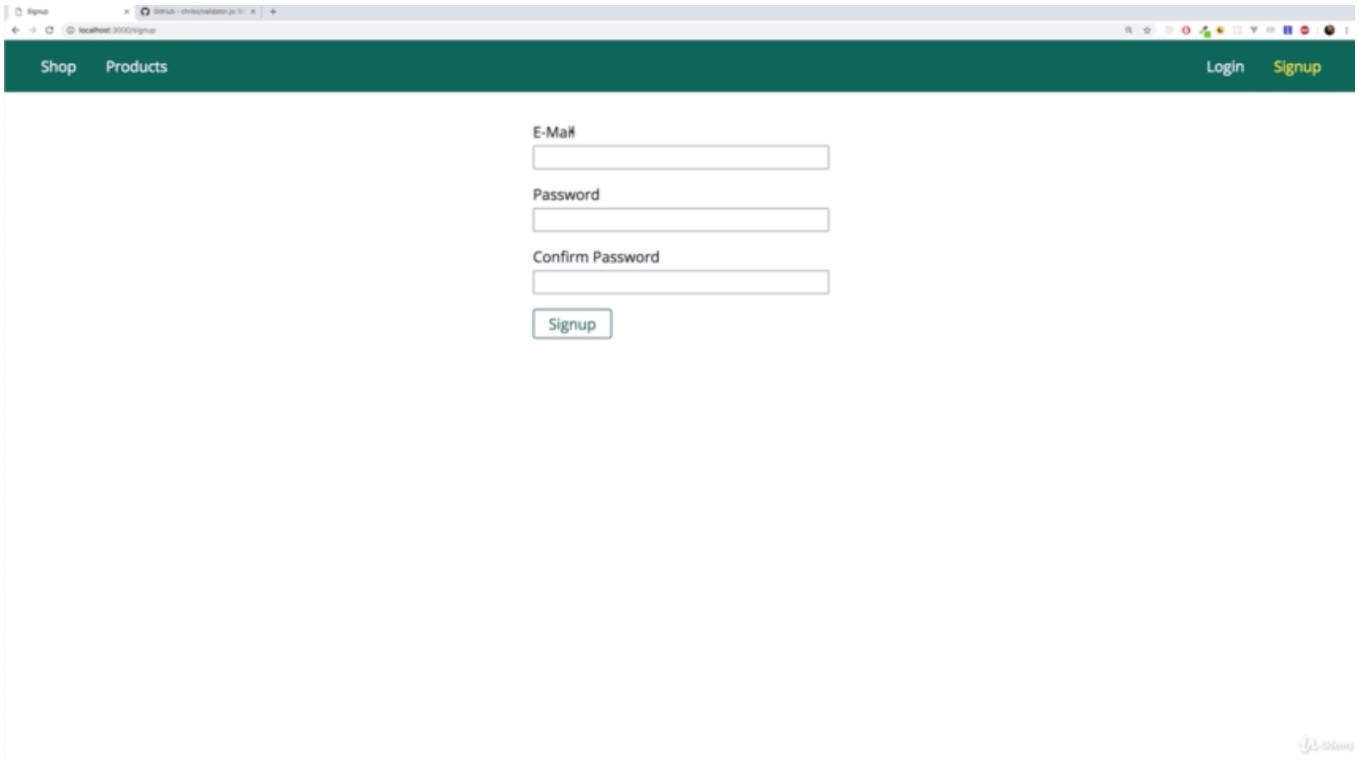
	check if the string is an email. options is an object which defaults to { allow_display_name: false, require_display_name: false, allow_utf8_local_part: true, require_tld: true, allow_ip_domain: false, domain_specific_validation: false } . If allow_display_name is set to true, the validator will also match Display Name <email-address> . If require_display_name is set to true, the validator will reject strings without the format Display Name <email-address> . If allow_utf8_local_part is set to false, the validator will not allow any non-English UTF8 character in email address' local part. If require_tld is set to false, e-mail addresses without having TLD in their domain will also be matched. If allow_ip_domain is set to true, the validator will allow IP addresses in the host part. If domain_specific_validation is true, some additional validation will be enabled, e.g. disallowing certain syntactically valid email addresses that are rejected by GMail.
isEmpty(str [, options])	check if the string has a length of zero. options is an object which defaults to { ignore_whitespace:false } .
isFQDN(str [, options])	check if the string is a fully qualified domain name (e.g. domain.com). options is an object which defaults to { require_tld: true, allow_underscores: false, allow_trailing_dot: false } .
isFloat(str [, options])	check if the string is a float. options is an object which can contain the keys min , max , gt , and/or lt to validate the float is within boundaries (e.g. { min: 7.22, max: 9.55 }) it also has locale as an option. min and max are equivalent to 'greater or equal' and 'less or equal', respectively while gt and lt are their strict counterparts. locale determine the decimal separator and is one of ['ar', 'ar-AE', 'ar-BH', 'ar-DZ', 'ar-EG', 'ar-IQ', 'ar-JO', 'ar-KW', 'ar-LB', 'ar-LY', 'ar-MA', 'ar-QA', 'ar-QM',

- there’s many thing and you can create your own validator


Shop
Products
Login
Signup
Please enter a valid email.
E-Mail
test@test.com
Password
Confirm Password
Signup


Shop
Products
Login
Signup
This email address if forbidden.
E-Mail
Password
Confirm Password
Signup





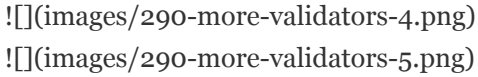
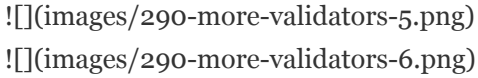
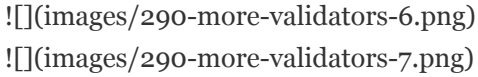
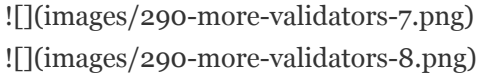
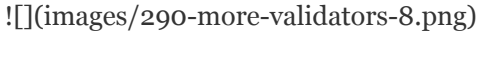

```
// ./routes/auth.js

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

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);
```

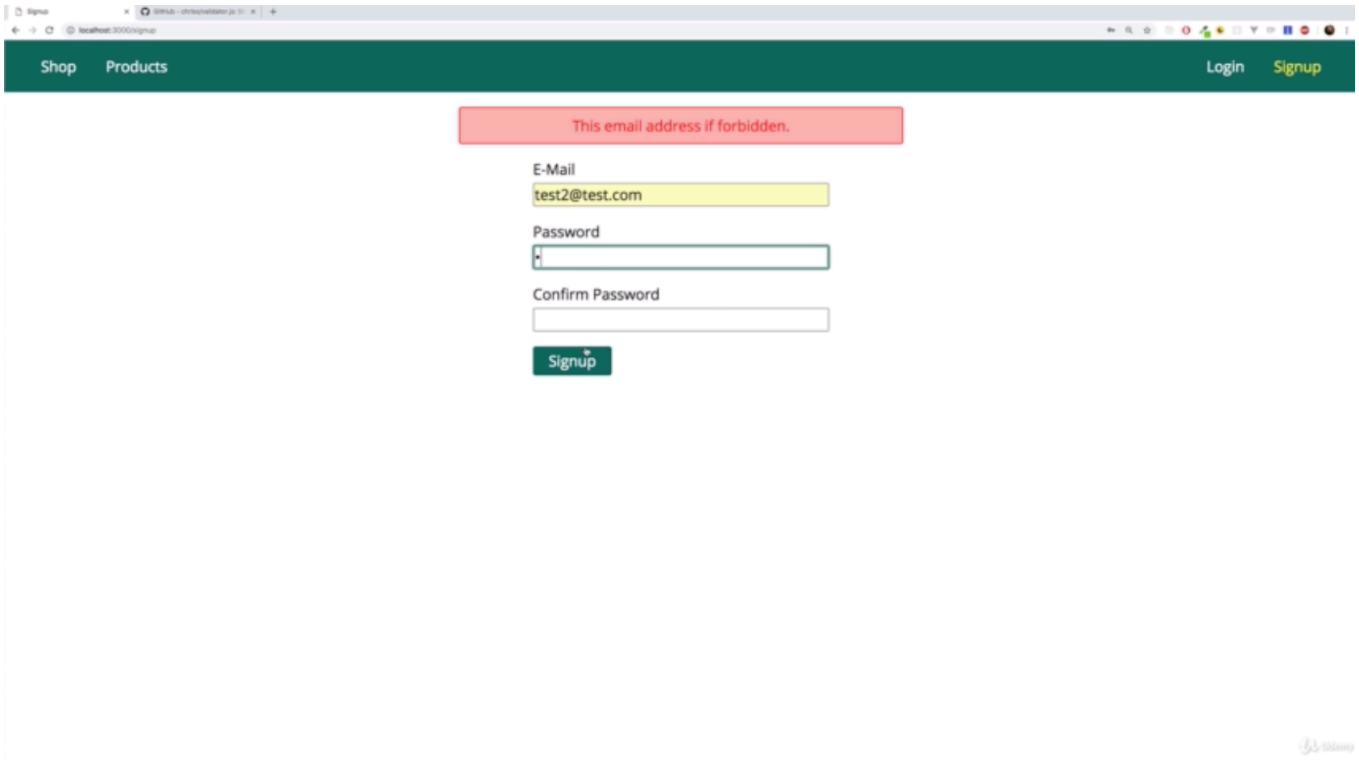
```
15
16 router.post(
17     '/signup',
18     check('email')
19         .isEmail()
20         .withMessage('Please enter a valid email')
21         .custom((value, {req}) => {
22             if (value === 'test@test.com'){
23                 throw new Error('This Email Address if forbidden.')
24             }
25             return true
26         }),
27     authController.postSignup);
28
29 router.post('/logout', authController.postLogout);
30
31 router.get('/reset', authController.getReset);
32
33 router.post('/reset', authController.postReset);
34
35 router.get('/reset/:token', authController.getNewPassword);
36
37 router.post('/new-password', authController.postNewPassword);
38
39 module.exports = router;
```

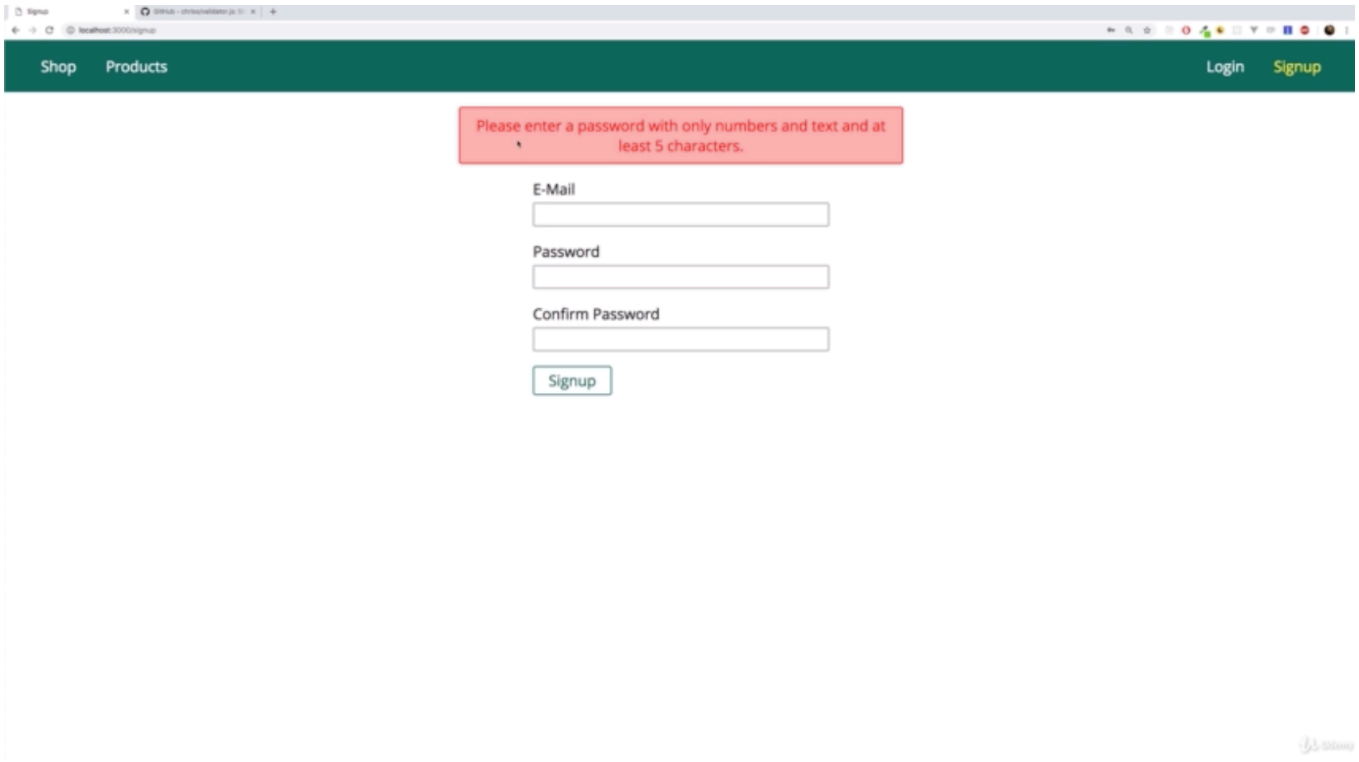
* Chapter 290: More Validators

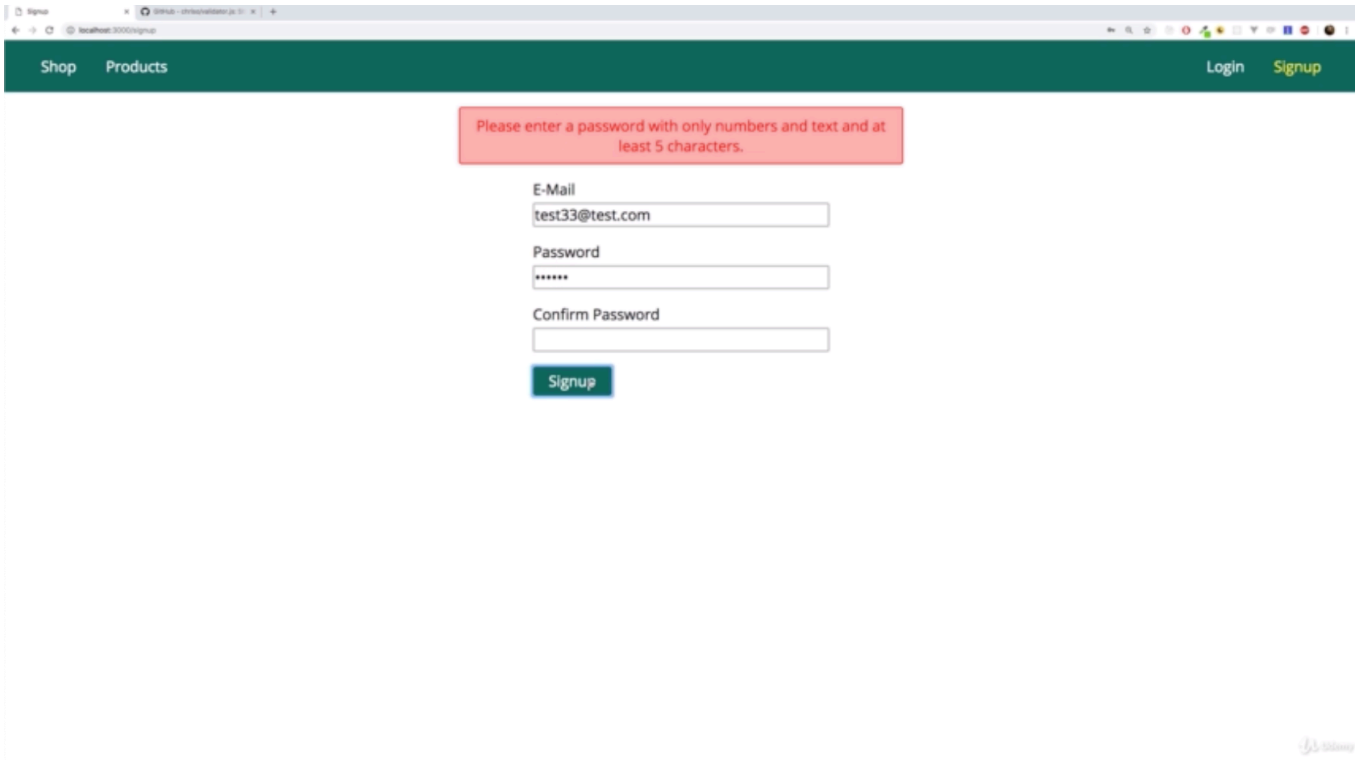
1. update
- ./routes/auth.js

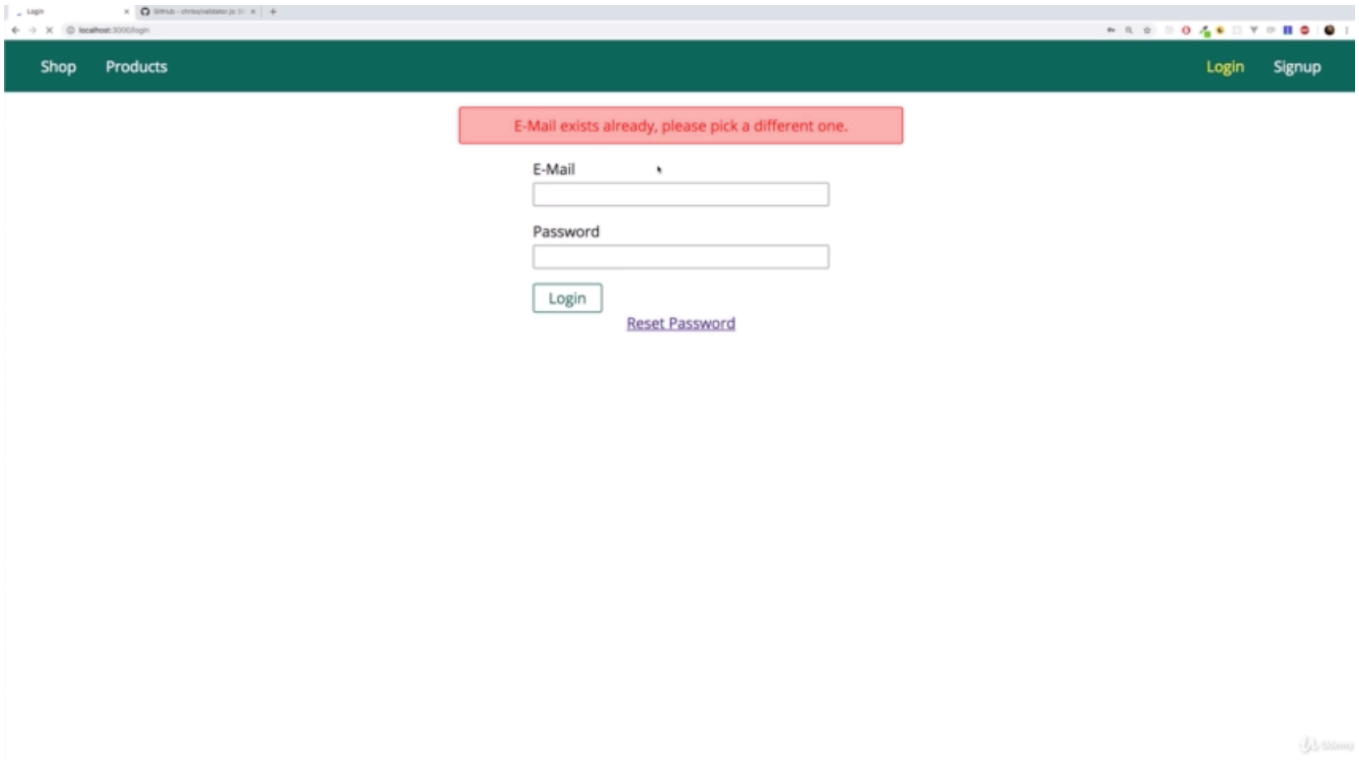


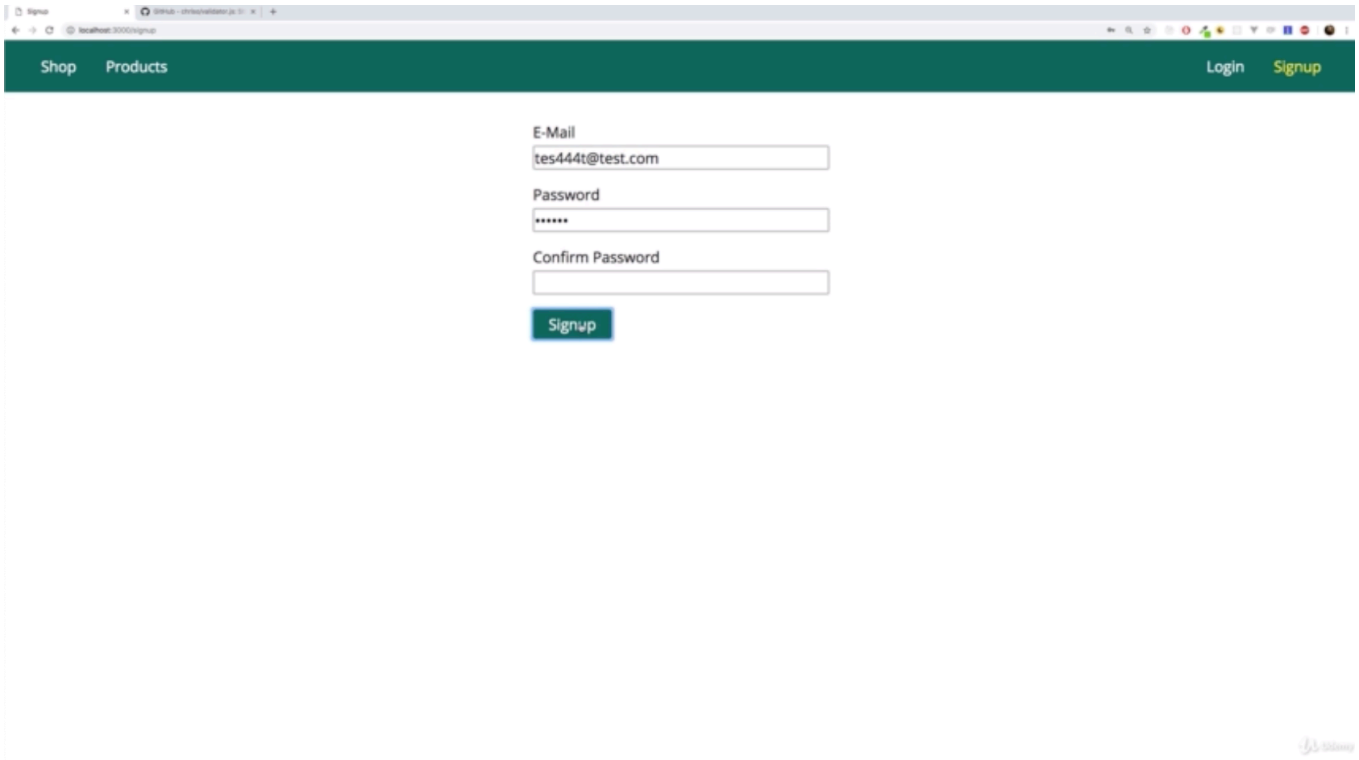







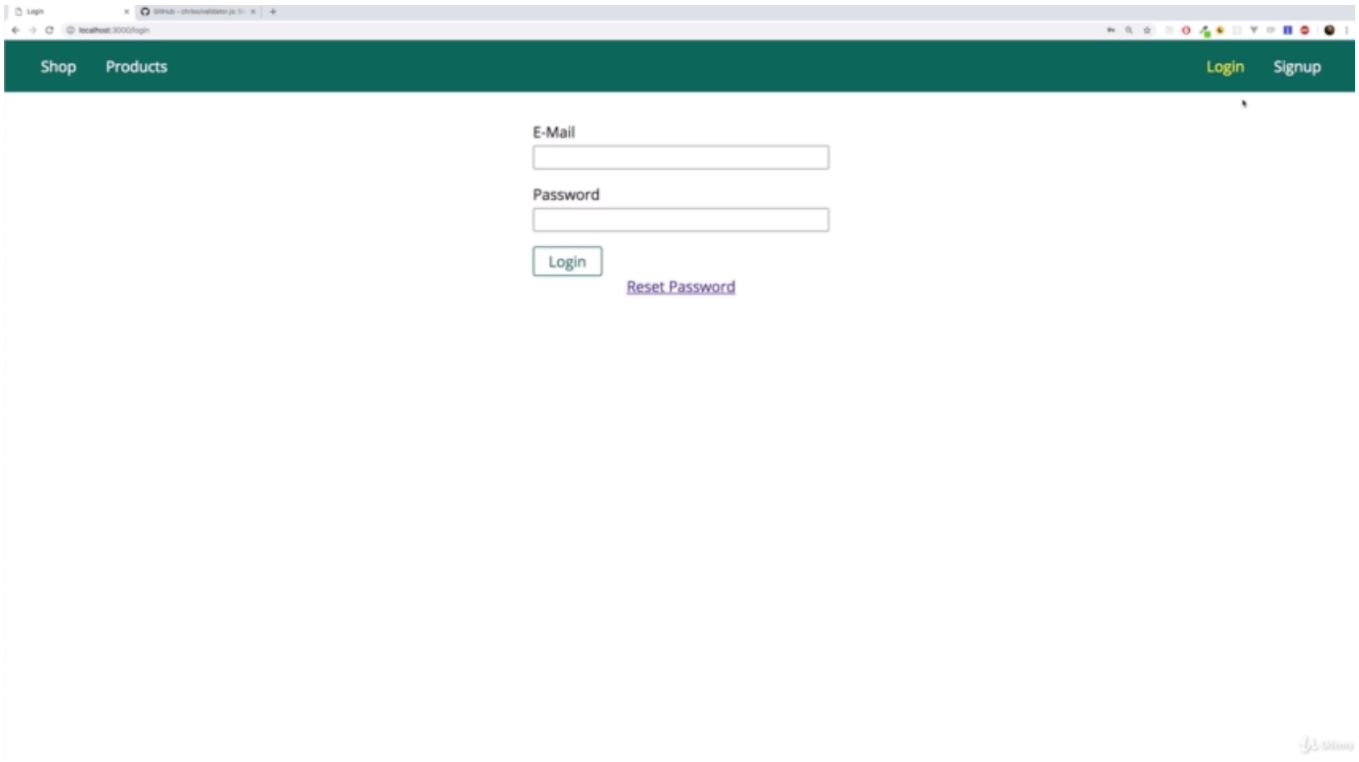

Shop
Products
Login
Signup
This email address if forbidden.
E-Mail
test2@test.com
Password
Confirm Password
Signup

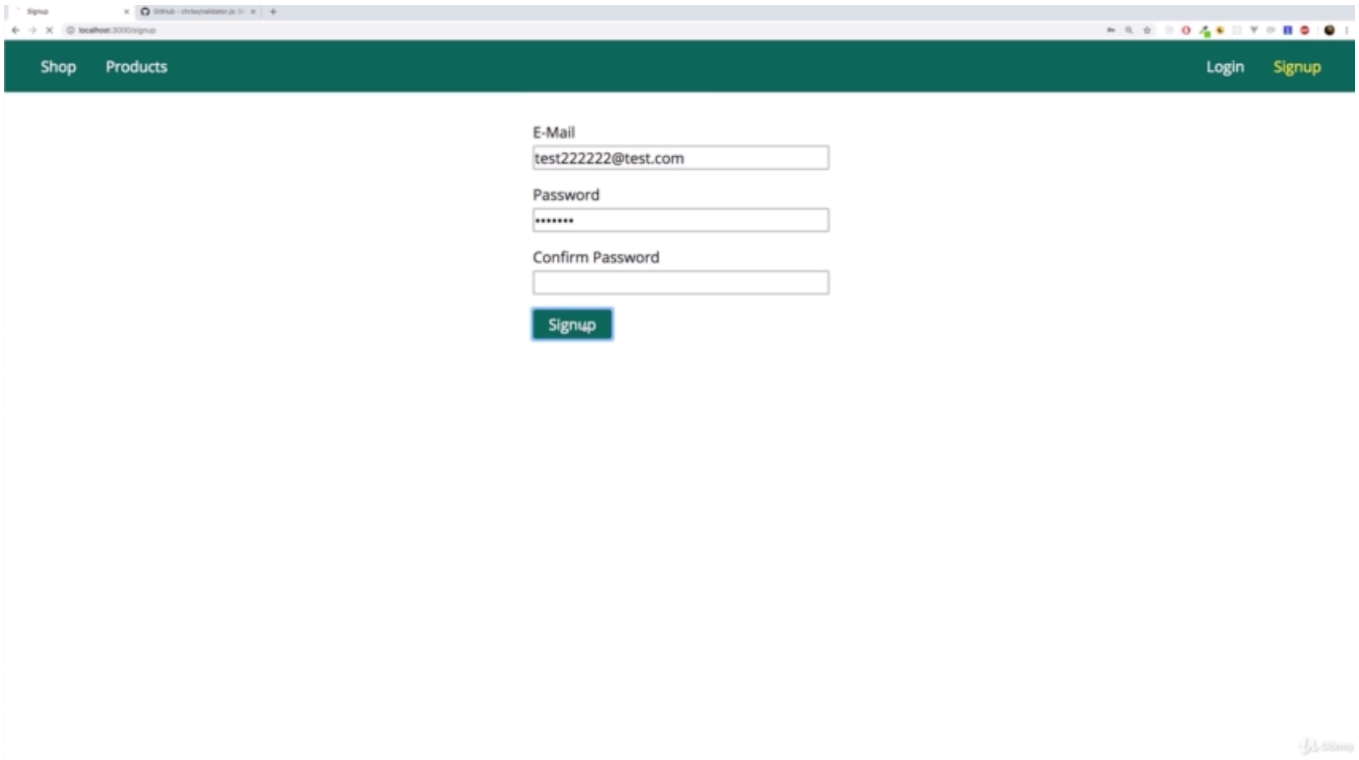

Shop
Products
Login
Signup
Please enter a password with only numbers and text and at least 5 characters.
E-Mail
Password
Confirm Password
Signup

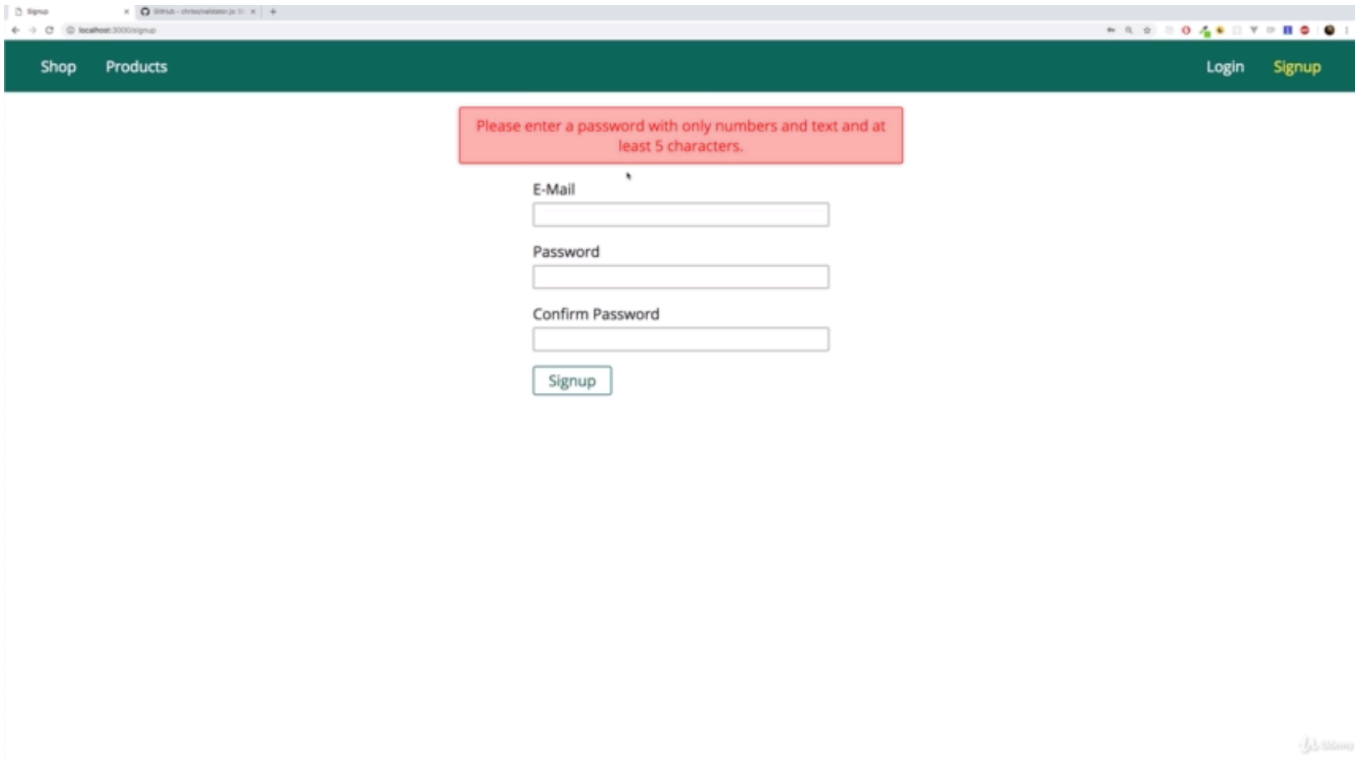

Shop
Products
Login
Signup
Please enter a password with only numbers and text and at least 5 characters.
E-Mail
test33@test.com
Password
Confirm Password
Signup


Shop
Products
Login
Signup
E-Mail already exists already, please pick a different one.
E-Mail
Password
Login
Reset Password


Shop
Products
Login
Signup
E-Mail
tes444t@test.com
Password
Confirm Password
Signup


Shop
Products
Login
Signup
E-Mail
Password
Login
Reset Password



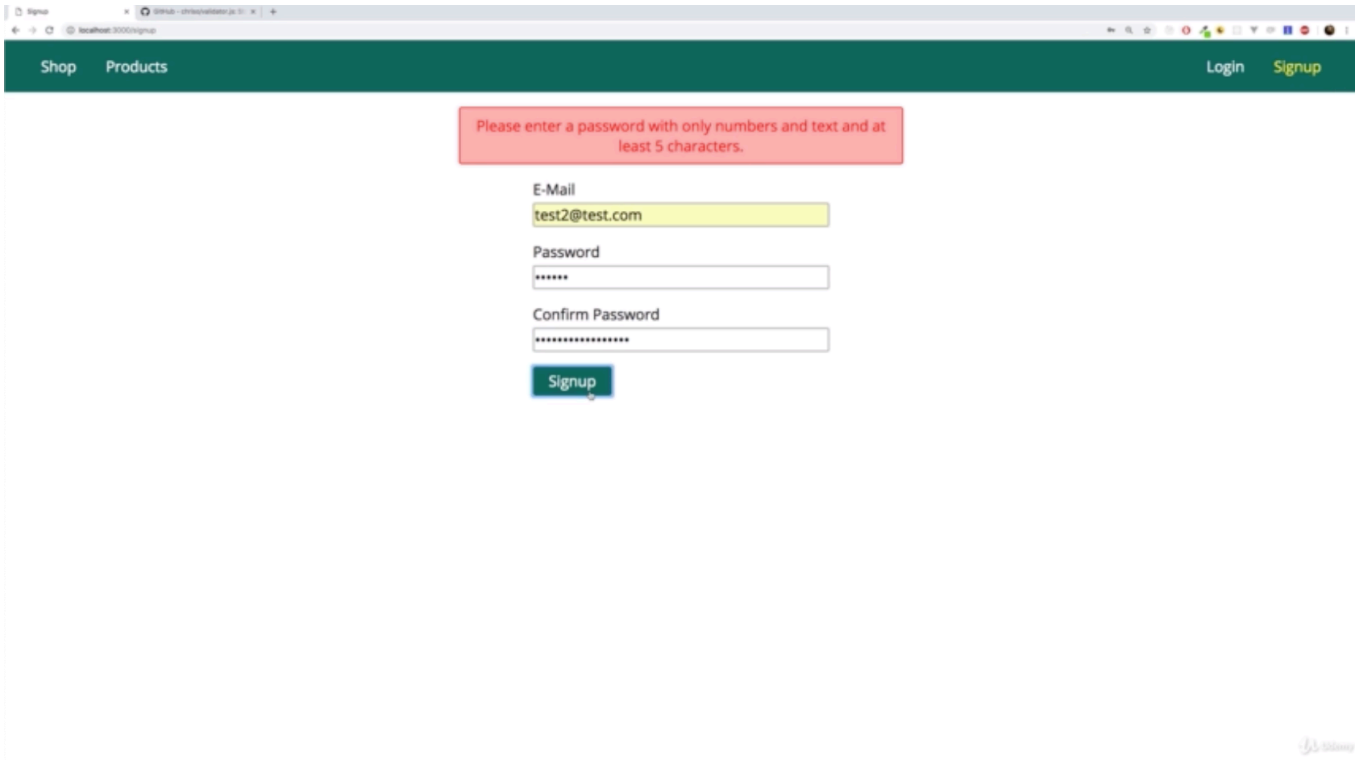


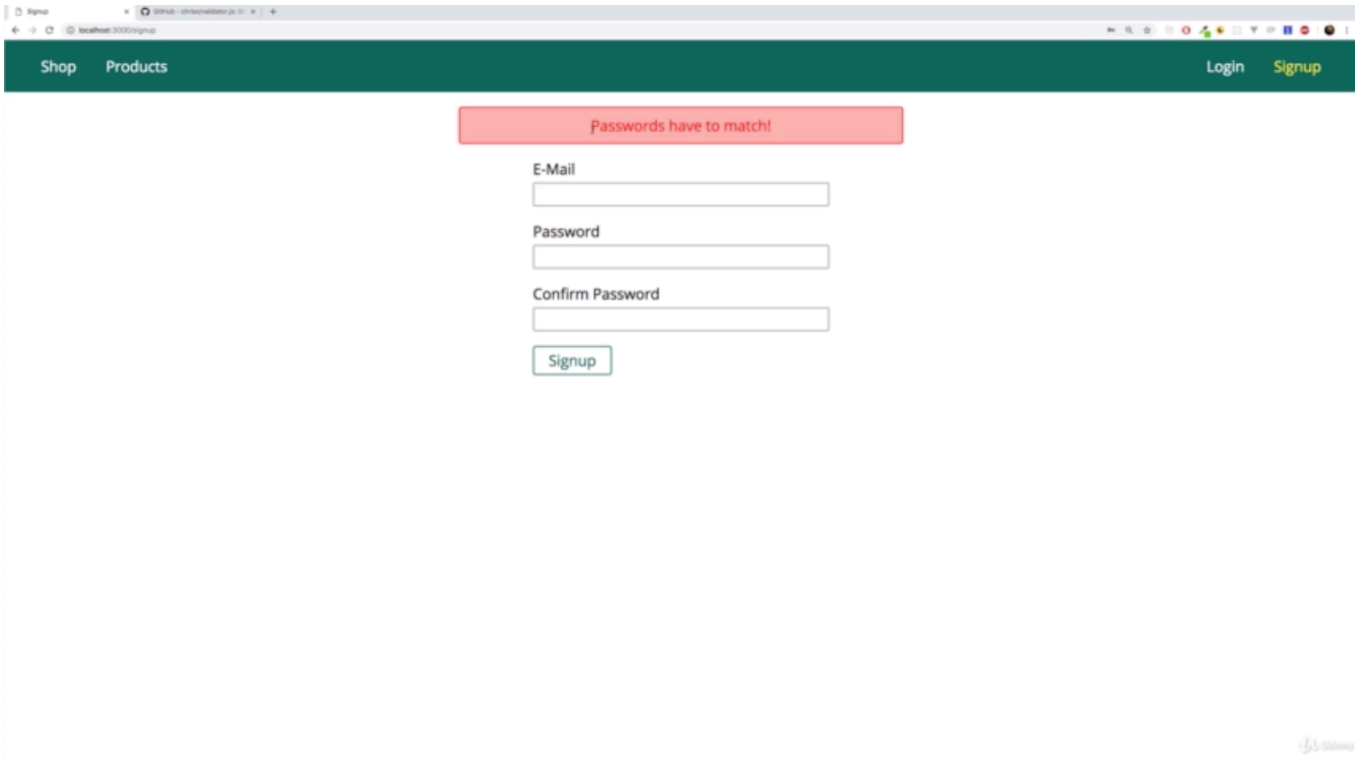

```
// ./routes/auth.js

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

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);
```

```
15
16 router.post(
17     '/signup',
18     [
19         check('email')
20             .isEmail()
21             .withMessage('Please enter a valid email')
22             .custom((value, {req}) => {
23                 if (value === 'test@test.com'){
24                     throw new Error('This Email Address if forbidden.')
25                 }
26                 return true
27             }),
28         /**this means please check password value in the body of the request
29          * so if there happens to be a password value in the headers,
30          * i don't care about that.
31         */
32         body(
33             'password',
34             /**if you want the same error message for all your validators
35              * but it should not be the default invalid value error message
36              * then grab error message in here the 2nd argument.
37              */
38             'Please enter a password with only numbers and text and at least 5 characters'
39         )
40             .isLength({min: 5})
41             .isAlphanumeric()
42     ],
43     authController.postSignup);
44
45 router.post('/logout', authController.postLogout);
46
47 router.get('/reset', authController.getReset);
48
49 router.post('/reset', authController.postReset);
50
51 router.get('/reset/:token', authController.getNewPassword);
52
53 router.post('/new-password', authController.postNewPassword);
54
55 module.exports = router;
```

* Chapter 291: Checking For Field Equality

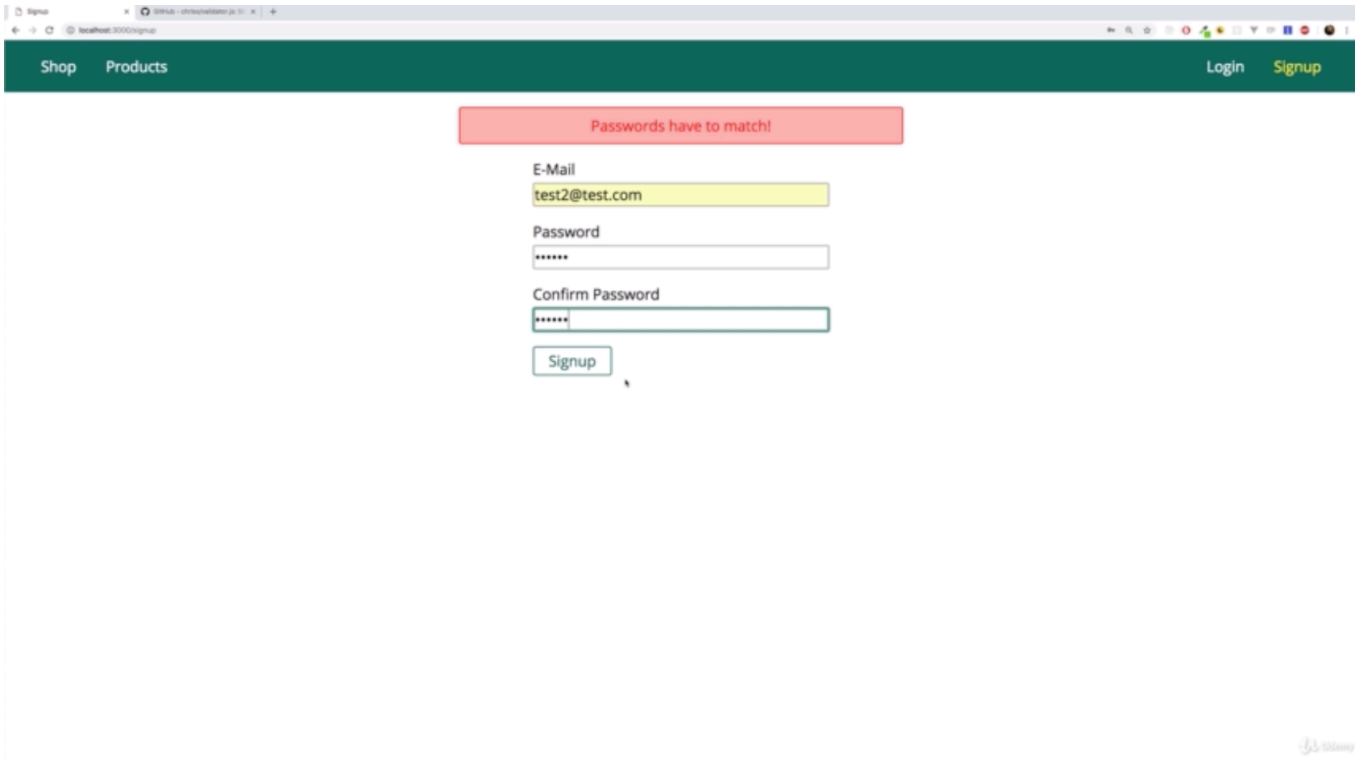
1. update
- ./routes/auth.js

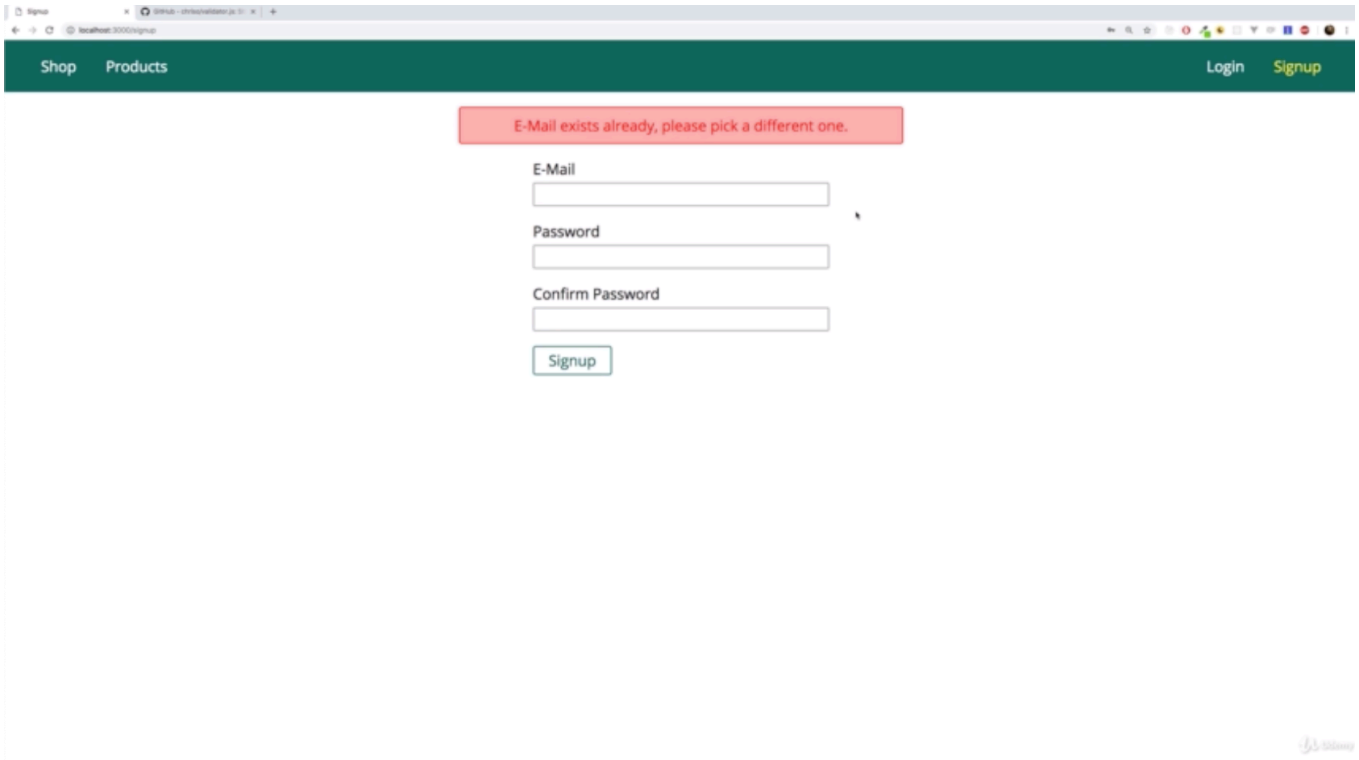




- ‘password’ and ‘confirm password’ is not equal

————————————————





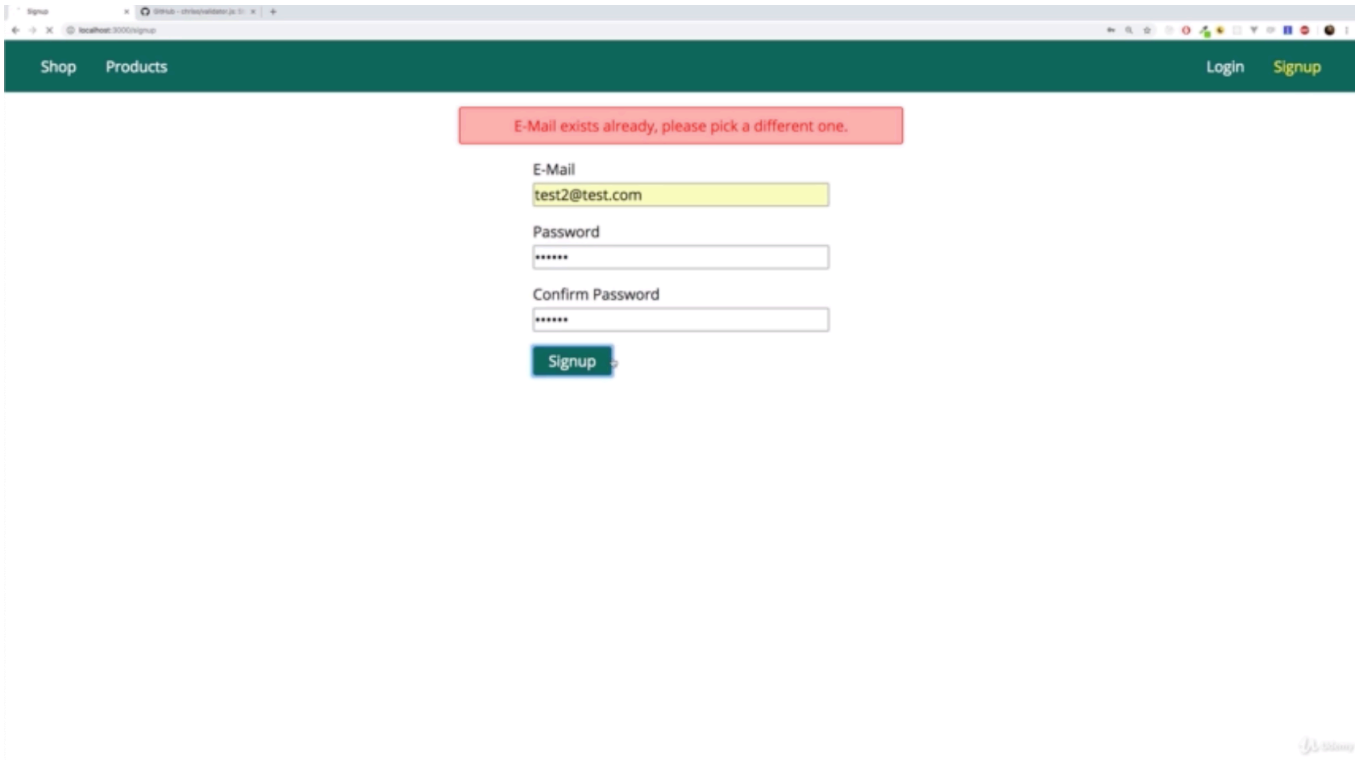
- ‘password' and ‘confirm password’ is equal.

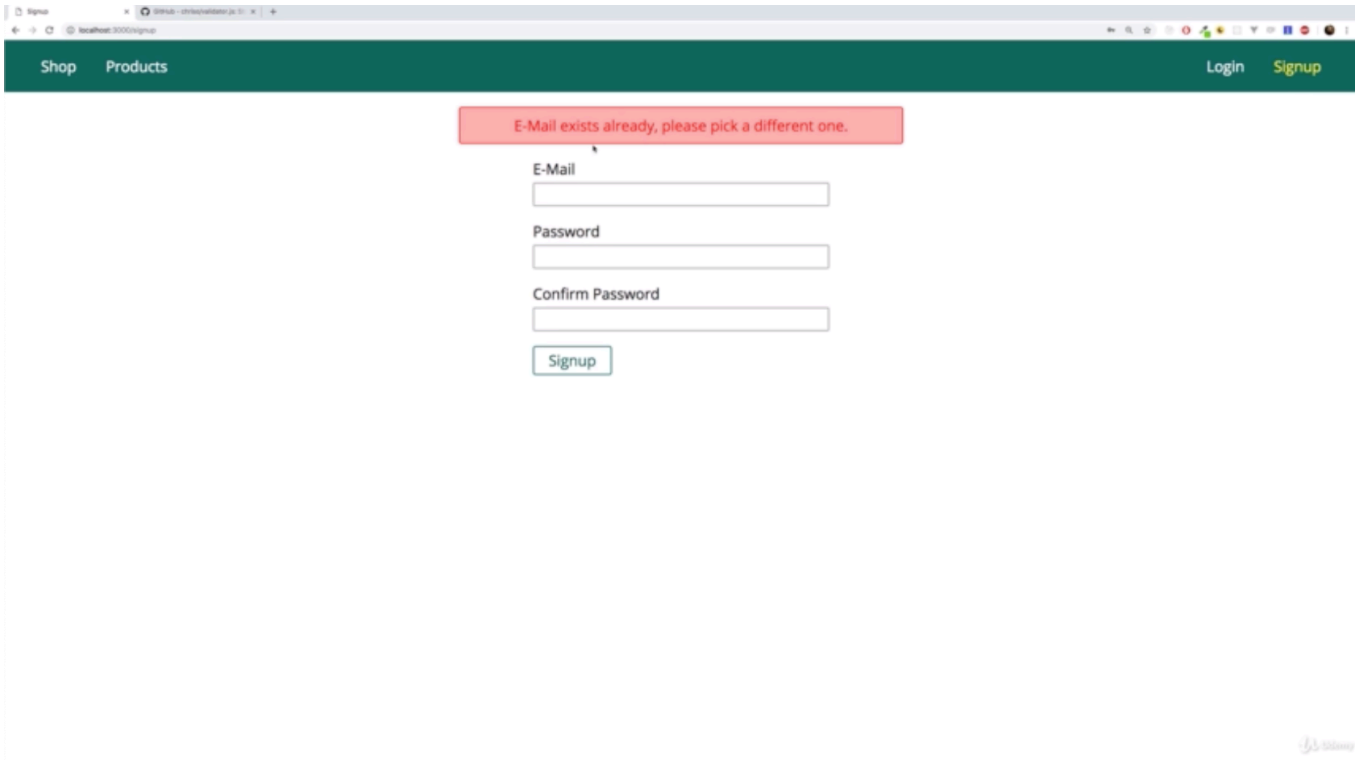
```
// ./routes/auth.js

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

const authController = require('../controllers/auth');

const router = express.Router();

router.get('/login', authController.getLogin);

router.get('/signup', authController.getSignup);

```

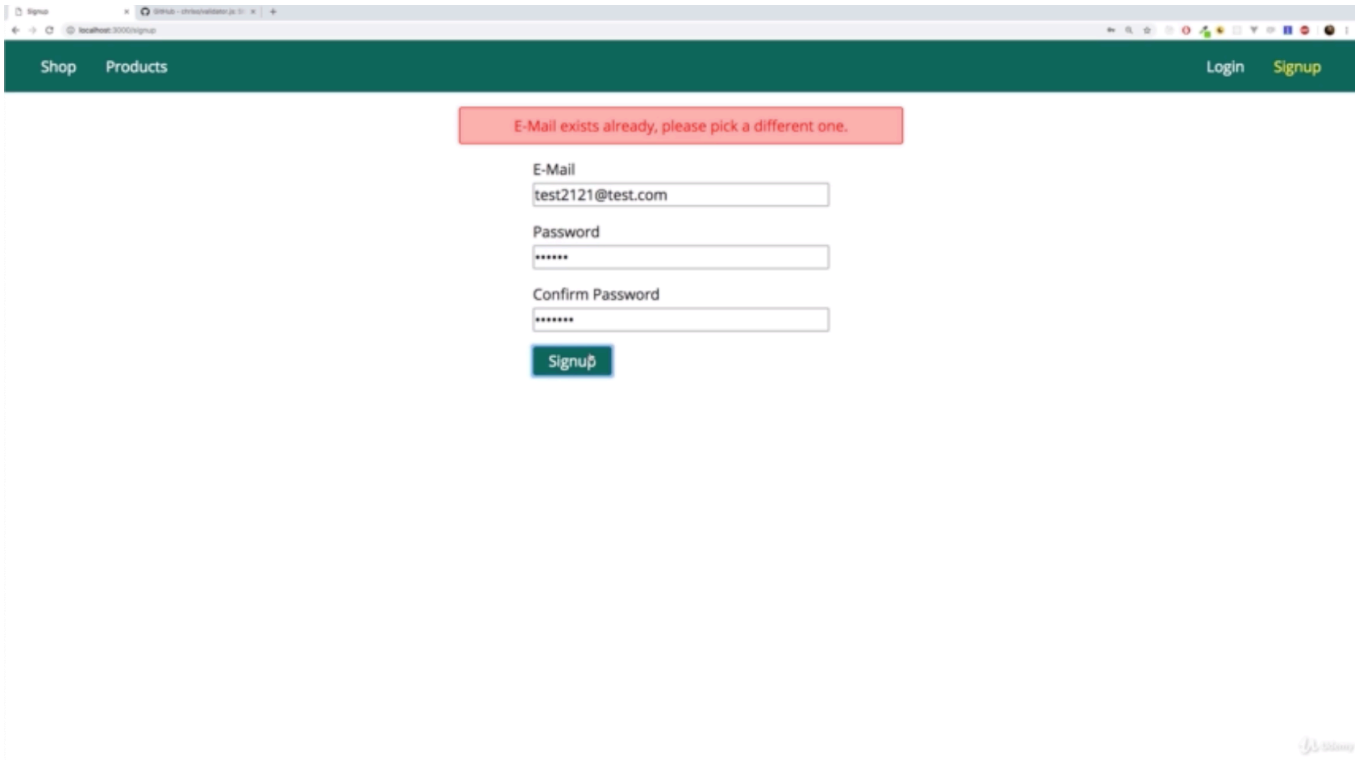
```
14 router.post('/login', authController.postLogin);
15
16 router.post(
17     '/signup',
18     [
19         check('email')
20             .isEmail()
21             .withMessage('Please enter a valid email')
22             .custom((value, {req}) => {
23                 if (value === 'test@test.com'){
24                     throw new Error('This Email Address if forbidden.')
25                 }
26                 return true
27             }),
28         body(
29             'password',
30             'Please enter a password with only numbers and text and at least 5 characters'
31         )
32             .isLength({min: 5})
33             .isAlphanumeric(),
34         body('confirmPassword').custom((value, {req}) => {
35             if (value !== req.body.password) {
36                 throw new Error('Passwords have to match!')
37             }
38             return true
39         })
40     ],
41     authController.postSignup);
42
43 router.post('/logout', authController.postLogout);
44
45 router.get('/reset', authController.getReset);
46
47 router.post('/reset', authController.postReset);
48
49 router.get('/reset/:token', authController.getNewPassword);
50
51 router.post('/new-password', authController.postNewPassword);
52
53 module.exports = router;
```

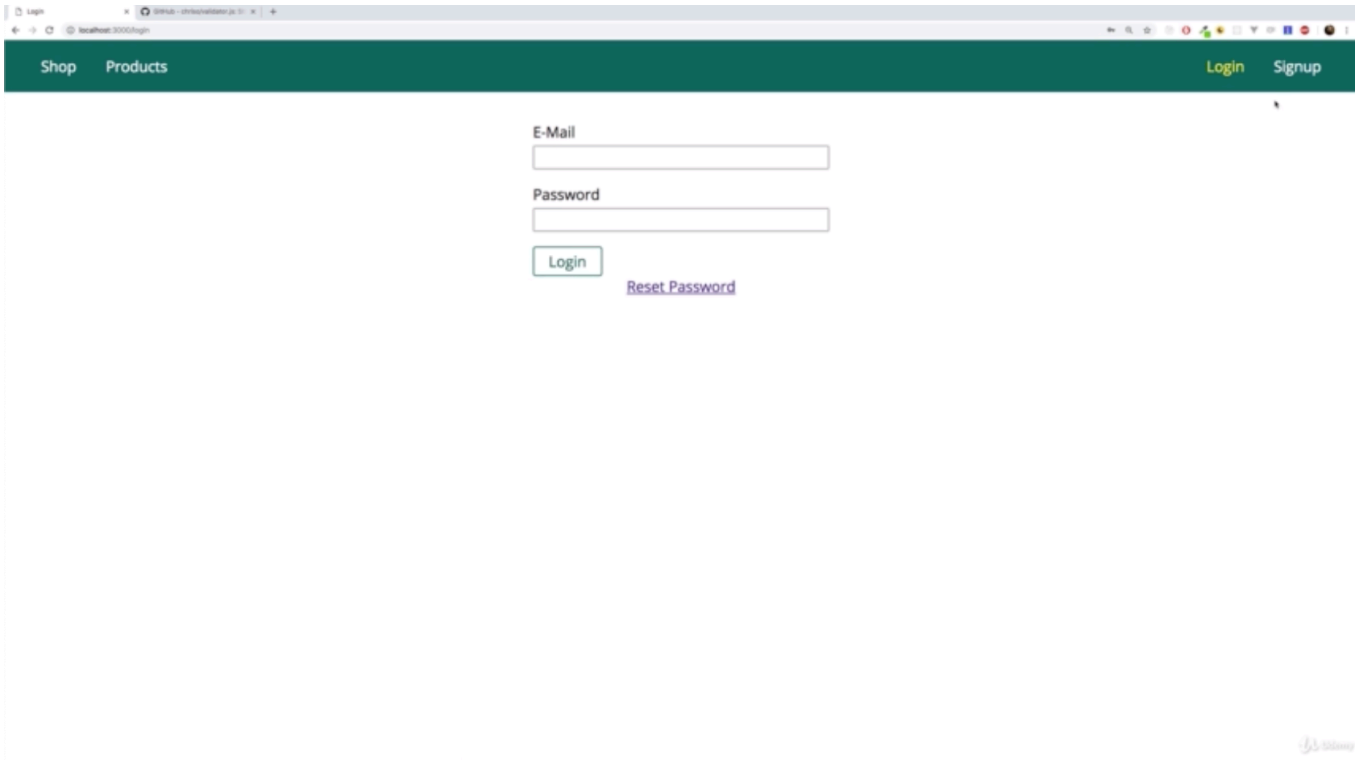
* Chapter 292: Adding Async Validation

1. update
- ./controllers/auth.js
- ./routes/auth.js


Shop
Products
Login
Signup
E-Mail exists already, please pick a different one.
E-Mail
test2@test.com
Password
Confirm Password
Signup


Shop
Products
Login
Signup
E-Mail exists already, please pick a different one.
E-Mail
Password
Confirm Password
Signup






```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');
const { validationResult } = require('express-validator/check')

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
```

```
      api_key:
      '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'
    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.');
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
          }
          req.flash('error', 'Invalid email or password.');
          res.redirect('/login');
```

```
      })
      .catch(err => {
        console.log(err);
        res.redirect('/login');
      });
    })
    .catch(err => console.log(err));
};

exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;
  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
    });
  }
    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);
      });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
```

```
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
    User.findOne({ email: req.body.email })
      .then(user => {
        if (!user) {
          req.flash('error', 'No account with that email found.');
          return res.redirect('/reset');
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/');
        transporter.sendMail({
          to: req.body.email,
          from: 'shop@node-complete.com',
          subject: 'Password reset',
          html: `
            <p>You requested a password reset</p>
            <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a new password.</p>
          `
        });
      })
      .catch(err => {
        console.log(err);
      });
  });
};

exports.getNewPassword = (req, res, next) => {
  const token = req.params.token;
  User.findOne({
    resetToken: token,
    resetTokenExpiration: { $gt: Date.now()}
  })
    .then(user => {
      let message = req.flash('error');
      if (message.length > 0) {
        message = message[0];
```

```
    } else {
      message = null;
    }
    res.render('auth/new-password', {
      path: '/new-password',
      pageTitle: 'New Password',
      errorMessage: message,
      userId: user._id.toString(),
      passwordToken: token
    });
  })
  .catch(err => {
    console.log(err);
  });
};

exports.postNewPassword = (req, res, next) => {
  const newPassword = req.body.password
  const userId = req.body.userId
  const passwordToken = req.body.passwordToken
  let resetUser

  User.findOne({
    resetToken: passwordToken,
    resetTokenExpiration: {$gt: Date.now()},
    _id: userId
  })
    .then(user => {
      resetUser = user
      return bcrypt.hash(newPassword, 12)
    })
    .then(hashedPassword => {
      resetUser.resetToken = undefined
      resetUser.resetTokenExpiration = undefined
      return resetUser.save()
    })
    .then(result => {
      res.redirect('/login')
    })
    .catch(err => {
      console.log(err)
    })

}
```

```
// ./routes/auth.js

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

const authController = require('../controllers/auth');
const User = require('../models/user')

const router = express.Router();

router.get('/login', authController.getLogin);
```

```
12
13 router.get('/signup', authController.getSignup);
14
15 router.post('/login', authController.postLogin);
16
17 router.post(
18     '/signup',
19     [
20         check('email')
21             .isEmail()
22             .withMessage('Please enter a valid email')
23             /**the express-validator package will check for a 'custom()' validator
24              * to return true or false,
25              * to return thrown error or to return a Promise.
26              * if 'then()' is a promise because every 'then()' block returns a new promise
27              * if we return a Promise,
28              * then express-validator will wait for this Promise to be fulfilled
29              * and if it fulfills with nothing no error,
30              * it treats this validation as successful
31              *
32              * if it resolves with some rejection in the end,
33              * which will happen if we make it into this if block,
34              * then express validator will detect this rejection
35              * and will store this as an error message.
36              *
37              * this is how we can add our own asynchronous validation
38              * because we have to reach out to the database
39              * which is not an instant task
40              * but express validator will wait for us here.
41             */
42             .custom((value, {req}) => {
43                 //if (value === 'test@test.com'){
44                 //    throw new Error('This Email Address if forbidden.')
45                 //}
46                 //return true
47                 return User.findOne({ email: value })
48                 .then(userDoc => {
49                   if (userDoc) {
50                     /**'Promise' is a built-in javascript object
51                      * and with reject, i throw an error inside of the promise
52                      * and i reject with this error message i used before
53                      */
54                     return Promise.reject(
55                         'E-Mail exists already, please pick a different one.'
56                     )
57                 }
58             })
59         }),
60         body(
61             'password',
62             'Please enter a password with only numbers and text and at least 5 characters'
63         )
64             .isLength({min: 5})
65             .isAlphanumeric(),
66         body('confirmPassword').custom((value, {req}) => {
67             if (value !== req.body.password) {
```

```
            throw new Error('Passwords have to match!')
          }
          return true
        })
    ],
    authController.postSignup);

router.post('/logout', authController.postLogout);

router.get('/reset', authController.getReset);

router.post('/reset', authController.postReset);

router.get('/reset/:token', authController.getNewPassword);

router.post('/new-password', authController.postNewPassword);

module.exports = router;
```

* Chapter 293: Keeping User Input

1. update
- ./controllers/auth.js
- ./views/auth/signup.ejs

- you can see that it kept the old e-mail because we return that with the response and we then output it in our view and we should do the same for password and confirm Password
————————————

- we kept all the data after clicking sign up button.

```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');
const { validationResult } = require('express-validator/check');

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
      api_key:
      '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'
    }
  })
);

exports.getLogin = (req, res, next) => {
```

```
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message,
    oldInput: {
      email: '',
      password: '',
      confirmPassword: ''
    }
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;

  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).render('auth/login', {
      path: '/login',
      pageTitle: 'Login',
      errorMessage: errors.array()[0].msg
    });
  }

  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        req.flash('error', 'Invalid email or password.');
        return res.redirect('/login');
      }
      bcrypt
        .compare(password, user.password)
        .then(doMatch => {
          if (doMatch) {
            req.session.isLoggedIn = true;
```

```
            req.session.user = user;
            return req.session.save(err => {
              console.log(err);
              res.redirect('/');
            });
          }
          req.flash('error', 'Invalid email or password.');
          res.redirect('/login');
        })
        .catch(err => {
          console.log(err);
          res.redirect('/login');
        });
    })
    .catch(err => console.log(err));
};

exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;

  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,
      oldInput: {
        email: email,
        password: password,
        confirmPassword: req.body.confirmPassword
      }
    });
  }

  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);
    });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
    User.findOne({ email: req.body.email })
      .then(user => {
        if (!user) {
          req.flash('error', 'No account with that email found.');
          return res.redirect('/reset');
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/');
        transporter.sendMail({
          to: req.body.email,
          from: 'shop@node-complete.com',
          subject: 'Password reset',
          html: `
            <p>You requested a password reset</p>
            <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
new password.</p>
          `
        });
      })
      .catch(err => {
```

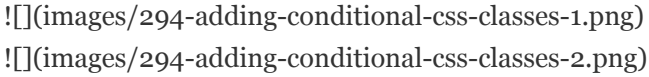
```
      console.log(err);
    });
  });
};

exports.getNewPassword = (req, res, next) => {
  const token = req.params.token;
  User.findOne({ resetToken: token, resetTokenExpiration: { $gt: Date.now() } })
    .then(user => {
      let message = req.flash('error');
      if (message.length > 0) {
        message = message[0];
      } else {
        message = null;
      }
      res.render('auth/new-password', {
        path: '/new-password',
        pageTitle: 'New Password',
        errorMessage: message,
        userId: user._id.toString(),
        passwordToken: token
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.postNewPassword = (req, res, next) => {
  const newPassword = req.body.password;
  const userId = req.body.userId;
  const passwordToken = req.body.passwordToken;
  let resetUser;

  User.findOne({
    resetToken: passwordToken,
    resetTokenExpiration: { $gt: Date.now() },
    _id: userId
  })
    .then(user => {
      resetUser = user;
      return bcrypt.hash(newPassword, 12);
    })
    .then(hashedPassword => {
      resetUser.password = hashedPassword;
      resetUser.resetToken = undefined;
      resetUser.resetTokenExpiration = undefined;
      return resetUser.save();
    })
    .then(result => {
      res.redirect('/login');
    })
    .catch(err => {
      console.log(err);
    });
};
```

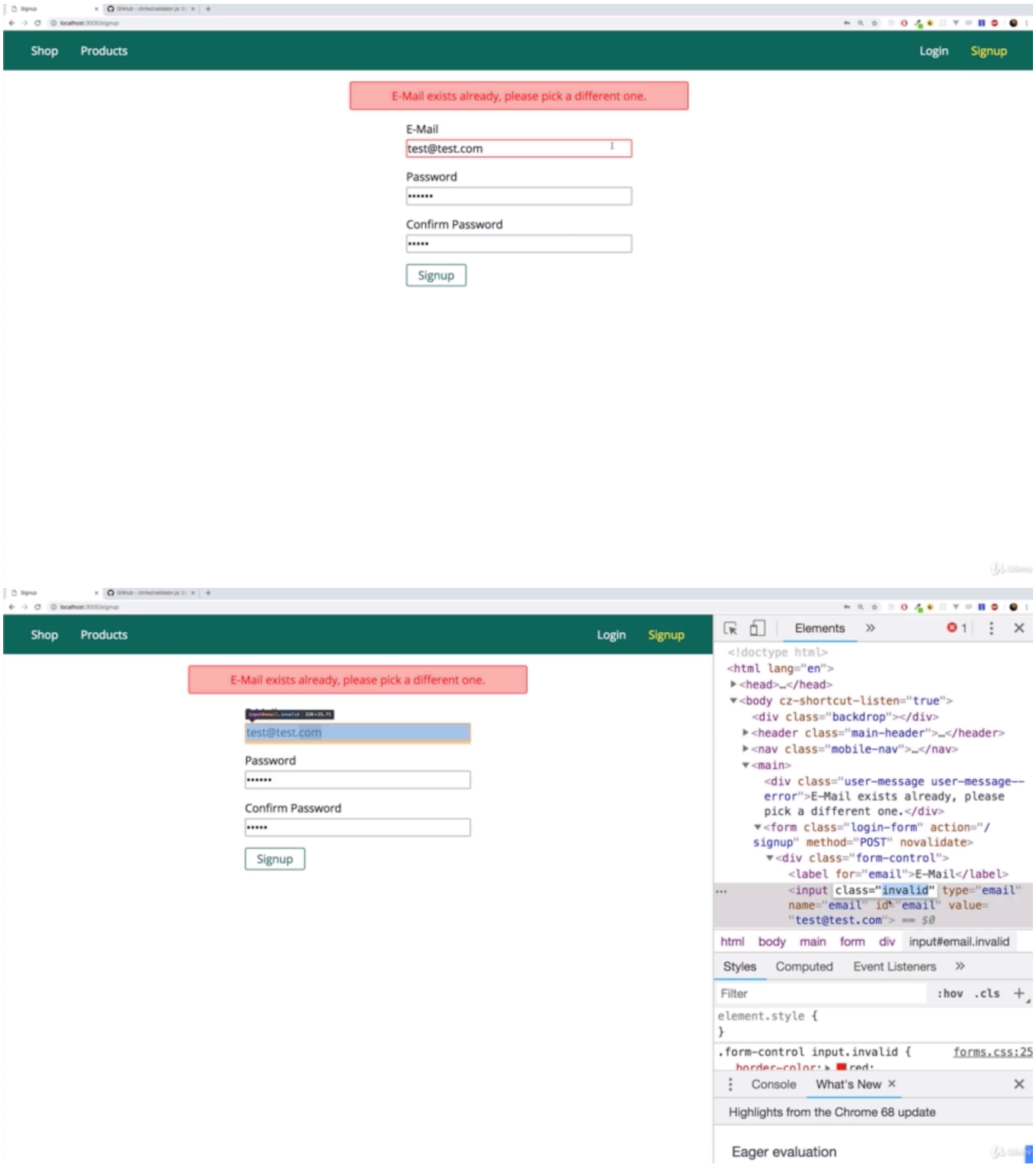
245

```
<!--./views/auth/signup.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="stylesheet" href="/css/auth.css">
</head>

<body>
   <%- include('../includes/navigation.ejs') %>

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="login-form" action="/signup" method="POST" novalidate>
            <div class="form-control">
                <label for="email">E-Mail</label>
                <input type="email" name="email" id="email" value="<%= oldInput.email %>">
            </div>
            <div class="form-control">
                <label for="password">Password</label>
                <input type="password" name="password" id="password" value="<%=
oldInput.password %>">
            </div>
            <div class="form-control">
                <label for="confirmPassword">Confirm Password</label>
                <input type="password" name="confirmPassword" id="confirmPassword" value="
<%= oldInput.confirmPassword %>">
            </div>
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Signup</button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

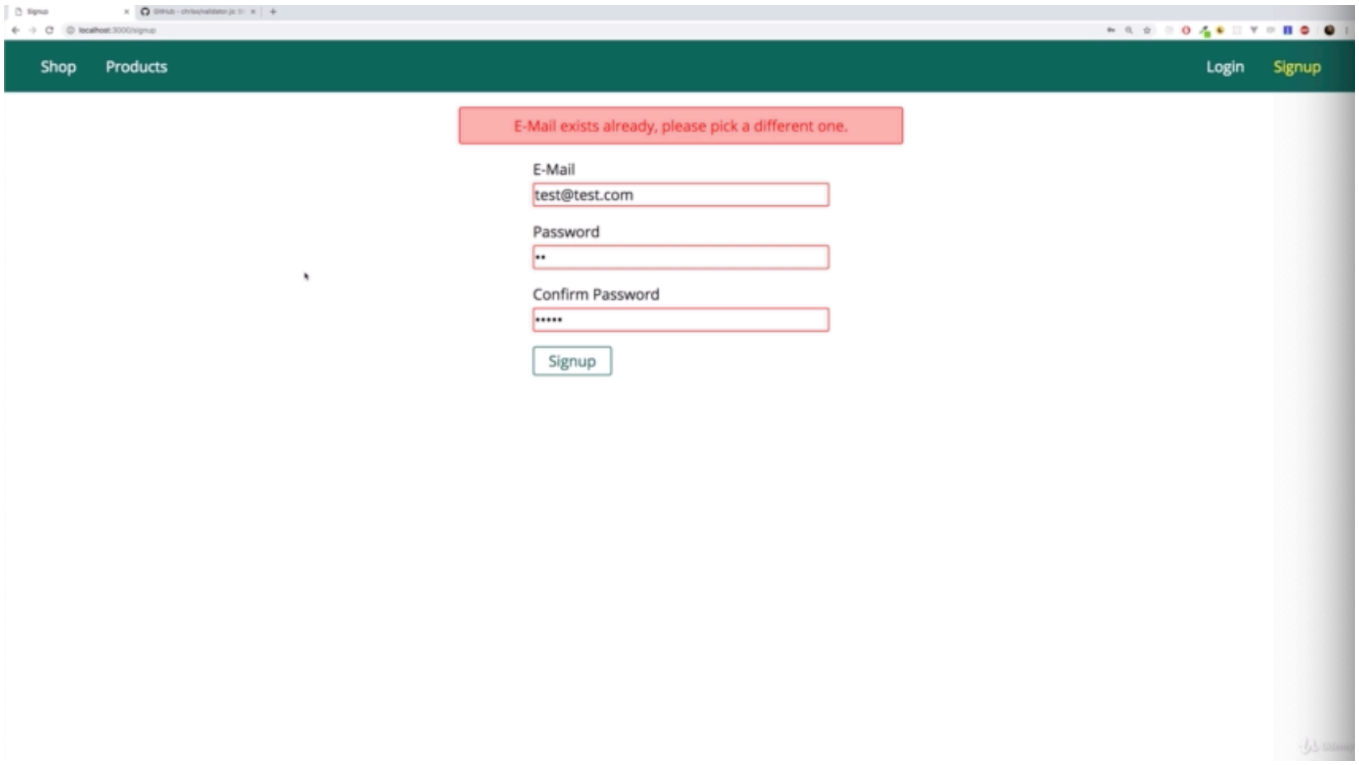
* Chapter 294: Adding Conditional CSS Classes

1. update
- ./controllers/auth.js
- ./views/singup.ejs
- ./public/css/forms.css





- that red box is because this email is invalid.




```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');
const { validationResult } = require('express-validator/check');

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
      api_key:
      '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'    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message,
    /**we don't get an error regarding this not being found
     * when it's first rendered.
     */
    oldInput: {
      email: '',
```

```
      password: ''
    },
    validationErrors: []
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message,
    oldInput: {
      email: '',
      password: '',
      confirmPassword: ''
    },
    /**because we got no errors. */
    validationErrors: []
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;

  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).render('auth/login', {
      path: '/login',
      pageTitle: 'Login',
      errorMessage: errors.array()[0].msg,
      oldInput: {
        email: email,
        password: password
      },
      validationErrors: errors.array()
    });
  }

  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        return res.status(422).render('auth/login', {
          path: '/login',
          pageTitle: 'Login',
          errorMessage: 'Invalid email or password.',
          oldInput: {
            email: email,
            password: password
          },
```

```
        validationErrors: []
      });
    }
    bcrypt
      .compare(password, user.password)
      .then(doMatch => {
        if (doMatch) {
          req.session.isLoggedIn = true;
          req.session.user = user;
          return req.session.save(err => {
            console.log(err);
            res.redirect('/');
          });
        }
        return res.status(422).render('auth/login', {
          path: '/login',
          pageTitle: 'Login',
          errorMessage: 'Invalid email or password.',
          oldInput: {
            email: email,
            password: password
          },
          validationErrors: []
        });
      })
      .catch(err => {
        console.log(err);
        res.redirect('/login');
      });
  })
  .catch(err => console.log(err));
};

exports.postSignup = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;

  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,
      oldInput: {
        email: email,
        password: password,
        confirmPassword: req.body.confirmPassword
      },
      /**so the full array is now returned as well
       * so with 'errorMessage', i'm returning not only just the first message which i picked to show
       * but also the full array of errors with 'validationErrors'
       */
      validationErrors: errors.array()
    });
```

```
  }

  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);
    });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
    User.findOne({ email: req.body.email })
      .then(user => {
```

```
      if (!user) {
        req.flash('error', 'No account with that email found.');
        return res.redirect('/reset');
      }
      user.resetToken = token;
      user.resetTokenExpiration = Date.now() + 3600000;
      return user.save();
    })
    .then(result => {
      res.redirect('/');
      transporter.sendMail({
        to: req.body.email,
        from: 'shop@node-complete.com',
        subject: 'Password reset',
        html: `
          <p>You requested a password reset</p>
          <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a new password.</p>
        `
      });
    })
    .catch(err => {
      console.log(err);
    });
  });
};

exports.getNewPassword = (req, res, next) => {
  const token = req.params.token;
  User.findOne({ resetToken: token, resetTokenExpiration: { $gt: Date.now() } })
    .then(user => {
      let message = req.flash('error');
      if (message.length > 0) {
        message = message[0];
      } else {
        message = null;
      }
      res.render('auth/new-password', {
        path: '/new-password',
        pageTitle: 'New Password',
        errorMessage: message,
        userId: user._id.toString(),
        passwordToken: token
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.postNewPassword = (req, res, next) => {
  const newPassword = req.body.password;
  const userId = req.body.userId;
  const passwordToken = req.body.passwordToken;
  let resetUser;

```

```
  User.findOne({
    resetToken: passwordToken,
    resetTokenExpiration: { $gt: Date.now() },
    _id: userId
  })
    .then(user => {
      resetUser = user;
      return bcrypt.hash(newPassword, 12);
    })
    .then(hashedPassword => {
      resetUser.password = hashedPassword;
      resetUser.resetToken = undefined;
      resetUser.resetTokenExpiration = undefined;
      return resetUser.save();
    })
    .then(result => {
      res.redirect('/login');
    })
    .catch(err => {
      console.log(err);
    });
};

```

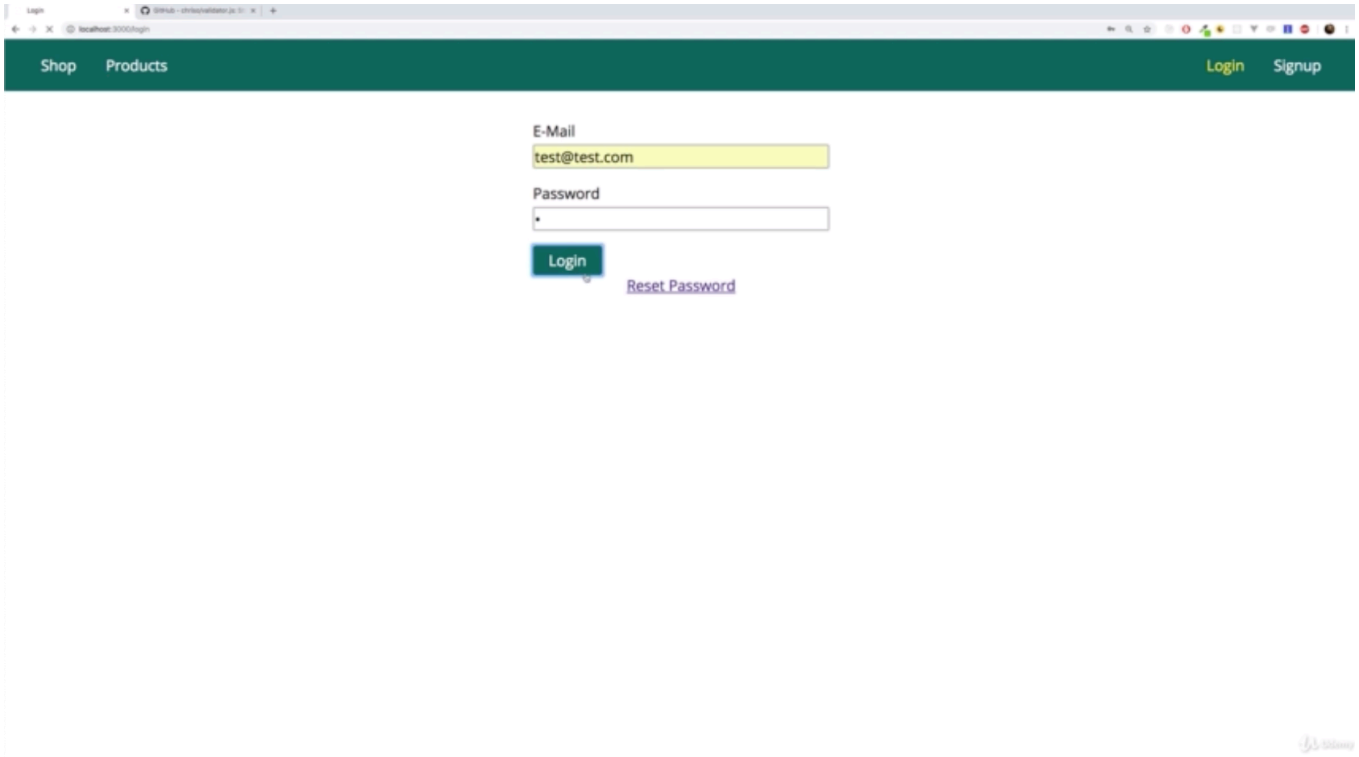
```
<!--./views/auth/signup.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="stylesheet" href="/css/auth.css">
</head>

<body>
   <%- include('../includes/navigation.ejs') %>

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="login-form" action="/signup" method="POST" novalidate>
            <div class="form-control">
                <label for="email">E-Mail</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'email') ? 'invalid' :
'' %>"
                    type="email"
                    name="email"
                    id="email"
                    value="<%= oldInput.email %>">
            </div>
            <div class="form-control">
                <label for="password">Password</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'password') ?
'invalid' : '' %>"
                    type="password"
                    name="password"
                    id="password"
```

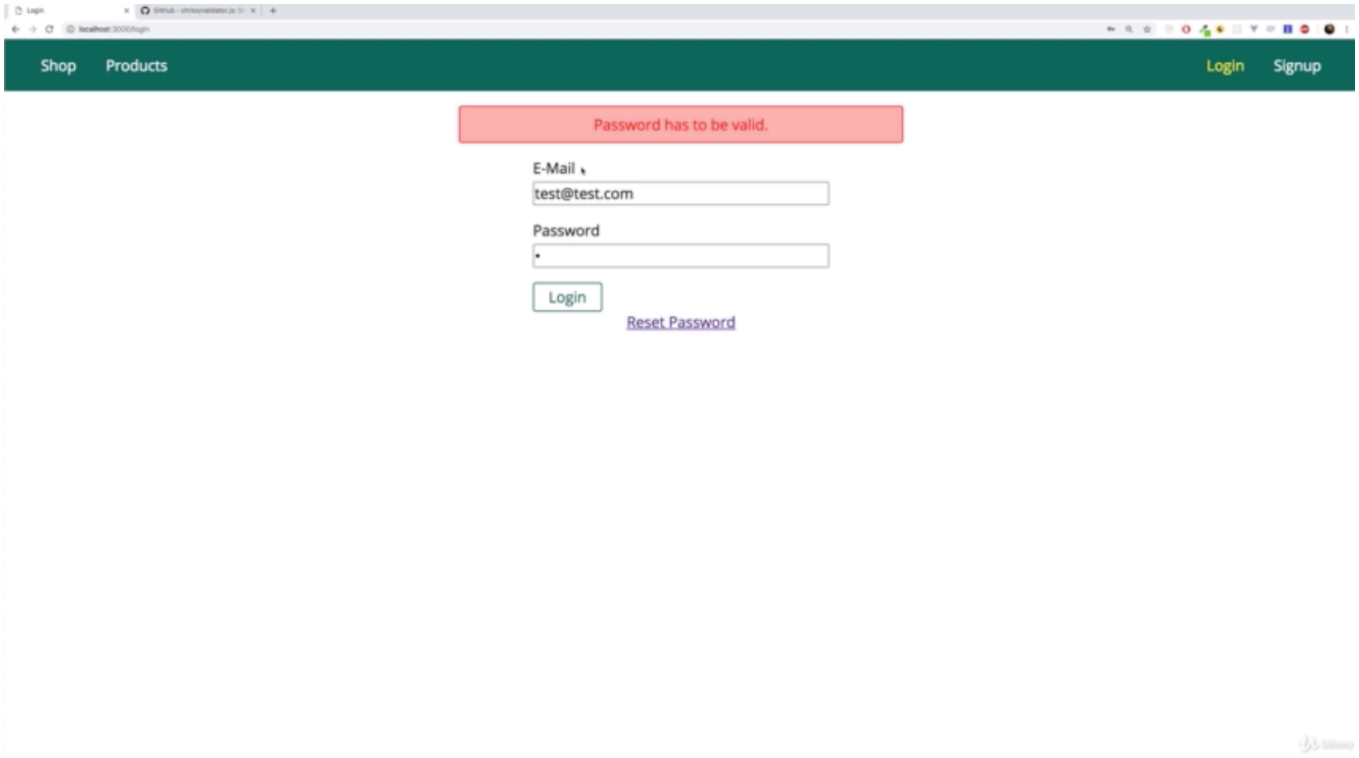
```
                value="<%= oldInput.password %>">
            </div>
            <div class="form-control">
                <label for="confirmPassword">Confirm Password</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'confirmPassword') ? 'invalid' : '' %>"
                    type="password"
                    name="confirmPassword"
                    id="confirmPassword"
                    value="<%= oldInput.confirmPassword %>">
            </div>
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Signup</button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

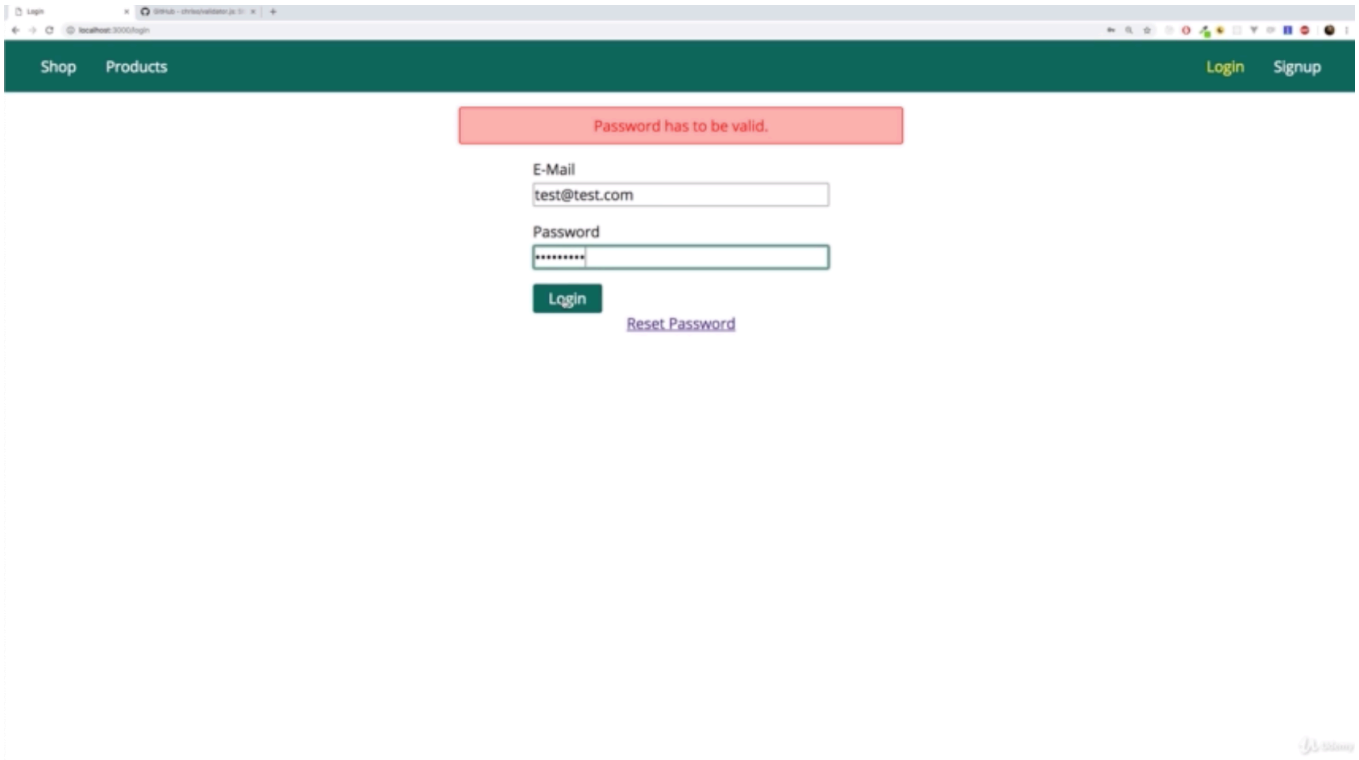
```
/*./public/css/forms.css*/

.form-control {
  margin: 1rem 0;
}

.form-control label,
.form-control input,
.form-control textarea {
  display: block;
  width: 100%;
  margin-bottom: 0.25rem;
}

.form-control input,
.form-control textarea {
  border: 1px solid #a1a1a1;
  font: inherit;
  border-radius: 2px;
}

.form-control input:focus,
.form-control textarea:focus {
  outline-color: #00695c;
}

.form-control input.invalid {
  border-color: red;
}
```

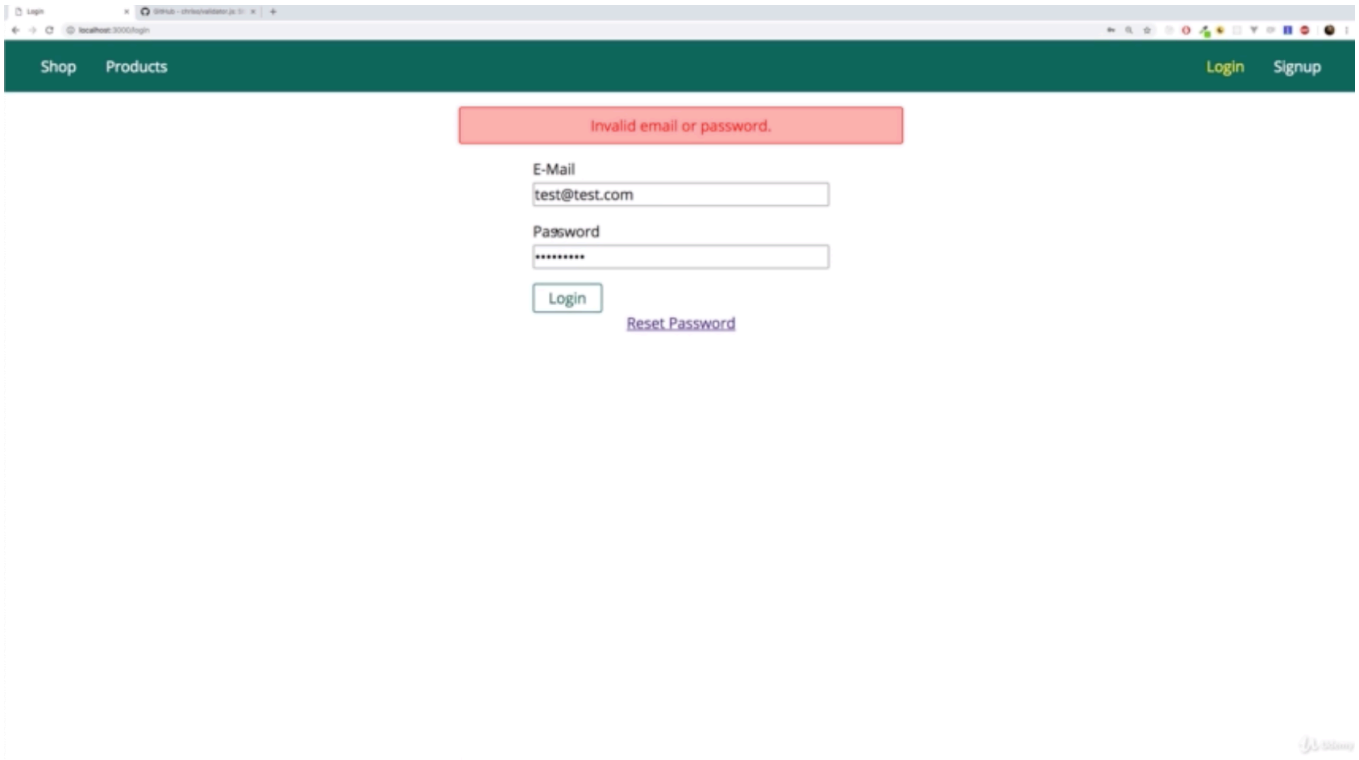
* Chapter 295: Adding Validation To Login

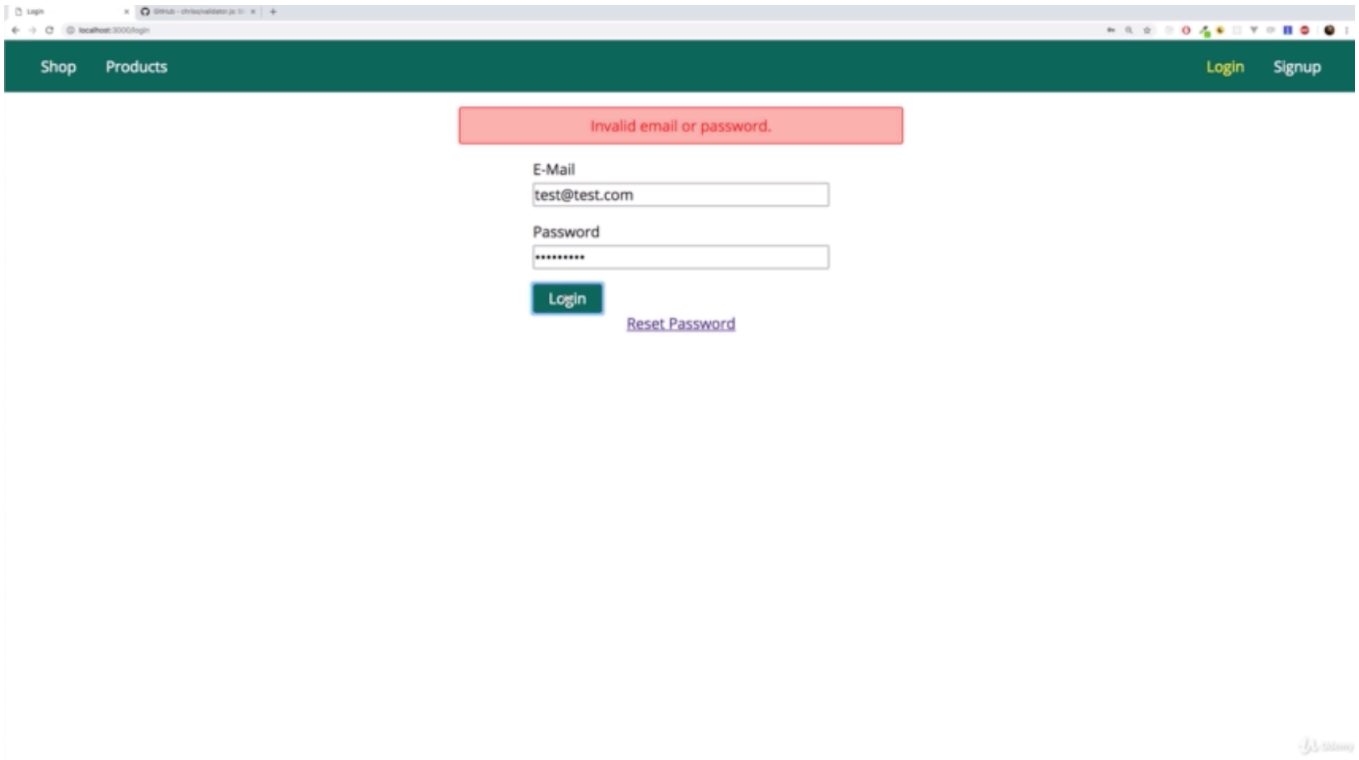
1. update
- ./controllers/auth.js
- ./views/auth/login.ejs

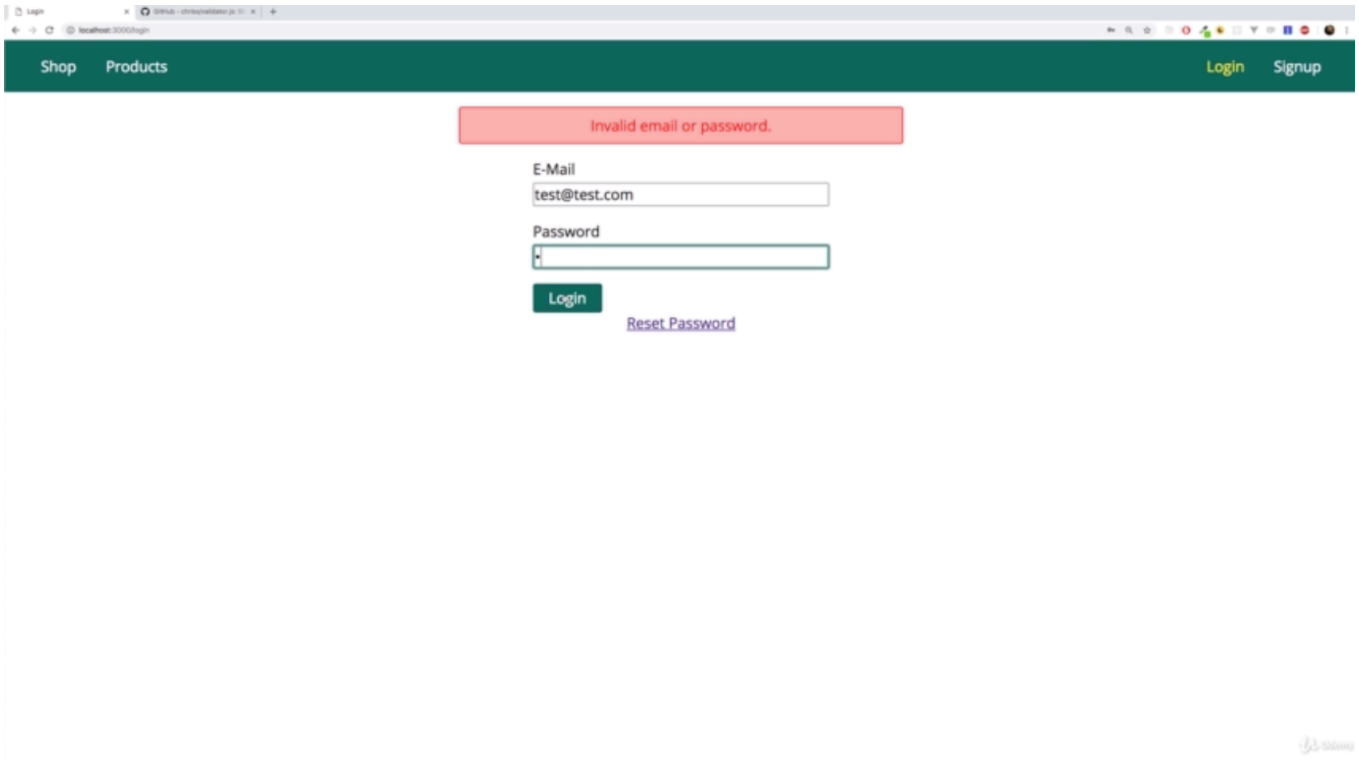

Shop
Products
Login
Signup
E-Mail
test@test.com
Password
Login
Reset Password

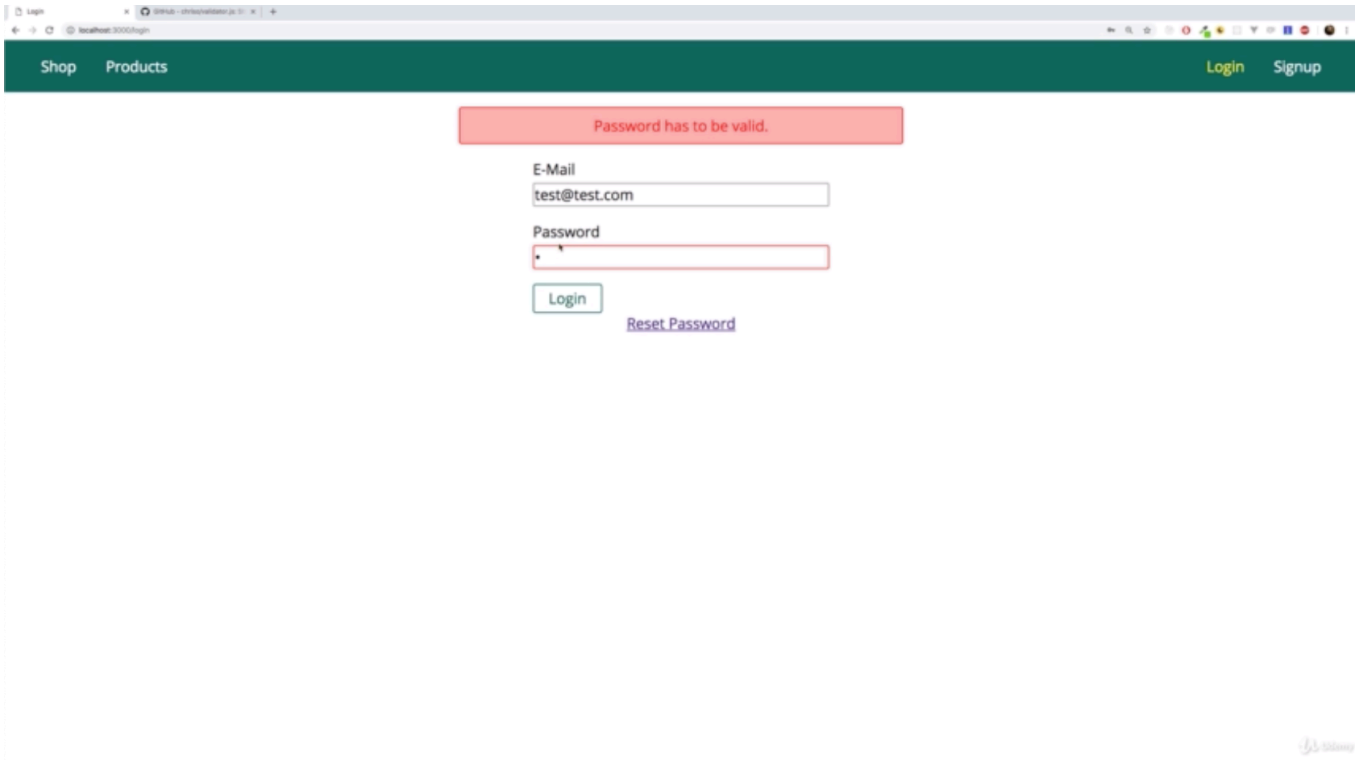

Shop
Products
Login
Signup
Password has to be valid.
E-Mail
test@test.com
Password
Login
Reset Password


Shop
Products
Login
Signup
Password has to be valid.
E-Mail
test@test.com
Password
Login
Reset Password


Shop
Products
Login
Signup
Invalid email or password.
E-Mail
test@test.com
Password
Login
Reset Password


Shop
Products
Login
Signup
Invalid email or password.
E-Mail
test@test.com
Password
Login
Reset Password


Shop
Products
Login
Signup
Invalid email or password.
E-Mail
test@test.com
Password
Login
Reset Password




```
//./controllers/auth.js

const crypto = require('crypto');

const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer');
const sendgridTransport = require('nodemailer-sendgrid-transport');
const { validationResult } = require('express-validator/check');

const User = require('../models/user');

const transporter = nodemailer.createTransport(
  sendgridTransport({
    auth: {
      api_key:
      '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'    }
  })
);

exports.getLogin = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: message,
    /**we don't get an error regarding this not being found
     * when it's first rendered.
     */
    oldInput: {
      email: '',
```

```
      password: ''
    },
    validationErrors: []
  });
};

exports.getSignup = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup',
    errorMessage: message,
    oldInput: {
      email: '',
      password: '',
      confirmPassword: ''
    },
    validationErrors: []
  });
};

exports.postLogin = (req, res, next) => {
  const email = req.body.email;
  const password = req.body.password;

  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).render('auth/login', {
      path: '/login',
      pageTitle: 'Login',
      errorMessage: errors.array()[0].msg,
      oldInput: {
        email: email,
        password: password
      },
      validationErrors: errors.array()
    });
  }

  User.findOne({ email: email })
    .then(user => {
      if (!user) {
        return res.status(422).render('auth/login', {
          path: '/login',
          pageTitle: 'Login',
          errorMessage: 'Invalid email or password.',
          oldInput: {
            email: email,
            password: password
          },
          /**we have to add some object with param
```

```
92          * and that in this case it could be param email and param password
93          * if you wanna make sure you don't show what led to the error like below
94          *
95          *    validationErrors: [{param: 'email', param: 'password'}]
96          *
97          * but you could leave that out and only give out the message
98          */
99         validationErrors: []
100       });
101     }
102     bcrypt
103       .compare(password, user.password)
104       .then(doMatch => {
105         if (doMatch) {
106           req.session.isLoggedIn = true;
107           req.session.user = user;
108           return req.session.save(err => {
109             console.log(err);
110             res.redirect('/');
111           });
112         }
113         return res.status(422).render('auth/login', {
114           path: '/login',
115           pageTitle: 'Login',
116           errorMessage: 'Invalid email or password.',
117           oldInput: {
118             email: email,
119             password: password
120           },
121           validationErrors: []
122         });
123       })
124       .catch(err => {
125         console.log(err);
126         res.redirect('/login');
127       });
128   })
129   .catch(err => console.log(err));
130 };
131
132 exports.postSignup = (req, res, next) => {
133   const email = req.body.email;
134   const password = req.body.password;
135
136   const errors = validationResult(req);
137   if (!errors.isEmpty()) {
138     console.log(errors.array());
139     return res.status(422).render('auth/signup', {
140       path: '/signup',
141       pageTitle: 'Signup',
142       errorMessage: errors.array()[0].msg,
143       oldInput: {
144         email: email,
145         password: password,
146         confirmPassword: req.body.confirmPassword
147       },
```

```
      validationErrors: errors.array()
    });
  }

  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);
    });
};

exports.postLogout = (req, res, next) => {
  req.session.destroy(err => {
    console.log(err);
    res.redirect('/');
  });
};

exports.getReset = (req, res, next) => {
  let message = req.flash('error');
  if (message.length > 0) {
    message = message[0];
  } else {
    message = null;
  }
  res.render('auth/reset', {
    path: '/reset',
    pageTitle: 'Reset Password',
    errorMessage: message
  });
};

exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
```

```
    User.findOne({ email: req.body.email })
      .then(user => {
        if (!user) {
          req.flash('error', 'No account with that email found.');
          return res.redirect('/reset');
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/');
        transporter.sendMail({
          to: req.body.email,
          from: 'shop@node-complete.com',
          subject: 'Password reset',
          html: `
            <p>You requested a password reset</p>
            <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a
new password.</p>
          `
        });
      })
      .catch(err => {
        console.log(err);
      });
  });
};

exports.getNewPassword = (req, res, next) => {
  const token = req.params.token;
  User.findOne({ resetToken: token, resetTokenExpiration: { $gt: Date.now() } })
    .then(user => {
      let message = req.flash('error');
      if (message.length > 0) {
        message = message[0];
      } else {
        message = null;
      }
      res.render('auth/new-password', {
        path: '/new-password',
        pageTitle: 'New Password',
        errorMessage: message,
        userId: user._id.toString(),
        passwordToken: token
      });
    })
    .catch(err => {
      console.log(err);
    });
};

exports.postNewPassword = (req, res, next) => {
  const newPassword = req.body.password;
  const userId = req.body.userId;
  const passwordToken = req.body.passwordToken;
```

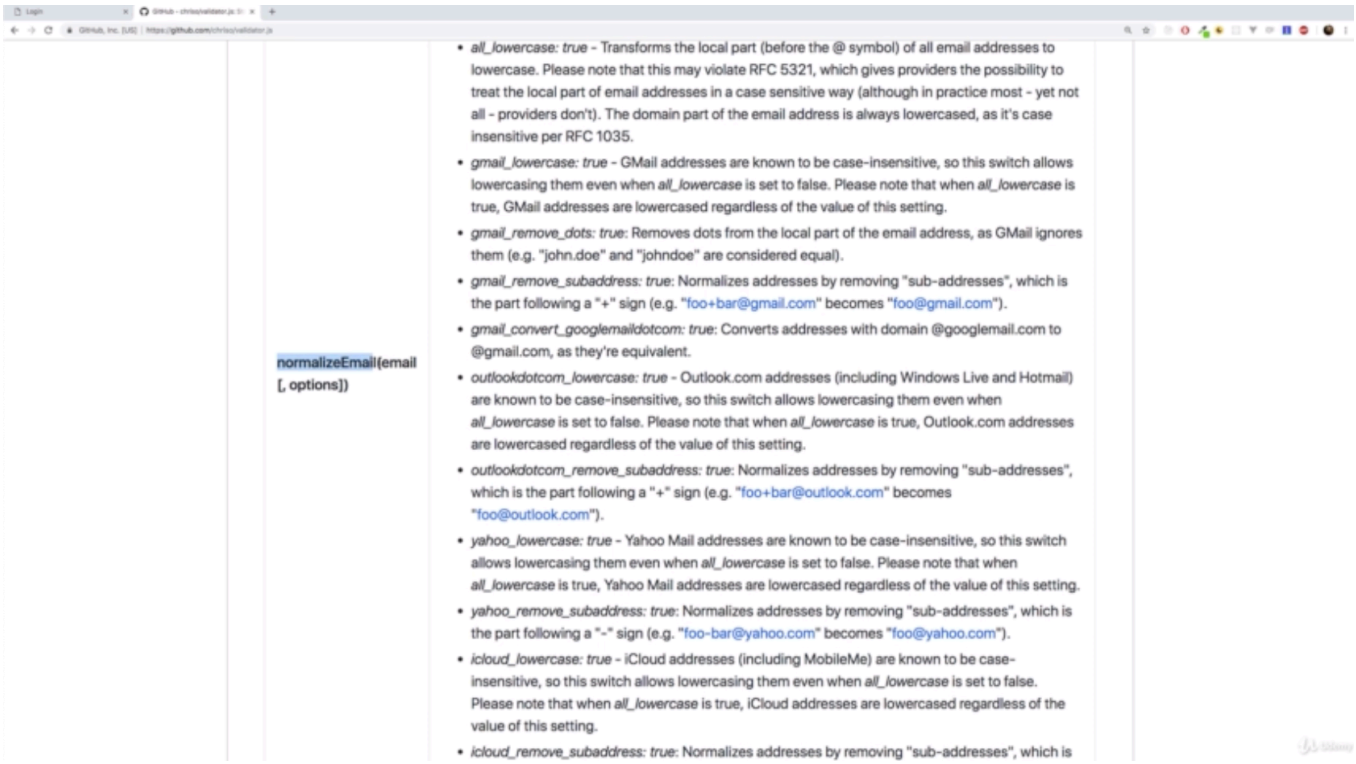
```
  let resetUser;

  User.findOne({
    resetToken: passwordToken,
    resetTokenExpiration: { $gt: Date.now() },
    _id: userId
  })
    .then(user => {
      resetUser = user;
      return bcrypt.hash(newPassword, 12);
    })
    .then(hashedPassword => {
      resetUser.password = hashedPassword;
      resetUser.resetToken = undefined;
      resetUser.resetTokenExpiration = undefined;
      return resetUser.save();
    })
    .then(result => {
      res.redirect('/login');
    })
    .catch(err => {
      console.log(err);
    });
};

```

```
<!--./views/auth/login.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="stylesheet" href="/css/auth.css">
</head>

<body>
   <%- include('../includes/navigation.ejs') %>

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="login-form" action="/login" method="POST">
            <div class="form-control">
                <label for="email">E-Mail</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'email') ? 'invalid' :
'' %>"
                    type="email"
                    name="email"
                    id="email"
                    value="<%= oldInput.email %>">
            </div>
            <div class="form-control">
                <label for="password">Password</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'password') ?
'invalid' : '' %>"
                    type="password"
```

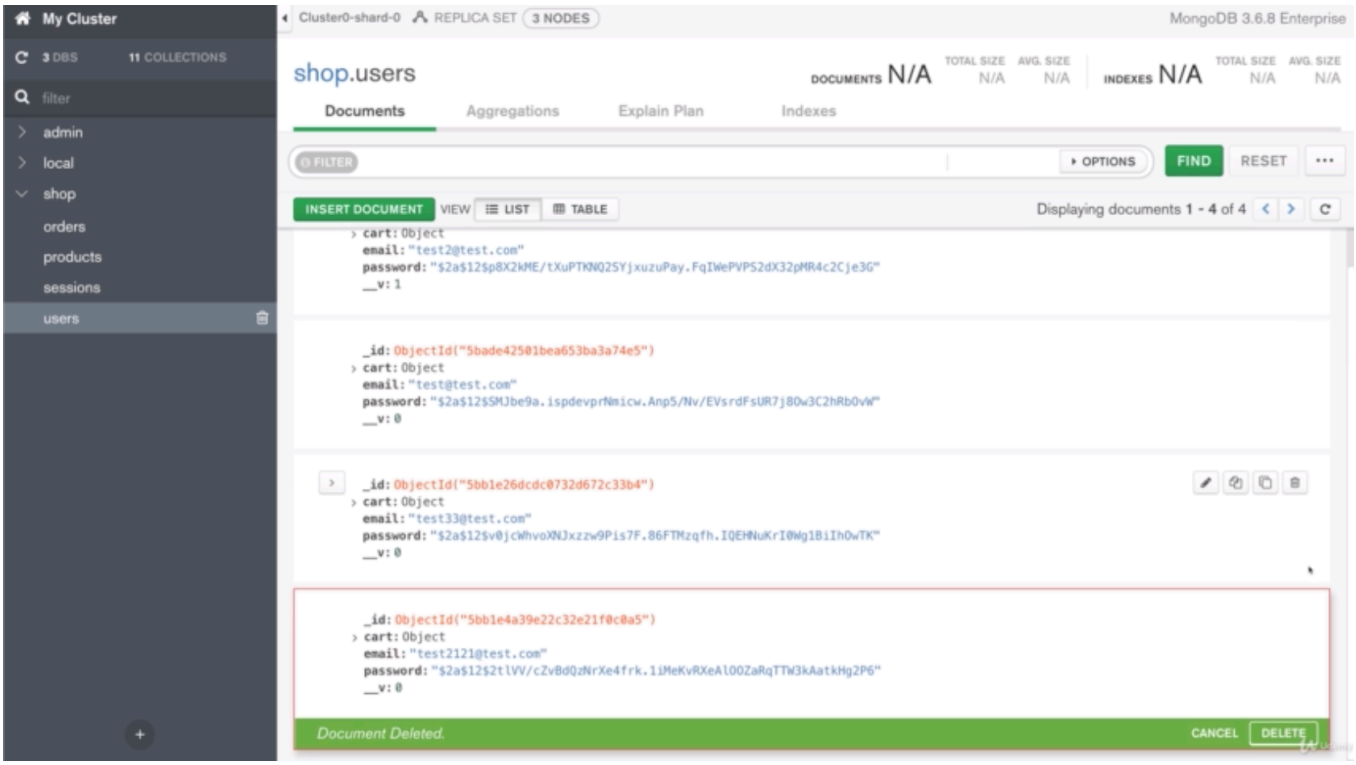
```
                    name="password"
                    id="password"
                    value="<%= oldInput.password %>">
            </div>
            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit">Login</button>
        </form>
        <div class="centered">
            <a href="/reset">Reset Password</a>
        </div>
    </main>
<%- include('../includes/end.ejs') %>
```

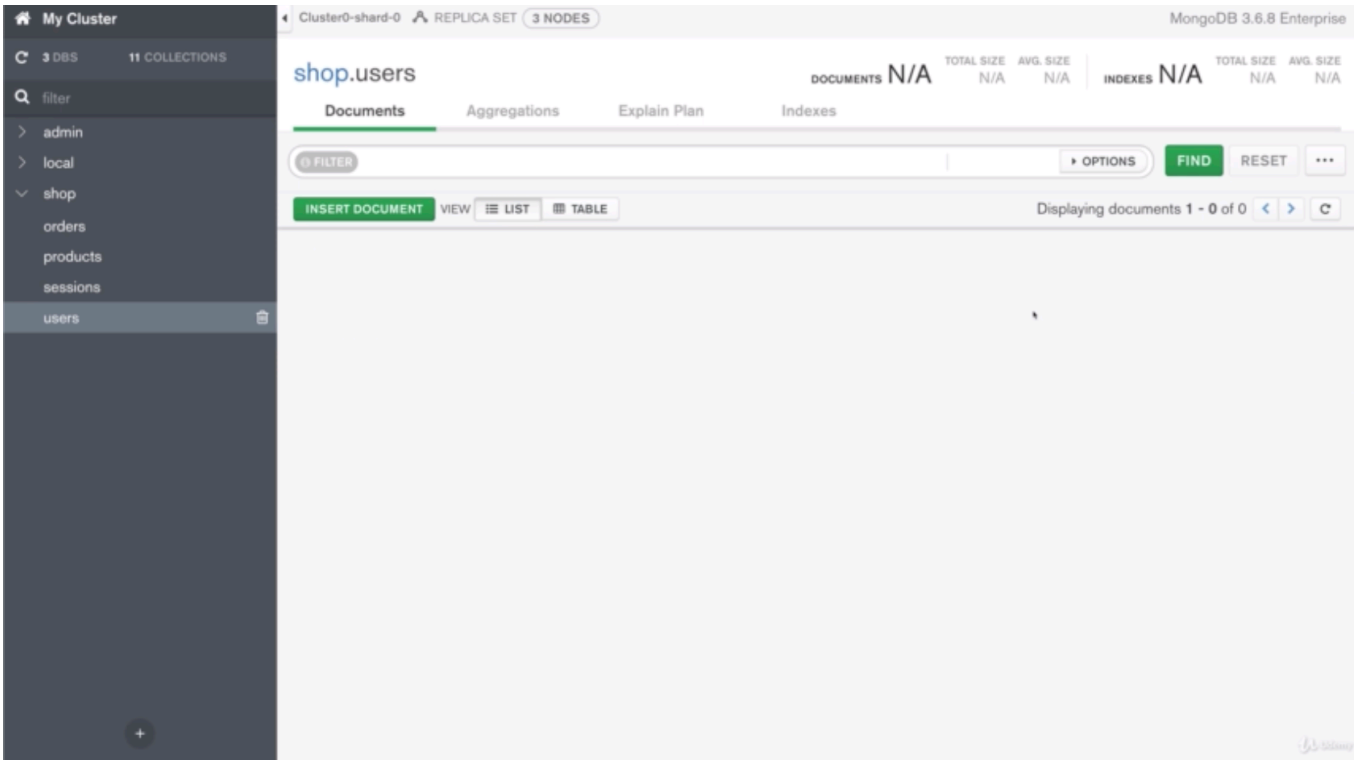
* Chapter 296: Sanitizing Data

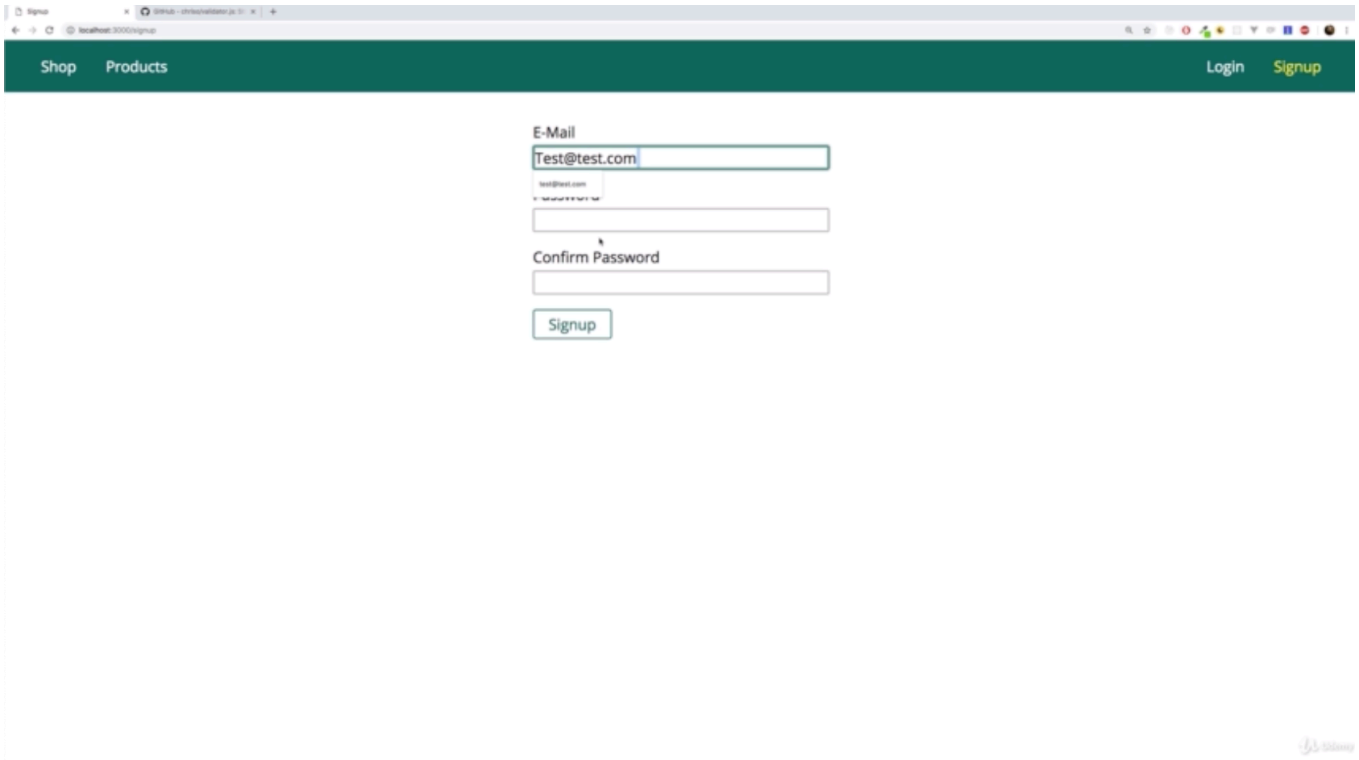
1. update
- ./routes/auth.js

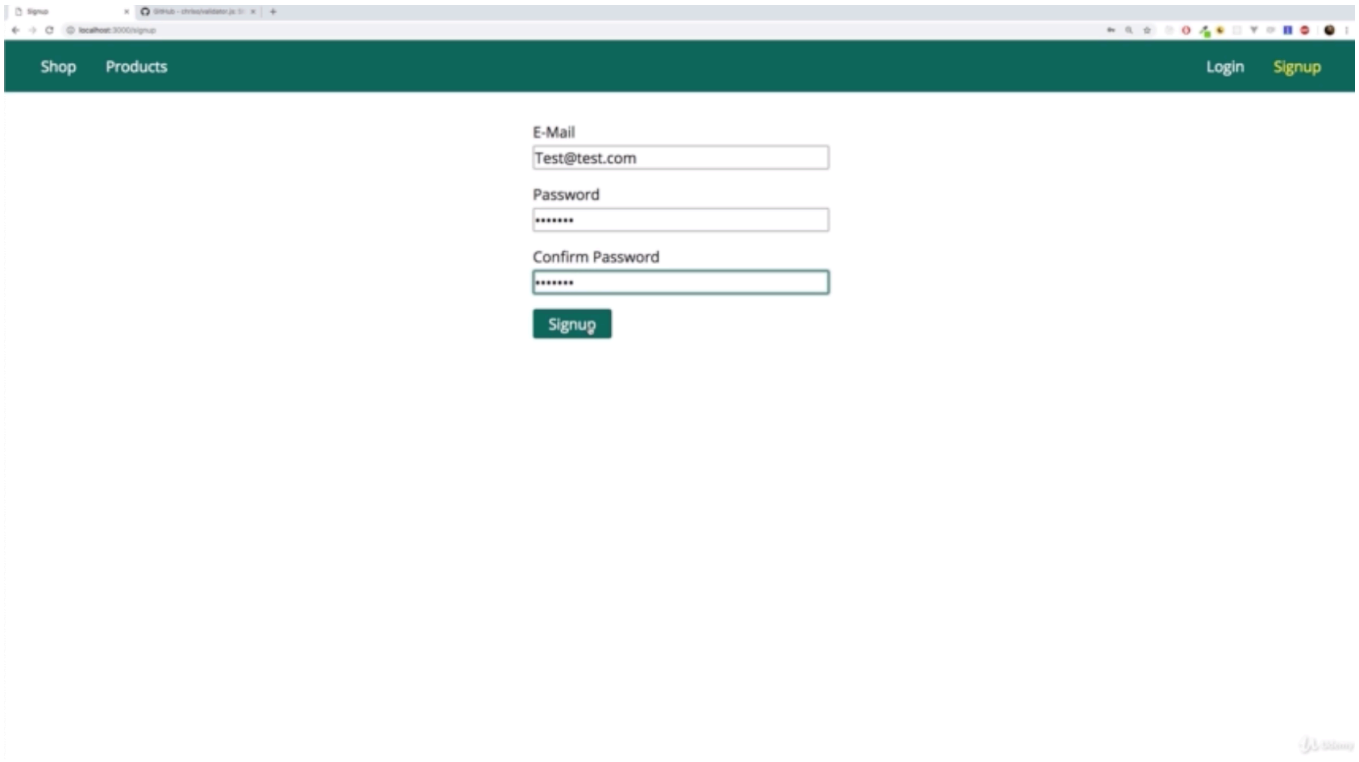


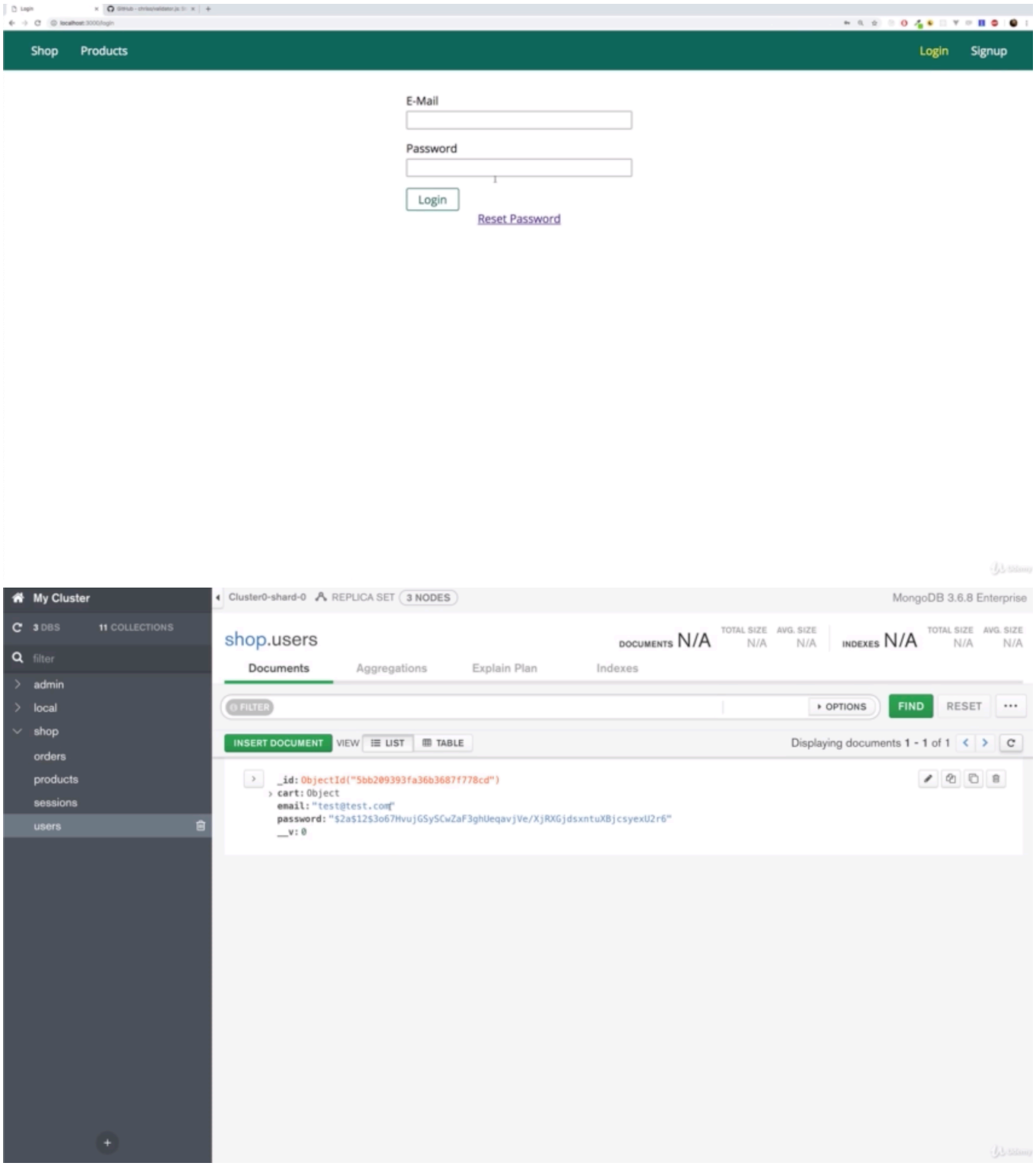
- you can ensure that there is no excess whitespace in a string passed by the user on the left or on the right. or you can normalize an email which means it's converted to lowercase and things like that.
- there are a couple of things you can do to make sure the data you get is not just valid but also is stored in a uniform way.


My Cluster
3 DBS
11 COLLECTIONS
filter
admin
local
shop
orders
products
sessions
users
Cluster0-shard-0
REPLICA SET
3 NODES
MongoDB 3.6.8 Enterprise
shop.users
DOCUMENTS N/A
TOTAL SIZE N/A
AVG. SIZE N/A
INDEXES N/A
TOTAL SIZE N/A
AVG. SIZE N/A
Documents
Aggregations
Explain Plan
Indexes
FILTER
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 4 of 4
cart: Object
email: "test2@test.com"
password: "$2a$12$p8X2kME/tXuPTKNQ2SYjxuzuPay.FqIWePVPS2dX32pMR4c2Cje3G"
__v: 1
_id: ObjectId("5bade42501bea653ba3a74e5")
cart: Object
email: "test@test.com"
password: "$2a$12$SMJbe9a.ispdevprNmicw.Anp5/Nv/EVsrdFsUR7j8Ow3C2hRbOvW"
__v: 0
_id: ObjectId("5bb1e26dcdc0732d672c33b4")
cart: Object
email: "test33@test.com"
password: "$2a$12$v0jcWhvoXNJxzzw9Pis7F.86FTMzqfh.IQEHNuKrI0Wg1BiIhOwTK"
__v: 0
_id: ObjectId("5bb1e4a39e22c32e21f0c0a5")
cart: Object
email: "test2121@test.com"
password: "$2a$12$2tlVV/cZvBdQzNrXe4frk.1iMeKvRXeAlOOZaRqTTW3kAatkHg2P6"
__v: 0
Document Deleted.
CANCEL
DELETE


My Cluster
3 DBS
11 COLLECTIONS
filter
admin
local
shop
orders
products
sessions
users
Cluster0-shard-0
REPLICA SET
3 NODES
MongoDB 3.6.8 Enterprise
shop.users
DOCUMENTS N/A
TOTAL SIZE N/A
AVG. SIZE N/A
INDEXES N/A
TOTAL SIZE N/A
AVG. SIZE N/A
Documents
Aggregations
Explain Plan
Indexes
FILTER
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 0 of 0


Shop
Products
Login
Signup
E-Mail
Test@test.com
Confirm Password
Signup


Shop
Products
Login
Signup
E-Mail
Test@test.com
Password
Confirm Password
Signup



- even though there's whitespace and uppercase, this can fix this automatically by '.trim()' and 'normalizeEmail()'

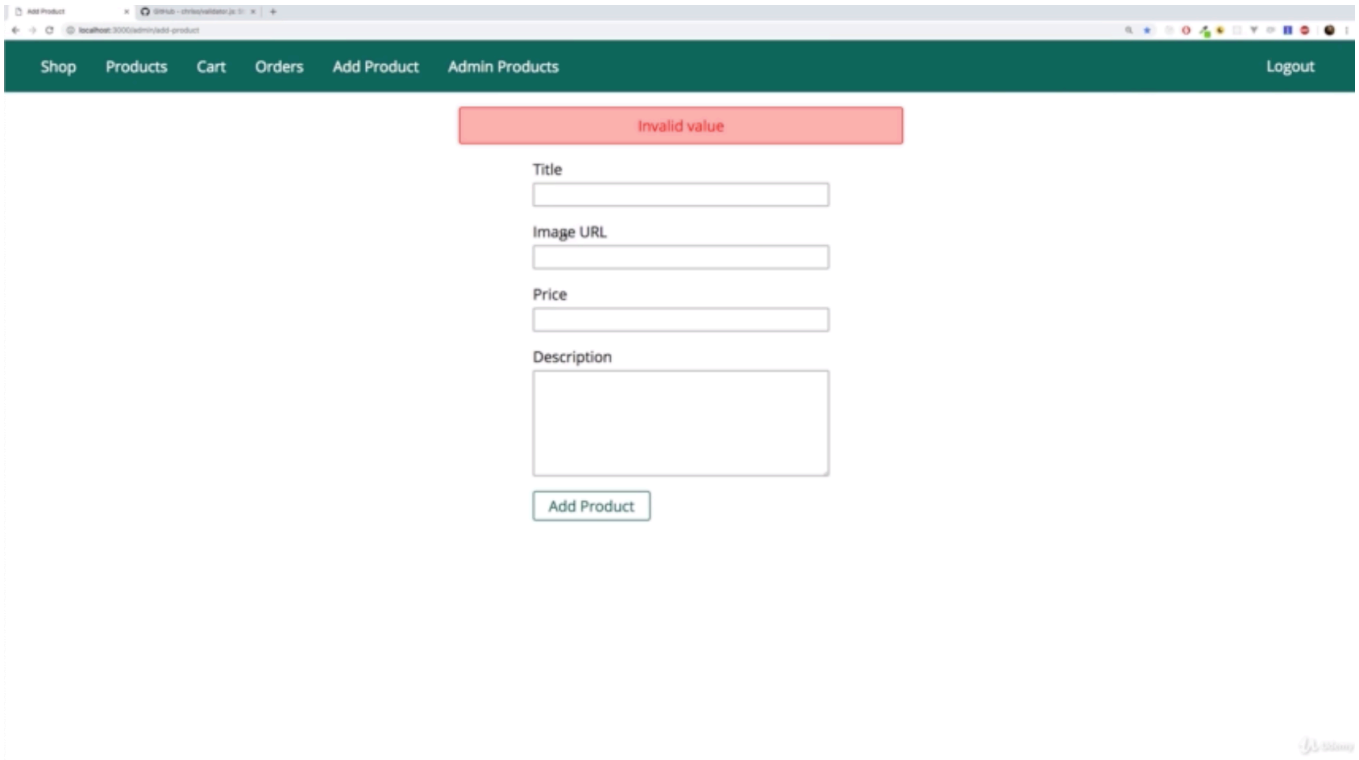
```
// ./routes/auth.js

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

const authController = require('../controllers/auth');
const User = require('../models/user')

const router = express.Router();

router.get(
    '/login',
    [
```

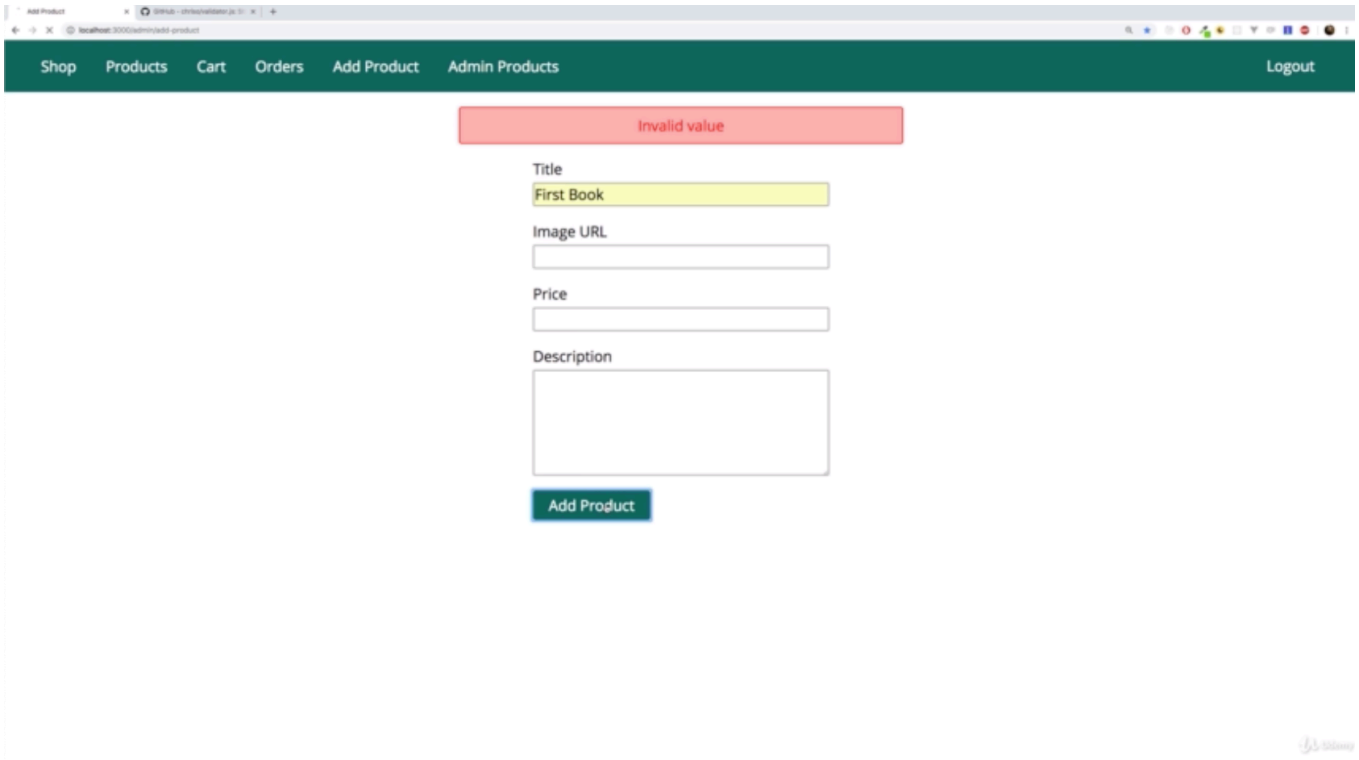
```
        body('email')
            .isEmail()
            .withMessage('Please enter a valid email address')
            /**'normalizeEmail()' is built-in sanitizers */
            .normalizeEmail(),
        body('password', 'Password has to be valid.')
            .isLength({ min: 5 })
            .isAlphanumeric()
            /**we could trin the password to remove excess whitespace */
            .trim()
    ],
    authController.getLogin)

router.get('/signup', authController.getSignup);

router.post('/login', authController.postLogin);

router.post(
    '/signup',
    [
        check('email')
            .isEmail()
            .withMessage('Please enter a valid email')
            .custom((value, {req}) => {
                //if (value === 'test@test.com'){
                //    throw new Error('This Email Address if forbidden.')
                //}
                //return true
                return User.findOne({ email: value })
                .then(userDoc => {
                  if (userDoc) {
                    return Promise.reject(
                        'E-Mail exists already, please pick a different one.'
                    )
                }
            })
        })
        .normalizeEmail(),
        body(
            'password',
            'Please enter a password with only numbers and text and at least 5 characters'
        )
            .isLength({min: 5})
            .isAlphanumeric()
            .trim(),
        body('confirmPassword')
            .trim()
            .custom((value, {req}) => {
            if (value !== req.body.password) {
                throw new Error('Passwords have to match!')
            }
            return true
        })
    ],
    authController.postSignup);

```

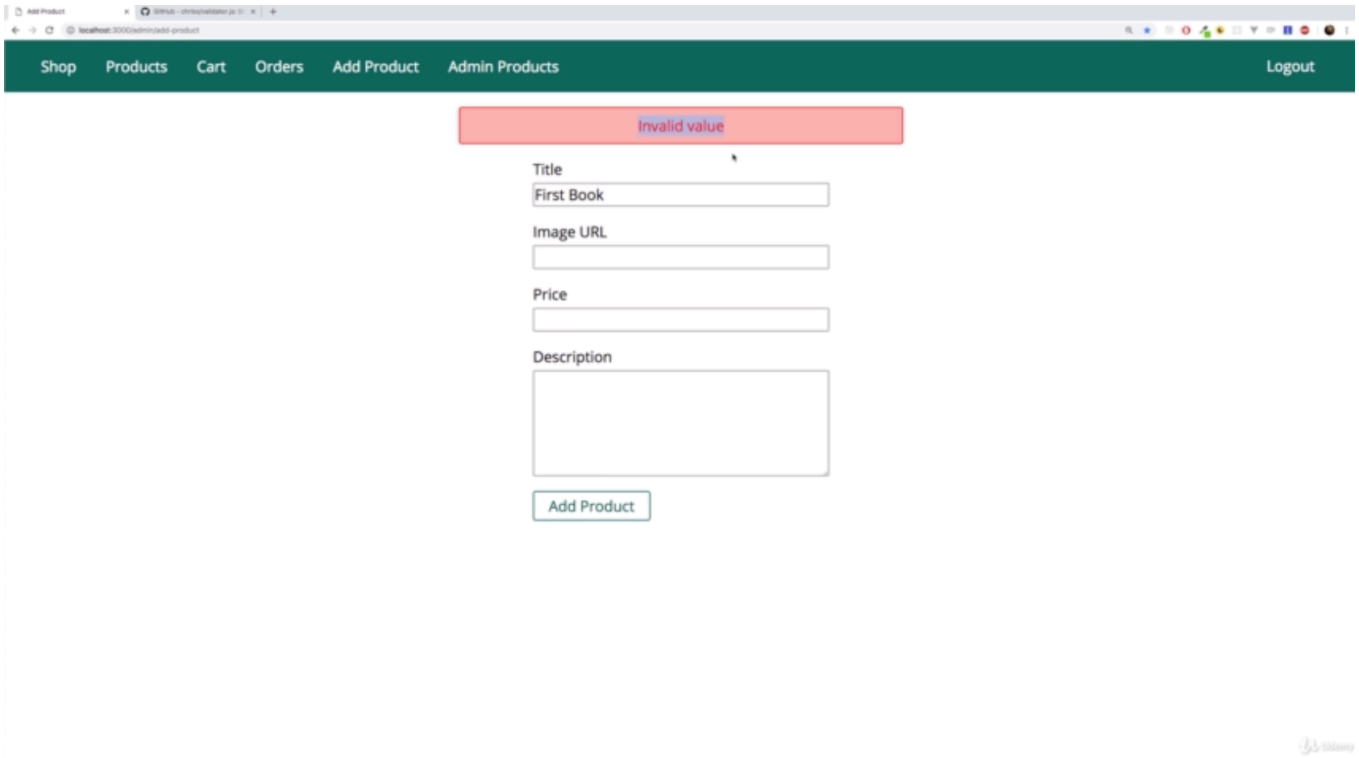
```
70 router.post('/logout', authController.postLogout);
71
72 router.get('/reset', authController.getReset);
73
74 router.post('/reset', authController.postReset);
75
76 router.get('/reset/:token', authController.getNewPassword);
77
78 router.post('/new-password', authController.postNewPassword);
79
80 module.exports = router;
```

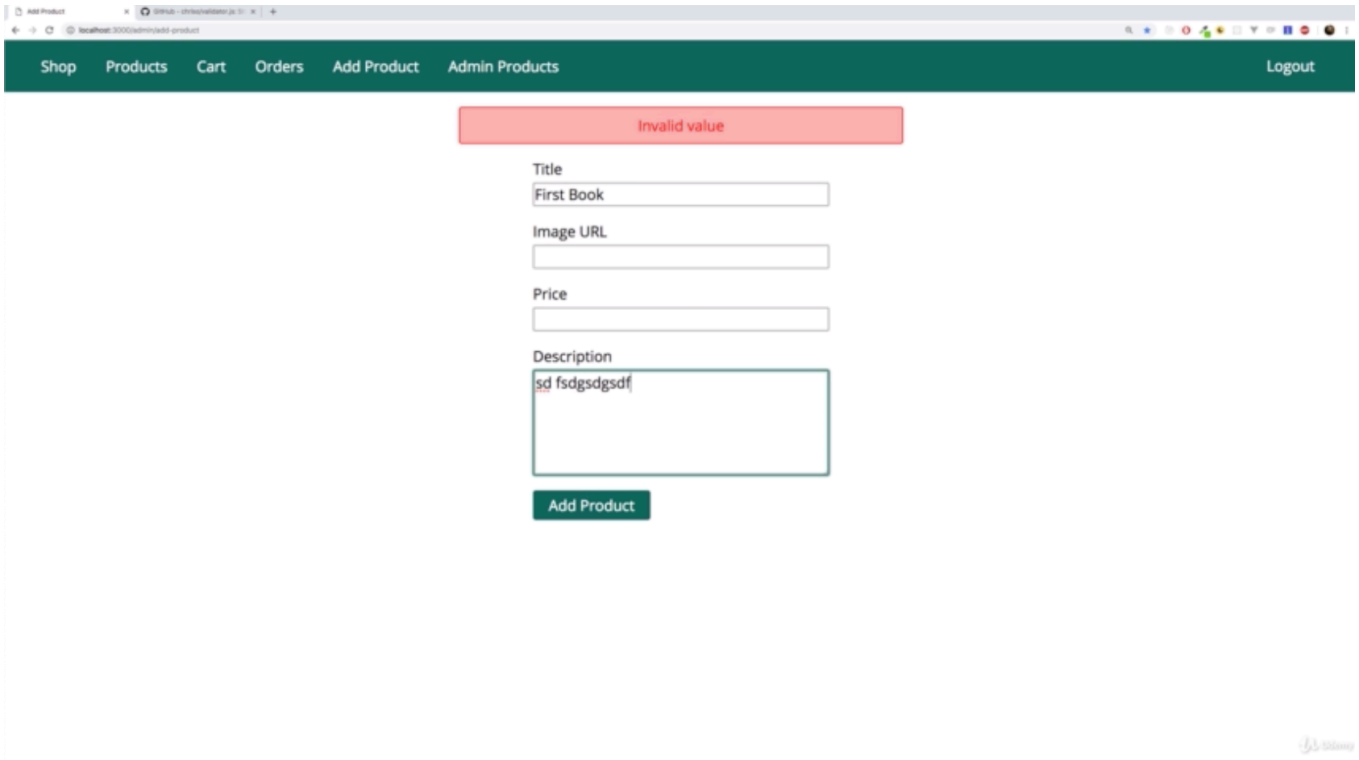
* Chapter 297: Validating Product Addition

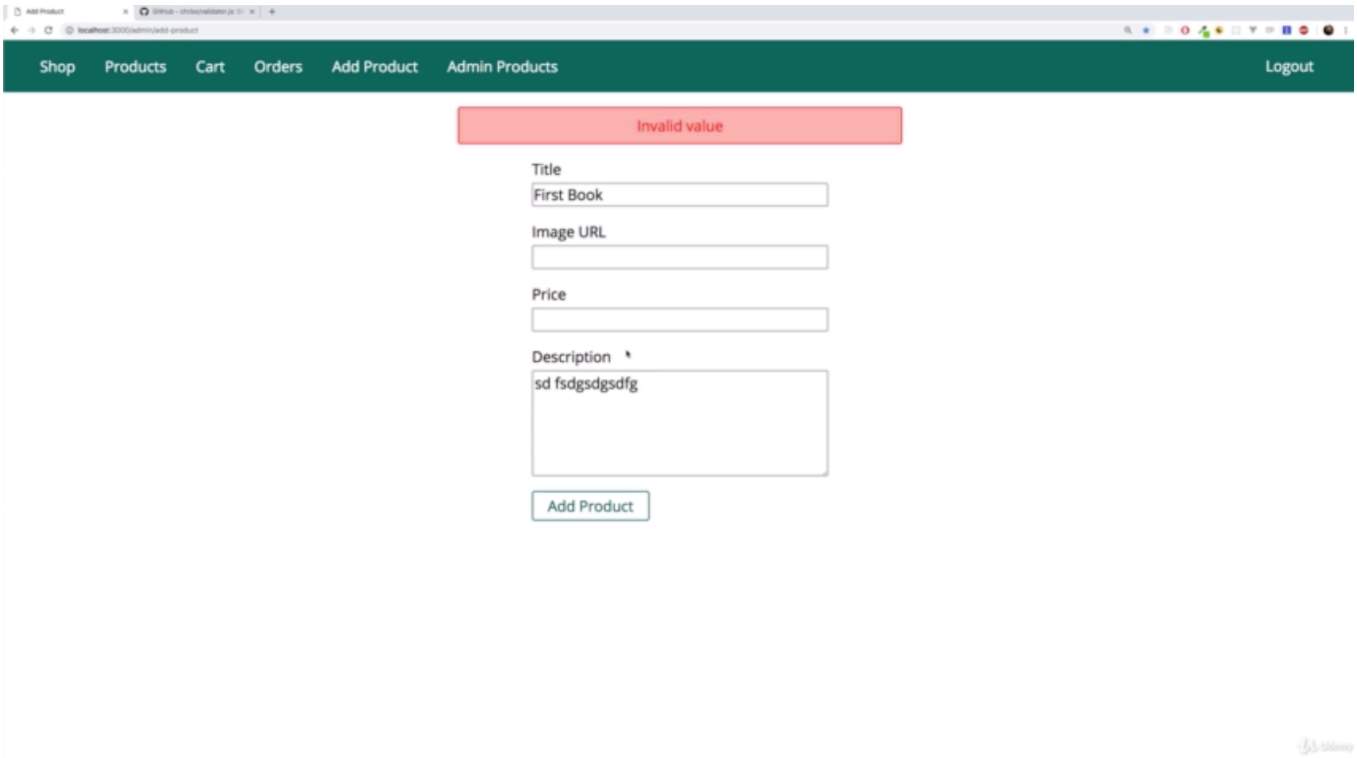
1. update
- ./routes/admin.js
- ./controllers/admin.js
- ./views/edit-product.ejs

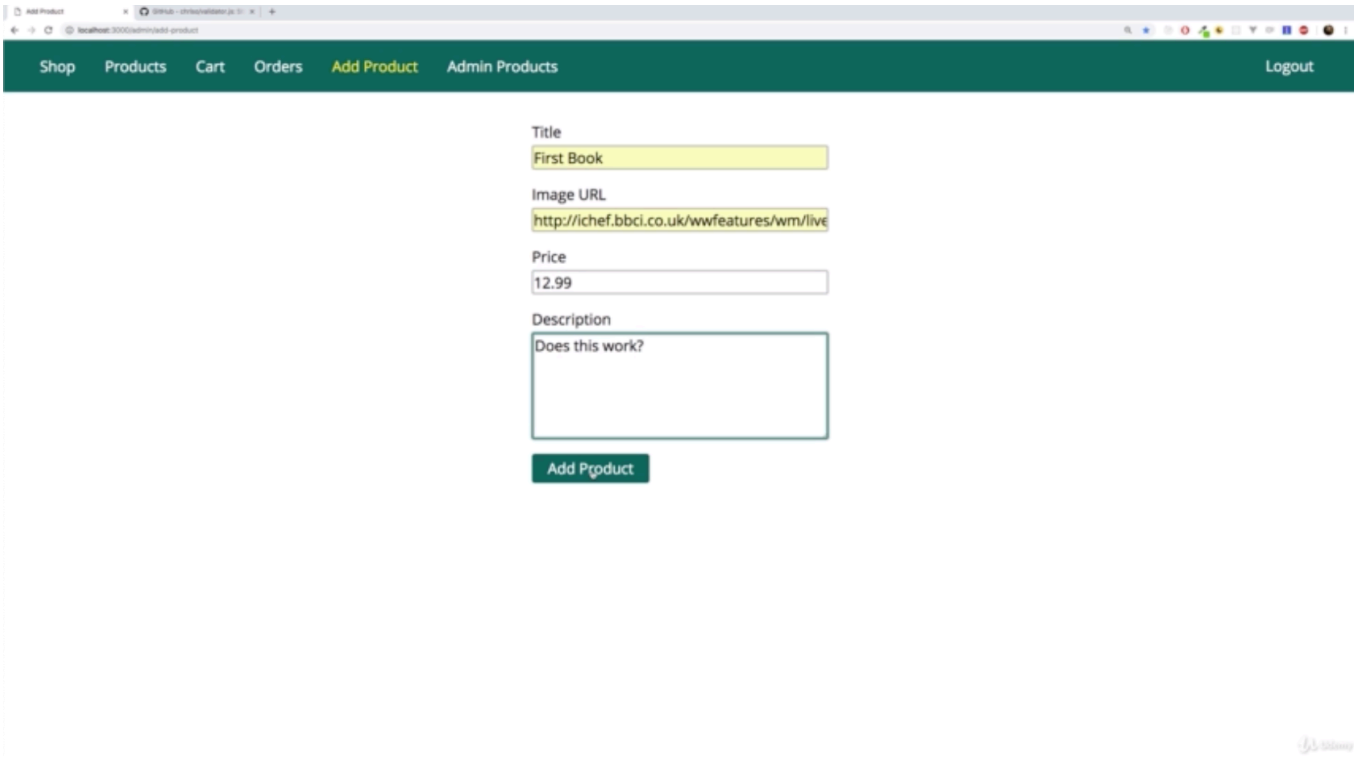

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
Image URL
Price
Description
Add Product

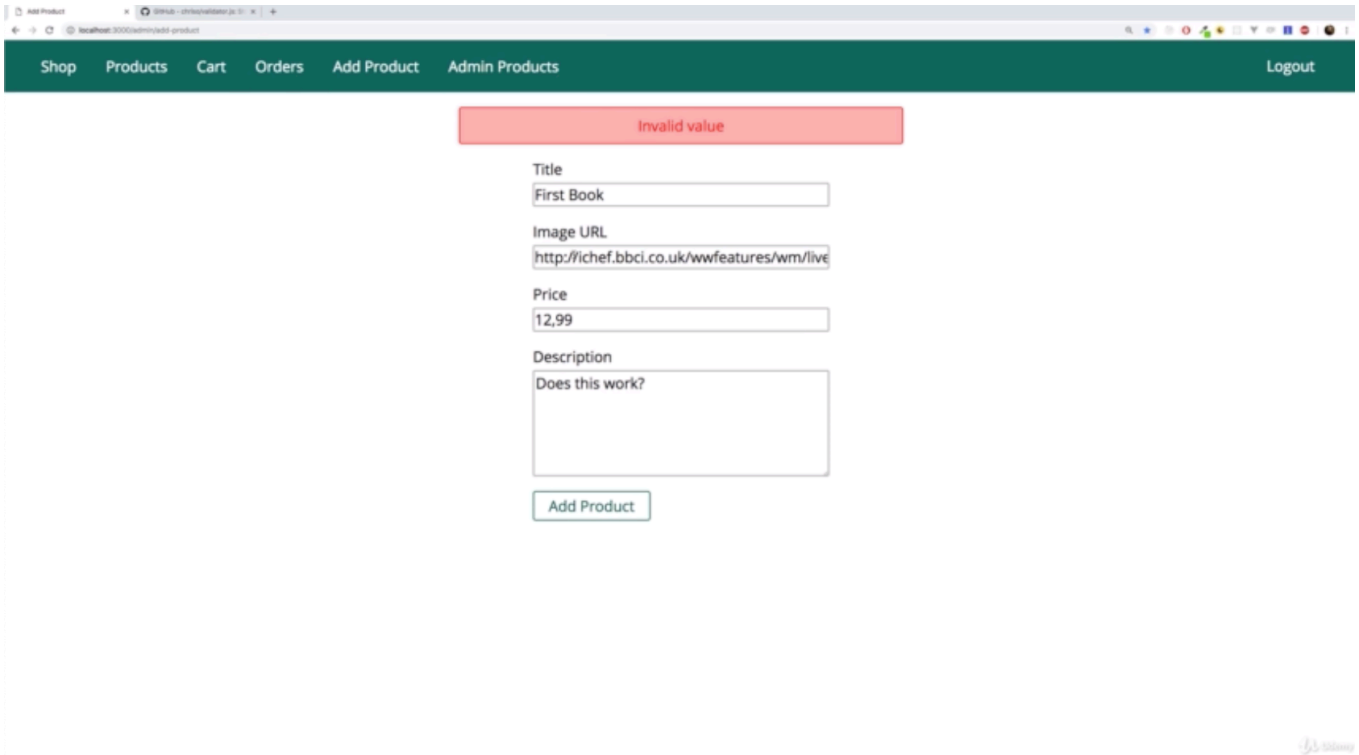

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
First Book
Image URL
Price
Description
Add Product

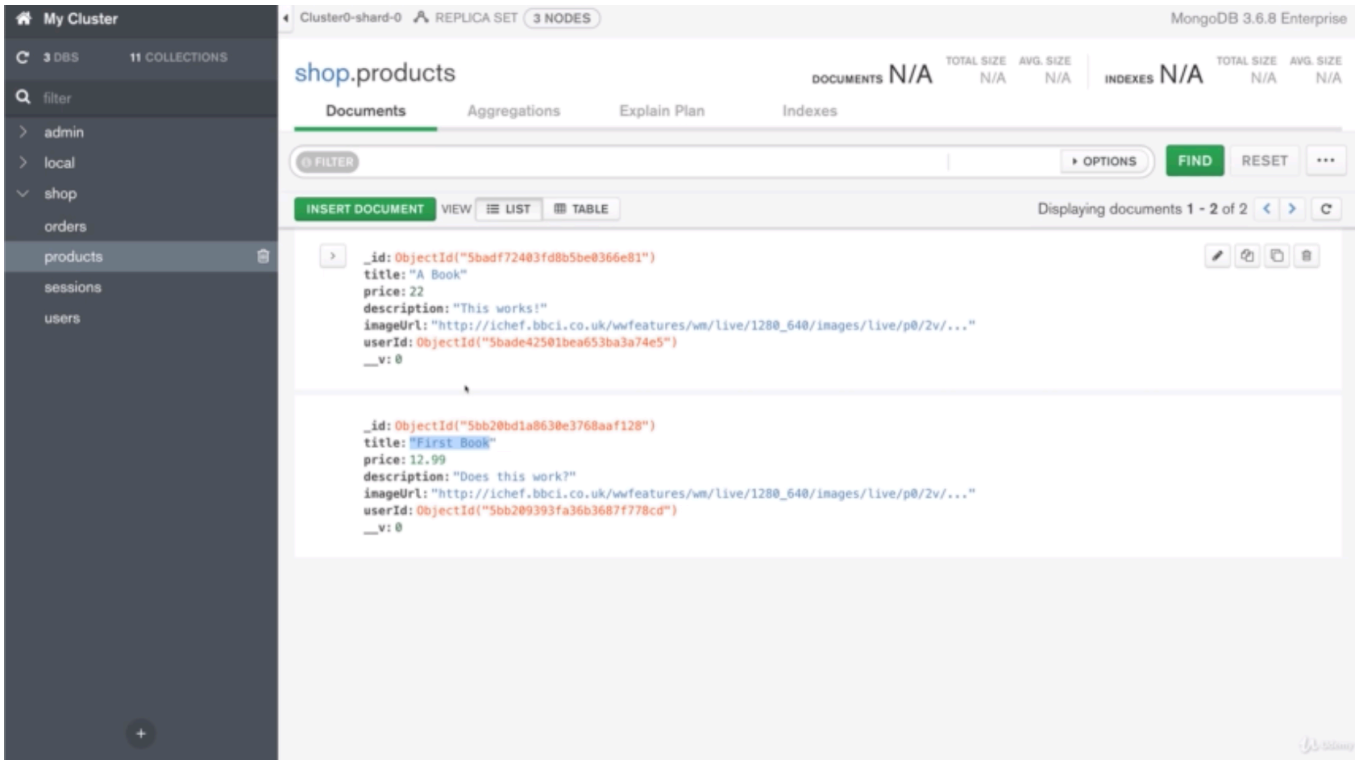

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
First Book
Image URL
Price
Description
Add Product

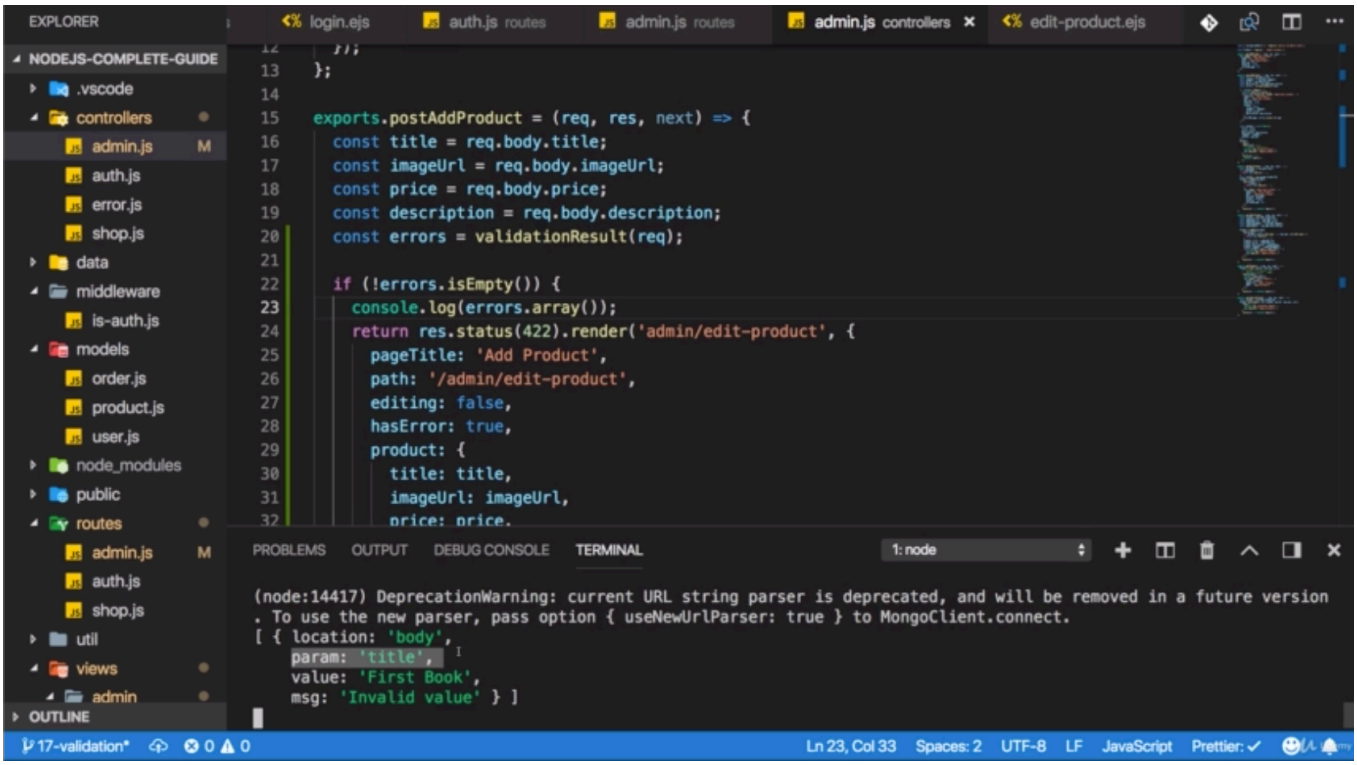

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
First Book
Image URL
Price
Description
sd fsdgsdgsdf
Add Product

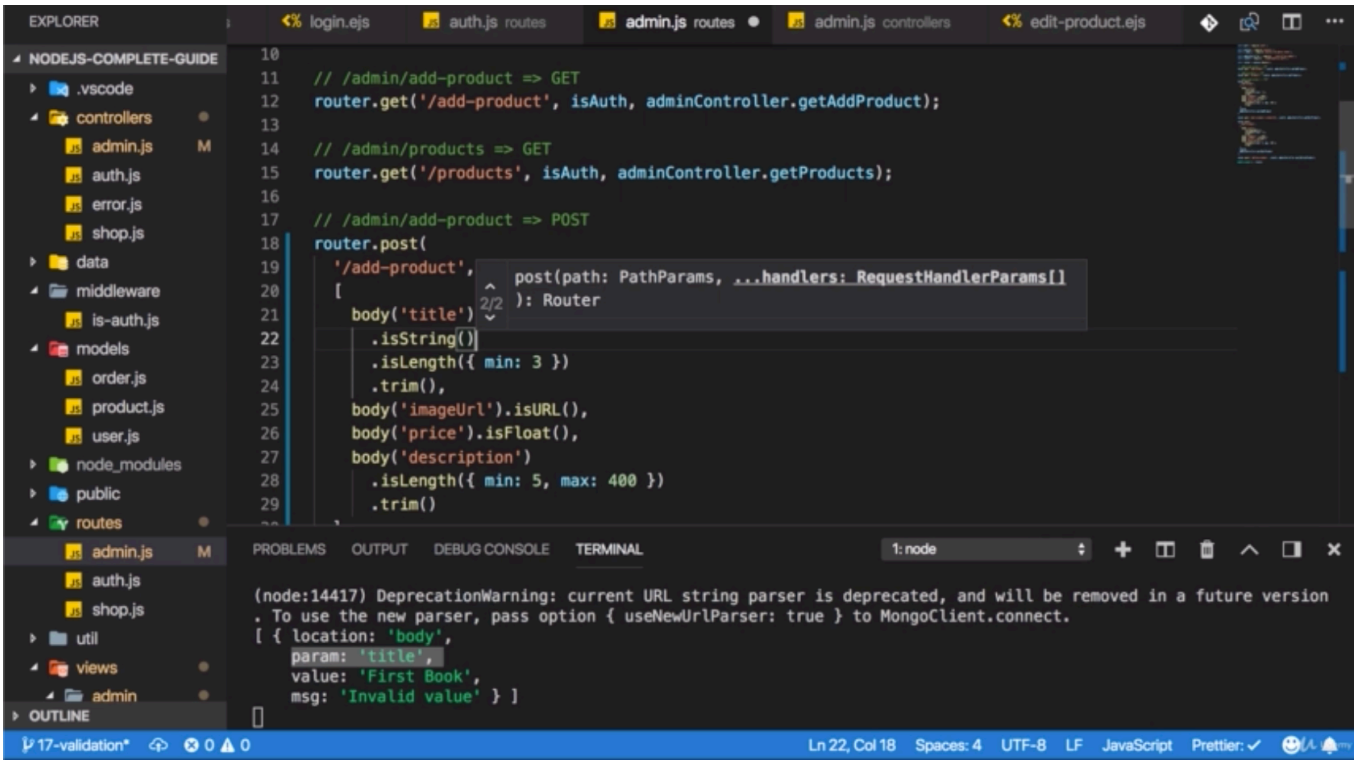

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
First Book
Image URL
Price
Description
sd fsdgsdgsdfg
Add Product

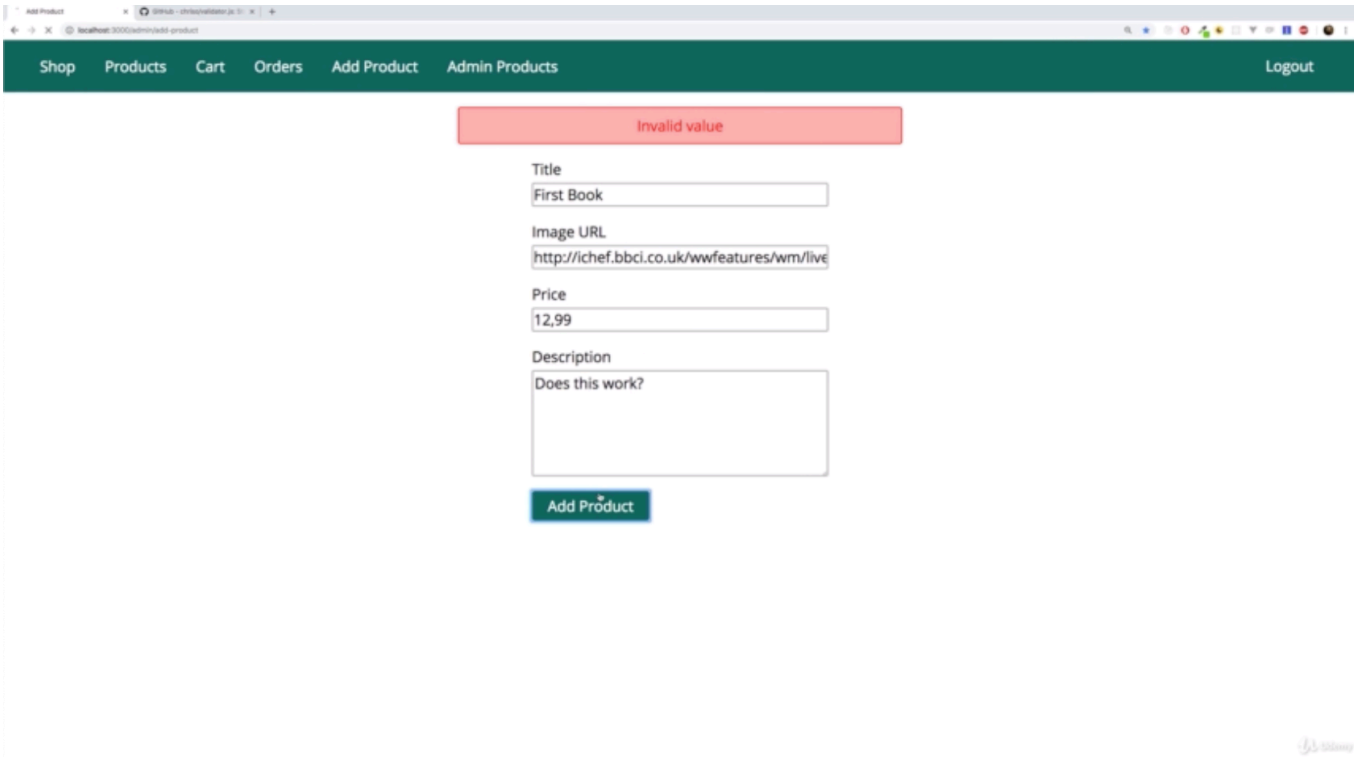

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Title
First Book
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
12.99
Description
Does this work?
Add Product

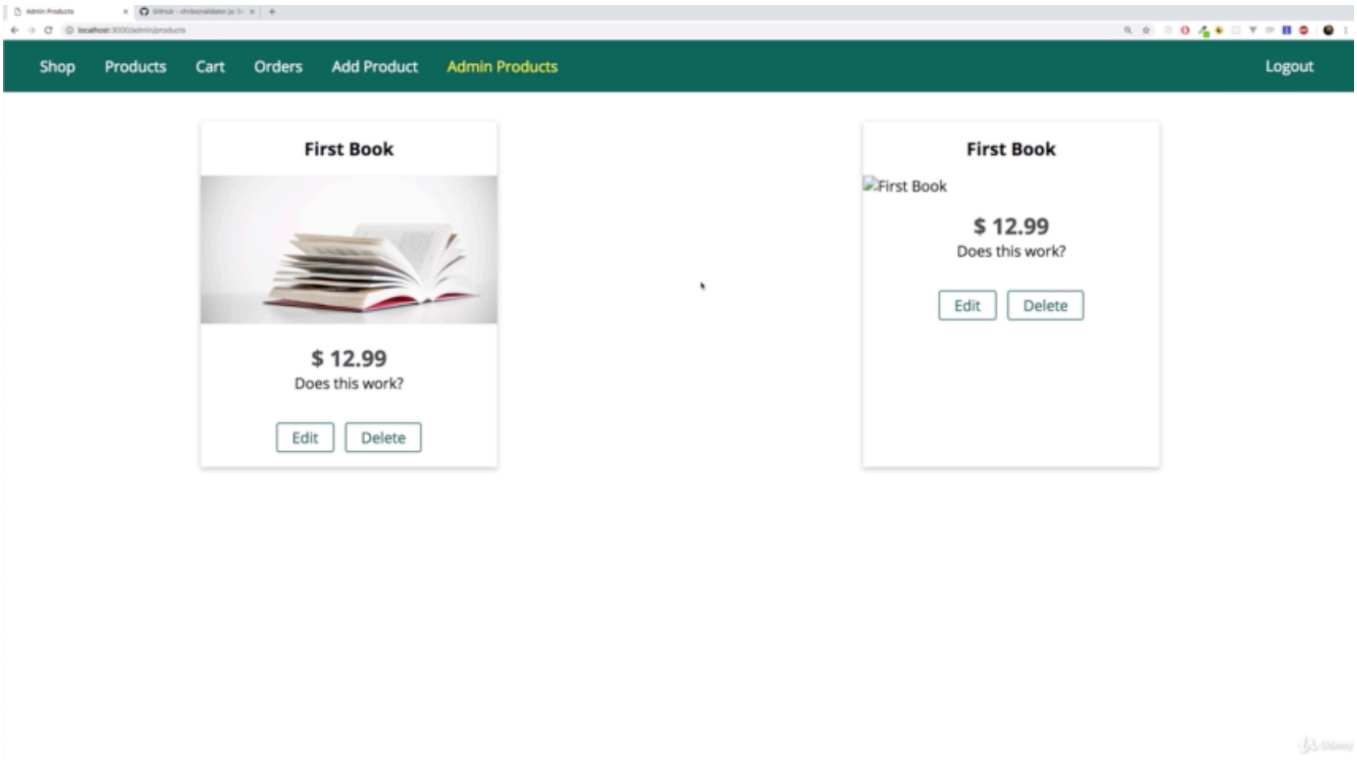

Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
First Book
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
12,99
Description
Does this work?
Add Product


My Cluster
3 DBS
11 COLLECTIONS
filter
admin
local
shop
orders
products
sessions
users
Cluster0-shard-0
REPLICA SET
3 NODES
MongoDB 3.6.8 Enterprise
shop.products
DOCUMENTS N/A
INDEXES N/A
Documents
Aggregations
Explain Plan
Indexes
FILTER
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW
LIST
TABLE
Displaying documents 1 - 2 of 2
_id: ObjectId("5badf72403fd8b5be0366e81")
title: "A Book"
price: 22
description: "This works!"
imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
userId: ObjectId("5bade42501bea653ba3a74e5")
__v: 0
_id: ObjectId("5bb20bd1a8630e3768aaf128")
title: "First Book"
price: 12.99
description: "Does this work?"
imageUrl: "http://ichef.bbci.co.uk/wwfeatures/wm/live/1280_640/images/live/p0/2v/..."
userId: ObjectId("5bb209393fa36b3687f778cd")
__v: 0


EXPLORER
login.ejs
auth.js routes
admin.js routes
admin.js controllers
edit-product.ejs
NODEJS-COMPLETE-GUIDE
.vscode
controllers
admin.js
auth.js
error.js
shop.js
data
middleware
is-auth.js
models
order.js
product.js
user.js
node_modules
public
routes
admin.js
auth.js
shop.js
util
views
admin
OUTLINE
});
};
exports.postAddProduct = (req, res, next) => {
const title = req.body.title;
const imageUrl = req.body.imageUrl;
const price = req.body.price;
const description = req.body.description;
const errors = validationResult(req);
if (!errors.isEmpty()) {
console.log(errors.array());
return res.status(422).render('admin/edit-product', {
pageTitle: 'Add Product',
path: '/admin/edit-product',
editing: false,
hasError: true,
product: {
title: title,
imageUrl: imageUrl,
price: price,
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: node
(node:14417) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[{ location: 'body',
param: 'title',
value: 'First Book',
msg: 'Invalid value' }]
17-validation*
0 0
Ln 23, Col 33
Spaces: 2
UTF-8
LF
JavaScript
Prettier:


EXPLORER
login.ejs
auth.js routes
admin.js routes
admin.js controllers
edit-product.ejs
NODEJS-COMPLETE-GUIDE
.vscode
controllers
admin.js
auth.js
error.js
shop.js
data
middleware
is-auth.js
models
order.js
product.js
user.js
node_modules
public
routes
admin.js
auth.js
shop.js
util
views
admin
OUTLINE
// /admin/add-product => GET
router.get('/add-product', isAuth, adminController.getAddProduct);
// /admin/products => GET
router.get('/products', isAuth, adminController.getProducts);
// /admin/add-product => POST
router.post(
'/add-product',
[
body('title')
.isString()
.isLength({ min: 3 })
.trim(),
body('imageUrl').isURL(),
body('price').isFloat(),
body('description')
.isLength({ min: 5, max: 400 })
.trim()
2/2
post(path: PathParams, ...handlers: RequestHandlerParams[]
): Router
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: node
(node:14417) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version
. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.
[{ location: 'body',
param: 'title',
value: 'First Book',
msg: 'Invalid value' }]
17-validation*
0 0
Ln 22, Col 18
Spaces: 4
UTF-8
LF
JavaScript
Prettier:


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
First Book
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
12,99
Description
Does this work?
Add Product


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
First Book
$ 12.99
Does this work?
Edit
Delete
First Book
First Book
$ 12.99
Does this work?
Edit
Delete

Shop Products Cart Orders Add Product Admin Products Logout

First Book

$ 12.99

Does this work?

Edit Delete

First Book

First Book

$ 12.99

Does this work?

Edit Delete

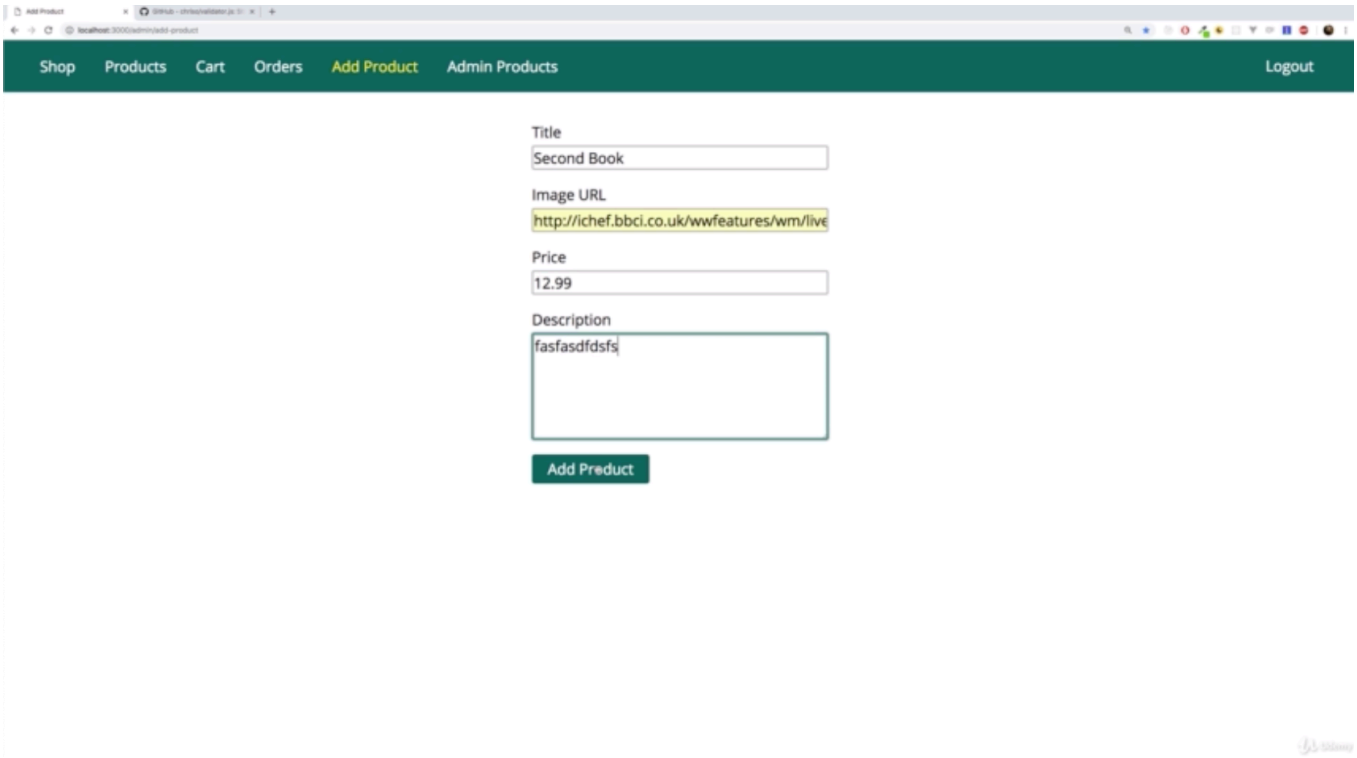
Shop Products Cart Orders Add Product Admin Products Logout

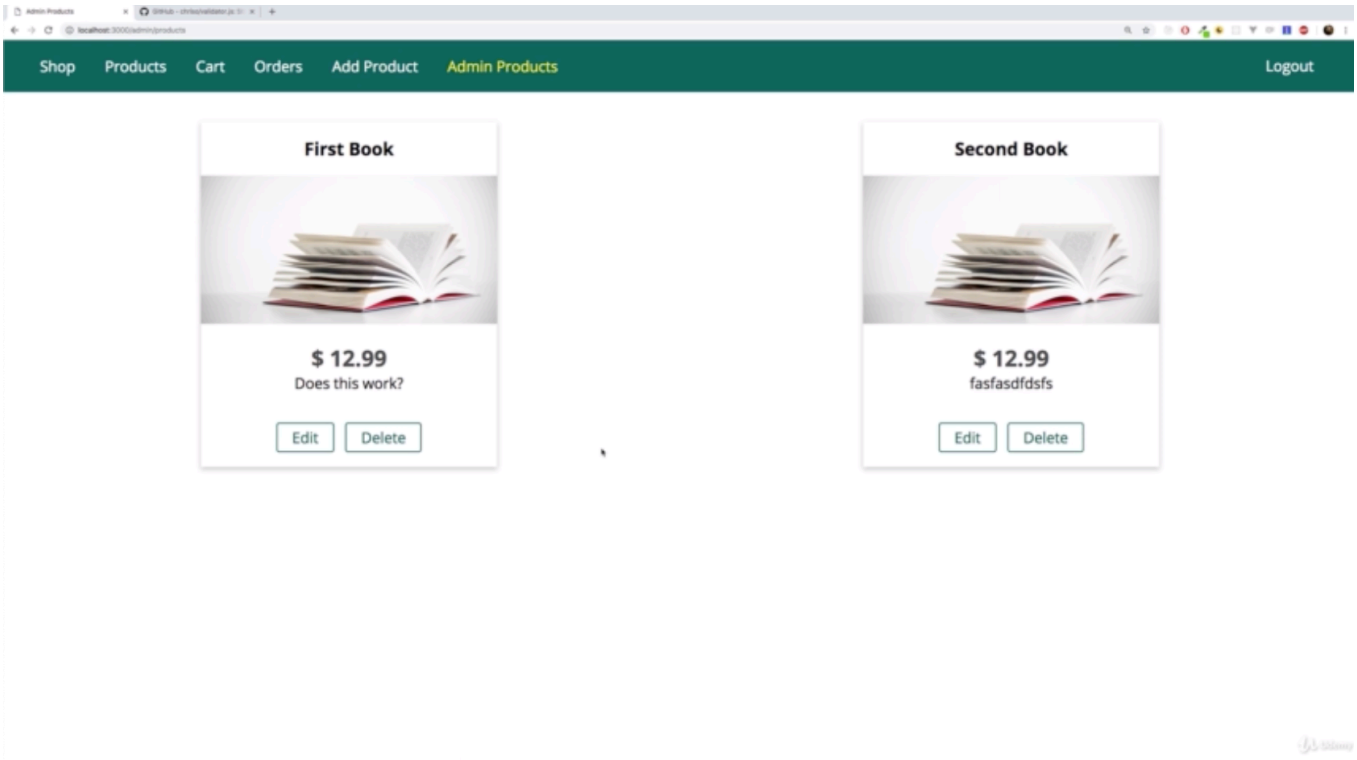
First Book

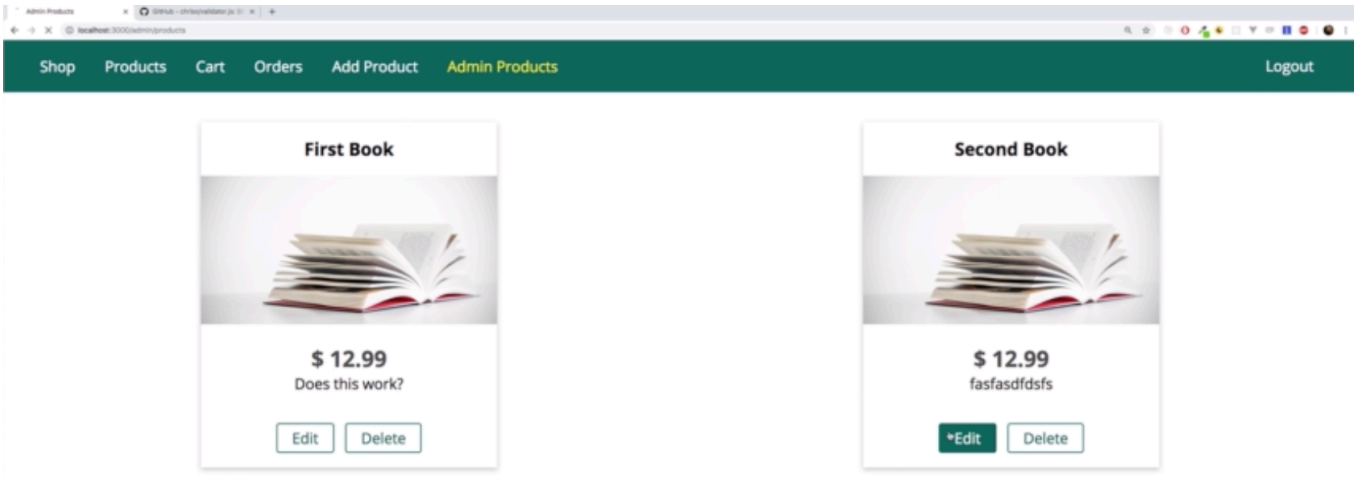
$ 12.99

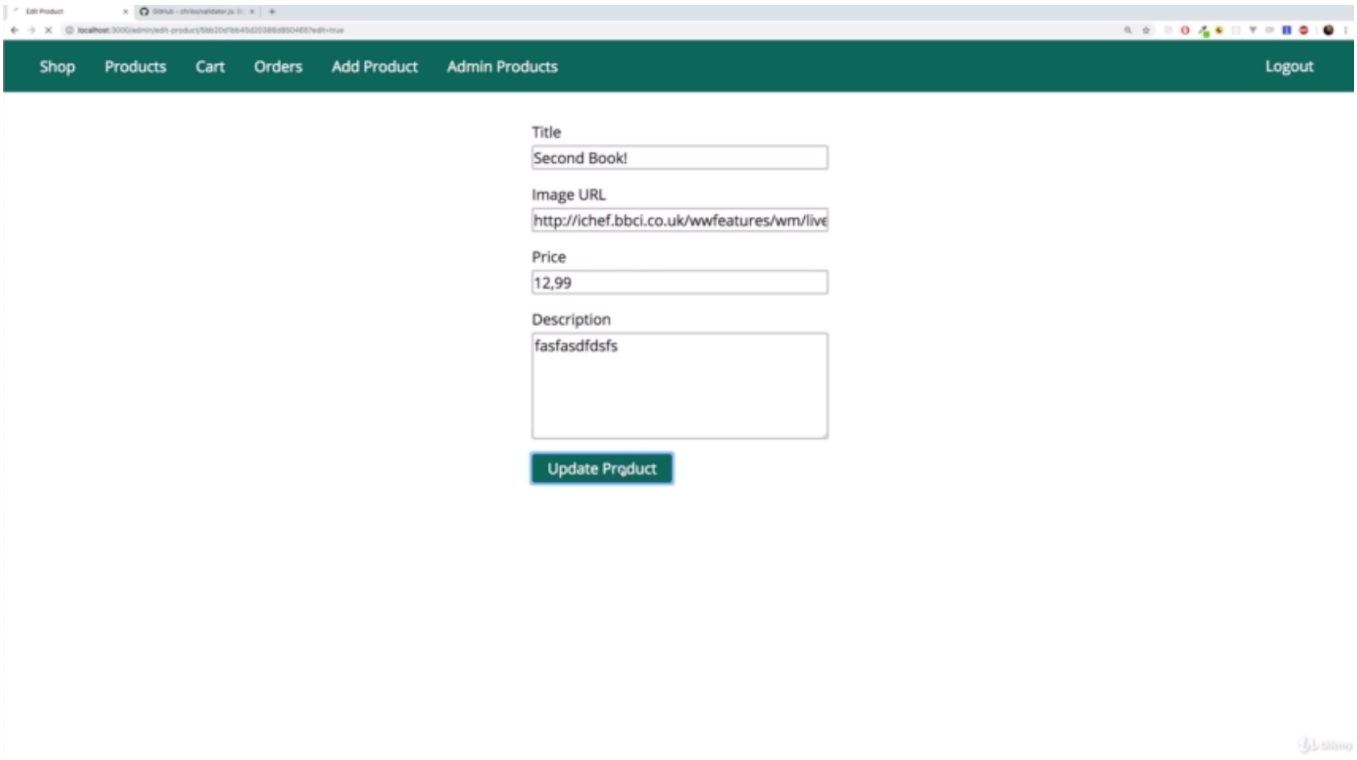
Does this work?

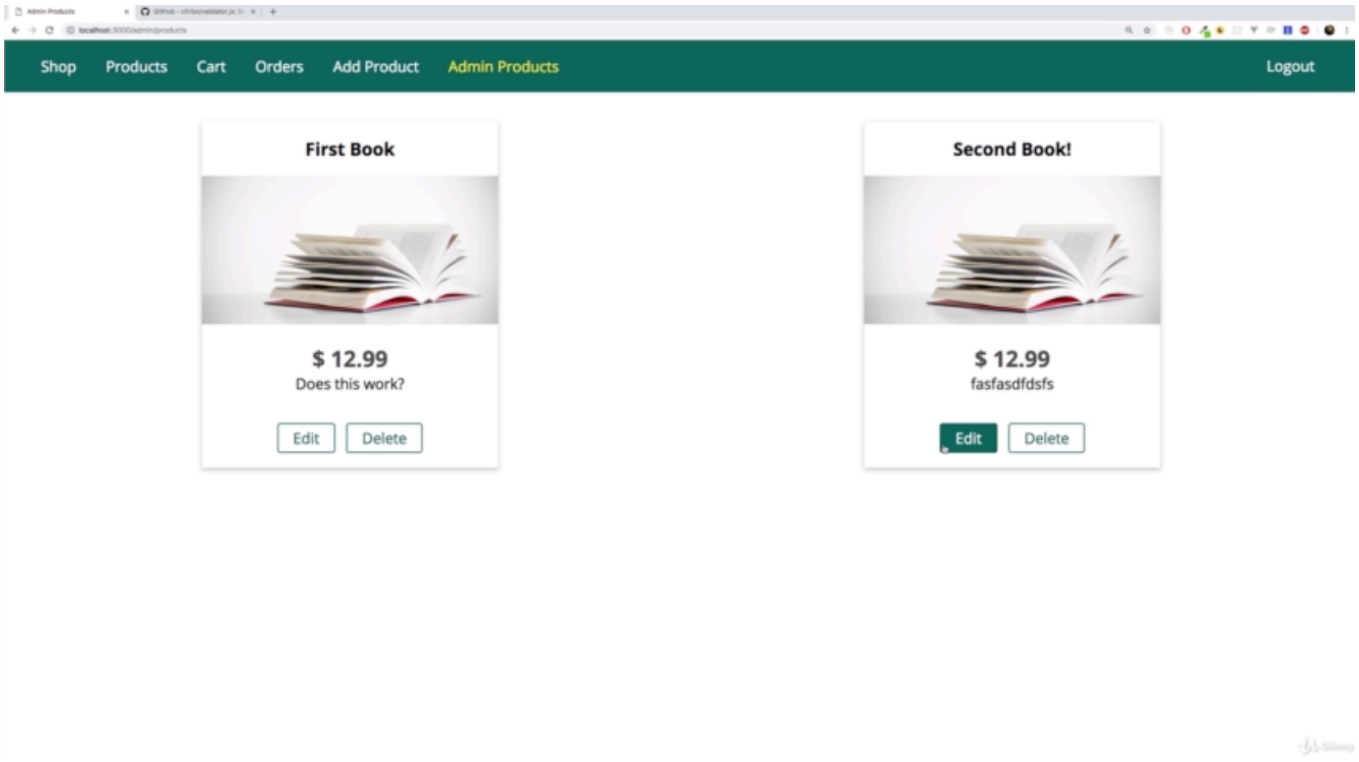
Edit Delete


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Title
Second Book
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
12.99
Description
fasfasdfdsfs
Add Product


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
First Book
$ 12.99
Does this work?
Edit
Delete
Second Book
$ 12.99
fasfasdfdsfs
Edit
Delete


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
First Book
$ 12.99
Does this work?
Edit
Delete
Second Book
$ 12.99
fasfasdfdsfs
Edit
Delete


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Title
Second Book!
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
12,99
Description
fasfasdfdsfs
Update Product




```
// ./routes/admin.js

const path = require('path');

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

const adminController = require('../controllers/admin');
const isAuth = require('../middleware/is-auth');

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', isAuth, adminController.getAddProduct);

// /admin/products => GET
router.get('/products', isAuth, adminController.getProducts);

// /admin/add-product => POST
router.post('/add-product', [
        body('title').isString().isLength({ min: 3 }).trim(),
        body('imageUrl').isURL(),
        body('price').isFloat(),
        body('description').isLength({ min: 5, max: 400 }).trim()
    ],
    isAuth,
    adminController.postAddProduct);

router.get('/edit-product/:productId', isAuth, adminController.getEditProduct);

router.post('/edit-product',[
        body('title').isAlphanumeric().isLength({ min: 3 }).trim(),
        body('imageUrl').isURL(),
        body('price').isFloat(),
        body('description').isLength({ min: 5, max: 400 }).trim()
```

```
    ],
    isAuth,
    adminController.postEditProduct);

router.post('/delete-product', isAuth, adminController.postDeleteProduct);

module.exports = router;
```

```
// ./controllers/admin.js

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

const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false,
    hasError: false,
    errorMessage: null
  });
};

exports.postAddProduct = (req, res, next) => {
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  const errors = validationResult(req)

  if(!errors.isEmpty()) {
    return res.status(422).render('admin/edit-product', {
      pageTitle: 'Add Product',
      path: '/admin/edit-product',
      editing: false,
      hasError: true,
      product: {
        title: title,
        imageUrl: imageUrl,
        price: price,
        description: description
      },
      errorMessage: errors.array()[0].msg
    })
  }

  const product = new Product({
    title: title,
    price: price,
    description: description,
    imageUrl: imageUrl,
    userId: req.user
  });
  product
    .save()
    .then(result => {
```

```
      // console.log(result);
      console.log('Created Product');
      res.redirect('/admin/products');
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getEditProduct = (req, res, next) => {
  const editMode = req.query.edit;
  if (!editMode) {
    return res.redirect('/');
  }
  const prodId = req.params.productId;
  Product.findById(prodId)
    .then(product => {
      if (!product) {
        return res.redirect('/');
      }
      res.render('admin/edit-product', {
        pageTitle: 'Edit Product',
        path: '/admin/edit-product',
        editing: editMode,
        product: product,
        hasError: false,
        errorMessage: null
        });
    })
    .catch(err => console.log(err));
};

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;

  Product.findById(prodId)
    .then(product => {
      /**i should convert both to a string
       * because i'm also checking for type equality.
       */
      if(product.userId.toString() !== req.user._id.toString()){
        return res.redirect('/')
      }
      product.title = updatedTitle;
      product.price = updatedPrice;
      product.description = updatedDesc;
      product.imageUrl = updatedImageUrl;
      return product.save()
        .then(result => {
          console.log('UPDATED PRODUCT!');
          res.redirect('/admin/products');
        })
```

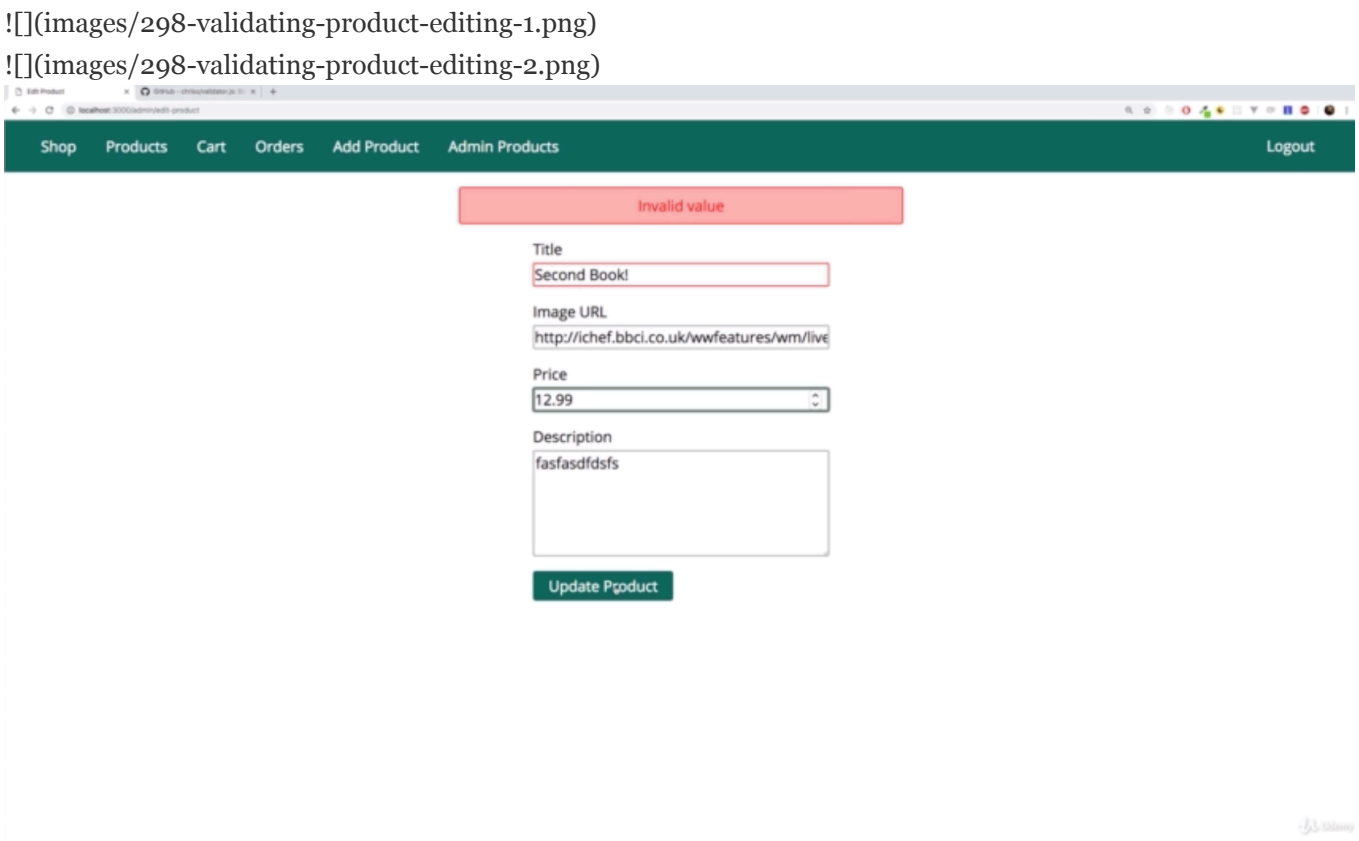
```
    })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.find({userId: req.user._id})
    // .select('title price -_id')
    // .populate('userId', 'name')
    .then(products => {
      console.log(products);
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  Product.deleteOne({_id: prodId, userId: req.user._id})
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

```

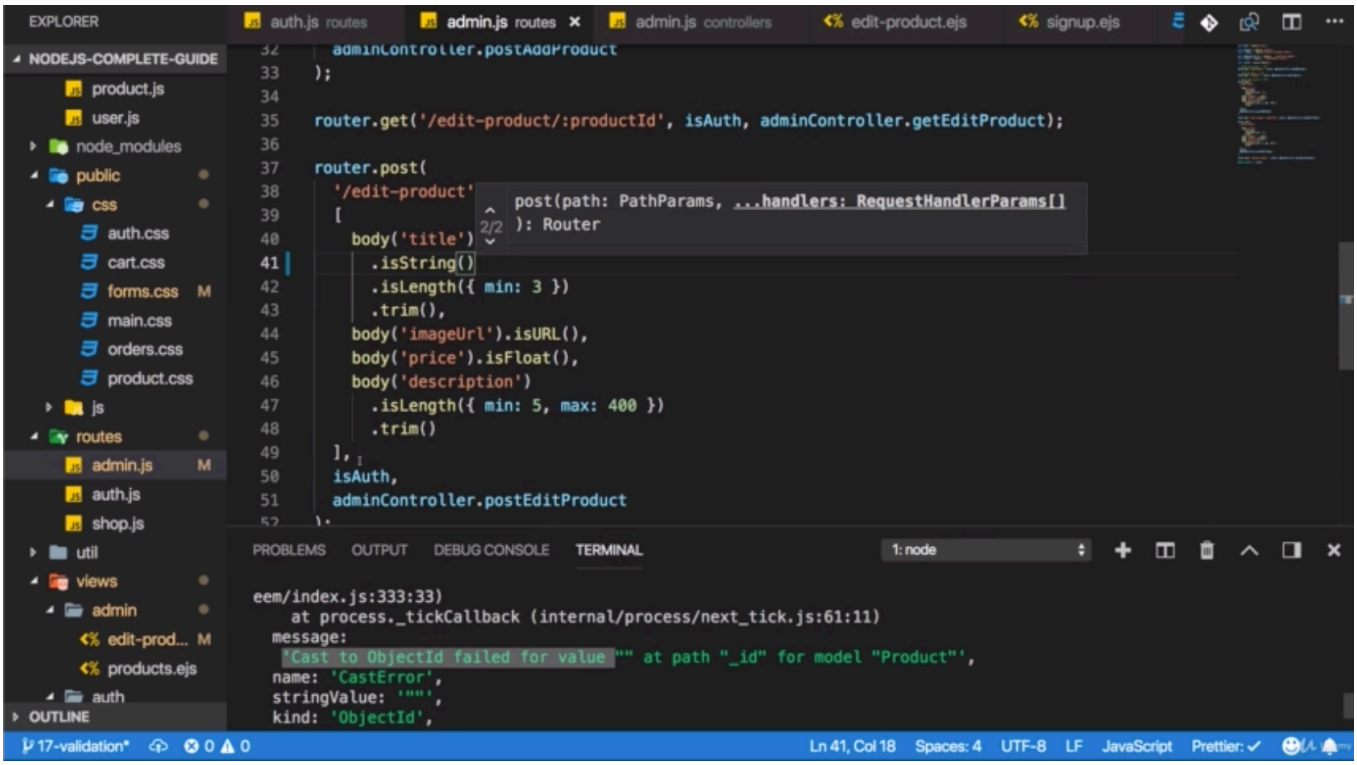
```
<!--./views/admin/edit-product.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="stylesheet" href="/css/product.css">
</head>

<body>
   <%- include('../includes/navigation.ejs') %>

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else { %>add-product<% } %>" method="POST">
            <div class="form-control">
                <label for="title">Title</label>
                <input type="text" name="title" id="title" value="<% if (editing || hasError) { %><%= product.title %><% } %>">
            </div>
            <div class="form-control">
                <label for="imageUrl">Image URL</label>
                <input type="text" name="imageUrl" id="imageUrl" value="<% if (editing || hasError) { %><%= product.imageUrl %><% } %>">
            </div>
            <div class="form-control">
```

```
25              <label for="price">Price</label>
26              <input type="number" name="price" id="price" step="0.01" value="<% if
(editing || hasError) { %><%= product.price %><% } %>">
27          </div>
28          <div class="form-control">
29              <label for="description">Description</label>
30              <textarea name="description" id="description" rows="5"><% if (editing ||
hasError) { %><%= product.description %><% } %></textarea>
31          </div>
32          <% if (editing) { %>
33              <input type="hidden" value="<%= product._id %>" name="productId">
34          <% } %>
35
36          <input type="hidden" name="_csrf" value="<%= csrfToken %>">
37          <button class="btn" type="submit"><% if (editing) { %>Update Product<% } else {
%>Add Product<% } %></button>
38      </form>
39    </main>
40 <%- include('../includes/end.ejs') %>
```

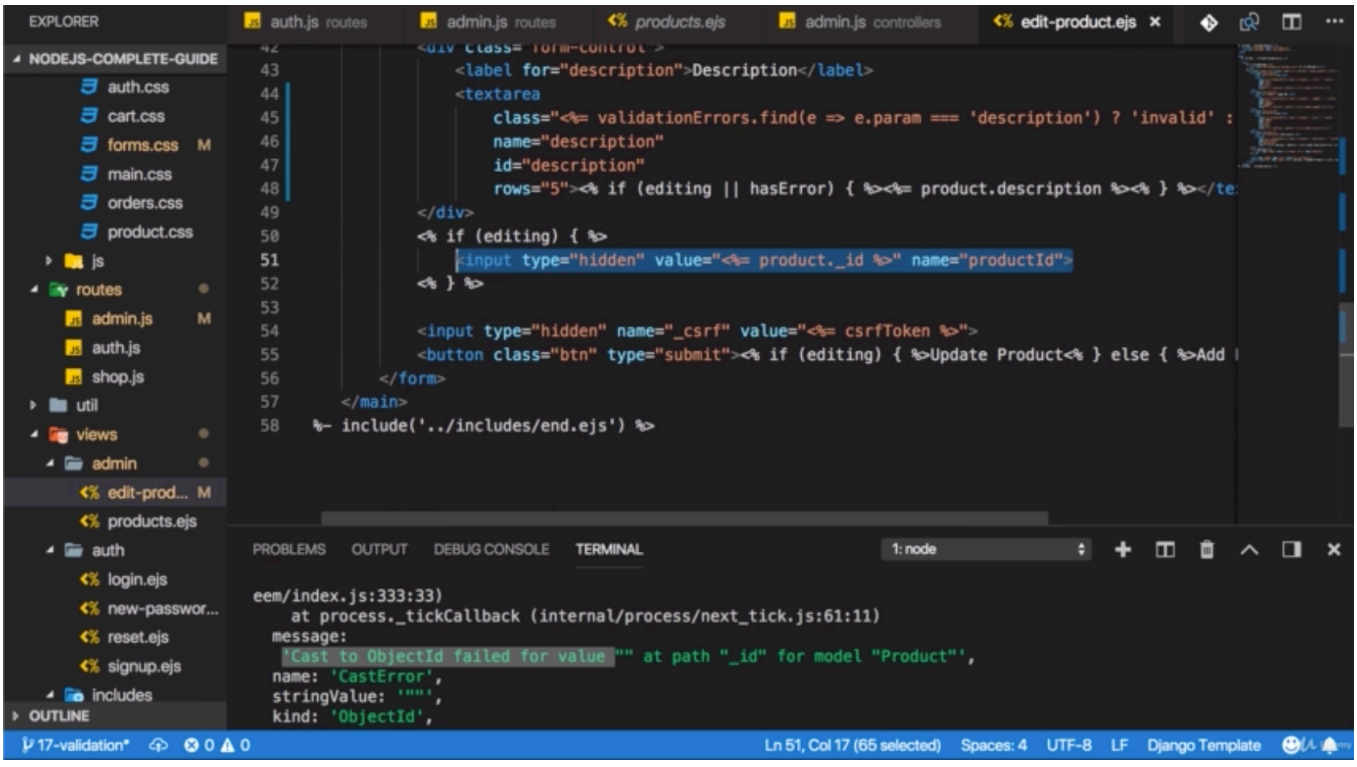
* Chapter 298: Validating Product Editing

1. update
- ./controllers/admin.js
- ./views/auth/edit-product.ejs
- ./public/css/forms.css
- ./routes/admin.js

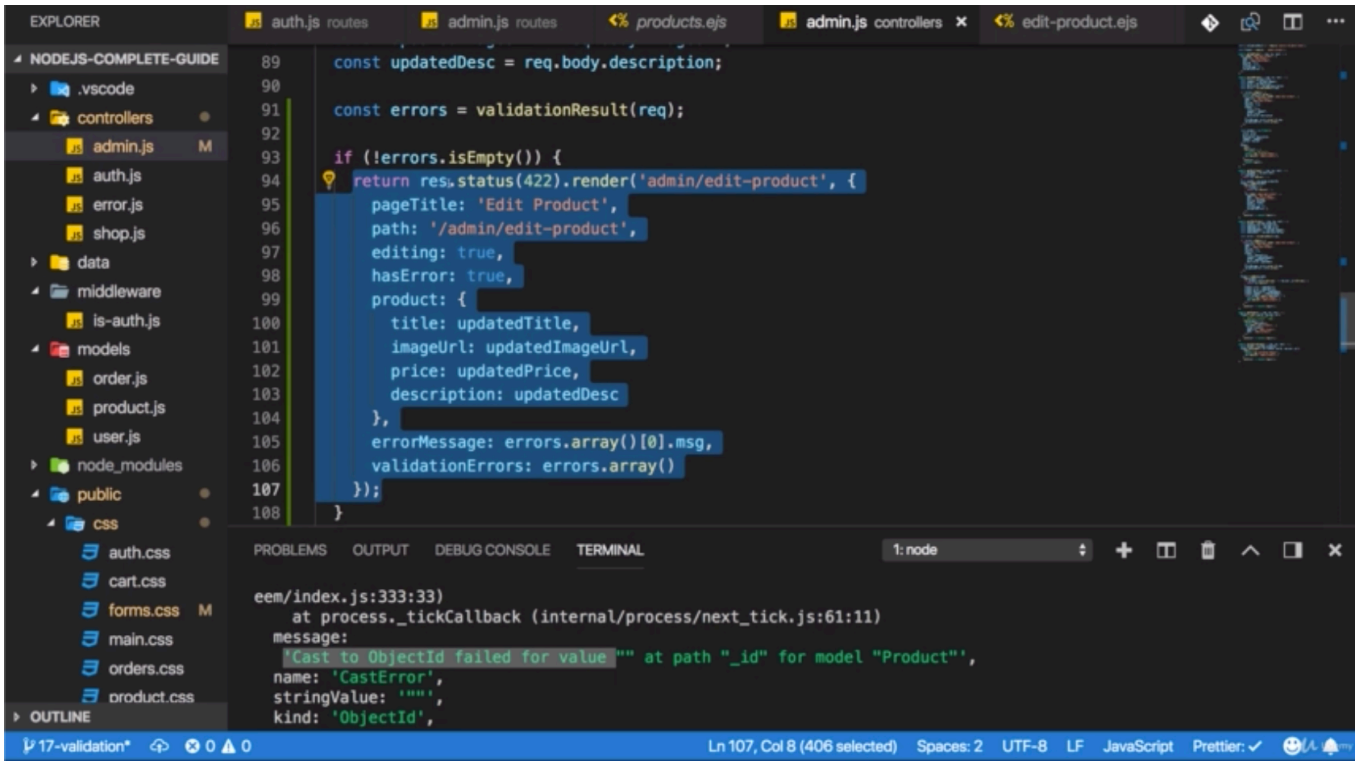




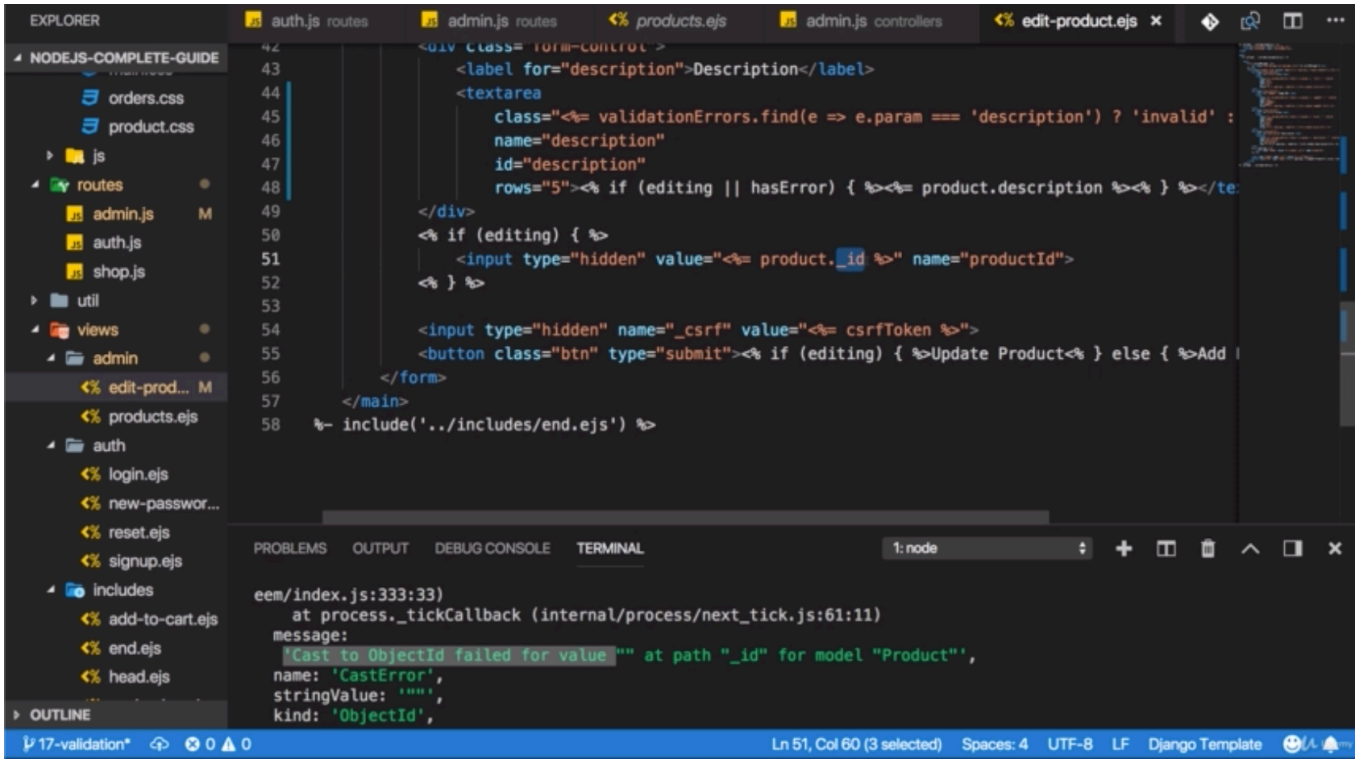
- what's wrong? because the case to object id failed for that value.

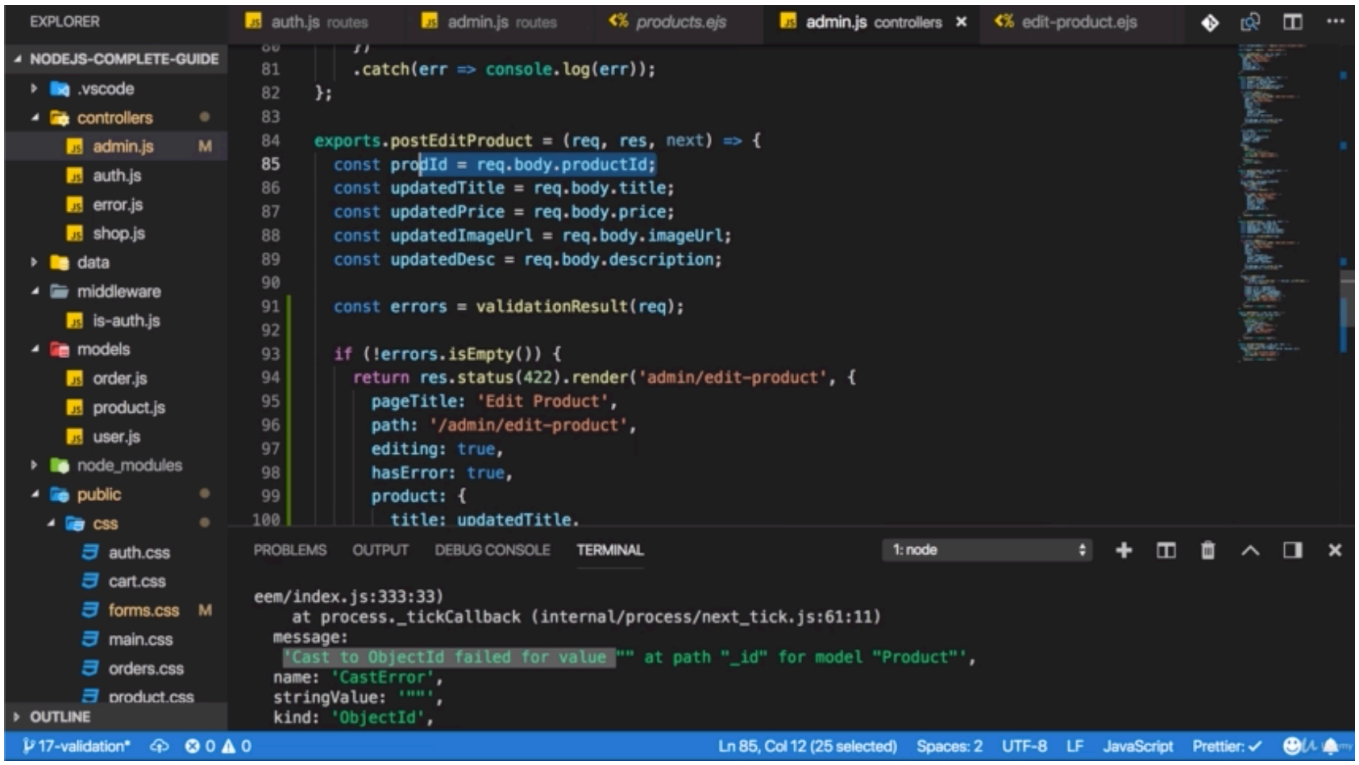


- we need that hidden input where i have my product._id. and i load that product_.id when i first render this page.

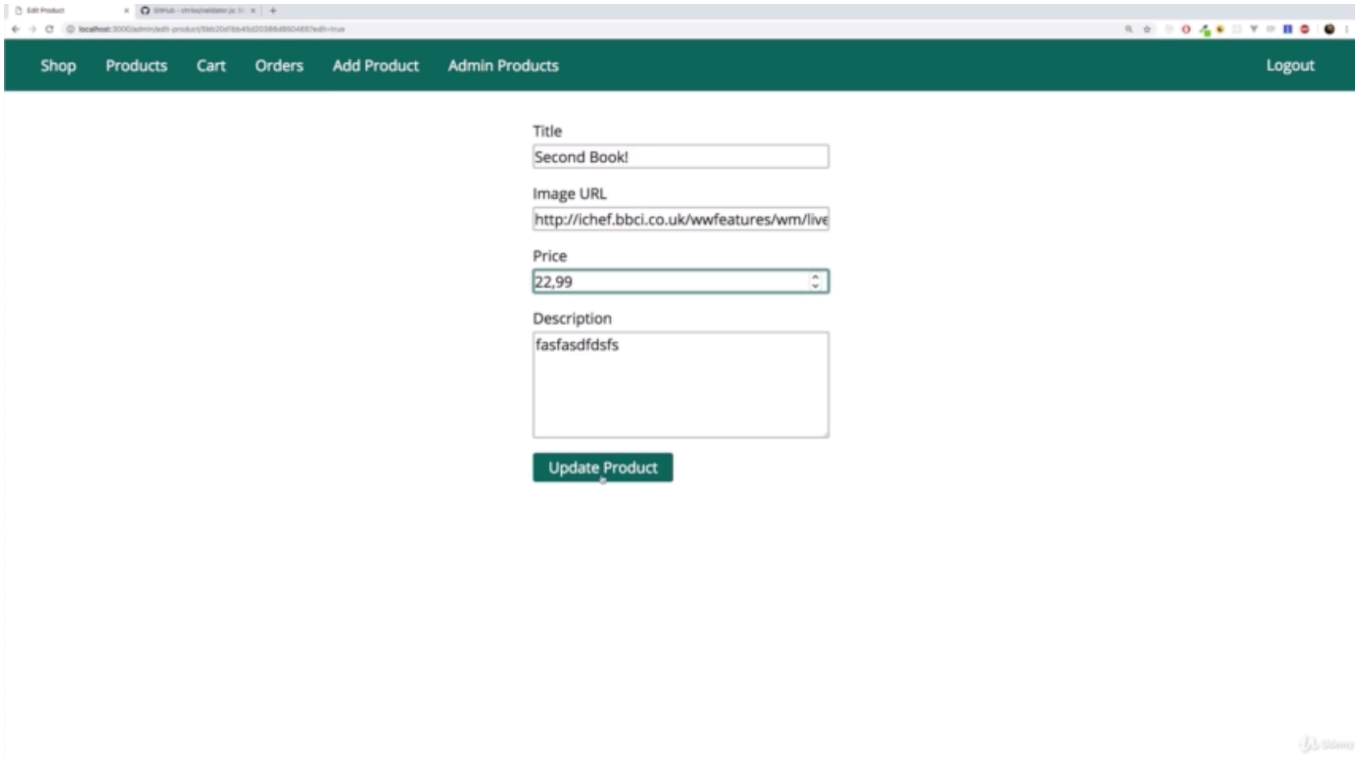


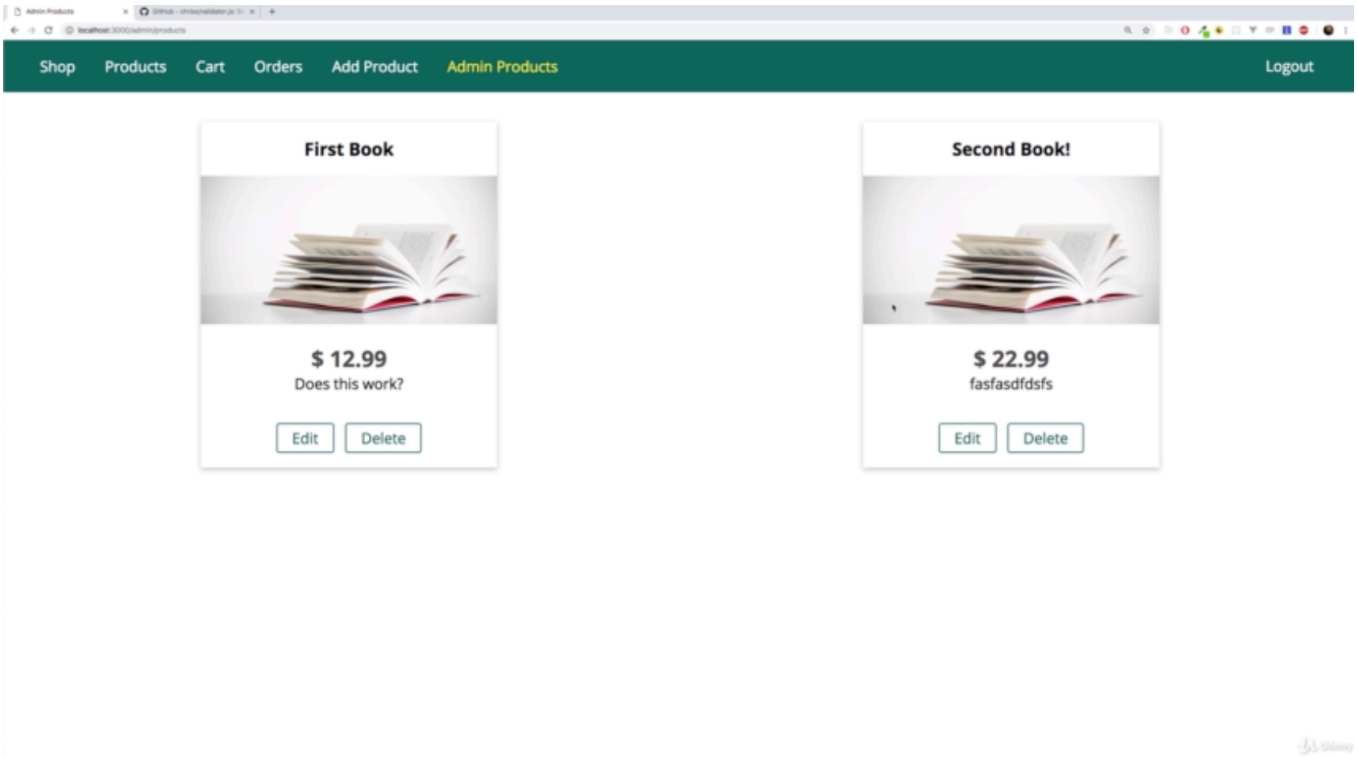
- but if i re-rendered it due to a validation error, so due to this render function in my if check here, then i only set title, imageUrl, price, description.

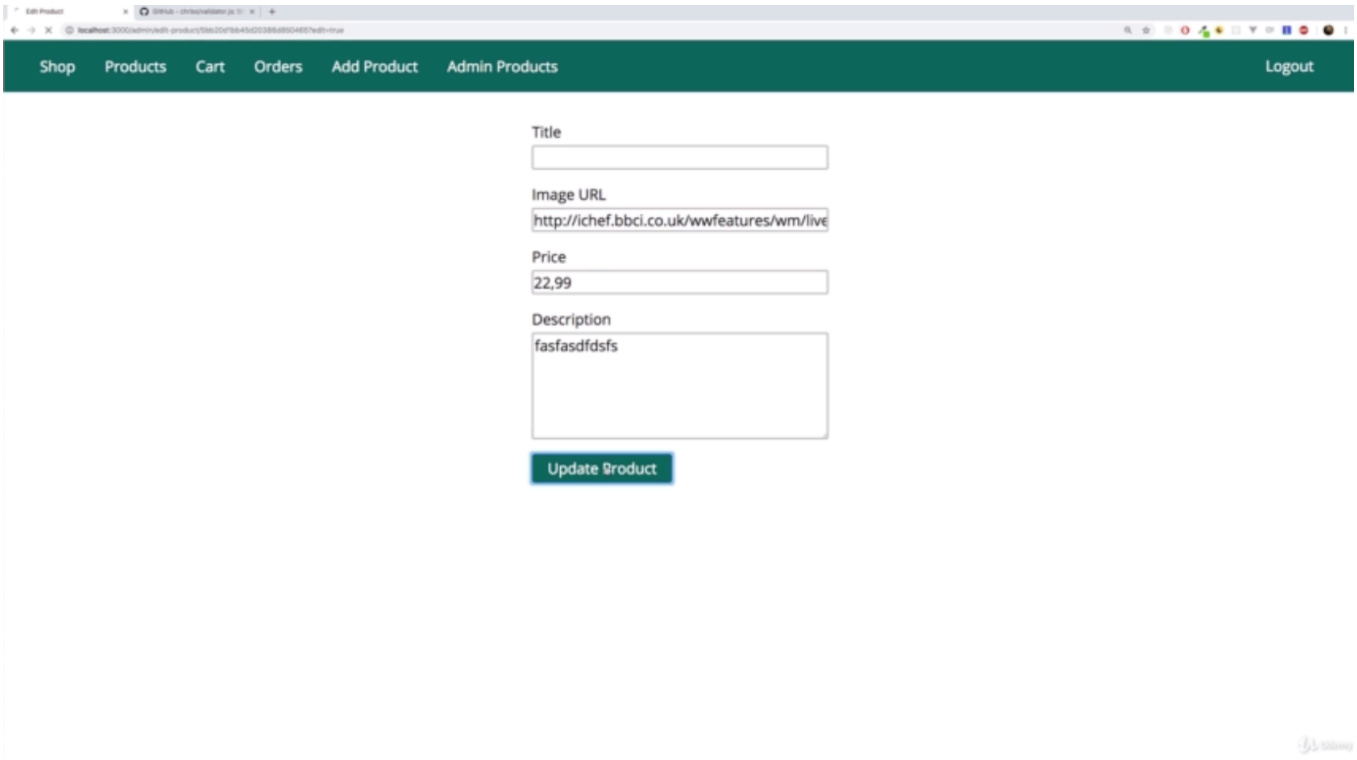


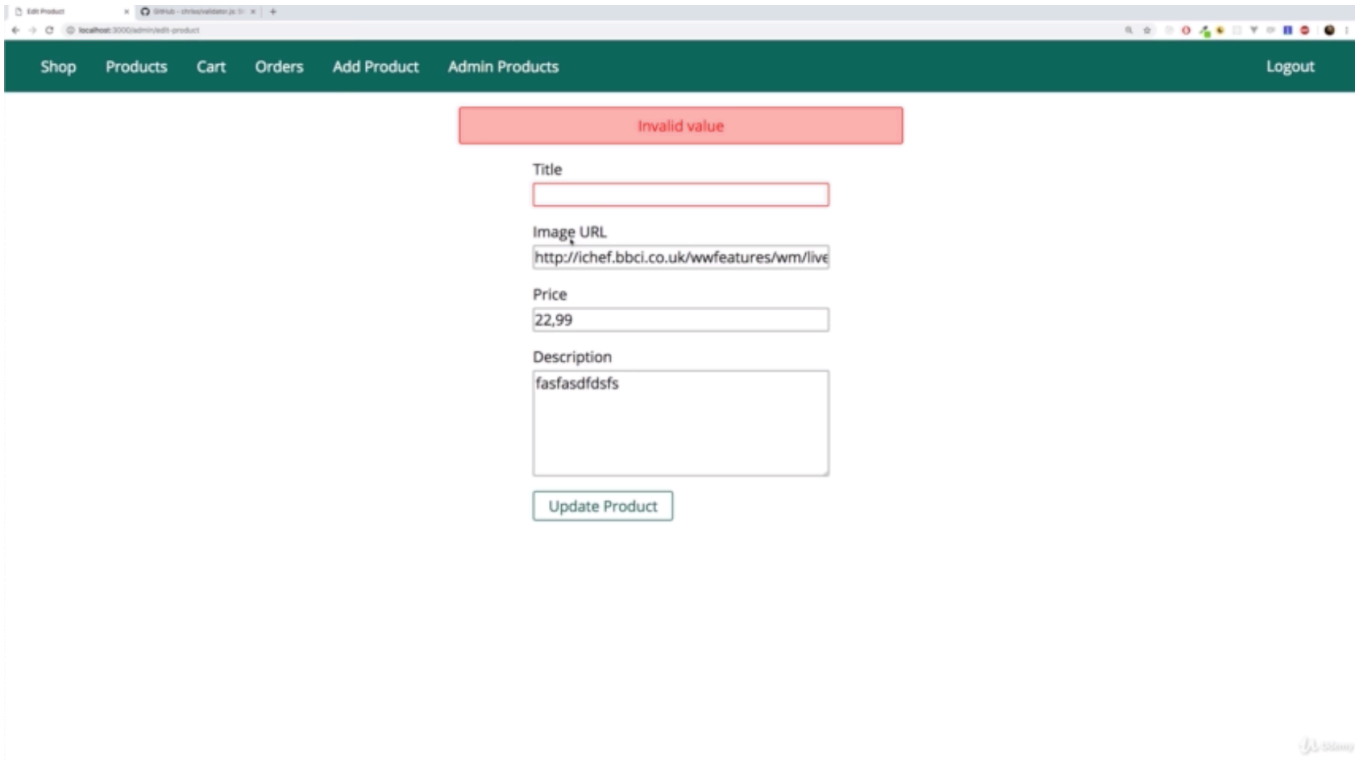


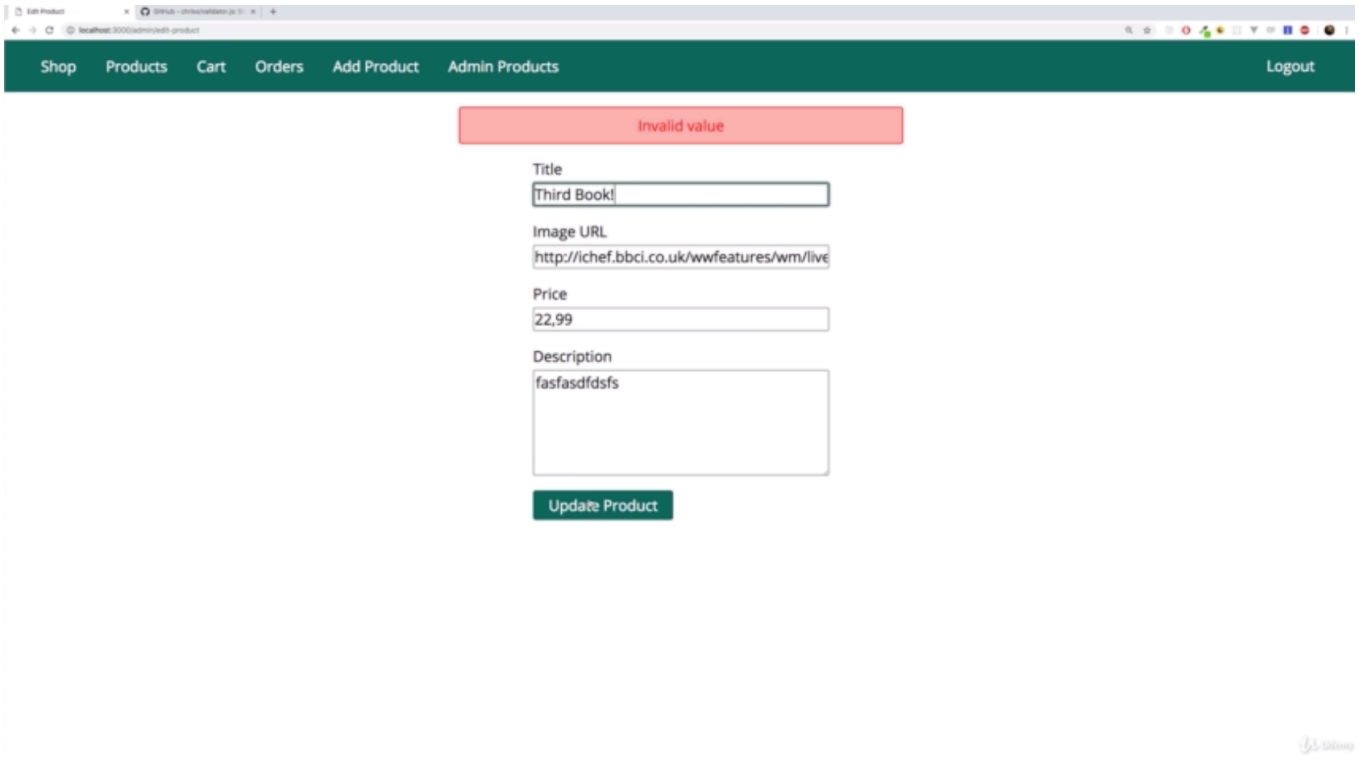
- i also need to set _id. otherwise it render the page, but if i try to submit it, that _id will be missing and i need that ID because i extract it here

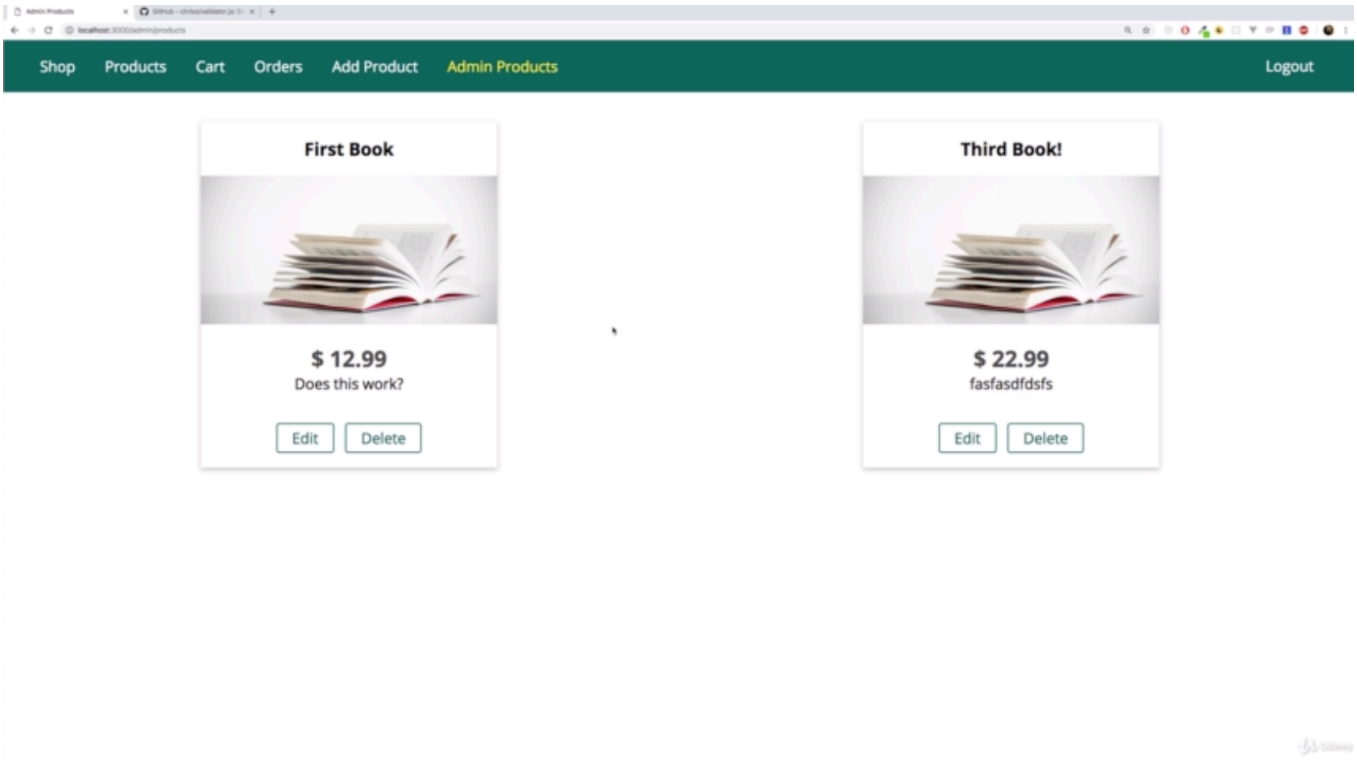



Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
First Book
$ 12.99
Does this work?
Edit
Delete
Second Book!
$ 22.99
fasfasdfdsfs
Edit
Delete


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Title
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
22,99
Description
fasfasdfdsfs
Update Product


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
22,99
Description
fasfasdfdsfs
Update Product


Shop
Products
Cart
Orders
Add Product
Admin Products
Logout
Invalid value
Title
Third Book!
Image URL
http://ichef.bbci.co.uk/wwfeatures/wm/live
Price
22,99
Description
fasfasdfdsfs
Update Product




```
// ./controllers/admin.js

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

const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
  res.render('admin/edit-product', {
    pageTitle: 'Add Product',
    path: '/admin/add-product',
    editing: false,
    hasError: false,
    errorMessage: null
  });
};

exports.postAddProduct = (req, res, next) => {
  const title = req.body.title;
  const imageUrl = req.body.imageUrl;
  const price = req.body.price;
  const description = req.body.description;
  const errors = validationResult(req)

  if(!errors.isEmpty()) {
    return res.status(422).render('admin/edit-product', {
      pageTitle: 'Add Product',
      path: '/admin/edit-product',
      editing: false,
      hasError: true,
      product: {
        title: title,
        imageUrl: imageUrl,
        price: price,
        description: description
      },
```

```
      errorMessage: errors.array()[0].msg
    })
  }

  const product = new Product({
    title: title,
    price: price,
    description: description,
    imageUrl: imageUrl,
    userId: req.user
  });
  product
    .save()
    .then(result => {
      // console.log(result);
      console.log('Created Product');
      res.redirect('/admin/products');
    })
    .catch(err => {
      console.log(err);
    });
};

exports.getEditProduct = (req, res, next) => {
  const editMode = req.query.edit;
  if (!editMode) {
    return res.redirect('/');
  }
  const prodId = req.params.productId;
  Product.findById(prodId)
    .then(product => {
      if (!product) {
        return res.redirect('/');
      }
      res.render('admin/edit-product', {
        pageTitle: 'Edit Product',
        path: '/admin/edit-product',
        editing: editMode,
        product: product,
        hasError: false,
        errorMessage: null,
        validationErrors: []
        });
    })
    .catch(err => console.log(err));
};

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;

  const errors = validationResult(req)

```

```
  if(!errors.isEmpty()) {
    return res.status(422).render('admin/edit-product', {
      pageTitle: 'Edit Product',
      path: '/admin/edit-product',
      editing: true,
      hasError: true,
      product: {
        title: updatedTitle,
        imageUrl: updatedImageUrl,
        price: updatedPrice,
        description: updatedDesc,
        _id: prodId
      },
      errorMessage: errors.array()[0].msg,
      validationErrors: errors.array()
    })
  }


  Product.findById(prodId)
    .then(product => {
      /**i should convert both to a string
       * because i'm also checking for type equality.
       */
      if(product.userId.toString() !== req.user._id.toString()){
        return res.redirect('/')
      }
      product.title = updatedTitle;
      product.price = updatedPrice;
      product.description = updatedDesc;
      product.imageUrl = updatedImageUrl;
      return product.save()
        .then(result => {
          console.log('UPDATED PRODUCT!');
          res.redirect('/admin/products');
        })
      })
    .catch(err => console.log(err));
};

exports.getProducts = (req, res, next) => {
  Product.find({userId: req.user._id})
    // .select('title price -_id')
    // .populate('userId', 'name')
    .then(products => {
      console.log(products);
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

exports.postDeleteProduct = (req, res, next) => {
```

```
  const prodId = req.body.productId;
  Product.deleteOne({_id: prodId, userId: req.user._id})
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};

```

```
<!--./views/admin/edit-product.ejs-->

<%- include('../includes/head.ejs') %>
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="stylesheet" href="/css/product.css">
</head>

<body>
   <%- include('../includes/navigation.ejs') %>

    <main>
        <% if (errorMessage) { %>
            <div class="user-message user-message--error"><%= errorMessage %></div>
        <% } %>
        <form class="product-form" action="/admin/<% if (editing) { %>edit-product<% } else { %>add-product<% } %>" method="POST">
            <div class="form-control">
                <label for="title">Title</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'title') ? 'invalid' : '' %>"
                    type="text"
                    name="title"
                    id="title"
                    value="<% if (editing || hasError) { %><%= product.title %><% } %>">
            </div>
            <div class="form-control">
                <label for="imageUrl">Image URL</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'imageUrl') ? 'invalid' : '' %>"
                    type="text"
                    name="imageUrl"
                    id="imageUrl"
                    value="<% if (editing || hasError) { %><%= product.imageUrl %><% } %>">
            </div>
            <div class="form-control">
                <label for="price">Price</label>
                <input
                    class="<%= validationErrors.find(e => e.param === 'price') ? 'invalid' : '' %>"
                    type="number"
                    name="price"
                    id="price"
                    step="0.01"
                    value="<% if (editing || hasError) { %><%= product.price %><% } %>">
            </div>
```

```
            <div class="form-control">
                <label for="description">Description</label>
                <textarea
                    class="<%= validationErrors.find(e => e.param === 'description') ? 'invalid' : '' %>"
                    name="description"
                    id="description"
                    rows="5"><% if (editing || hasError) { %><%= product.description %><% } %></textarea>
            </div>
            <% if (editing) { %>
                <input type="hidden" value="<%= product._id %>" name="productId">
            <% } %>

            <input type="hidden" name="_csrf" value="<%= csrfToken %>">
            <button class="btn" type="submit"><% if (editing) { %>Update Product<% } else { %>Add Product<% } %></button>
        </form>
    </main>
<%- include('../includes/end.ejs') %>
```

```
/*./public/css/forms.css*/

.form-control {
  margin: 1rem 0;
}

.form-control label,
.form-control input,
.form-control textarea {
  display: block;
  width: 100%;
  margin-bottom: 0.25rem;
}

.form-control input,
.form-control textarea {
  border: 1px solid #a1a1a1;
  font: inherit;
  border-radius: 2px;
}

.form-control input:focus,
.form-control textarea:focus {
  outline-color: #00695c;
}

.form-control input.invalid,
.form-control textarea.invalid {
  border-color: red;
}
```

```
// ./routes/admin.js

const path = require('path');

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

```
const { body } = require('express-validator/check')

const adminController = require('../controllers/admin');
const isAuth = require('../middleware/is-auth');

const router = express.Router();

// /admin/add-product => GET
router.get('/add-product', isAuth, adminController.getAddProduct);

// /admin/products => GET
router.get('/products', isAuth, adminController.getProducts);

// /admin/add-product => POST
router.post('/add-product', [
        body('title').isString().isLength({ min: 3 }).trim(),
        body('imageUrl').isURL(),
        body('price').isFloat(),
        body('description').isLength({ min: 5, max: 400 }).trim()
    ],
    isAuth,
    adminController.postAddProduct);

router.get('/edit-product/:productId', isAuth, adminController.getEditProduct);

router.post('/edit-product',[
        body('title').isString().isLength({ min: 3 }).trim(),
        body('imageUrl').isURL(),
        body('price').isFloat(),
        body('description').isLength({ min: 5, max: 400 }).trim()
    ],
    isAuth,
    adminController.postEditProduct);

router.post('/delete-product', isAuth, adminController.postDeleteProduct);

module.exports = router;
```