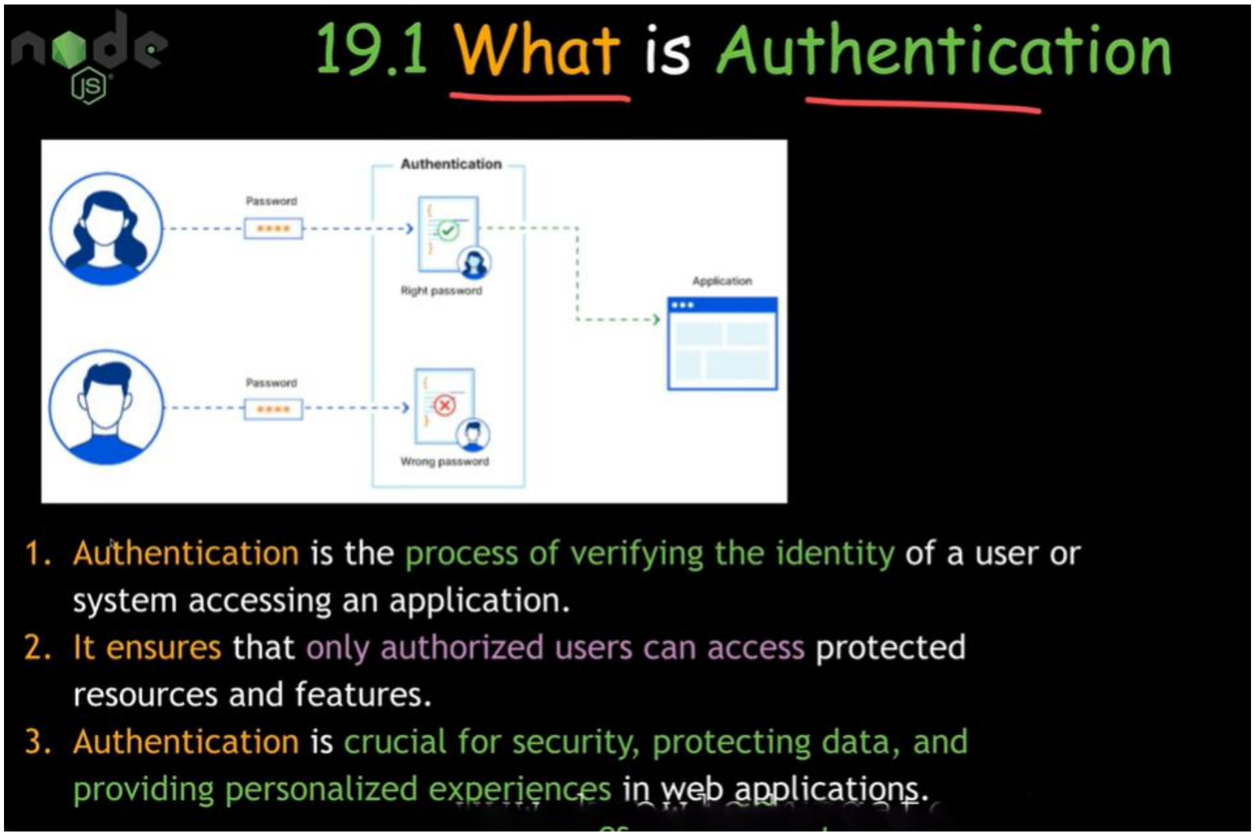
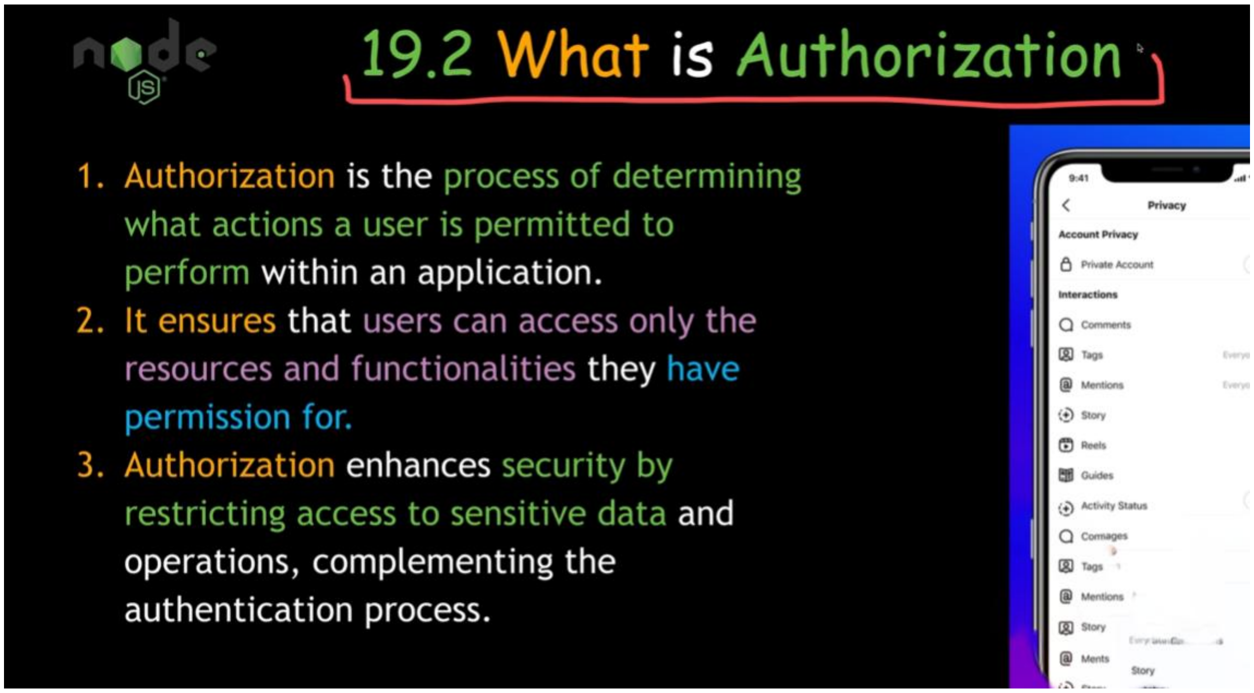
Authentication and Autherization::



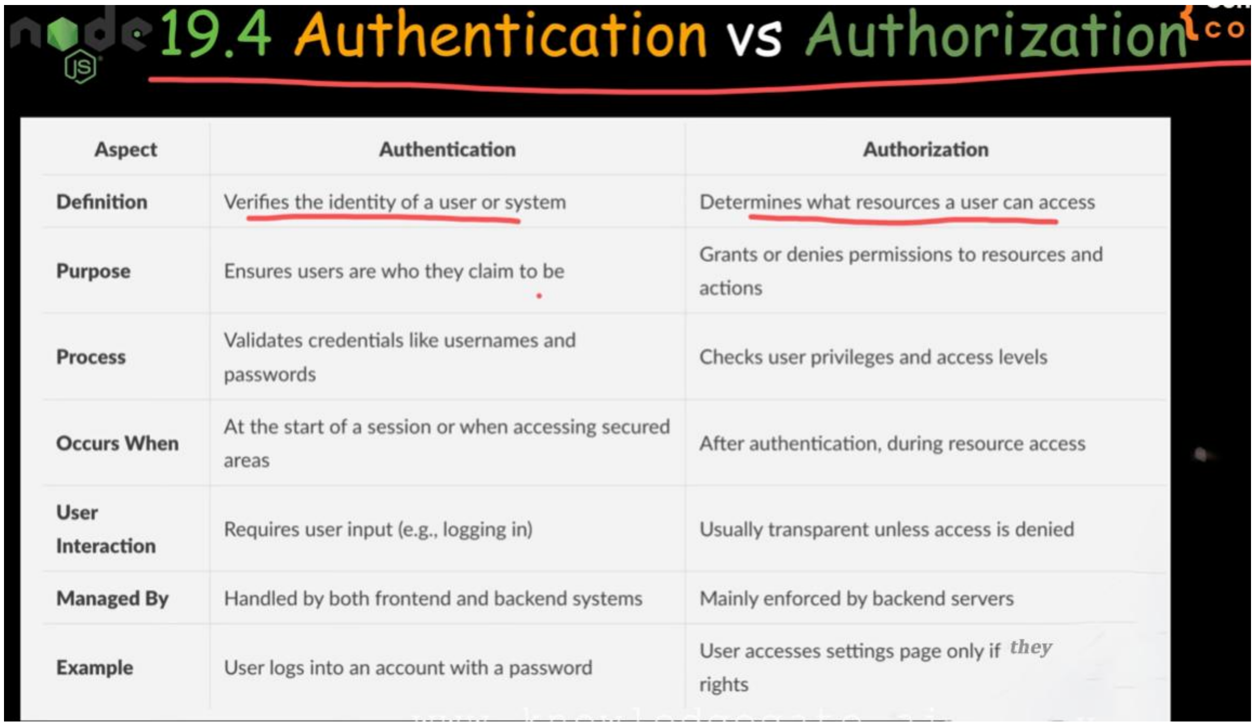
19.1 What is Authentication

1. Authentication is the process of verifying the identity of a user or system accessing an application.
2. It ensures that only authorized users can access protected resources and features.
3. Authentication is crucial for security, protecting data, and providing personalized experiences in web applications.



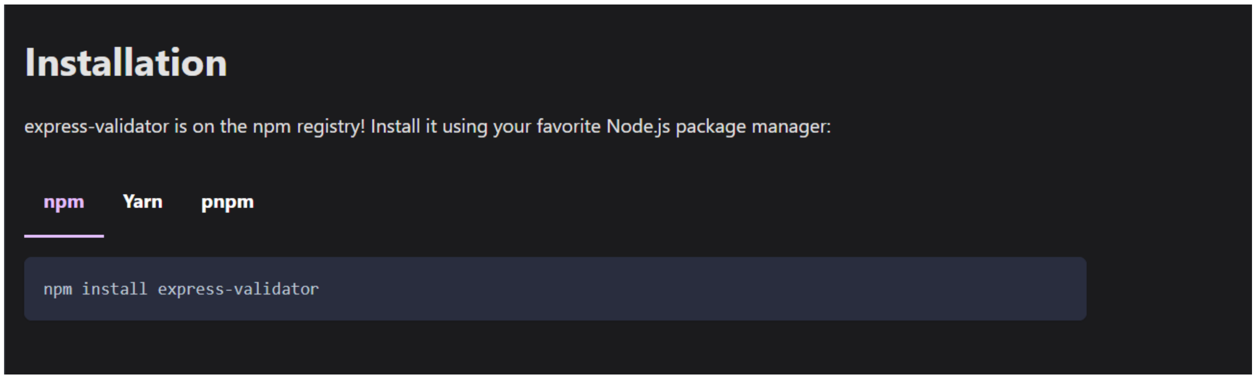
19.2 What is Authorization

1. Authorization is the process of determining what actions a user is permitted to perform within an application.
2. It ensures that users can access only the resources and functionalities they have permission for.
3. Authorization enhances security by restricting access to sensitive data and operations, complementing the authentication process.



19.4 Authentication vs Authorization

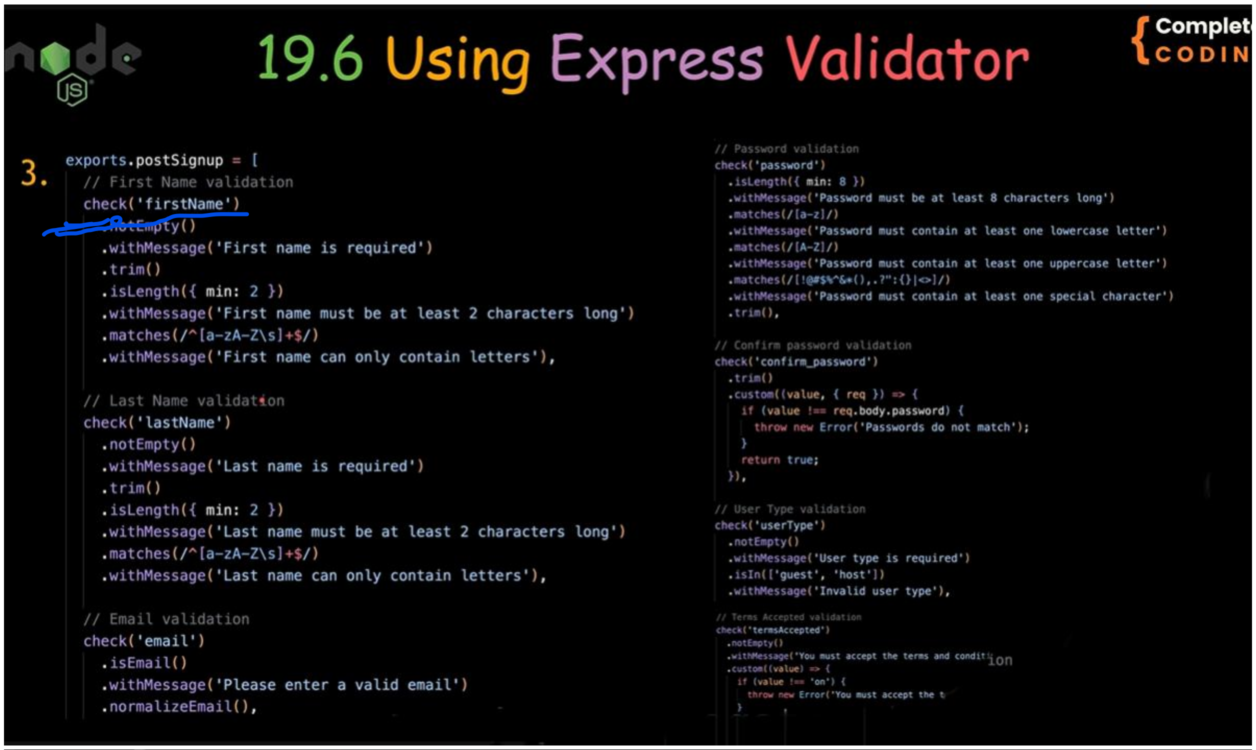
Aspect	Authentication	Authorization
Definition	Verifies the identity of a user or system	Determines what resources a user can access
Purpose	Ensures users are who they claim to be	Grants or denies permissions to resources and actions
Process	Validates credentials like usernames and passwords	Checks user privileges and access levels
Occurs When	At the start of a session or when accessing secured areas	After authentication, during resource access
User Interaction	Requires user input (e.g., logging in)	Usually transparent unless access is denied
Managed By	Handled by both frontend and backend systems	Mainly enforced by backend servers
Example	User logs into an account with a password	User accesses settings page only if they rights



Installation

express-validator is on the npm registry! Install it using your favorite Node.js package manager:

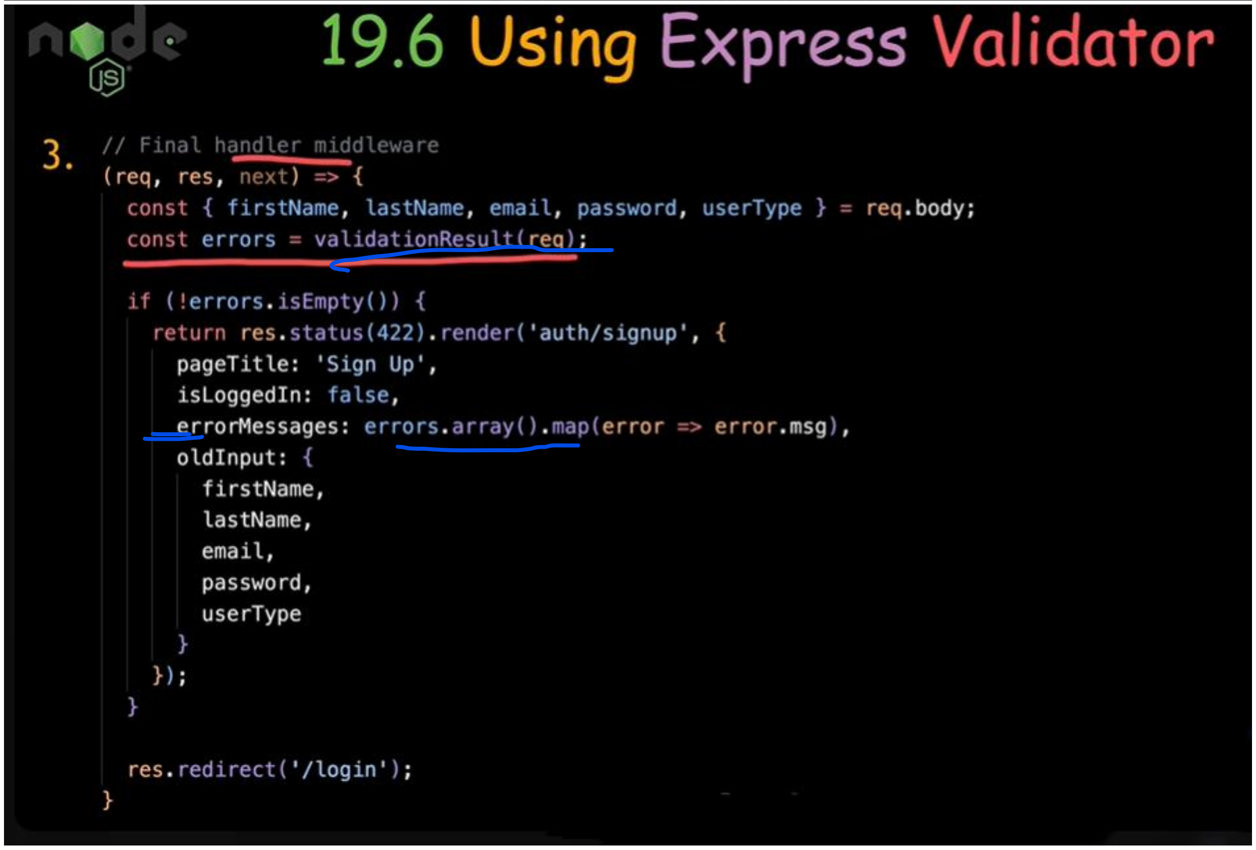
```
npm install express-validator
```



19.6 Using Express Validator

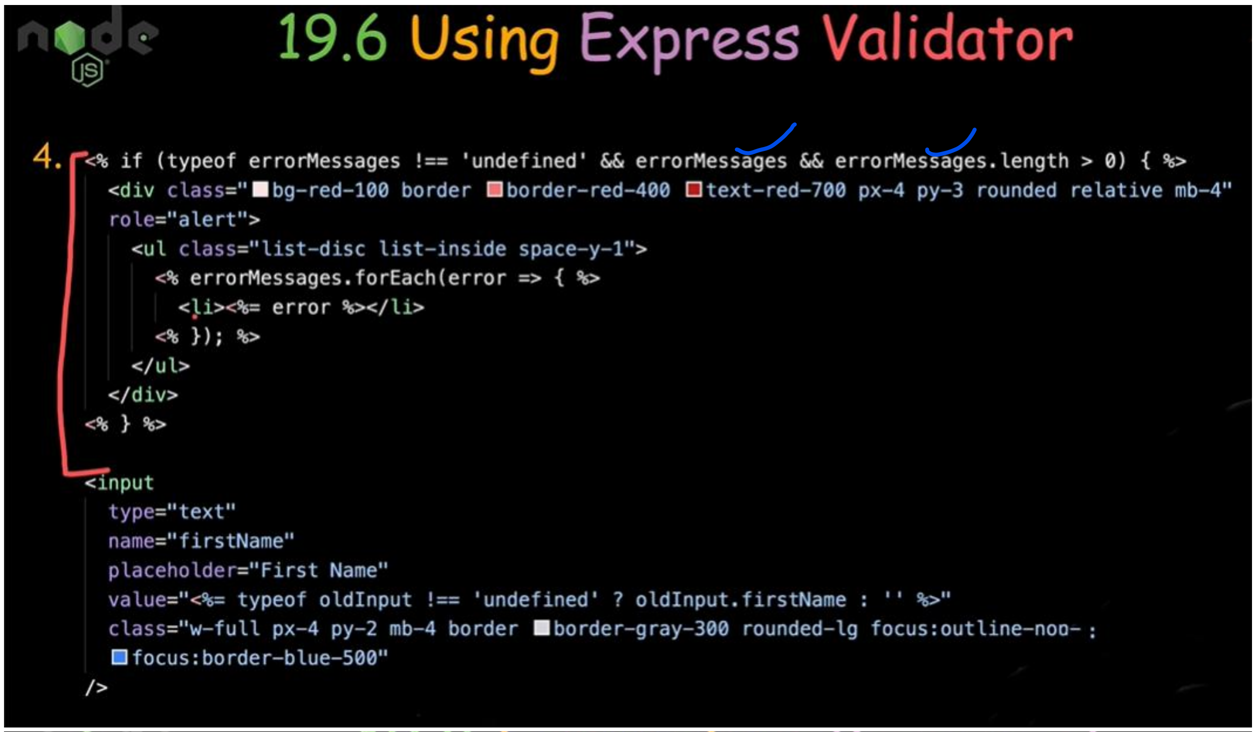
3.
```
exports.postSignup = [
  // First Name validation
  check('firstName')
    .notEmpty()
    .withMessage('First name is required')
    .trim()
    .isLength({ min: 2 })
    .withMessage('First name must be at least 2 characters long')
    .matches(/^[a-zA-Z\s]+$/)
    .withMessage('First name can only contain letters'),

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

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

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

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

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

// Terms Accepted validation
check('termsAccepted')
  .notEmpty()
  .withMessage('You must accept the terms and conditiion
  .custom((value) => {
    if (value !== 'on') {
      throw new Error('You must accept the t
    }
```



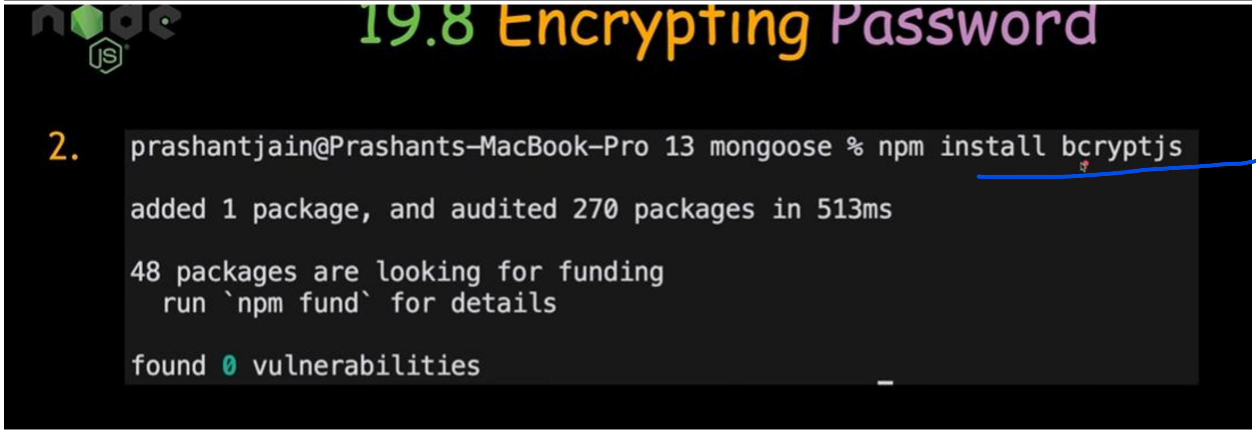
19.6 Using Express Validator

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

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

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



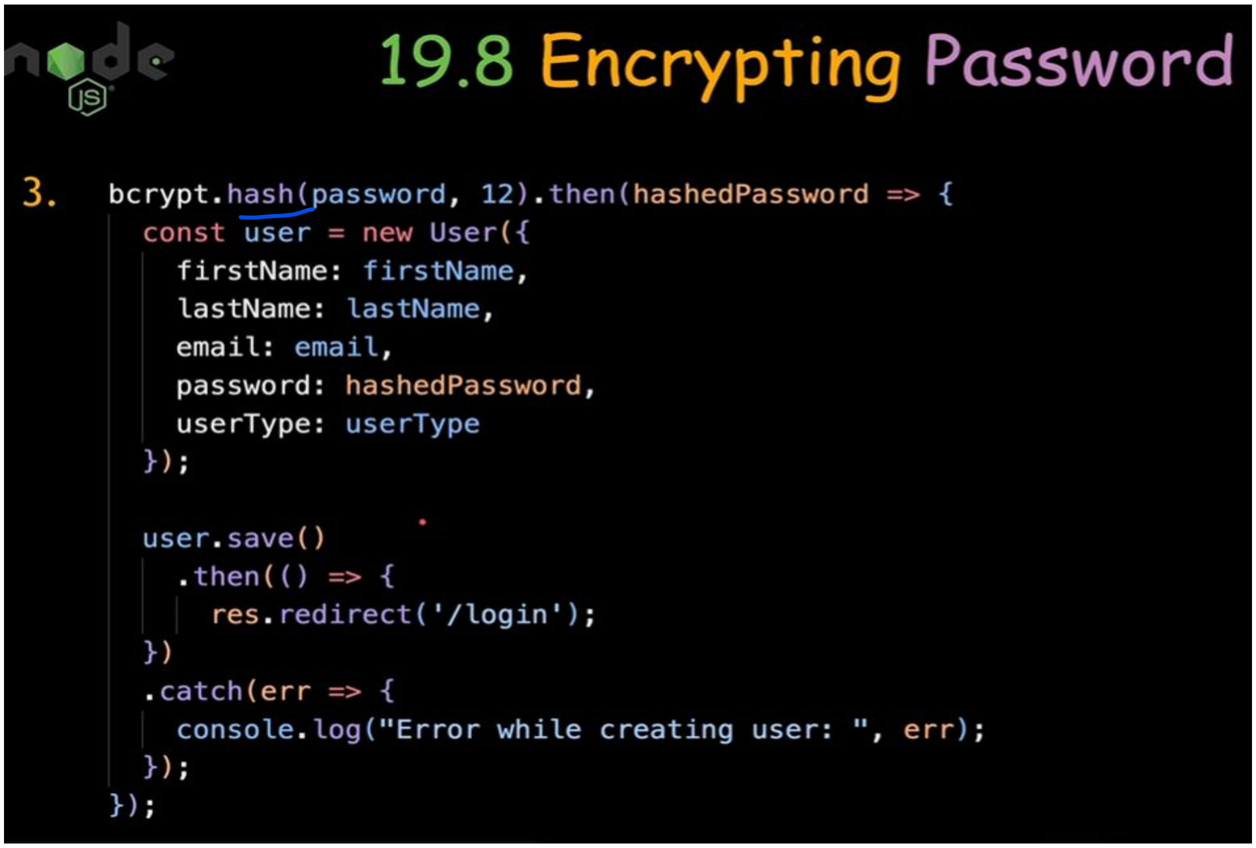
19.6 Using Express Validator

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

<input
  type="text"
  name="firstName"
  placeholder="First Name"
  value="<%= typeof oldInput !== 'undefined' ? oldInput.firstName : '' %>"
  class="w-full px-4 py-2 mb-4 border border-gray-300 rounded-lg focus:outline-non- :
  focus:border-blue-500"
/>
```



19.8 Encrypting Password

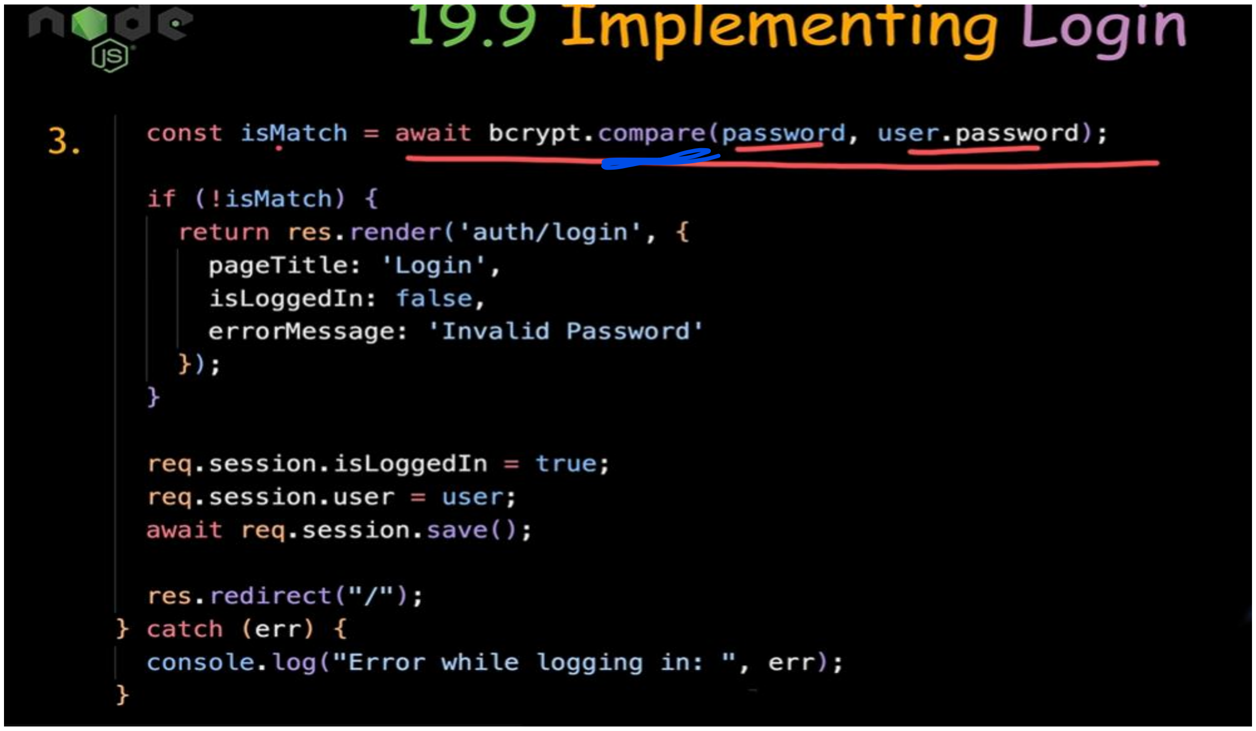
2.
```
prashantjain@Prashants-MacBook-Pro 13 mongoose % npm install bcryptjs

added 1 package, and audited 270 packages in 513ms

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

found 0 vulnerabilities
```



19.8 Encrypting Password

3.

```
bcrypt.hash(password, 12).then(hashedPassword => {
  const user = new User({
    firstName: firstName,
    lastName: lastName,
    email: email,
    password: hashedPassword,
    userType: userType
  });

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



19.9 Implementing Login

3.

```
    const isMatch = await bcrypt.compare(password, user.password);

    if (!isMatch) {
      return res.render('auth/login', {
        pageTitle: 'Login',
        isLoggedIn: false,
        errorMessage: 'Invalid Password'
      });
    }

    req.session.isLoggedIn = true;
    req.session.user = user;
    await req.session.save();

    res.redirect("/");
  } catch (err) {
    console.log("Error while logging in: ", err);
  }
```