**Lecture 250**

**What is Authentication?**

* Refer notes

**Lecture 251**

**How is authentication implemented?**

* Refer notes

**Lecture 254**

**Implementing an authentication**

* Refer code 01-implementing-an-authentication-flow
* Controllers/auth.js 🡪 postSignUp method
* Models/user.js 🡪 removed name and added password field
* App.js 🡪 removed code for adding new user when app starts up

**Lecture 255**

**Encrypting passwords**

* Refer code 02-encrypting-passwords
* Controllers/auth.js 🡪 postSignup method

**Lecture 257**

**Adding the Signin functionality**

* Refer code 04-adding-the-signin-functionality
* Controllers/auth.js 🡪 postLogin method

**Lecture 258**

**Working on route protection**

* Refer code 04-adding-the-signin-functionality
* Controllers/admin.js 🡪 getAddProduct method

**Lecture 259**

**Using middlewares to protect routes**

* Refer code 05-using-middleware-to-protect-routes
* Using the previous method to protect routes is not scalable, since we have to add that piece of code to every route.
* Middlwares/is-auth.js
* Routes/admin.js,shop.js 🡪 adding the auth middlware to the routes to be protected
* Route protection prevents the user from accessing the protected routes, when he try to access it by typing the url manually , or when he is logged in and then try to access the protected routes after deleting the session id cookie in the browser.

**Lecture 260**

**Understanding CSRF Attacks**

* Refer notes
* <https://portswigger.net/web-security/csrf>
* <https://owasp.org/www-community/attacks/csrf>
* <https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/>

**Lecture 261**

**Using a CSRF Token**

* Refer code 06 - using csrf token
* Npm install –save csurf
* App.js 🡪 regsistering the middlware
* Controllers/shop.js 🡪 getIndex method
* Navigation.ejs 🡪 adding hidden field for csrf token 🡪 name of field should be \_csrf
* The CSRF token will be sent along with all requests that are sensitive or that can change the state of a user. The server validates this token. The server can make sure that the requests are not coming from any fake pages , since they cannot get this token and send with the requests.
* It failed here because that logout action here actually was a post request and for any non-get requests because you change data via post requests typically, for any such requests, this package will look for the existence of a csrf token in your views, in the request basically in the request body. Now to make sure that such a token is there, we first of all need to ensure we have it available in our views, to do that we have to pass data into our view.

**Lecture 262**

**Adding CSRF protection**

* Refer code 06-adding-csrf-protection
* App.js 🡪 adding new middlware for setting isauthenticated and csrf token variables
* Added the hidden input field for csrf token to all .ejs files which contains forms
* Removed the isAuthenticated being passed manually to all views.
* Fixed the bug related to Order now button
  + When creating user we are storing only email and password , and not name.

Removing the name attribute and adding email attribute in Order model

* Using the same in postOrder method in controllers/shop.js
* So we added csrf token protection but it will only work if we are visiting our get index page. Now we want to have such a token and by the way also, our authentication status on every page we render. What we can do to get it onto every page is we can remove it from our render function here and instead tell expressjs and this is now totally unrelated from csrf thing, tell expressjs that we have some data that should be included in every rendered view.
* We'll do this in the app.js file and there after this middleware where we extract the user but before all our routes, I'll add another middleware, a normal middleware with our normal middleware function with the three arguments and in there, we can use a special feature provided by expressjs, we can access a special field on the response, the locals field. This allows us to set local variables that are passed into the views, local simply because well they will only exist in the views which are rendered.
* We have to add a hiddeninput field to every view to use the csrf token variable we have passed.

**Lecture 264 - 267**

**Providing user feedback**

* Refer code 08-providing-user-feedback
* Npm install –save connect-flash
* App.js 🡪 adding flash middlware
* Controllers/auth.js 🡪 postLogin method -> setting the error when the credentials are wrong –invalid email or password

getLogin, getSignup method 🡪 passing the error to the view

* Controllers/auth.js 🡪 postSignup method 🡪 setting the error when the user with given email already exists.
* Login.ejs ,signup.ejs🡪 displaying the error message if present.
* The main thing is that it's easy to pass data to our views inside of the render method as you can see, it was never a problem to get our data into the views there. However it is a huge problem if we want to pass some data into the rendered view when we are redirecting as we are doing it here because upon a redirect, technically a new request is started, a new request to /login in this case and on that new request, we don't know that we got here because the user entered an invalid e-mail or anything like that, when we triggered this new request this is treated in the same way as a request that was triggered by clicking on the login button in our menu. So we have no way of finding out if we want to display an error message or not and hence in the render method of get login where we show that login page, we don't know if we want to include some error message.
* Now to solve that problem and store some data before we redirect which we then use in the brand new request that is triggered by the redirect, how could we do that? Well you learned if you want to store data across requests, you need a session. So we can use a session for that but of course I don't want to store the error message in the session permanently, I want to add something to the error message, kind of flash it onto the session and once the error message was then used, so once we pulled it out of the session and did something with it, I want to remove it from the session so that for subsequent requests, this error message is not part of the session anymore and for this, we can use another package which makes this really easy. We install that with npm install --save and then the package is called connect

**Lecture 268**

**Wrap Up**

