For this home work use the latest rbp version you have. Please make a copy of server/index.js like:

```
cd server
cp index.js index.1.js
cd ..
```

I have already shown you in my videos how to use the csrf middleware to get a token and send it to the client as a header in the request handler for a head request. I also showed how to protect the signin page using the csrfProtection function (which was just a name I assigned to a function returned by calling csrf()). In the documentation for express-session, you can find a brief mention about a function called session.regenerate.

(1) I want you to implement the parameter function you are supposed to pass to session.regenerate. In that function a newly generated session would be available. If you had stored data previously in the session, all of that would be lost. So, you need to keep a copy of that session before you call session.regenerate. Inside, your parameter function, copy back any data you have stored in the previous session onto the new session. Also, call the req.csrfToken inside this parameter function and create a response header under the name X-CSRF-Token. When you call session.regenerate, express-session doesn't automatically send the connect.sid cookie to the browser. You have to manually do that inside your parameter function. In order to do that, check the JavaScript files of the express-session package to figure out how they do it and reimplement/adapt their code for your purpose(s). Print the csrfToken generated to the console.

(2) In HomePage/saga.js, after you receive the response for '/signin', print the csrfToken received from the server, and then send another get request to '/userhome' with the csrfToken you just received from the server in the response to the '/signin' request. In a real application you would do this request to '/userhome' in UserHome/saga.js after loading the UserHome component. We will do that in CSC440.

(3) Write a get handler for '/userhome' on your sever side code. Make sure you use csrfProtection as a middleware in the handler sequence. Print the csrfToken sent by the client. Send a suitable response to the client (can be html or json).

If you implemented everything correctly, then the csrfToken you print in all the three parts above would be the same and it would be different from the csrfToken printed in the get_csrf_token function.

Upload server/index.js and app/containers/HomePage/saga.js on canvas. I don't need the other files.

Congratulations! Now you have a working csrf-protected web application. You can add bells and whistles to it as you see fit going forward. Consider it as my graduation gift to you! Remember to add XSS protection when you allow users to provide contents such as blogs and comments. If you have a XSS vulnerable application then csrf protection means nothing.