Deleting and Updating Object in Database + Distributing Views

Introduction

In this lecture, we will be focussing on improvising what we already have and on the very important part of **CRUD,** that is, the **U** { **update** } and **D** { **delete** }.

Deleting a Post (Authorized)

- The first step is to learn how to delete a post. While deleting the post, we need to remember that there are comments associated with the posts therefore we need to delete all the comments as well.
- To delete the post we need to create an action. The same action will delete the comments associated with the post.
- We need a route that maps to that action.
- We need to place a link to the route that we created to map the actions.
- We will be making this format of the route- " **/posts/destroy/id** " where id will be the string params.
- Before deleting any post, we need to find whether that post id exists in the database or not.
-

- Additionally, before deleting any post, we need to make sure that the user who is deleting the post is the same one as who created the post.

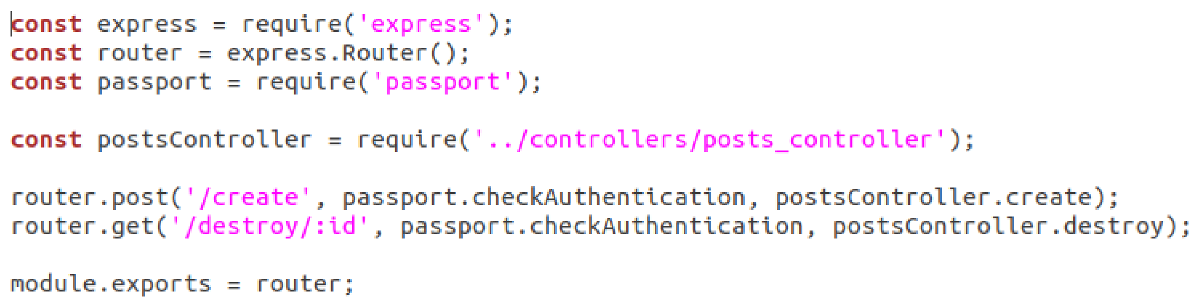
NOTE - When we are comparing the ids of two objects we need to convert them into a string. Using {.id } means we are converting the object id into a string.

- If the current user and post user are the same then, simply remove the post associated with the current user along with the comments that are associated with that post.
- If the current user and post user don't match, then redirect to the same page on which the user was earlier.

{ **posts_controllers.js }**

```
const Post = require('../models/post');
const Comment = require('../models/comment');

module.exports.create = function(req, res){
    Post.create({
        content: req.body.content,
        user: req.user._id
    }, function(err, post){
        if(err){console.log('error in creating a post'); return;}

        return res.redirect('back');
    });
}


module.exports.destroy = function(req, res){
    Post.findById(req.params.id, function(err, post){
        // .id means converting the object id into string
        if (post.user == req.user.id){
            post.remove();

            Comment.deleteMany({post: req.params.id}, function(err){
                return res.redirect('back');
            });
        }else{
            return res.redirect('back');
        }

    });
}
```

{ **posts.js** }



```
const express = require('express');
const router = express.Router();
const passport = require('passport');

const postsController = require('../controllers/posts_controller');

router.post('/create', passport.checkAuthentication, postsController.create);
router.get('/destroy/:id', passport.checkAuthentication, postsController.destroy);

module.exports = router;
```

- Create the button for deletion, and the button should be visible only when the user is logged in and If the current user and post user are the same.

Authorization on different levels -

- At the view level - show the button only if the user is authorized to.
- At the Router level - Allow the user to be able to send the request only when the user is logged in.
- At the action level inside the controller - Allow the post to be deleted only if the post and the user that is sending the request are the same.

Deleting a Comment (Authorized)

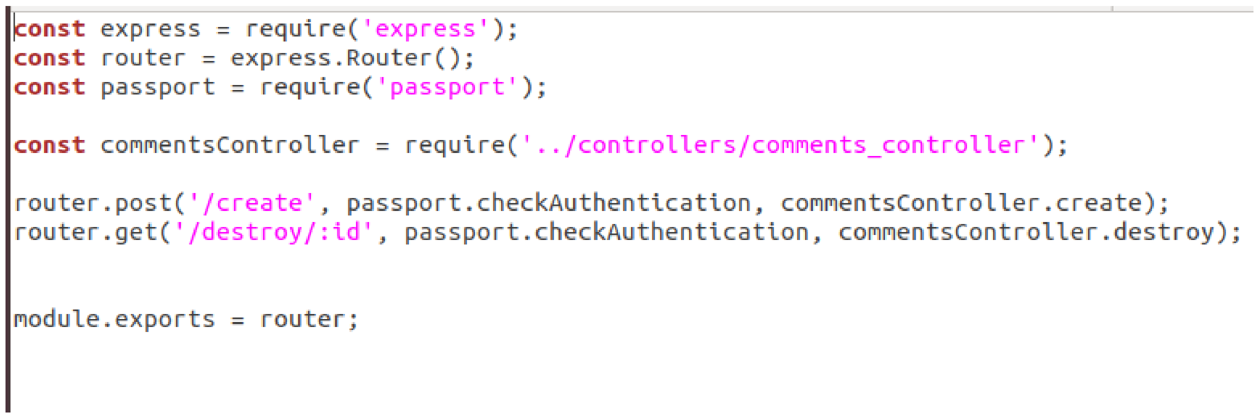
- Deleting comments will be a little different from deleting posts because comments can be done on another user's posts also or on the user itself post.
- We will create an action, a route, and a delete button with authorization and authentication checks.
- We need to find whether the comment we want to delete exists in the database. If it exists, we need the id of the comment.
- Before deleting a comment, we need to fetch the post id of that comment we want to delete.

{ **comments_controller.js** }

```
    Post.findById(req.body.post, function(err, post){

        if (post){
            Comment.create({
                content: req.body.content,
                post: req.body.post,
                user: req.user._id
            }, function(err, comment){
                // handle error

                post.comments.push(comment);
                post.save();

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

    });
```

```
odule.exports.destroy = function(req, res){
    Comment.findById(req.params.id, function(err, comment){
        if (comment.user == req.user.id){

            let postId = comment.post;

            comment.remove();

            Post.findByIdAndUpdate(postId, { $pull: {comments: req.params.id}}, function(err, post){
                return res.redirect('back');
            })
        }else{
            return res.redirect('back');
        }
    });
```

{ **comments.js**}



```
const express = require('express');
const router = express.Router();
const passport = require('passport');

const commentsController = require('../controllers/comments_controller');

router.post('/create', passport.checkAuthentication, commentsController.create);
router.get('/destroy/:id', passport.checkAuthentication, commentsController.destroy);


module.exports = router;
```

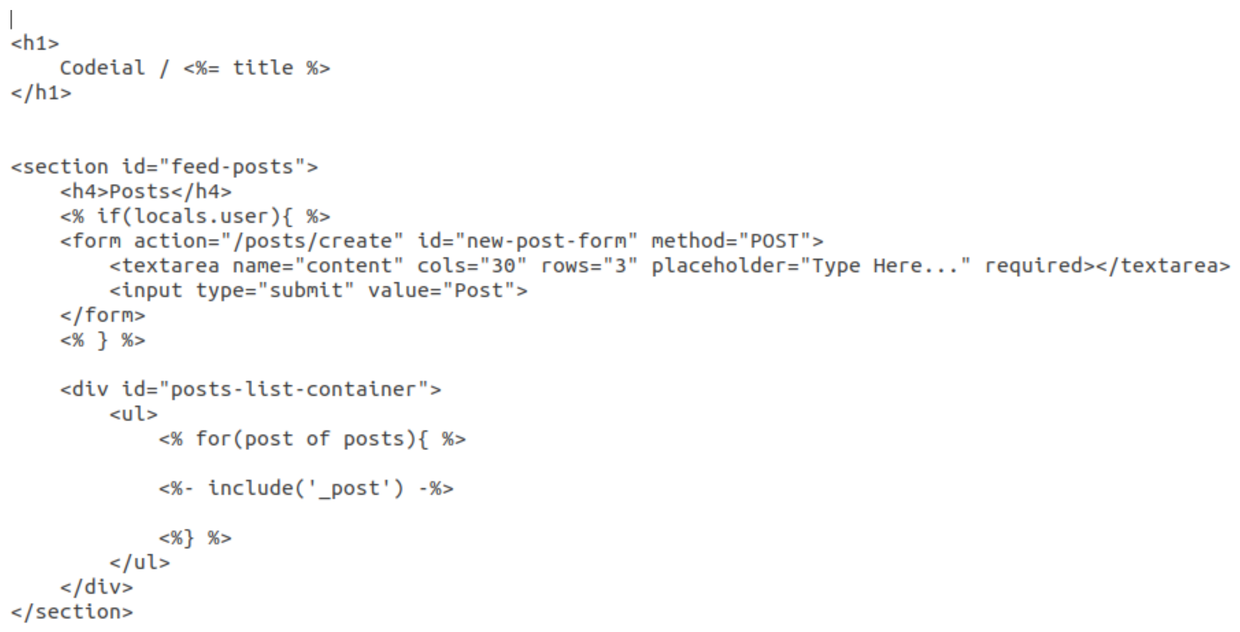
- In { **comments_controller.js**} we have put the check of user-id matching, in the routes { **comments.js**} we have put the check if the user is logged in.
- Create the delete button and it will be only visible when the user id matches with the local user-id and the user are signed in.

EXTRA - You can add another level of authentication - If the comments posts user-id match the current user-id the user should be able to delete the comment which is there on the user's posts.

Distributing the Code into Partials

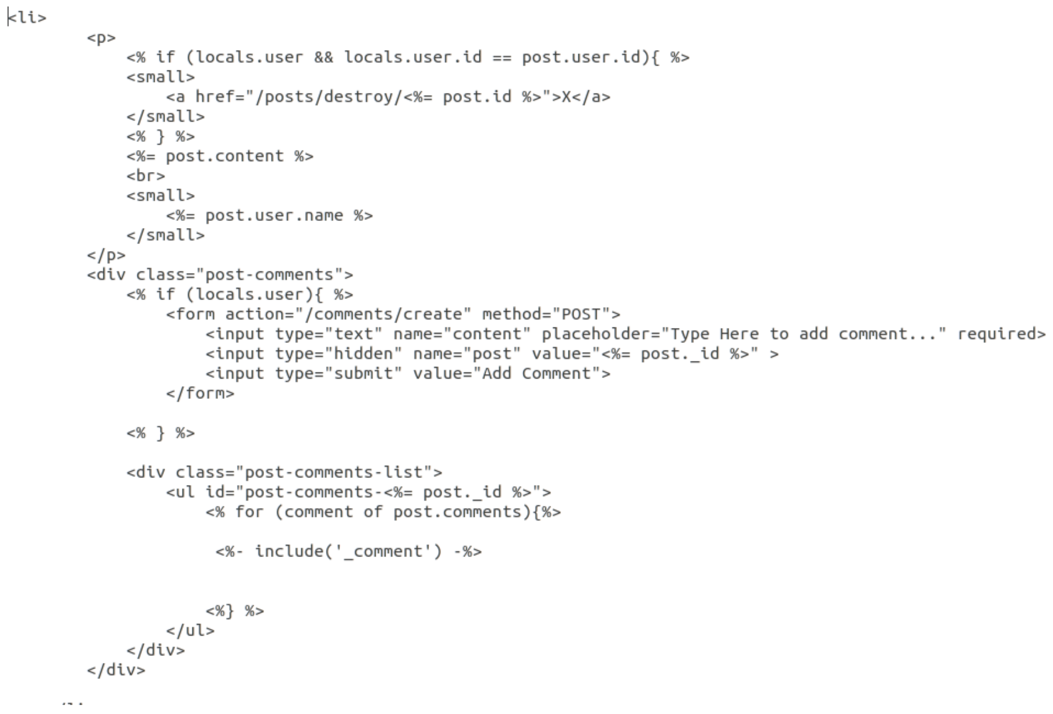
- As the code base grows the number of lines in a file also grows. To solve this problem, we will be wisely distributing the larger files into smaller files by creating partials.
- The largest file in this project is **{home. ejs}.** Hence, we will be using partials to distribute the code of **{home. ejs}** into multiple new files **{_post.ejs } & {_comment.ejs}.**

{home.ejs}



```
|
<h1>
    Codeial / <%= title %>
</h1>


<section id="feed-posts">
    <h4>Posts</h4>
    <% if(locals.user){ %>
    <form action="/posts/create" id="new-post-form" method="POST">
        <textarea name="content" cols="30" rows="3" placeholder="Type Here..." required></textarea>
        <input type="submit" value="Post">
    </form>
    <% } %>

    <div id="posts-list-container">
        <ul>
            <% for(post of posts){ %>

            <%- include('_post') -%>

            <%} %>
        </ul>
    </div>
</section>
```

{_post.ejs}



```
<li>
        <p>
            <% if (locals.user && locals.user.id == post.user.id){ %>
            <small>
                <a href="/posts/destroy/<%= post.id %>">X</a>
            </small>
            <% } %>
            <%= post.content %>
            <br>
            <small>
                <%= post.user.name %>
            </small>
        </p>
        <div class="post-comments">
            <% if (locals.user){ %>
                <form action="/comments/create" method="POST">
                    <input type="text" name="content" placeholder="Type Here to add comment..." required>
                    <input type="hidden" name="post" value="<%= post._id %>" >
                    <input type="submit" value="Add Comment">
                </form>

            <% } %>

            <div class="post-comments-list">
                <ul id="post-comments-<%= post._id %>">
                    <% for (comment of post.comments){%>

                     <%- include('_comment') -%>


                    <%} %>
                </ul>
            </div>
        </div>
```

User Profile links

- Now moving towards the update part of { **CRUD** }, we want the user to update his/her profile.
- For that, we have to first display the list of all the users on the home page **{home. ejs}** only if the user is signed in.
- To show all the users, we need to get all the lists of users from the controller **{ home_controller.js}.**

- After getting the list of all the users, we have to change the routes, as we have changed the routes of the profile.
- As the route is changed, the actions also need to be changed.

Updating a User's profile

- We have to first display the form for update when the user is viewing his/her profile page in **{users_profile.ejs}**.
- If the user's id matches with the profile page user-id, then show the form. If the user doesn't match, then we will show only the profile information.
- We have to create a form in **{users_profile.ejs}** where the action will go to { **users/update }**
- The form will be mapped to a route which will be mapped to action inside the controller.
- We have to create a route and action to be able to submit the form.
- We will create the action first and then map it to the routes.
- For creating an action, we have to call the **module.exports.update** in **{users_controller.js}** and check the update request { if the currently logged-in user id is equal to requesting user-id } then only update is allowed.
- If the id of the current user and requesting user-id don't match, then return the HTTP status code {**401**} for the unauthorized user.
- We will create a route inside {**users.js**} which is mapped to the action we have created.

EXTRA - For further read, you can refer to the link below.

https://mongoosejs.com/docs/documents.html

Improving SCSS of the home page

- We have to improve the design of our website. For that, we will create an SCSS file for the home page and we will align the feed section and the user section side by side using flex.
- Create a new file inside the assets folder {**home.scss**} and link this file with **{home.ejs}.**
- Create the styling page as your assignment using flex property.

Summarizing

- We began by deleting the post and the associated comments with it.
- We found the posts and the comments which were there in the posts and deleted them.
- We distributed the home page code into multiple partials.
- We created the user profile pages to be editable if the same user is logged in the user can be enabled to edit the profile page and if the other user is logged in then that user will only be able to view the profile.
- We beautify the home page by converting it into flex.