Database Relations (Posts, Comments)

Introduction

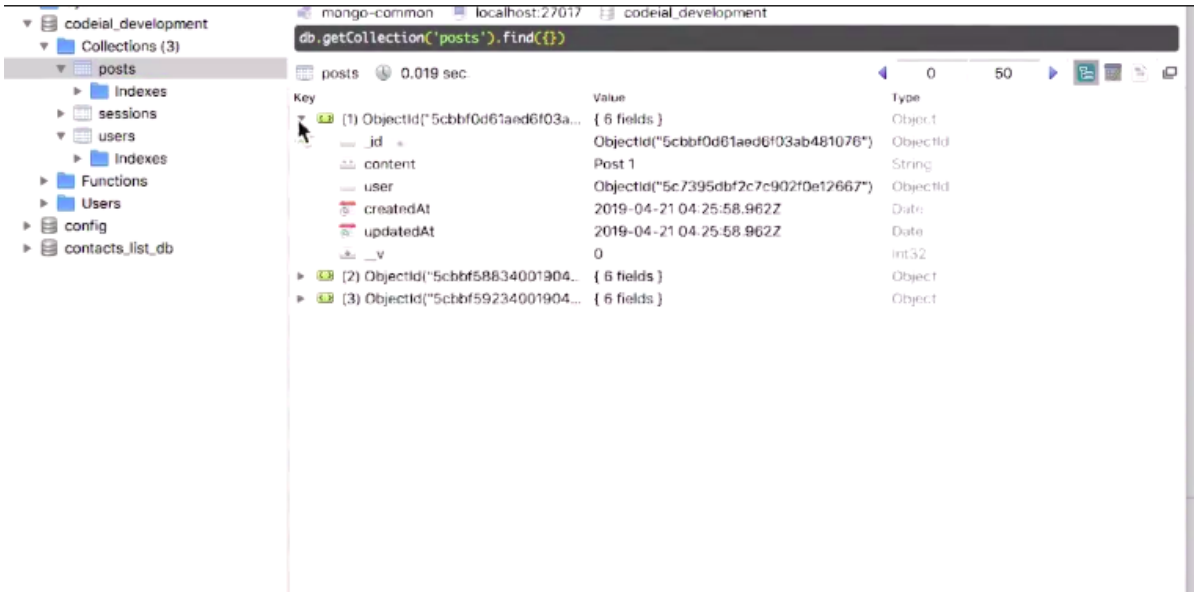
We will create posts and comments in the application that we are building. But before that, we need to understand how all of these can be added to a database.

Node.js:: Explaining 1:1 and 1:M

- Whenever we are storing any data inside the database, we are storing it in such a way that it models some part of the real world.
- Let us take an example of the user and a password. Usually, a single user can have one password. Hence, this relationship will always be a **1:1** relationship.
- There is an eg of school and students. Usually, one school consists of multiple students so this represents a **1:M** relationship or vice-versa **M:1**.
- We have another relationship called **M: M** which can be represented using the example of authors and books { One author can have many books and one book can have many authors }.
- The relationship between posts and comments will have a **1:M** relationship.

Creating Schema for Posts

- We need to create a schema for posts and link it to the users. Whenever a post is posted on a website it has to be coming from a user, someone who is logged in. Thus, a post needs to have a user in it.
- We need to create a file inside the models folder **{ post.js }**.
- We need to import mongoose inside the file **{ post.js }**.
- For creating a schema we need to use **mongoose.Schema** method, which will contain multiple fields.
- Timestamps automatically introduce two fields created at and updated at.



{ post.js }

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


const postSchema = new mongoose.Schema({
    content: {
        type: String,
        required: true
    },
    user: {
        type:  mongoose.Schema.Types.ObjectId,
        ref: 'User'

    }
},{
    timestamps: true
});

const Post = mongoose.model('Post', postSchema);
module.exports = Post;
```

- We need to go to the views and create a form from where this collection will have a document in it { i.e an entry is created in the database }.

{ views/home.ejs }

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


<section id="feed-posts">
    <h4>Posts</h4>
    <form action="" id="new-post-form" method="POST">
        <textarea name="content" cols="30" rows="3" placeholder="Type Here..."></textarea>
        <input type="submit" value="Post">
    </form>
</section>
```

Saving Posts to the DB

- We need to create an action inside the controller to save the data that is coming from the form into the database. We also and a route for mapping the form to the action.

- We need to create a new file inside the controllers folder { **posts_controller.js** } for the actions.
- We need to import the post schema that we have created inside the models folder.

{ **posts_controller.js** }

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

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');
    });
}
```

- We need to create a new file inside the routes folder { **posts.js** } for mapping the form to the action.

{ **posts.js** }

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


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

router.post('/create', postsController.create);


module.exports = router;
```

- We have created the router and we will call it from { index.js } file to make it usable.

{ **routes/index.js** }

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

const router = express.Router();
const homeController = require('../controllers/home_controller');

console.log('router loaded');


router.get('/', homeController.home);
router.use('/users', require('./users'));
router.use('/posts', require('./posts'));

// for any further routes, access from here
// router.use('/routerName', require('./routerfile'));


module.exports = router;
```

- We need to update this in the action of the form.

{home.ejs }

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


<section id="feed-posts">
    <h4>Posts</h4>
    <form action="/posts/create" id="new-post-form" method="POST">
        <textarea name="content" cols="30" rows="3" placeholder="Type Here..."></textarea>
        <input type="submit" value="Post">
    </form>
</section>
```

Display Post and Related User

- We will be showing the post on the home page.
- We need to find all the posts belonging to a user, then show them inside the home views.
- We need to show the name of the user who is posting the posts. For that, we need to prepopulate the user.

- Whenever we are loading the post from the database, we will load the whole user and show the desired data of the user on the home page. This is called pre-populating the user.
- Initially, the **id** of the user is only showing on the home page but now we need to fetch the user also from the database.

EXTRA:
You can check out the link below to understand more about populate -
https://mongoosejs.com/docs/populate.html

{ home_controller.js }

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

module.exports.home = function(req, res){
    // console.log(req.cookies);
    // res.cookie('user_id', 25);

    // Post.find({}, function(err, posts){
    //     return res.render('home', {
    //         title: "Codeial | Home",
    //         posts:  posts
    //     });
    // });

    // populate the user of each post
    Post.find({}).populate('user').exec(function(err, posts){
        return res.render('home', {
            title: "Codeial | Home",
            posts:  posts
        });
    })

}

// module.exports.actionName = function(req, res){}
```

{ home.ejs }

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


<section id="feed-posts">
    <h4>Posts</h4>
    <form action="/posts/create" id="new-post-form" method="POST">
        <textarea name="content" cols="30" rows="3" placeholder="Type Here..."></textarea>
        <input type="submit" value="Post">
    </form>

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

            <li>
                <p>
                    <%= post.content %>
                    <br>
                    <small>
                        <%= post.user.name %>
                    </small>
                </p>

            </li>

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

Check Authentication on Creating Post

- We need to establish the user identity by restricting the form to be visible to the part only when the user is signed in.

{ home.ejs }

```
<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..."></textarea>
        <input type="submit" value="Post">
    </form>
    <% } %>

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

            <li>
                <p>
                    <%= post.content %>
                    <br>
                    <small>
                        <%= post.user.name %>
                    </small>
                </p>

            </li>

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

- We will put a check on the action level so that no one other than the user who is signed in will be able to write a post.

{ 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);


module.exports = router;
```

Creating Schema for Comments

- There can be a user that can have multiple posts and each post can have multiple comments on it. A case where a post has no comments is also possible.
- Inside the schema of the post, there is a reference to the user.

- Inside the comments, there will be two references of users and posts.
- A post can have an array of comments. Although, instead of using an array, we will just store an array of different objects, each object representing a comment.
- We will create a schema for comments inside the models folder by creating a new file **{ comment.js }.**

  //Repeated steps that are similar to posts schema

  { comment.js }

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


const commentSchema = new mongoose.Schema({
    content: {
        type: String,
        required: true
    },
    // comment belongs to a user
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
    post: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Post'
    }
},{
    timestamps: true
});

const Comment = mongoose.model('Comment', commentSchema);
module.exports = Comment;
```

- Whenever we are loading a post, we need to find out all the comments inside that post so that we include the ids of all the comments in the array inside the post schema.

{ posts.js }

```
const postSchema = new mongoose.Schema({
    content: {
        type: String,
        required: true
    },
    user: {
        type:  mongoose.Schema.Types.ObjectId,
        ref: 'User'

    },
    // include the array of ids of all comments in this post schema itself
    comments: [
        {
            type:  mongoose.Schema.Types.ObjectId,
            ref: 'comment'
        }
    ]
},{
    timestamps: true
});
```

Adding Comments to the DB

- We will create a form for comments, then create an action for comments, and then a route for it.
- We need to show the form only when the user is logged in.
- The method of the form will be **POST,** as we are going to send the data.
- We need to send the **id** of the post to which comments need to be added.

{ home.ejs }

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

        <li>
            <p>
                <%= 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...">
                        <input type="hidden" name="post" value="<%= post._id %>" >
                        <input type="submit" value="Add Comment">
                    </form>

                <% } %>
            </div>

        </li>

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

- We need to create the controller for the comments inside the controller folder **{ comments_controller.js }.**
- We need to create a comment on the post. But before that, we need to find whether that post exists or not.

{ comments_controller.js }

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

module.exports.create = function(req, res){
    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('/');
            });
        }

    });
}
```

- We need to create the routes for the comments inside the routes folder.

{ comments.js }

{ 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);


module.exports = router;
```

- We have created the router and we need to call it from { index.js } file to make it usable.

{ routes/index.js }

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

const router = express.Router();
const homeController = require('../controllers/home_controller');

console.log('router loaded');


router.get('/', homeController.home);
router.use('/users', require('./users'));
router.use('/posts', require('./posts'));
router.use('/comments', require('./comments'));

// for any further routes, access from here
// router.use('/routerName', require('./routerfile));


module.exports = router;
```

Nesting Population:: Display Comments and Related User

- We need to show comments alongside the post and also the author of the comments.
- We will preload the comments and then display them.
- We need to preload multiple comments and the user of that comment.

{home_controller.js }

```
module.exports.home = function(req, res){
    // console.log(req.cookies);
    // res.cookie('user_id', 25);

    // Post.find({}, function(err, posts){
    //     return res.render('home', {
    //         title: "Codeial | Home",
    //         posts:  posts
    //     });
    // });

    // populate the user of each post
    Post.find({})
    .populate('user')
    .populate({
        path: 'comments',
        populate: {
            path: 'user'
        }
    })
    .exec(function(err, posts){
        return res.render('home', {
            title: "Codeial | Home",
            posts:  posts
        });
    })

}
```

- We need to show the comments on the home page.

{ home.ejs }

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

        <p>
            <%= comment.content %>
            <br>
            <small>
                <%= comment.user.name %>
            </small>
        </p>
        <%} %>
    </ul>
</div>
```

Summarizing

- We have created three many to one relationships {the user has many posts, the user has many comments and posts has many comments }.
- We stored an array of the comment **ids** inside the post to make the traversing fast.