Friends + Likes

Introduction:: New Relationships :: In Database

We have learned three types of relationships - one to one, one to many, and many to many {posts and comments }.

We will be focussing upon the polymorphic relationship { likes }, many to many relationships implemented via join, and self-referential many to many relationships.

Polymorphic Relations

- Poly means multiple forms. In polymorphic relations, there can be multiple types of parents.
- We will be implementing **likes** functionality using polymorphic relations.
- The table for likes will comprise fields that are { user, parent type, parent id }.
- In the case of a no-SQL database { MongoDB }, initially likes will have a **user** field with the reference.
- Then there would be a field named as a **parent,** which will have reference type and reference id inside of it.

***EXTRA*: Assignment**
Create a button to toggle like and a button that has a count of likes.

Schema Setup:: Likes

- Mongoose gives the feature of dynamic references { you refer to different documents dynamically depending upon which object the like is being placed on }.
- We have to create a schema for likes { **like.js** } inside the model folder.
- We have to require mongoose in the file.
- We have to define different fields inside the file { user, the type on which the like has been placed, and the object id on which the like has been placed }.
- RefPath - We are going to place a path to some other field that is there and that field is going to define which type of object the like has been taking place.
- We have to tell the post that it is going to have an array of like id's.
- Whenever we are looking out for the likes of a single comment, we should have an array of those likes inside the comment itself to make it easy to reference.

{ post.js }

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

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

{ comment.js }

```
    // comment belongs to a user
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
    post: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Post'
    },
    likes: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Like'
        }
    ]
},{
    timestamps: true
});

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

{ like.js }

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


const likeSchema = new mongoose.Schema({
    user: {
        type: mongoose.Schema.ObjectId
    },
    // this defines the object id of the liked object
    likeable: {
        type: mongoose.Schema.ObjectId,
        require: true,
        refPath: 'onModel'
    },
    // this field is used for defining the type of the liked object since this is a dynamic reference
    onModel: {
        type: String,
        required: true,
        enum: ['Post', 'Comment']
    }
}, {
    timestamps: true
});


const Like = mongoose.model('Like', likeSchema);
module.exports = Like;
```

- The **enum** keyword is used to restrict a value to a fixed set of values. It tells that the value of **onmodel** (a property that we defined) in each like, can either be on a post or comment and nothing other than that.

Actions and Routes:: Likes

- We have to create controllers actions and specific routes for the actions that we will create.
- We will create an action in the likes controller that would be called toggle likes.
- We have to create a new file inside the controllers folder { likes_controller.js }.
- We need to import three models { likes, posts, and comments }.

{ likes_controller.js }

```
const Like = require("../models/like");
const Post =  require("../models/post");
const Comment = require('../models/comment');
module.exports.toggleLike = async function(req, res){
    try{// likes/toggle/?id=abcdef&type=Post
        let likeable;
        let deleted = false;
        if (req.query.type == 'Post'){
            likeable = await Post.findById(req.query.id).populate('likes');
        }else{
            likeable = await Comment.findById(req.query.id).populate('likes');
        }// check if a like already exists
        let existingLike = await Like.findOne({
            likeable: req.query.id,
            onModel: req.query.type,
            user: req.user._id
        })// if a like already exists then delete it
        if (existingLike){
            likeable.likes.pull(existingLike._id);
            likeable.save();
            existingLike.remove();
            deleted = true;
        }else{// else make a new like
            let newLike = await Like.create({
                user: req.user._id,
                likeable: req.query.id,
                onModel: req.query.type
            });likeable.likes.push(newLike._id);
            likeable.save();
        }return res.json(200, {
            message: "Request successful!",
            data: {
                deleted: deleted
            }})}catch(err){
        console.log(err);
        return res.json(500, {
            message: 'Internal Server Error'}); }}
```

- We need to define the routes. For that, we create a new file **{ likes.js }** inside the routes folder.

{ likes.js }

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

const router = express.Router();
const likesController = require('../controllers/likes_controller');


router.post('/toggle', likesController.toggleLike);


module.exports = router;
```

{ 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'));
router.use('/likes', require('./likes'));


router.use('/api', require('./api'));

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


module.exports = router;
```

***EXTRA*: Assignment**
You have to create a link which when clicked in via AJAX will send in a request to the routes and while deleting posts or comments you need to delete the associated likes also.

Making Friendships

- Consider that, there are two tables named **Books** and **Author**. Both the tables contain a common attribute, **Authors**. Instead of placing the common attribute in two different tables, we will create a middle table for that common attribute and remove that attribute from both tables itself.. The common table is called a join table. It contains the unique attribute id of both tables, and will help us reference values from each table based on the **Authors** attribute.
- For friends { a user is a friend of a user }, there is going to be one table.
- There will be a user's table containing the id, name, and email field, instead of storing everything in a friendship column. We will create a friendship table in which we will store the user's id and the friend's id.

- Both these column ids will refer to the users table.

<u>EXTRA</u>: Assignment
You have to create a link which when clicked in via AJAX will send in a request to the routes and while deleting posts or comments you need to delete the associated likes also.

Understanding the Code:: Friendship

- You have to create a section that will contain the list of all the friends and users.
- For each friend, we need to show a cross button we can remove that person from the friend list.
- Whenever we will be looking at another user's profile, there should be an option of a button to add a friend.
- Whenever we click that button dynamically using AJAX, it will add the person to the users friend lists.
- After adding the button should change into remove using toggle action.
- We have to create a schema for friendship inside the models folder { **friendship.js** }.

The images of the final result are attached in this document at the end.

{ **friendship.js** }

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


const friendshipSchema = new mongoose.Schema({
    // the user who sent this request
    from_user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
    // the user who accepted this request, the naming is just to understand, otherwise, the users won't see a difference
    to_user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User'
    },
},{
    timestamps: true
});

const Friendship = mongoose.model('Friendship', friendshipSchema);
module.exports = Friendship;
```

- To store the array of friendships for super-fast access we will store it inside the { **user.js** } file when we are trying to find out the friendship of the user.

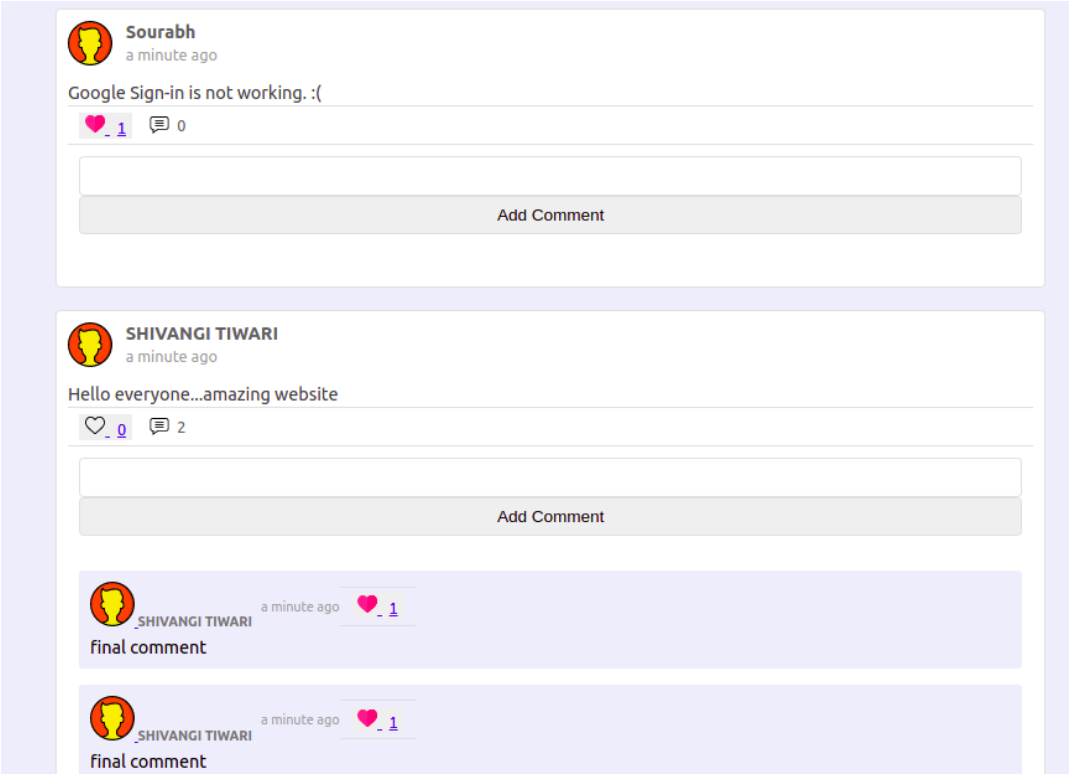
  { **user.js** }

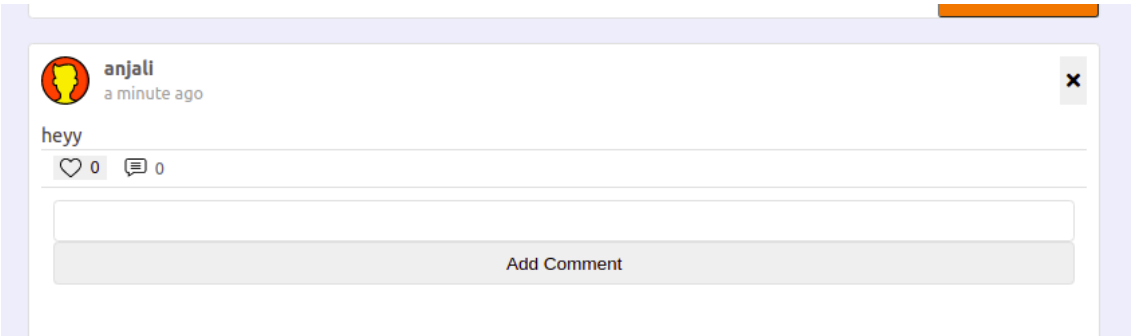
```
const userSchema = new mongoose.Schema({
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    name: {
        type: String,
        required: true
    },
    avatar: {
        type: String
    },
    friendships: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Friendship'
        }
    ]

}, {
    timestamps: true
});
```

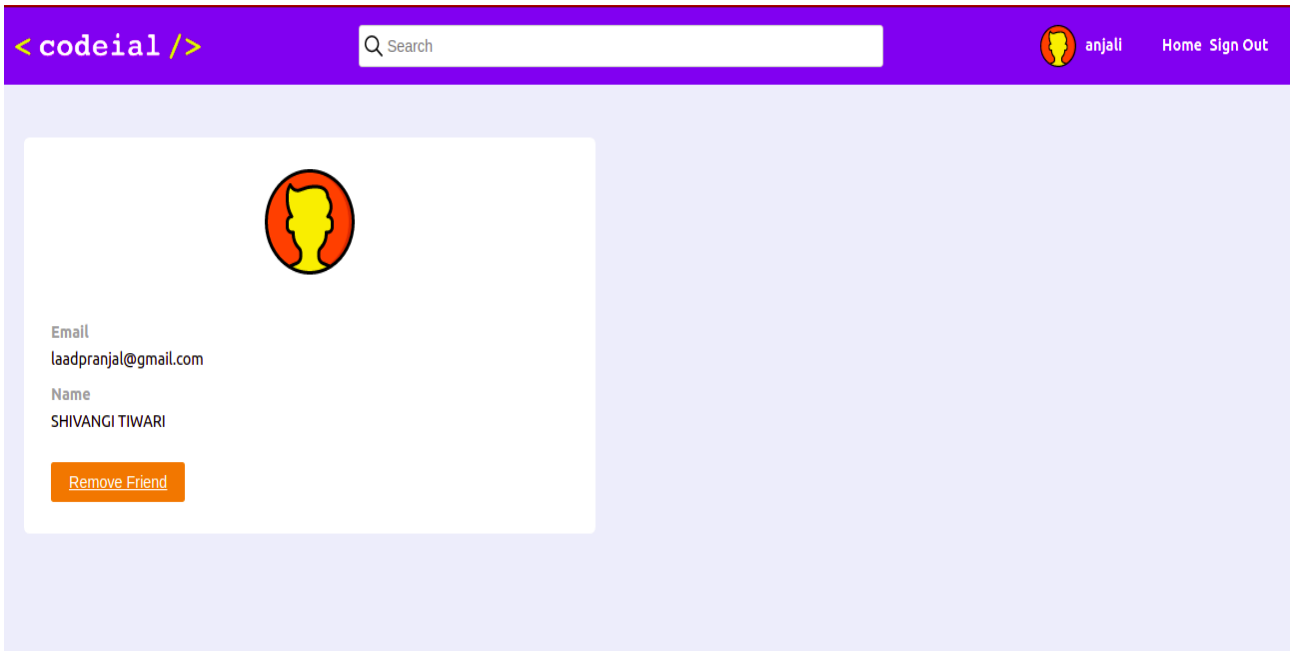
<u>EXTRA</u>: You can refer to the below page also.

{Liking the post and comments }





{ Adding friend }



{ Showing up the friend list }