Node.js: ricreare un social network come Instagram in ExpressJS

In questo articolo vedremo come implementare un social network di base simile ad Instagram con ExpressJS.

Requisiti

Gli utenti dovranno essere in grado di registrarsi, effettuare, il login, modificare il loro profilo, creare post e seguire altri utenti. All'atto della registrazione dovrà essere inviata loro un'e-mail di benvenuto.

Database

Useremo MongoDB con due collezioni: `users` e `posts`. Gestiremo la connessione e l'interazione con il database tramite il modulo Mongoose.

La collezione `users` avrà questo schema per i documenti.

```
'use strict';

const mongoose  = require('mongoose');

const { Schema }  = mongoose;

const UserSchema = new Schema({
    name: String,
    username: String,
    email: String,
    password: String,
    url: { type: String, default: '' },
```

```
    image: { type: String, default: 'default.png' },
    description: { type: String, default: '' },
    posts: Array,
    followers: Array,
    following: Array

},{collection: 'users'});

module.exports = mongoose.model('users', UserSchema);
```

Gli utenti saranno collegati ai loro post tramite l'array `posts` che conterrà una serie di Object ID in modo da poter reperire i post con l'operatore `$in`.

A loro volta `followers` e `following` conterranno una serie di Object ID relativi agli utenti.

La collezione `posts` ha invece il seguente schema.

```
'use strict';

const mongoose  = require('mongoose');

const { Schema }  = mongoose;

const PostSchema = new Schema({
    description: String,
    image: String,
    user: String,
    date: Date

},{collection: 'posts'});

module.exports = mongoose.model('posts', PostSchema);
```

Il campo `user` conterrà lo username dell'utente che ha creato il post. Volendo questo valore si può sostituire con uno di tipo Object ID.

Invio e-mail

Useremo *NodeMailer* e Mailtrap per l'invio delle e-mail in fase di sviluppo. La configurazione di Mailtrap è la seguente:

```
mail: {
      from: 'noreply@express.localhost',
      settings: {
            host: 'smtp.mailtrap.io',
            port: 2525,
            auth: {
              user: '',
              pass: ''
            }
      }
}
```

Inserite le vostre credenziali che trovate nella mailbox che avete creato su Mailtrap.

Registrazione

Dopo aver convalidato i dati in ingresso con il modulo *validator*, creiamo l'account utente ed inviamo l'e-mail di benvenuto.

```
router.post('/register', async (req, res, next) => {
    const { name, email, username, password,
password_confirmation } = req.body;
    const errors = [];
```

```
    if(validator.isEmpty(name)) {
        errors.push({
            param: 'name',
            msg: 'Name is a required field.'
        });
    }

    if(!validator.isEmail(email)) {
        errors.push({
            param: 'email',
            msg: 'Invalid e-mail address.'
        });
    }

    if(!validator.isAlphanumeric(username)) {
        errors.push({
            param: 'username',
            msg: 'Invalid username.'
        });
    }

    if(validator.isEmpty(password)) {
        errors.push({
            param: 'password',
            msg: 'Password is a required field.'
        });
    }

    if(password !== password_confirmation) {
        errors.push({
            param: 'password_confirmation',
            msg: 'Passwords do not match.'
        });
    }
```

```
    try {
        const usernameExists = await
users.countDocuments({ username: username });
        const emailExists = await
users.countDocuments({ email: email });

        if(usernameExists === 1) {
            errors.push({
                param: 'username',
                msg: 'Invalid username.'
            });
        }

        if(emailExists === 1) {
            errors.push({
                param: 'email',
                msg: 'Invalid e-mail address.'
            });
        }

    } catch(err) {
        res.json({ error: err });
    }

    if(errors.length > 0) {
        res.json({ errors });
    } else {
        const encPwd =
crypto.createHash('sha256').update(password).digest('
hex');

        const newUser = new users({
            name,
```

```
            email,
            username,
            password: encPwd

        });

        const mailer = new Mail({
            from,
            settings
        });

        try {
            await newUser.save();
            await mailer.send({ to: email, subject:
'Welcome to Express Instagram', body: `Welcome
${username}!` });
        } catch(err) {
            res.json({ error: err });
        }



        res.json({ success: true });
    }
});
```

Login

Quando l'utente effettua il login con successo, viene creato un oggetto `user` nella sessione corrente.

```
router.post('/login', async (req, res, next) => {
    const { email, password } = req.body;
```

```
    const encPwd =
crypto.createHash('sha256').update(password).digest('
hex');

    try {
        const user = await users.findOne({ email:
email, password: encPwd });
        if(user) {
            req.session.user = user;
            res.json({ success: true, username:
user.username });
        } else {
            res.json({ success: false });
        }
    } catch(err) {
        res.json({ success: false });
    }
});
```

Modifica del profilo

Questa route ha come modulo aggiuntivo *multer* che serve a gestire l'upload dei file.

```
router.post('/profile/:username/edit', async (req,
res, next) => {
    if(req.session.user) {
        const { username } = req.params;
        try {
            const upload = new Upload({
                filename: 'image',
                destination: UPLOAD_PATH +
'profiles',
```

```
                newName:
crypto.createHash('sha256').update(Date.now().toStrin
g()).digest('hex')
            });

            const uploaded = await upload.save(req,
res);

            if(uploaded.done) {
                const { url, description } =
uploaded.body;
                const {  file } = uploaded;
                const data = {
                    url,
                    description,
                    image: file.filename
                };
                await users.findOneAndUpdate({
username: username }, { $set: data });

                res.json({ updated: true, username
});
            } else {
                res.json({ updated: false });
            }
        } catch(err) {
            res.json(err);
        }
    } else {
        res.sendStatus(403);
    }
});
```

Creazione dei post

La creazione di un post prevede anche l'upload di un'immagine associata al post, quindi anche in questo caso verrà usato il modulo *multer*.

```
router.post('/posts/create', async (req, res, next)
=> {
    if(req.session.user) {
        try {
            const upload = new Upload({
                filename: 'image',
                destination: UPLOAD_PATH + 'posts',
                newName:
crypto.createHash('sha256').update(Date.now().toStrin
g()).digest('hex')
            });

            const uploaded = await upload.save(req,
res);

            if(uploaded.done) {
                const { description } =
uploaded.body;
                const {  file } = uploaded;
                const errors = [];

                if(validator.isEmpty(description)) {
                    errors.push({
                        param: 'description',
                        msg: 'Description is a
required field.'
                    });
                }
```

```
                if(errors.length > 0) {
                    fs.unlinkSync(file.path);
                    res.json({ errors });
                } else {
                    const newPost = new posts({
                        description,
                        image: file.filename,
                        user:
req.session.user.username,
                        date: new Date()
                    });

                    newPost.save().then(post => {
                        users.findOneAndUpdate({ _id:
req.session.user._id}, { $push: { posts: post._id }
}).then(result => {
                            res.json({ created: true,
postid: post._id });
                        });
                    });
                }
            } else {
                res.json({ created: false });
            }
        } catch(err) {
            res.json(err);
        }
    } else {
        res.sendStatus(403);
    }
});
```

Quando il documento viene creato, il suo Object ID viene inserito nel campo `posts` dell'utente associato.

Follow / Unfollow

Questa feature fa una distinzione tra l'utente che segue un altro utente (follower) e l'utente che viene seguito. Quando l'utente clicca su "Follow", il codice lato client invia alla route gli Object ID del follower e del following.

Nella view in EJS avremo questa situazione:

```
<% if(isLoggedIn) { %>
                        <% if(user && user._id !=
currentUser._id) { %>
                        <% const following =
user.following;
                           const action =
following.includes(currentUser._id) ? 'unfollow' :
'follow';
                        %>
                         <button class="btn btn-
primary ml-4" data-action="<%= action %>" data-
follower="<%= user._id %>" data-following="<%=
currentUser._id %>" id="follow-btn">Follow</button>
                        <% } %>
                    <% } %>
```

`currentUser` è l'utente reperito dal database, mentre `user` è l'utente presente in sessione (l'utente che ha effettuato il login). Se l'utente non sta visualizzando il suo profilo ma quello di un altro utente, allora il pulsante verrà visualizzato e l'action del pulsante sarà `follow` o `unfollow` a seconda del fatto che l'Object ID del profilo visualizzato sia presente o meno nell'array `following` dell'utente che sta visualizzando quel profilo.

La nostra route tiene conto della action inviata via AJAX in questo modo:

```
router.post('/follow', async (req, res, next) => {
    const { follower, following, action } = req.body;
    try {
        switch(action) {
            case 'follow':
                await Promise.all([
                    users.findByIdAndUpdate(follower,
{ $push: { following: following }}),

users.findByIdAndUpdate(following, { $push: {
followers: follower }})

                ]);
            break;

            case 'unfollow':
                await Promise.all([
                    users.findByIdAndUpdate(follower,
{ $pull: { following: following }}),

users.findByIdAndUpdate(following, { $pull: {
followers: follower }})

                ]);
            break;

            default:
                break;
        }

        res.json({ done: true });

    } catch(err) {
        res.json({ done: false });
```

```
    }
});
```

Il codice lato client semplicemente aggiunge la logica del cambio dinamico del testo del pulsante e del numero di follower in base alla action corrente.

```
var $followBtn = $( "#follow-btn" );

    if( $followBtn.length ) {
        $followBtn.click(function() {
            var data = {
                follower: $followBtn.data( "follower"
),
                following: $followBtn.data(
"following" ),
                action: $followBtn.data( "action" )
            };

            $.post( "/follow", data, function( res )
{
              if( res.done ) {
                var text = $followBtn.data( "action"
) === "follow" ? "Unfollow" : "Follow";
                var count = parseInt( $( "#followers"
).text(), 10 );

                if( text === "Unfollow" ) {
                    $( "#followers" ).text( count + 1
);
                    $followBtn.data( "action",
"unfollow" ).text( text );
                } else {
                    $( "#followers" ).text( count - 1
);
```

```
                    $followBtn.data( "action",
"follow" ).text( text );
                }
            }
        });
    });
}
```

Codice sorgente

GitHub