Node.js: come implementare l'autenticazione a due fattori in ExpressJS

In questo articolo vedremo come implementare l'autenticazione a due fattori in Node.js con ExpressJS.

Useremo MongoDB come database e definiremo due collezioni, una per gli utenti ed una per i codici di autenticazione.

La prima collezione ha il seguente schema in Mongoose:

```
'use strict';

const mongoose  = require('mongoose');

const { Schema }  = mongoose;

const UserSchema = new Schema({
    username: String,
    password: String,
    email: String


},{collection: 'users'});

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

I codici di autenticazione avranno invece il seguente schema:

```
'use strict';

const mongoose  = require('mongoose');

const { Schema }  = mongoose;

const AuthTokenSchema = new Schema({
    value: String,
    expires: {
        type: Number,
        default: Date.now()
    }

},{collection: 'authtokens'});

module.exports = mongoose.model('authtoken',
AuthTokenSchema);
```

Poiché invieremo il codice di autenticazione via e-mail, abbiamo bisogno di una classe wrapper che racchiuda la logica del modulo Nodemailer.

```
'use strict';

const nodemailer = require('nodemailer');


class Mail {
    constructor({ from, settings }) {
        this.settings = settings;
        this.options = {
            from: from,
            to: '',
            subject: '',
```

```
            text: '',
            html: ''

        };
    }

    send(to, subject, body) {
        if(nodemailer && this.options) {
            let self = this;
            const transporter =
nodemailer.createTransport(self.settings);

            self.options.to = to;
            self.options.subject = subject;
            self.options.text = body;

            if(transporter !== null) {
                return new Promise((resolve, reject)
=> {

transporter.sendMail(self.options, (error, info) => {
                        if(error) {
                            reject(false);
                        } else {
                            resolve(true);
                        }
                    });
                });
            }
        }
    }
}

module.exports = Mail;
```

Abbiamo anche bisogno di gestire le sessioni ed i messaggi di errore flash da restituire dopo un redirect HTTP. Installiamo quindi i moduli richiesti.

```
npm install express-session connect-flash --save
```

Per mostrare un messaggio flash nelle view possiamo usare il seguente approccio.

```
<% if(message.length > 0) { %>
  <div class="alert alert-danger"><%= message %>
</div>
<% } %>
```

Per concludere il nostro setup, dobbiamo creare un semplice middleware che impedisca di accedere a determinate route se l'utente non è autenticato.

```
'use strict';

module.exports = (req, res, next) => {
    if(!req.session.user) {
        return res.redirect('/');
    }
    next();
};
```

Il primo passaggio da implementare è il login. Se l'utente si autentica con successo, creiamo il codice di autenticazione e lo inviamo tramite e-mail salvando l'utente nella sessione corrente.

```
'use strict';

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

```
const express = require('express');
const router = express.Router();
const User = require('../models/user');
const AuthToken = require('../models/authtoken');
const Mail = require('../classes/Mail');
const { mail: mailSettings } = require('../config');
const auth = require('../middleware/auth');

// Pagina di login
router.get('/', (req, res, next) => {

    res.render('index', {
        message: req.flash('message')
    });
});

// Route del logout

router.get('/logout', auth, (req, res, next) => {
    delete req.session.user;
    res.redirect('/');
});

// Pagina di inserimento del codice di autenticazione

router.get('/auth', auth, (req, res, next) => {

    res.render('auth', {
        message: req.flash('message')
    });
});

// La bacheca dell'utente
```

```
router.get('/dashboard', auth, (req, res, next) => {
    res.render('dashboard');
});


// Elaborazione del login

router.post('/auth', async (req, res, next) => {

    const { username, password } = req.body;
    const encPassword =
crypto.createHash('md5').update(password).digest('hex
');

    try {
        const user = await User.findOne({ username:
username, password: encPassword});

        if(user) {
            const code = Math.floor(Math.random() *
(999999 - 100000) + 100000).toString();

            const authToken = new AuthToken({
                value: code
            });

            authToken.save();

            req.session.user = user;

            res.redirect('/auth');

            const mail = new Mail({
                from: mailSettings.from,
```

```
                settings: mailSettings.settings
            });

            await mail.send(user.email, 'Your Auth
Code', `Your Auth Code is: ${code}`);
        } else {
            req.flash('message', 'Invalid login.');
            res.redirect('/');
        }
    } catch(err) {
        res.sendStatus(500);
    }
});

//...

module.exports = router;
```

Noterete che l'invio dell'e-mail nel codice precedente avviene dopo il redirect HTTP: questa scelta serve ad impedire che l'utente attenda più del dovuto a causa dell'elaborazione della transazione SMTP.

Quando l'utente effettua con successo il login viene reindirizzato alla pagina che contiene il form di verifica del codice di autenticazione. La route che effettua la verifica è la seguente:

```
router.post('/verify-auth', auth, async (req, res,
next) => {
    const { code } = req.body;

    try {
        const authToken = await AuthToken.findOne({
value: code });
```

```
        if(!authToken) {
            req.flash('message', 'Invalid Auth
Code.');
            return res.redirect('/');
        }

        const expiresIn = 1000 * 60 * 60 * 15; // 15
minuti

        if((Date.now() - authToken.expires) >
expiresIn) {
            req.flash('message', 'Invalid Auth
Code.');
            return res.redirect('/');
        }

        await authToken.remove(); // Rimuoviamo il
codice

        res.redirect('/dashboard');

    } catch(err) {
        res.sendStatus(500);
    }

});
```

Volendo perfezionare l'implementazione, si può usare un gateway esterno come Clickatell per l'invio di un SMS con il codice di autenticazione in luogo dell'e-mail.