Node.js: login con i social network

In questo articolo vedremo come implementare il login con i social network in Node.js.

Facebook

Useremo Passport come middleware di autenticazione tramite OAuth con una sua strategy specifica dedicata a Facebook.

Passport inserisce nell'oggetto `request` metodi e proprietà. Una di queste proprietà è l'oggetto `user` che contiene i dati dell'utente restituiti dalla piattaforma su cui si effettua l'autenticazione. Questo oggetto viene salvato nella sessione corrente ed è disponibile in tutte le route.

Occorre precisare che l'app creata su Facebook con il servizio Facebook Login dovrà essere live e approvata per poter funzionare in un ambiente di produzione. L'implementazione che segue opera in modalità development e quindi Facebook non effettua le normali verifiche richieste in un'app live.

Creiamo un file `.env` con le nostre credenziali di accesso.

```
FACEBOOK_CLIENT_ID=your-client-id  
FACEBOOK_CLIENT_SECRET=your-client-secret  
SESSION_SECRET=choose-a-random-string
```

Carichiamo la configurazione all'inizio del file principale.

```
'use strict';  
  
require('dotenv').config();
```

Ora definiamo le risorse principali della nostra applicazione.

```
const path = require('path');
const express = require('express');
const passport = require('passport');
const { Strategy } = require('passport-facebook');
const { FACEBOOK_CLIENT_ID, FACEBOOK_CLIENT_SECRET,
SESSION_SECRET } = process.env;
const port = process.env.PORT || 3000;
const app = express();
const routes = require('./routes');
```

A questo punto configuriamo Passport con le sue impostazioni di base.

```
passport.use(new Strategy({
  clientID: FACEBOOK_CLIENT_ID,
  clientSecret: FACEBOOK_CLIENT_SECRET,
  callbackURL: '/return'
},
  (accessToken, refreshToken, profile, cb) => {
    return cb(null, profile);
  }));

passport.serializeUser((user, cb) => {
  cb(null, user);
});

passport.deserializeUser((obj, cb) => {
  cb(null, obj);
});
```

`callbackURL` definisce l'URL a cui l'utente verrà reindirizzato dopo l'autenticazione OAuth. In produzione dovete impostare l'URL completo

nelle impostazioni OAuth del servizio Facebook Login nella vostra app Facebook.

Ora possiamo inizializzare Passport e collegarlo alla sessione.

```
app.use(require('express-session')({ secret:
SESSION_SECRET, resave: true, saveUninitialized: true
}));
app.use(passport.initialize());
app.use(passport.session());

app.use('/', routes);

app.listen(port);
```

Dobbiamo quindi definire le route della nostra applicazione tenendo a mente che ci dovrà essere una route per il login, una per il logout ed una per l'URL di callback OAuth. In tutti i casi effettueremo un reindirizzamento HTTP sulla home page.

```
'use strict';

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

router.get('/', (req, res, next) => {
  const { user } = req;
  res.render('home', { user });
});

router.get('/login/facebook',
passport.authenticate('facebook'));
```

```

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

router.get('/return',
  passport.authenticate('facebook', {
    failureRedirect: '/' })),
  (req, res, next) => {
    res.redirect('/');
  });

module.exports = router;

```

La view della home page verrà modificata dinamicamente in base al fatto che un utente abbia effettuato il login o meno.

```

<% if (!user) { %>
  <h1>Welcome!</h1>
  <p class="mt-5"><a href="/login/facebook"
class="loginBtn loginBtn--facebook">Login with
Facebook</a></p>
<% } else { %>
  <h1>Hello, <%= user.displayName %>.</h1>
  <p class="mt-5"><a href="/logout" class="btn btn-
primary">Logout</a></p>
<% } %>

```

Le informazioni che Facebook restituisce variano in base ai permessi che la vostra app ha richiesto alla piattaforma. Si tenga presente che un'app Facebook deve necessariamente essere esaminata ed approvata prima di passare ad uno status pienamente live.

Codice sorgente

GitHub

Twitter

Useremo Passport come middleware di autenticazione tramite OAuth con una sua strategy specifica dedicata a Twitter.

Passport inserisce nell'oggetto `request` metodi e proprietà. Una di queste proprietà è l'oggetto `user` che contiene i dati dell'utente restituiti dalla piattaforma su cui si effettua l'autenticazione. Questo oggetto viene salvato nella sessione corrente ed è disponibile in tutte le route.

Occorre fare una precisazione sugli URL di callback di un'applicazione Twitter. Twitter distingue due tipi di ambienti: sviluppo e produzione. In sviluppo un URL come `http://localhost:3000/return` può essere gestito da Passport come relativo, ossia semplicemente `/return`. Al contrario in produzione l'URL di callback inserito in Passport nella fase di inizializzazione deve essere assoluto.

Creiamo un file `.env` con le nostre credenziali di accesso.

```
TWITTER_CONSUMER_KEY=your-consumer-key  
TWITTER_CONSUMER_SECRET=your-consumer-secret  
SESSION_SECRET=choose-a-random-string
```

Carichiamo la configurazione all'inizio del file principale.

```
'use strict';  
  
require('dotenv').config();
```

Ora definiamo le risorse principali della nostra applicazione.

```

const path = require('path');
const express = require('express');
const passport = require('passport');
const { Strategy } = require('passport-twitter');
const { TWITTER_CONSUMER_KEY,
TWITTER_CONSUMER_SECRET, SESSION_SECRET } =
process.env;
const port = process.env.PORT || 3000;
const app = express();
const routes = require('./routes');

```

A questo punto configuriamo Passport con le sue impostazioni di base.

```

passport.use(new Strategy({
  consumerKey: TWITTER_CONSUMER_KEY,
  consumerSecret: TWITTER_CONSUMER_SECRET,
  callbackURL: '/return'
},
(accessToken, refreshToken, profile, cb) => {
  return cb(null, profile);
}));

passport.serializeUser((user, cb) => {
  cb(null, user);
});

passport.deserializeUser((obj, cb) => {
  cb(null, obj);
});

```

`callbackURL` definisce l'URL a cui l'utente verrà reindirizzato dopo l'autenticazione OAuth. In produzione dovete impostare l'URL completo nelle impostazioni dell'app Twitter.

Ora possiamo inizializzare Passport e collegarlo alla sessione.

```
app.use(require('express-session')({ secret:
SESSION_SECRET, resave: true, saveUninitialized: true
}));
app.use(passport.initialize());
app.use(passport.session());

app.use('/', routes);

app.listen(port);
```

Dobbiamo quindi definire le route della nostra applicazione tenendo a mente che ci dovrà essere una route per il login, una per il logout ed una per l'URL di callback OAuth. In tutti i casi effettueremo un reindirizzamento HTTP sulla home page.

```
'use strict';

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

router.get('/', (req, res, next) => {
  const { user } = req;
  res.render('home', { user });
});

router.get('/login/twitter',
passport.authenticate('twitter'));

router.get('/logout', (req, res, next) => {
  req.logout();
  res.redirect('/');
```

```
});

router.get('/return',
  passport.authenticate('twitter', { failureRedirect:
  '/' })),
  (req, res, next) => {
    res.redirect('/');
  });

module.exports = router;
```

La view della home page verrà modificata dinamicamente in base al fatto che un utente abbia effettuato il login o meno.

```
<% if (!user) { %>
  <h1>Welcome!</h1>
  <p class="mt-5"><a href="/login/twitter"
class="loginBtn loginBtn--twitter">Login with
Twitter</a></p>
<% } else { %>
  <h1>Hello, <%= user.displayName %>.</h1>
  <p class="mt-5"><a href="/logout" class="btn btn-
primary">Logout</a></p>
<% } %>
```

Codice sorgente

GitHub

Google

Useremo Passport come middleware di autenticazione tramite OAuth con una sua strategy specifica dedicata a Google.

Passport inserisce nell'oggetto `request` metodi e proprietà. Una di queste proprietà è l'oggetto `user` che contiene i dati dell'utente restituiti dalla piattaforma su cui si effettua l'autenticazione. Questo oggetto viene salvato nella sessione corrente ed è disponibile in tutte le route.

Occorre fare una precisazione sugli URL di callback di un'applicazione Google. Google distingue due tipi di ambienti: sviluppo e produzione. In sviluppo un URL come `http://localhost:3000/return` può essere gestito da Passport come relativo, ossia semplicemente `/return`. Al contrario in produzione l'URL di callback inserito in Passport nella fase di inizializzazione deve essere assoluto.

Nella Google Developers Console vanno specificati sia il dominio base dell'applicazione sia gli URL assoluti per il callback OAuth e quello dell'origine consentita per l'accesso. Lo scope OAuth sarà `profile`.

Creiamo un file `.env` con le nostre credenziali di accesso.

```
GOOGLE_CLIENT_ID=your-client-id
GOOGLE_CLIENT_SECRET=your-client-secret
SESSION_SECRET=choose-a-random-string
```

Carichiamo la configurazione all'inizio del file principale.

```
'use strict';

require('dotenv').config();
```

Ora definiamo le risorse principali della nostra applicazione.

```
const path = require('path');
const express = require('express');
const passport = require('passport');
const { Strategy } = require('passport-google-
```

```
oauth20');
const { GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET,
SESSION_SECRET } = process.env;
const port = process.env.PORT || 3000;
const app = express();
const routes = require('./routes');
```

A questo punto configuriamo Passport con le sue impostazioni di base.

```
passport.use(new Strategy({
  clientID: GOOGLE_CLIENT_ID,
  clientSecret: GOOGLE_CLIENT_SECRET,
  callbackURL: '/return'
}),
(accessToken, refreshToken, profile, cb) => {
  return cb(null, profile);
}));

passport.serializeUser((user, cb) => {
  cb(null, user);
});

passport.deserializeUser((obj, cb) => {
  cb(null, obj);
});
```

`callbackURL` definisce l'URL a cui l'utente verrà reindirizzato dopo l'autenticazione OAuth. In produzione dovete impostare l'URL completo nelle impostazioni dell'app Google.

Ora possiamo inizializzare Passport e collegarlo alla sessione.

```
app.use(require('express-session')({ secret:
SESSION_SECRET, resave: true, saveUninitialized: true
```

```
});  
app.use(passport.initialize());  
app.use(passport.session());  
  
app.use('/', routes);  
  
app.listen(port);
```

Dobbiamo quindi definire le route della nostra applicazione tenendo a mente che ci dovrà essere una route per il login, una per il logout ed una per l'URL di callback OAuth. In tutti i casi effettueremo un reindirizzamento HTTP sulla home page.

```
'use strict';  
  
const express = require('express');  
const passport = require('passport');  
const router = express.Router();  
  
router.get('/', (req, res, next) => {  
  const { user } = req;  
  res.render('home', { user });  
});  
  
router.get('/login/google',  
passport.authenticate('google', { scope: ['profile']  
}));  
  
router.get('/logout', (req, res, next) => {  
  req.logout();  
  res.redirect('/');  
});  
  
router.get('/return',
```

```
passport.authenticate('google', { failureRedirect:
  '/' } ),
  (req, res, next) => {
    res.redirect('/');
  });

module.exports = router;
```

La view della home page verrà modificata dinamicamente in base al fatto che un utente abbia effettuato il login o meno.

```
<% if (!user) { %>
  <h1>Welcome!</h1>
  <p class="mt-5"><a href="/login/google"
class="loginBtn loginBtn--google">Login with
Google</a></p>
<% } else { %>
  <h1>Hello, <%= user.displayName %>.</h1>
  <p class="mt-5"><a href="/logout" class="btn btn-
primary">Logout</a></p>
<% } %>
```

Codice sorgente

GitHub

Linkedin

Useremo Passport come middleware di autenticazione tramite OAuth con una sua strategy specifica dedicata a LinkedIn.

Passport inserisce nell'oggetto `request` metodi e proprietà. Una di queste proprietà è l'oggetto `user` che contiene i dati dell'utente restituiti dalla

piattaforma su cui si effettua l'autenticazione. Questo oggetto viene salvato nella sessione corrente ed è disponibile in tutte le route.

Occorre fare una precisazione sugli URL di callback di un'applicazione LinkedIn. LinkedIn distingue due tipi di ambienti: sviluppo e produzione. In sviluppo un URL come `http://localhost:3000/return` può essere gestito da Passport come relativo, ossia semplicemente `/return`. Al contrario in produzione l'URL di callback inserito in Passport nella fase di inizializzazione deve essere assoluto.

Nella sezione sviluppatori di LinkedIn vanno specificati gli URL assoluti per il callback OAuth. Gli scope OAuth vanno definiti in Passport nella fase di inizializzazione.

Creiamo un file `.env` con le nostre credenziali di accesso.

```
LINKEDIN_API_KEY=your-client-id
LINKEDIN_SECRET_KEY=your-client-secret
SESSION_SECRET=choose-a-random-string
```

Carichiamo la configurazione all'inizio del file principale.

```
'use strict';

require('dotenv').config();
```

Ora definiamo le risorse principali della nostra applicazione.

```
const path = require('path');
const express = require('express');
const passport = require('passport');
const { Strategy } = require('passport-linkedin-oauth2');
const { LINKEDIN_API_KEY, LINKEDIN_SECRET_KEY,
```

```
SESSION_SECRET } = process.env;
const port = process.env.PORT || 3000;
const app = express();
const routes = require('./routes');
```

A questo punto configuriamo Passport con le sue impostazioni di base.

```
passport.use(new Strategy({
  clientID: LINKEDIN_API_KEY,
  clientSecret: LINKEDIN_SECRET_KEY,
  callbackURL: '/return',
  scope: ['r_emailaddress', 'r_liteprofile'],
  state: true
}),
(accessToken, refreshToken, profile, cb) => {
  return cb(null, profile);
}));

passport.serializeUser((user, cb) => {
  cb(null, user);
});

passport.deserializeUser((obj, cb) => {
  cb(null, obj);
});
```

`callbackURL` definisce l'URL a cui l'utente verrà reindirizzato dopo l'autenticazione OAuth. In produzione dovete impostare l'URL completo nelle impostazioni dell'app LinkedIn.

Ora possiamo inizializzare Passport e collegarlo alla sessione.

```
app.use(require('express-session')({ secret:
SESSION_SECRET, resave: true, saveUninitialized: true
```

```
});  
app.use(passport.initialize());  
app.use(passport.session());  
  
app.use('/', routes);  
  
app.listen(port);
```

Dobbiamo quindi definire le route della nostra applicazione tenendo a mente che ci dovrà essere una route per il login, una per il logout ed una per l'URL di callback OAuth. In tutti i casi effettueremo un reindirizzamento HTTP sulla home page.

```
'use strict';  
  
const express = require('express');  
const passport = require('passport');  
const router = express.Router();  
  
router.get('/', (req, res, next) => {  
  const { user } = req;  
  res.render('home', { user });  
});  
  
router.get('/login/linkedin',  
passport.authenticate('linkedin'));  
  
router.get('/logout', (req, res, next) => {  
  req.logout();  
  res.redirect('/');  
});  
  
router.get('/return',  
  passport.authenticate('linkedin', {
```

```
failureRedirect: '/', successRedirect: '/' }),
  (req, res, next) => {
    res.redirect('/');
  });

module.exports = router;
```

La view della home page verrà modificata dinamicamente in base al fatto che un utente abbia effettuato il login o meno.

```
<% if (!user) { %>
  <h1>Welcome!</h1>
  <p class="mt-5"><a href="/login/linkedin"
class="loginBtn loginBtn--linkedin">Login with
Linkedin</a></p>
<% } else { %>
  <h1>Hello, <%= user.displayName %>.</h1>
  <p class="mt-5"><a href="/logout" class="btn btn-
primary">Logout</a></p>
<% } %>
```

Codice sorgente

GitHub