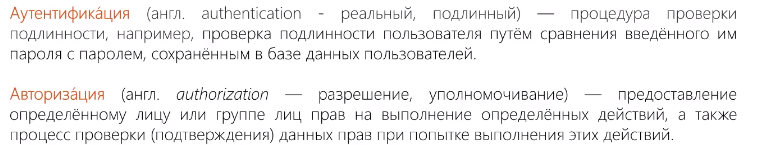
**Збереження пароля у базі даних**

|  |  |
| --- | --- |
|  |  |
|  | const mongoose = require('mongoose');  const crypto = require('crypto');  const jwt = require('jsonwebtoken');  const { Schema } = mongoose;  const UsersSchema = new Schema({  email: String,  **hash: String,** //🡨 поле, де буде зберігатися хеш пароля  **salt: String,** //🡨 поле, де буде зберігатися ключ  });  //--------------- Функція для формування хешу пароля -----------------  **UsersSchema.methods.setPassword** = function(password) {  this.salt = crypto.randomBytes(16).toString('hex');  this.hash = crypto.pbkdf2Sync(password, this.salt, 10000, 512, 'sha512').toString('hex');  };  //---------------- Функція для перевірки правильності пароля ------------  **UsersSchema.methods.validatePassword** = function(password) {  //----------- Формуємо хеш переданого (для перевірки) пароля ----  const hash = crypto.pbkdf2Sync(password, this.salt, 10000, 512, 'sha512').toString('hex');  //------------ Перевіряємо, чи одержано такий же хеш як у базі -------------  return this.hash === hash;  };  mongoose.model('Users', UsersSchema); |
|  |  |
|  |  |
|  |  |

==============================================================================================================================================

**Passport**

<http://www.passportjs.org/docs/login/>



Passport is authentication middleware for Node. It is designed to serve a singular purpose: authenticate requests. When writing modules, encapsulation is a virtue, so Passport delegates all other functionality to the application. This separation of concerns keeps code clean and maintainable, and makes Passport extremely easy to integrate into an application.

In modern web applications, authentication can take a variety of forms. Traditionally, users log in by providing a username and password. With the rise of social networking, single sign-on using an OAuth provider such as Facebook or Twitter has become a popular authentication method. Services that expose an API often require token-based credentials to protect access.

Passport recognizes that each application has unique authentication requirements. Authentication mechanisms, known as strategies, are packaged as individual modules. Applications can choose which strategies to employ, without creating unnecessary dependencies.

Despite the complexities involved in authentication, code does not have to be complicated.

app.post('/login', passport.authenticate('local', { successRedirect: '/',

failureRedirect: '/login' }));

Install

npm install passport

npm install passport-local

Configure

Three pieces need to be configured to use Passport for authentication:

Authentication strategies

Application middleware

Sessions (optional)

|  |  |  |
| --- | --- | --- |
| Задаємо стратегію |  | var passport = require('passport')  , **LocalStrategy = require('passport-local').Strategy;** |
| Конфігурація  (використання сесій не є обов’язковим) |  | **var session = require("express-session"),**  bodyParser = require("body-parser");  app.use(express.static("public"));  **app.use(session({ secret: "cats" }));**  app.use(bodyParser.urlencoded({ extended: false }));  **app.use(passport.initialize());**  **app.use(passport.session());** |
| Здійснюємо перевірку і викликаємо  Verify Callback | Якщо помилка при одержанні інформації з бази  return done(**err**);  Якщо все добре  return done(null, **user**);  якщо помилка в паролі  return done(null, **false**); | passport.use(**new LocalStrategy**(  function(username, password, done) {  User.findOne( { username: username }, function (err, user) {  if (err) { **return done(err);** }  if (!user) {  return **done(null, false,** { message: 'Incorrect username.' });  }  if (!user. **validatePassword** (password)) {  return **done(null, false**, { message: 'Incorrect password.' });  }  return **done(null, user);**  });  }  )); |
|  |  | app.post('/login',  passport.authenticate('local', { successRedirect: '/',  failureRedirect: '/login',  failureFlash: true })  ); |
| Зчитування даних про користувача | Якщо користувач авторизувався, то відповідний об’єкт user доступний через об’єкт запиту  **req.user**  Якщо користувач не авторизований, то req.user  дорівнює null | app.get(**"/secret"**,**function** (req,res) {  **if**(**req.user**)  res.send(**"ДОСТУП ДОЗВОЛЕНО"**);  **else** res.send(**"ДОСТУП ЗАБОРОНЕНО"**); });  (можна було описати і функцію-middleware) |

|  |  |  |
| --- | --- | --- |
|  | ------- Форма для автентифікації ---------- |  |
|  | <form action**="/login" method="post">**  <div>  <label>Username:</label>  <input type="text" name="**username**"/>  </div>  <div>  <label>Password:</label>  <input type="password" name="**password**"/>  </div>  <div>  <input type="submit" value="Log In"/>  </div>  </form> |  |

**Log In**

Passport exposes a login() function on req (also aliased as logIn()) that can be used to establish a login session.

**req.login**(user, function(err) {

if (err) { return next(err); }

return res.redirect('/users/' + req.user.username);

});

When the login operation completes, user will be assigned to req.user.

Note: passport.authenticate() middleware invokes req.login() automatically. This function is primarily used when users sign up, during which req.login() can be invoked to automatically log in the newly registered user.

**Log Out**

Passport exposes a logout() function on req (also aliased as logOut()) that can be called from any route handler which needs to terminate a login session. Invoking logout() will remove the req.user property and clear the login session (if any).

**app.get('/logout'**, function(req, res){

req.logout();

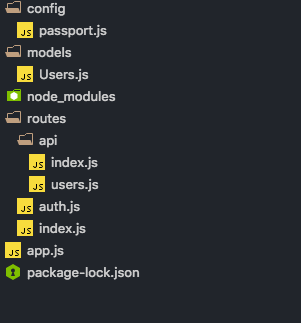
res.redirect('/');

});

Приклад. **Див. архів PassportAuth.rar**

**JSON Web Token** (**JWT**) — это открытый стандарт ([RFC 7519](https://tools.ietf.org/html/rfc7519)) для создания токенов доступа, основанный на формате [JSON](https://ru.wikipedia.org/wiki/JSON). Как правило, используется для передачи данных [авторизации](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F" \o "Авторизация) в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

<https://medium.freecodecamp.org/learn-how-to-handle-authentication-with-node-using-passport-js-4a56ed18e81e>



Приклад

|  |  |  |
| --- | --- | --- |
| ------------------- app.js ------------------ | --------------- models/User.js ----------- | ----- config/passport.js ------- |
| |  | | --- | | const express = require('express'); | |  | const path = require('path'); | |  | const bodyParser = require('body-parser'); | |  | const session = require('express-session'); | |  | const cors = require('cors'); | |  | const mongoose = require('mongoose'); | |  | const errorHandler = require('errorhandler'); | |  |  | |  | //Configure mongoose's promise to global promise | |  | mongoose.promise = global.Promise; | |  |  | |  | //Configure isProduction variable | |  | const isProduction = process.env.NODE\_ENV === 'production'; | |  |  | |  | //Initiate our app | |  | const app = express(); | |  |  | |  | //Configure our app | |  | app.use(cors()); | |  | app.use(require('morgan')('dev')); | |  | app.use(bodyParser.urlencoded({ extended: false })); | |  | app.use(bodyParser.json()); | |  | app.use(express.static(path.join(\_\_dirname, 'public'))); | |  | app.use(session({ secret: 'passport-tutorial', cookie: { maxAge: 60000 }, resave: false, saveUninitialized: false })); | |  |  | |  | if(!isProduction) { | |  | app.use(errorHandler()); | |  | } | |  |  | |  | //Configure Mongoose | |  | mongoose.connect('mongodb://localhost/passport-tutorial'); | |  | mongoose.set('debug', true); | |  |  | |  | //Error handlers & middlewares | |  | if(!isProduction) { | |  | app.use((err, req, res) => { | |  | res.status(err.status || 500); | |  |  | |  | res.json({ | |  | errors: { | |  | message: err.message, | |  | error: err, | |  | }, | |  | }); | |  | }); | |  | } | |  |  | |  | app.use((err, req, res) => { | |  | res.status(err.status || 500); | |  |  | |  | res.json({ | |  | errors: { | |  | message: err.message, | |  | error: {}, | |  | }, | |  | }); | |  | }); | |  |  | |  | app.listen(8000, () => console.log('Server running on http://localhost:8000/')); | | |  | | --- | | const mongoose = require('mongoose'); | |  | const crypto = require('crypto'); | |  | const jwt = require('jsonwebtoken'); | |  |  | |  | const { Schema } = mongoose; | |  |  | |  | const UsersSchema = new Schema({ | |  | email: String, | |  | hash: String, | |  | salt: String, | |  | }); | |  |  | |  | UsersSchema.methods.setPassword = function(password) { | |  | this.salt = crypto.randomBytes(16).toString('hex'); | |  | this.hash = crypto.pbkdf2Sync(password, this.salt, 10000, 512, 'sha512').toString('hex'); | |  | }; | |  |  | |  | UsersSchema.methods.validatePassword = function(password) { | |  | const hash = crypto.pbkdf2Sync(password, this.salt, 10000, 512, 'sha512').toString('hex'); | |  | return this.hash === hash; | |  | }; | |  |  | |  | UsersSchema.methods.generateJWT = function() { | |  | const today = new Date(); | |  | const expirationDate = new Date(today); | |  | expirationDate.setDate(today.getDate() + 60); | |  |  | |  | return jwt.sign({ | |  | email: this.email, | |  | id: this.\_id, | |  | exp: parseInt(expirationDate.getTime() / 1000, 10), | |  | }, 'secret'); | |  | } | |  |  | |  | UsersSchema.methods.toAuthJSON = function() { | |  | return { | |  | \_id: this.\_id, | |  | email: this.email, | |  | token: this.generateJWT(), | |  | }; | |  | }; | |  |  | |  | mongoose.model('Users', UsersSchema); | | |  | | --- | | const mongoose = require('mongoose'); | |  | const passport = require('passport'); | |  | const LocalStrategy = require('passport-local'); | |  |  | |  | const Users = mongoose.model('Users'); | |  |  | |  | passport.use(new LocalStrategy({ | |  | usernameField: 'user[email]', | |  | passwordField: 'user[password]', | |  | }, (email, password, done) => { | |  | Users.findOne({ email }) | |  | .then((user) => { | |  | if(!user || !user.validatePassword(password)) { | |  | return done(null, false, { errors: { 'email or password': 'is invalid' } }); | |  | } | |  |  | |  | return done(null, user); | |  | }).catch(done); | |  | })); | |
|  |  |  |
| ------------ auth.js ---------------------- | ------- routes/ index.js -------------------- | --------- api/index.js ---------------- |
| |  | | --- | | onst jwt = require('express-jwt'); | |  |  | |  | const getTokenFromHeaders = (req) => { | |  | const { headers: { authorization } } = req; | |  |  | |  | if(authorization && authorization.split(' ')[0] === 'Token') { | |  | return authorization.split(' ')[1]; | |  | } | |  | return null; | |  | }; | |  |  | |  | const auth = { | |  | required: jwt({ | |  | secret: 'secret', | |  | userProperty: 'payload', | |  | getToken: getTokenFromHeaders, | |  | }), | |  | optional: jwt({ | |  | secret: 'secret', | |  | userProperty: 'payload', | |  | getToken: getTokenFromHeaders, | |  | credentialsRequired: false, | |  | }), | |  | }; | |  |  | |  | module.exports = auth; | | |  | | --- | | const express = require('express'); | |  | const router = express.Router(); | |  |  | |  | router.use('/api', require('./api')); | |  |  | |  | module.exports = router; | | |  | | --- | | const express = require('express'); | |  | const router = express.Router(); | |  |  | |  | router.use('/users', require('./users')); | |  |  | |  | module.exports = router; | |
|  |  |  |
| -------------- api/user.js ------------- |  |  |
| |  | | --- | | const mongoose = require('mongoose'); | |  | const passport = require('passport'); | |  | const router = require('express').Router(); | |  | const auth = require('../auth'); | |  | const Users = mongoose.model('Users'); | |  |  | |  | //POST new user route (optional, everyone has access) | |  | router.post('/', auth.optional, (req, res, next) => { | |  | const { body: { user } } = req; | |  |  | |  | if(!user.email) { | |  | return res.status(422).json({ | |  | errors: { | |  | email: 'is required', | |  | }, | |  | }); | |  | } | |  |  | |  | if(!user.password) { | |  | return res.status(422).json({ | |  | errors: { | |  | password: 'is required', | |  | }, | |  | }); | |  | } | |  |  | |  | const finalUser = new Users(user); | |  |  | |  | finalUser.setPassword(user.password); | |  |  | |  | return finalUser.save() | |  | .then(() => res.json({ user: finalUser.toAuthJSON() })); | |  | }); | |  |  | |  | //POST login route (optional, everyone has access) | |  | router.post('/login', auth.optional, (req, res, next) => { | |  | const { body: { user } } = req; | |  |  | |  | if(!user.email) { | |  | return res.status(422).json({ | |  | errors: { | |  | email: 'is required', | |  | }, | |  | }); | |  | } | |  |  | |  | if(!user.password) { | |  | return res.status(422).json({ | |  | errors: { | |  | password: 'is required', | |  | }, | |  | }); | |  | } | |  |  | |  | return passport.authenticate('local', { session: false }, (err, passportUser, info) => { | |  | if(err) { | |  | return next(err); | |  | } | |  |  | |  | if(passportUser) { | |  | const user = passportUser; | |  | user.token = passportUser.generateJWT(); | |  |  | |  | return res.json({ user: user.toAuthJSON() }); | |  | } | |  |  | |  | return status(400).info; | |  | })(req, res, next); | |  | }); | |  |  | |  | //GET current route (required, only authenticated users have access) | |  | router.get('/current', auth.required, (req, res, next) => { | |  | const { payload: { id } } = req; | |  |  | |  | return Users.findById(id) | |  | .then((user) => { | |  | if(!user) { | |  | return res.sendStatus(400); | |  | } | |  |  | |  | return res.json({ user: user.toAuthJSON() }); | |  | }); | |  | }); | |  |  | |  | module.exports = router; | |  |  |