Desarrollo de Aplicaciones Web Empresariales

Exequiel Fuentes Lettura
exequiel.fuentes@ucn.cl

Información de contacto

- Exequiel Fuentes Lettura
  - Email: exequiel.fuentes@ucn.cl
  - Horario de Atención: Jueves y Viernes, bloque C

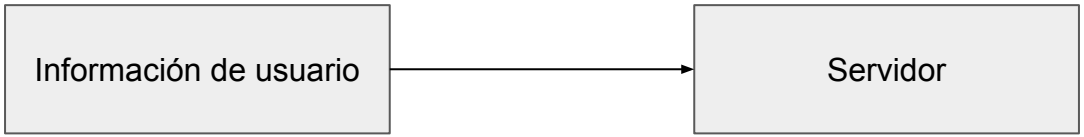
- Departamento de Ingeniería de Sistemas y Computación
  - Oficina: Y1 - 329
  - http://www.disc.ucn.cl

Arquitectura de aplicaciones Web: Auth paso a paso

Autenticación desde 0 con Node.js

- Cuando un usuario ingresa o se registra en un sitio, se sabe quien es porque ellos envían información al servidor. Se puede usar esa información para crear un nuevo registro en la base de datos.
- Pero, cómo se autentifica cuando, por ejemplo, se recarga la página?
  - Respuesta: Sesión
- Una sesión es un concepto usado para indicar que se debe mantener a los usuarios “Login”.
- Se utiliza en el cliente un mecanismo de persistencia de autenticación:
  - Cookie
  - Web Storage

Autenticación desde 0 con Node.js



- Nombre
- Apellido
- Email
- Etc

- Recuperar información de la sesión
- Verificar / Actualizar
- Procesar solicitud

- Manos a la obra...

Se requiere

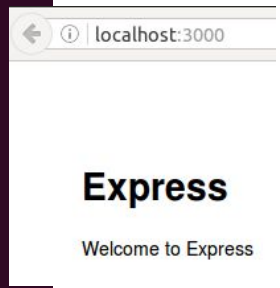
- Git (opcional)
- Node.js
- Express
- MongoDB

Paso 1: Crear el esqueleto

> sudo npm install express-generator -g
> cd workspace/dawe
> express --view=ejs authapp
> cd authapp
> npm install
> DEBUG=authapp:* npm start

Windows
> set DEBUG=myapp:* & npm start

```
exequiel@zen:dawe$ express --view=ejs authapp

   create : authapp
   create : authapp/package.json
   create : authapp/app.js
   create : authapp/public
   create : authapp/public/javascripts
   create : authapp/public/images
   create : authapp/public/stylesheets
   create : authapp/public/stylesheets/style.css
   create : authapp/routes
   create : authapp/routes/index.js
   create : authapp/routes/users.js
   create : authapp/views
   create : authapp/views/index.ejs
   create : authapp/views/error.ejs
   create : authapp/bin
   create : authapp/bin/www

   install dependencies:
     $ cd authapp && npm install

   run the app:
     $ DEBUG=authapp:* npm start
```



Instalar dependencias

```
> npm install mongoose --save
> npm install passport --save
> npm install passport-local --save
> npm install connect-flash --save
> npm install bcrypt-nodejs --save
> npm install method-override --save
> npm install express-session --save
> |
```

package.json

```
{
  "name": "authapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
```

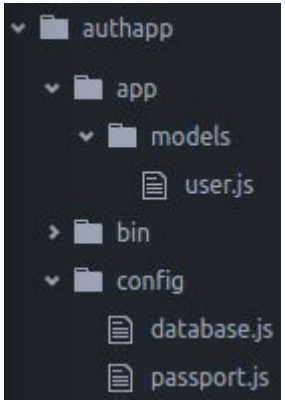
```
  "dependencies": {
    "bcrypt-nodejs": "0.0.3",
    "body-parser": "~1.15.2",
    "connect-flash": "^0.1.1",
    "cookie-parser": "~1.4.3",
    "debug": "~2.2.0",
    "ejs": "~2.5.2",
    "express": "~4.14.0",
    "express-session": "^1.14.2",
    "method-override": "^2.3.6",
    "mongoose": "^4.6.7",
    "morgan": "~1.7.0",
    "passport": "^0.3.2",
    "passport-local": "^1.0.0",
    "serve-favicon": "~2.3.0"
  }
}
```

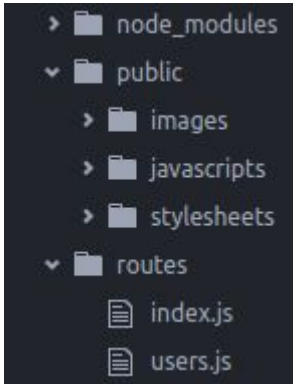
Para qué son las librerías

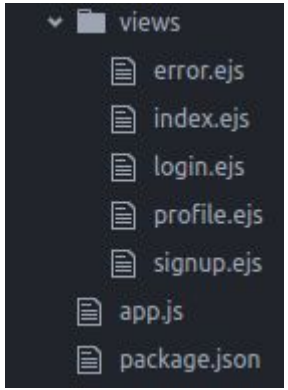
- Express es el framework.
- Ejs sirve para construir plantillas.
- Mongoose para utilizar MongoDB.
- Passport ayuda con la autenticación (se puede configurar para usar diferentes métodos como Twitter, Facebook, Google+, etc).
- Connect-flash permite pasar mensajes de sesión.
- Bcrypt-nodejs permite “encode” el password.

Crear algunos directorios y archivos

> mkdir -p app/models
> touch app/models/user.js
> mkdir config
> touch config/database.js config/passport.js
> touch views/login.ejs views/signup.ejs views/profile.ejs







Veamos app.js

```
'use strict'

// dependencies
const express = require('express');
const path = require('path');
const favicon = require('serve-favicon');
const logger = require('morgan');
const cookieParser = require('cookie-parser');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const passport = require('passport');
const session = require('express-session');
const flash = require('connect-flash');

// import routes
const index = require(path.join(__dirname, 'routes',
'index'));
//const users = require(path.join(__dirname, 'routes',
'users'));

const app = express();
```

```
// configuration
const configDB = require(path.join(__dirname, 'config',
'database'));
mongoose.connect(configDB.url, function(err, res) {
  if(err) {
    console.log('ERROR: No fue posible conectarse a la base
de datos. ' + err);
    throw err;
  }
});

require(path.join(__dirname, 'config', 'passport'));

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public',
'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
```

Veamos app.js

```
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// required for passport
app.use(session({
  secret: process.env.SESSION_SECRET ||
'thesecretisnotsecret',
  saveUninitialized: false,
  resave: false
}));
app.use(passport.initialize());
app.use(passport.session());
app.use(flash());

// routes
app.use('/', index);
//app.use('/users', users);
```

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  let err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ?
err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Veamos database.js

```
module.exports = {
  'url' : 'mongodb://localhost/dawe_db'
};
```

- No olvidar subir el servidor mongoDB:

> mongod --dbpath /var/lib/mongodb -fork --logpath /var/log/mongodb.log

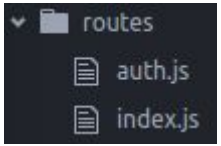
- Y para bajarlo:

> mongod --shutdown --dbpath /var/lib/mongodb

Hagamos unos cambios

- En la carpeta routes tenemos index.js y users.js
  - index.js contendrá todas las rutas que no son de autenticación.
  - user.js lo renombramos a auth.js. Este contendrá todas las rutas relacionadas con autenticación. Y luego, modificamos nuestro archivo app.js para incluir los cambios.

> mv routes/users.js routes/auth.js



- En app.js, cambiamos la línea 17:
  //const users = require(path.join(__dirname, 'routes', 'users'));
  const auth = require(path.join(__dirname, 'routes', 'auth'));

Hagamos unos cambios

- Después de importar, necesitamos incluir la ruta ahora. Entonces la línea 56 queda:
  //app.use('/users', users);
  app.use('/', auth);
- Ahora escribamos las rutas. Tenemos que editar auth.js y index.js en routes.

Editar routes/auth.js

```
'use strict'

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

// Render the login page.
router.get('/login', function(req, res, next) {
  res.render('login', {
    title: 'Autenticacion con Node.js',
    message: req.flash('loginMessage')
  });
});

// Render the signup page.
router.get('/signup', function(req, res, next) {
  res.render('signup', {
    title: 'Autenticacion con Node.js',
    message: req.flash('signupMessage')
  });
});
```

Editar routes/auth.js

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

module.exports = router;
```

Editar routes/index.js

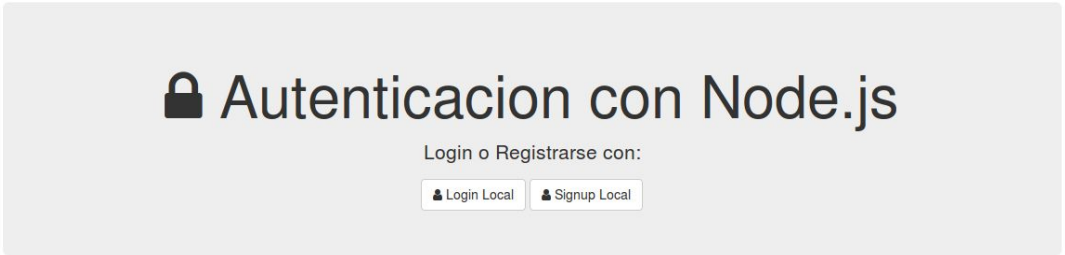
```
'use strict'

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

// Render the home page.
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Autenticacion con Node.js' });
});

// Render the profile page.
router.get('/profile', isLoggedIn, function (req, res, next) {
  res.render('profile', { user: req.user });
});

// route middleware to make sure a user is logged in
function isLoggedIn(req, res, next) {
  if (req.isAuthenticated())
    return next();
  res.redirect('/');
}

module.exports = router;
```

Editar views/index.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
```

Editar views/index.ejs

```
  <body>
    <div class="container">
      <div class="jumbotron text-center">
          <h1><span class="fa fa-lock"></span> <%= title %></h1>

          <p>Login o Registrarse con:</p>

          <a href="/login" class="btn btn-default">
            <span class="fa fa-user"></span> Login Local
          </a>
          <a href="/signup" class="btn btn-default">
            <span class="fa fa-user"></span> Signup Local
          </a>
      </div>
    </div>
  </body>
</html>
```



Editar views/login.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div class="container">
      <div class="col-sm-6 col-sm-offset-3">
        <h1><span class="fa fa-sign-in"></span> Login</h1>

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

Editar views/login.ejs

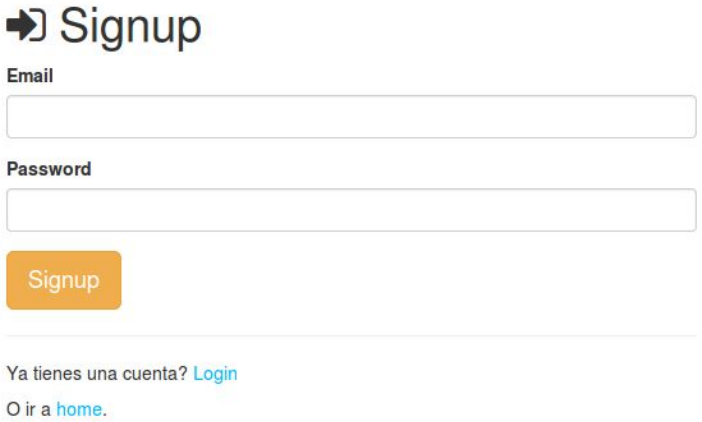
```
        <form action="/login" method="post">
            <div class="form-group">
                <label>Email</label>
                <input type="text" class="form-control" name="email">
            </div>
            <div class="form-group">
                <label>Password</label>
                <input type="password" class="form-control" name="password">
            </div>

            <button type="submit" class="btn btn-warning btn-lg">Login</button>
        </form>

        <hr>
        <p>Necesita una cuenta? <a href="/signup">Signup</a></p>
        <p>O ir a <a href="/">home</a>.</p>
      </div>
    </div>
  </body>
</html>
```

Editar views/signup.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div class="container">
      <div class="col-sm-6 col-sm-offset-3">
        <h1><span class="fa fa-sign-in"></span> Signup</h1>

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

Editar views/signup.ejs

```
        <form action="/signup" method="post">
          <div class="form-group">
            <label>Email</label>
            <input type="text" class="form-control" name="email">
          </div>
          <div class="form-group">
            <label>Password</label>
            <input type="password" class="form-control" name="password">
          </div>

          <button type="submit" class="btn btn-warning btn-lg">Signup</button>
        </form>

        <hr>
        <p>Ya tienes una cuenta? <a href="/login">Login</a></p>
        <p>O ir a <a href="/">home</a>.</p>
      </div>
    </div>
  </body>
</html>
```



Hasta ahora tenemos...

- Configurada la mayor parte del ejercicio y buena parte de la autenticación.
  - Nos basaremos en lo que tenemos para construir la siguiente parte.
- Librerías instaladas, configurada la aplicación, conectada a la base de datos, creadas las rutas y creadas las vistas.
- Los siguientes pasos son:
  - Crear el modelo.
  - Configurar passport para autenticación local.
  - Y otras cosas adicionales.

Creando el modelo

- Se creará un modelo para representar al usuario.
- Se agregará la capacidad para conectarse con múltiples cuentas en caso de ser necesario.
- Para las cuentas locales se almacenará el email y el password. Para las cuentas externas se almacenará su id, token y otra información de usuario.
- Puedes cambiar estos campos por lo que quieras en el futuro.

Editar app/models/user.js

```
'use strict'

const mongoose = require('mongoose');
const bcrypt   = require('bcrypt-nodejs');

// Se define el esquema para el usuario
const userSchema = mongoose.Schema({
  local        : {
    email      : { type: String, unique: true },
    password   : String,
  },
  facebook     : {
    id         : String,
    token      : String,
    email      : String,
    name       : String
  },
  twitter      : {
    id         : String,
    token      : String,
    displayName: String,
    username   : String
  },
```

Editar app/models/user.js

```
  google       : {
    id         : String,
    token      : String,
    email      : String,
    name       : String
  }
});

// Metodo para generar un hash para el password
userSchema.methods.generateHash = function(password) {
  return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);
};

// Metodo para validad el password, recordar que el password se guarda encriptado
// y ya no se puede desencriptar
userSchema.methods.validPassword = function(password) {
  return bcrypt.compareSync(password, this.local.password);
};

// Se deja el esquema visible a la aplicacion
module.exports = mongoose.model('User', userSchema);
```

Configurando passport

- La configuración de passport estará contenida en el archivo config/passport.js.
- Ideal que todos los archivos de configuración de la aplicación estén en la carpeta config.
  - Más limpio y conciso
- Se configurará la “estrategia” para la autenticación local, adicionalmente para facebook, twitter y google.
- Se recomienda leer: http://passportjs.org/docs

Editar config/passport.js

```
'use strict'

const path = require('path');
const LocalStrategy = require('passport-local').Strategy;

const User = require(path.join(__dirname, '..', 'app', 'models', 'user'));

module.exports = function(passport) {
  // Usado para serializar el usuario para la sesion
  passport.serializeUser(function(user, done) {
    done(null, user.id);
  });

  // Usado para deserializar el usuario
  passport.deserializeUser(function(id, done) {
    User.findById(id, function(err, user) {
      done(err, user);
    });
  });
```

Editar config/passport.js

```
passport.use('local-signup',
  new LocalStrategy({
    // by default, local strategy uses username and password, we will override with email
    usernameField: 'email',
    passwordField: 'password',
    passReqToCallback: true // allows us to pass back the entire request to the callback
  },
  function(req, email, password, done) {
    // asynchronous
    // User.findOne wont fire unless data is sent back
    process.nextTick(function() {
      // find a user whose email is the same as the forms email
      // we are checking to see if the user trying to login already exists
      User.findOne({ 'local.email':  email }, function(err, user) {
        // if there are any errors, return the error
        if (err)
          return done(err);
```

Editar config/passport.js

```
          if (user) {
            return done(null,
              false,
              req.flash('signupMessage', 'Este email ya fue tomado'));
          } else {
            // if there is no user with that email create the user
            let newUser = new User();

            newUser.local.email = email;
            newUser.local.password = newUser.generateHash(password);

            newUser.save(function(err) {
              if (err)
                  throw err;
              return done(null, newUser);
            });
          }
        });
      });
    })
  );
};
```

Se requiere hacer una refactorización

- Se debe cambiar los archivos auth.js, index.js y app.js para permitir pasar parámetros desde app.js a las funciones definidas en auth.js e index.js.
- Adicionalmente, debemos agregar una función para procesar el método post de la acción signup.
    - Esta función se agregará en auth.js dado que allí se manejan las rutas para la autenticación.

Editar routes/auth.js

```
'use strict'

module.exports = function(app, passport) {
  // Render the login page.
  app.get('/login', function(req, res, next) {
    res.render('login', {
      title: 'Autenticacion con Node.js',
      message: req.flash('loginMessage')
    });
  });

  // Render the signup page.
  app.get('/signup', function(req, res, next) {
    res.render('signup', {
      title: 'Autenticacion con Node.js',
      message: req.flash('signupMessage')
    });
  });
```

Editar routes/auth.js

```
  // Process the signup form
  app.post('/signup', passport.authenticate('local-signup', {
    successRedirect: '/profile', // redirect to the secure profile section
    failureRedirect: '/signup', // redirect back to the signup page if there is an error
    failureFlash: true // allow flash messages
  }));

  // logout
  app.get('/logout', function(req, res, next) {
    req.logout();
    res.redirect('/');
  });
};
```

Editar routes/index.js

```
'use strict'

module.exports = function(app) {
  // Render the home page.
  app.get('/', function(req, res, next) {
    res.render('index', {
      title: 'Autenticacion con Node.js'
    });
  });

  // Render the profile page.
  app.get('/profile', isLoggedIn, function (req, res, next) {
    res.render('profile', {
      title: 'Autenticacion con Node.js',
      user: req.user
    });
  });
```

Editar routes/index.js

```
  // route middleware to make sure a user is logged in
  function isLoggedIn(req, res, next) {
    // if user is authenticated in the session, carry on
    if (req.isAuthenticated())
      return next();

    // if they aren't redirect them to the home page
    res.redirect('/');
  }
};
```

Editar app.js

- Quitemos la sección “import routes”. Debería quedar de esta forma:

```
...
const flash = require('connect-flash');

const app = express();
...
```

- Pasemos “passport” como parámetro en la sección “configuration”. Debería quedar de esta forma:

```
...
});

require(path.join(__dirname, 'config', 'passport'))(passport);

// view engine setup
...
```

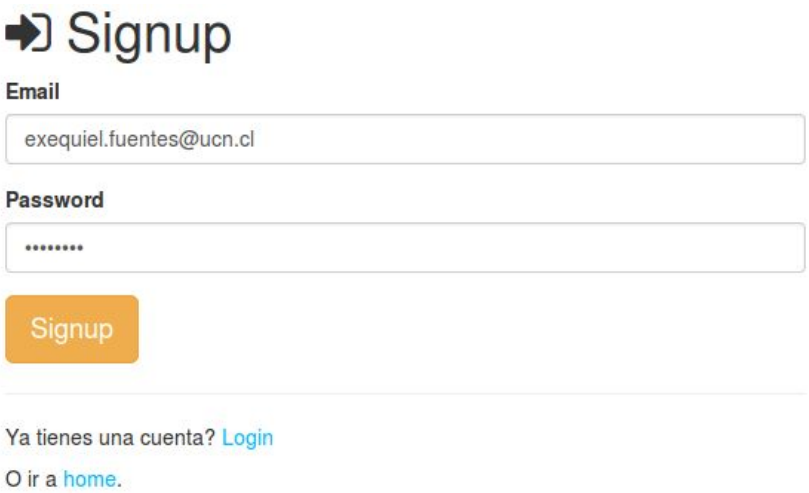
Editar app.js

- Quitemos la sección “routes”. Debería quedar de esta forma:

```
...
// import routes
require(path.join(__dirname, 'routes', 'index'))(app);
require(path.join(__dirname, 'routes', 'auth'))(app, passport);
...
```

Probemos la aplicación

- Ingresemos un usuario nuevo (la página profile debería estar vacía)



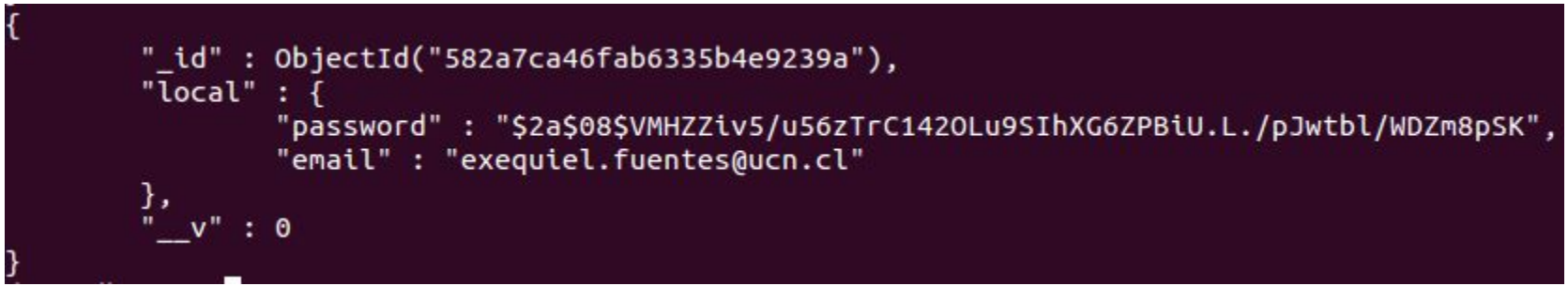
Probemos la aplicación

- Veamos los datos almacenados en la base de datos

> mongo

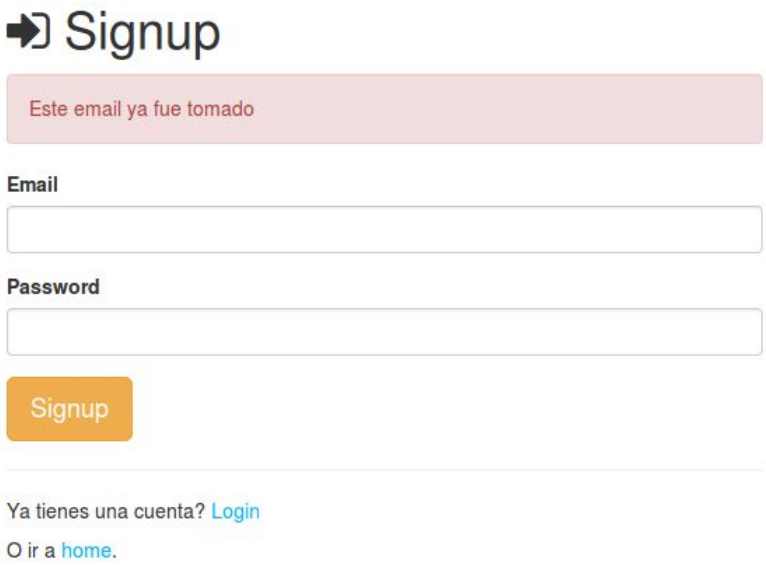
> use dawe_db

> db.users.find().pretty();



Probemos la aplicación

- Qué sucede si tratamos de ingresar el usuario nuevamente



Agregar login local, editar config/passport.js

```
passport.use('local-login',
    new LocalStrategy({
      // by default, local strategy uses username and password, we will
      // override with email
      usernameField: 'email',
      passwordField: 'password',
      passReqToCallback: true // allows us to pass back the entire request to the callback
    },
    // callback with email and password from our form
    function(req, email, password, done) {
      // find a user whose email is the same as the forms email
      // we are checking to see if the user trying to login already exists
      User.findOne({ 'local.email' :  email }, function(err, user) {
        // if there are any errors, return the error before anything else
        if (err)
            return done(err);
```

Agregar login local, editar config/passport.js

```
        // if no user is found, return the message
        if (!user)
            return done(null,
              false,
              // req.flash is the way to set flashdata using connect-flash
              req.flash('loginMessage', 'Usuario no encontrado'));

        // if the user is found but the password is wrong
        if (!user.validPassword(password))
            return done(null,
              false,
              // create the loginMessage and save it to session as flashdata
              req.flash('loginMessage', 'Password equivocado'));

        // all is well, return successful user
        return done(null, user);
      });
    })
);
```

Procesar login (post), editar routes/auth.js

```
...
  // Process the login form
  app.post('/login', passport.authenticate('local-login', {
    successRedirect : '/profile', // redirect to the secure profile section
    failureRedirect : '/login', // redirect back to the signup page if there is an error
    failureFlash : true // allow flash messages
  }));
...
```

Probemos los cambios

- Ingresamos un usuario registrado con el password equivocado
- Ingresamos un usuario no registrado

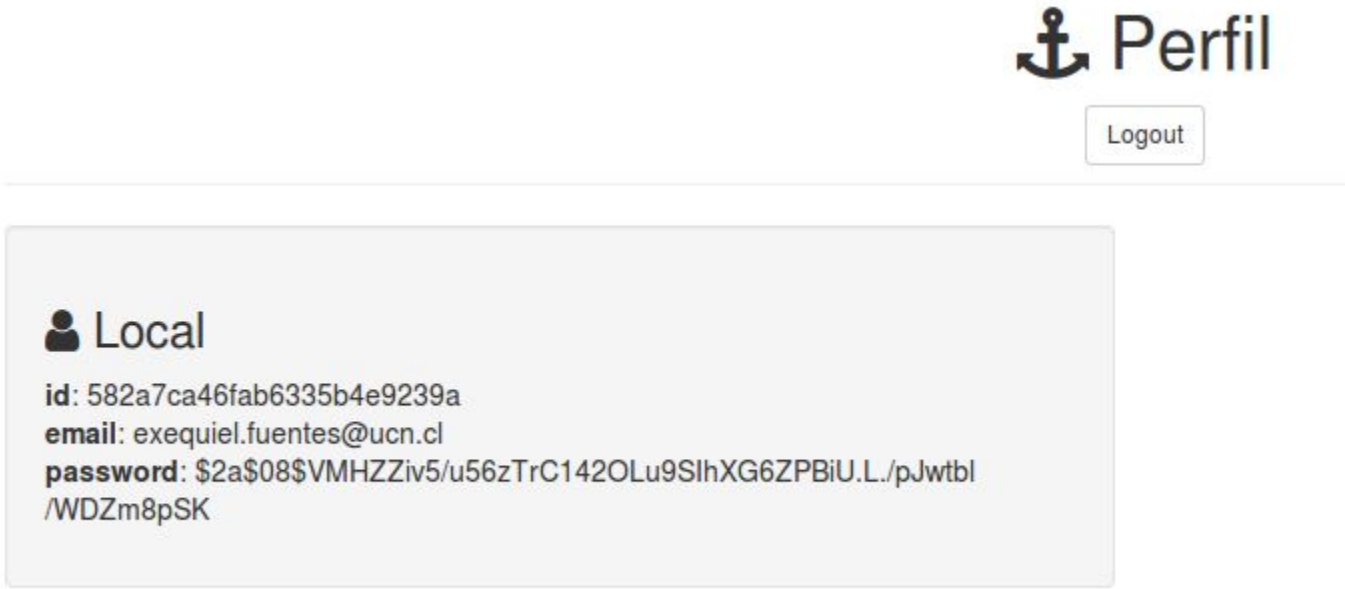
Editar views/profile.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
    <link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <div class="container">
      <div class="page-header text-center">
        <h1><span class="fa fa-anchor"></span> Perfil</h1>
        <a href="/logout" class="btn btn-default btn-sm">Logout</a>
      </div>
```

Editar views/profile.ejs

```
      <div class="row">
        <div class="col-sm-6">
          <div class="well">
            <h3><span class="fa fa-user"></span> Local</h3>
            <p>
                <strong>id</strong>: <%= user._id %><br>
                <strong>email</strong>: <%= user.local.email %><br>
                <strong>password</strong>: <%= user.local.password %>
            </p>
          </div>
        </div>
      </div>

    </div>
  </body>
</html>
```

Finalmente, probamos



Mejorar la seguridad

- Agregar proteccion para CSRF (Cross-site request forgery o falsificación de petición en sitios cruzados).
- Instalar https://github.com/expressjs/csurf :

> npm install csurf --save

- Ahora, editar app.js y agregar:

```
...
const flash = require('connect-flash');
const csrf = require('csurf');
...
app.use(flash());

//Initiate CSRF on middleware
app.use(csrf());
```

Mejorar la seguridad

- Agregamos “csrfToken: req.csrfToken()” en las funciones login y signup que responden al método ***get*** en routes/auth.js

```
...
 app.get('/login', function(req, res, next) {
    res.render('login', {
      title: 'Autenticacion con Node.js',
      message: req.flash('loginMessage'),
      csrfToken: req.csrfToken()
    });
  });
...
```

```
...
  app.get('/signup', function(req, res, next) {
    res.render('signup', {
      title: 'Autenticacion con Node.js',
      message: req.flash('signupMessage'),
      csrfToken: req.csrfToken()
    });
  });
...
```

Mejorar la seguridad

- Editamos los archivos views/login.ejs y views/signup.ejs para agregar el valor de “csrfToken” como un campo oculto. Este genera un nuevo valor cada vez que se refresca la página, el cual es verificado en el servidor.

```
<form action="/login" method="post">
  <input type="hidden" name="_csrf" value="91MKxAm8-RWMacX0G6burzsfIozIU4lcegQU">
  <div class="form-group">
    <label>Email</label>
    <input type="text" class="form-control" name="email">
  </div>


<form action="/signup" method="post">
  <input type="hidden" name="_csrf" value="QECe2zyx-17_5cBrqmlg_ZWuconITf-gO_DM">
  <div class="form-group">
    <label>Email</label>
    <input type="text" class="form-control" name="email">
  </div>
```

Editar views/login.ejs y views/signup.ejs

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

        <form action="/login" method="post">
          <input type="hidden" name="_csrf" value="<%= csrfToken %>">
          <div class="form-group">
            <label>Email</label>
...
```

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

        <form action="/signup" method="post">
          <input type="hidden" name="_csrf" value="<%= csrfToken %>">
          <div class="form-group">
            <label>Email</label>
...
```

Agregar más seguridad a la sesión

- Editar app.js y agregar:

```
…
// required for passport
app.use(session({
  secret: process.env.SESSION_SECRET || 'thesecretisnotsecret',
  saveUninitialized: false,
  resave: false,
  duration: 30 * 60 * 1000,
  activeDuration: 5 * 60 * 1000,
  httpOnly: true, // no deja a javascript acceder a las cookies
  secure: true, // cookies solo en https
  ephemeral: true, // destruye cookies cuando el browser cierra
}));
app.use(passport.initialize());
...
```

SSL/HTTPS

- Creamos una carpeta para almacenar los certificados
  - Estos son sólo para ambiente de desarrollo, luego tienes que obtener un certificado digital firmado por una entidad.
  - `> mkdir -p resources/certs`
- Crear los certificados:

```
> cd resources/certs
> openssl genrsa 1024 > file.pem
> openssl req -new -key file.pem -out csr.pem
> openssl x509 -req -days 365 -in csr.pem -signkey file.pem -out file.crt
```

```
Country Name (2 letter code) [AU]:CL
State or Province Name (full name) [Some-State]:Antofagasta
Locality Name (eg, city) []:Antofagasta
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UCN
Organizational Unit Name (eg, section) []:DISC
Common Name (e.g. server FQDN or YOUR name) []:DAWE
Email Address []:exequiel.fuentes@ucn.cl
```

```
Signature ok
subject=/C=CL/ST=Antofagasta/L=Antofagasta/O=UCN/OU=DISC/CN=DAWE/emailAddress=exequiel.fuentes@ucn.cl
Getting Private key
```

Editar bin/www

```
#!/usr/bin/env node

'use strict'

// Module dependencies.
const path = require('path');
const app = require(path.join(__dirname, '..', 'app'));
const debug = require('debug')('authapp:server');
const http = require('http');
const https = require('https');
const fs = require("fs");
const config = {
    key: fs.readFileSync(
      path.join(__dirname, '..', 'resources', 'certs',
'file.pem')),
    cert: fs.readFileSync(
      path.join(__dirname, '..', 'resources', 'certs',
'file.crt'))
};

// Get port from environment and store in Express.
const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
```

```
const ssl_port = normalizePort(process.env.SSL_PORT ||
'3443');

// Create HTTP server.
let server = http.createServer(app);

// Create HTTPS server.
let ssl_server = https.createServer(config, app);

// Listen on provided port, on all network interfaces.
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

// Listen on provided ssl_port
ssl_server.listen(ssl_port);
server.on('listening', onSSLListening);

/**
 * Normalize a port into a number, string, or false.
 */
function normalizePort(val) {
  let port = parseInt(val, 10);
```

Editar bin/www

```
  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */
function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }
```

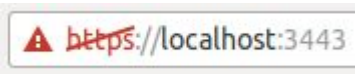
```
  let bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated
privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */
```

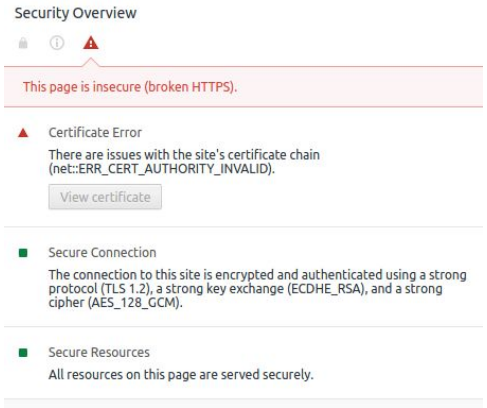
Editar bin/www

```
function onListening() {
  let addr = server.address();
  let bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}

/**
 * Event listener for HTTPS server "listening" event.
 */
function onSSLListening() {
  let addr = ssl_server.address();
  let bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('SSL Listening on ' + bind);
}
```

SSL/HTTPS

- Ahora se puede navegar por http://localhost:3000/ y https://localhost:3443/ y se obtiene los mismos resultados.
- Nótese que el certificado no es válido porque no está firmado por una autoridad de confianza.





SSL/HTTPS

- El siguiente paso es redirigir el tráfico que viene desde HTTP a HTTPS.
- Para hacer esto necesitamos modificar el middleware.
  - Editar app.js
  - Editar bin/www (otra vez)
  - Se debe cortar la función ***normalizePort*** desde bin/www y pegarla en app.js antes de la línea "***module.exports = app;***"

Editar app.js y agregar

```
...
app.use(csrf());

// Get port from environment and store in Express.
const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
const ssl_port = normalizePort(process.env.SSL_PORT || '3443');
app.set('ssl_port', ssl_port);

// import routes
app.all('*', function(req, res, next) {
  if(req.secure) {
    return next();
  };
  res.redirect('https://' + req.host + ':' + ssl_port + req.url);
});
require(path.join(__dirname, 'routes', 'index'))(app);
...
```

Editar bin/www y modificar

```
...
const config = {
    key: fs.readFileSync(
      path.join(__dirname, '..', 'resources', 'certs', 'file.pem')),
    cert: fs.readFileSync(
      path.join(__dirname, '..', 'resources', 'certs', 'file.crt'))
};

// Get port from app
const port = app.get('port')
const ssl_port = app.get('ssl_port')

// Create HTTP server.
let server = http.createServer(app);
...
```

Subir a un repositorio

- Crear los siguientes archivos:

> touch README.md .gitignore public/images/.gitkeep public/javascripts/.gitkeep

- Editar .gitignore y agregar:

```
.DS_Store
*.log

npm-debug.log
node_modules
```

- Inicializar repositorio:

> git init

Subir a un repositorio

- Crear un repositorio remoto en GitHub (o cualquier otro sitio)
- Apuntar hacia el repositorio remoto. Por ejemplo:

> git remote add origin https://github.com/<username>/authapp.git

- Editar README.md y agregar un texto informativo. Se sugiere utilizar la herramienta en línea http://dillinger.io/ para ver si el archivo está bien formateado.
- Agregar archivos:

> git add .

Subir a un repositorio

- Agregar los archivos al repositorio local:

> git commit -m "Commit inicial"

- Subir los archivos al repositorio remoto:

> git push origin master

- Código disponible en: https://github.com/efulet/authapp

¿Preguntas?