ETSIT UPM

dit UPM

Proyecto de la asignatura CORE

Desarrollo de un Blog

Tema 5: Sesiones.

CORE 2013-2014

ver: 2014-04-23

Índice

- Crear gestión de sesiones *(registro, login y logout)*:
  - Definir las rutas:
    - **`get /login`** **new** - obtener el formulario para hacer login.
    - **`post /login`** **create** - para crear la sesión.
    - **`get /logout`** **destroy** - para logout.
  - Controlador con los middlewares:
    - **`controllers/session_controller.js`**.
      - Contiene los middlewares que atiende a cada ruta.
      - También contiene un middleware para comprobar si el usuario ha hecho login o no.
        - Este middleware se añadirá a las rutas que queramos proteger con autenticación.
  - Las vistas:
    - **`views/session/new.ejs`** - *presenta el formulario de login*.

- Usar en la query un parámetro **redir** para redireccionar a la URL deseada inicialmente.
- Layout: mostrar nombre del usuario logeado y botones para login, logout y register.
- Desplegar en Heroku.
- El ejemplo está disponible en la rama **tema5**.
  - http://github.com/CORE-UPM/blog_2014

Definir las Rutas

- Queremos:
  - que los usuarios tengan que hacer **login** (*crear una sesión*) para poder publicar.
  - que los usuarios logeados puedan hacer logout (*destruyendo la sesión*).
- Para gestionar estas dos operaciones defino tres rutas en **routes/index.js**.

```
var sessionController =
                    require('../controllers/session_controller');
. . .
router.get('/login',  sessionController.new);
              // obtener el formulario a rellenar para hacer login.
router.post('/login', sessionController.create);
              // enviar formulario para crear la sesión.
router.get('/logout', sessionController.destroy);
              // destruir la sesión actual.
```

  - Para que un usuario pueda hacer login, antes debe crearse un objeto **User** con su login, password, nombre, email, etc, es decir, debe **registrarse**.
  - Los middlewares de las rutas se han creado en **controllers/session_controller.js**.

Proteger algunas rutas

- Para acceder a algunas rutas se requiere hacer antes login.
  - Ejemplo: Solo pueden publicar los usuarios que han hecho login.
- Si un usuario que no ha hecho login intenta entrar en una página que requiera estar logeado:
  - se le redirigirá automáticamente a la página con el formulario para hacer login.
    - Al URL de la página de login se añadirá una query con un parámetro, llamado **redir**, cuyo valor es el URL original al que se intentó conectar el usuario.
  - Una vez realizado el login, se redirigirá al usuario a la página a la que deseaba acceder inicialmente, cuyo URL se guardó en el parámetro **redir**.
- Crearé un middleware, llamado **loginRequired** para comprobar si el usuario ha hecho login.
  - Si el usuario ya hizo login anteriormente se continuará con los siguientes middlewares.
  - Si el usuario no ha hecho login, se le redireccionará a una pantalla de login.

- Modificamos las rutas ya existentes en **routes/index.js** para añadir el middleware **loginRequired** en las rutas que requieran login.

```
router.get('/posts', postController.index);

router.get('/posts/new', sessionController.loginRequired,
                         postController.new);

router.get('/posts/:postid([0-9]+)', postController.show);

router.post('/posts', sessionController.loginRequired,
                      postController.create);

router.get('/posts/:postid([0-9]+)/edit', sessionController.loginRequired,
                                          postController.edit);

router.put('/posts/:postid([0-9]+)', sessionController.loginRequired,
                                     postController.update);

router.delete('/posts/:postid([0-9]+)', sessionController.loginRequired,
                                        postController.destroy);
```

```
router.get('/users', userController.index);

router.get('/users/new', userController.new);

router.get('/users/:userid([0-9]+)', userController.show);

router.post('/users', userController.create);

router.get('/users/:userid([0-9]+)/edit', sessionController.loginRequired,
                                          userController.edit);

router.put('/users/:userid([0-9]+)', sessionController.loginRequired,
                                     userController.update);

router.delete('/users/:userid([0-9]+)', sessionController.loginRequired,
                                        userController.destroy);
```

controllers / session_controller.js

```
// Middleware: Se requiere hacer login.
//
// Si el usuario ya hizo login anteriormente entonces existira
// el objeto user en req.session, por lo que continuo con los demas
// middlewares o rutas.
// Si no existe req.session.user, entonces es que aun no he hecho
// login, por lo que me redireccionan a una pantalla de login.
// Guardo cual es mi url para volver automaticamente a esa url
// despues de hacer login.
//
exports.loginRequired = function (req, res, next) {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/login?redir=' + req.url);
    }
};
```

```
// Formulario para hacer login
//
// Es la tipica ruta REST que devuelve un formulario para crear
// un nuevo recurso.
// Paso como parametro el valor de redir (es una url a la que
// redirigirme despues de hacer login) que me han puesto en la
// query (si no existe uso /).
//
exports.new = function(req, res) {

    res.render('session/new',
               { redir: req.query.redir || '/'
               });
};
```

```
// Crear la sesion, es decir, hacer el login.
//
// El formulario mostrado por /login usa como action este metodo.
// Cojo los parametros que se han metido en el formulario y hago
// login con ellos, es decir crea la sesion.
// Uso el metodo autenticar exportado por user_controller para
// comprobar los datos introducidos.
// Si la autenticacion falla, me redirijo otra vez al formulario de login.
// Notar que el valor de redir lo arrastro siempre.
exports.create = function(req, res) {

    var redir = req.body.redir || '/'

    var login     = req.body.login;
    var password  = req.body.password;

    var uc = require('./user_controller');
    uc.autenticar(login, password, function(error, user) {

        if (error) {
            req.flash('error', 'Se ha producido un error: '+error);
            res.redirect("/login?redir="+redir);
            return;
        }

        // IMPORTANTE: creo req.session.user.
        // Solo guardo algunos campos del usuario en la sesion.
        // Esto es lo que uso para saber si he hecho login o no.
        req.session.user = {id:user.id, login:user.login, name:user.name};

        // Vuelvo al url indicado en redir
        res.redirect(redir);
    });
};
```

```
// Logout
//
// Para salir de la session simplemente destruyo req.session.user
//
exports.destroy = function(req, res) {

    delete req.session.user;
    req.flash('success', 'Logout.');
    res.redirect("/login");
};
```

- Para saber si el usuario ha hecho login, uso el objeto de sesión **req.session**.
  - Cuando un usuario haya realizado login con éxito, creo el objeto **req.session.user** con los datos del usuario.
  - Si no existe el objeto **req.session.user** significa que el usuario no ha hecho login.

- Nota: El soporte para **req.session** se introdujo en el tema 3 (*Mensajes de Flash*) con el módulo **express-session**.

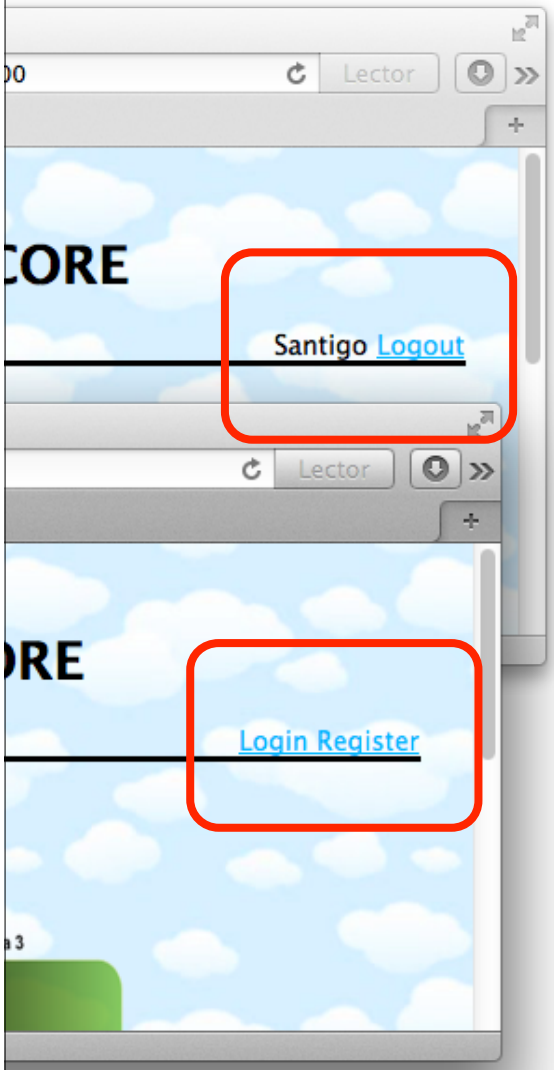
Pantalla de Login

- La pantalla para hacer login se genera con la vista **views/session/new.ejs**.

```
<h2>Login</h2>

<form method='POST' action='/login'>

  <input type='hidden' name='redir' value='<%= redir %>'>

  <div class="field">
    <label for="login">Login:</label><br />
    <input type="text" id="login" name="login" size="30" />
  </div>

  <div class="field">
    <label for="password">Password:</label><br />
    <input type="password" id="password" name="password" size="30" />
  </div>

  <div class="actions">
    <input name="commit" type="submit" value="Login" />
  </div>

</form>
```

views/session/new.ejs

Layout: Info de Login



- En el layout de la aplicación muestro:
  - Cuando el usuario ha hecho login:
    - Su **nombre**.
    - Botón de **logout**.
  - Cuando no ha hecho login:
    - Botón de **login**.
    - Botón de **registro**.

```
. . .
<div id='logininfo'>
   <% if (session.user) { %>

      <%= session.user.name %>
      <a href='/logout'> Logout </a>

   <% } else { %>

      <a href='/login?redir=/posts'> Login </a>
      <a href='/users/new'> Register </a>

   <% } %>
</div>
. . .
```

views/layout.ejs

```
#logininfo {
  float: right;
  clear: both;
}
```

public/stylesheets/style.css

- El fichero de layout necesita acceder a **req.session.user** para realizar esta tarea.
  - Dado que las vistas no tienen acceso a la variable **req**.
    - Creo en **app.js** un helper dinámico para que las vista accedan a **req.session**.
    - Hay que añadirlo antes de la definición de las rutas.

```
// Helper dinamico:

app.use(function(req, res, next) {

   // Hacer visible req.session en las vistas
   res.locals.session = req.session;

   next();
});
```

app.js

Despliegue en Heroku

Despliegue en Heroku

- Congelar cambios en git.
  - Ejecutar comandos **git add**, **git commit**, etc.
- Entrar en modo mantenimiento:

```
(local)$ heroku maintenance:on
```

- Actualizar versión en Heroku ejecutando sólo uno de estos comandos:

```
(local)$ git push -f heroku tema5:master
(local)$ git push heroku master
```

Copiar en la rama **master** de **Heroku**. El primer comando copia en contenido **local** de la rama **tema5** en la rama **master** de **Heroku**. El segundo comando copia el contenido **local** de la rama **master** en la rama **master** de **Heroku**. La opción **-f** (forzar) puede usarse para forzar la operación en caso de problemas.

- Salir del modo mantenimiento:

```
(local)$ heroku maintenance:off
```

Examen

Preguntas

- Cerrar la sesión automáticamente tras cinco minutos de inactividad.
- Si un usuario modifica su nombre, el cambio no se refleja en la cabecera de la página. Arreglar este fallo.