Proyecto de las asignatura CORE

Desarrollo de un Blog

Tema 7: Comentarios.

CORE 2013-2014

ver: 2014-05-11

Índice

- Actualizar **models** con la definición del modelo de comentario y las nuevas relaciones con las tablas de usuarios y posts.
- Definir las rutas en **routes/index.js.**
- Crear el controlador **controllers/comment_controller.js**.
- Crear las vistas **views/comments/*.ejs**.
- Retocar el middleware y la vista **show** de los **posts** para gestionar los comentarios.
  - Al mostrar un post:
    - Mostrar todos sus comentarios.
    - Mostrar un formulario para crear nuevos comentarios.
- Desplegar en Heroku.
- Disponible en la rama tema7.

  `http://github.com/CORE-UPM/blog_2014`

Notas

Este tema consiste en aplicar otra vez todos los conceptos vistos en los temas anteriores.

Objetivos: Crear comentarios de los posts existentes.

Hay que hacer los mismos pasos que en los temas anteriores, con pequeños retoques.

Una novedad de este tema el uso de rutas anidadas.

`/posts/33/comments` se refiere a todos los comentarios del post 33.

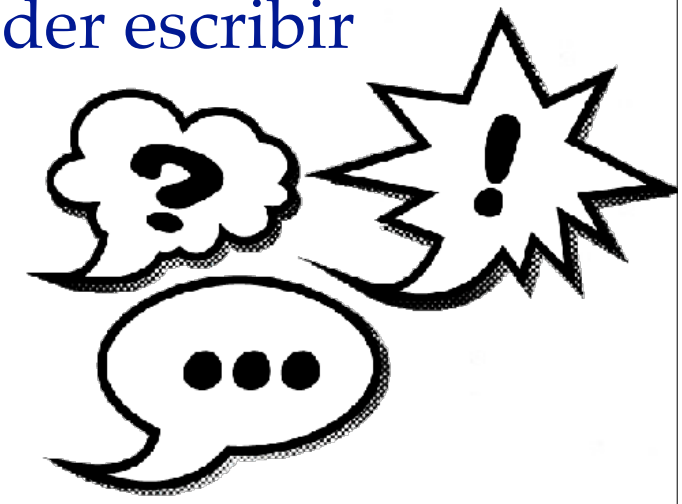
`/posts/33/comments/44` se refiere al comentario 44 del post 33.

Adicionalmente, para mejorar la usabilidad de la aplicación, no manejaremos los comentarios como una nueva sección, sino que los manejaremos desde las vistas de los posts.

- Al visualizar un post (views/posts/show.ejs):
  - Mostraremos todos los comentarios de ese post.
  - Mostraremos enlaces de edición y borrado de los comentarios.
  - Mostraremos un formulario vacío para crear nuevos comentarios.

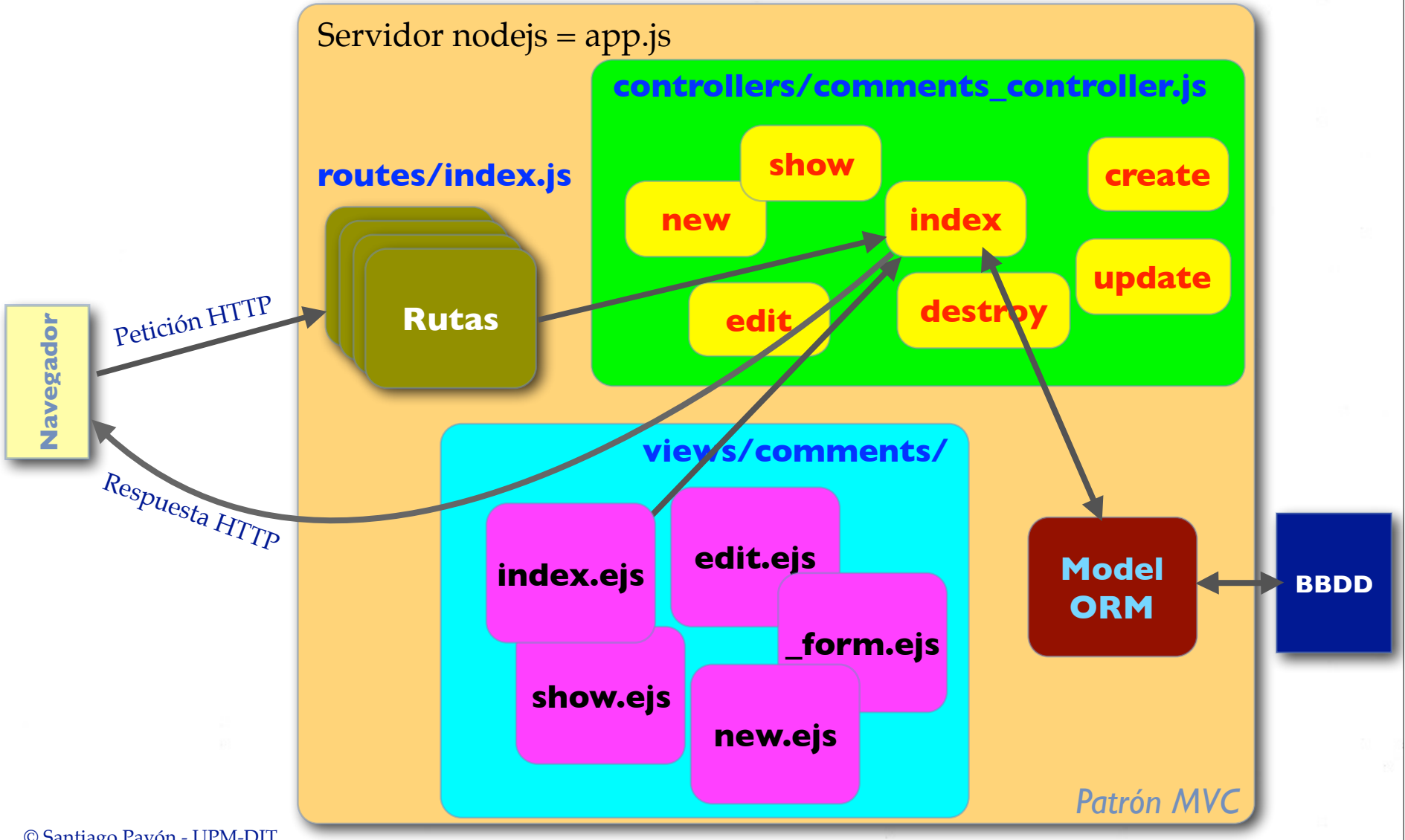
Detalles

- El usuario debe hacer login para poder escribir comentarios.



- Relaciones:
  - 1 a N entre posts y comentarios.
  - 1 a N entre usuarios y comentarios.
- Campos de los comentarios:
  - **`id`**, **`AuthorId`**, **`PostId`**, **`body`**, **`createdAt`**, **`updatedAt`**
- Formato de rutas anidadas:
  - Ejemplo: **`/posts/33/comments/44`**

Esto es lo que hay que hacer:



Definir Modelo **Comment**

- En **`models/index.js`**:
  - Importamos la definición del modelo **Comment**.
  - Declaramos las relaciones **1-a-N** conlos **Usuarios** y con los **Posts**.
    - Añadimos una restricción (`onDelete:'cascade'`) para que al borrar un **Post**, se borren automáticamente todos sus comentarios.
    - Esta restricción también se podría haber añadido en la relación User-Post para que al borrar un usuario se borrarán sus posts, pero no lo hemos hecho.
- En **`models/comment.js`** añadimos la definición del modelo **Comment**:

```
. . .

// Importar la definicion de las clases.
var Post = sequelize.import(path.join(__dirname,'post'));
var User = sequelize.import(path.join(__dirname,'user'));
var Comment = sequelize.import(path.join(__dirname,'comment'));


// Relaciones
User.hasMany(Post, {foreignKey: 'AuthorId'});
User.hasMany(Comment, {foreignKey: 'AuthorId'});
Post.hasMany(Comment);

Post.belongsTo(User, {as: 'Author', foreignKey: 'AuthorId'});
Comment.belongsTo(User, {as: 'Author', foreignKey: 'AuthorId'});
Comment.belongsTo(Post);


// Exportar los modelos:
exports.Post = Post;
exports.User = User;
exports.Comment = Comment;
```

Nombre de la clave externa

Nombre del atributo creado. Métodos creados son hasAuthor, setAuthor, getAuthor,…

Nombre de la clave externa

models/index.js

```
// Definicion del modelo Comment:

module.exports = function(sequelize, DataTypes) {
  return sequelize.define('Comment',
      { body: {
            type: DataTypes.TEXT,
            validate: {
                notEmpty: { msg: "El cuerpo del comentario no puede estar vacío" }
            }
        }
      });
}
```

Los campos **id**, **AuthorId**, **PostId**, **createdAt**, **updatedAt** no hay que definirlos.

Se crean solos por la configuración de Sequelize y por las relaciones definidas.

models/comment.js

Definir las Rutas

- Cambios en **routes/index.js**:
  - Se carga (*require*) el controlador de comentarios donde vamos a crear los middlewares que necesitemos.
  - Se definen las rutas de los comentarios:
    - Se ha optado por definirlas **anidadas** a los posts.
    - Además de los middlewares típicos, usaremos otros middlewares para:
      - Forzar a que **sólo puedan hacer comentarios los usuarios logueados**.
        - Este middleware ya existe. Se creo en el **tema 5 - Sesiones**.
          - **session_controller -> loginRequired**
      - **Solamente si el usuario logueado es el autor de un comentario, podrá editarlo o borrarlo**.
        - Este middleware es nuevo, pero es idéntico a:
          - **post_controller -> loggedUserIsAuthor**

```
var commentController =
  require('../controllers/comment_controller.js');

...

// Auto-Loading:

router.param('postid',postController.load);
router.param('userid', userController.load);
router.param('commentid', commentController.load);

…
```

// CONTINÚA

routes/index.js

```
. . .

router.get('/posts/:postid([0-9]+)/comments',
  commentController.index);

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

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

router.post('/posts/:postid([0-9]+)/comments',
   sessionController.loginRequired,
   commentController.create);

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

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

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

routes/index.js

El Controlador de Comentarios

```
var models = require('../models');

var userController = require('./user_controller');

/*
*  Auto-loading :commentid
*/
exports.load = function(req, res, next, id) {

   models.Comment
         .find(id)
         .success(function(comment) {
             if (comment) {
                 req.comment = comment;
                 next();
             } else {
                 req.flash('error', 'No existe el comentario con id='+id+'.');
                 next(new Error('No existe el comentario con id='+id+'.'));
             }
         })
         .error(function(error) {
             next(error);
         });
};
```

controllers/comment_controller.js

```
/*
* Comprueba que el usuario logeado es el author.
*/
exports.loggedUserIsAuthor = function(req, res, next) {

    if (req.session.user && req.session.user.id == req.comment.AuthorId) {
        next();
    } else {
        console.log('Op proh: Usuario logeado no es el autor del com.');
        res.send(403);
    }
};
```

```
// GET /posts/33/comments
exports.index = function(req, res, next) {

    models.Comment
        .findAll({where: {PostId: req.post.id},
                  order: [['updatedAt','DESC']],
                  include: [ {model: models.User, as: 'Author'} ]})
        .success(function(comments) {
            res.render('comments/index', {
                comments: comments,
                post: req.post
            });
        })
        .error(function(error) {
            next(error);
        });
};
```

Carga ansiosa de los autores de los comentarios.

Disponibles en **comments[i].author**.

```
// GET /posts/33/comments/66
exports.show = function(req, res, next) {

   // Buscar el autor del post
    models.User
        .find(req.post.AuthorId)
        .success(function(user) {

            // Añado el autor del post como el atributo "author".
            // Si no encuentro el autor uso el valor {}.
            req.post.author = user || {};

            // Buscar el autor del comentario
            models.User
                .find(req.comment.AuthorId)
                .success(function(user) {

                    // Añado el autor del comentario como el atributo "author".
                    // Si no encuentro el autor uso el valor {}.
                    req.comment.author = user || {};

                    res.render('comments/show', {
                        comment: req.comment,
                        post: req.post
                    });
                })
                .error(function(error) {
                    next(error);
                });
        })
        .error(function(error) {
            next(error);
        });
};
```

```
// GET /posts/33/comments/new
exports.new = function(req, res, next) {

    var comment = models.Comment.build(
        { body: 'Introduzca el texto del comentario'
        });

    res.render('comments/new', {comment: comment,
                                post: req.post,
                                validate_errors: {}
                               });
};


// GET /posts/33/comments/66/edit
exports.edit = function(req, res, next) {

    res.render('comments/edit', {comment: req.comment,
                                 post: req.post,
                                 validate_errors: {}
                                });
};
```

```
// POST /posts/33/comments
exports.create = function(req, res, next) {

    var comment = models.Comment.build(
        { body: req.body.comment.body,
          AuthorId: req.session.user.id,
          PostId: req.post.id
        });

    var validate_errors = comment.validate();
    if (validate_errors) {
        console.log("Errores de validación:", validate_errors);

        req.flash('error', 'Los datos del formulario son incorrectos.');
        for (var err in validate_errors) {
            req.flash('error', validate_errors[err]);
        };

        res.render('comments/new', {comment: comment,
                                    post: req.post,
                                    validate_errors: validate_errors});
        return;
    }

    comment.save()
        .success(function() {
            req.flash('success', 'Comentario creado con éxito.');
            res.redirect('/posts/' + req.post.id );
        })
        .error(function(error) { next(error); });
};
```

Valores obtenidos del formulario de creación, de los datos de la sesión y del autoloading del Post.

```
// PUT /posts/33/comments/66
exports.update = function(req, res, next) {

    req.comment.body = req.body.comment.body;

    var validate_errors = req.comment.validate();
    if (validate_errors) {
        console.log("Errores de validación:", validate_errors);

        req.flash('error', 'Los datos del formulario son incorrectos.');
        for (var err in validate_errors) {
            req.flash('error', validate_errors[err]);
        };

        res.render('comments/edit', {comment: req.comment,
                                     post: req.post,
                                     validate_errors: validate_errors});
        return;
    }

    req.comment.save(['body'])
        .success(function() {
            req.flash('success', 'Commentario actualizado con éxito.');
            res.redirect('/posts/' + req.post.id );
        })
        .error(function(error) {
            next(error);
        });
};
```

Solo actualizo el campo **body** en la BBDD.

```
// DELETE /posts/33/comments/66
exports.destroy = function(req, res, next) {

    req.comment.destroy()
        .success(function() {
            req.flash('success', 'Comentario eliminado con éxito.');
            res.redirect('/posts/' + req.post.id );
        })
        .error(function(error) {
            next(error);
        });
};
```

Las Vistas de los Comentarios

En la llamada **res.render** deben pasar el comentario y su post en las variables **comment** y **post**.

```
<h2>Comentario</h2>

<p>
  <em><%= comment.author && comment.author.name || 'Sin autor' %></em>
</p>

<p>
  <%= comment.updatedAt.toLocaleDateString() %>
</p>

<p><%- escapeText(comment.body) %></p>

<% if (session.user && session.user.id == comment.AuthorId) {%>

  <a href="/posts/<%= post.id %>/comments/<%= comment.id %>/edit"> Editar </a>

<% } %>

<a href="/posts/<%= post.id %>/comments"> Volver </a>
```

views/comments/show.ejs

En la llamada **res.render** deben pasar los comentarios y su post en las variables **comments** y **post**.

```
<header>
  <h2> Comentarios </h2>
</header>

<% for (var i in comments) { %>
  <article>
     <header>
        <em> <%= comments[i].author && comments[i].author.name || "Sin autor" %> </em>
        <p>
          <%= comments[i].updatedAt.toLocaleDateString() %>
        </p>
     </header>

     <p> <%- escapeText(comments[i].body) %> </p>

     <footer>
      <% if (session.user && session.user.id == comments[i].AuthorId) {%>

        <% var formname = 'fci' + i; %>
        <form method='post' action='/posts/<%= post.id %>/comments/<%= comments[i].id %>'
              id='<%= formname %>'>
          <input type='hidden' name='_method' value='delete'>
          <a href="/posts/<%= post.id %>/comments/<%= comments[i].id %>/edit"> Editar </a>
          <a href=""
             onclick="confirmarSubmit('¿Seguro que desea borrar el comentario?',
                                      '<%= formname %>'); return false"> Borrar </a>
        </form>
      <% } %>
     </footer>
  </article>
<%  }; %>

<footer>
  <nav>
<!— <a href="/posts/<%= post.id %>/comments/new"> Crear nuevo Comentario </a> -->
  </nav>
</footer>
```

views/comments/index.ejs

```
<div class='<%- validate_errors.body ? "invalid_field"
                                     : "field" %>'>

    <label for="comment_body">Contenido</label><br />
    <textarea id="comment_body" name="comment[body]"
       rows="10" cols="80"><%= comment.body %></textarea>
</div>

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

En la llamada **res.render** de las vistas que incluyen este fichero deben pasar los errores de validación y el comentario en las variables **validate_errors** y **comment**.

views/comments/_form.ejs

```
<h2>Nuevo Comentario</h2>

<form method='post' action='/posts/<%= post.id %>/comments'>

  <%- include _form.ejs %>

</form>

<A href="/posts/<%= post.id %>"> Cancelar </a>
```

En la llamada **res.render** de este fichero deben pasar el post en la variable **post**. Y además pasar todas las variables que necesite **_form.ejs**.

views/comments/new.ejs

```
<h2>Editar Comentario</h2>

<form method='post'
      action='/posts/<%= post.id %>/comments/<%= comment.id %>'>

  <input type='hidden' name='_method' value='put'>

  <%- include _form.ejs %>

</form>


<A href="/posts/<%= post.id %>"> Cancelar </a>
```

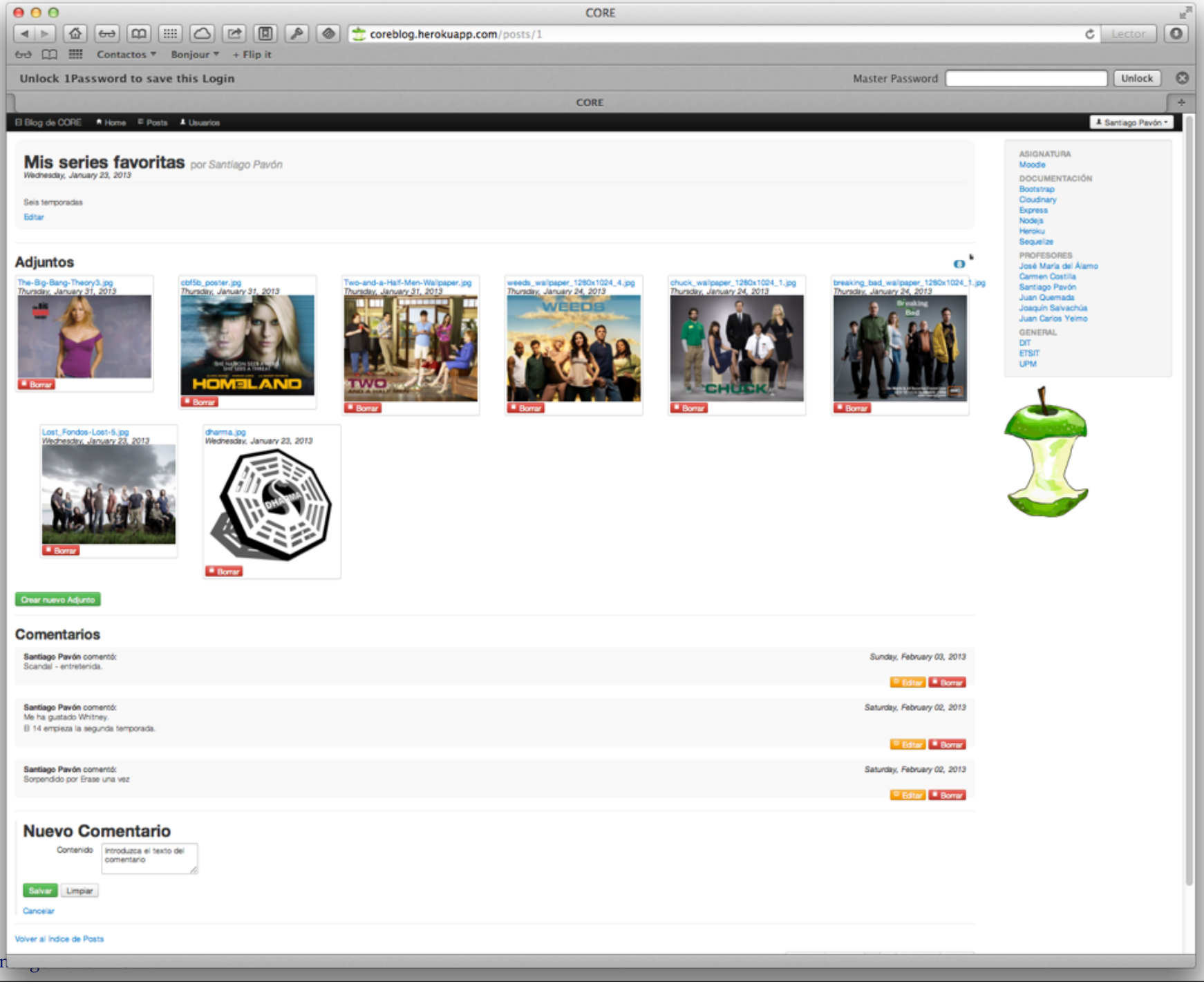
En la llamada **res.render** de este fichero deben pasar el comentario y su post en las variables **comment** y **post**.
Y además pasar todas las variables que necesite **_form.ejs**.

views/comments/edit.ejs

Post# show: Mostrar Comentarios.

- La manera más cómoda de ver los comentarios de un **Post** es mostrarlos dentro de la vista **show** del **Post**.
  - Hay que retocar dos cosas:
    - El middleware **post_controller# show** para que busque todos los comentarios del post.
      - En el renderizado de la vista **views/posts/show.ejs** pasará los comentarios en un nuevo parámetro **comments**.
    - La vista **views/posts/show.ejs** visualiza los comentarios que le ha pasado el middleware incluyendo la vista **views/comments/index.ejs**.

- También es muy cómodo tener un formulario precreado en la vista **views/posts/show** del Post para poder crear nuevos comentarios directamente desde esta vista.
  - Hay que retocar el middleware **show** y la vista **show** de los **Posts**:
    - El middleware **post_controller# show** creará un objeto **Comment** nuevo vacío que pasará a la vista **views/posts/show.ejs** en el parámetro **comment**.
    - La vista **views/posts/show.ejs** incluirá la vista **views/comments/new.ejs** para presentar el formulario que permitirá crear directamente un nuevo comentario.
      - Dado que ya tenemos este formulario disponible, queda mejor eliminar el enlace de crear nuevos formularios de **comments/index.ejs**.


CORE
coreblog.herokuapp.com/posts/1
Lector
Contactos
Bonjour
+ Flip it
Unlock 1Password to save this Login
Master Password
Unlock
CORE
Blog de CORE
Home
Posts
Usuarios
Santiago Pavón
Mis series favoritas por Santiago Pavón
Wednesday, January 23, 2013
Seis temporadas
Editar
Adjuntos
The-Big-Bang-Theory3.jpg
Thursday, January 31, 2013
cbf5b_poster.jpg
Thursday, January 31, 2013
HOMELAND
Two-and-a-Half-Men-Wallpaper.jpg
Thursday, January 31, 2013
TWO
weeds_wallpaper_1280x1024_4.jpg
Thursday, January 24, 2013
WEEDS
chuck_wallpaper_1280x1024_1.jpg
Thursday, January 24, 2013
CHUCK
breaking_bad_wallpaper_1280x1024_1.jpg
Thursday, January 24, 2013
Lost_Fondos-Lost-5.jpg
Wednesday, January 23, 2013
dharma.jpg
Wednesday, January 23, 2013
Borrar
Crear nuevo Adjunto
Comentarios
Santiago Pavón comentó:
Scandal - entretenida.
Sunday, February 03, 2013
Editar
Borrar
Santiago Pavón comentó:
Me ha gustado Whitney.
El 14 empieza la segunda temporada.
Saturday, February 02, 2013
Santiago Pavón comentó:
Sorpendido por Erase una vez
Saturday, February 02, 2013
Nuevo Comentario
Contenido
Introduzca el texto del comentario
Salvar
Limpiar
Cancelar
Volver al índice de Posts
ASIGNATURA
Moodle
DOCUMENTACIÓN
Bootstrap
Cloudinary
Express
Nodejs
Heroku
Sequelize
PROFESORES
José María del Álamo
Carmen Costilla
Santiago Pavón
Juan Quemada
Joaquín Salvachúa
Juan Carlos Yelmo
GENERAL
DIT
ETSIT
UPM

Actualizar post#show

```
// GET /posts/33
exports.show = function(req, res, next) {

    // Buscar el autor
    models.User.find(req.post.AuthorId)
       .success(function(user) {

            // Si encuentro al autor lo añado como el atributo author,
            // si no lo encuentro añado {}.
            req.post.author = user || {};

            // Buscar los comentarios del post
            models.Comment
                 .findAll({where: {PostId: req.post.id},
                           order: [['updatedAt','DESC']],
                           include: [{ model: models.User, as: 'Author' }]
                 })
                 .success(function(comments) {
                    var new_comment = models.Comment.build({
                        body: 'Introduzca el texto del comentario' });
                    res.render('posts/show', {
                        post: req.post,
                        comments: comments,
                        comment: new_comment,
                        validate_errors: {} });
                 })
                 .error(function(error) {next(error);});
        })
        .error(function(error) { next(error); });
};
```

Comentarios del Post.

Comentario vacío.

controllers/post_controller.js

```
<h2>Post</h2>

<article>
    <p>
      <b><%= post.title %></b>
      <br />
      by
      <em><%= post.author && post.author.name || 'Sin autor' %></em>
    </p>

    <p>
      <%= post.updatedAt.toLocaleDateString() %>
    </p>

    <p><%- escapeText(post.body) %></p>


    <% if (session.user && session.user.id == post.AuthorId) {%>

       <a href="/posts/<%= post.id %>/edit"> Editar </a>

    <% } %>
</article>

<hr />
<%- include ../comments/index.ejs %>
<hr />

<% if (session.user) { %>
   <blockquote>
      <%- include ../comments/new.ejs %>
   </blockquote>
   <hr />
<% } %>

<a href="/posts"> Volver al índice de Posts</a>
```

Mostrar los comentarios del Post.

Mostrar formulario para crear un nuevo comentario.

Solo se muestra su el usuario hizo login.

views/posts/show.ejs

Post# destroy: Borrar los Comentarios.

- Para no dejar comentarios huérfanos es necesario borrar los comentarios de los posts que borremos.
  - Hay que retocar el método **`destroy`** de **post_controller**.
    - Lo implementaremos creando un **`Sequelize.Utils.QueryChainer()`** al que añadiremos las operaciones de borrado del post y de sus comentarios.
- Alternativa: Añadir la opción **onDelete** al modelo **Post**.

Notas

Sequelize Query-Chainer

Se usa para realizar varias operaciones de una vez.
Pueden ejecutarse de forma concurrente o secuencial.

Una vez que se han realizado todas las operaciones se genera un evento **success**.

Actualizar post#destroy

```
// DELETE /posts/33
exports.destroy = function(req, res, next) {

    var Sequelize = require('sequelize');
    var chainer = new Sequelize.Utils.QueryChainer

    // Obtener los comentarios:
    req.post.getComments()
       .success(function(comments) {
           for (var i in comments) {
                // Eliminar un comentario:
                chainer.add(comments[i].destroy());
           }
           // Eliminar el post:
           chainer.add(req.post.destroy());
           // Ejecutar el chainer:
           chainer.run()
            .success(function(){
                 req.flash('success', 'Post eliminado con éxito.');
                 res.redirect('/posts');
            })
            .error(function(errors){ next(errors[0]); })
       })
       .error(function(error) { next(error); });
};
```

controllers/
post_controller.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 tema7:master
(local)$ git push heroku master
```

Copiar en la rama **master** de **Heroku**. El primer comando copia en contenido **local** de la rama **tema7** 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