



POLITÉCNICA

ETSIT
UPM

dit
UPM

Proyecto de la asignatura CORE **Desarrollo de un Blog**

Tema 4: Crear Usuarios.

CORE 2013-2014

ver: 2014-04-30

Índice

- Objetivo del tema: Añadir usuarios.
 - Diseño de los recursos REST de usuarios.
 - Implementar las rutas, modelo, vistas, controlador.
 - Añadir gestión de los Passwords.
- Desplegar en Heroku.
- El ejemplo está disponible en la rama tema4.
 - http://github.com/CORE-UPM/blog_2014



El objetivo de este tema es añadir usuarios a la aplicación. En un futuro los usuarios serán los autores de los posts y comentarios publicados.

El trabajo a realizar en este tema es idéntico al realizado con los posts, pero en vez de la palabra **post** usaremos la palabra **user**.

Crearemos las rutas REST típicas, definiéndolas a **routes/index.js**.

Los middlewares que se ejecutarán para cada ruta se crearán en el módulo **controllers/user_controller.js**.

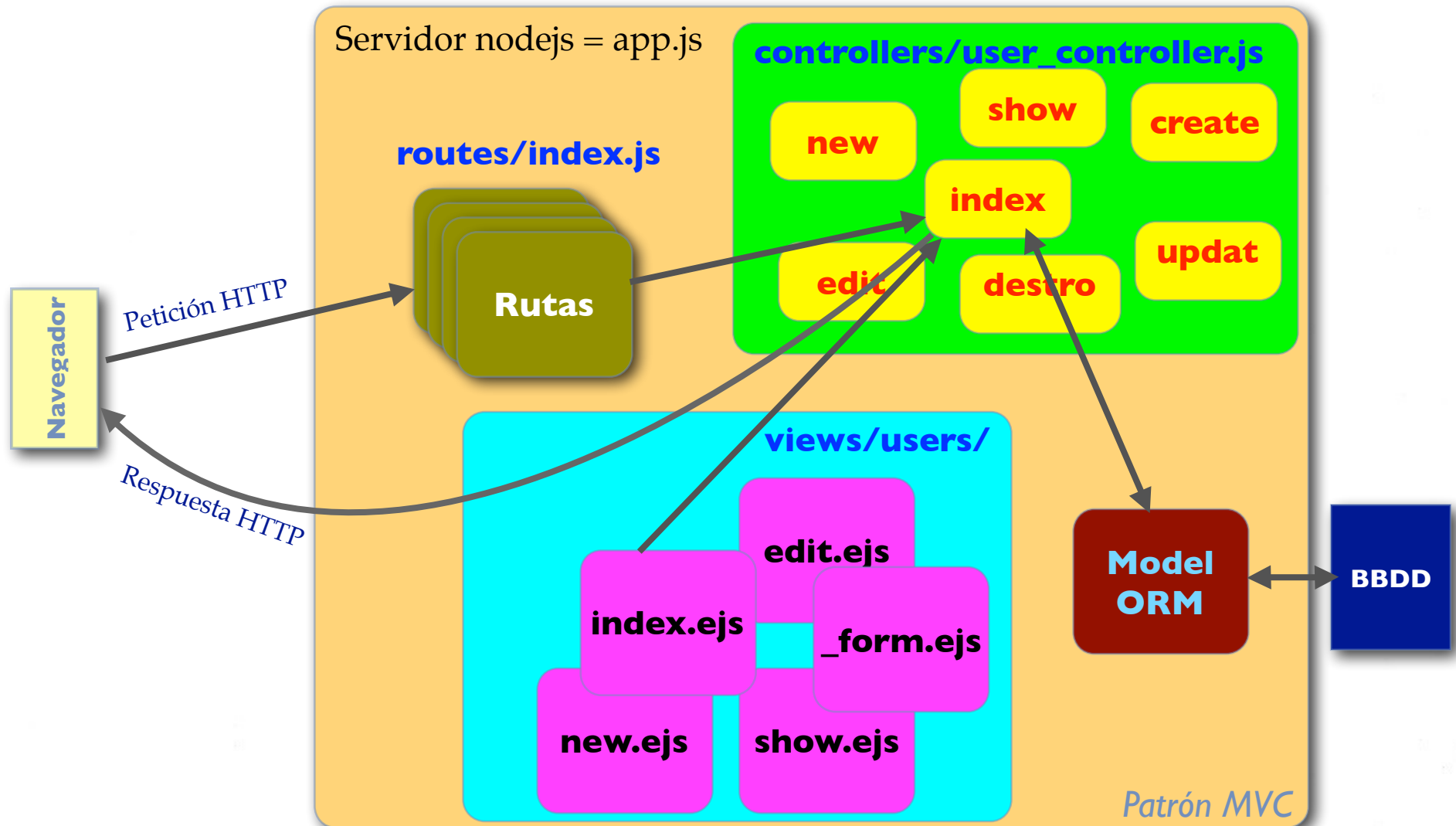
La definición del modelo de usuarios se añadirá a **models/index.js** y **models/user.js**.

Las vistas se crearán en **views/users/*.ejs**.

Objetivo

- Añadir usuarios al blog.
 - Los usuarios serán los autores de los posts y comentarios.
 - Nota: La primera parte de este tema consiste principalmente en duplicar los ficheros y sentencias que se han creado para los Posts
 - Renombrando post por user

Esto es lo que hay que hacer:



Diseñar Rutas REST para Users

Método HTTP	URL	Acción
GET	<code>/users</code>	<code>UserController.index</code>
GET	<code>/users/new</code>	<code>UserController.new</code>
GET	<code>/users/:userid</code>	<code>UserController.show</code>
POST	<code>/users</code>	<code>UserController.create</code>
GET	<code>/users/:userid/edit</code>	<code>UserController.edit</code>
PUT	<code>/users/:userid</code>	<code>UserController.update</code>
DELETE	<code>/users/:userid</code>	<code>UserController.destroy</code>

Crear las Rutas

- Editamos el fichero **routes/index.js**.
- Requerimos el módulo controlador de los Users que contiene los middlewares.
 - Añadimos:

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

- Y definimos las rutas añadiendo:

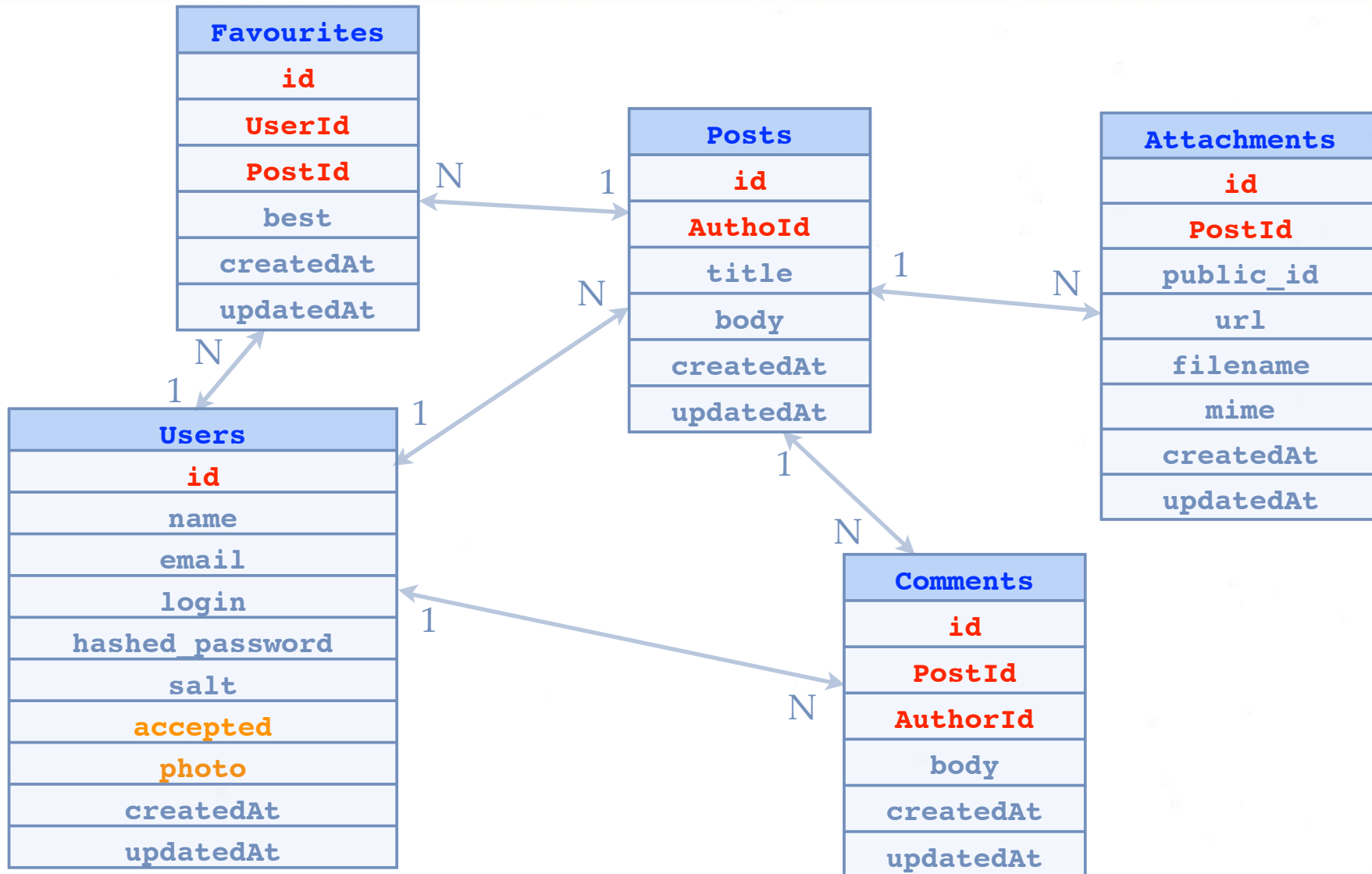
```
router.param('userid', userController.load);

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

Esquema de la Tabla Users

- La tabla **Users** tendrá las siguientes columnas:
 - **id**
 - Clave primaria.
 - En un entero que se autoincrementa automáticamente.
 - **name**
 - String con el nombre completo del usuario.
 - **email**
 - String con la dirección de correo electrónico del usuario.
 - **login**
 - String con el login del usuario. **No puede estar repetido.**
 - **hashed_password**
 - String con el hash del password del usuario.
 - **salt**
 - String con el salt usado para crear el hash del password.
 - Y también: **createdAt** y **updatedAt**.

Bases de Datos



Definir el Modelo User

- Añadimos en **models/index.js** las siguientes líneas:

```
...  
var Post = sequelize.import(path.join(__dirname, 'post'));  
var User = sequelize.import(path.join(__dirname, 'user'));  
...  
exports.Post = Post;  
exports.User = User;  
...  
sequelize.sync();
```

- El modulo **models** exporta el modelo **User**, que importamos desde el fichero **models/user.js**.

```

var path = require('path');

var Sequelize = require('sequelize');

// Configurar Sequelize para usar SQLite. Uso una expresion regular para extraer
// los valores de acceso a la base de datos
var vals = process.env.DATABASE_URL.match(/(.*?)\:\/\/\/(.*?)\:(.*?)@(.*?)\:(.*?)\/(.*?)\/);

var DATABASE_PROTOCOL = vals[1];
var DATABASE_DIALECT = vals[1];
var DATABASE_USER = vals[2];
var DATABASE_PASSWORD = vals[3];
var DATABASE_HOST = vals[4];
var DATABASE_PORT = vals[5];
var DATABASE_NAME = vals[6];

var sequelize = new Sequelize(DATABASE_NAME, DATABASE_USER, DATABASE_PASSWORD,
    { dialect: DATABASE_DIALECT,
      protocol: DATABASE_PROTOCOL,
      port: DATABASE_PORT,
      host: DATABASE_HOST,
      storage: process.env.DATABASE_STORAGE, // solo local en .env
      omitNull: true // para postgres
    });

// Importar la definicion de las clases.
// La clase Xxxx se importa desde el fichero xxxx.js.
var Post = sequelize.import(path.join(__dirname, 'post'));
var User = sequelize.import(path.join(__dirname, 'user'));

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

// Crear las tablas en la base de datos que no se hayan creado aun.
// En un futuro lo haremos con migraciones.
sequelize.sync();

```

models/index.js

```
// Definicion de la clase User:
```

```
module.exports = function(sequelize, DataTypes) {  
  return sequelize.define('User',  
    { login: {  
      type: DataTypes.STRING,  
      validate: {  
        notEmpty: { msg: "El campo login no puede estar vacío" }  
      }  
    },  
    name: {  
      type: DataTypes.STRING,  
      validate: {  
        notEmpty: { msg: "El campo nombre no puede estar vacío" }  
      }  
    },  
    email: {  
      type: DataTypes.TEXT,  
      validate: {  
        isEmail: { msg: "El formato del email introducido no es correcto" },  
        notEmpty: { msg: "El campo email no puede estar vacío" }  
      }  
    },  
    hashed_password: {  
      type: DataTypes.STRING  
    },  
    salt: {  
      type: DataTypes.STRING  
    }  
  });  
}
```

models/user.js

Las vistas

- Los ficheros EJS con las vistas los crearé en el directorio **views/users**.
 - **index.ejs**
 - Muestran los usuarios que el controlador le pasa en el array **users**.
 - **new.ejs**
 - Muestra un formulario para crear un nuevo usuario. Se pasa un objeto **user** nuevo.
 - **edit.ejs**
 - Muestra un formulario para editar el usuario que el controlador le pasa en la variable **user**.
 - **_form.ejs**
 - Vista parcial usada por **new.ejs** y **edit.ejs** con los campos del formulario. Hereda el objeto **user** que le han pasado como parámetro a **new.ejs** y a **edit.ejs**.
 - **show.ejs**
 - Muestra el contenido del usuario que el controlador le pasa en la variable **user**.

```

<header>
  <h2> Usuarios </h2>
</header>

<article>
<table>
  <tr>
    <th>Login</th>
    <th>Name</th>
    <th>Email</th>
    <th></th>
  </tr>

  <% for (var i in users) { %>
    <tr>
      <td> <%= users[i].login %> </td>
      <td> <%= users[i].name %> </td>
      <td> <%= users[i].email %> </td>
      <td>
        <% var formname = 'fui' + i; %>
        <form method='post' action='/users/<%= users[i].id %>' id='<%= formname %>'>
          <input type='hidden' name='_method' value='delete'>
          <a href="/users/<%= users[i].id %>"> Ver </a>
          <a href="/users/<%= users[i].id %>/edit"> Editar </a>
          <a href="" onclick="confirmarSubmit('¿Seguro que desea borrar el usuario?',
            '<%= formname %>'); return false"> Borrar </a>
        </form>
      </td>
    </tr>
  <% }; %>
</table>
</article>

<footer>
  <nav> <a href="/users/new"> Crear nuevo Usuario </a> </nav>
</footer>

```

users: Array donde me pasan los usuarios a mostrar.

views/users/index.ejs


```
<h2>Usuario</h2>
```

```
<p>  
  <b>Login:</b>  
  <%= user.login %>  
</p>
```

```
<p>  
  <b>Name:</b>  
  <%= user.name %>  
</p>
```

```
<p>  
  <b>Email:</b>  
  <%= user.email %>  
</p>
```

```
<a href="/users/<%= user.id %>/edit"> Edit </a>  
<a href="/users"> Back </a>
```

user: Variable donde me pasan el usuario a mostrar.



views / users / show.ejs

```
<h1>Nuevo Usuario</h1>
```

```
<form method='post' action='/users'>
```

```
  <%- include _form.ejs %>
```

```
</form>
```

```
<A href="/users"> Volver </a>
```

views / users / new.ejs

user: Variable donde me pasan el usuario a editar.

```
<h1>Editar Usuario</h1>
```

```
<form method='post' action='/users/<%= user.id %>'>
```

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

```
  <%- include _form.ejs %>
```

```
</form>
```

```
<a href="/users"> Volver </a>
```

views/users/edit.ejs

```

<div class='<%- validate_errors.login ? "invalid_field" : "field" %>'>
  <% if ( ! user.id) { %>
    <label for="user_login">Login:</label><br />
    <input type="text" id="user_login" name="user[login]" size="30"
      value='<%= user.login %>' />
  <% } else { %>
    Login: <%= user.login %>
  <% } %>
</div>

```

Un objeto sin **id** es nuevo.

Los objetos con **id** son los que he sacado de la base de datos.

```

<div class='<%- validate_errors.name ? "invalid_field" : "field" %>'>
  <label for="user_name">Nombre</label><br />
  <input type="text" id="user_name" name="user[name]" size="30"
    value='<%= user.name %>' />
</div>

```

```

<div class='<%- validate_errors.email ? "invalid_field" : "field" %>'>
  <label for="user_email">Email</label><br />
  <input type="text" id="user_email" name="user[email]" size="30"
    value='<%= user.email %>' />
</div>

```

user: Variable donde me pasan el usuario a editar.

```

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

```

(Versión sin Passwords) `views/users/_form.ejs`

El controlador: controllers/user_controller.js

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

/*
 * Autoloading :userid
 */
exports.load = function(req, res, next, id) {
  models.User
    .find(id)
    .success(function(user) {
      if (user) {
        req.user = user;
        next();
      } else {
        req.flash('error', 'No existe el usuario con id='+id+'.');
        next('No existe el usuario con id='+id+'.');
      }
    })
    .error(function(error) {
      next(error);
    });
};
```

(Versión sin Passwords)

```
// GET /users
exports.index = function(req, res, next) {
  models.User
    .findAll({order: ['name']})
    .success(function(users) {
      res.render('users/index', {
        users: users
      });
    })
    .error(function(error) {
      next(error);
    });
};
```

(Versión sin Passwords)


```
// GET /users/33
exports.show = function(req, res, next) {
  res.render('users/show', {user: req.user});
};

// GET /users/new
exports.new = function(req, res, next) {

  var user = models.User.build(
    { login: 'Tu login',
      name: 'Tu nombre',
      email: 'Tu email'
    });

  res.render('users/new', {user: user,
                           validate_errors: {} });
};

// GET /users/33/edit
exports.edit = function(req, res, next) {

  res.render('users/edit', {user: req.user,
                           validate_errors: {} });
};
```

(Versión sin Passwords)

```

// POST /users
exports.create = function(req, res, next) {

  var user = models.User.build(
    { login: req.body.user.login,
      name: req.body.user.name,
      email: req.body.user.email,
      hashed_password: '',
      salt: ''
    });

  // El login debe ser unico:
  models.User.find({where: {login: req.body.user.login}})
    .success(function(existing_user) {
      if (existing_user) {
        console.log("Error: El usuario \""+ req.body.user.login +"\" ya existe: "+existing_user.values);
        req.flash('error', "Error: El usuario \""+ req.body.user.login +"\" ya existe.");
        res.render('users/new', { user: user,
          validate_errors: {
            login: 'El usuario \''+ req.body.user.login +'\' ya existe.'
          }
        });
      } else {
        var validate_errors = user.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('users/new', {user: user,
            validate_errors: validate_errors});
          return;
        }
        user.save()
          .success(function() {
            req.flash('success', 'Usuario creado con éxito. ');
            res.redirect('/users');
          })
          .error(function(error) {next(error);});
      }
    })
    .error(function(error) {next(error);});
};

```

(Versión sin Passwords)

```

// PUT /users/33
exports.update = function(req, res, next) {

    // req.user.login = req.body.user.login; // No se puede editar.
    req.user.name = req.body.user.name;
    req.user.email = req.body.user.email;

    var validate_errors = req.user.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('users/edit', {user: req.user,
                                   validate_errors: validate_errors});
        return;
    }

    req.user.save(['name', 'email'])
        .success(function() {
            req.flash('success', 'Usuario actualizado con éxito. ');
            res.redirect('/users');
        })
        .error(function(error) {next(error)});
};

```

No compruebo que el login sea único, ya que no lo cambio.

(Versión sin Passwords)

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

  req.user.destroy()
    .success(function() {
      req.flash('success', 'Usuario eliminado con éxito.');
```

```
      res.redirect('/users');
```

```
    })
    .error(function(error) {
      next(error);
    });
};
```

(Versión sin Passwords)

Barra de Navegación

- Se ha ampliado la barra de navegación del layout (**views/layout.ejs**).

```
<!DOCTYPE html>
<html>
<head>
  ...
</head>

<body>
  <header>
    <h1> Computación en REd - CORE</h1>
    <nav>
      <a href="/">Home</a>
      <a href="/posts">Posts</a>
      <a href="/users">Usuarios</a>
      <a href="/creditos.html">Créditos</a>
    </nav>
  </header>

  <section>
    ...
```

Passwords



Para Soportar Passwords

- Cambios a realizar:
 - Los formularios (`_form.ejs`) deben pedir dos veces el password.
 - Guardar el password.
 - Primero veremos cómo vamos a guardar los passwords
 - Crear en el controlador de usuarios métodos para generar un salt aleatorio, encriptar un password y autenticar a un usuario.
 - Cambiar el método **create** del controlador de usuarios para generar los valores de los campos `salt` y `hashed_password` que se guardarán en la base de datos.
 - Comprobando que el password introducido no sea vacío.
 - Cambiar el método **update** del controlador de usuarios para que no se cambie el password si no se introduce un nuevo valor para éste.
 - Se permite actualizar el nombre y el email de un usuario si tocar su password.
 - Cambiar el método de Auto-Load (`load`) de los usuarios para no meter los campos de **salt** y **hashed_password** en el objeto `req.user` creado.
 - Si se desea (opcional), en la llamada a **findAll** que hace el método **index**, restringir los campos que se obtienen.

Pedir el Password

- Cambios en los formularios:
 - En el formulario de creación de un nuevo usuario se deberá introducir el password del usuario.
 - En el formulario de edición se introducirá el password sólo cuando se quiera modificarlo.
- Para evitar errores, el password se confirmará en los formularios usando dos campos de password.
 - Se comprobará que los valores introducidos son iguales ejecutando una función javascript en el propio navegador.
- Cambiamos **views/users/_form.ejs**.

views/users/_form.ejs

Javascript ejecutado
en el cliente

Sólo existe en el formulario de
creación de un usuario nuevo.

```
<script>  
function confirmPassword() {  
  var pw1 = document.getElementById('user_password').value;  
  var pw2 = document.getElementById('user_confirm_password').value;
```

```
<% if ( ! user.id) { %>  
  if (!pw1 || !pw2) {  
    alert('Debe rellenar los campos de Password y Confirmación.');
```

```
    return false;  
  }  
  <% } %>  
  
  if (pw1 === pw2) {  
    return true;  
  } else {  
    alert('Los passwords introducidos no coinciden.');
```

```
    return false;  
  }  
}  
</script>
```

El password y la confirmación
deben ser iguales

continúa ...

```
<!-- CAMPOS YA EXISTENTES (login, name, email) AQUI -->
```

```
. . .
```

```
<div class="field">  
  <label for="user_password">Password:</label><br />  
  <input type="password" id="user_password"  
    name="user[password]" size="30" value="" />  
</div>
```

El valor del password se pasa al servidor en **user[password]**

```
<div class="field">  
  <label for="user_confirm_password">Confirm Password:</label><br />  
  <input type="password" id="user_confirm_password" size="30" value="" />  
</div>
```

```
<div class="actions">  
  <input name="commit" type="submit" value="Salvar"  
    onclick='return confirmPassword()' />  
</div>
```

Sólo realiza el submit si los dos passwords son iguales

¿Cómo se Guarda el Password?

- En la base de datos no se guarda el password en claro.
- Se guardan dos valores:
 - **salt**: string aleatorio usado para generar el valor del campo **hashed_password**.
 - **hashed_password**: un hash de **password+salt**.
 - Se crea mezclando el password en claro con el salt, ejecutando un SHA1 digest, y devolviendo 40 caracteres hexadecimales.

Autenticar

- Creamos en el controlador de usuarios métodos para generar un salt aleatorio, encriptar passwords en claro, y autenticar usuarios:

- **createNewSalt()**

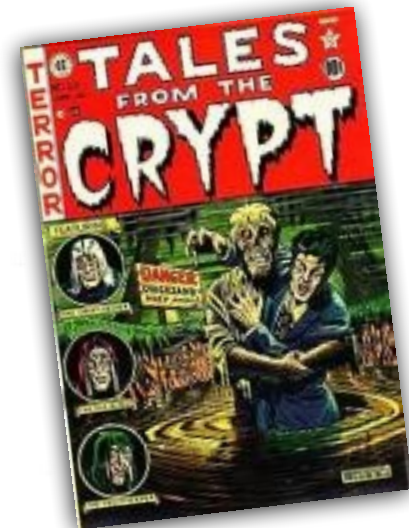
- Crea un string aleatorio para usar como salt.

- **encriptarPassword(password, salt)**

- Encripta un password en claro.
 - Mezcla el password en claro con el salt proporcionado,
 - ejecuta un SHA1 digest,
 - y devuelve 40 caracteres hexadecimales.

- **autenticar(login, password, callback)**

- Autenticar un usuario.
 - Busca el usuario con el login dado en la base de datos y comprueba su password.
 - Si todo es correcto ejecuta **callback(null,user)**.
 - Si la autenticación falla o hay errores se ejecuta **callback(error)**.
- Este método lo exportaremos en el módulo `user_controller.js`.
 - Se usará en un futuro para hacer **login** y crear una sesión.



- (Posible mejora: pensar en meter estos métodos como una expansión del modelo User, o redefinir los métodos getter y setter.)

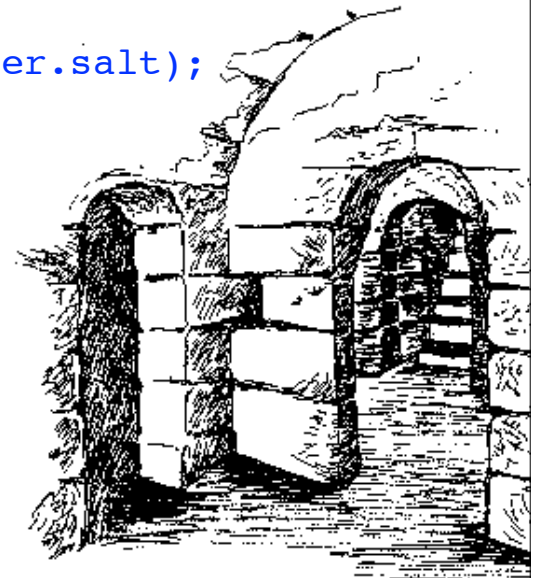
user_controller.js

```
var crypto = require('crypto');

function createNewSalt() {
    return Math.round((new Date().valueOf() * Math.random())) + '';
};

function encriptarPassword(password, salt) {
    return crypto.createHmac('sha1', salt).update(password).digest('hex');
};

exports.autenticar = function(login, password, callback) {
    models.User.find({where: {login: login}})
        .success(function(user) {
            if (user) {
                var hash = encriptarPassword(password, user.salt);
                if (hash === user.hashed_password) {
                    callback(null, user);
                    return;
                }
            }
            callback(new Error('Password erróneo.'));
        })
        .error(function(err) {next(err);});
};
```



Salvar Password

- Método **create**: Al salvar un nuevo usuario, genero los valores **salt** y **hashed_password** antes de llamar a **save**.
 - borrar las líneas que ponían estos valores a "".
 - también compruebo que el password no sea vacío.
- Método **update**: Al editar un usuario, sólo actualizo los campos **salt** y **hashed_password** si el usuario metió un nuevo password.
 - Mejora pendiente: para permitir cambiar un password debería pedirse el valor del password antiguo.

user_controller.js

```
exports.create = function(req, res, next) {
```

```
  var user = models.User.build(  
    { login: req.body.user.login,  
      name: req.body.user.name,  
      email: req.body.user.email,  
      hashed_password: '',  
      salt: ''  
    });
```

...

```
  // El password no puede estar vacío  
  if ( ! req.body.user.password ) {  
    req.flash('error', 'El campo Password es obligatorio.');
```

```
    res.render('users/new', {user: user,  
                             validate_errors: {  
                               password: 'El campo Password es obligatorio.'});  
    return;  
  }
```

```
  user.salt = createNewSalt();  
  user.hashed_password = encriptarPassword(req.body.user.password, user.salt);
```

```
  user.save()
```

...

```
}
```

```

exports.update = function(req, res, next) {

    // req.user.login = req.body.user.login; // No se puede editar.
    req.user.name = req.body.user.name;
    req.user.email = req.body.user.email;

    var validate_errors = req.user.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('users/edit', {user: req.user,
            validate_errors: validate_errors});

        return;
    }

    var fields_to_update = ['name', 'email'];

    // ¿Cambio el password?
    if (req.body.user.password) {
        req.user.salt = createNewSalt();
        req.user.hashed_password = encriptarPassword(req.body.user.password,
            req.user.salt);

        fields_to_update.push('salt');
        fields_to_update.push('hashed_password');
    }

    req.user.save(fields_to_update)
        .success(function() {
            req.flash('success', 'Usuario actualizado con éxito.');
```

```

            res.redirect('/users');
        })
        .error(function(error) {next(error);});
};

```

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 tema4:master
```

```
(local)$ git push heroku master
```

Copiar en la rama `master` de `Heroku`. El primer comando copia en contenido `local` de la rama `tema4` 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

- Para que un usuario pueda cambiar su password, debe introducir su password antiguo.



