Proyecto de la asignatura CORE

Desarrollo de un Blog

Tema 3: Añadir Mensajes Flash.

CORE 2013-2014

ver: 2014-04-23

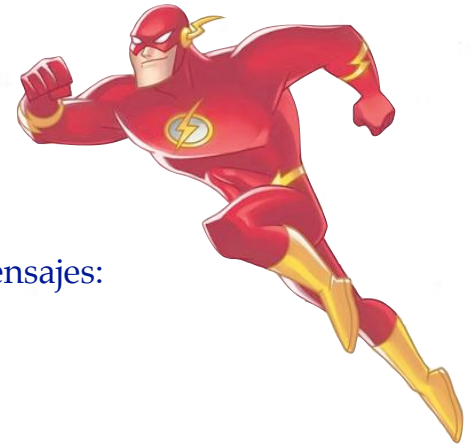
Índice

- Incorporar soporte para mensajes Flash.
  - Añadir en el layout etiquetas para mostrar los mensajes de flash.
  - Actualizar app.js y post_controller para generar mensajes de flash.
- Resaltar campos inválidos en formularios.
  - Pasar los errores de validación a la función de renderizado para que resalte los campos inválidos.
  - Modificar los formularios para que muestren de forma destacada los campos inválidos.
  - Usando varios estilos CSS.
- Desplegar en Heroku.
  - El ejemplo está disponible en la rama **`tema3`**.

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

¿Qué son los mensajes Flash?

- Son mensajes que el servidor guarda mientras atiende las peticiones HTTP recibidas, y que mostrará en la próxima vista generada.
- Los mensajes se guardan y se recuperan en el objeto **session** usando una función que suele llamarse **flash**.
  - Los mensajes se guardan agrupados por tipo.
    - Mensajes de **error**, mensajes de **información**, mensajes de **éxito**, etc…
  - La siguiente página visualizada mostrará todos los mensajes guardados.



- Por ejemplo:
  - Mientras se atiende una petición HTTP se van almacenando distintos tipos de mensajes:
    - informando del resultado de alguna operación realizada.
      - (**success**: *post modificado con éxito*)
    - informando de un cambio de estado en algún elemento.
      - (**info**: *hay 22 productos en el carrito*)
    - informando de algún error o problema.
      - (**error**: *no existe el producto solicitado*)
    - ...
  - Y al preparar (render) la siguiente página HTML (pantalla) a mostrar, se sacan los mensajes flash de la sesión y se insertan en la página para que el usuario los vea.
    - Lo normal es incluir los mensajes flash en el fichero de layout para que se muestren siempre, independientemente de cual sea la página presentada.

Instalar Soporte de Sesiones

- Los mensajes flash se guardan en el objeto de sesión **`req.session`** para que persistan entre peticiones al servidor.
- Para soportar sesiones se necesitan los módulos:

  `cookie-parser` *(ya lo instalo express-generator)*

  `express-session` *(Tenemos que instalarlo)*

- Para instalar **express-session**:
  - Ejecutar este comando para instalar el paquete:

    ```
    $ npm install --save express-session
    ```

  - Añadir a **app.js**:

    ```
    . . .
    var session = require('express-session');
    var bodyParser = require('body-parser');
    . . .
    app.use(cookieParser('coreblog 2014'));
    app.use(session());
    . . .
    ```

  - Documentación: `https://github.com/expressjs/session`

secret para firmar la cookie y evitar manipulaciones

Instalar detrás de cookieParser

Soporte de Flash

- El soporte de mensajes de flash lo proporciona el módulo **express-flash**.
- Este módulo se instala ejecutando:

```
$ npm install --save express-flash
```

  - Documentación: https://www.npmjs.org/package/express-flash

- Añadimos en **app.js**:

```
. . .
var flash = require('express-flash');
. . .
app.use(partials());
app.use(flash());
. . .
```

  - CUIDADO: El middleware express-flash debe instalarse después del middleware de sesiones.
  - CUIDADO: El middleware express-flash debe instalarse después del middleware express-partials, ya que éste redefine la función res.render y tapa los cambios realizados por express-flash.

- Este middleware modifica la función **res.render** para que las vistas puedan acceder a los mensajes de flash a través de **`res.locals.messages`**.

API

- Los mensajes flash se guardan agrupados por tipos usando la llamada:

  `req.flash(tipo, msg)`

  - Ejemplos:

```
req.flash('error', 'Mensaje de error.');
req.flash('info', 'Mensaje de información.');
req.flash('success', 'Mensaje de éxito.');
```

    - En el Blog usamos estos tres tipos de mensajes: error, info y success.

- Express-flash simplifica la forma en que se recuperan estos mensajes. Ha modificado la función **res.render** para que las vistas puedan acceder a los mensajes de flash a través de un objeto que crea y asigna a **res.locals.messages**.

  - Sin este cambio, los mensajes de flash se recuperarían con las siguientes llamadas:

  `req.flash(tipo)`

    - Devuelve un array con todos los mensajes del tipo indicado.
    - Y los borra de la flash.

  `req.flash()`

    - Devuelve un hash con todos los mensajes guardados.
      - La clave es el tipo
      - El valor es un array con los mensajes del tipo
    - Y los borra de la flash.

Mostrar Mensajes de Flash

- Los mensajes de Flash guardados los mostraremos desde el fichero de layout.
  - Añadimos en el fichero **views/layout.ejs** el código para mostrar los mensajes.
  - Añadimos también algunas reglas de estilo en el fichero **public/stylesheets/style.css**.
- Hay que retocar el controlador de los posts para crear mensajes de flash donde queramos.

```
<section>
      <% if (messages.info) { %>
          <article id='flashinfo'>
            <h3>Información de interés:</h3>
            <ul> <% for (var i in messages.info) { %>
                    <li> <%= messages.info[i] %> </li>
                 <% } %> </ul> </article>
       <% } %>
       <% if (messages.success) { %>
          <article id='flashsuccess'>
            <h3>Tareas completadas con éxito:</h3>
            <ul> <% for (var i in messages.success) { %>
                    <li> <%= messages.success[i] %> </li>
                 <% } %> </ul> </article>
       <% } %>
       <% if (messages.error) { %>
          <article id='flasherror'>
            <h3>Errores encontrados:</h3>
            <ul> <% for (var i in messages.error) { %>
                    <li> <%= messages.error[i] %> </li>
                 <% } %> </ul> </article>
       <% } %>
 </section>
```

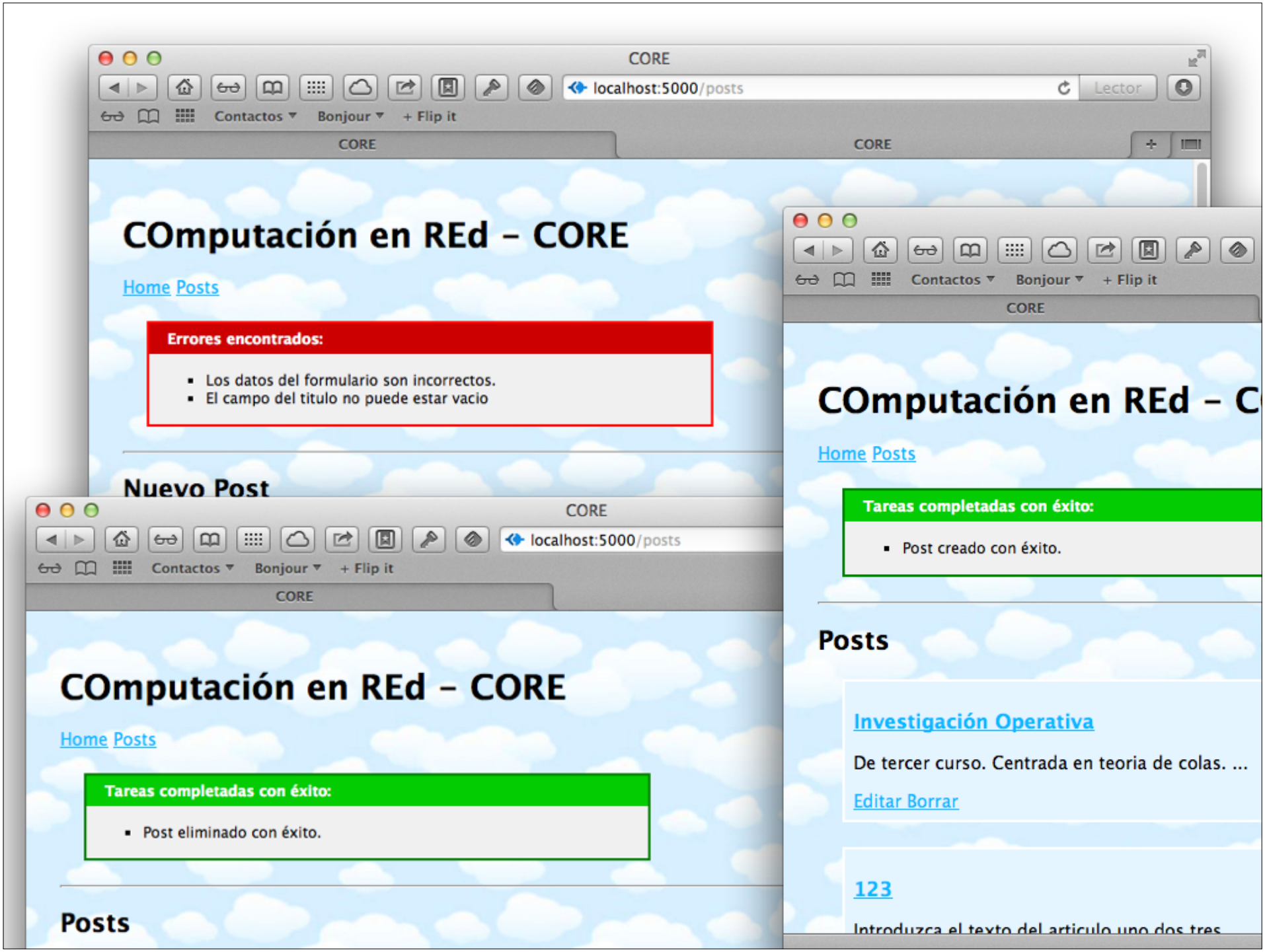
layout.ejs

```css
#flashinfo {
  width: 450px;
  border: 2px solid blue;
  padding: 7px;
  padding-bottom: 0;
  margin-bottom: 20px;
  background-color: #f0f0f0;
}

#flashinfo h3 {
    text-align: left;
    font-weight: bold;
    padding: 5px 5px 5px 15px;
    font-size: 12px;
    margin: -7px;
    margin-bottom: 0px;
    background-color: #00c;
    color: #fff;
}

#flashinfo ul li {
    font-size: 12px;
    list-style: square;
}
```

```css
#flashsuccess {
  width: 450px;
  border: 2px solid green;
  padding: 7px;
  padding-bottom: 0;
  margin-bottom: 20px;
  background-color: #f0f0f0;
}

#flashsuccess h3 {
    text-align: left;
    font-weight: bold;
    padding: 5px 5px 5px 15px;
    font-size: 12px;
    margin: -7px;
    margin-bottom: 0px;
    background-color: #0c0;
    color: #fff;
}

#flashsuccess ul li {
    font-size: 12px;
    list-style: square;
}
```

```css
#flasherror {
  width: 450px;
  border: 2px solid red;
  padding: 7px;
  padding-bottom: 0;
  margin-bottom: 20px;
  background-color: #f0f0f0;
}

#flasherror h3 {
    text-align: left;
    font-weight: bold;
    padding: 5px 5px 5px 15px;
    font-size: 12px;
    margin: -7px;
    margin-bottom: 0px;
    background-color: #c00;
    color: #fff;
}

#flasherror ul li {
    font-size: 12px;
    list-style: square;
}
```

style.css


CORE
localhost:5000/posts
Contactos
Bonjour
+ Flip it
CORE
CORE
COmputación en REd - CORE
Home Posts
Errores encontrados:
Los datos del formulario son incorrectos.
El campo del titulo no puede estar vacio
Nuevo Post
CORE
localhost:5000/posts
Contactos
Bonjour
+ Flip it
CORE
COmputación en REd - CORE
Home Posts
Tareas completadas con éxito:
Post eliminado con éxito.
Posts
Contactos
Bonjour
+ Flip it
CORE
COmputación en REd - C
Home Posts
Tareas completadas con éxito:
Post creado con éxito.
Posts
Investigación Operativa
De tercer curso. Centrada en teoria de colas. ...
Editar Borrar
123
Introduzca el texto del articulo uno dos tres

```
var validate_errors = post.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('posts/new', {post: post});
   return;
}
```

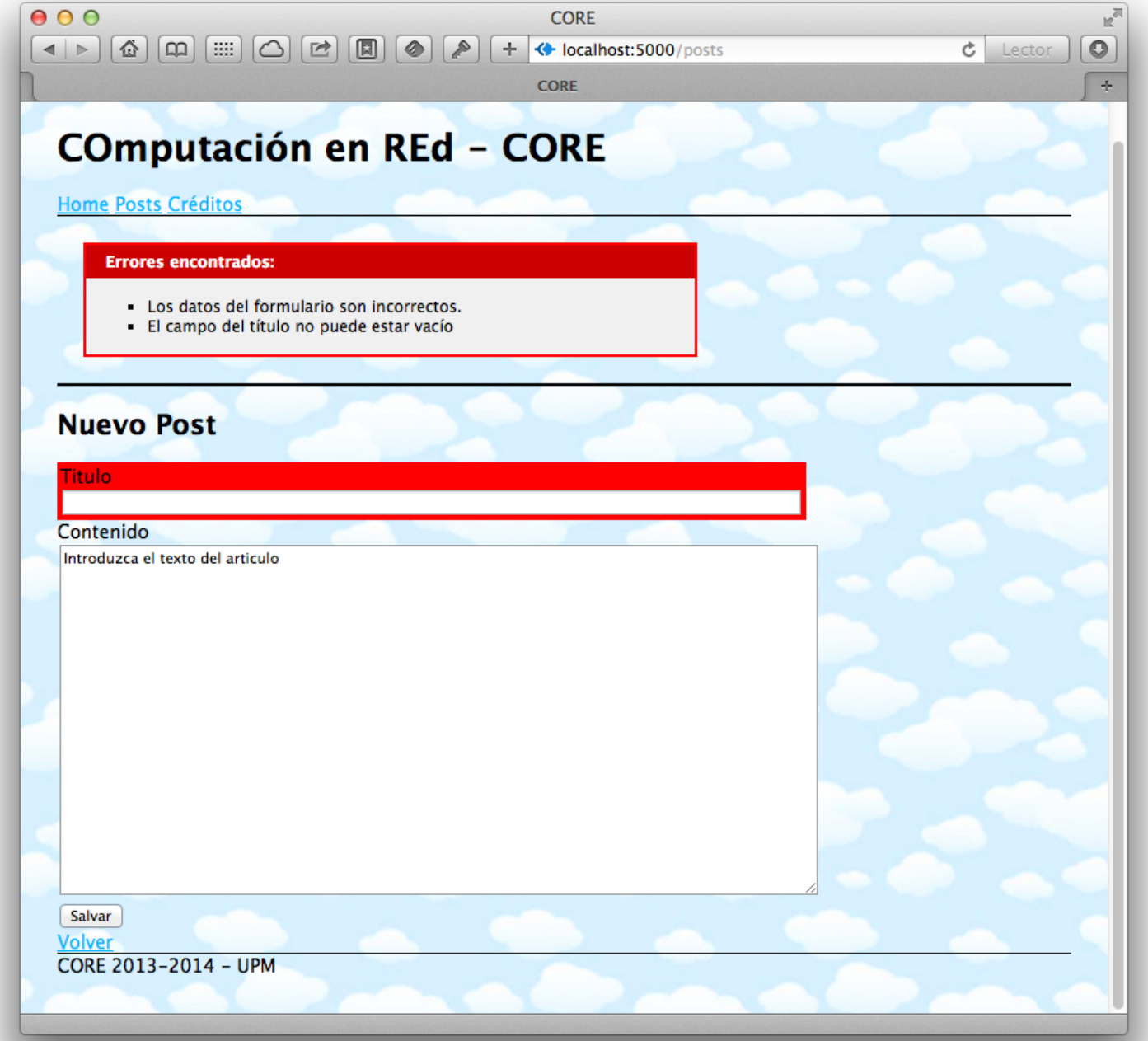
```
post.save()
    .success(function() {
        req.flash('success', 'Post creado con éxito.');
        res.redirect('/posts');
    })
    .error(function(error) {
        next(error);
    });
```

Resaltar Campos Inválidos en Formularios

Resaltar Campos Inválidos

- Objetivo:
  - Resaltar en rojo los campos inválidos en los formularios para que se localicen fácilmente.
- Usaremos dos estilos CSS:
  - Los cámpos correctos del formulario usarán la clase **.field**.
  - Los campos inválidos usarán la clase **.invalid_field**.

```
.invalid_field {
  padding: 2px;
  background-color: red;
  display: table;
}

.field {
  width: 100%;
}
```



Pasar Errores al Formulario

- Los formularios se crean:
  - desde el método **new** para crear un nuevo post:
    ```
    res.render('posts/new',{post: post});
    ```
  - desde el método **create** cuando los datos introducidos en el formulario **new** no son válidos:
    ```
    res.render('posts/new',{post: post});
    ```
  - desde el método **edit** para editar un post existente:
    ```
    res.render('posts/edit',{post: req.post});
    ```
  - desde el método **update** si los datos introducidos en el formulario **edit** no son válidos:
    ```
    res.render('posts/edit',{post: req.post});
    ```
- Modificaré estas sentencias añadiendo un parámetro que indique cuales son los campos inválidos del formulario que hay que resaltar.
  - En los casos 2 y 4 los errores de validación están en la variable **validate_errors**.
    ```
    res.render('posts/new',{post:post, validate_errors: validate_errors});
    res.render('posts/edit',{post:req.post, validate_errors:validate_errors});
    ```
  - En los casos 1 y 3 no hay errores los errores de validación.
    ```
    res.render('posts/new',{post:post, validate_errors: {}});
    res.render('posts/edit',{post:req.post, validate_errors: {}});
    ```

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

    var post = models.Post.build(
        { title: 'Introduzca el titulo',
          body:  'Introduzca el texto del articulo'
        });

    res.render('posts/new', {post: post,
                             validate_errors: {} });
};

exports.create = function(req, res, next) {
    . . .
    var validate_errors = post.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('posts/new', {post: post,
                                 validate_errors: validate_errors});
        return;
    }
    . . .
};
```

controllers/post_controller.js

```
exports.edit = function(req, res, next) {
    res.render('posts/edit', {post: req.post,
                              validate_errors: {} });
};

exports.update = function(req, res, next) {
    . . .
    var validate_errors = req.post.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('posts/edit', {post: req.post,
                                  validate_errors: validate_errors});
        return;
    }
    . . .

};
```

controllers/post_controller.js

Hay errores en el campo **title**

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

    <label for="post_title">Titulo</label><br />
    <input type="text" id="post_title" name="post[title]"
           size='80' value='<%= post.title%>' />
</div>

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

Hay errores en el campo **body**

```
    <label for="post_body">Contenido</label><br />
    <textarea id="post_body" name="post[body]"
              rows="20" cols="80"><%= post.body %></textarea>
</div>
```

views/posts/_form.ejs

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 tema3:master
(local)$ git push heroku master
```

Copiar en la rama **master** de **Heroku**. El primer comando copia en contenido **local** de la rama **tema3** 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

- Añada un mensaje flash en el ejercicio de Búsquedas (tema 2) que informe del número de posts encontrados en cada búsqueda.