Proyecto de la asignatura CORE

# Desarrollo de un Blog

*Tema 1 : Crear el Proyecto.*

CORE 2013-2014

ver: 2014-04-23

# Índice

- La funcionalidad del Blog.
- Crear esqueleto de la aplicación.
- Crear páginas estáticas.
- Gestión de errores y Páginas no encontradas.
- Layout de Aplicación.
- Desplegar en Heroku. (`http://coreblog.herokuapp.com`)

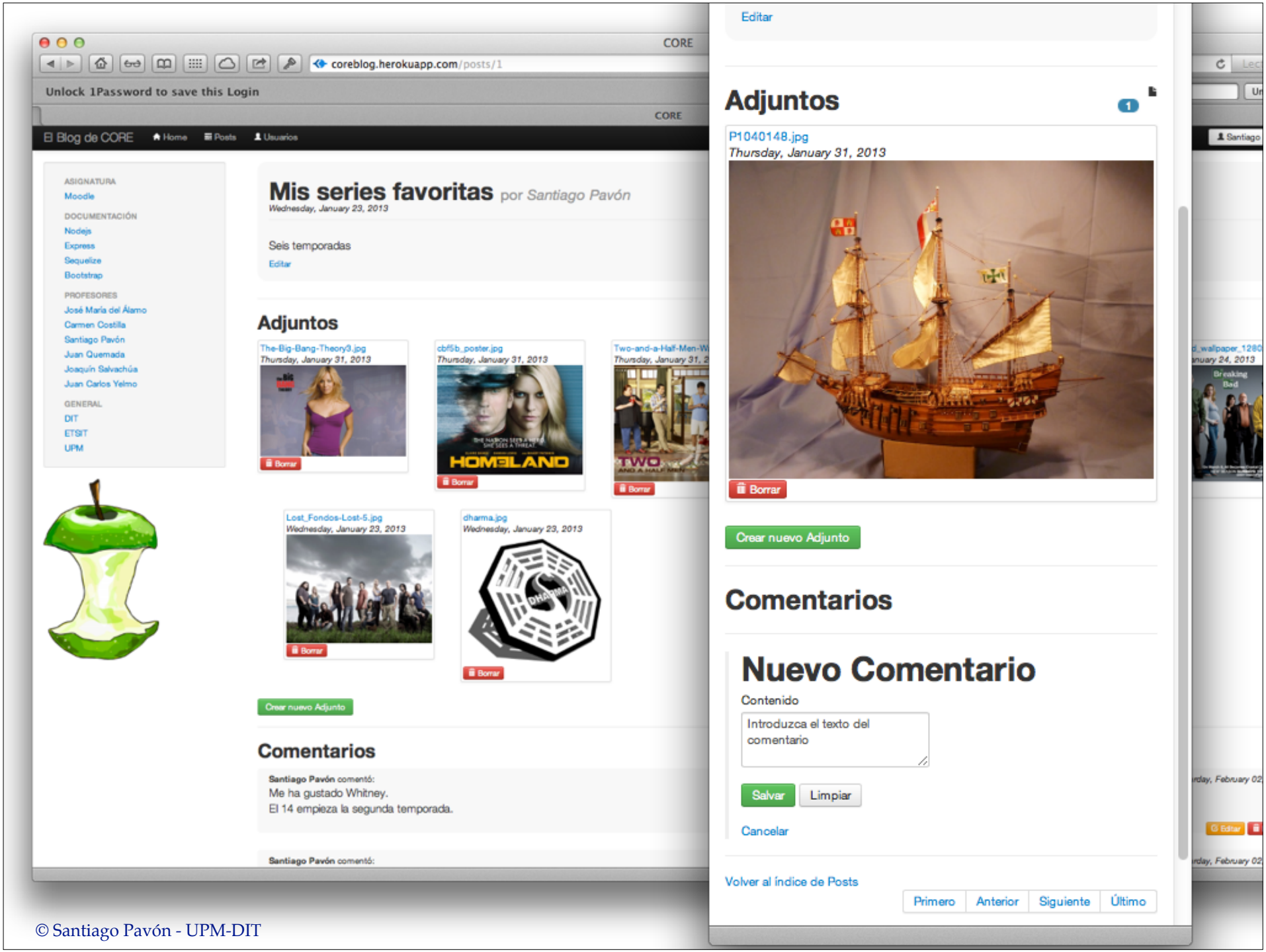
    - El código desarrollado en este tema está disponible en la rama **`tema1`**.

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

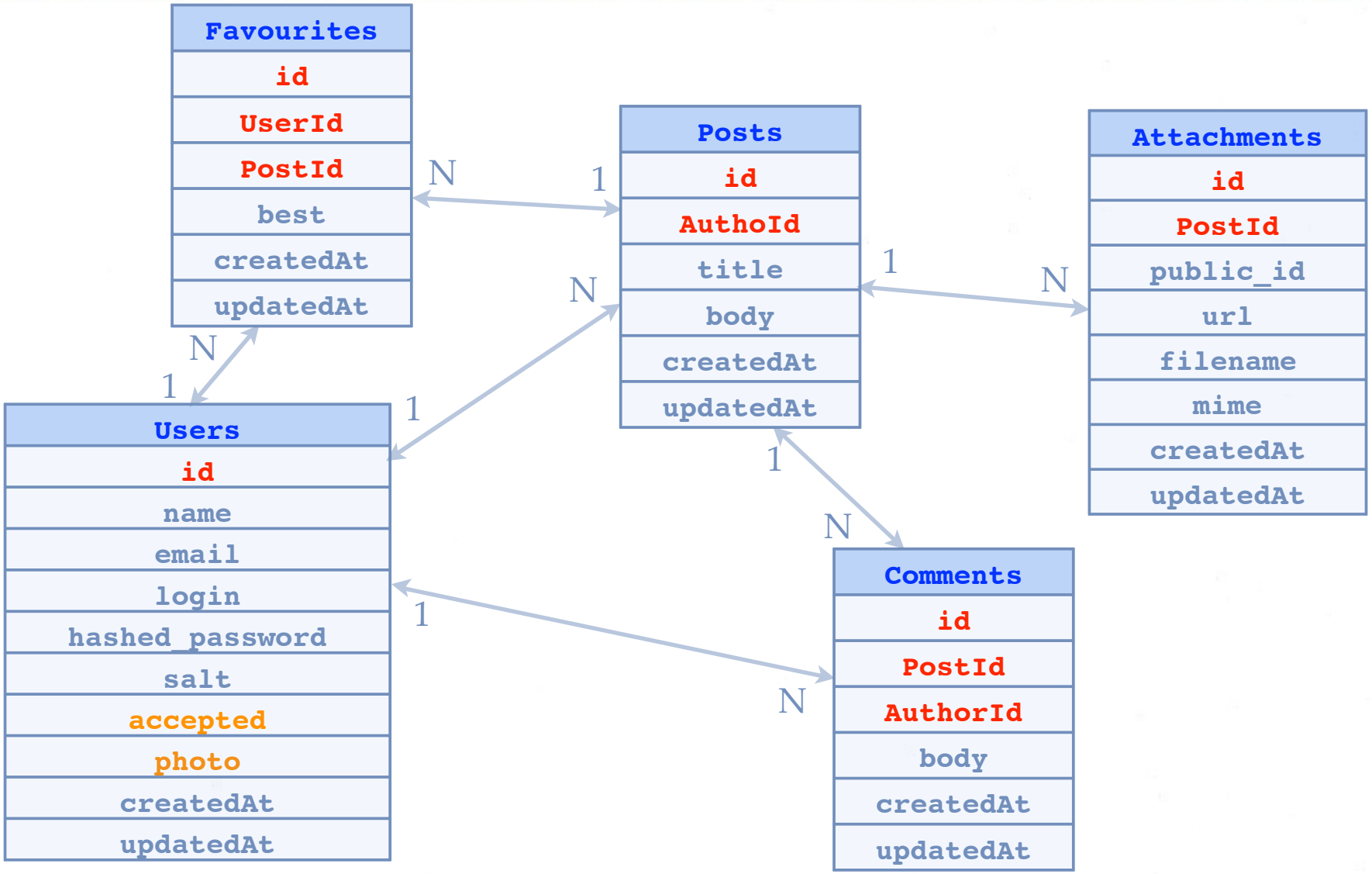
# La Funcionalidad del Blog

# Ideas Generales

- Servicio web de publicación de artículos (posts).
- Cualquier usuario puede leer el blog.
- Para publicar un post o comentar un post hay que registrarse y logearse.
- El contenido de un post o comentario sólo puede actualizarlo su autor.
- Los posts pueden llevar imágenes adjuntas.
- Se paginarán los contenidos.
- Soporte para formatos XML, JSON, ...
- Responsive design.
- Despliegue en la nube.
- . . .



# Bases de Datos



# Bases de Datos

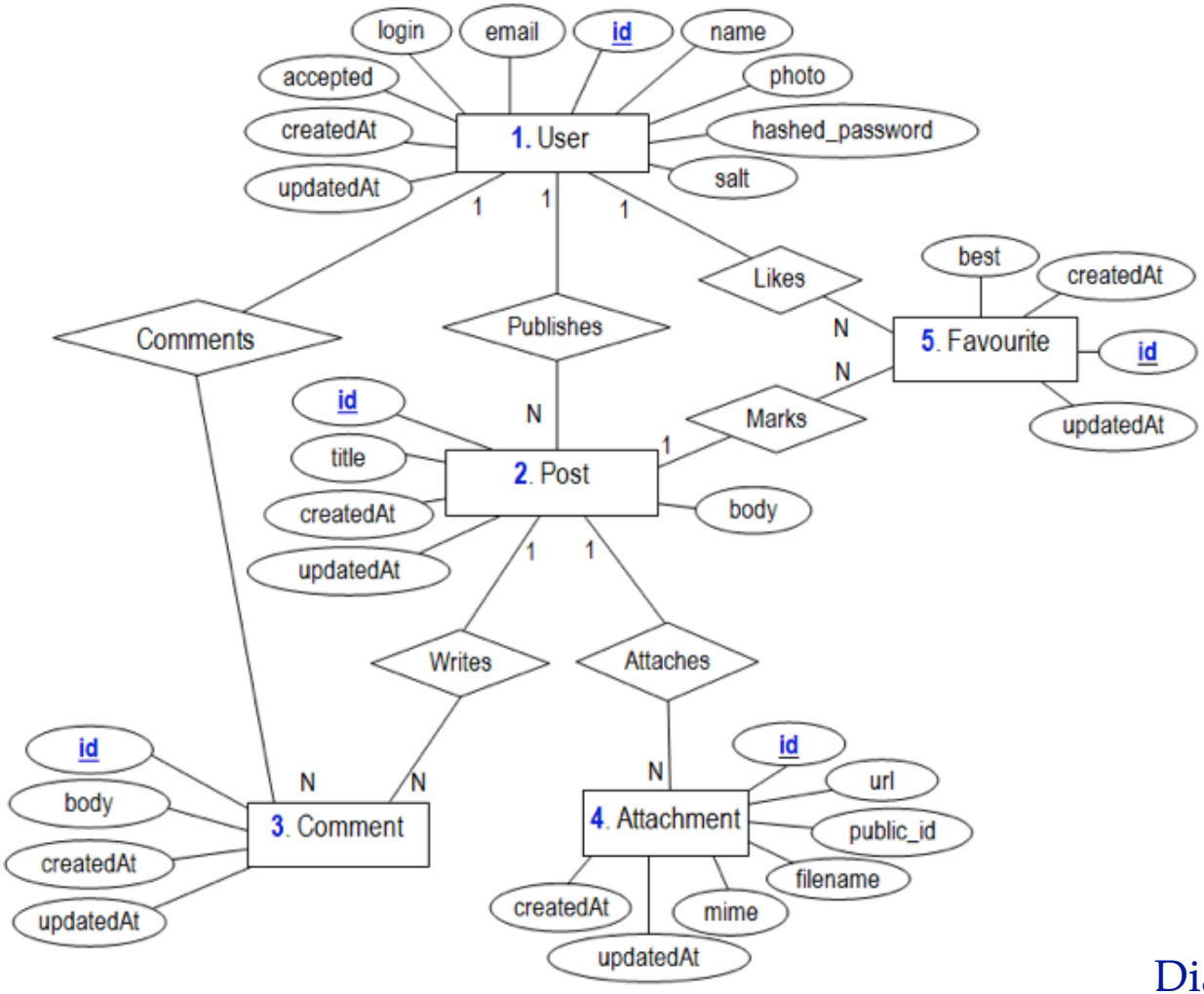
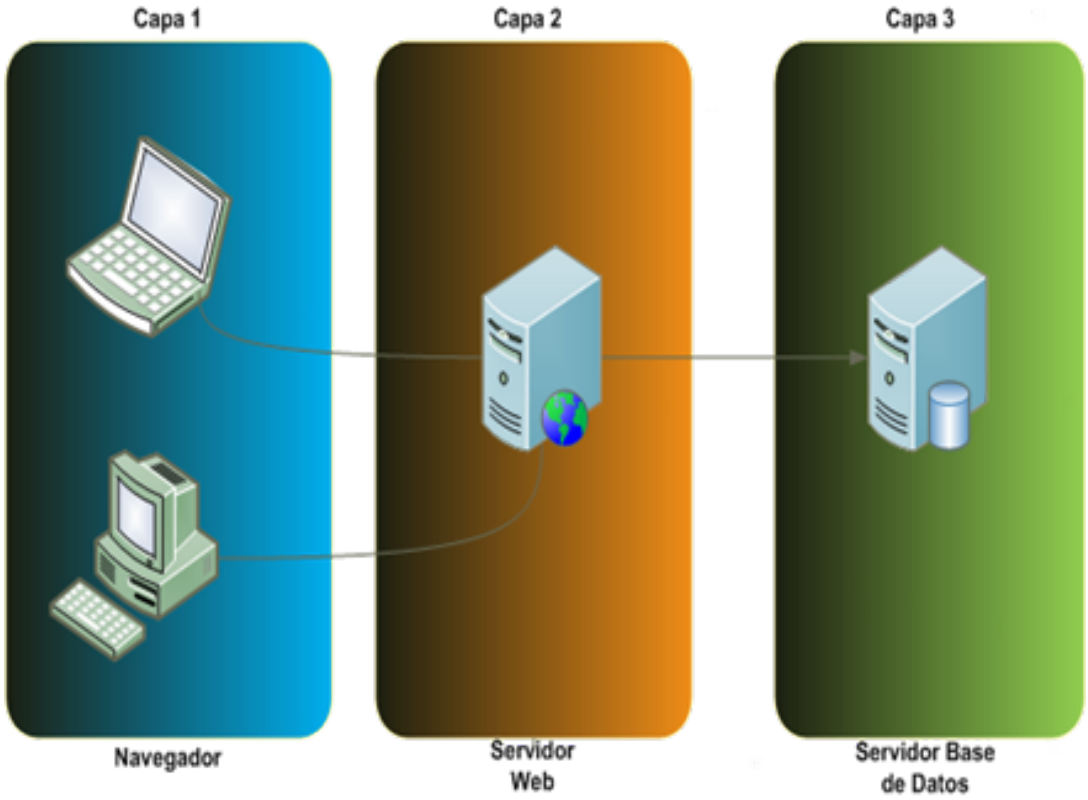



Diagrama E/R
Carmen Costilla

# Arquitectura en Tres Capas

- Frontend
  - Las vistas
- Middleware
  - La lógica de la aplicación
- Backend
  - Persistencia de la información



# Alternativas de Implementación

- Servidores Web
  - Apache, Ngnix, ...
- Servidor de aplicaciones:
  - Java EE, Rails, Sinatra, Nodejs, PHP, ...
- Frameworks:
  - expressjs, angularjs, ...
- Vistas:
  - JSP, ERB, EJS, Jade, ...
- Bases de datos
  - NoSQL: MongoDB, CouchDB
  - SQL: SQLite, MySQL, Postgres, Oracle
- Despliegue:
  - Heroku, Joyent, Nodejitsu, …

- Nuestra selección:
  - Node.js
  - Expressjs
  - EJS
  - SQLite y Postgres
  - Heroku

# Crear Esqueleto de la Aplicación

# Crear Esqueleto de la Aplicación

- Instalar el módulo express:

```
$ sudo npm install -g express-generator

$ express -V    // Ver qué versión hemos instalado
 4.0.0
```

- Creamos la aplicación en el directorio blog:

```
$ express --ejs blog // Crea ficheros iniciales del proyecto
$ cd blog            // Cambiarse al directorio blog
$ npm install        // Instala dependencias declaradas en package.json
```

- Ejecutamos la aplicación:

```
$ npm start        // Comando definido en package.json
```

- Y nos conectamos con un navegador a `http://localhost:3000`.

Notas

Se supone que **nodejs** ya está instalado.

**Generador de Express - Versión: 4.0.0**

Hay que instalar **express-generator**, que es el paquete utilizado para generar la primera versión de la aplicación usando **expressjs**.

Este paquete se instala con el comando:

```
$ [sudo] npm install [-g] express-generator
```

- Puede ser necesario desinstalar antes otras versiones del paquete **express** si aparezcan conflictos de instalación entre diferentes versiones.

Para instalar **express-generator** en la cuenta local, ejecutar: "**npm install express-generator**". El ejecutable **express** estará en el path **~/node_modules/express-generator/bin/express**, o en **~/node_modules/.bin/express**.

Para instalar express de forma global, para que esté accesible en todo el sistema, ejecutar "**sudo npm install -g express-generator**". En este caso el ejecutable express estará en **/usr/local/bin/express**.

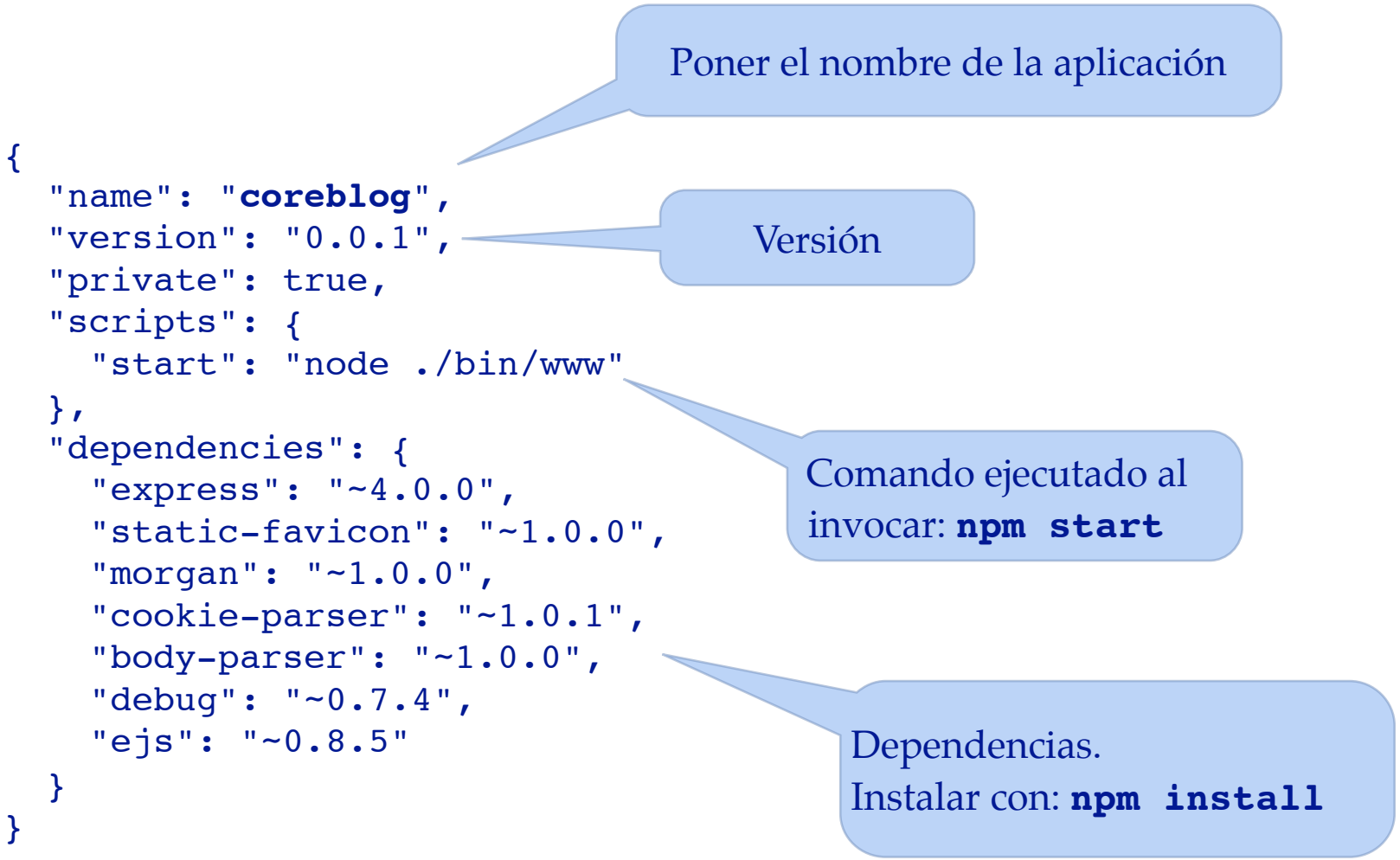
Como motor de vistas de la aplicación queremos usar EJS. Para ello añadimos la opción **--ejs** al crear nuevas aplicaciones.

Documentación:

- nodejs
  - http://nodejs.org
  - https://npmjs.org
  - https://npmjs.org/doc/json.html
- expressjs
  - http://expressjs.com
  - https://www.npmjs.org/package/express-generator
- EJS
  - https://github.com/visionmedia/ejs

# Motor de Vistas EJS

- **EJS** = Javascript embebido.
- Los ficheros de vistas EJS contienen:
  - texto HTML.
  - código javascript entre las marcas **<%** y **%>**.
  - expresiones javascript entre las marcas **<%=** y **%>**.
    - El valor de las expresiones se incorpora al texto HTML.
    - Previamente se escapan los caracteres conflictivos para evitar inyección de código:
      - **<** se sustituye por **<**
      - **>** se sustituye por **>**
      - **&** se sustituye por **&**
      - ...
  - expresiones javascript entre las marcas **<%-** y **%>**.
    - El valor de las expresiones se incorpora al texto HTML.
    - No se escapan los caracteres conflictivos.
  - inclusión de ficheros con **<% include** *path_del_fichero_a_incluir* **%>**.
- Documentación: `https://github.com/visionmedia/ejs`




# package.json

```
#!/usr/bin/env node
var debug = require('debug')('my-application');
var app = require('../app');

app.set('port', process.env.PORT || 3000);

var server = app.listen(app.get('port'), function() {
  debug('Express server listening on port ' + server.address().port);
});
```

## bin/www

```
var express = require('express');
var path = require('path');
var favicon = require('static-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');


var routes = require('./routes/index');
var users = require('./routes/users');


var app = express();


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


app.use(favicon());
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use(cookieParser());
app.use(express.static(path.join(__dirname,
                                 'public')));

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

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

/// error handlers

// development error handler
// will print stacktrace
if (app.get('env') === 'development') {
    app.use(function(err, req, res, next) {
        res.status(err.status || 500);
        res.render('error', {
            message: err.message,
            error: err
        });
    });
}

// production error handler
// no stacktraces leaked to user
app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.render('error', {
        message: err.message,
        error: {}
    });
});


module.exports = app;
```

app.js

# Página Raíz
# Páginas Estáticas

# Página Raíz

- **express-generator** creó en **app.js** dos manejadores de rutas (dos instancias de **Router**) para atender las URLs que empiezan por **/** y por **/users**.

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

- El manejador de las rutas que empiezan por **/** se configura en el módulo **routes/index.js**.

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

  - Solo maneja la ruta raíz (/), *renderizando* la vista **/views/index.ejs**.

- Vamos a cambiar el contenido de la página raíz de nuestra aplicación:
  - Cambiamos el contenido de **routes/index.js** por:

```
exports.index = function(req, res){
   // res.render('index', { title: 'Express' });
   res.render('index');
};
```

  - Editamos el fichero **views/index.ejs** para crear el nuevo contenido de nuestra página raíz.
    - El contenido de este fichero puede verse en la siguiente transparencia.

    - He usado una imagen (**layers.png**) en el body, y otra imagen (**bg.jpg**) como fondo de pantalla:
      - Copiar los ficheros **layers.png** y **bg.jpg** en el directorio **public/images** para que los sirva el middleware static.

    - He modificado también la hoja de estilos **public/stylesheets/style.css**:

```
body {
  background-image: url("/images/bg.jpg");
  padding: 20px;
  font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;
}
section {
  border-style: solid;
  border-width: 1px 0px 1px 0px;
}
. . .
```

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <link rel="stylesheet" type="text/css" href="/stylesheets/style.css">
   <title>CORE</title>
</head>

<body>
   <header>
     <h1> COmputación en REd - CORE</h1>
     <nav>
        <a href="/">Home</a>
        <a href="/creditos.html">Créditos</a>
     </nav>
   </header>

   <section>
      <h2>Bienvenido a CORE</h2>

      <p>Esta asignatura te llegará al corazón.</p>

      <img src='images/layers.png' alt='layers'/>
   </section>

   <footer>
      CORE 2013-2014 - UPM
   </footer>
</body>
</html>
```

## El nuevo **views/index.ejs**

# Páginas Estáticas

- Las páginas estáticas se sirven desde el directorio **public**.
  - Consultar **app.js** para ver que el middleware que sirve páginas estáticas (**express.static**) está configurado para servir los ficheros del subdirectorio **public**.
- Crear en el directorio **public** el siguiente fichero:
  - **creditos.html** - página con información sobre los autores del proyecto.

    - Notad que en **views/index.ejs** ya añadimos un enlace a esta página de créditos.

```
<a href="/creditos.html">Créditos</a>
```

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <link rel="stylesheet" type="text/css" href="/stylesheets/style.css">
   <title>CORE</title>
</head>

<body>
   <header>
     <h1> COmputación en REd - CORE</h1>
     <nav>
        <a href="/">Home</a>
        <a href="/creditos.html">Créditos</a>
     </nav>
   </header>

   <section>
     <h2>Autores</h2>

     <ul>
         <li>Nombre del primer autor</li>
         <li>Nombre del primer autor</li>
     </ul>
   </section>

   <footer>
      CORE 2013-2014 - UPM
   </footer>
</body>
</html>
```

public/creditos.html

# Limpieza: Eliminar /users

- Eliminar las rutas de **/users**
  - En **app.js** borramos:
    - la línea donde se carga el módulo routes/user.

```
var users = require('./routes/users');
```

    - la línea donde se configura la ruta a **/users**.

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

  - Y borramos el fichero **routes/users.js**.

CORE

localhost:3000

Lector

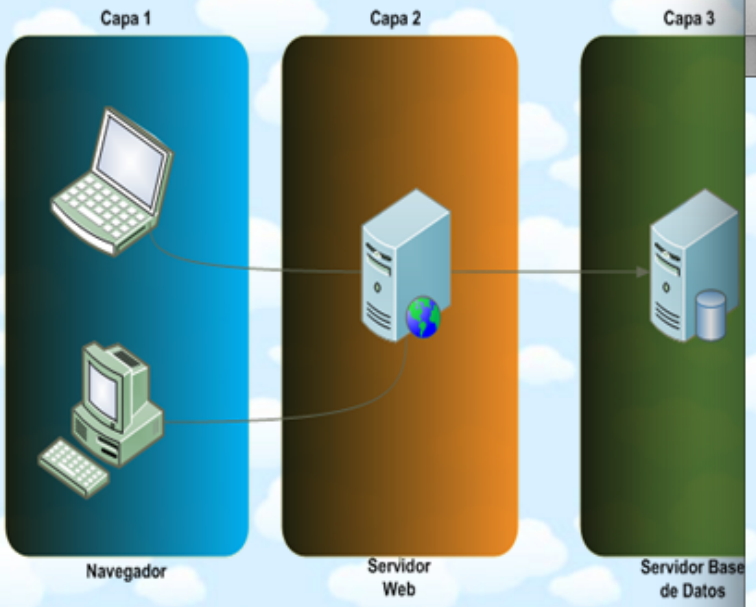
CORE

# COmputación en REd – CORE

Home Créditos

## Bienvenido a CORE

Esta asignatura te llegará al corazón.




CORE 2013-2014 – UPM

CORE

localhost:3000/creditos.html

CORE

# COmputación en REd – CORE

Home Créditos

## Autores

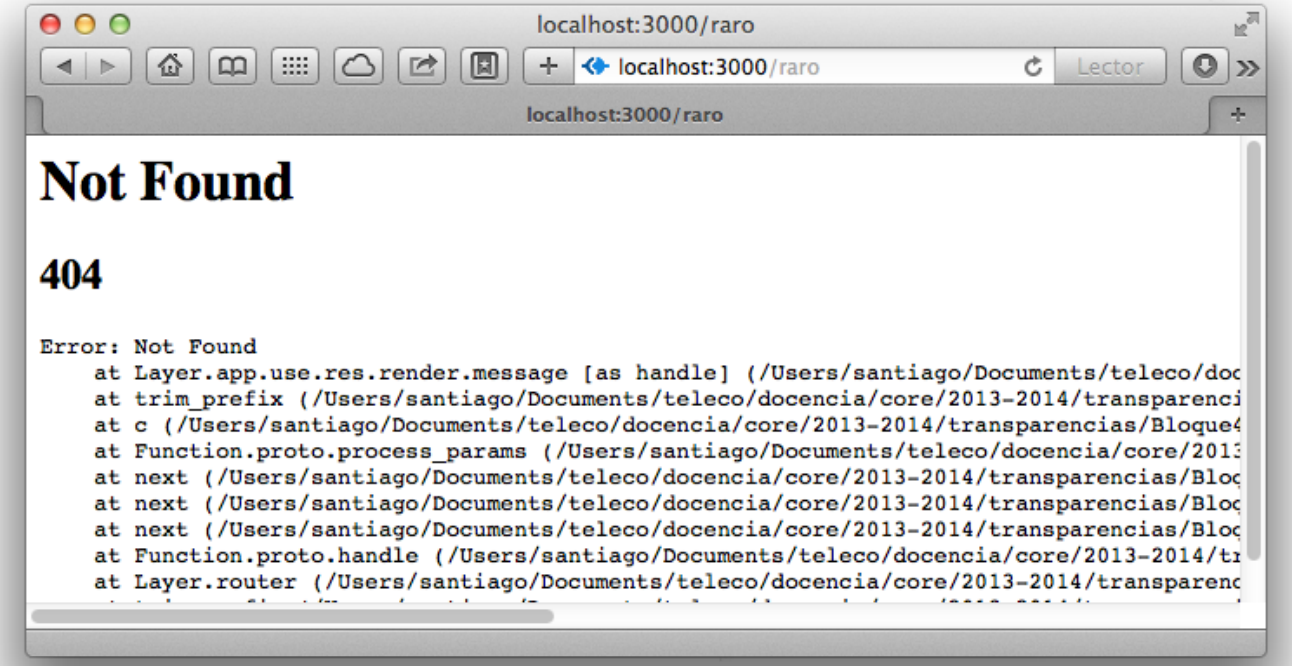
- Nombre del primer autor
- Nombre del primer autor

CORE 2013-2014 – UPM

# Middleware de Gestión de Errores

# Gestión de Errores

- Al final de **app.js** se han incluido dos middlewares para gestionar los errores que se producen.
  - Uno para cuando estamos en modo **desarrollo**, y otro para **producción**.
    - El middleware de **desarrollo** renderiza la página **views/error.ejs** informando del error ocurrido, y **mostrando** el stack de llamadas.
    - El middleware de **producción** renderiza la página **views/error.ejs** informando del error, pero **sin mostrar** el stack de llamadas.



- Dado que la página **views/error.ejs** no tiene el mismo aspecto que **index.ejs** o **creditos.html**, la modificaremos para que presente el mismo aspecto.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <link rel="stylesheet" type="text/css" href="/stylesheets/style.css">
   <title>CORE</title>
</head>

<body>
   <header>
     <h1> COmputación en REd - CORE</h1>
     <nav>
        <a href="/">Home</a>
        <a href="/creditos.html">Créditos</a>
     </nav>
   </header>

   <section>
        <h2> Atención </h2>

        <p>
        El Servidor ha experimentado un pequeño problema:
        </em>(Código: <%= error.status %>)</em>
        </p>

        <p>
        <strong> <%= message %> </strong>
        </p>

        <pre><%= error.stack %></pre>
   </section>

   <footer>
      CORE 2013-2014 - UPM
   </footer>
</body>
</html>
```

# El nuevo **views/error.ejs**

# Layout de Aplicación

# Layout de Aplicación

- Los ficheros **views/index.ejs**, **views/error.ejs**, y otros ficheros que crearemos en el futuro, son páginas HTML con muchas partes en común (cabeceras, pies de página, menús, …).
  - Para no repetir la parte común HTML en todos ellos, usaré un layout de aplicación.
- Usaré el módulo **express-partials** para tener soporte de layouts.
  - Documentación: **`https://github.com/publicclass/express-partials`**
  - Este módulo modifica **`res.render()`** para que por defecto inserte la vista EJS generada dentro del fichero **`views/layout.ejs`**, sustituyendo a la variable **`body`**.
  - Instalación de express-partials:
    - Ejecutar:

```
$ npm install --save express-partials
```

    - Añadir en **app.js** las líneas:

```
var partials = require('express-partials');
. . .
app.use(partials()); // antes del middleware routes
```

Notas

**Comentarios sobre el uso de layouts:**
El uso de layouts permite que todas las vistas estén enmarcadas en una página común. En las últimas versiones, express ha delegado el uso de layouts, partials y otras características en los motores de plantillas de generación de vistas. Dado que estamos interesados en usar layouts, instalaré el paquete express-partials, que añade soporte para layouts de aplicacion y para partials.

El fichero de layout por defecto es **views/layout.ejs**. En él se sustituye la variable "**body**" por el resultado de renderizar la vista a mostrar.

El paquete **express-partials** redefine el método **res.render** para que acepte un parámetro adicional que indica cual es el layout a usar, usando **views/layout.ejs** como layout por defecto en caso de omitir este parámetro.

También podemos evitar el uso de este paquete, y programar nosotros mismo el uso de layouts usando el siguiente código:

```
res.render('fichero.ejs',opciones,function(err,html) {
   res.render('layout.ejs', {body: html});
});
```

Otra forma de evitar el uso de este paquete, es crear unos ficheros **head.ejs** y **foot.ejs** para incluir al principio y al final de cada vista. De esta forma, todos los ficheros de vistas serían así:

```
<% include head %>

<h1>Titulo</h1>
<p>Datos</p>

<% include foot %>
```

Notas

**Comentarios sobre el uso de vistas parciales:**

El uso de vistas parciales es muy cómodo cuando queremos reutilizar el mismo código ejs en varios sitios, o aplicarlo sobre todos los elementos de una colección.

En nuestro caso, las necesidades de compartir código ejs son muy sencillas, por lo que usaré la directiva **`include`** del motor ejs en vez de utilizar vistas parciales.

```
{
  "name": "coreblog",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~4.0.0",
    "static-favicon": "~1.0.0",
    "morgan": "~1.0.0",
    "cookie-parser": "~1.0.1",
    "body-parser": "~1.0.0",
    "debug": "~0.7.4",
    "ejs": "~0.8.5",
    "express-partials": "~0.2.0"
  }
}
```

# package.json

```
var express = require('express');
var path = require('path');
var favicon = require('static-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var partials = require('express-partials');

var routes = require('./routes/index');

var app = express();

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

app.use(favicon());
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use(partials());

app.use('/', routes);

. . .
```

Cargar el módulo

Instalar el middleware antes de configurar las rutas.

app.js

# Crear el Layout de la Aplicación

- Crearemos el fichero **views/layout.ejs**.
  - Contiene el layout de la aplicación.
    - Es una página HTML que contiene la sentencia **<%- body %>**.
  - Todas las vistas renderizadas sustituirán a la variable **body** en el layout.
- Una vez creado el layout de la aplicación debemos simplificar las plantillas **views/index.ejs** y **views/error.ejs** eliminando todo lo que ya proporciona el layout.
  - Nota: La página estática de créditos debería transformarse en otra plantilla EJS que también hiciera uso del layout de aplicación.
    - Se deja como ejercicio al alumno.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <link rel="stylesheet" type="text/css" href="/stylesheets/style.css">
   <title>CORE</title>
</head>

<body>
   <header>
     <h1> COmputación en REd - CORE</h1>
     <nav>
         <a href="/">Home</a>
         <a href="/creditos.html">Créditos</a>
     </nav>
   </header>

   <section>
     <%- body %>
   </section>

   <footer>
      CORE 2013-2014 - UPM
   </footer>
</body>
</html>
```

**<%- body %>**
se sustituye por la vista EJS que se renderice

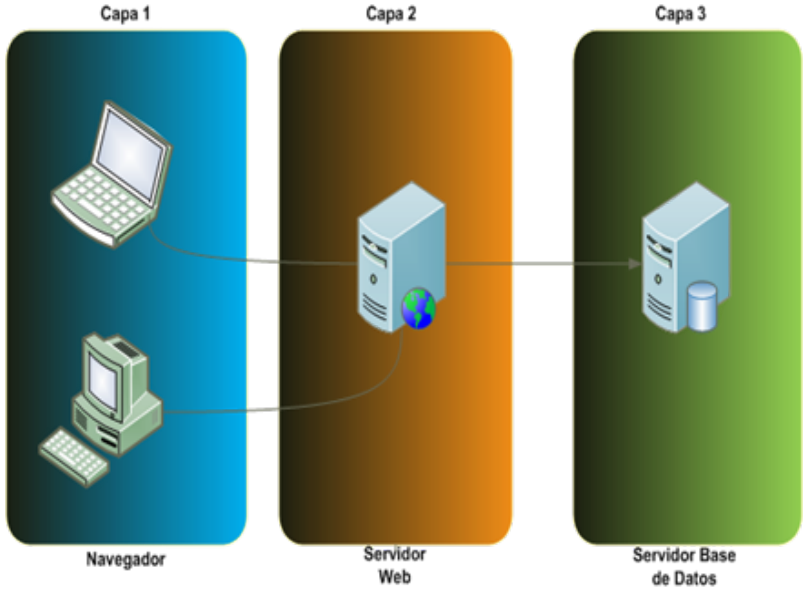
views/layout.ejs

# views/index.ejs

```
<h2>Bienvenido a CORE</h2>

<p>Esta asignatura te llegará al corazón.</p>

<img src='images/layers.png' alt='layers'/>
```



```
<h2> Atención </h2>

<p>
El Servidor ha experimentado un pequeño problema:
</em>(Código: <%= error.status %>)</em>
</p>

<p>
<strong> <%= message %> </strong>
</p>

<pre><%= error.stack %></pre>
```

views/error.ejs


# Desplegar en Heroku

# Pasos

- Pasos a seguir:
  - Crearse una cuenta en Heroku.
  - Instalar Heroku Toolbelt.
  - Crear la aplicación en Heroku.
  - Preparar la aplicación para poder desplegarla.
  - Desplegar.
- Documentación:
  - Sitio Web de Heroku:
    **`http://www.heroku.com`**
  - En este sitio hay muchos artículos que leer.
    - Empezar por **Dev Center > Overview > Getting Started**:
      **`https://devcenter.heroku.com/articles/quickstart`**
    - Detalles de despliegue para Nodejs en:
      **`https://devcenter.heroku.com/articles/nodejs`**

Notas

Para desplegar la aplicación podemos elegir entre desplegar en nuestros propios servidores, o desplegar en servidores en la nube.

Vamos a desplegar en Heroku porque soporta el despliegue de nuestro tipo de aplicación, y porque permite tener cuentas gratuitas. Los alumnos pueden abrirse su propia cuenta y desplegar sus prácticas a coste cero.

La opción de coste cero proporciona sólo una máquina virtual, un tiempo de respuesta bajo, limitación a pocas peticiones simultáneas, poco espacio en la base de datos, backups poco frecuentes, ... Pero es suficiente para hacer las prácticas.

En la página de Heroku está disponible toda la documentación necesaria sobre su uso. En este tema simplemente se ilustrarán los comandos básicos y los cambios a realizar en la aplicación para hacer el despliegue. No se entrará en muchos detalles.

En un caso real será necesario contratar más recursos, usar un servidor de base de datos mejor, registrar dominios, etc...

- Cada alumno debe crearse una cuenta en Heroku

  `https://api.heroku.com/signup/devcenter`

- Instalarse Heroku Toolbelt

  `https://toolbelt.heroku.com`

  - Son los programas necesarios para gestionar y configurar los despliegues desde la estación de desarrollo.

- Desde un terminal, hacer login:

```
$ heroku login
```

  - Nota: al hacer login nos preguntan si deseamos crear, caso de no existir, una clave pública. También nos preguntan si queremos subirla a Heroku. Contestad que si.

- En caso de problemas:
  - Para crear una clave nueva y subirla, ejecutar:

```
$ heroku keys:add
```

  - Para crear unas claves nuevas, ejecutar:

```
$ ssh-keygen -t rsa
```

  - Para subir una clave pública ya existente, ejecutar:

```
$ heroku keys:add ~/.ssh/id_rsa.pub
```

  - Si no acepta la clave publica añadida, ejecutar el siguiente comando para indicar cual es nuestra cuenta en heroku:

```
$ heroku accounts:set nuestra_cuenta
```

- Añadir a **`package.json`** una nueva sección indicando la versión de **nodejs** y **npm** que debe usar el servidor en Heroku para ejecutar nuestra aplicación y para instalar los paquetes de los que dependemos.

```
"engines": {
  "node": "0.10.x",
  "npm": "1.3.x"
}
```

- Crear en la raíz del proyecto el fichero **`Procfile`**.
  - Este fichero contiene los comandos que deben ejecutarse en el servidor de Heroku para lanzar los distintos tipos de procesos que vamos a usar.
  - En nuestro caso, este fichero sólo contiene el comando a ejecutar para lanzar el servidor web.

```
web: node ./bin/www
```

- Las aplicaciones se copian, en **Heroku** usando **git**.
  - Por tanto, nuestra aplicación debe estar en un repositorio git para poder subirla a Heroku.
    - Puede ser un repositorio local, uno alojado en github, ...
  - Si nuestro proyecto no está en git, podemos crear un repositorio local.
    - Primero crear el fichero **.gitignore** añadiendo los elementos que git debe ignorar:

```
node_modules
npm-debug.log
```

    - Segundo, ejecutar los siguientes comandos:

```
$ git init
$ git add .
$ git commit -m "Version inicial"
```

      - Estos comandos deben ejecutarse desde dentro del directorio **blog**.
  - También podemos clonar la versión del proyecto disponible en GitHub con el comando:

```
$ git clone git@github.com:CORE-UPM/blog_2014.git
```

- Crear la aplicación en heroku

```
$ heroku create
```

  - Este comando da de alta en Heroku nuestra aplicación, crea la máquina virtual donde se ejecutará el servicio, crea el URL público para acceder a la aplicación, crea un remote git en nuestro repositorio llamado **heroku** que apunta al repositorio git en Heroku, etc...
  - Nos devuelve:

```
Creating stark-depths-3830... done, stack is cedar
http://stark-depths-3830.herokuapp.com/ |
git@heroku.com:stark-depths-3830.git
Git remote heroku added
```

    - El URL donde está ejecutándose nuestra aplicación.
    - El URL git donde debemos subir el código de nuestra aplicación.

- Para cambiar el nombre de la aplicación creada en heroku, ejecute el comando:

```
$ heroku apps:rename otronombrequemegustemas
```

- Desplegar el código de nuestra aplicación en heroku, es decir, subir los ficheros javascript, ejs, etc. a Heroku:
  - Normalmente subiremos los ficheros ejecutando:

```
$ git push heroku master
```

Subir mi rama **master** al remoto **heroku**

    - donde **heroku** es el nombre de un *remote git* al que subimos (*push*) la versión de la aplicación apuntada por nuestra rama **master**.
      - En ocasiones puede ser necesario añadir la opción **-f** a push si la versión actual en heroku contiene cambios que no están en nuestra rama master, o estamos haciendo cosas avanzadas/raras con ramas, …

- Para probar la aplicación:
  - Conectarse desde un navegador al URL asignado a la aplicación.

```
http://otronombrequemegustemas.herokuapp.com
```

  - También podemos lanzar automáticamente un navegador ejecutando:

```
$ heroku open
```

# Dashboard

- El dashboard de heroku:

  `https://dashboard.heroku.com`

  - Permite ajustar nuestro perfil, administrar nuestras aplicaciones, gestionar los addons, ...

# Examen

# Preguntas

1. Desarrolle un periódico digital.
   - Debe ser una aplicación desarrollada con express.
   - Las secciones del periódico se implementarán como páginas estáticas.
     - Cree tres secciones/páginas: noticias, deportes y corazón.
   - Añada middlewares para manejar errores y páginas no encontradas.
   - Añada soporte para tener un layout de aplicación.
   - Despliegue el periódico en Heroku.