El proyecto Quiz y MVC

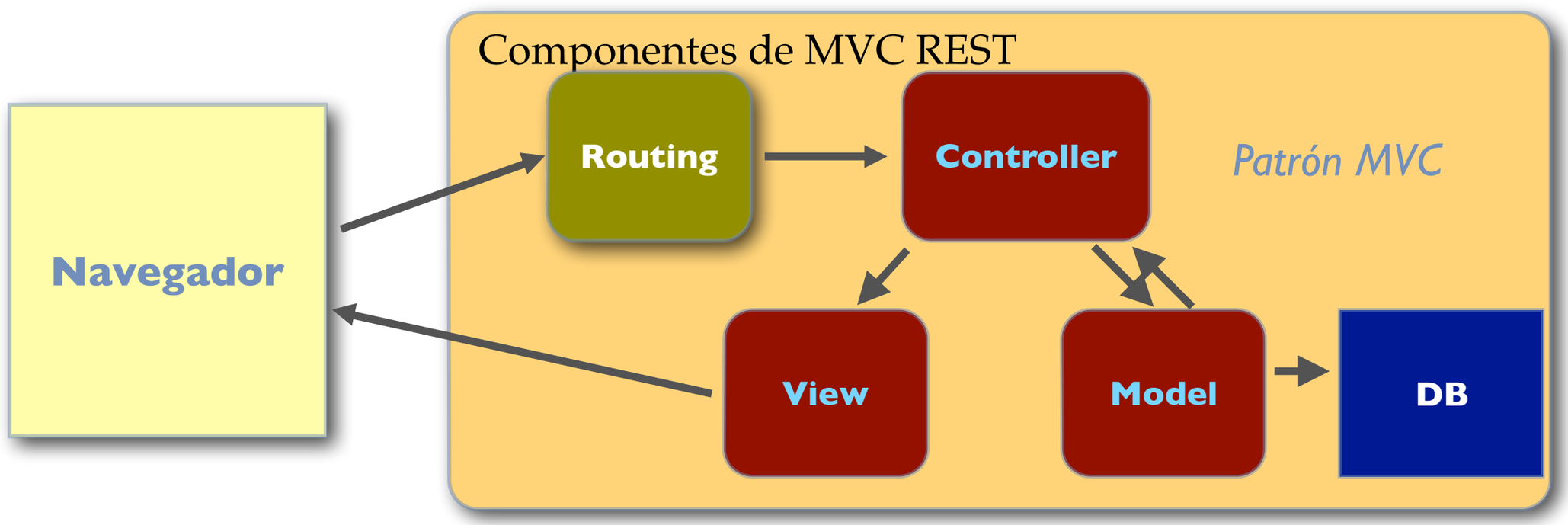
Juan Quemada, DIT - UPM

©

Patron MVC
(Adaptado a Web)

MVC: propuesto por **Trygve Reenskaug** en 1979 para estructurar aplicaciones basadas en ventanas

- **Modelo**: Datos de la aplicación y sus restricciones
  - En una BD, en un fichero (este ejemplo), ...
- **Vista**: Genera vistas de usuario
  - Prepara la página Web que se envía al cliente
- **Controlador**: orquesta la aplicación
  - Contiene la lógica de la aplicación
- **Routing**: asocia URLs con acciones del controlador



MVC y la estructura de un equipo

- MVC divide el trabajo de diseño entre distintos especialistas
  - **Modelo**: especialista en bases de datos (técnico)
  - **Vistas:** diseñador (creativo, artista)
  - **Controlador**: programador de la lógica de la aplicación (técnico)
  - **Router:** arquitecto diseñador del interfaz de la aplicación (técnico)

- Además existen otras especialidades
  - **Administrador:** administrador de sistemas en la nube (técnico)
  - **Diseñador de experiencia:** usabilidad, interacción, marca, ... (mixto)
  - **SEO (S**earch **E**ngine **O**ptimizatión): posiciona portal en buscadores (Google, …)
  - **Community Manager:** gestiona la relación con los usuarios
  - .....

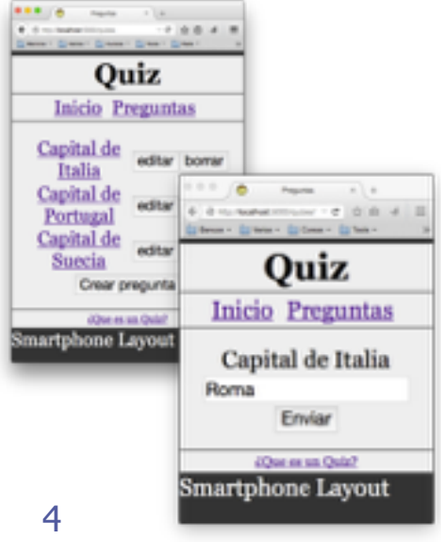
©

Proyecto Quiz

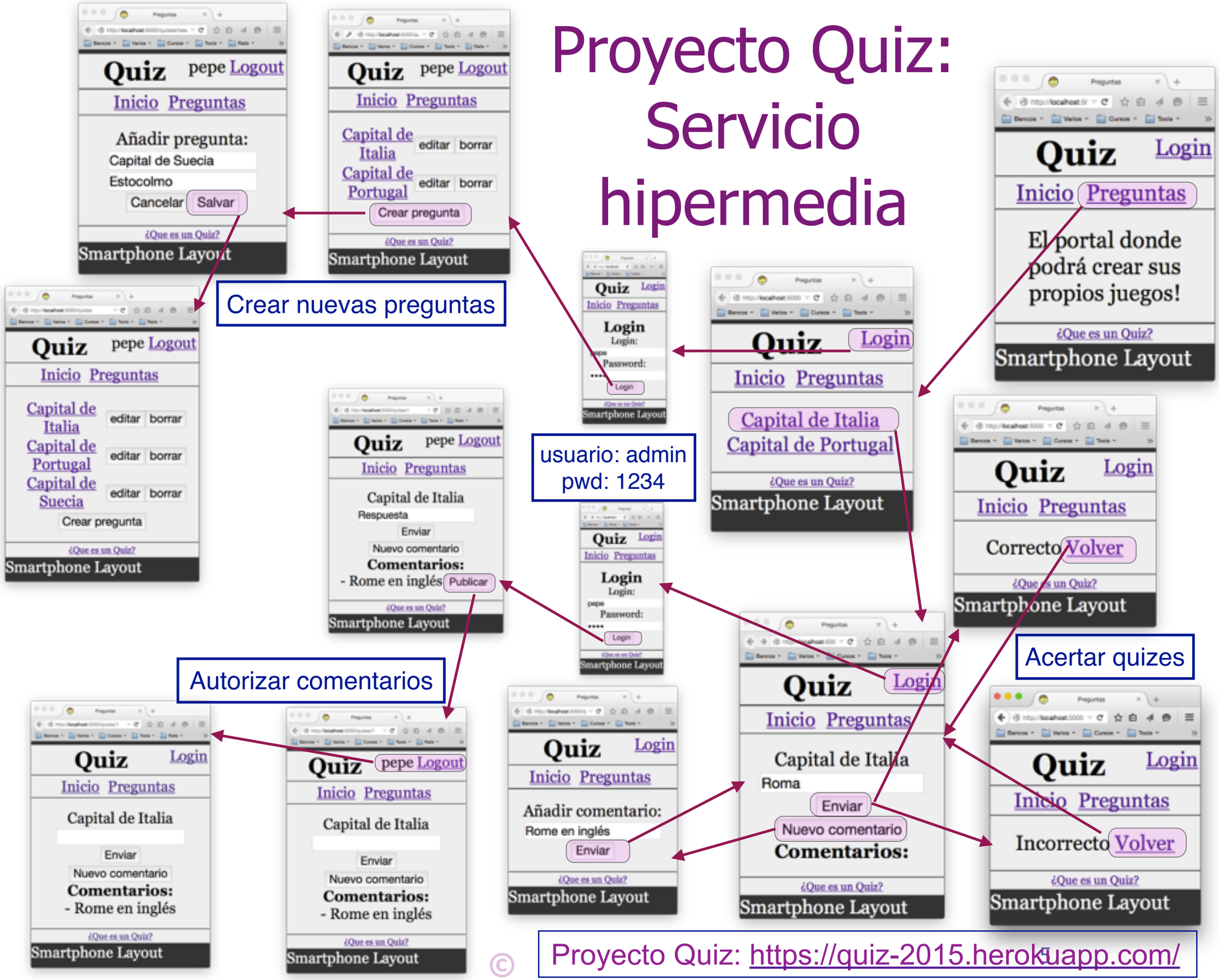
- ✦**Paso 1**: Esqueleto del proyecto con express-generator
- ✦**Paso 2**: Primera página y el favicon
- ✦**Paso 3**: Primera pregunta
- ✦**Paso 4**: Vistas parciales y marco
- ✦**Paso 5**: CSS adaptable a móviles y PCs
- ✦**Paso 6**: Despliegue en la nube (Heroku)
- ✦**Paso 7**: La base de datos: sequelize.js y SQLite
- ✦**Paso 8**: Desplegar en Heroku con Postgres
- ✦**Paso 9**: Lista de preguntas
- ✦**Paso 10**: Autoload de la DB
- ✦**Paso 11**: Crear preguntas
- ✦**Paso 12**: Validación de entradas
- ✦**Paso 13**: Editar preguntas
- ✦**Paso 14**: Borrar preguntas
- ✦**Paso 15**: Crear comentario
- ✦**Paso 16**: Autenticación y sesión
- ✦**Paso 17**: Autorización
- ✦**Paso 18**: Moderación de comentarios
- ✦**Paso 19**: HTTPS - HTTP Seguro

Objetivo: Crear un pequeño portal Web con un juego de adivinanzas (quizes) usando MVC y vistas adaptables a móvil. Quiz ilustra también el uso de herramientas de gestión de proyectos.

El proyecto Quiz y sus versiones están en **GITHUB** https://github.com/jquemada/quiz-2015/commits/master y se ha desplegado y está operativo en **heroku** https://quiz-2015.herokuapp.com/



Proyecto Quiz: Servicio hipermedia



Proyecto Quiz: https://quiz-2015.herokuapp.com/

©

Quiz: Interfaz REST, DB y MVC

id	pregunta	resp.
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....

id	texto	publicar
1	Nuevo comentario	TRUE
2	Fue Sintra en el pasado	FALSE
3	Rome en ingles	TRUE
4	.....	

Método	Ruta	Acciones y controladores	Vistas
	Gestion de Session		
GET	/login	sessionController.**new**	session/new.ejs
POST	/login	sessionController.**create**	
GET	/logout	sessionController.**destroy**	
	Gestion de Quizes (preguntas)		
GET	/quizes	quizController.**index**	quizes/index.ejs
GET	/quizes/new	quizController.**new**	quizes/new.ejs
GET	/quizes/:id	quizController.**show**	quizes/show.ejs
GET	/quizes/:id/answer	quizController.**answer**	quizes/answer.ejs
POST	/quizes	quizController.**create**	
GET	/quizes/:id/edit	quizController.**edit**	quizes/edit.ejs
PUT	/quizes/:id	quizController.**update**	
DELETE	/quizes/:id	quizController.**destroy**	
	Gestion de Comentarios		
GET	/quizes/:id/comments/new	commentController.**new**	comments/new.ejs
POST	/quizes/:id/comments	commentController.**create**	
GET	/quizes/:id/comments/:id/publish	commentController.**publish**	

©

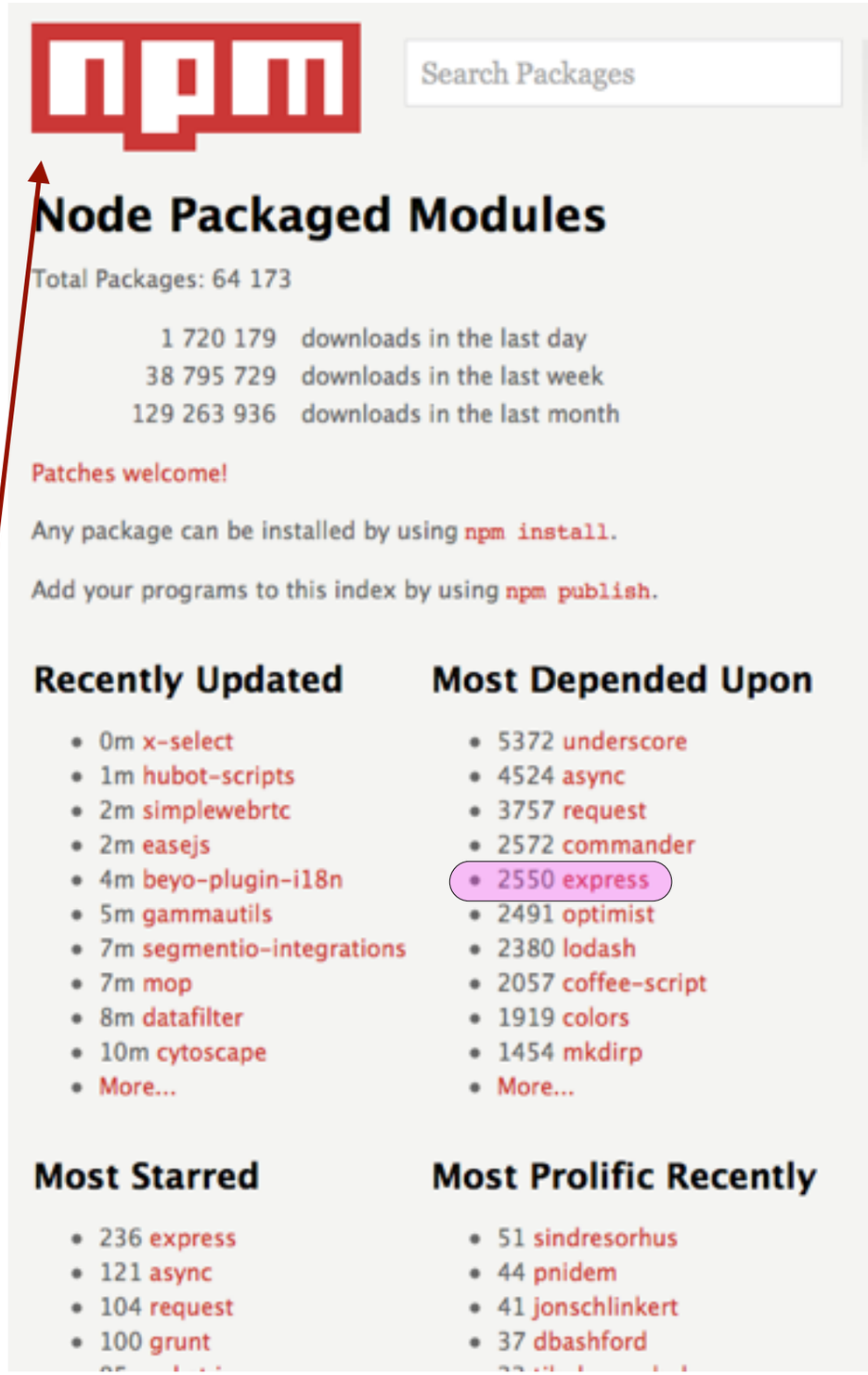
Final del tema

Quiz 1a : Esqueleto del proyecto con express-generador

Juan Quemada, DIT - UPM

Paquetes node.js: npm

- **Paquete**
  - programa empaquetado utilizable por terceras personas
    - comando del SO, programa independiente, librería, ..
- **npm (Node Package System)**
  - Ecosistema de gestión de **paquetes de node.js**
    - Soporta la creación e instalación de paquetes
      - y sus dependencias (otros paquetes usados)
    - Tiene muchas otras funciones
- **repositorio npm** o **npm registry** : https://www.npmjs.org
  - servidor con un repositorio de paquetes de libre acceso
    - Permite instalación, publicación, búsqueda, ….
- La **comunidad node.js crece muy deprisa**
  - gracias a la sencillez y eficacia de **npm** y del **registry**



Comando npm

```
venus:s7 jq$ npm -h

Usage: npm <command>

where <command> is one of:
    add-user, adduser, apihelp, author, bin, bugs, c, cache,
    completion, config, ddp, dedupe, deprecate, docs, edit,
    explore, faq, find, find-dupes, get, help, help-search,
    home, i, info, init, install, isntall, la, link, list, ll,
    ln, login, ls, outdated, owner, pack, prefix, prune,
    publish, r, rb, rebuild, remove, restart, rm, root,
    run-script, s, se, search, set, show, shrinkwrap, star,
    start, stop, submodule, tag, test, tst, un, uninstall,
    unlink, unpublish, unstar, up, update, version, view,
    whoami

npm <cmd> -h     quick help on <cmd>
npm -l           display full usage info
npm faq          commonly asked questions
npm help <term>  search for help on <term>
npm help npm     involved overview

Specify configs in the ini-formatted file:
    /Users/jq/.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@1.1.63 /usr/local/lib/node_modules/npm
venus:s7 jq$
```

- **npm** es también el nombre de un comando de gestión de paquetes de node.js
  - Se utiliza fundamentalmente para instalar, descubrir, desarrollar o publicar paquetes node.js
- El comando npm se comunica con el servidor central cuando lo necesita
  - https://www.npmjs.org
- El comando npm entiende el formato de un paquete y lo procesa adecuadamente
  - tiene mucha funcionalidad
    - Doc: https://www.npmjs.org/doc/
    - La documentación también se puede obtener por consola como cualquier comando (tipo UNIX).

npm install <paquete> // instala un paquete del servidor

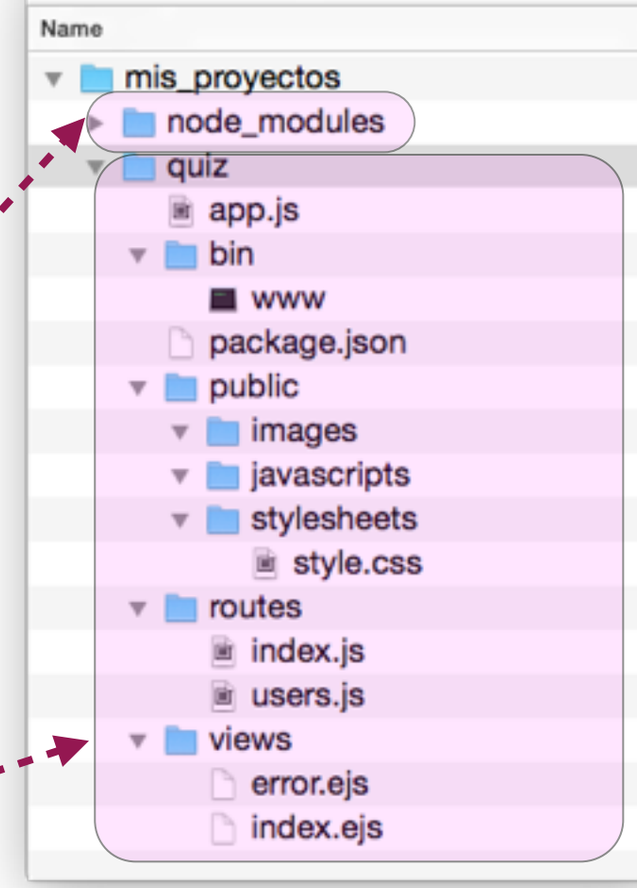
npm init <paquete> // inicializa paquete (fichero package.json)

npm publish <paquete> // publica un paquete en el servidor

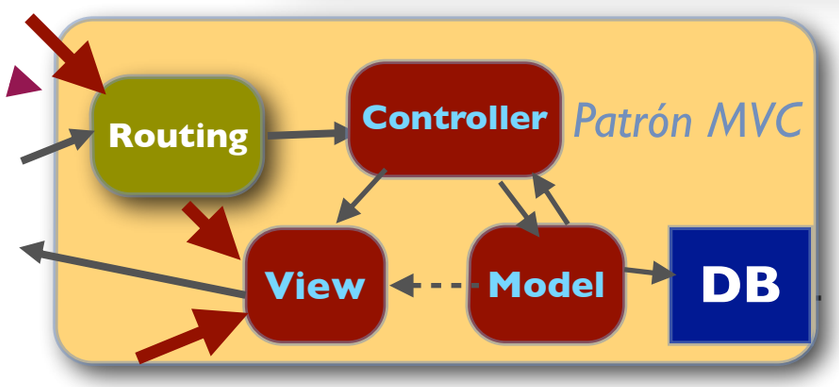
.....

Quiz 1: Creación del esqueleto

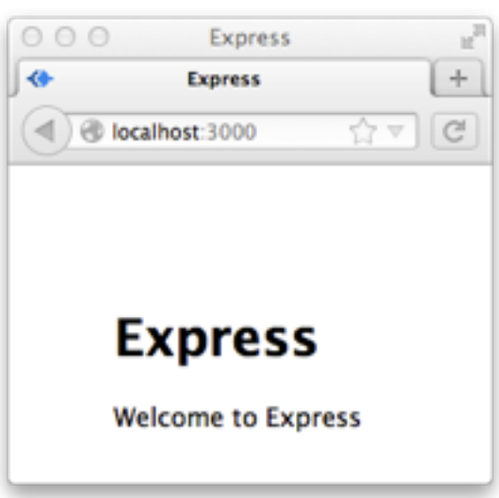
Objetivo: El paso del diseño Quiz 1 crea con el comando **express (generator)** el directorio del proyecto con un esqueleto de programas y directorios que iremos enriqueciendo en los siguientes pasos



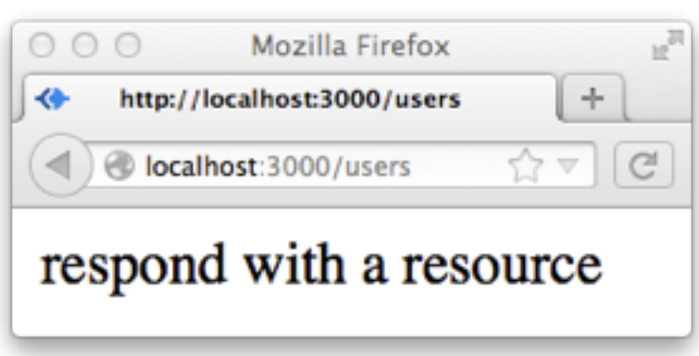
- **Paso 1**: Instalar paquete **express-generator**
  - Generador de esqueletos de proyectos express
- **Paso 2**: generar el esqueleto del proyecto Quiz
  - Utilizando express-generator
- **Paso 3**: Guardar versión (commit) con git
  - Antes de guardarlo conviene ejecutarlo y probarlo



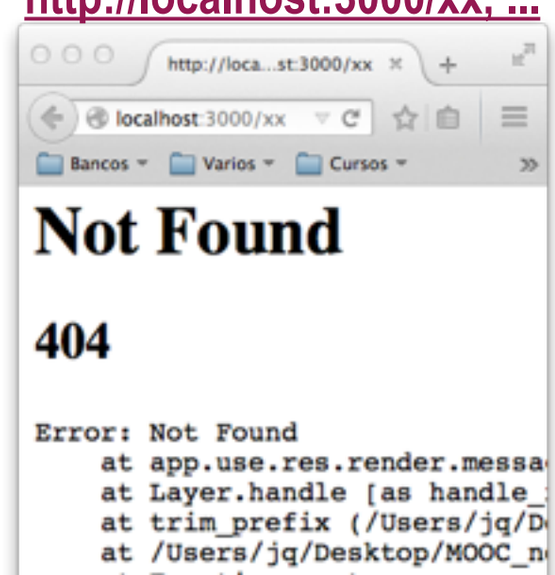
http://localhost:3000/



http://localhost:3000/users



http://localhost:3000/xx, ...

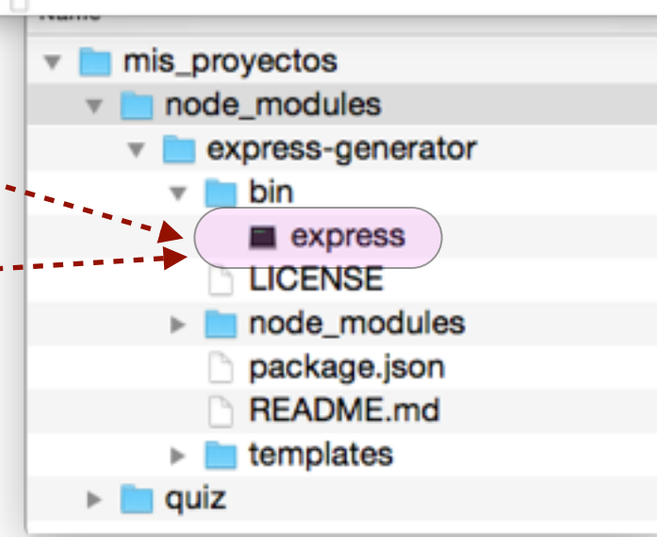


©

express-generator

- **express-generator** es un paquete npm que
  - Instala el comando **express**
    - desde el servidor central **www.npmjs.org**
- **express** se instala con “npm install” en el directorio
  - **node_modules/express-generator/bin/**
- El comando **express** crea proyectos express
  - Se invoca desde mis_proyectos con el path completo
    - **“node_modules/express-generator/bin/express …….”**

```
venus:s9 jq$
venus:s9 jq$ node_modules/express-generator/bin/express --help

  Usage: express [options] [dir]

  Options:

    -h, --help          output usage information
    -V, --version       output the version number
    -e, --ejs           add ejs engine support (defaults to jade)
        --hbs           add handlebars engine support
    -H, --hogan         add hogan.js engine support
    -c, --css <engine>  add stylesheet <engine> support (less|sty
    -f, --force         force on non-empty directory

venus:s9 jq$
```



// Recomendamos crear un directorio para todos los
// proyectos generados con express-generator:

..$ mkdir mis_proyectos
..$ cd mis_proyectos

// Se instala la versión 4.9.0 de express-generator en el
// directorio **node_modules/express-generator** con

..$ npm install express-generator@4.9.0

// Quiz está probado solo con v.4.9.0 y no se
// garantiza que funcione con otras versiones.

https://www.npmjs.org

Search Packages

express-generator

```
venus-5:s8 jq$ npm install express-generator@4.9.0
express-generator@4.9.0 node_modules/express-generator
├── commander@1.3.2 (keypress@0.1.0)
└── mkdirp@0.5.0 (minimist@0.0.8)
venus-5:s8 jq$
```

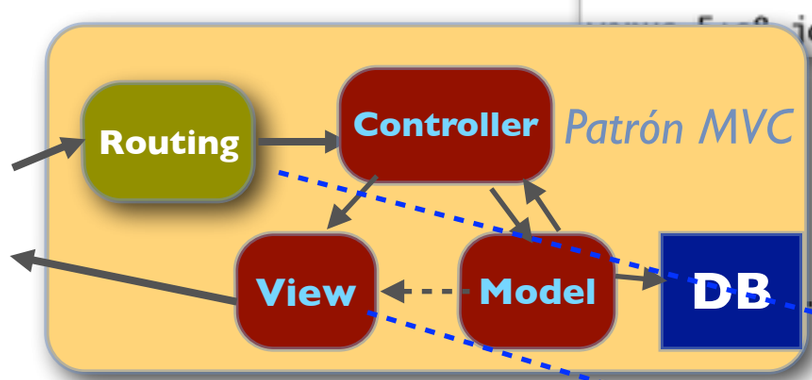
194 downloads in the last day

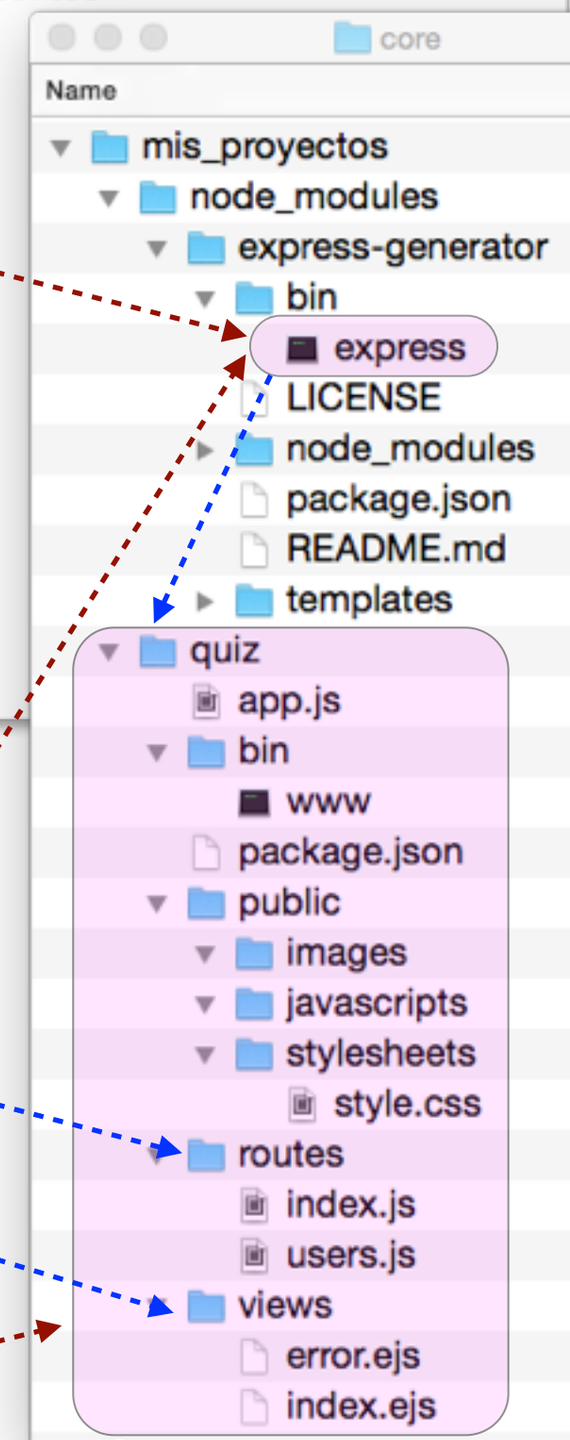
©

Crear proyecto Quiz

- ✦Al ejecutar el **comando express**
  - Se crea el esqueleto del proyecto quiz
    - se debe invocar con el path completo al comando
    - Opción **--ejs**: instala el generador de vistas EJS
    - Parámetro **quiz**: directorio del proyecto/esqueleto
      - Donde se crea el esqueleto del proyecto
- ✦Esqueleto de quiz
  - Directorio donde albergar
    - todos los elementos básicos del proyecto
      - Aplicación
      - Vistas
      - Enrutador
      - ……

```
venus-5:s8 jq$ node_modules/express-generator/bin/express --ejs quiz

   create : quiz
   create : quiz/package.json
   create : quiz/app.js
   create : quiz/public
   create : quiz/public/javascripts
   create : quiz/public/stylesheets
   create : quiz/public/stylesheets/style.css
   create : quiz/routes
   create : quiz/routes/index.js
   create : quiz/routes/users.js
   create : quiz/public/images
   create : quiz/views
   create : quiz/views/index.ejs
   create : quiz/views/error.ejs
   create : quiz/bin
   create : quiz/bin/www

   install dependencies:
     $ cd quiz && npm install

   run the app:
     $ DEBUG=quiz ./bin/www

venus-5:s8 jq$
```





// Invocamos así el comando **express** desde el directorio **mis_proyectos**

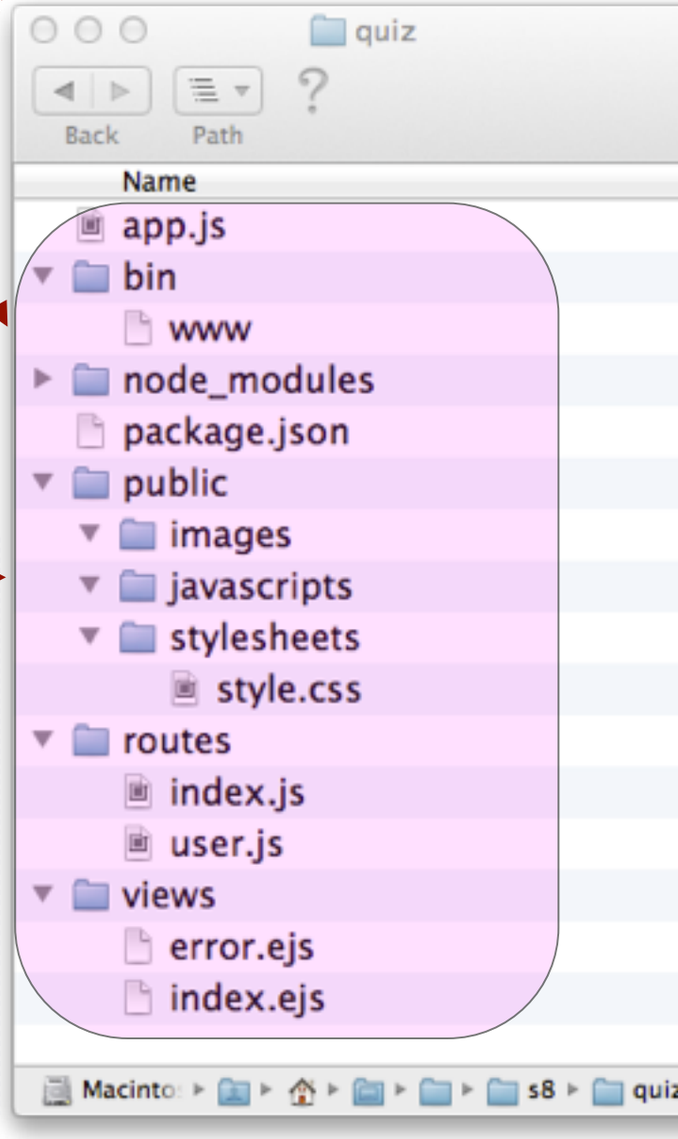
// la opción **--ejs** indica que se utilice el generador de vistas **EJS**
// el parametro **quiz** indica el nombre del directorio de proyecto creado

..mis_proyectos$ node_modules/express-generator/bin/express --ejs quiz

©

Guardar versión GIT

- express-generator genera el esqueleto del proyecto
  - Guardamos este primer paso de Quiz en el repositorio git
    - Creamos el proyecto y el repositorio con: “git init”
    - Indicamos lo que no queremos guardar en “.gitignore”
      - En mi caso (MAC) el directorio node_modules y el fichero .DS_store
    - Subimos los demás ficheros al indice con: “git add .”
    - Creamos la primera versión: git commit -m “esqueleto express-generator”
- Ficheros de configuración: empiezan por punto, p.e. .gitignore
  - no se ven en las ventanas y para editarlos se puede:
    - Copiar a fichero visible, p.e. “cp .gitignore xx”, editar y a la inversa
    - Editar con sublime text abriendo todo el directorio



..$ cd quiz // entramos en **directorio quiz**

..$ git init // iniciamos la gestión del proyecto quiz con git

// Identificamos los ficheros que no queremos guardar en **.gitignore**

..$ git add . // añadimos todo lo contenido en directorio al area de cambios

// generamos versión identificada como: espress-generator skeleton

..$ git commit -m “esqueleto espress-generator"

Final del tema

©

Quiz 1b: El esqueleto del proyecto

Juan Quemada, DIT - UPM

Ficheros creados por express-generador

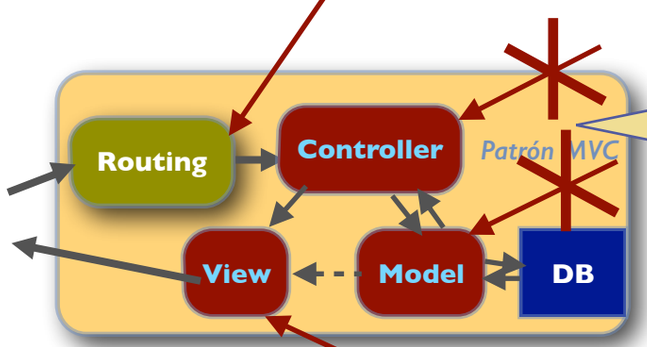
Aplicación express.js

Directorio con ejecutables: www - programa de arranque

Fichero de definición/configuración del paquete

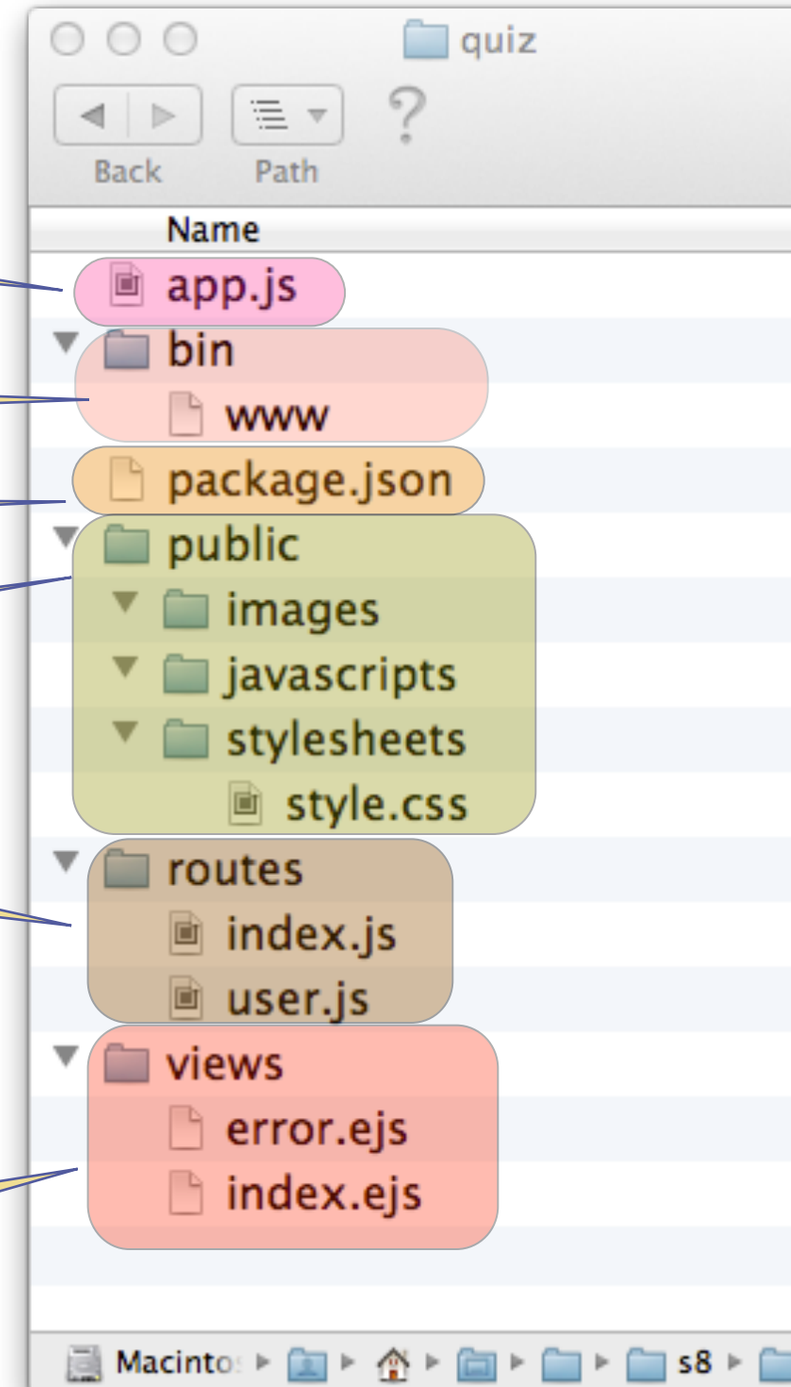
Directorios de recursos públicos: imágenes, estilos y librerías JavaScript

Enrutador



No se generan

Vistas

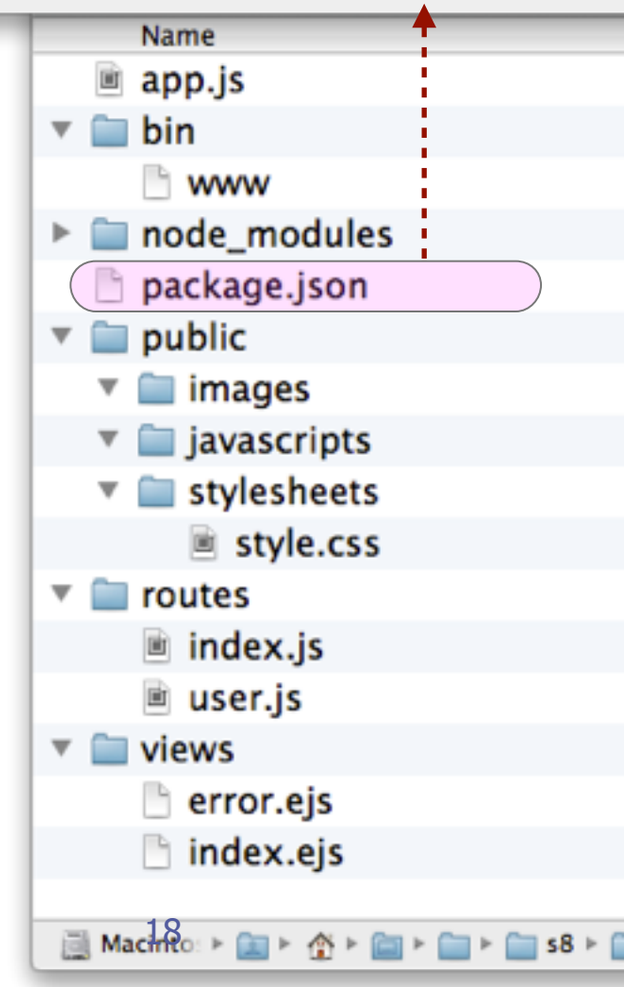


package.json

- **package.son** es un fichero en formato JSON que define
  - las características del paquete npm de este proyecto
    - Doc: https://www.npmjs.org/doc/files/package.json.html
- **express-generator** genera solo los siguientes parámetros:
  - name, version, private, scripts y dependencies
- Los paquetes usados en el proyecto son las “dependencies”
  - Se instalan con: **npm install**
- Los scripts son los comandos del proyecto
  - Suelen incluir: arranque (start), pruebas (test), despliegue (deploy), ..

```
// package.json define a través de las dependencias (dependencies:)
// los paquetes que nuestro proyecto necesita. Los instalamos así

..$ npm install  // instala todos los paquetes indicados en package.json
                 // quiz/node_modules contiene el software instalado

..$ npm start    // invoca script de arranque (start) en package.json
```

```
{
  "name": "quiz",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~4.9.0",
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "morgan": "~1.3.0",
    "serve-favicon": "~2.1.3",
    "debug": "~2.0.0",
    "ejs": "~0.8.5"
  }
}
```



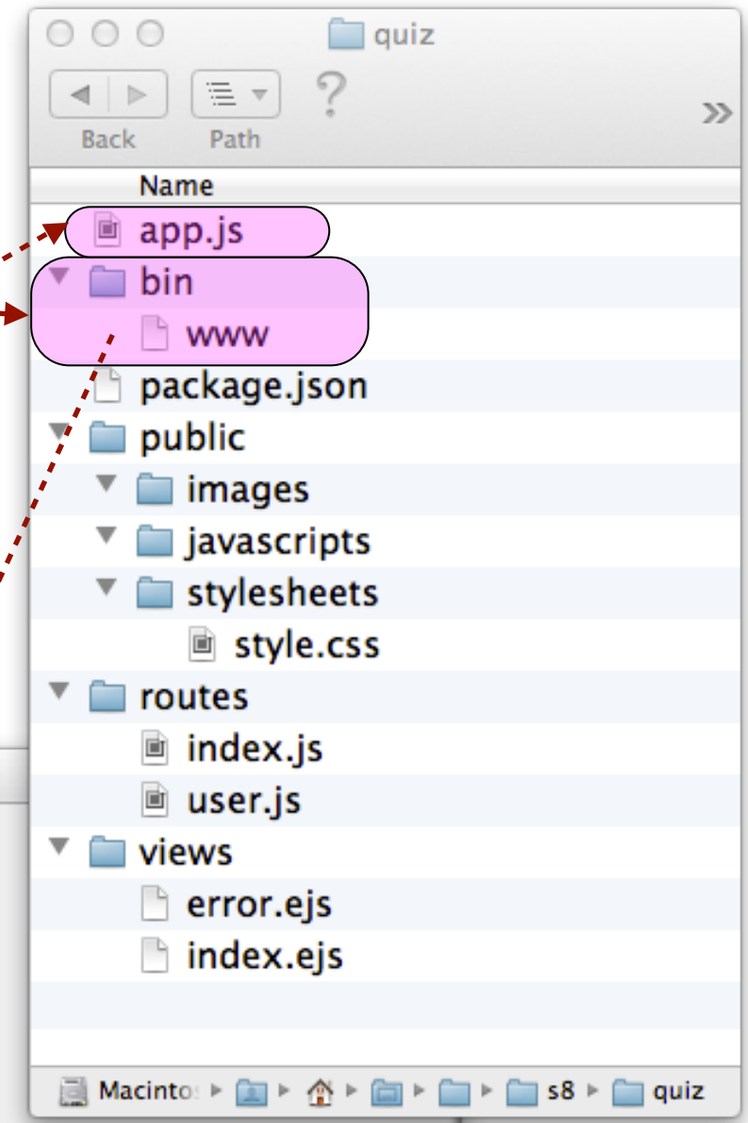
// arranque de la aplicación

..$ node bin/www

..$ npm start

Programa de arranque

- El fichero **bin/www**
  - es el programa de arranque de la aplicación
- La aplicación está en el fichero **app.js**
  - Se importa como un modulo en **bin/www**
    - que arranca el servidor express



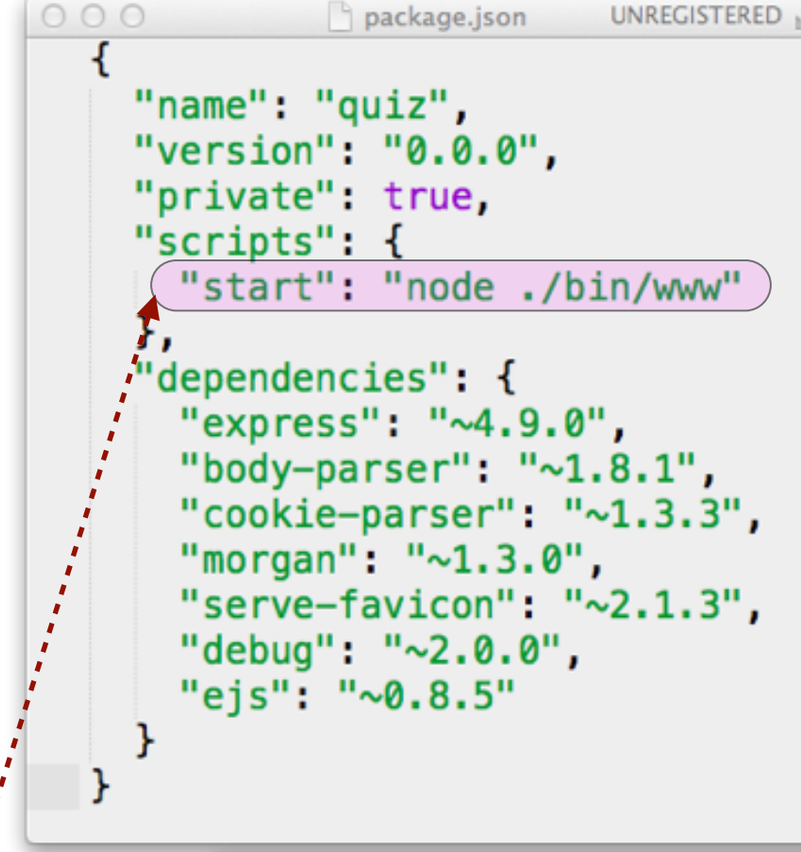
www

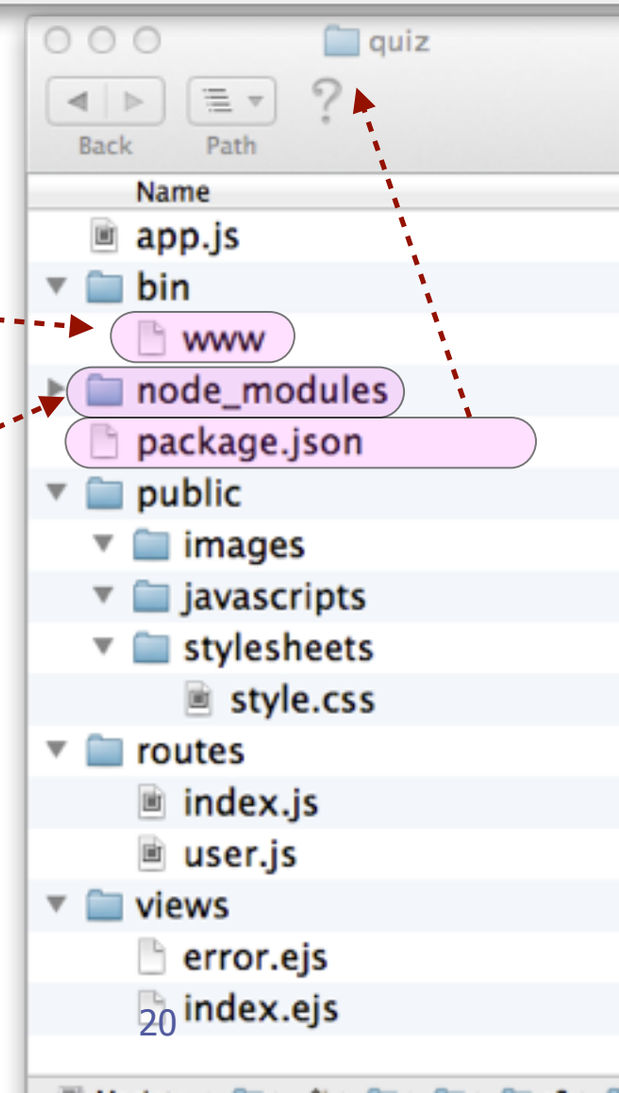
```
#!/usr/bin/env node
var debug = require('debug')('quiz');
var app = require('../app');

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

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

Ejecución de quiz

- **npm install** instala las dependencias definidas en package.json
  - son todos los paquetes necesarios para ejecutar express.js
- Con las dependencias instaladas se arranca el servidor:
  - Con el script de arranque (**start**) definido en package.json
    - **npm start** -> **"start": "node ./bin/www"**
  - Invocando directamente bin/www con node
    - **node ./bin/www**

```
// package.json define a través de las dependencias (dependencies:)
// los paquetes que nuestro proyecto necesita. Los instalamos así

..$ cd quiz              // entramos en directorio quiz

..$ npm install  // instala todos los paquetes indicados en package.json
                 // quiz/node_modules contiene el software instalado

..$ node bin/www  // invocación directa de bin/www con node
..$ npm start     // script de arranque a través de package.json
```





©

app.js I

app.js UNREGISTERED

```
var express = require('express');
var path = require('path');
var favicon = require('serve-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');

// uncomment after placing your favicon in /public
//app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

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

importar paquetes con middlewares

importar enrutadores

crear aplicación

instalar generador de vistas EJS

instalar middlewares

instalar enrutadores

app.js: instala MWs con **app.use()**.

app.js importa primero con require(….)los MWs que va a instalar.

Luego instala los MWs en el mismo orden en que deben ejecutarse cuando llegue una transacción HTTP.

Cada transacción HTTP ejecuta los MWs instalados en el mismo orden en que han sido instalados.

app.js II

app.js UNREGISTERED

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

// undefined route, catch 404 and forward 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  // print 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: {} // don't print err (empty object)
    });
});

module.exports = app;
```

asociar rutas a sus gestores

resto de rutas: genera error 404 de HTTP

gestión de errores durante el desarrollo

gestión de errores de producción

exportar app para comando de arranque

Se crean e instalan 3 MWs

1) El primer MW atiende las rutas no atendidas por los MWs anteriores, generando un error HTTP 404 (recurso no encontrado).

2) MW gestor de errores solo para la fase de desarrollo del proyecto.

3) MW gestor de errores solo para la fase de despliegue del proyecto.

Por ultimo exporta la aplicación, de forma que pueda ser importada en bin/www

Rutas REST

app.js

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

......

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

ruta "/"

index.js — UNREGISTERED

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

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

module.exports = router;
```

routes/index.js

index.ejs — UNREGISTERED

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet'
          href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

views/index.ejs

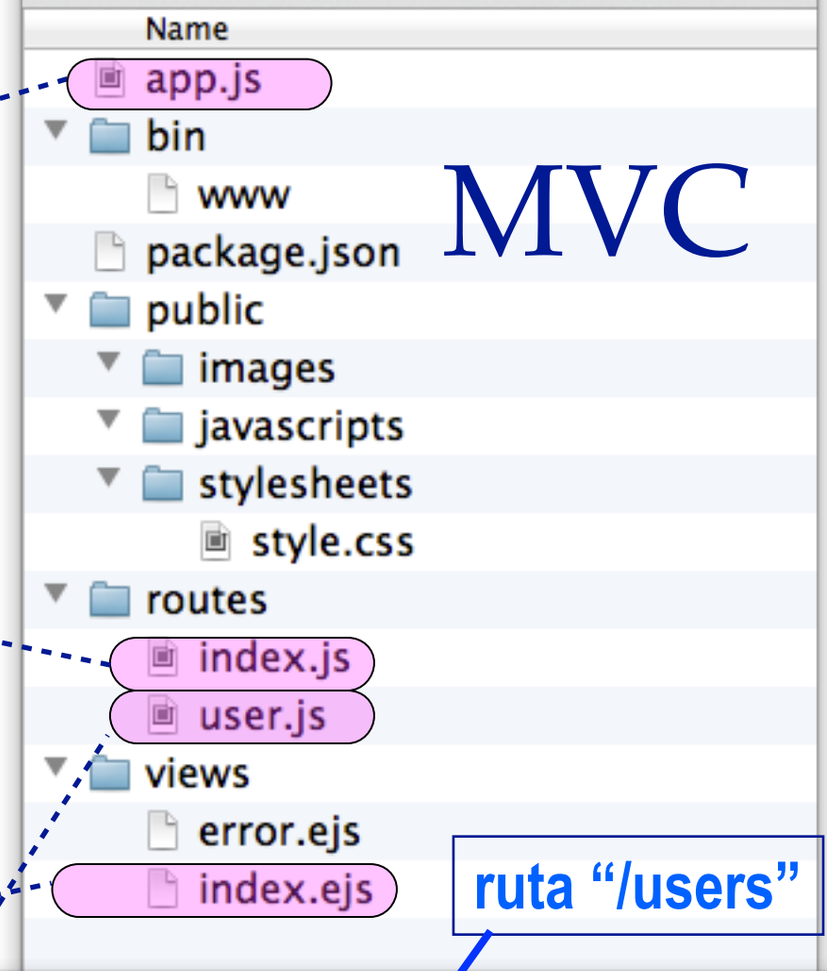
Express

Express

localhost:3000

Express

Welcome to Express



MVC

ruta "/users"

users.js — UNREGISTERED

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

/* GET users listing. */
router.get('/', function(req, res) {
  res.send('respond with a resource');
});

module.exports = router;
```

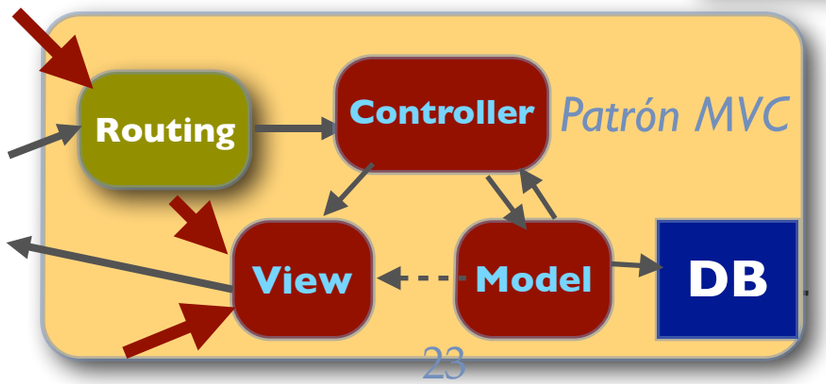
routes/user.js

Mozilla Firefox

http://localhost:3000/users

localhost:3000/users

respond with a resource



© Juan Quemada - UPM-DIT

Errores

app.js UNREGISTERED

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

// undefined route, catch 404 and forward 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 // print 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: {} // don't print err (empty object)
    });
});

module.exports = app;
```

rutas indefinidas

p.e. localhost:3000/xx

desarrollo

producción

©

Name

- app.js
- bin
  - www
- package.json
- public
  - images
  - javascripts
  - stylesheets
    - style.css
- routes
  - index.js
  - user.js
- views
  - error.ejs
  - index.ejs

error.ejs UNREGISTERED

```
<h1><%= message %></h1>
<h2><%= error.status %></h2>
<pre><%= error.stack %></pre>
```

http://loca...st:3000/xx

localhost:3000/xx

Bancos Varios Cursos

Not Found

404

```
Error: Not Found
    at app.use.res.render.messa
    at Layer.handle [as handle_
    at trim_prefix (/Users/jq/D
    at /Users/jq/Desktop/MOOC_n
    at Function.proto.process_p
```

Final del tema

©

Quiz 2: Primera página y favicon

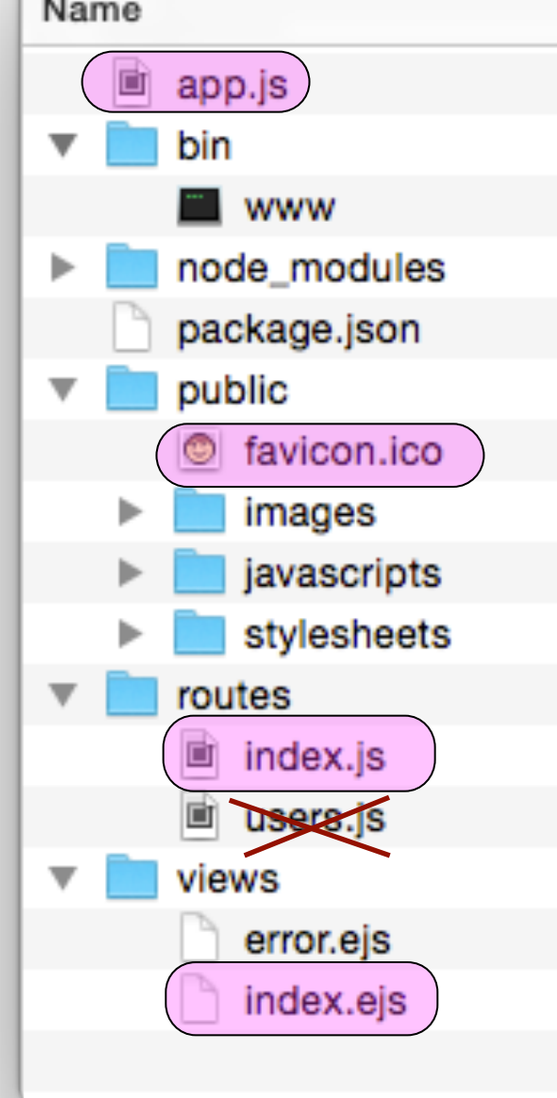
Juan Quemada, DIT - UPM

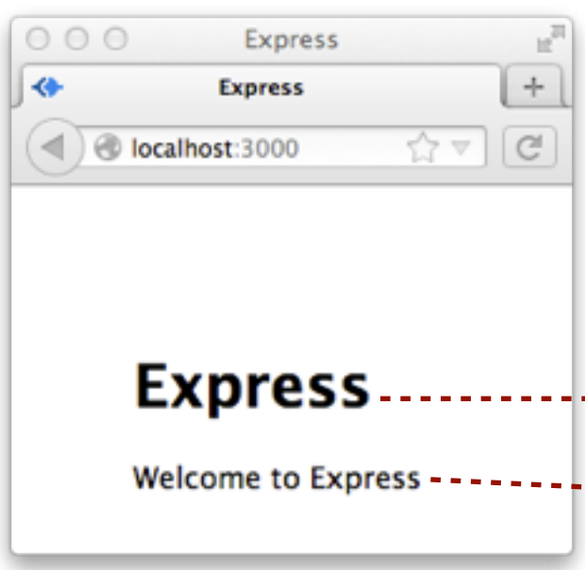
©

Quiz 2: Primera página y favicon

Objetivo: El paso del diseño Quiz 2 modificamos el esqueleto del proyecto para transformar la pagina (**views/index.ejs**) asociada a la ruta **/** en la página de entrada a nuestro portal Quiz. Además aprovechamos para **quitar la ruta /users** y **añadir un favicon**.

- **Paso 1**: Se transforma la página de entrada en: **GET /**
  - En página de bienvenida al portal
    - Modificar ficheros **routes/index.js** y **views/index.ejs**
- **Paso 2**: quitamos ruta **/users** creada por express-generator
  - borramos **routes/users.js** y modificamos **app.js**
- **Paso 3**: añadimos favicon
- **Paso 4**: Guardamos versión (commit) con git





EJS: Embbeded JavaScript

index.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>Bienvenido a <%= title %></h1>
    <p>El portal donde podrá crear sus propios juegos!</p>
  </body>
</html>
```

vista index.ejs

- EJS = Javascript embebido en HTML.
- Los ficheros de vistas EJS contienen texto HTML que incluye código JavaScript o comandos delimitados entre las marcas `<%` código `%>`.

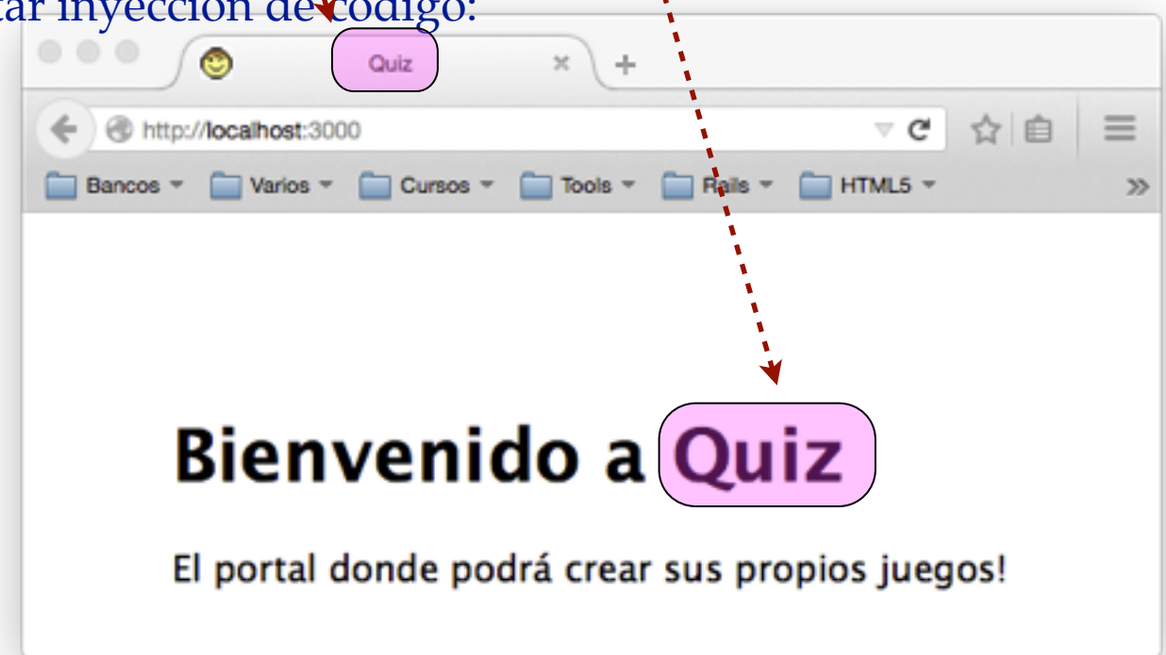
- código javascript entre las marcas `<%` código `%>`.

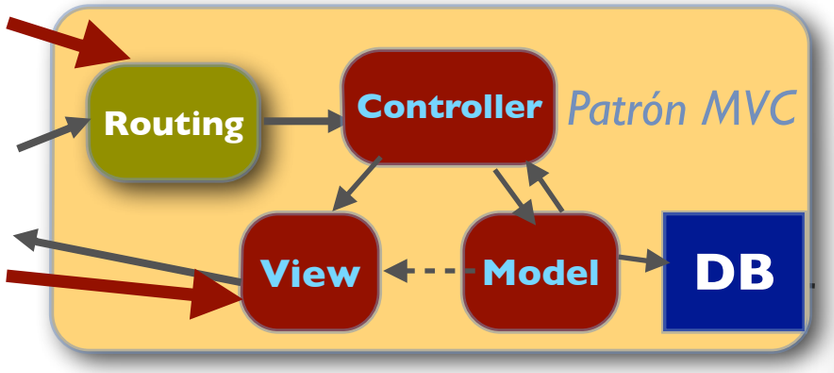
- expresiones javascript entre las marcas `<%=` expresión `%>`.
  - El valor de las expresiones se incorpora al texto HTML.
  - Previamente se escapan los caracteres conflictivos para evitar inyección de código:
    - `<` se sustituye por `<`
    - `>` se sustituye por `>`
    - `&` se sustituye por `&`
    - ...

- expresiones javascript entre las marcas `<%-` expresión `%>`.
  - El valor de las expresiones se incorpora al texto HTML.
  - No se escapan los caracteres conflictivos.

- inclusión de ficheros con `<% include` *path_a_fichero* `%>`.

- Documentación: https://github.com/visionmedia/ejs

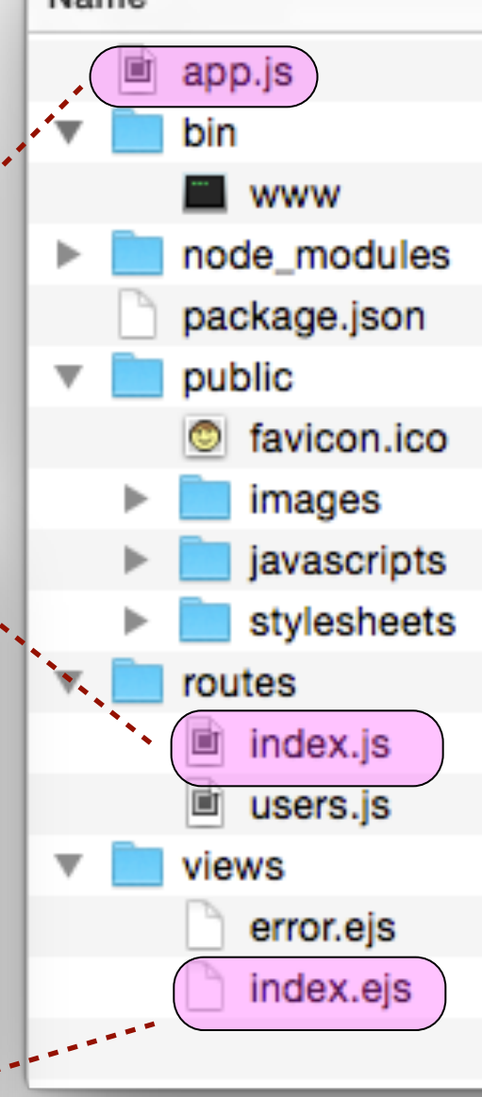




app.js UNREGISTERED

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

app.js



Las **vistas EJS** se generan en el objeto de respuesta **res** y se envían al cliente con

res.render(vista, params)

- **vista** es el nombre del fichero con la vista (con extensón **.ejs)**
- **params** es un objeto con parámetros a sustituir en la vista.

index.js UNREGISTERED

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

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

module.exports = router;
```

Ruta /

index.ejs UNREGISTERED

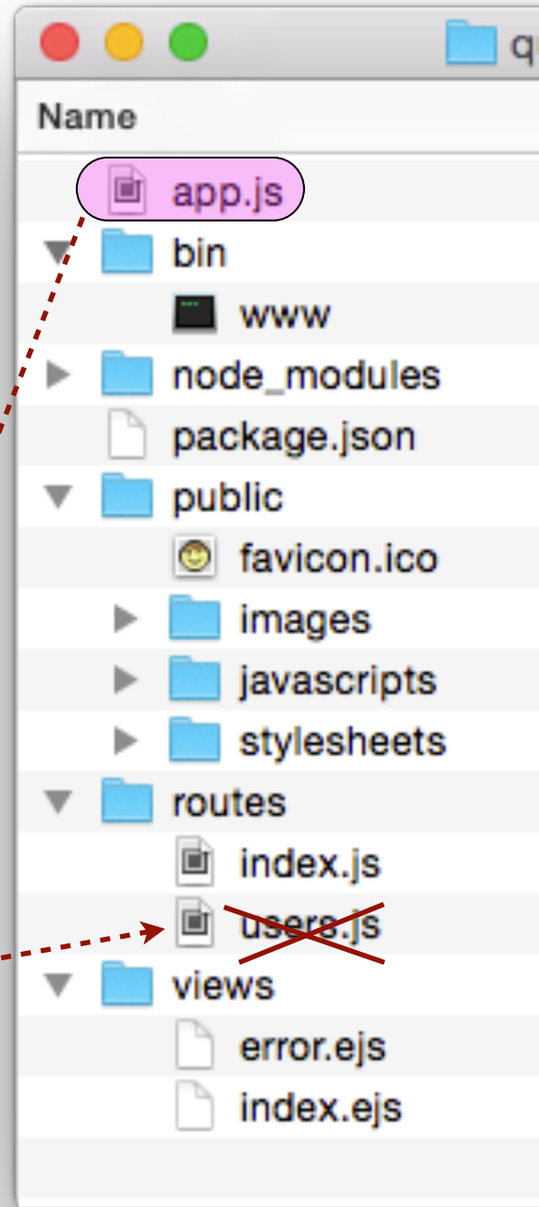
```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>Bienvenido a <%= title %></h1>
    <p>El portal donde podrá crear sus propios juegos!</p>
  </body>
</html>
```

vista index.ejs

Quitar ruta /user

- express-generator genera **2 rutas**
  - La ruta **"/"** de acceso a la página de entrada
    - Esta ruta se reutiliza para la página principal
  - La ruta **"/users"** para la gestión de usuarios
    - Esta ruta **no se reutiliza** y la **borramos**

- Para borrar la ruta **/users** debemos
  - Borrar su fichero de rutas **routes/users.js**
  - Borrar en **app.js** la sentencia
    - **var users = require('./routes/users');**
  - Borrar en app.js la sentencia
    - **app.use('/users', users);**

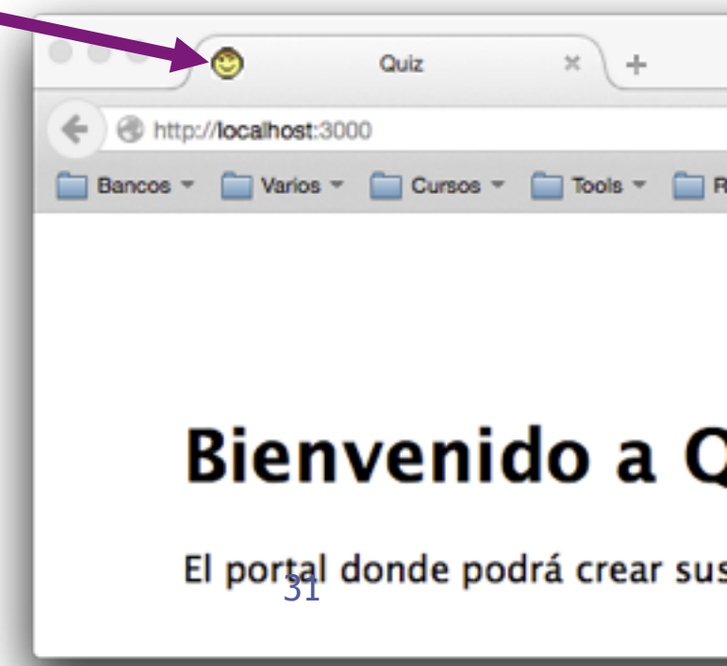


app.js UNREGISTERED

```
8  var routes = require('./routes/index');
9  var users = require('./routes/users');
10 .......
11 app.use('/', routes);
12 app.use('/users', users);
```

Favicon

favicon.ico

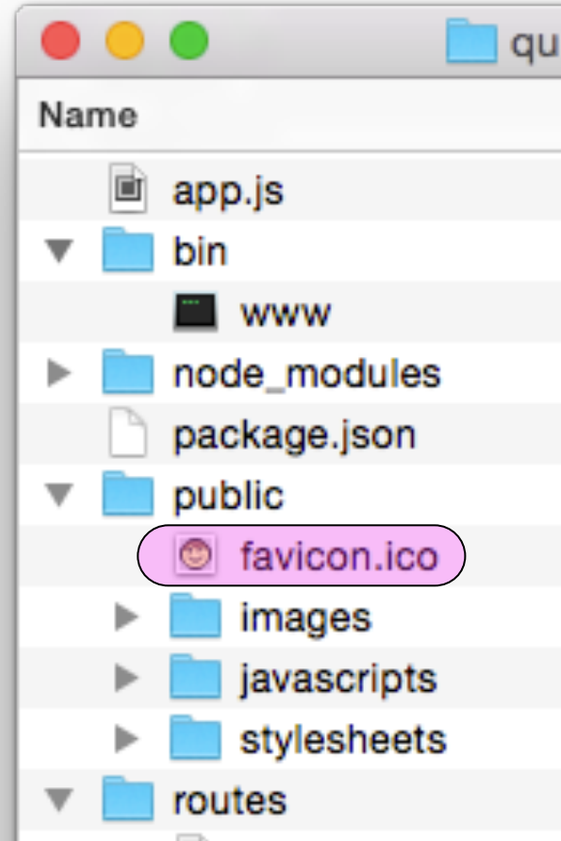


- Se puede mostrar un icono en la pestaña de una página
  - Se denomina **favicon** (favourite icon) yes de 16x16 pixels
    - Se suele guardar en el fichero **favicon.ico**
      - Ejemplo de editor de favicon: http://www.favicon.cc/

- **app.js** incluye el MW **serve-favicon** que sirve un favicon
  - El MW se activa descomentarizando su instalación en app.js
    - Además hay que incluir el fichero “favicon.ico” en el directorio public

app.js UNREGISTERED

```
// uncomment after placing your favicon in /public
// app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(logger('dev'));
```



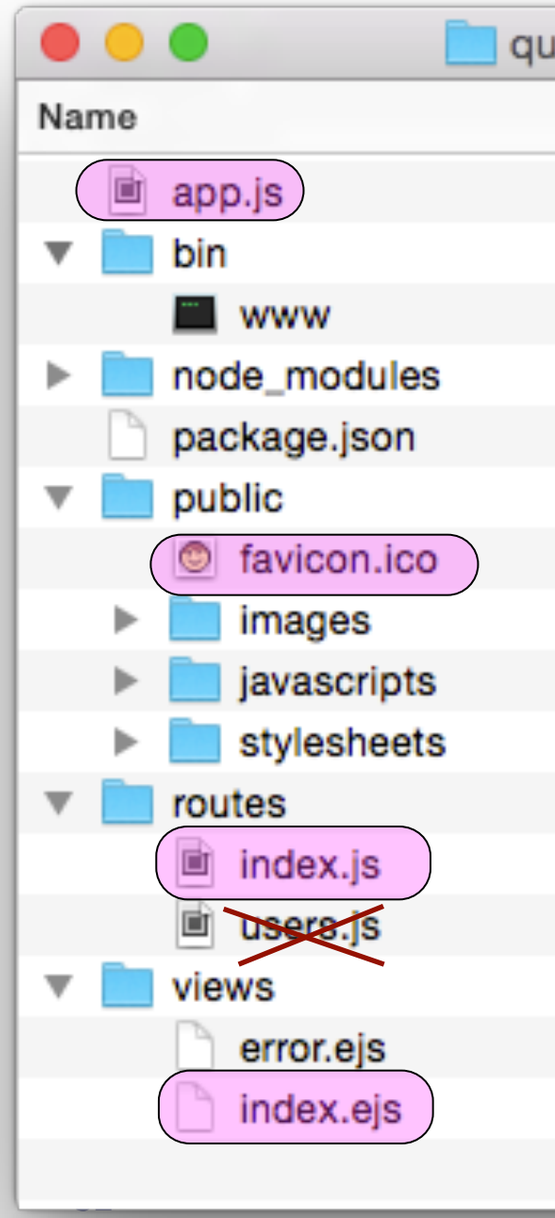
- Hoy se recomienda incluir el favicon desde la página HTML
  - Con un enlace al icono en la cabecera de la página
    - <link rel="icon" href="favicon.ico" type="image/x-icon" />

©

Guardar versión GIT

- ✦Para guardar los cambios debemos incluirlos en el índice (área de cambios)
  - Con el comando: **git add .**
- ✦Generamos una versión en la rama master
  - Con el comando: **git commit -m “identificación de versión”**
- ✦El repositorio **.git** contiene ahora las 2 primeras versiones
  - podemos verlas con: **git log** o con **git log --oneline**

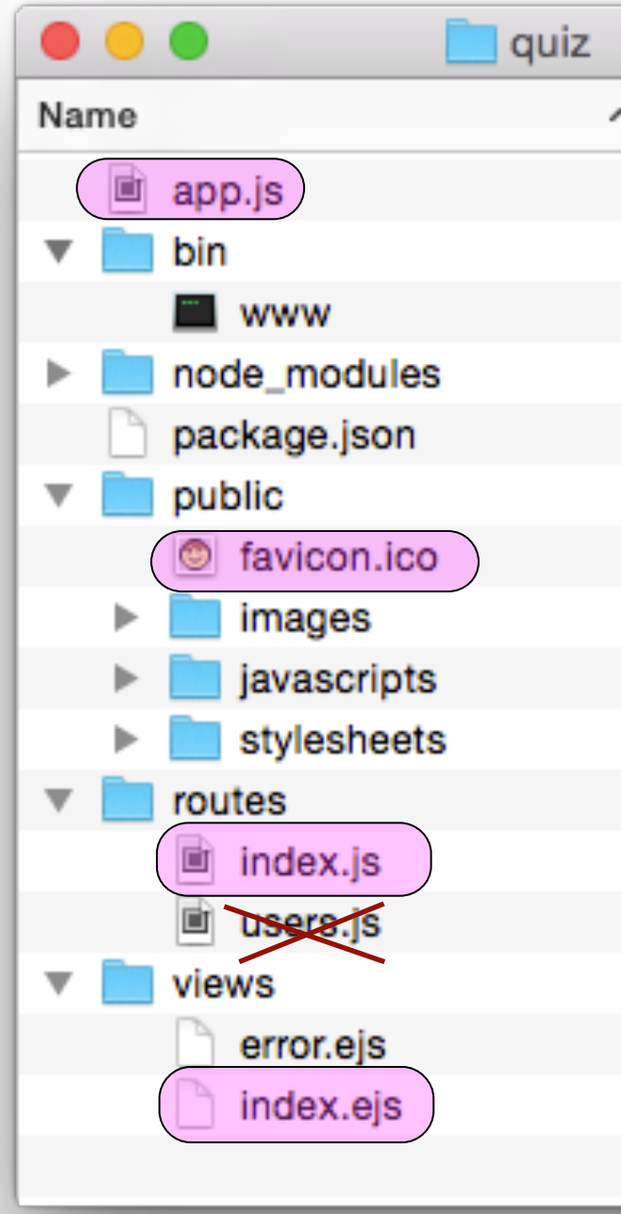
```
..$ git add .   // añadimos todo lo contenido en directorio al area de cambios

        // generamos versión identificada como: Primera página y favicon
..$ git commit -m “Primera página y favicon”
[master 984f3ee] Primera página y favicon
 5 files changed, 5 insertions(+), 15 deletions(-)
 create mode 100644 public/favicon.ico
 delete mode 100644 routes/users.js

        // listamos los commits en formato de una línea con
..$ git log --oneline
6fa18d0 Primera página y favicon
49b1f9c esqueleto express-generator
..$
```



©

git status: el índice

```
// git status muestra el estado del índice o área de cambios

..$ git status                                  // los cambios están sin indexar
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

                modified:   app.js
                modified:   routes/index.js
                deleted:    routes/users.js
                modified:   views/index.ejs

Untracked files:
  (use "git add <file>..." to include in what will be committed)

                public/favicon.ico

no changes added to commit (use "git add" and/or "git commit -a")

..$ git add .                                   // añadimos todos los cambios al índice

..$ git status                                  // todos los cambios están en el índice
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

                modified:   app.js
                new file:   public/favicon.ico
                modified:   routes/index.js
                deleted:    routes/users.js
                modified:   views/index.ejs

..$ git commit -m "Primera página y favicon"          // generar versión
[favicon 83c5e74] Primera página y favicon
 5 files changed, 5 insertions(+), 15 deletions(-)
 create mode 100644 public/favicon.ico
 delete mode 100644 routes/users.js

..$ git status                                  // la nueva versión no tiene cambios
nothing to commit, working directory clean
..$
```



Final del tema

Ejercicio: Añadir página de autores a Quiz y desplegar en Heroku

Para esta entrega se debe añadir a la versión Quiz 2 “Primera página y favicón” una página adicional de autor que estará accesible a través de la primitiva del interfaz: “GET /author”.

La página principal tendrá un enlace adicional para acceder a la nueva página.

La nueva página de autor tendrá un enlace en algún lugar, que permita volver a la página principal.

La página incluirá el nombre de los 2 autores (miembros de la pareja), una foto y un pequeño video de presentación del portal.

Recomendación: Hacer una rama en un clon del proyecto Quiz desde Quiz 2, que añada dicha funcionalidad.

Quiz 3: Primera pregunta

Juan Quemada, DIT - UPM

Quiz 3: Primera pregunta

Objetivo: crear la primera pregunta "Capital de Italia" y su respuesta. Se debe crear para ello el directorio de controladores y quiz_controller.js.

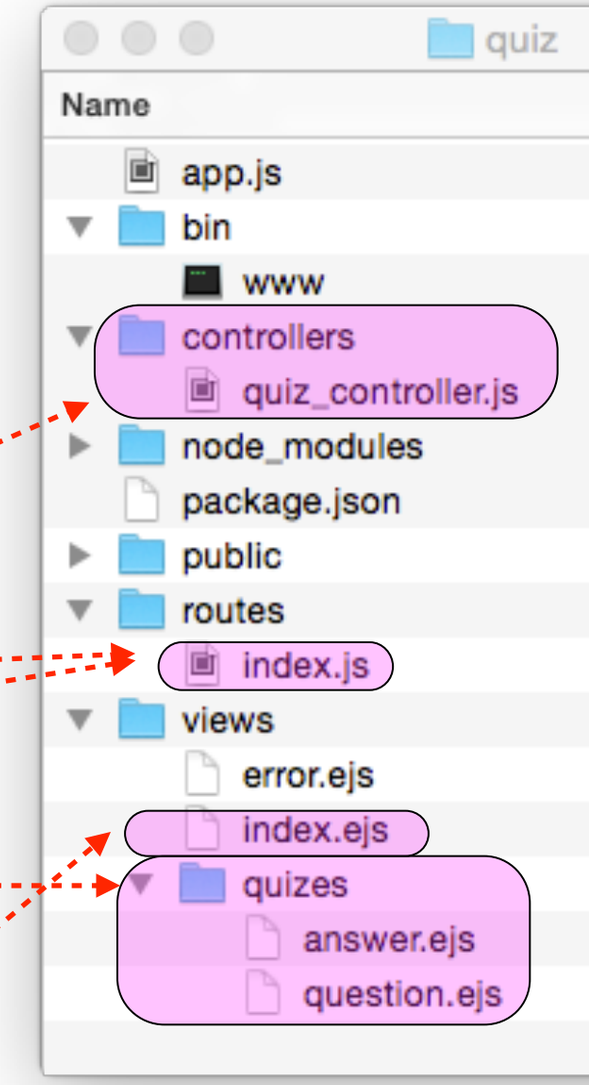
✦**Paso 1**: Añadir MVC de **pregunta** y **respuesta**

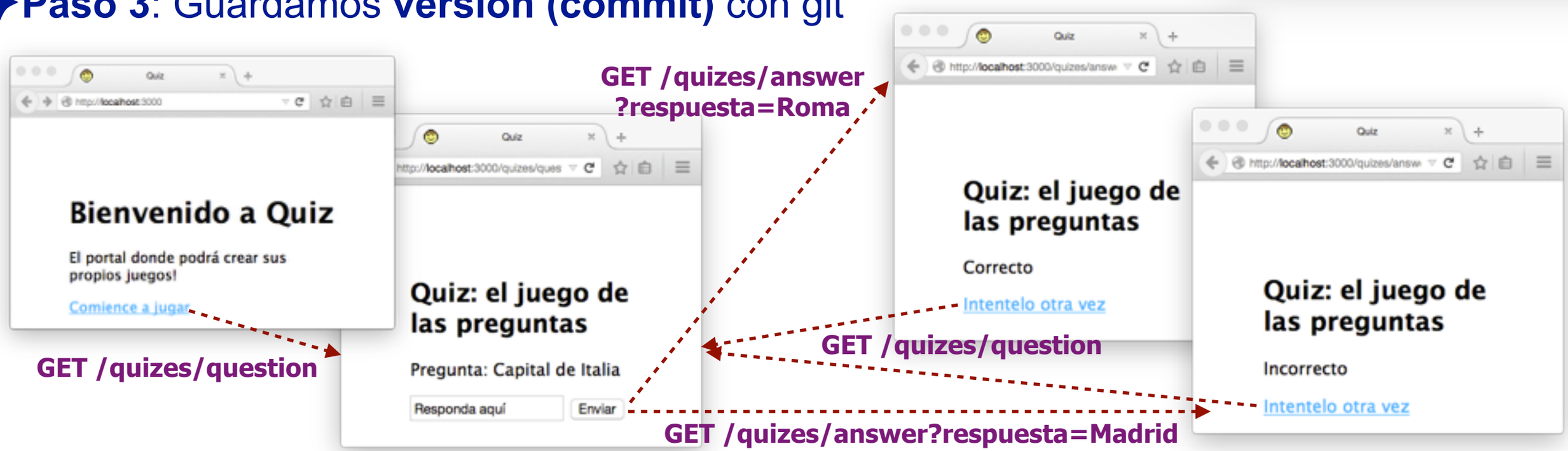
- a: Crear directorio **controllers** y el controlador **quiz_controller.js**
- b: Importar **quiz_controller.js** en **routes/index.js.js**
- c: Introducir nuevas rutas en el enrutador **routes/index.js**
  - Rutas: **GET /quizes/question** y **GET /quizes/answer**
- d: Crear directorio **view/quizes** y las vistas **question.ejs** y **answer.ejs**

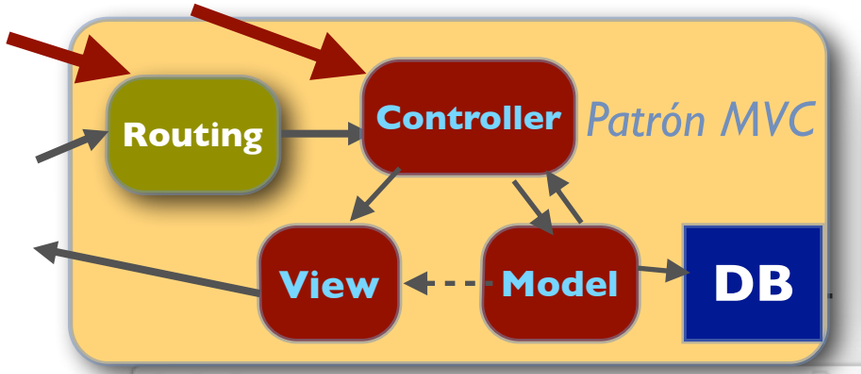
✦**Paso 2**: Añadimos enlace a la pregunta en la página de entrada

- Así enlazamos la pregunta desde la página de entrada (hipermedia)

✦**Paso 3**: Guardamos **versión (commit)** con git







app.js UNREGISTERED

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

app.js

index.js UNREGISTERED

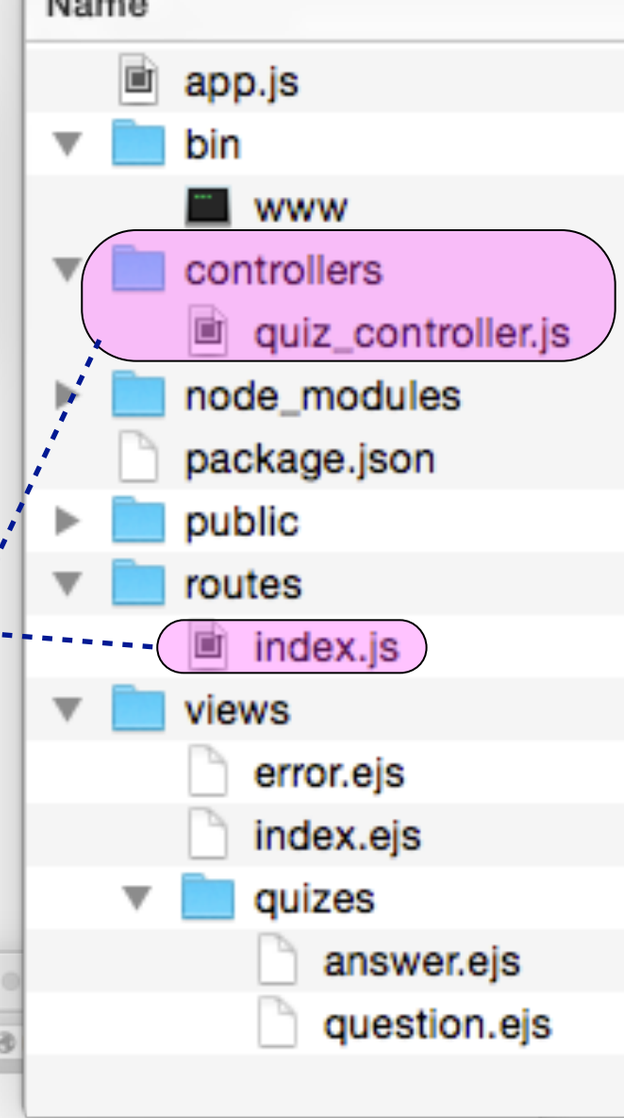
Pasos 1b y 1c: rutas /quizes/....

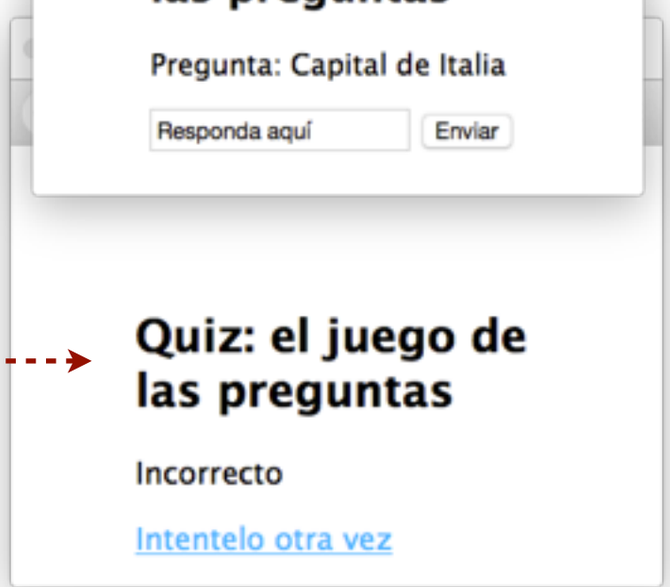
```
var express = require('express');
var router = express.Router();

var quizController = require('../controllers/quiz_controller');

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

router.get('/quizes/question', quizController.question);
router.get('/quizes/answer',   quizController.answer);

module.exports = router;
```

quiz_controller.js UNREGISTERED

```
// GET /quizes/question
exports.question = function(req, res) {
  res.render('quizes/question', {pregunta: 'Capital de Italia'});
};

// GET /quizes/answer
exports.answer = function(req, res) {
  if (req.query.respuesta === 'Roma'){
    res.render('quizes/answer', {respuesta: 'Correcto'});
  } else {
    res.render('quizes/answer', {respuesta: 'Incorrecto'});
  }
};
```

Paso 1a: controllers/quiz_controller.js





Paso 1a: controllers/quiz_controller.js

```
// GET /quizes/question
exports.question = function(req, res) {
    res.render('quizes/question', {pregunta: 'Capital de Italia'});
};

// GET /quizes/answer
exports.answer = function(req, res) {
    if (req.query.respuesta === 'Roma'){
        res.render('quizes/answer', {respuesta: 'Correcto'});
    } else {
        res.render('quizes/answer', {respuesta: 'Incorrecto'});
    }
};
```

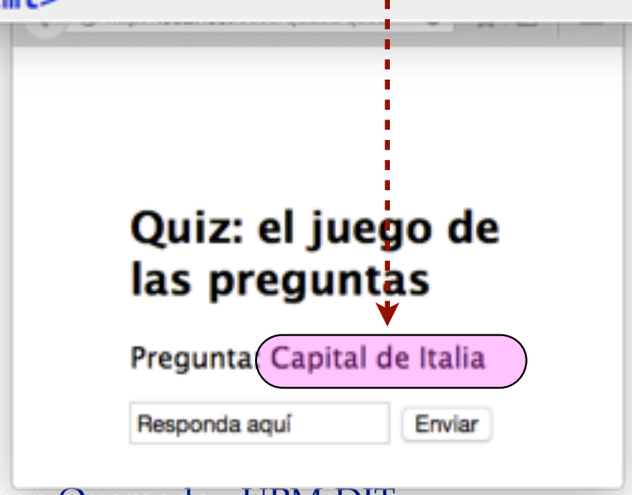
- www
- controllers
  - quiz_controller.js
- node_modules
- package.json
- public
- routes
  - index.js
- views
  - error.ejs
  - index.ejs
  - quizes
    - answer.ejs
    - question.ejs

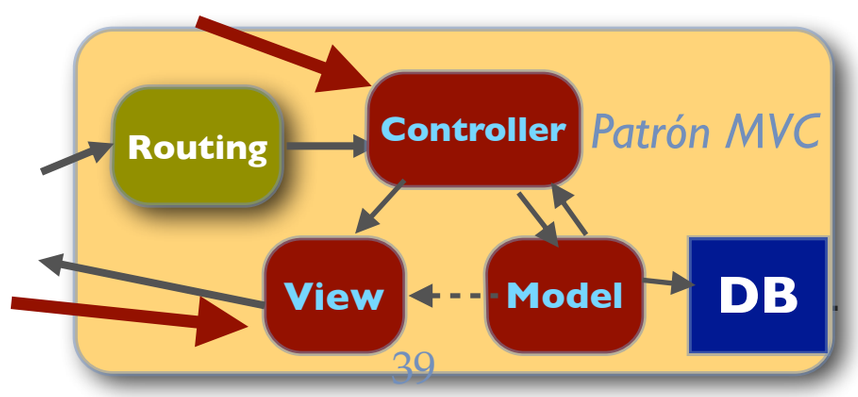
Paso 1d: views/quizes/question.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"><title>Quiz</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h2>Quiz: el juego de las preguntas</h2>

    <form method="get" action="/quizes/answer">
      Pregunta: <%= pregunta %> <p>
      <input type="text" name="respuesta" value="Responda aquí"/>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Paso 1d: views/quizes/answer.ejs

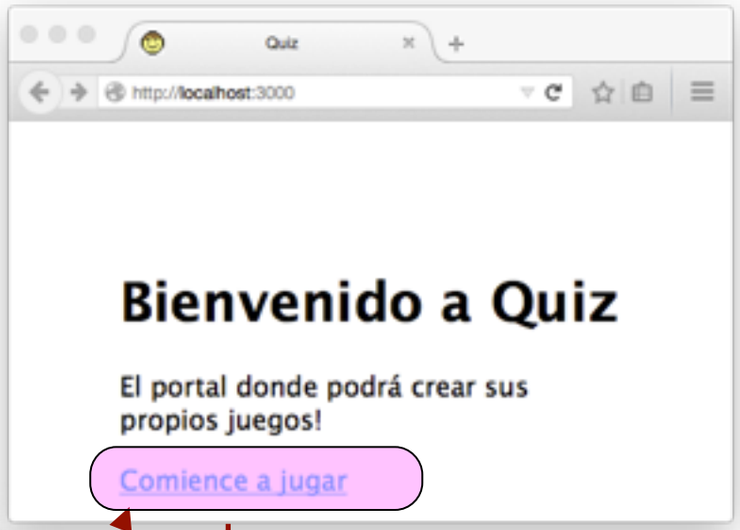
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Quiz</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
  <h2>Quiz: el juego de las preguntas</h2>

  <%= respuesta %><p>
  <a href="/quizes/question"> Intentelo otra vez </a>
  </body>
</html>
```





Paso 2: Modificación de la página de entrada

- Añadimos un enlace a la pregunta en la página de entrada
  - En la vista **views/index.ejs**
- Así tenemos un servicio hypermedia
  - Donde podemos navegar entre páginas
    - A través de los enlaces



index.ejs UNREGISTERED

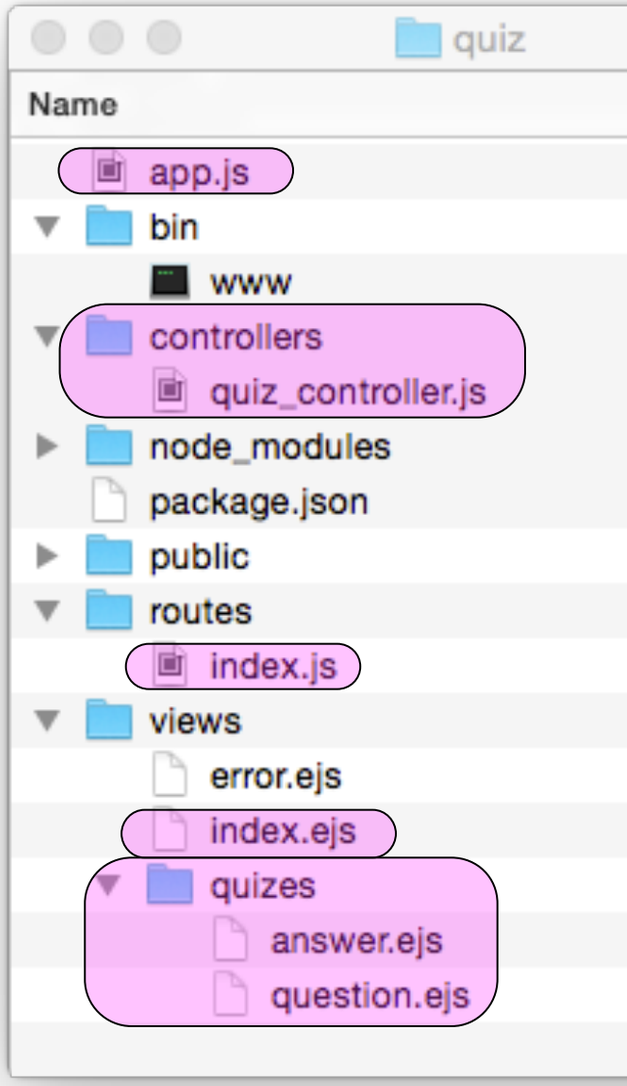
```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1>Bienvenido a <%= title %></h1>
    <p>El portal donde podrá crear sus propios juegos!</p>
    <p><a href="/quizes/question">Comience a jugar</a></p>
  </body>
</html>
```

GET /quizes/question

Guardar version y corregirla

- ✦Los cambios se pueden guardar con la opción **-a** de **git-commit**
  - Opción -a equivale realizar antes "**git add .**": **git commit -a -m "identificación de versión"**
- ✦Si debemos corregir un error después de cerrar la versión
  - La actualizamos con la opción **"--amend"** : **git commit --amend**

```
..$ git commit -a -m "Primera pregunta"   // opción -a equivale a "git add ."

..$ git log --oneline   // Hay tres versiones generadas
9600670 Primera pregunta
984f3ee  Primera página y favicon
9971c09 esqueleto express-generator

..$  …..   // si descubrimos un error, lo corregimos y cuando hemos
           // que el error esta corregido y la versión es la correcta

..$ git commit --amend  -m "Primera pregunta"  // Actualizar versión con las
                                               // correcciones introducidas.

..$ git log --oneline   // Siguen las mismas versiones
9f1ab23  Primera pregunta
984f3ee  Primera página y favicon
9971c09 esqueleto express-generator

..$
```



Final del tema

Quiz 4: Marco de la aplicación

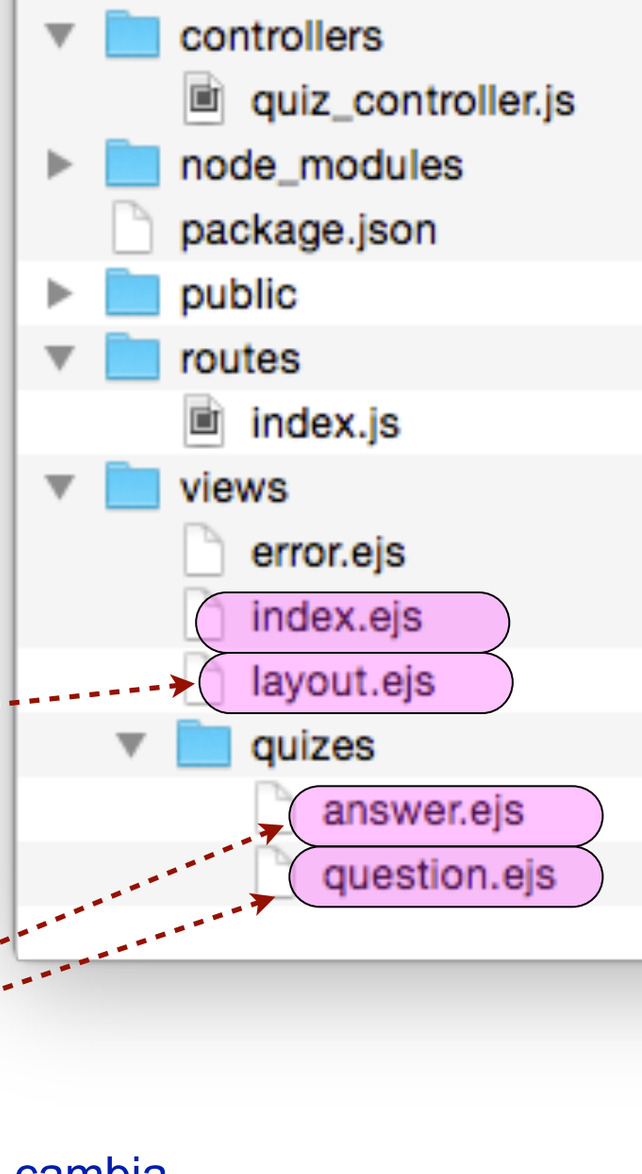
Juan Quemada, DIT - UPM

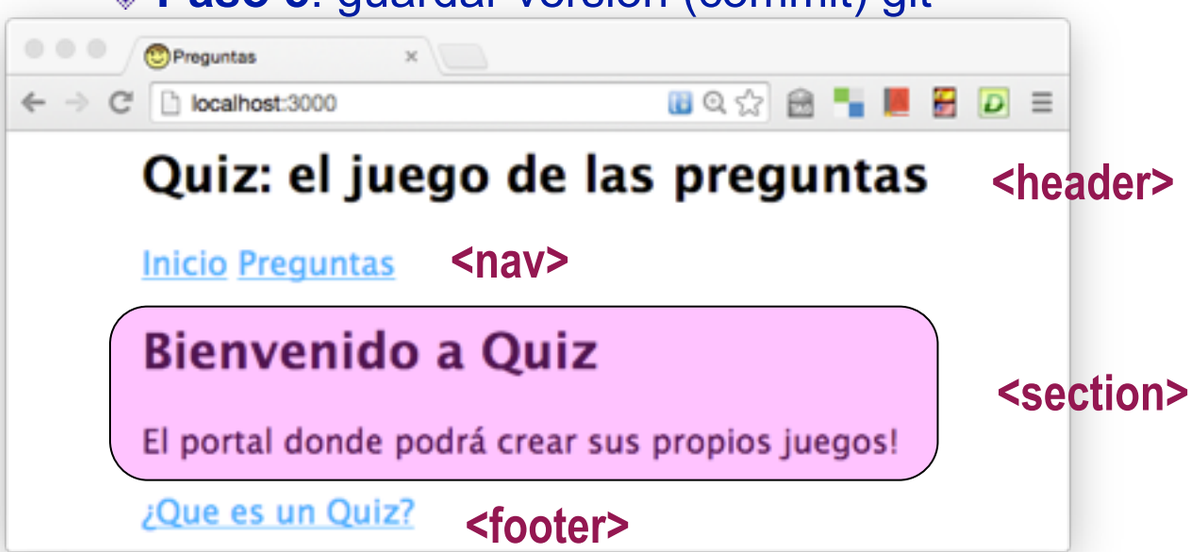
©

Quiz 4: Marco de la aplicación

Objetivo: Introducir un marco (layout.ejs) común a todas las vistas para no tener replicar el mismo código HTML en todas las vistas utilizando el middleware **express-partials** que soporta vistas parciales.

- **Paso 1**: Instalar el **paquete express-partials**
  - Añade vistas parciales y permite incluir un marco (layout) único
- **Paso 2**: Importar e instalar **express-partials** en **app.js**
- **Paso 3:** Añadir el marco único **(views/layout.ejs)** con las marcas HTML5
  - Cabecera: **<header>**
  - Navegación y enlaces: **<nav>**
  - Vistas que cambian: **<section>**
  - Pie de página: **<footer>**
- **Paso 4**: Se rehacen las vistas para renderizar en **<section>** solo el código que cambia
- **Paso 5**: guardar version (commit) git





Paso 1: express-partials

- **express-partials** es un middleware
  - Lo instalamos con la opción **--save**
    - Así se guarda la dependencia en **package.json**
      - **Sin opción —save**: no se guarda la dependencia
- **package.json**
  - Debe incluir todas las dependencias los módulos usados
    - Así es posible reinstalar Quiz con: **"npm install"**
  - Quiz está probado con: **express-partials@0.3.0**
- Una vez instalado tenemos que modificar
  - **app.js** para incluir el middleware
  - **las vistas** para incluir el marco (layout)

```
// instala express-partials en directorio app/node_modules
// opción --save guarda dependencia en package.json

// instala versión del proyecto en GITHUB y transparrencias
..$ npm install --save express-partials@0.3.0

// instala última versión estable en www.npmjs.org
..$ npm install --save express-partials
```

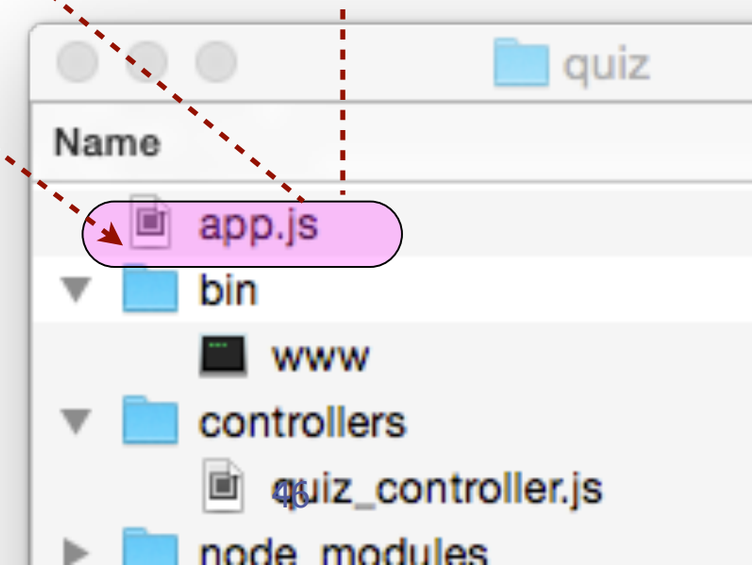
Paso 2: Instalar middleware

app.js UNREGISTERED

```
........
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(partials());
.........
```

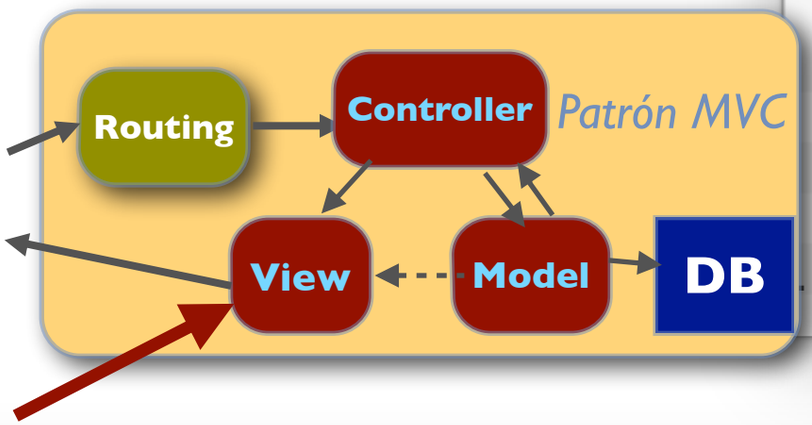
app.js

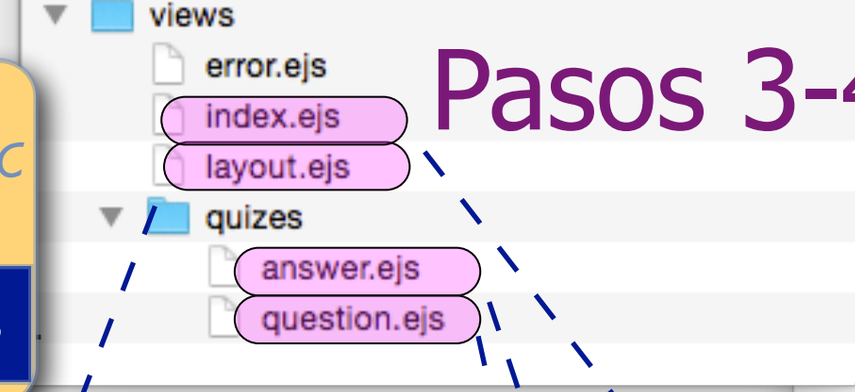
- El módulo **express-partials** importa una factoría que debe invocarse con () para generar el MW a instalar
  - Una vez instalado en el proyecto (transp. anterior)
    - Hay que **importarlo e instalarlo en app.js**

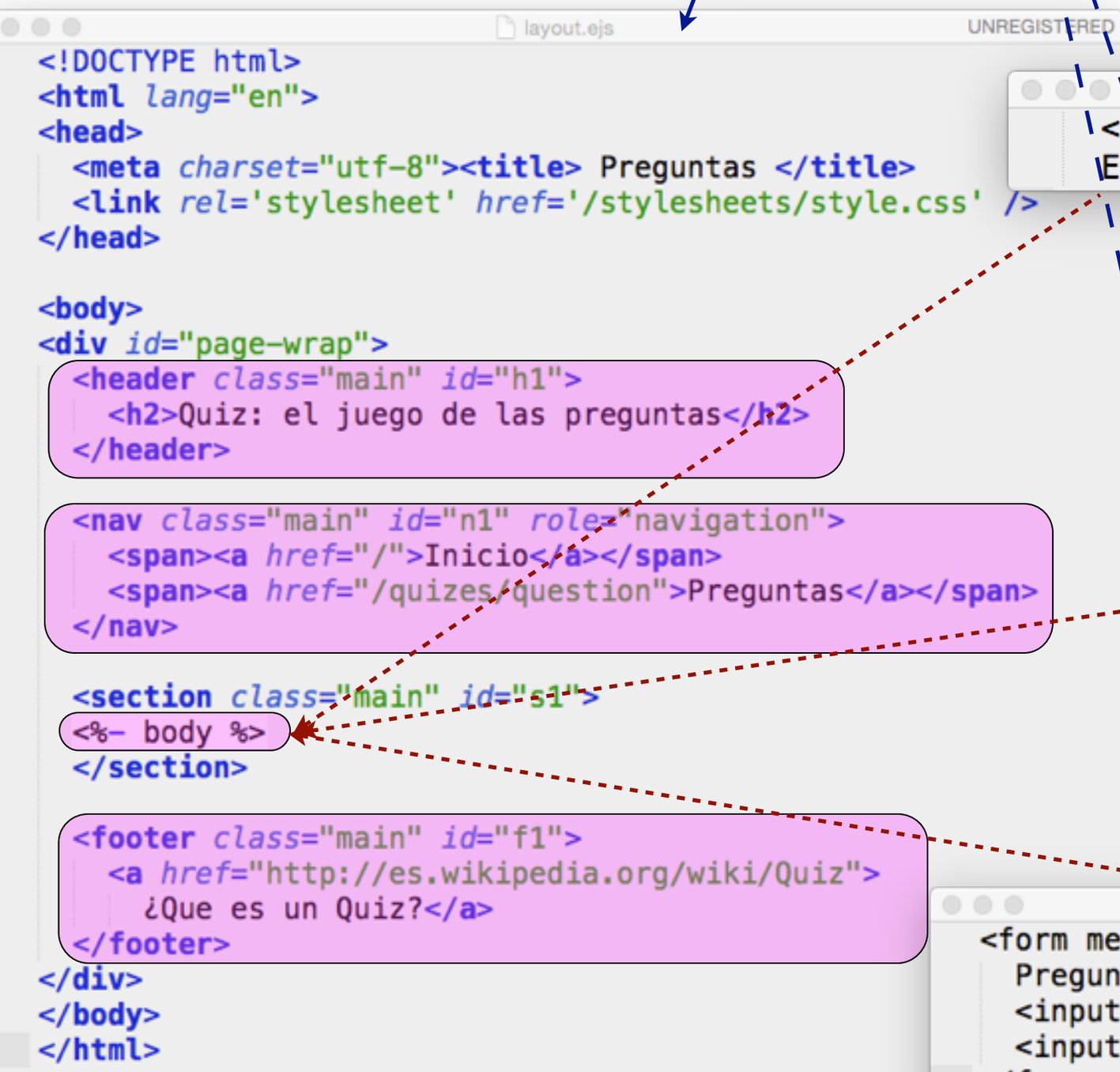


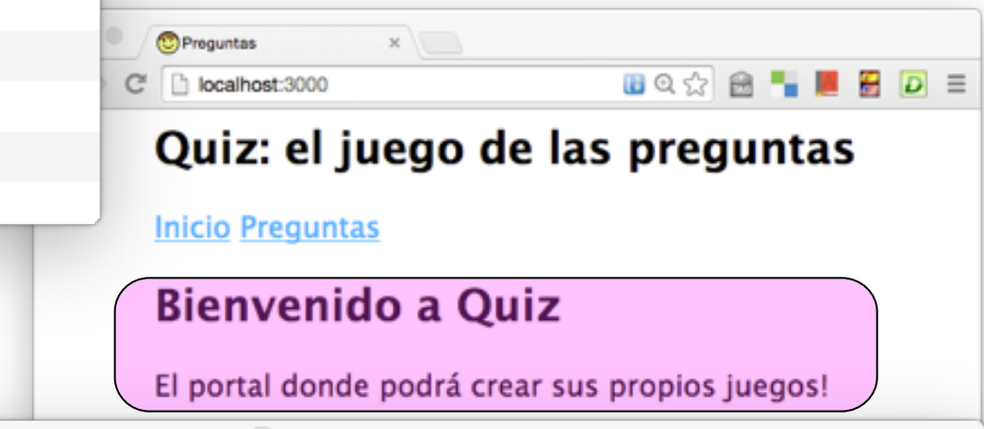
©

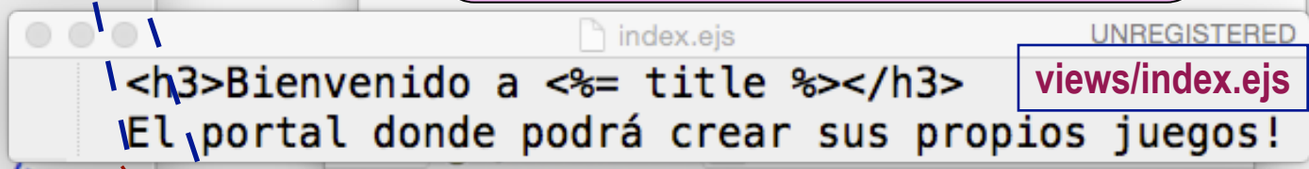
Pasos 3-4: Modificar vistas

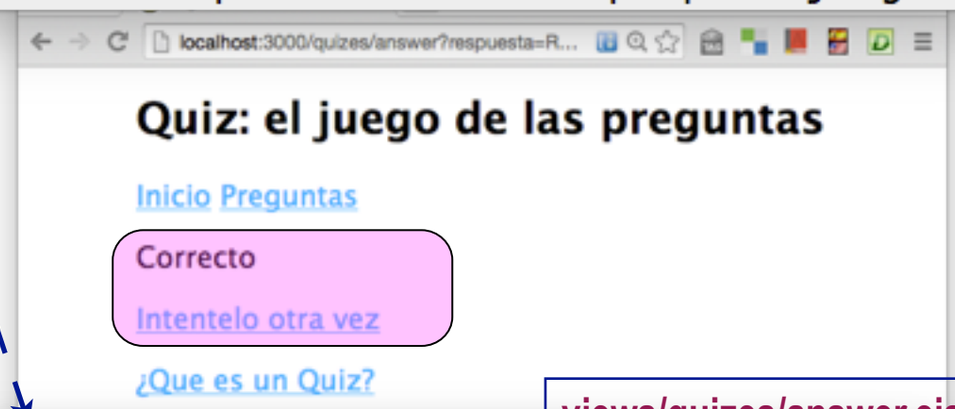


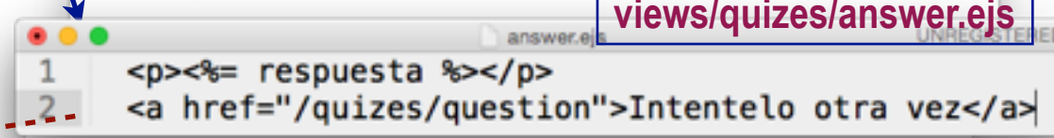


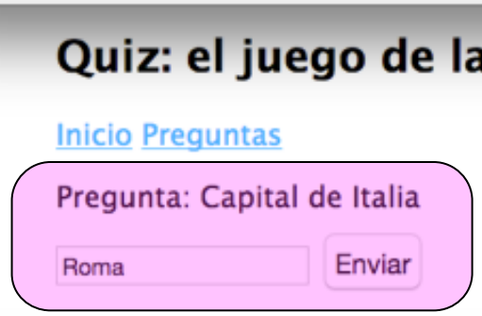


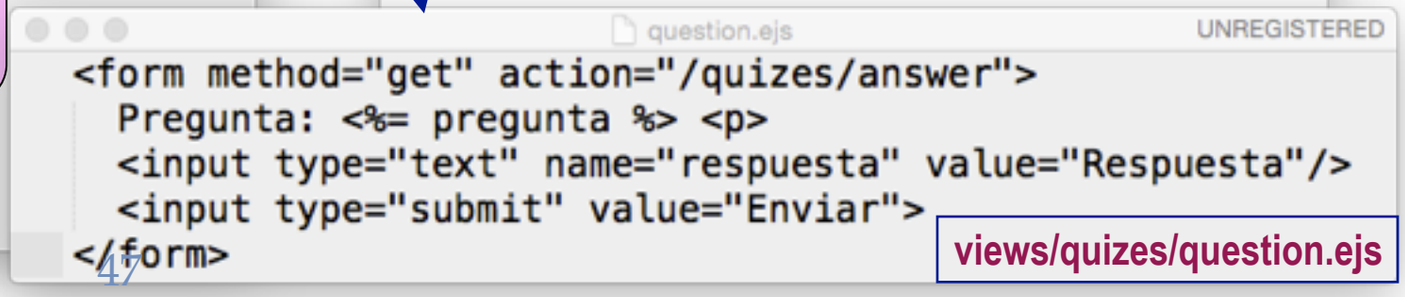












Final del tema

©

Quiz 5: CSS adaptable a móviles

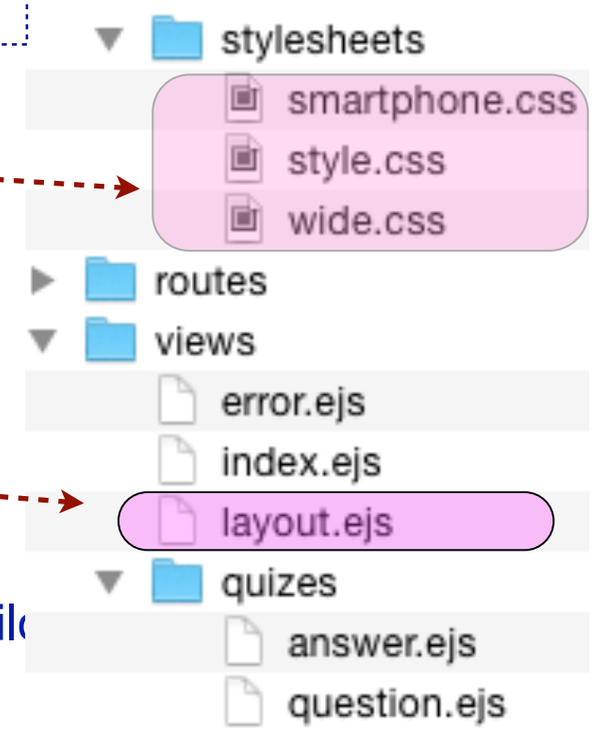
Juan Quemada, DIT - UPM

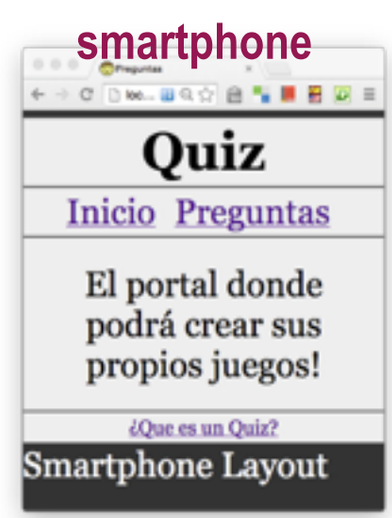
©

Quiz 5: CSS adaptable a móviles

Objetivo: Introducir estilos CSS que adapten la visualización de las páginas de Quiz a PCs y móviles utilizando un estilo adaptativo y **media-queries**.

- **Paso 1**: Se definen los 3 estilos CSS a utilizar
  - Estilo básico adaptable con facilidad
  - Reglas CSS adicionales para pantalla ancha
  - Reglas CSS adicionales para smartphone
- **Paso 2:** Se modifica el marco (**views/layout.ejs**)
  - para que utilice cada estilo en la pantalla asociada
    - Se utilizan **media-queries** para determinar cuando aplicar cada estilo
- **Paso 3**: guardar version (commit) git





Diseño Web adaptable

- El diseño adaptable (**responsive**) busca uniformizar el acceso
  - desde cualquier dispositivo, ya sea PC, móvil, tableta, smartTV, ...
    - optimizando la experiencia de usuario en cada uno de ellos
- El diseño adaptable utiliza una única definición HTML de interfaz o página
  - que se adapta al dispositivo con diferentes ficheros CSS
- La adaptación al tamaño de pantalla
  - Se puede realizar con **media queries** o con **JavaScript/DOM**
- Algunas referencias
  - Tutorial: http://www.alistapart.com/articles/responsive-web-design/
  - Ejemplos: http://css-tricks.com/resolution-specific-stylesheets/

CSS adaptable

- Vamos a utilizar una unica página HTML
  - con un estilo CSS básico
    - al que se añadirá con media queries
      - estilos CSS adicionales
        - para adaptar a PC y teléfono

- Media query para página pequeña (smatphone)
  - **media="only screen and (max-width: 480)"**

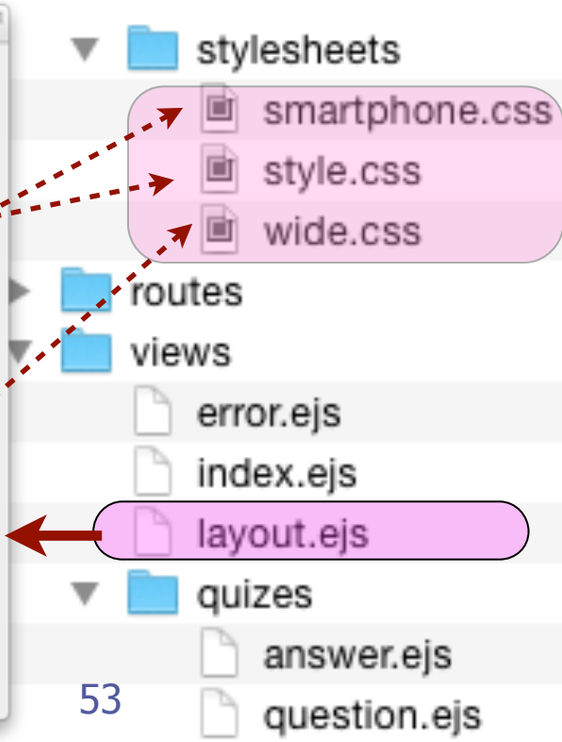
- Media query para pantalla ancha (PC)
  - **media= "only screen and (min-width: 530) and (min-device-width: 481)"**

El atributo **media** de <link ..>

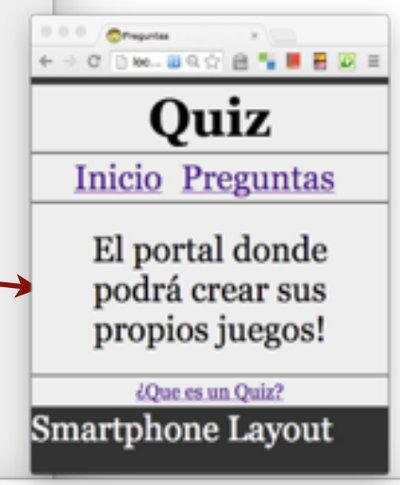
- El **atributo media** de la **marca <link>** de CSS3 detecta el dispositivo utilizado
  - http://www.w3schools.com/tags/att_link_media.asp
- El atributo **media** condiciona la carga del fichero de estilos al dispositivo utilizado
  - **Smartphone** con pantalla menor de 480 px
    - **media="only screen and (max-device-width: 480)"**
  - **Pantalla ancha** con el navegador en una ventana de anchura mayor a 600 px
    - **media= "only screen and (max-width: 600) and (min-device-width: 481)"**
- Añadimos al fichero **layout.ejs** estilos con **media queries** de carga condicional

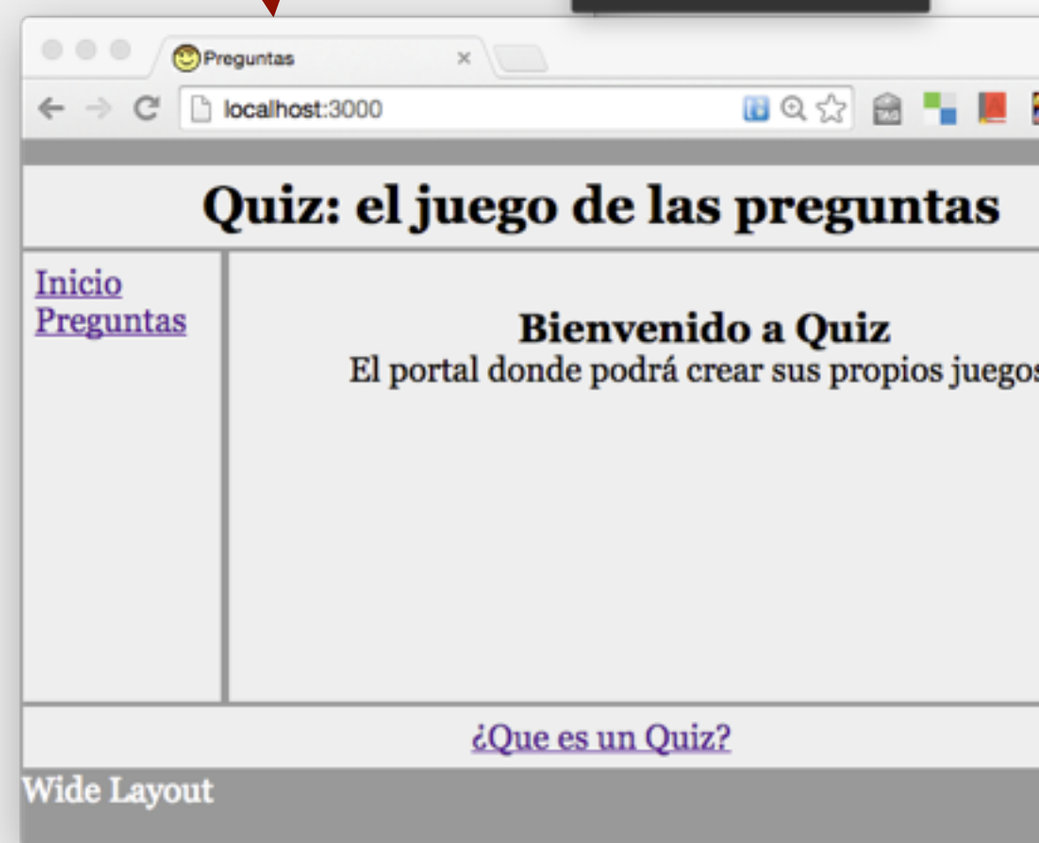
layout.ejs

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"><title> Quiz </title>
  <link rel='stylesheet' type='text/css' href='/stylesheets/style.css' />
  <link rel='stylesheet'
        type='text/css'
        media='only screen and (min-width: 530px) and (min-device-width: 481px)'
        href='/stylesheets/wide.css' />
  <link rel='stylesheet'
        type='text/css'
        media='only screen and (max-width: 480px)'
        href='/stylesheets/smartphone.css' />
</head>
```



```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"><title> Preguntas </title>
  <link rel='stylesheet' href='/stylesheets/style.css' />
  <link rel='stylesheet'
        type='text/css'
        media='only screen and (min-width: 530px) and (min-device-width: 481px)'
        href='/stylesheets/wide.css' />
  <link rel='stylesheet'
        type='text/css'
        media='only screen and (max-width: 480px)'
        href='/stylesheets/smartphone.css' />

</head>

<body>
<div id="page-wrap">
  <header class="main" id="h1">
    <h2>Quiz<span>: el juego de las preguntas</span></h2>
  </header>

  <nav class="main" id="n1" role="navigation">
    <span><a href="/">Inicio</a></span>
    <span><a href="/quizes/question">Preguntas</a></span>
  </nav>

  <section class="main" id="s1">
    <div><%- body %></div>
  </section>

  <footer class="main" id="f1">
    <a href="http://es.wikipedia.org/wiki/Quiz">
      ¿Que es un Quiz?</a>
  </footer>
</div>
</body>
</html>
```





Paso 2: layout.ejs

©

Paso 1a: style.css

```html
.....
<body>
<div id="page-wrap">
  <header class="main" id="h1">
    <h2>Quiz<span>: el juego de las preguntas</span></h2>
  </header>

  <nav class="main" id="n1" role="navigation">
    <span><a href="/">Inicio</a></span>
    <span><a href="/quizes/question">Preguntas</a></span>
  </nav>

  <section class="main" id="s1">
    <div><%- body %></div>
  </section>

  <footer class="main" id="f1">
    <a href="http://es.wikipedia.org/wiki/Quiz">
      ¿Que es un Quiz?</a>
  </footer>
</div>
</body>
</html>
```

style.css UNREGISTERED

```css
*                          { margin: 0; padding: 0; }
body                       { font: 16px Georgia, serif; }

#page-wrap                 { margin: 2% auto; color: white; }
#page-wrap > .main         { border: 1px solid #999; padding: 1%;
                             margin-bottom: 1px; color: black;}

section > div              { padding: 20px; }
header, nav, section       { background: #eee; text-align: center;}
footer                     { background: #eee; text-align: center;}
nav span                   { margin: 0px 10px 0px 0px;}
```

Paso 1b: wide.css

wide.css UNREGISTERED

```
body { background: #999; }

#page-wrap:after { content: "Wide Layout"; }

nav        { float: left; width: 15%; height: 200px ;}
nav span   { margin: 0px 100px 0px 0px;}
section    { float: right; width: 80%; height: 200px ;}
footer     { clear: both;  display: block; }
```

style.css + wide.css

app.html UNREGISTERED

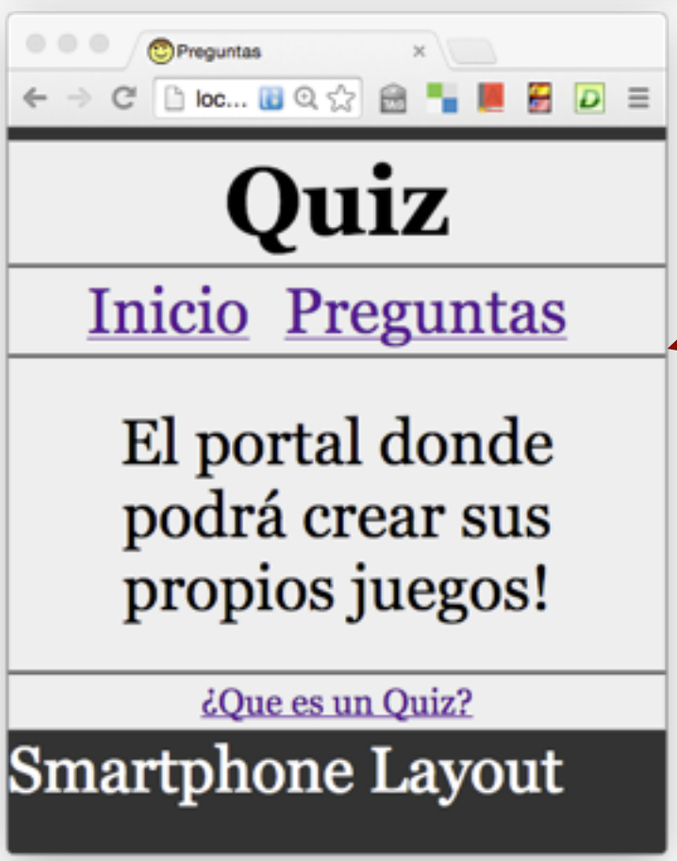
```
<link rel='stylesheet'
      media='only screen and (min-width: 530px) and (min-device-width: 481px)'
      href='/css/wide.css' />
```

Style.css + Smartphone.css

Paso 1c: smatphone.css

smartphone.css UNREGISTERED

```
body                    { background: #333; }

#page-wrap:after { content: "Smartphone Layout"; }

h2 span, section h3 { display: none; }
body                    { font-size: xx-large; }
footer, input           { font-size: large; }
```



app.html UNREGISTERED

```
<link rel='stylesheet'
      media='only screen and (max-width: 480px)'
      href='/css/smartphone.css' />
```

Final del tema

©

Ejercicio: Quiz con página de autores

Añadir a la versión Quiz 5 “CSS adaptable a móviles” una página adicional de autor que estará accesible a través de la primitiva del interfaz: “GET /author”.

La página se enlazara desde la barra de navegación con un enlace adicional, de nombre autor.

La página incluirá el nombre de los 2 autores (miembros de la pareja), una foto y un pequeño video de presentación del portal.

Recomendación: Hacer una rama en un clon del proyecto Quiz desde Quiz 5, que añada dicha funcionalidad.

Quiz 6: Despliegue en Heroku

Juan Quemada, DIT - UPM

©

Quiz 6: Despliegue en Heroku

Objetivo: desplegar Quiz en la nube para hacerla accesible a otros usuarios que se quieran conectar con un navegador desde cualquier punto de Internet.

Quiz es una aplicación de servidor que puede desplegarse en un servidor público. La vamos a desplegar en Heroku, que suministra un servicio PaaS (Platform as a Service) para node.js, con un despliegue muy sencillo y gratuito para aplicaciones con prestaciones limitadas.

Un servicio PaaS pone a nuestra disposición servidores virtuales con soporte para desplegar remotamente una aplicación de servidor en node.js.

Tutorial de despliegue con node.js:
https://devcenter.heroku.com/articles/getting-started-with-nodejs

- **Paso 1**: Crear una cuenta en Heroku: http://www.heroku.com
- **Paso 2**: Instalar Heroku Toolbelt
- **Paso 3**: Crear una aplicación en Heroku donde desplegar Quiz
- **Paso 4**: Adaptar la aplicación Quiz para despliegue en Heroku
- **Paso 5**: Versión git y despliegue de Quiz en Heroku

Cuenta en Heroku y Heroku Toolbelt

- Heroku: servicio en la nube de tipo PaaS
  - Publica directamente aplicaciones en node.js, PHP, RoR, Python, ..
- Paso 1: debemos crear una cuenta en Heroku para publicar Quiz en la nube
  - https://www.heroku.com/
- Paso 2: instalar Heroku Toolbelt, las herramientas de gestión de Heroku
  - https://toolbelt.heroku.com
- Heroku Toolbelt instala en nuestro ordenador 2 comandos
  - **heroku** gestiona el despliegue de aplicaciones desde nuestro PC
  - **foreman** ejecuta localmente aplicaciones adaptadas a Heroku
  - Además necesita **git** para desplegar y si no está instalado debe instalarse

Comados heroku y foreman

quiz — bash — 76×15

```
venus-5:quiz jq$
venus-5:quiz jq$ foreman --help
Commands:
  foreman check                    # Validate your application's Procfile
  foreman export FORMAT LOCATION   # Export the application to another ...
  foreman help [COMMAND]           # Describe available commands or one...
  foreman run COMMAND [ARGS...]    # Run a command using your applicati...
  foreman start [PROCESS]          # Start the application (or a specif...
  foreman version                  # Display Foreman gem version

Options:
  -f, [--procfile=PROCFILE]  # Default: Procfile
  -d, [--root=ROOT]          # Default: Procfile directory

venus-5:quiz jq$
```

quiz — bash — 73×44

```
venus-5:quiz jq$
venus-5:quiz jq$ heroku --help
Your version of git is 1.9.3. Which has serious security vulnerabilities.
More information here: https://blog.heroku.com/archives/2014/12/23/update
_your_git_clients_on_windows_and_os_x
Usage: heroku COMMAND [--app APP] [command-specific-options]

Primary help topics, type "heroku help TOPIC" for more details:

  addons    #  manage addon resources
  apps      #  manage apps (create, destroy)
  auth      #  authentication (login, logout)
  config    #  manage app config vars
  domains   #  manage custom domains
  logs      #  display logs for an app
  ps        #  manage dynos (dynos, workers)
  releases  #  manage app releases
  run       #  run one-off commands (console, rake)
  sharing   #  manage collaborators on an app

Additional topics:

  certs        #  manage ssl endpoints for an app
  drains       #  display drains for an app
  features     #  manage optional features
  fork         #  clone an existing app
  git          #  manage git for apps
  help         #  list commands and display help
  keys         #  manage authentication keys
  labs         #  manage optional features
  maintenance  #  manage maintenance mode for an app
  members      #  manage membership in organization accounts
  orgs         #  manage organization accounts
  pg           #  manage heroku-postgresql databases
  pgbackups    #  manage backups of heroku postgresql databases
  plugins      #  manage plugins to the heroku gem
  regions      #  list available regions
  stack        #  manage the stack for an app
  status       #  check status of heroku platform
  twofactor    #
  update       #  update the heroku client
  version      #  display version

venus-5:quiz jq$
```

©

Aplicaciones Heroku

- Heroku es un **servicio de tipo PaaS** (Platform as a Service)
  - Permite instalar aplicaciones en node.js (y otros) que se ejecutan en la nube
    - El servicio es gratuito para aplicaciones con recursos limitados
      - Una sola maquina virtual (1 Dynos), 300 MBytes HD, ….
- **Aplicaciones**: se ejecutan en una o más máquinas virtuales, denominadas Dynos
  - **Dynos**: programa equivalente a una máquina servidora (ordenador físico)
    - Un Dynos se ejecuta en un contenedor Linux ligero y aislado de otros
- Las aplicaciones en Heroku se gestionan remotamente de 2 formas equivalentes
  - Con el comando "**heroku**" de Heroku Toolbelt
  - Accediendo al **Dashboard** de Heroku con un navegador
- Cada aplicación Heroku tiene un **repositorio remoto git asociado**
  - Para desplegar en la nube debemos enviar el código a dicho repositorio

Paso 3: Crear aplicación Heroku con Toolbelt

// Después de crear una cuenta en Heroku e instalar Heroku Toolbelt en
// nuestro ordenador local podemos crear y desplegar aplicaciones en Heroku

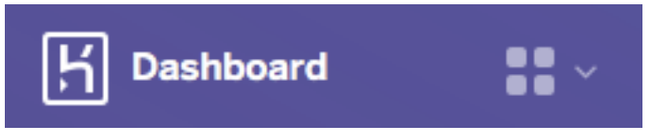
..$ heroku login // Nos conectamos a nuestra cuenta en Heroku con las mismas credenciales
Enter your Heroku credentials.
Email: jquemada@dit.upm.es
Password (typing will be hidden):
Authentication successful.

// Como la conexión debe ser segura, si no tenemos una clave pública creada, nos puede
// preguntar además si la crea y la sube a Heroku (para que la comunicación sea segura)
// -> **se debe contestar que si**

..$ heroku create // crea una aplicación en heroku con un nombre aleatorio: **pure-reaches-3917**
Creating pure-reaches-3917... done, stack is cedar-14
https://pure-reaches-3917.herokuapp.com/ | https://git.heroku.com/pure-reaches-3917.git
Git remote heroku added!

// Cambiamos el nombre/URL de la aplicación creada a uno propio nuestro
// el nombre/URL pasará a ser: **https://quiz-2015-pepe.herokuapp.com/**
// en vez del generado automaticamente: **https://pure-reaches-3917.herokuapp.com/**
..$ heroku apps:rename quiz-2015

..$ heroku open --app quiz-2015 // abre el navegador en el URL https://quiz-2015.herokuapp.com/



Heroku Dashboard

- El Dashboard es un interfaz Web interactivo de gestión de aplicaciones
  - Las aplicaciones también se pueden crear o modificar desde el Dashboard
    - Permite hacer muchas operaciones de Heroku interactivamente
      - Es más sencillo y claro de manejar para principiantes

https://postgres.heroku.com

https://dashboard.heroku.com

Dashboard

jquemada@dit.upm.es

FAVORITES

Favorite any app to pin it here in the sidebar

Personal Apps

Apps

quiz-2015

Production Check

Resources | Code | Metrics | Activity | Access | Settings

Name

quiz-2015

nombre de la aplicación

Rename

Config Variables

Reveal Config Vars

Variables deconfiguración

Config vars change the way your app behaves. In addition to creating your own, some addons come with their own.

Info

Region	us
Stack	cedar-14
Framework	Node.js
Git URL	git@heroku.com:quiz-2015.git
Repo size	18.9KB
Slug size	10.1MB of 300MB

Datos de la máquina virtual desplegada y del código instalado

repositorio git remoto

Domains

Add your custom domains here then point your DNS to Heroku.

Domain Names

quiz-2015.herokuapp.com

URL de la aplicación desplegada

Dashboard de Heroku

Paso 4: Adaptar Quiz a Heroku

- El despliegue de Quiz en Heroku necesita
  - Crear el fichero Procfile y reestructurar package.json

Procfile UNREGISTERED

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

quiz

Name

- app.js
- bin
- controllers
  - quiz_controller.js
- models
- node_modules
- package.json
- Procfile
- public
- routes
- views

- **Procfile:** fichero de arranque en Heroku en directorio raíz
  - con el contenido "**web: node ./bin/www**"
    - Aplicación arranca ahora con "**foreman start**"
      - Creando un entornosimilar al de heroku

- **package.json**
  - debe incluir un apartado "**engines**" que declara
    - la versión de engines a instalar en Heroku
      - Incluir versión de "**npm**" y "**node**" en "**engines**"

package.json UNREGISTERED

```
{
  "name": "quiz",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "engines": {
    "node": "0.10.x",
    "npm": "1.4.x"
  },
  "dependencies": {
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "debug": "~2.0.0",
    "ejs": "~0.8.5",
    "express": "~4.9.0",
    "express-partials": "^0.3.0",
    "morgan": "~1.3.0",
    "serve-favicon": "~2.1.3",
  }
}
```

Paso 5: versión y despliegue en Heroku

- Para desplegar en Heroku debemos subir al repositorio git de Heroku
  - El código de la aplicación a desplegar con "**git push heroku master**"
    - Para desplegar consolidamos la versión, normalmente en la rama master
- "**heroku create**" configura el repositorio remoto git como **heroku** en nuestro PC
  - El repositorio remoto **heroku** se puede configurar también con el comando
    - "**git remote add heroku URL**" -> URL dado en Dashboard
  - En el dashboard podemos ver también los datos del repositorio remoto git

```
..$ cd quiz        // entramos en directorio del proyecto quiz

..$ git add .      // añadimos los cambios al índice

                   // generamos versión "Despliegue en Heroku" en la rama master
..$ git commit -m "Despliegue en Heroku"

                   // Subir la última versión de la rama master a Heroku
..$ git push heroku master        // !!!Quiz ya esta desplegado!!!

..$ heroku open  // Lanzar el navegador conectado a la aplicación
```

Final del tema

©

Ejercicio: Desplegar Quiz con página de autores en Heroku

Añadir a la versión Quiz 6 “Despliegue en Heroku” una página adicional de autor que estará accesible a través de la primitiva del interfaz: “GET /author”.

La página se enlazara desde la barra de navegación con un enlace adicional, de nombre autor.

La página incluirá el nombre de los 2 autores (miembros de la pareja), una foto y un pequeño video de presentación del portal.

Desplegar dicha versión en Heroku y entregar el enlace a dicha versión por Moodle.

Recomendación: Hacer una rama en un clon del proyecto Quiz desde Quiz 6, que añada dicha funcionalidad y desplegar dicha rama en Heroku.

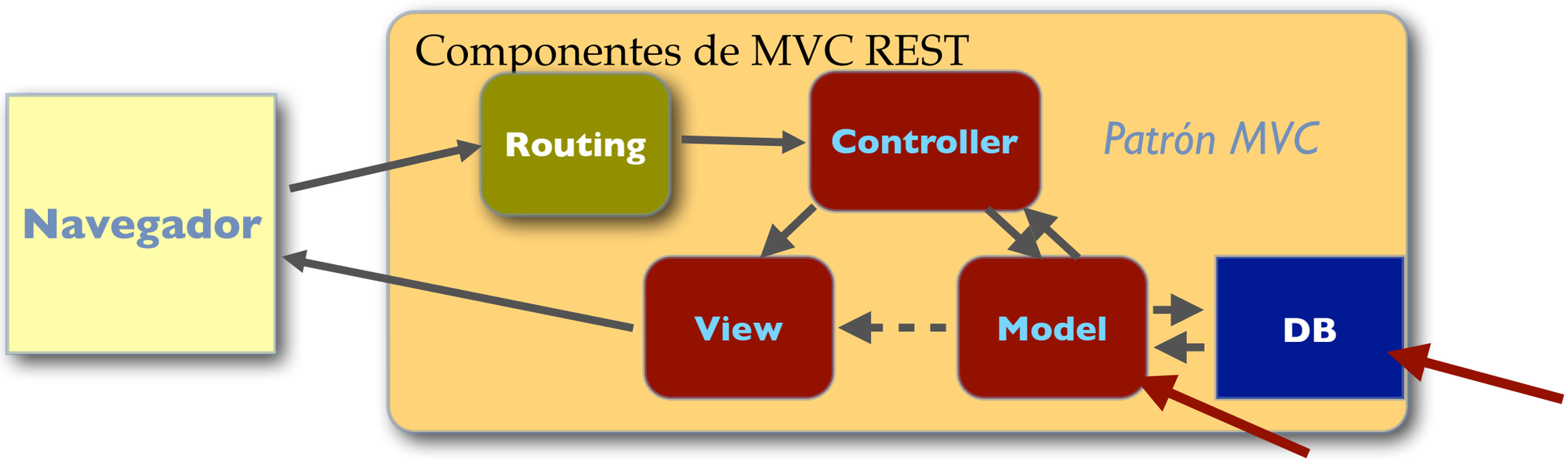
Quiz 7a: La base de datos

Juan Quemada, DIT - UPM

Quiz 7: Introducción de una base de datos

- En Quiz 7 introducimos el modelo y la base de datos (DB)
  - Quiz funciona exactamente igual, pero la pregunta se guarda en la DB

- Esto es habitual en una aplicación MVC de servidor
  - Para poder guardar y gestionar muchas preguntas



Bases de Datos

id	pregunta	respuesta
1	¿Cual es la Capital de Italia?	Roma
2	¿Quien descubrió América?	Cristobal Colón
3	¿Capital de Portugal?	Lisboa
4	.....	.....
5	.....	.....

- **Base de datos** (**DB** - Data Base): gestiona grandes volúmenes de datos eficazmente
  - Los **grandes portales Web** guardan sus datos en **bases de datos**
- **Bases de datos relacionales** (**RDBMS** - Relational Data Base Manag. System)
  - Suelen guardar los **bloques de datos** como filas de una **tabla** (ver figura)
    - **Relaciones entre tablas** crean estructuras de datos complejas
  - **RDBMSs** mas habituales: **SQLite**, **MySQL**, **Postgres**, **MariaDB**, **Oracle**, …
- Las bases de datos tipo **NoSQL** son cada vez más habituales
  - Hay bases de datos **clave-valor**, de **grafos**, de **documentos**, ..
    - Son más escalables y gestionan mejor estructuras de datos específicas

Tablas de una Base de Datos

- La información de una base de datos de tipo RDBMS se estructura en tablas
  - Cada fila de una tabla es un registro de datos del mismo tipo
    - Por ejemplo, preguntas y respuestas, datos de usuario, artículos de una tienda, ..
- Cada registro (fila) tiene además de los datos que guarda varios elementos
  - **Clave primaria**: identifica unívocamente un registro de la tabla
  - Puede tener un campo con la fecha de creación: **createdAt** en sequelize.js
  - Puede tener un campo con la fecha de última modificación:**updatedAt** en sequelize.js
- El acceso a la información de una tabla se realiza con **SQL** (Structured Query Language)

id	pregunta	respuesta	createdAt	updatedAt
1	¿Capital de Italia?	Roma	2011-03-21 10:44:06	2011-03-26 22:34:01
2	¿Quien descubrió América?	Cristobal Colón	2012-03-21 10:44:06	2012-04-26 00:34:01
3	¿Capital de Portugal?	Lisboa	2011-11-22 10:11:33	2012-03-29 22:00:00
4	.....	.....	.....	.....
5	.....	.....	.....	.....

Relaciones entre tablas

id	autor	pregunta	respuesta	createdAt	updatedAt
1	1	¿Capital de Italia?	Roma	2011-03-21 10:44:06	2011-03-26 22:34:01
2	2	¿Quien descubrió América?	Cristobal Colón	2012-03-21 10:44:06	2012-04-26 00:34:01
3	4	¿Capital de Portugal?	Lisboa	2011-11-22 10:11:33	2012-03-29 22:00:00
4	.....	.....	.....	.....	.....
5	.....	.....	.....	.....	.....

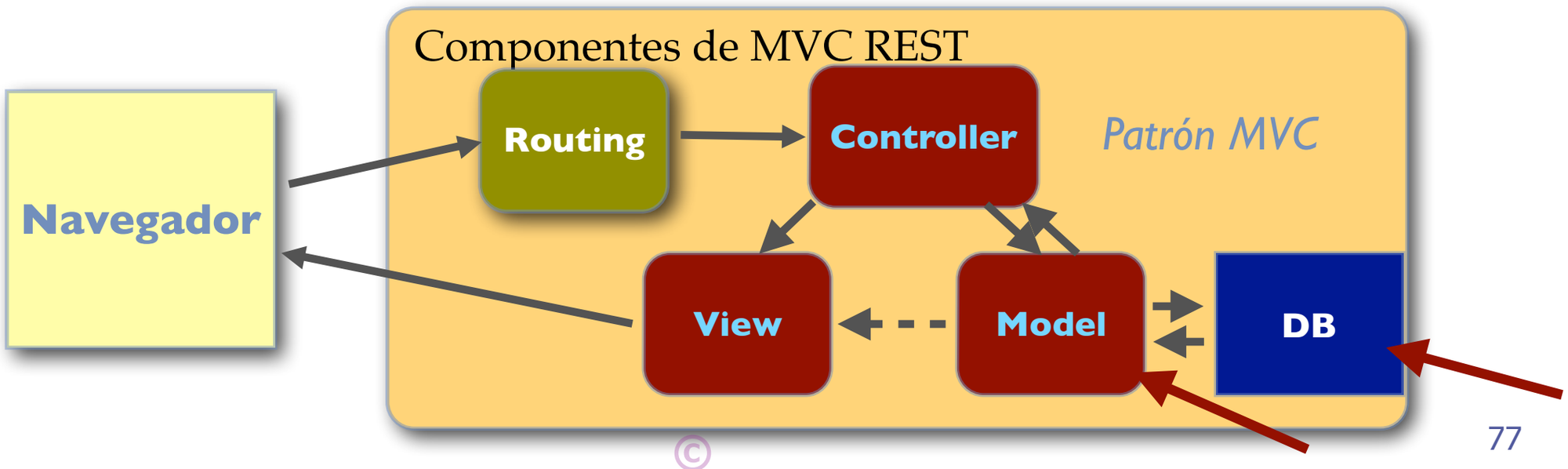
id	name	email	....	....	....	....	....	....	....
1	Juan López	jlopez@gmail.com	....	....	....	....	....	....	....
2	Javier Tello	javiertello@upm.es	....	....	....	....	....	....	....
3	.....	.....	....	....	....	....	....	....	....
4	Gabriel Tomé	gabit@dit.upm.es	....	....	....	....	....	....	....
5	.....	.....	....	....	....	....	....	....	....
	.....	.....	....	....	....	....	....	....	....

- Una RDBMS permite estructurar la información en varias tablas
  - Una tabla con las preguntas, otra con los datos del autor, otra con comentarios, etc
- P. e., una **relación** puede identificar los datos del autor de una pregunta en otra tabla
  - Para ello, el **campo autor** en la **tabla de preguntas**
    - contendrá la **clave externa** que identifica al autor en su tabla de usuarios

Modelo y DB

id	pregunta	respuesta
1	¿Cual es la Capital de Italia?	Roma
2	¿Quien descubrió América?	Cristobal Colón
3	¿Capital de Portugal?	Lisboa
4	.....	.....
5	.....	.....

- **Modelo**: es el responsable de mantener el estado de la aplicación (los recursos)
  - Suele comprobar las reglas (de negocio) que debe seguir la aplicación
  - Con **RDBMSs** se suele utilizar un **ORM - Object-Relational-Mapping**
    - Transforma objetos y métodos de acceso en operaciones SQL y viceversa
  - **sequelize.js** es un ORM sencillo para acceder a la base de datos (modelo)
    - http://sequelizejs.com/
- **Base de datos**: utilizamos **sqlite3** para desarrollo y **Postgres** para despliegue
  - Un ORM accede a una base de datos con SQL (Structured Query Language)



Final del tema

©

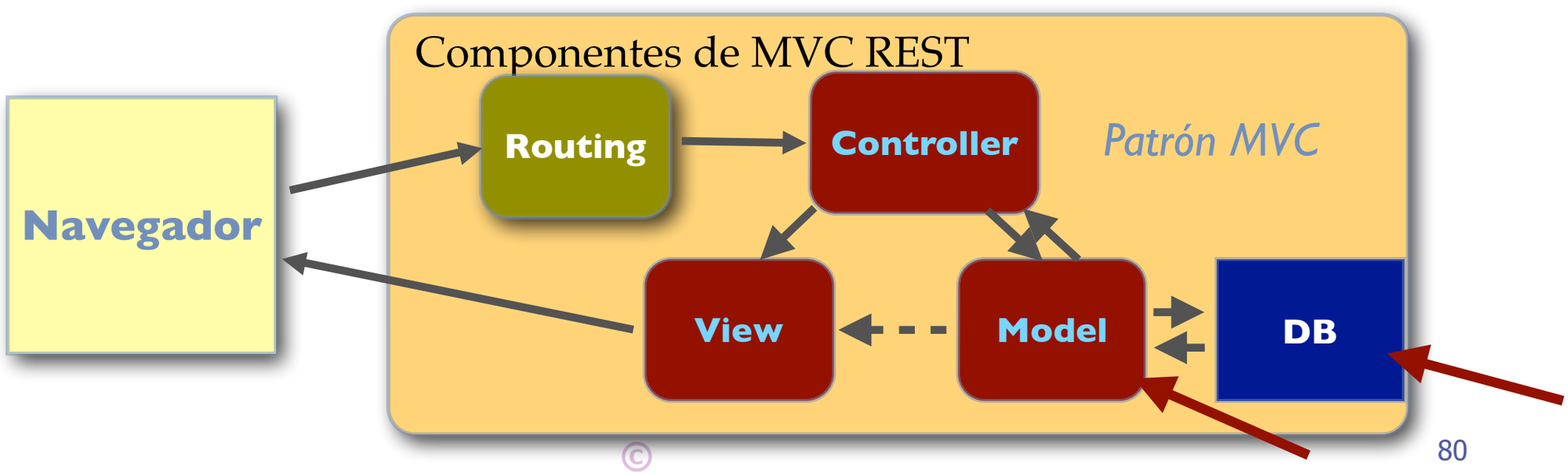
Quiz 7b: sequelize.js y SQLite

Juan Quemada, DIT - UPM

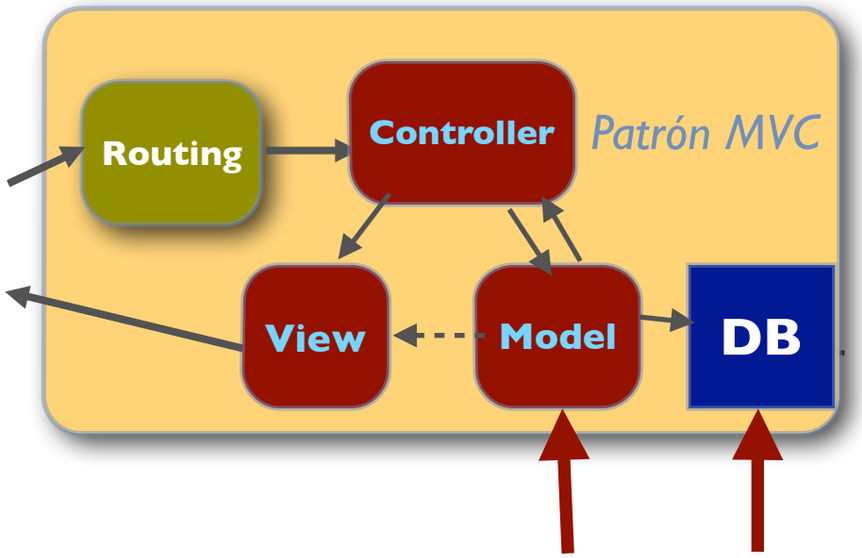
Quiz 7: sequelize.js y SQLite

Objetivo: Añadir la base de datos sqlite3 y sequelize completando el patrón MVC en el entorno de desarrollo local. La base de datos se inicializa con la pregunta/ respuesta "Capital de Italia: Roma". La DB podrá gestionar múltiples preguntas a partir de ahora.

- **Paso 1**: Instalar **sequelize** y **sqlite3**
- **Paso 2:** Crear el modelo en el directorio **models** con las definiciones necesarias
  - Fichero **models/quiz.js:** definición de **tabla Quiz** de preguntas y respuestas.
  - Fichero **models/models.js:** crea e inicializa la tabla Quiz utilizando sequelize
- **Paso 3:** Modificar **quiz_controller.js** para que busque la pregunta en la BD
- **Paso 4:** Actualizar **.gitignore** para que no guarde fichero **quiz.sqlite** con DB
- **Paso 5**: Guardar versión (commit) con git



Paso 1: Sequelize y SQLite3



- **sequelize:** es un ORM para express.js
  - Permite acceder a SQLite, MySQL, Postgres, …
    - http://sequelizejs.com/articles/getting-started
    - http://sequelize.readthedocs.org/en/latest/
- **sqlite3** es una base de datos relacional muy sencilla
  - Utilizada habitualmente en desarrollo (despliegue utilizará Postgres)
    - http://www.sqlite.org
- **Sequelize** accede a **sqlite3** utilizando **SQL** (Structured Query Language)
  - **SQL** es un lenguaje de acceso a bases de datos
    - http://es.wikipedia.org/wiki/SQL
- Ambos paquetes los instalamos con npm
  - Desde el servidor https://www.npmjs.com

// instalar **sequelize** y **sqlite3** en directorio **node_modules** con **npm**

..$ npm install --save sequelize

..$ npm install --save sqlite3

```
{
  "name": "application-name",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~3.4.8",
    "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",
    "sequelize": "~1.7.0",
    "sqlite3": "~2.2.0"
  }
}
```

Paso 2a: tabla Quiz

El fichero **quiz.sqlite** es el que contiene realmente la DB y sus datos. Este fichero se crea automáticamente al arrancar la aplicación con

$ npm start

Ojo! No crearlo con el editor. Recomendación: Borrarlo cuando la DB de problemas. Se creará al volver a hacer "npm start"

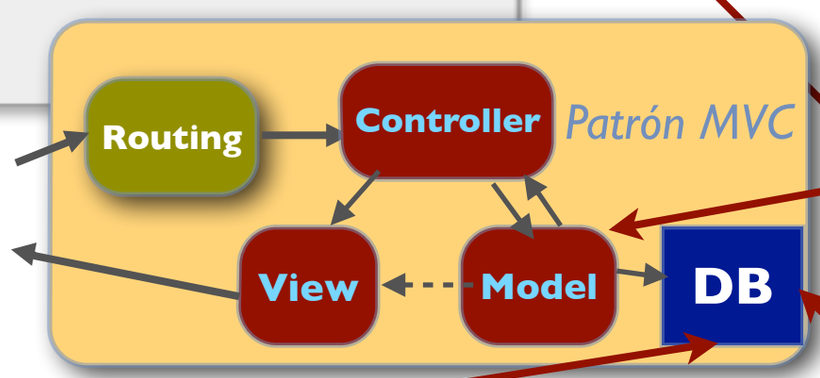
El fichero **quiz.js** es parte del modelo (**directorio models**) y define la estructura de la tabla de quizes (preguntas) con 2 campos (tipo string):

- pregunta: DataTypes.STRING
- respuesta: DataTypes.STRING

quiz.js

```
// Definicion del modelo de Quiz

module.exports = function(sequelize, DataTypes) {
  return sequelize.define('Quiz',
            { pregunta:  DataTypes.STRING,
              respuesta: DataTypes.STRING,
            });
}
```

id	pregunta	respuesta
1	¿Cual es la Capital de Italia?	Roma
2	¿Quien descubrió América?	Cristobal Colón
3	¿Capital de Portugal?	Lisboa
4	.....	.....
5	.....	.....



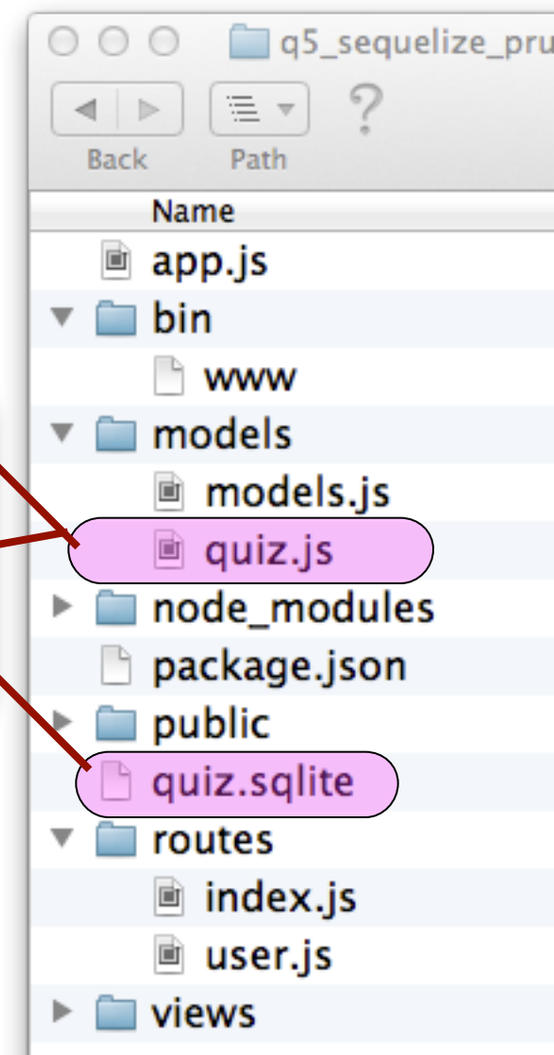


Tabla Quiz

© Juan Quemada - UPM-DIT

Tipos de datos soportados en sequelize.js

```
Sequelize.STRING                    // VARCHAR(255)
Sequelize.STRING(1234)              // VARCHAR(1234)
Sequelize.STRING.BINARY             // VARCHAR BINARY
Sequelize.TEXT                      // TEXT

Sequelize.INTEGER                   // INTEGER
Sequelize.BIGINT                    // BIGINT
Sequelize.BIGINT(11)                // BIGINT(11)
Sequelize.FLOAT                     // FLOAT
Sequelize.FLOAT(11)                 // FLOAT(11)
Sequelize.FLOAT(11, 12)             // FLOAT(11,12)

Sequelize.DECIMAL                   // DECIMAL
Sequelize.DECIMAL(10, 2)            // DECIMAL(10,2)

Sequelize.DATE                      // DATETIME for mysql / sqlite, TIMESTAMP WITH T
IME ZONE for postgres
Sequelize.BOOLEAN                   // TINYINT(1)

Sequelize.ENUM('value 1', 'value 2')  // An ENUM with allowed values 'value 1' and 'va
lue 2'
Sequelize.ARRAY(Sequelize.TEXT)     // Defines an array. PostgreSQL only.

Sequelize.BLOB                      // BLOB (bytea for PostgreSQL)
Sequelize.BLOB('tiny')              // TINYBLOB (bytea for PostgreSQL. Other options
 are medium and long)
Sequelize.UUID                      // UUID datatype for PostgreSQL and SQLite, CHAR
(36) BINARY for MySQL (use defaultValue: Sequelize.UUIDV1 or Sequelize.UUIDV4 to make
sequelize generate the ids automatically)
```

id	pregunta	respuesta
1	¿Cual es la Capital de Italia?	Roma
2	¿Quien descubrió América?	Cristobal Colón
3	¿Capital de Portugal?	Lisboa
4	.....	.....
5	.....	.....

Los campos de la tabla están tipados y deben contener alguna de los tipos aquí indicados.

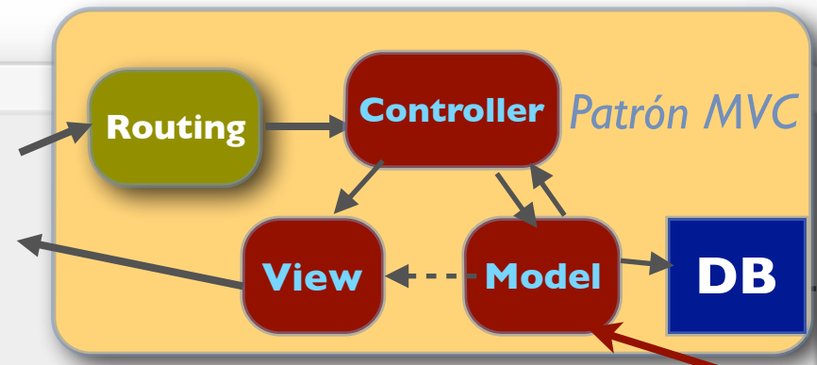
El tipo STRING almacena texto.

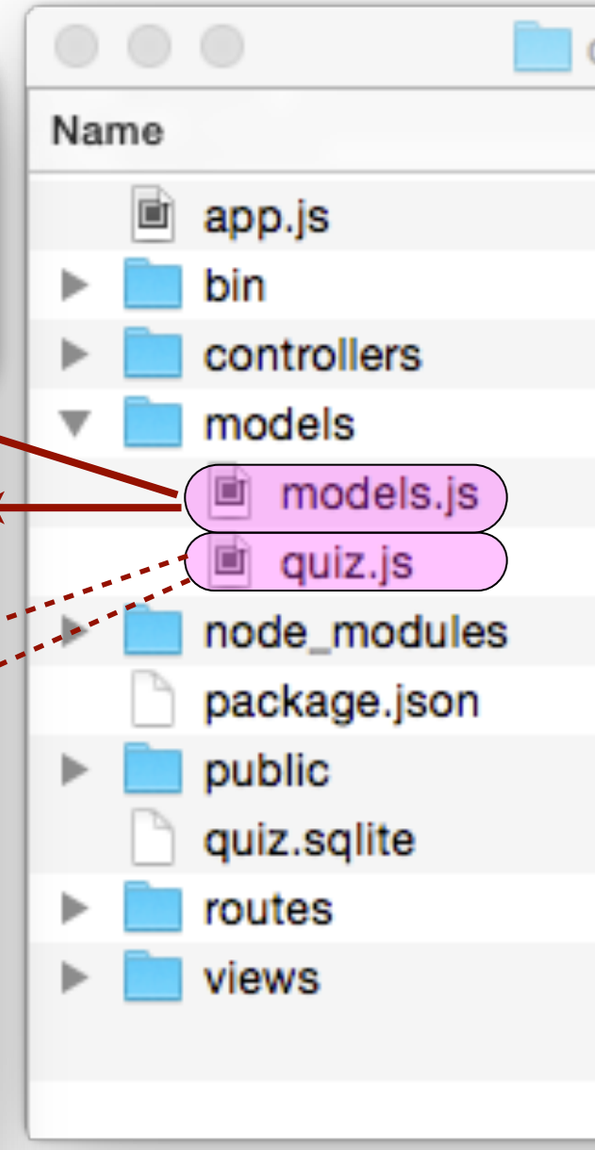
Paso 2b: models.js

models.js construye la DB y el modelo importando (**quiz.js**)
sequelize.sync() construye la DB según define el modelo.

models.js

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

// Cargar Modelo ORM
var Sequelize = require('sequelize');

// Usar BBDD SQLite:
var sequelize = new Sequelize(null, null, null,
                              {dialect: "sqlite", storage: "quiz.sqlite"}
                             );

// Importar la definicion de la tabla Quiz en quiz.js
var Quiz = sequelize.import(path.join(__dirname,'quiz'));

exports.Quiz = Quiz; // exportar definición de tabla Quiz

// sequelize.sync() crea e inicializa tabla de preguntas en DB
sequelize.sync().success(function() {
  // success(..) ejecuta el manejador una vez creada la tabla
  Quiz.count().success(function (count){
    if(count === 0) {   // la tabla se inicializa solo si está vacía
      Quiz.create({ pregunta: 'Capital de Italia',
                    respuesta: 'Roma'
                  })
      .success(function(){console.log('Base de datos inicializada')});
    };
  });
});
```





Paso 2b: inicializar la DB

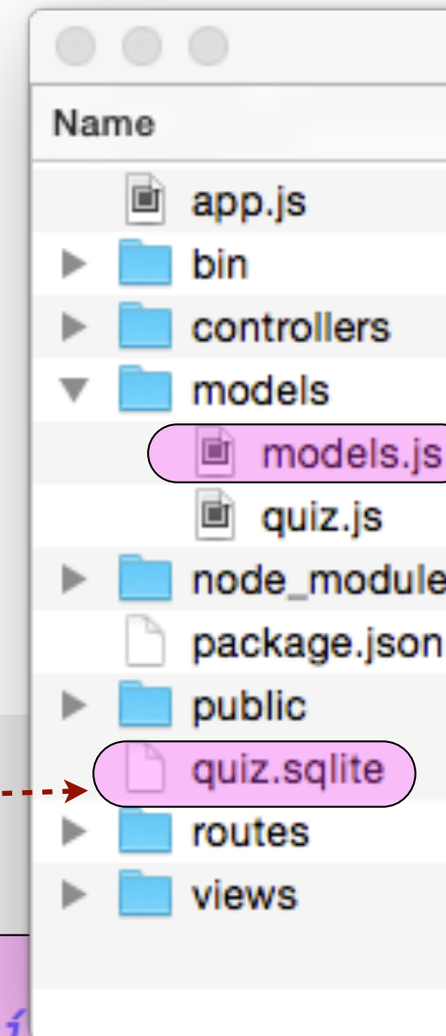
id	pregunta	respuesta
1	¿Cual es la Capital de Italia?	Roma
2		

- El método **sequelize.sync()** crea automáticamente el fichero **quiz.sqlite** con la **DB** y sus datos iniciales, si la **DB** no existe. Si existe sincroniza con nuevas definiciones del modelo, siempre que sean compatibles con anteriores.

- El callback de **sequelize.sync().success(funtion(){..})** se ejecuta cuando el fichero **quiz.sqlite** esta sincronizado. El código de esta función introduce en la DB la pregunta de la versión anterior, para que todo funcione igual.

- **Quiz.count().success(..)** devuelve en count el número de filas de la tabla.

- **Quiz.create(.. objeto ..)** crea la primera pregunta y la guarda en la tabla. Los campos de la tabla deben tener el mismo nombre que las propiedades

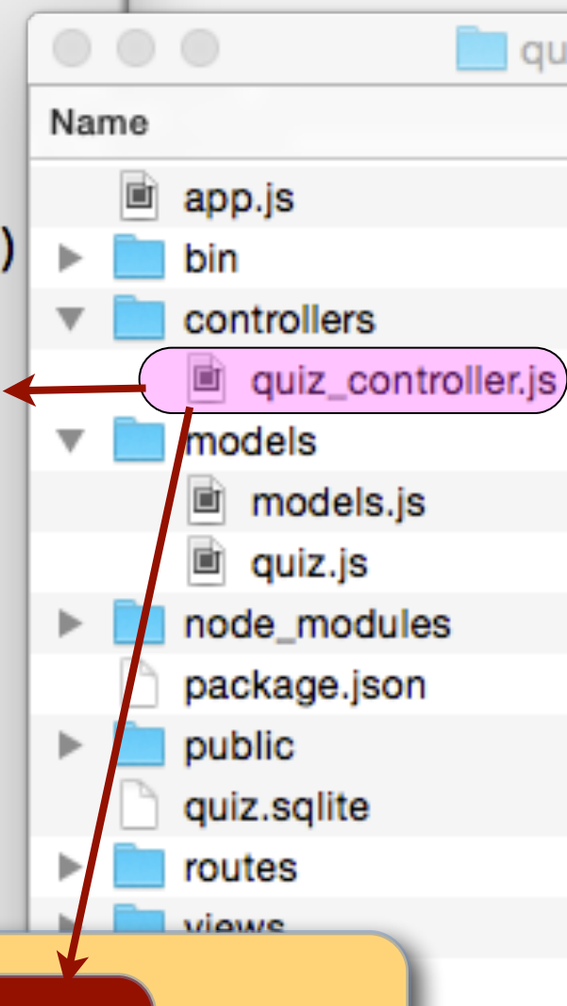



```
// sequelize.sync() crea e inicializa tabla de preguntas en DB
sequelize.sync().success(function() {
  // success(..) ejecuta el manejador una vez creada la tabla
  Quiz.count().success(function (count){
    if(count === 0) {   // la tabla se inicializa solo si está vacía
      Quiz.create({ pregunta: 'Capital de Italia',
                    respuesta: 'Roma'
                  })
      .success(function(){console.log('Base de datos inicializada')});
    };
  });
});
```

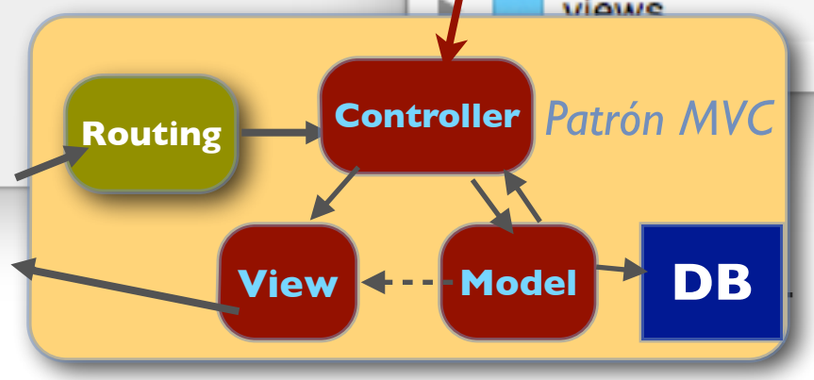
- El controlador importa el modelo para poder acceder a DB.
- Con los métodos **models.Quiz.findAll()** o **find()** buscamos los datos en la tabla **Quiz** y los procesamos en el callback del método **success(..)**.
- En el ejemplo usamos **findAll()** para buscar el array de elementos de la tabla **Quiz** y como solo tiene una pregunta en la tabla, cogemos el primer elemento **quiz[0].**

quiz_controller.js UNREGISTERED

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

// GET /quizes/question
exports.question = function(req, res) {
  models.Quiz.findAll().success(function(quiz) {
    res.render('quizes/question', { pregunta: quiz[0].pregunta})
  })
};

// GET /quizes/answer
exports.answer = function(req, res) {
  models.Quiz.findAll().success(function(quiz) {
    if (req.query.respuesta === quiz[0].respuesta) {
      res.render('quizes/answer', { respuesta: 'Correcto' });
    } else {
      res.render('quizes/answer', { respuesta: 'Incorrecto'});
    }
  })
};
```



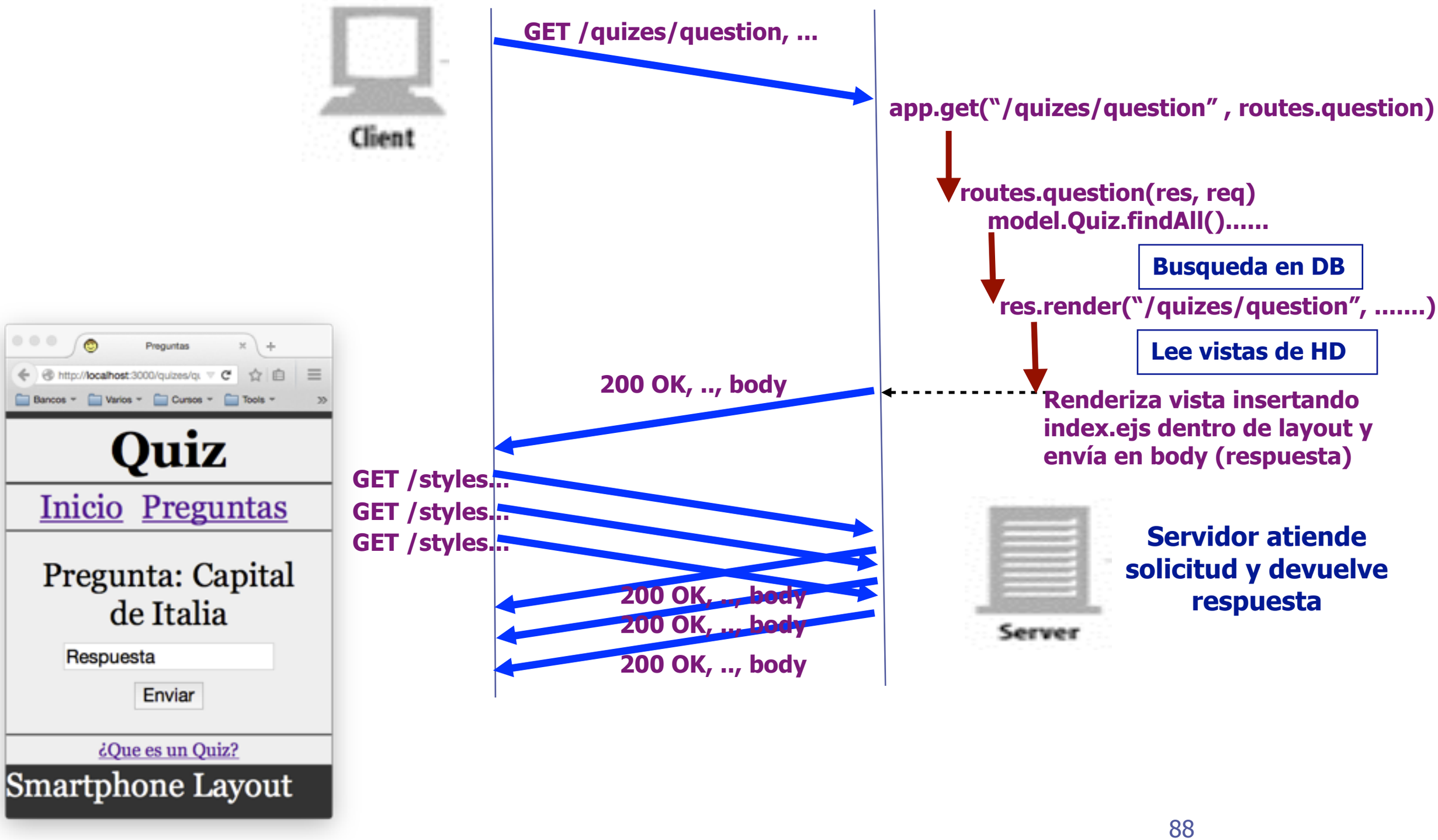
Paso 3: controlador



Final del tema

©

Ejecución de la transacción: GET /index



Quiz 8: Despliegue DB en Heroku

Juan Quemada, DIT - UPM

Quiz 6: Despliegue DB en Heroku

Objetivo: Adaptar la configuración de sequelize.js para que en el entorno local use la base de datos SQLite y en el despliegue en Heroku utilice la base de datos Postgres.

SQLite es una base de datos muy sencilla, adecuada para un entorno de desarrollo, pero para dar servicio real es preferible usar una base de datos con mejores prestaciones como Postgres.

- Los pasos a seguir son los siguientes
  - Paso 1: Añadir la base de datos Postgres como un add-on al despliegue en Heroku
  - Paso 2: Añadir el URL de acceso la DB en la variable de entorno DATABASE_URL
  - Paso 3: Pasar la dependencia de SQLite a “devDependencies”
  - Paso 4: Incluir dependencia de base de datos Postgres en package.json
  - Paso 5: Añadir fichero local .env con variables de entorno para SQLite
  - Paso 6: Adaptar models/models.js para el entorno local y en Heroku
  - Paso 7: Prueba de ejecución local con foreman start
  - Paso 8: Guardar versión en git y desplegar en heroku
- Tutorial de despliegue con node.js:
  - https://devcenter.heroku.com/articles/getting-started-with-nodejs#provision-a-database

Pasos 1 y 2: Añadir DB Postgress a Heroku

✦P1: desplegar DB Postgres en Heroku

- Puede hacerse de 2 formas
  1. Utilizando el Dashboard de Heroku
  2. Utilizando Heroku Toolbelt desde nuestra consola local

✦P1a: desplegar desde el Dashboard

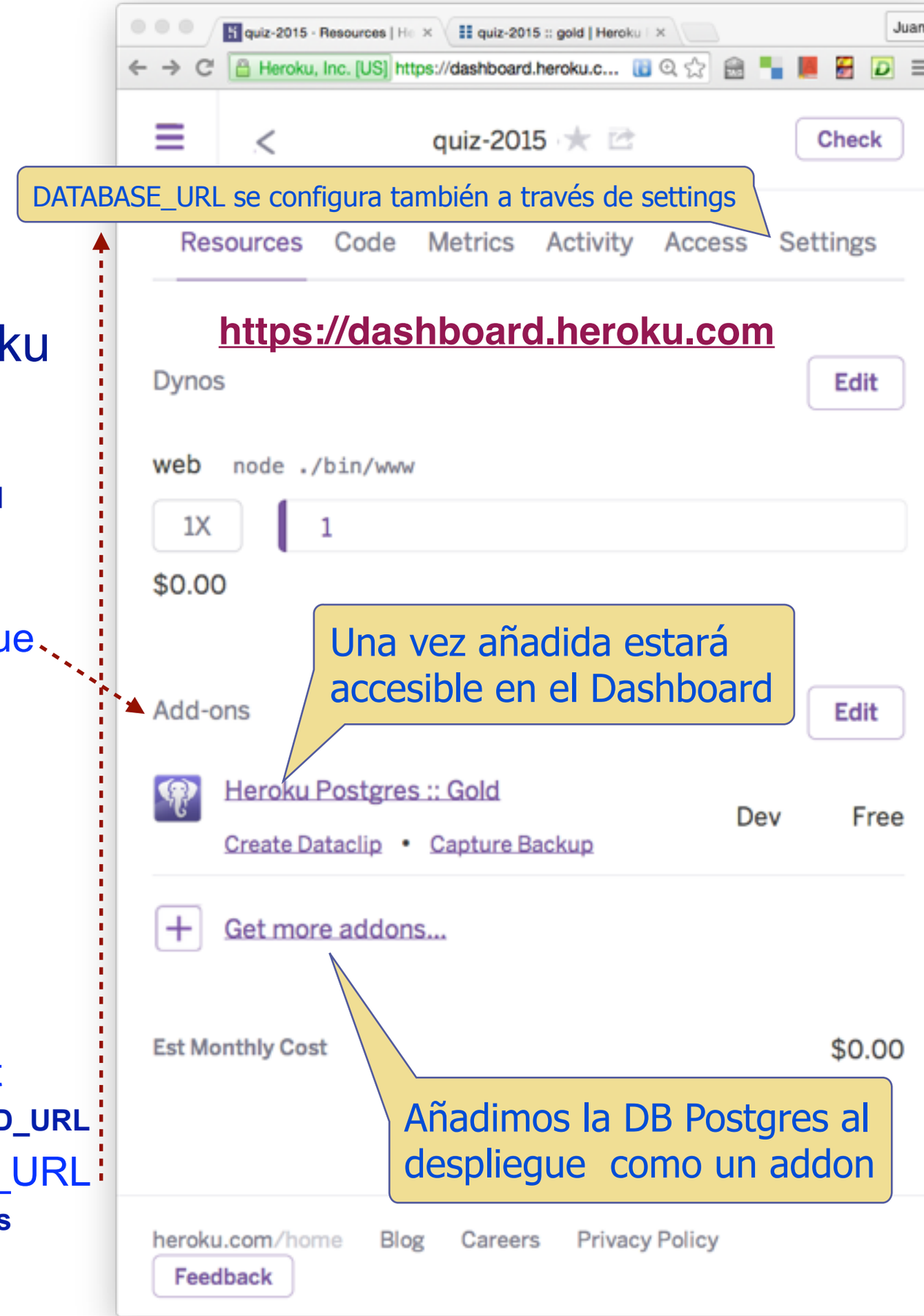
- Debe añadirse como un addon al despliegue

✦P1b: desplegar con Heroku Toolbelt

- Debe ejecutarse el comando
  - **heroku addons:add heroku-postgresql:dev**
- La documentación se despliega con
  - **heroku addon:docs heroku-postgresql**

✦P2: Inicializar DATABASE_URL

- Inicializar con comando de Heroku Toolbelt
  - **heroku pg:promote HEROKU_POSTGRESQL_GOLD_URL**
- Dashboard permite configurar DATABASE_URL
  - En el apartado **Config Variables** del **panel de settings**



Heroku, Inc. (US) | https://postgres.heroku.com/databases/quiz-2015-heroku-postgresql-gold

Bancos | Varios | Cursos | Tools | Rails | HTML5 | HTML5-RTC | MiTwitter | CTING | DIT | FloorControl | Most Visited | Global3 | PPP-FI-BSCW

quiz-2015 :: gold

Base de datos Postgres desplegada para Quiz en Heroku

Connection Settings

Host	ec2-184-73-229-220.compute-1.amazonaws.com
Database	dcn147eorv25ra
User	tfewgztclgzzpw
Port	5432
Password	Hide
Psql	heroku pg:psql --app quiz-2015 HEROKU_POSTGRESQL_GOLD
URL	Hide postgres://tfewgztclgzzpw@ec2-184-73-229-220.compute-1.amazonaws.com:5432/dcn147eorv25ra

Parámetros individuales extraidos del URL

There are a variety of ways to connect to a Heroku Postgres database, find out how to do so via psql, Java, Ruby, or Node.js.

URL de la DB Postgres generado automáticamente por Heroku. Le asigna un color aleatorio, GOLD en este caso.

Este URL se asigna a la variable de entorno DATABASE_URL con el comando de Heroku Toolbelt:

$heroku pg:promoteb HEROKU_POSTGRESQL_GOLD_URL

Statistics

Plan	Dev
Status	Available
Connections	0/20
Data Size	6.5 MB
Tables	1
PG Version	9.3.5
Created	2015-01-06T20:42:00Z
Rollback	Unsupported
Rows	1/10000 (In compliance)

Estadísticas de uso de la base de datos en Heroku

Dashboard de Heroku

Modificar proyecto Quiz

✦Paso 3: crear parámetro "**devDependencies**" en package.json

- Pasar "**sqlite3**" de "**dependencies**" a "**devDependencies**"
  - "**devDependencies**" se instalarán solo cuando instalemos el paquete en el entorno local

✦Paso 4: añadir dependencia a Postgres a package.json

- Comando "**npm install --save ..**" (guarda instalación en package.json)

```
// Instalar la DB Postgres en el entorno local guardando la dependencia en
// package.json para que al desplegar en Heroku se despliegue también

$ npm install --save pg
$ npm install --save pg-hstore  //Ojo! en proyecto en GITHUB solo esta pg
```

package.json UNREGISTERE

```
{
  "name": "quiz",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "engines": {
    "node": "0.10.x",
    "npm": "1.4.x"
  },
  "devDependencies": {
    "sqlite3": "^3.0.4"
  },
  "dependencies": {
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "debug": "~2.0.0",
    "ejs": "~0.8.5",
    "express": "~4.9.0",
    "express-partials": "^0.3.0",
    "morgan": "~1.3.0",
    "pg": "^4.1.1",
    "sequelize": "^2.0.0-rc4",
    "serve-favicon": "~2.1.3"
  }
}
```

✦Paso 5: Añadir variables de entorno a despliegue local

- DATABASE_URL y DATABASE_STORAGE se crean en fichero .env
  - El fichero .env se creará en el directorio raíz del proyecto con el siguiente contenido

.env UNREGISTERED

```
DATABASE_URL=sqlite://:@:/
DATABASE_STORAGE=quiz.sqlite
```

- El nombre .env se ha añadido a .gitignore
  - Ha sido un error y se quito de .gitignore en Quiz 19
    - Sin .env no aranca bien con "foreman start"

.gitig UNREGISTERED

```
node_modules
.DS_Store
npm-debug.log
quiz.sqlite
.env
```


©

Paso 6: adaptar modelo a despliegue en Heroku

models.js

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

// Postgres DATABASE_URL = postgres://user:passwd@host:port/database
// SQLite   DATABASE_URL = sqlite://:@:/
var url = process.env.DATABASE_URL.match(/(.*)\:\/\/(.*?)\:(.*)@(.*)\:(.*)\/(.*)/);
var DB_name  = (url[6]||null);
var user     = (url[2]||null);
var pwd      = (url[3]||null);
var protocol = (url[1]||null);
var dialect  = (url[1]||null);
var port     = (url[5]||null);
var host     = (url[4]||null);
var storage  = process.env.DATABASE_STORAGE;

// Cargar Modelo ORM
var Sequelize = require('sequelize');

// Usar BBDD SQLite o Postgres
var sequelize = new Sequelize(DB_name, user, pwd,
  { dialect:  protocol,
    protocol: protocol,
    port:     port,
    host:     host,
    storage:  storage,  // solo SQLite (.env)
    omitNull: true      // solo Postgres
  }
);

// Importar definicion de la tabla Quiz
var quiz_path = path.join(__dirname,'quiz');
var Quiz = sequelize.import(quiz_path);

exports.Quiz = Quiz; // exportar tabla Quiz

// sequelize.sync() inicializa tabla de preguntas en DB
sequelize.sync().then(function() {
  // then(..) ejecuta el manejador una vez creada la tabla
```

- ✦ Adaptar **models/models.js**
  - La DB se configura ahora con las variables
    - DATABASE_URL en entornos local y heroku
    - DATABASE_STORAGE solo en entorno local
      - En node ambas son **propiedades de process.env**
- ✦ URL de DATABASE_URL
  - El formato del URL es
    - Postgres: postgres://user:passwd@host:port/database
    - SQLite: sqlite://:@:/
  - Extraemos sus componentes con RegExp
    - match(/(.*)\:\/\/(.*?)\:(.*)@(.*)\:(.*)\/(.*)/)
      - que devuelve un array con los parámetros
  - Los parámetros se generan así bién
    - tanto para entorno local como para Heroku Postgres
- ✦ El entorno local con SQLite
  - Necesita además el parámetro storage
    - que se define con la variable DATABASE_STORAGE

©

Ejecución local, versión y despliegue en Heroku

- ✦Paso 7: A partir de ahora deberemos arrancar la aplicación localmente
  - Con el comando "foreman start" de Heroku Toolbelt
    - **OJO!** "foreman start" arranca el servidor en el puerto 5000, habrá que probar con URLs: "**http://localhost:5000/….**"
- ✦Paso 8a: Generamos una nueva versión en la rama master
  - Con el comando: **git commit -m "identificación de versión"**
    - La última versión puede ser modificada con: **git add .** y luego **git commit —amend**
- ✦Paso 8b: Desplegamos en Heroku con: "**git push heroku master**"
  - "**git push heroku master -f**" cambia la versión desplegada en Heroku por una modificada
    - Así podemos **corregir errores en la versión desplegada** cuando modificamos la versión local con **--amend**

```
// A partir de esta versión deberemos arrancar la aplicación localmente con el comando foreman
..$ foreman start      // Ejecutar foreman en el directorio quiz arranca el servidor el el puerto 5000

// OJO! a partir de ahora debemos usar URLS de tipo "http://localhost:5000/…." para acceso local

..$ git add .      // añadimos los cambios al índice

                   // generamos versión "Despliegue DB en Heroku" en la rama master
..$ git commit -m "Despliegue DB en Heroku"

                   // Subir la última versión de la rama master a Heroku
..$ git push heroku master      // !!!Quiz se ha desplegado en Heroku con Postgres!!!
```

Final del tema

©

Ejercicio: Desplegar Quiz con página de autores en Heroku

Añadir a la versión Quiz 8 “Despliegue DB en Heroku” una página adicional de autor que estará accesible a través de la primitiva del interfaz: “GET /author”.

La página se enlazara desde la barra de navegación con un enlace adicional, de nombre autor.

La página incluirá el nombre de los 2 autores (miembros de la pareja), una foto y un pequeño video de presentación del portal.

Desplegar dicha versión en Heroku y entregar el enlace a dicha versión por Moodle.

Recomendaciones:

1) Hacer una rama en un clon del proyecto Quiz desde Quiz 8, que añada dicha funcionalidad y desplegar dicha rama.
2) Integrar Quiz 8 con “merge …” en la rama de introducción de las extensiones anteriores solicitadas en los ejercicios entregables.

Quiz 9: Lista de preguntas

Juan Quemada, DIT - UPM

Quiz 9: Múltiples preguntas

Objetivo: Modificar la aplicación Quiz para que gestione correctamente una lista de preguntas, en vez de una única pregunta. Para ello hay que adaptar MVC de pregunta y respuesta, y añadir la lista de preguntas a MVC.

✦**Paso 1**: Adaptar MVC de pregunta y respuesta a colección de recursos

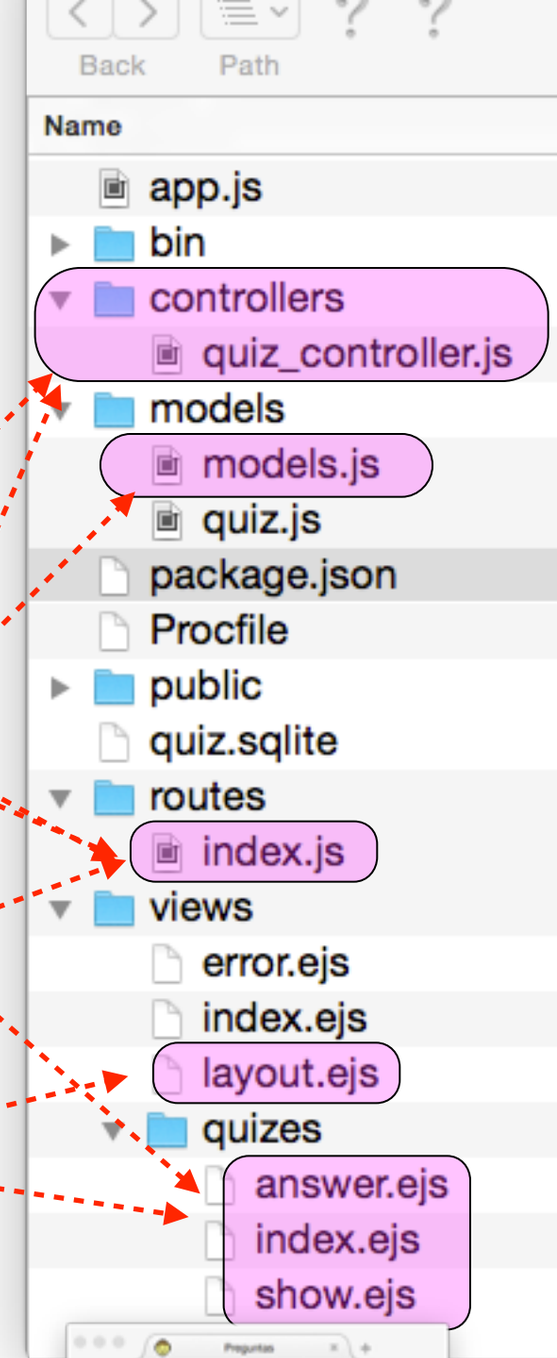
- a: Cambiar en controlador **quiz_controller.js**: **question** por **show** y modificar **answer**
- b1: Cambiar en **index.js**: **GET /quizes/question** por **GET /quizes/:quizId**
- b1: Cambiar en **index.js**: **GET /quizes/answer** por **GET /quizes/:quizId/answer**
- c: Adaptar vistas cambiando **answer.ejs** por **show.ejs** y modificando **answer.ejs**

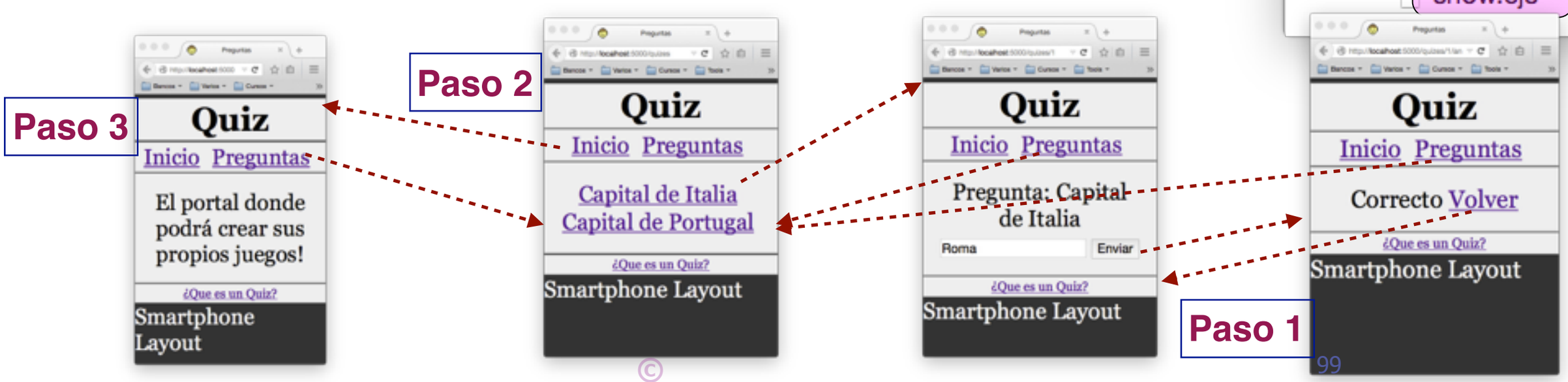
✦**Paso 2**: Añadir lista de preguntas: **GET /quizes**

- a: Añadir al controlador **quiz_controller.js** la acción **index** asociada a la ruta **/quizes**
- b: Añadir otra pregunta al inicializar la base de datos en **models/models.js**
- c: Añadir la ruta **/quizes** (lista de preguntas) al enrutador **routes/index.js**
- d: Crear nueva vista con lista de preguntas **views/quizes/index.ejs**

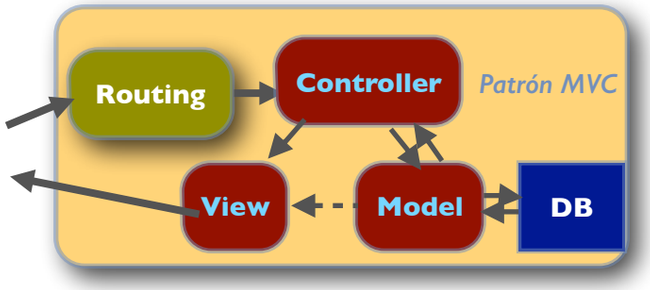
✦**Paso 3**: Modificar **layout.ejs** para enlazar con la lista de preguntas

✦**Paso 4**: Guardar **versión (commit)** git y subir a **Heroku**





Recursos y nombres



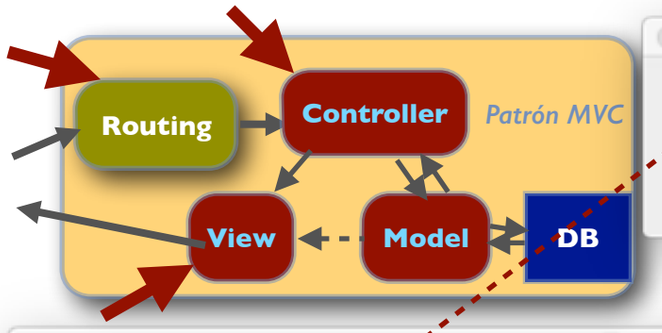
Cuando se diseña según el patrón MVC una tabla de una base de datos suele contener una lista de recursos. Para facilitar el diseño se siguen convenios de nombres similares para los elementos:

- La tabla suele tener el mismo nombre del recurso y por eso hemos denominado al modelo **models/quiz.js** y al objeto que implementa el acceso ORM a la tabla **Quiz**.
- El controlador de acceso a esa tabla suele llevar el nombre delante: **controllers/quiz_controller.js**
- Las vistas de un recurso van en un directorio con el nombre del recurso: **quizes**
- Las rutas (path) también empiezan por el nombre del recurso en plural: **/quizes**.
  - La lista de recursos usa el nombre del recurso en plural: **/quizes**
  - Un recurso concreto se identifica por el índice en la tabla como parámetro de ruta con una expresión regular que restringe a números: **/quizes/:quizId(\\d+)**
  - Otras acciones sobre un recurso concreto como comprobar la respuesta, llevan además el nombre de la acción: **/quizes/:quizId(\\d+)/answer**

id	pregunta	respuesta
1	**Capital de Italia**	Roma
2	**Capital de Portugal**	Lisboa
3	.....	.....
4	.....	.....
5	.....	.....

index.js UNREGISTERED

```
// Definición de rutas de /quizes
router.get('/quizes',                        quizController.index);
router.get('/quizes/:quizId(\\d+)',          quizController.show);
router.get('/quizes/:quizId(\\d+)/answer',   quizController.answer);
```



Paso 1b: rutas **/quizes/:quizId(\\d+)** y **/quizes/:quizId(\\d+)/answer**

```
// Definición de rutas de /quizes
router.get('/quizes',                        quizController.index);
router.get('/quizes/:quizId(\\d+)',          quizController.show);
router.get('/quizes/:quizId(\\d+)/answer',   quizController.answer);
```

routes/index.js

```
// GET /quizes/:id
exports.show = function(req, res) {
  models.Quiz.find(req.params.quizId).then(function(quiz) {
    res.render('quizes/show', { quiz: quiz});
  })
};

// GET /quizes/:id/answer
exports.answer = function(req, res) {
  models.Quiz.find(req.params.quizId).then(function(quiz) {
    if (req.query.respuesta === quiz.respuesta) {
      res.render('quizes/answer',
                 { quiz: quiz, respuesta: 'Correcto'});
    } else {
      res.render('quizes/answer',
                 { quiz: quiz, respuesta: 'Incorrecto'});
    }
  })
};
```

controllers/quiz_controller.js

Paso 1

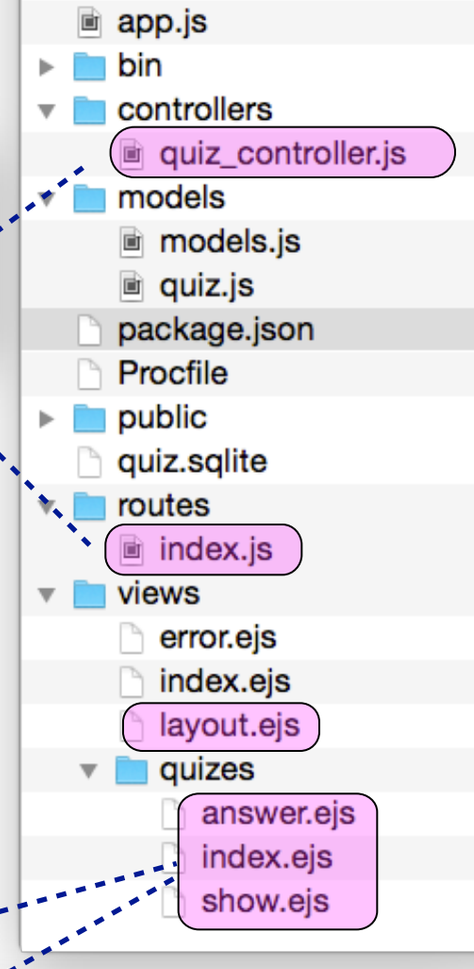
```
<p>
  <%= respuesta %>
  <a href="/quizes/<%= quiz.id %>">Volver</a>
</p>
```

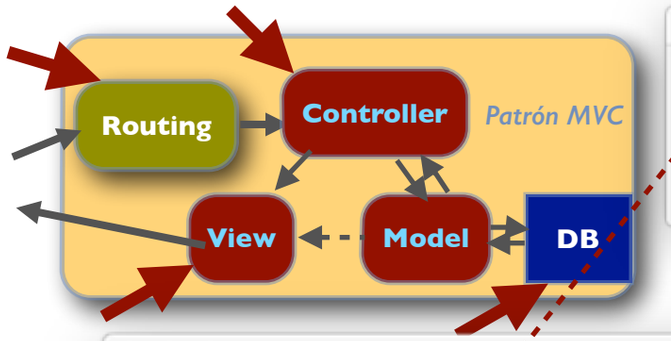
Paso 1c: vista views/quiz/answer.ejs

Paso 1a: controladores de **show** y **answer**

Paso 1c: vista views/quizes/show.ejs

```
<form method="get" action="/quizes/<%= quiz.id %>/answer">

  Pregunta: <%= quiz.pregunta %> <p>
  <input type="text" name="respuesta" value="Respuesta"/>
  <input type="submit" value="Enviar">
</form>
```





Paso 2c: rutas /quizes/:id y /quizes/:id/answer

```
// Definición de rutas de /quizes
router.get('/quizes',                          quizController.index);
router.get('/quizes/:quizId(\\d+)',            quizController.show);
router.get('/quizes/:quizId(\\d+)/answer',     quizController.answer);
```

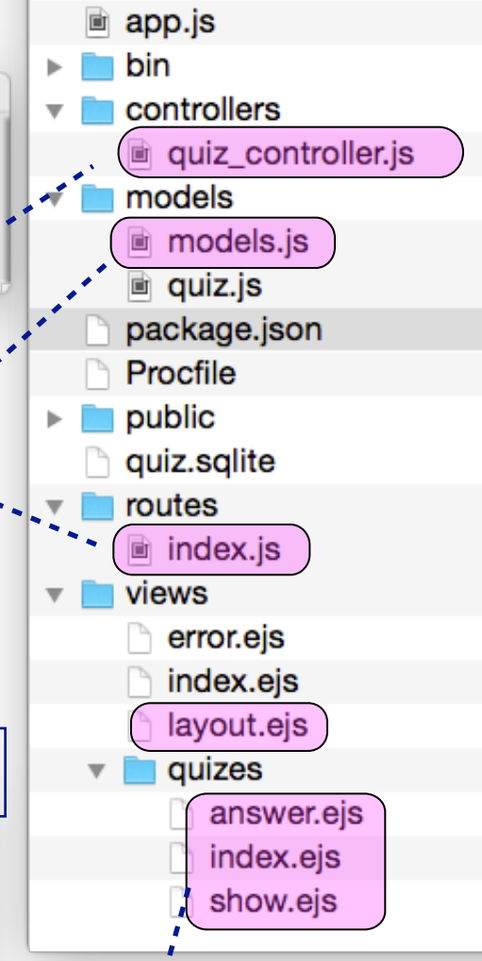
routes/index.js

Paso 2a: controlador de index

```
// GET /quizes
exports.index = function(req, res) {
  models.Quiz.findAll().then(function(quizes) {
    res.render('quizes/index.ejs', { quizes: quizes});
  })
};
```

controllers/quiz_controller.js

Paso 2



models/models.js

```
// sequelize.sync() inicializa tabla de preguntas en DB
sequelize.sync().then(function() {
  // then(..) ejecuta el manejador una vez creada la tabla
  Quiz.count().then(function (count){
    if(count === 0) {   // la tabla se inicializa solo si está vacía
      Quiz.create({ pregunta: 'Capital de Italia',
                    respuesta: 'Roma'
                  });
      Quiz.create({ pregunta: 'Capital de Portugal',
                    respuesta: 'Lisboa'
                  })
      .then(function(){console.log('Base de datos inicializada')});
    };
  });
});
```

Paso 2b: inicializar DB

id	pregunta	respuesta
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....
5	.....	.....

Paso 2d: vista views/quizes/index.ejs

```
<table>
<% var i; for (i=0; i < quizes.length; i++) { %>
  <tr><td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td></tr>
<% } %>
</table>
```

Paso 3

```
16
17 <body>
18 <div id="page-wrap">
19   <header class="main" id="h1">
20     <h2>Quiz<span>: el juego de las preguntas</span></h2>
21   </header>
22
23   <nav class="main" id="n1" role="navigation">
24     <span><a href="/">Inicio</a></span>
25     <span><a href="/quizes">Preguntas</a></span>
26   </nav>
27
28   <section class="main" id="s1">
29     <div>  <table>
30
31      <tr><td><a href="quizes/1">Capital de Italia</a></td></tr>
32
33      <tr><td><a href="quizes/2">Capital de Portugal</a></td></tr>
34
35   </table>
36
37 </div>
38   </section>
39
40   <footer class="main" id="f1">
41     <a href="http://es.wikipedia.org/wiki/Quiz">
42       ¿Que es un Quiz?</a>
43   </footer>
44 </div>
45 </body>
46 </html>
```

Paso 3: enlaze a **/quizes**

models.js
quiz.js
package.json
Procfile
public
quiz.sqlite
routes
index.js
views
error.ejs
index.ejs
layout.ejs
quizes
answer.ejs
index.ejs
show.ejs

Código HTML de
- layout.ejs +
- views/quizes/index.ejs
(Captura de Firefox
Tools -> Web Developer -> Page-Source)

Quiz
Inicio Preguntas
Capital de Italia
Capital de Portugal
¿Que es un Quiz?
Smartphone Layout

Vista views/quizes/index.ejs

index.ejs

```
<table>
<% var i; for (i=0; i < quizes.length; i++) { %>
   <tr><td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td></tr>
<% } %>
</table>
```

Bucle-for: renderíza el código HTML del bloque en cada iteración. Generará una fila de la tabla HTML por cada elemento del array quizes, que contiene todos los objetos de la tabla Quiz de la DB,

Final del tema

Ejercicio: Quiz con cajetín de búsqueda I

Añadir a la versión Quiz 9 “Lista de preguntas” un cajetín de búsqueda el la parte superior de la cabecera principal que permita buscar quizes en la DB.

En el cajetín se podrá teclear un texto que incluya cualquier carácter o símbolo, incluso caracteres en blanco o retornos de carro con caracteres escapados.

El resultado de la búsqueda se mostrará en una página accesible a través de la primitiva del interfaz: “GET /search?search=….”.

La página mostrará los quizes, que incluyan el texto tecleado como parte de la pregunta, ordenados alfabéticamente.

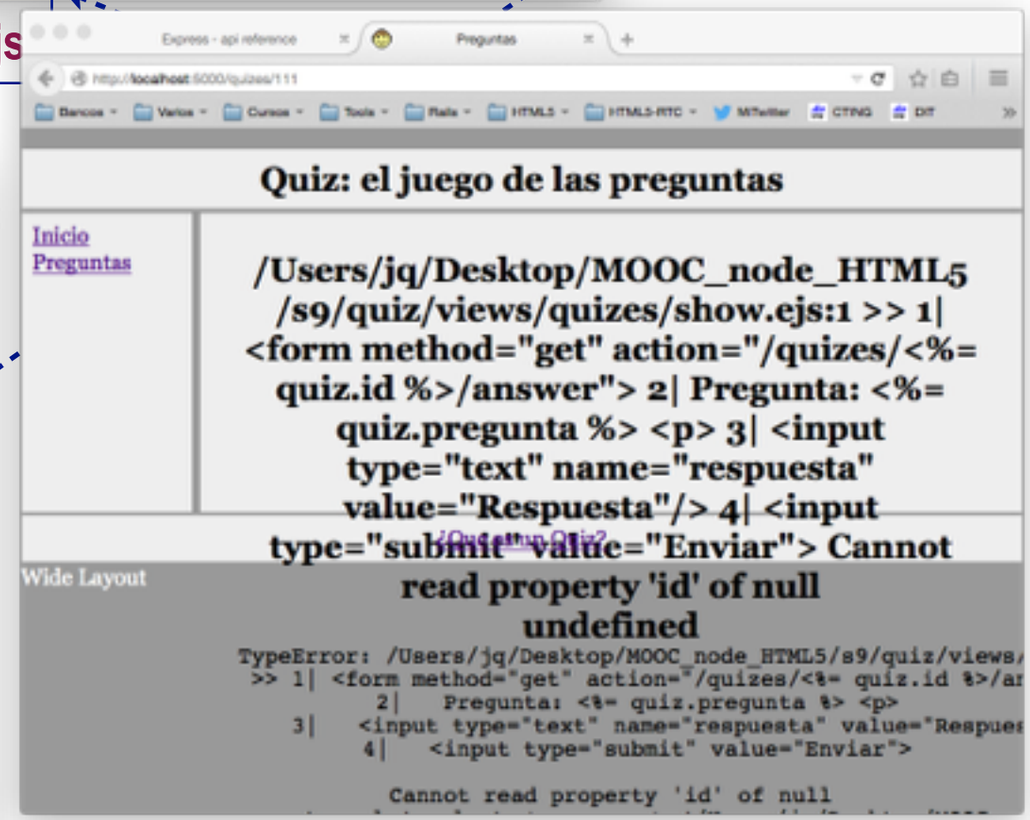
Desplegar dicha versión en Heroku y entregar el enlace a dicha versión por Moodle.

Recomendación: Integrar Quiz 9 con “merge …” en la rama de introducción de las extensiones anteriores solicitadas en los ejercicios entregables y añadir después la funcionalidad pedida.

Paso 1a: rutas /quizes/:quizId(\\d+) y /quizes/:quizId(\\d+)/answer

```
// Definición de rutas de /quizes
router.get('/quizes',                          quizController.index);
router.get('/quizes/:quizId(\\d+)',            quizController.show);
router.get('/quizes/:quizId(\\d+)/answer',     quizController.answer);
```

routes/index.js



controllers/quiz_controller.js

```
// GET /quizes/:id
exports.show = function(req, res) {
  models.Quiz.find(req.params.quizId).then(function(quiz) {
    res.render('quizes/show', { quiz: quiz});
  })
};

// GET /quizes/:id/answer
exports.answer = function(req, res) {
  models.Quiz.find(req.params.quizId).then(function(quiz) {
    if (req.query.respuesta === quiz.respuesta) {
      res.render('quizes/answer',
                 { quiz: quiz, respuesta: 'Correcto'});
    } else {
      res.render('quizes/answer',
                 { quiz: quiz, respuesta: 'Incorrecto'});
    }
  })
};
```

Paso 1a: controladores de **show** y **answer**

Paso 1c: vista views/quizes/show.ejs

```
<form method="get" action="/quizes/<%= quiz.id %>/answer">

  Pregunta: <%= quiz.pregunta %> <p>
  <input type="text" name="respuesta" value="Respuesta"/>
  <input type="submit" value="Enviar">
</form>
```

Aquí se añade también la expresión regular **(\\d+)** a la ruta **GET /quizes/:id(\\d+)**, porque si no restringimos **:id** solo a valores numéricos, aceptara también **GET /quizes/new**. Esto se resuelve también situando **GET /quizes/new** antes que **GET /quizes/:id** en **routes/index.js**.

Quiz 10: Autoload

Juan Quemada, DIT - UPM

Quiz 10: Autoload

Objetivo: Factorizar la carga de un objeto de la DB como un MW de autoload que gestiona la lectura y los casos de error. Los errores se enviarán al MW de error generado por express-generator.

✦**Paso 1**: Introducir gestión de errores en **GET /quizes**

- La gestión de errores se introduce directamente en la primitiva

✦**Paso 2**: Definir la función **load()** de lectura de un quiz identificado por **:quizId**

- Gestiona también los errores de **GET /quizes/:quizId** y **GET /quizes/:quizId/answer**

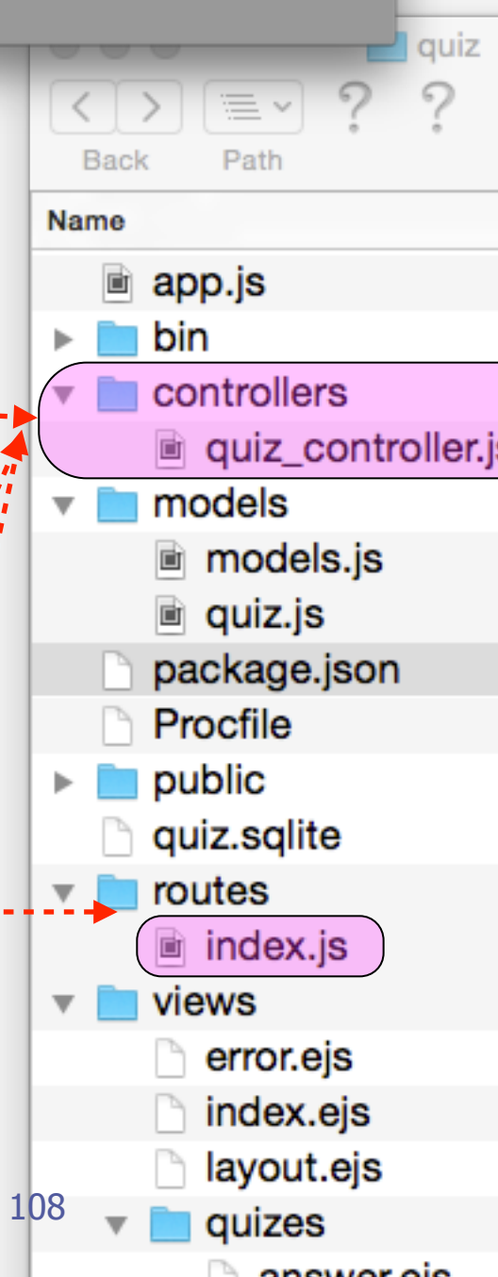
✦**Paso 3**: Adaptarlas siguientes rutas de acceso a preguntas para usar **load():**

- **GET /quizes/:quizId**
- **GET /quizes/:quizId/answer**

✦**Paso 4**: Instalar el middleware load() en **router/index.js**

- Instalada antes de **index.js** para que **load()** cargue quiz **antes** de invocar la ruta asociada

✦**Paso 4**: Guardar **versión (commit)** git y subir a **Heroku**



controllers/quiz_controller.js

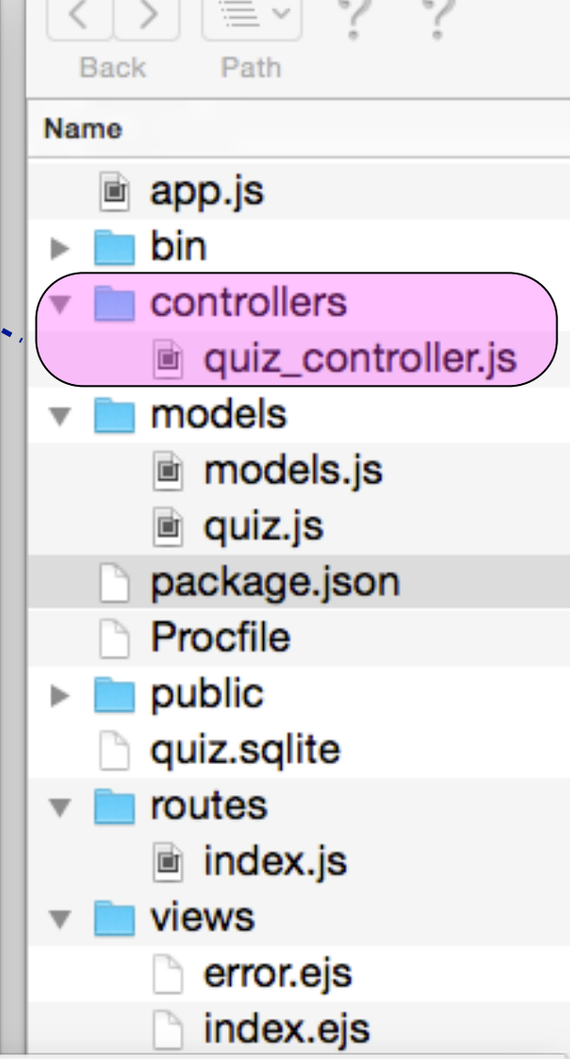
```
var models = require('../models/models.js');

// Autoload - factoriza el código si ruta incluye :quizId
exports.load = function(req, res, next, quizId) {
  models.Quiz.find(quizId).then(
    function(quiz) {
      if (quiz) {
        req.quiz = quiz;
        next();
      } else { next(new Error('No existe quizId=' + quizId)); }
    }
  ).catch(function(error) { next(error);});
};

// GET /quizes
exports.index = function(req, res) {
  models.Quiz.findAll().then(
    function(quizes) {
      res.render('quizes/index', { quizes: quizes});
    }
  ).catch(function(error) { next(error);})
};

// GET /quizes/:id
exports.show = function(req, res) {
  res.render('quizes/show', { quiz: req.quiz});
};

// GET /quizes/:id/answer
exports.answer = function(req, res) {
  var resultado = 'Incorrecto';
  if (req.query.respuesta === req.quiz.respuesta) {
    resultado = 'Correcto';
  }
  res.render('quizes/answer', {quiz: req.quiz, respuesta: resultado});
};
```

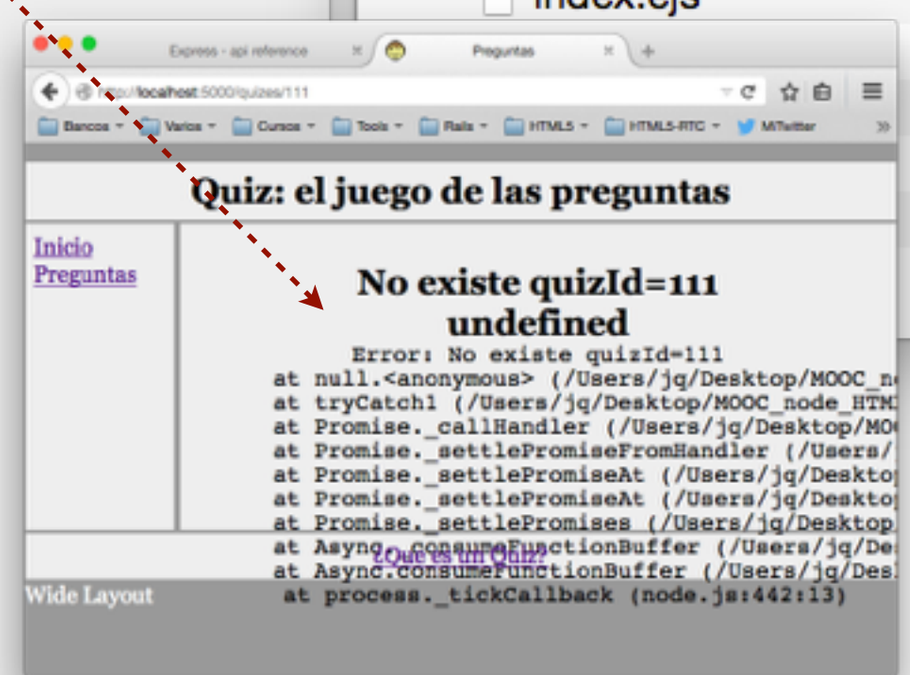
Paso 2

Paso 1c: vista views/quizes/show.ejs

Paso 1

Paso 3





Ejecución de load()

index.js UNREGISTERED

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

var quizController = require('../controllers/quiz_controller');

// Página de entrada (home page)
router.get('/', function(req, res) {
  res.render('index', { title: 'Quiz' });
});

// Autoload de comandos con :quizId
router.param('quizId', quizController.load);  // autoload :quizId

// Definición de rutas de /quizes
router.get('/quizes',                        quizController.index);
router.get('/quizes/:quizId(\\d+)',          quizController.show);
router.get('/quizes/:quizId(\\d+)/answer', quizController.answer);

module.exports = router;
```

Paso 4

- app.js
- bin
- controllers
  - quiz_controller.js
- models
  - models.js
  - quiz.js
- package.json
- Procfile
- public
- quiz.sqlite
- routes
  - index.js
- views
  - error.ejs
  - index.ejs
  - layout.ejs
  - quizes
    - answer.ejs
    - index.ejs
    - show.ejs

quizController.load() se instala para que **se ejecute antes** que lo necesiten las rutas show y answer y solo en el caso de que **path contenga :quizId**, referenciando un recurso en la tabla Quiz de la base de datos que deba ser procesado por el controlador.

Se instala con el método **param()** de express (http://expressjs.com/4x/api.html#router.param), para que **router.param('quizId', quizController.load)** solo invoque **quizController.load** si existe el parámetro **:quizId** está en algún lugar de la cabecera HTTP (en query, body o param).

Final del tema

©

Ejercicio: Quiz con cajetín de búsqueda II

Añadir a la versión Quiz 10 “Autoload” un cajetín de búsqueda el la parte superior de la cabecera principal que permita buscar quizes en la DB.

En el cajetín se podrá teclear un texto que incluya cualquier carácter o símbolo, incluso caracteres en blanco o retornos de carro con caracteres escapados.

El resultado de la búsqueda se mostrará en una página accesible a través de la primitiva del interfaz: “GET /search?search=….”.

La página mostrará los quizes, que incluyan el texto tecleado como parte de la pregunta, ordenados alfabéticamente.

Desplegar dicha versión en Heroku y entregar el enlace a dicha versión por Moodle.

Recomendación: Integrar Quiz 10 con “merge …” en la rama donde se están introduciendo la extensión solicitada en el ejercicio anterior (Quiz 9).

También se puede hacer una rama en un clon del proyecto Quiz desde Quiz 10, que añada la funcionalidad pedida en Quiz 9 y en Quiz 5 (no recomendado).

Quiz 11: Crear preguntas

Juan Quemada, DIT - UPM

Quiz 10: Crear preguntas

Objetivo: Introducir en la aplicación Quiz un formulario que permita crear preguntas nuevas y añadirlas a la base de datos.

✦**Paso 1**: Añadir formulario de creación de preguntas en **GET /quizes/new**

- a: Añadir en **quiz_controller.js** la acción **new** asociada a la ruta **/quizes/new**
- b: Añadir en **routes/index.js** la ruta **GET /quizes/new**
- c: Añadir vista con formulario de creación de pregunta: **views/quizes/new.ejs**

✦**Paso 2**: Añadir **POST /quizes/create** para añadir preguntas a DB

- a: Añadir controlador **create** a **quiz_controller.js**
- b: Añadir el filtro de **POST /quizes/create** en **routes/index.js**

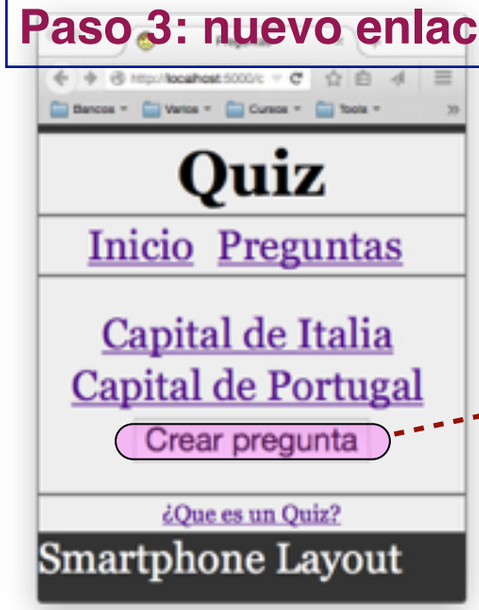
✦**Paso 3**: Cambiar config. **bodyParser.urlencoded()** en **app.js**

✦**Paso 4**: Añadir en vista **index.ejs** enlace a creación de preguntas

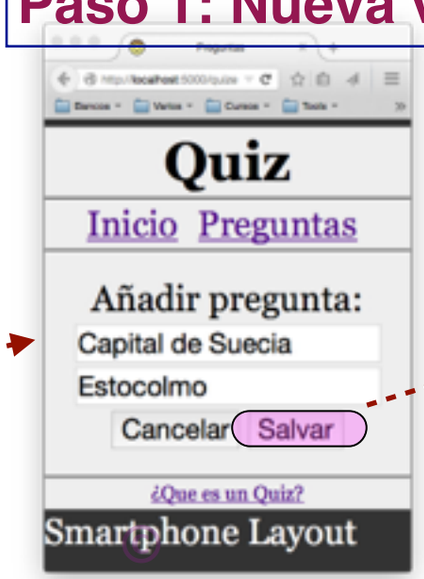
✦**Paso 5**: Guardamos **versión (commit)** git y subir a **Heroku**

id	pregunta	respuesta
1	**Capital de Italia**	Roma
2	**Capital de Portugal**	Lisboa
3	**Capital de Suecia**	Estocolmo
4	.....	.....
5	.....	.....

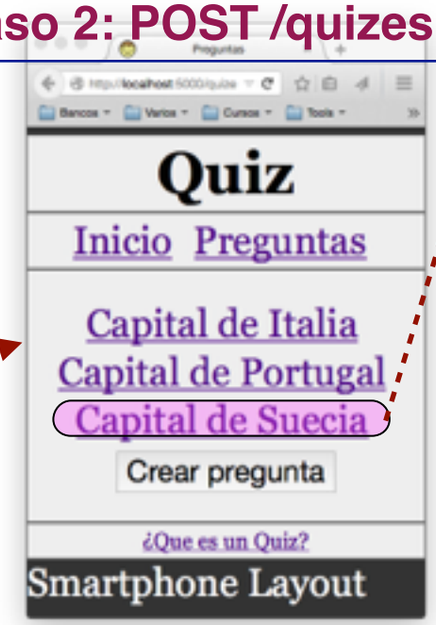
Paso 3: nuevo enlace



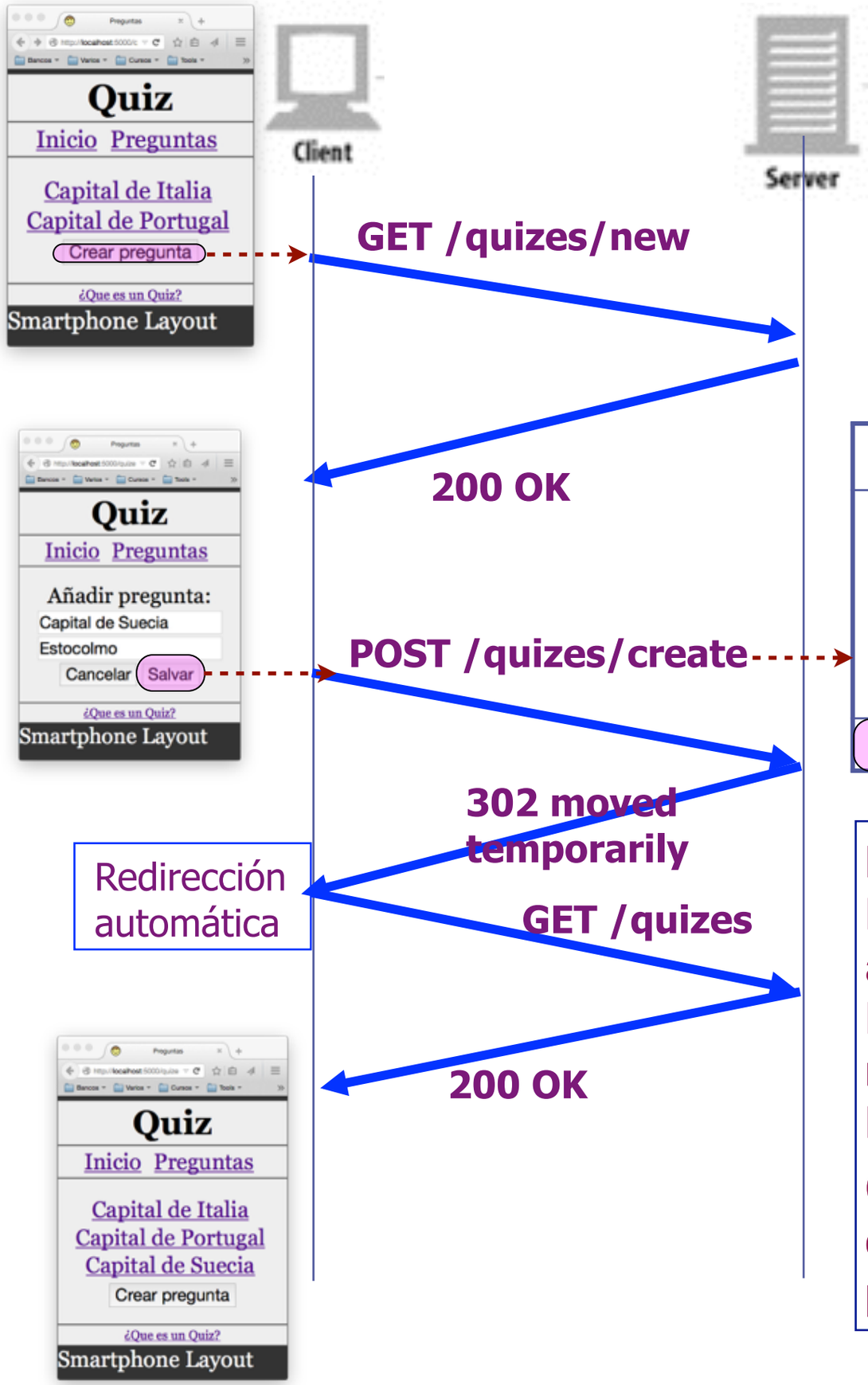
Paso 1: Nueva vista



Paso 2: POST /quizes



HTTP: Creación de nuevo quiz



POST /quizes/create HTTP/1.1

Host: localhost
Accept: text/*, image/*
.....
Content-type: application/x-www-form-urlencoded
Content-length: 17

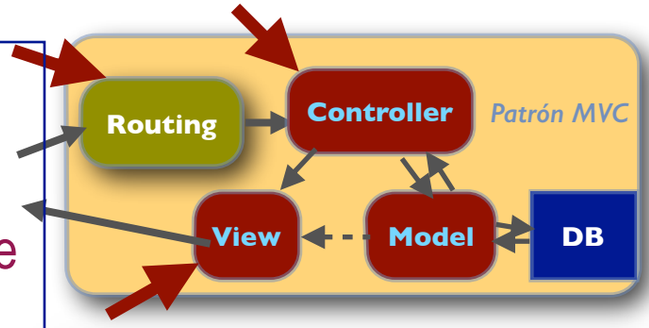
quiz[pregunta]=Capital+de+Suecia&quiz[respuesta]=Estocolmo

La primitiva **HTTP POST** solo se puede invocar en HTML desde un formulario, por lo que primero se accede al formulario de creación de quizes, para invocar HTTP POST después desde el formulario, una vez que se han rellenado los campos del formulario. La transacción **HTTP POST** se envía al pulsar el botón de enviar.

Como la transacción POST solo crea un recurso (quiz) en la base de datos, al acabar redirecciona a la lista de preguntas donde ya aparecerá la pregunta creada.

Paso 1 - GET /quizes/new: Añade el formulario de creación de nuevas preguntas.

El método **build(..)** de sequelize (http://sequelize.readthedocs.org/en/latest/docs/instances/) creo un objeto no persistente asociado a la tabla Quiz, con las propiedades inicializadas. Este objeto se utiliza aquí solo para renderizar las vistas.



Paso 1

index.js — UNREGISTERED

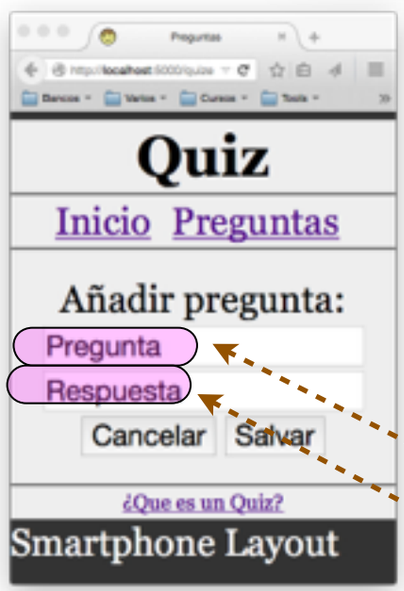
Paso 1b: ruta /quizes/new

```
// Definición de rutas de /quizes
router.get('/quizes',                          quizController.index);
router.get('/quizes/:quizId(\\d+)',            quizController.show);
router.get('/quizes/:quizId(\\d+)/answer',     quizController.answer);
router.get('/quizes/new',                      quizController.new);
router.post('/quizes/create',                  quizController.create);
```

quiz_controller.js — UNREGISTERED

Paso 1a: controlador de **new**

```
// GET /quizes/new
exports.new = function(req, res) {
  var quiz = models.Quiz.build( // crea objeto quiz
    {pregunta: "Pregunta", respuesta: "Respuesta"}
  );

  res.render('quizes/new', {quiz: quiz});
};
```



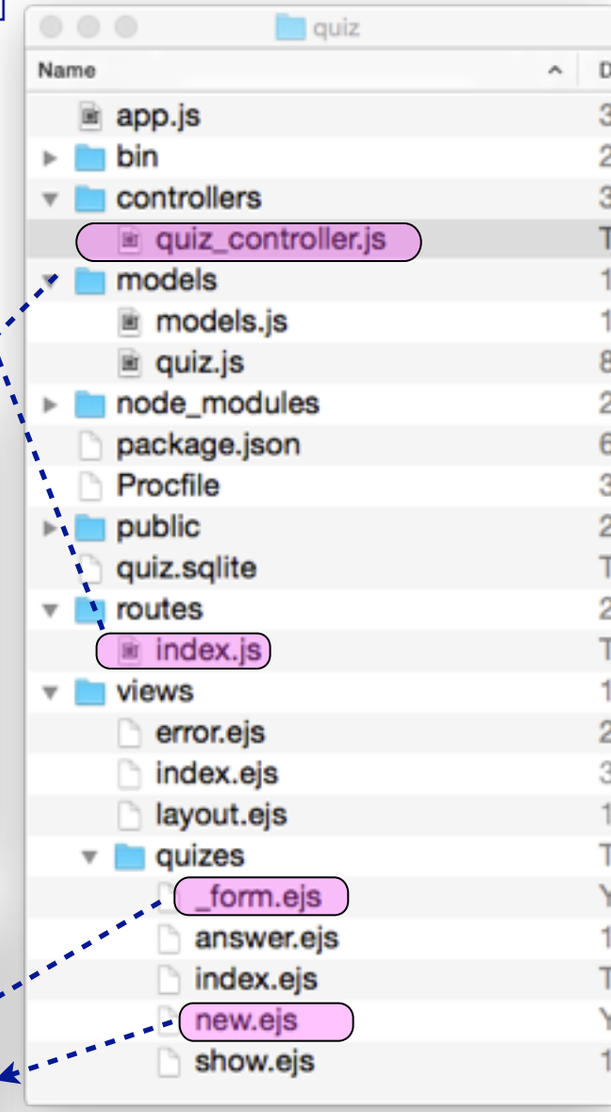
Paso 1c: vista views/quizes/new.ejs

```
1 Añadir pregunta: <p>
2
3 <form method="post" action="/quizes/create" >
4   <% include _form.ejs %>
5 </form>
```

Paso 1c: vista views/quiz/answer.ejs

_form.ejs — UNREGISTERED

```
1 <label for="preg" class="rem">Pregunta:</label>
2 <input type="text" id="preg" name="quiz[pregunta]" value="<%= quiz.pregunta %>"/> <p>
3 <label for="resp" class="rem">Respuesta:</label>
4 <input type="text" id="resp" name="quiz[respuesta]" value="<%= quiz.respuesta %>"/> <p>
5 <a href="/quizes"><button type="button">Cancelar</button></a>
6 <input type="submit" value="Salvar">
```



Paso 2

Paso 2b: ruta /quizes/create

index.js — UNREGISTERED

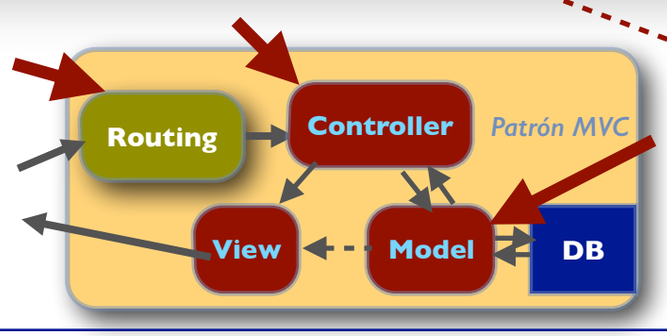
```
// Definición de rutas de /quizes
router.get('/quizes',                          quizController.index);
router.get('/quizes/:quizId(\\d+)',            quizController.show);
router.get('/quizes/:quizId(\\d+)/answer',     quizController.answer);
router.get('/quizes/new',                      quizController.new);
router.post('/quizes/create',                  quizController.create);
```

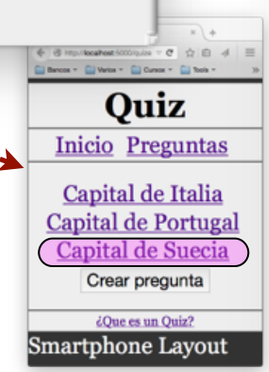
Paso 2a: controlador de **create**

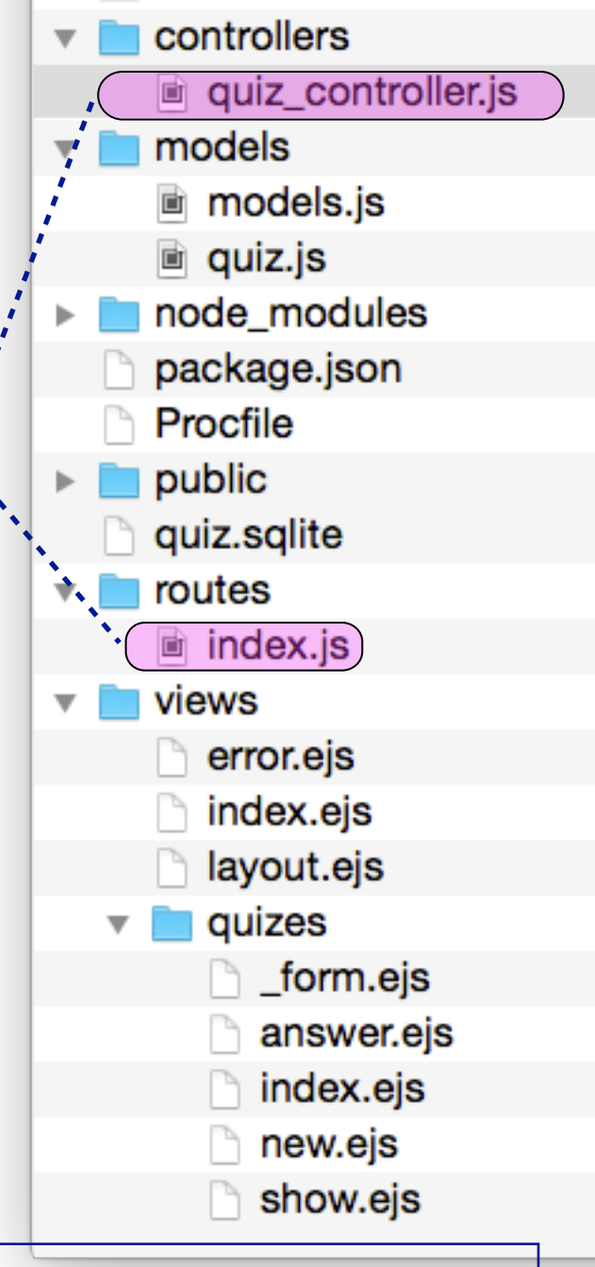
quiz_controller.js — UNREGISTERED

```
// POST /quizes/create
exports.create = function(req, res) {
  var quiz = models.Quiz.build( req.body.quiz );

// guarda en DB los campos pregunta y respuesta de quiz
  quiz.save({fields: ["pregunta", "respuesta"]}).then(function(){
    res.redirect('/quizes');
  })   // Redirección HTTP (URL relativo) lista de preguntas
};
```

id	pregunta	respuesta
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	Capital de Suecia	Estocolmo
4	.....	.....







Paso 2 - POST /quizes/create: añade la primitiva que introduce nuevos quizes en la DB.

El controlador **create** de **POST /quizes/create** genera el objeto **quiz** con **models.Quiz.build(req.body.quiz),** incializandolo con los parámetros enviados desde el formulario, que están accesibles en **req.body.quiz.**

quiz.save({fields: ["pregunta", "respuesta"]}) almacena el objeto no persistente **quiz** (solo las propiedades **"pregunta"** y **"respuesta")** en la **tabla Quiz de la DB**. Como este objeto se ha creado nuevo con build, se crea una nueva entrada (fila) en la tabla Quiz. Mas información en: http://sequelize.readthedocs.org/en/latest/docs/instances/

Una primitiva HTTP **POST /quizes/create** no tiene vista asociada. Al acabar realiza una **redirección HTTP** a la lista de preguntas invocando el método **res.redirect('/quizes')** de express (http://expressjs.com/4x/api.html#res.redirect)

Paso 3

```
22  22   app.use(bodyParser.json());
23     - app.use(bodyParser.urlencoded({ extended: false }));
    23 + app.use(bodyParser.urlencoded());
24  24   app.use(cookieParser());
```

Paso 3

quiz_controller.js — UNREGISTERED

```
// POST /quizes/create
exports.create = function(req, res) {
  var quiz = models.Quiz.build( req.body.quiz );

// guarda en DB los campos pregunta y respuesta de quiz
  quiz.save({fields: ["pregunta", "respuesta"]}).then(function(){
    res.redirect('/quizes');
  })   // Redirección HTTP (URL relativo) lista de preguntas
};
```

_form.ejs — UNREGIS

```
1 <label for="preg" class="rem">Pregunta:</label>
2 <input type="text" id="preg" name="quiz[pregunta]" value="<%= quiz.pregunta %>"/> <p>
3 <label for="resp" class="rem">Respuesta:</label>
4 <input type="text" id="resp" name="quiz[respuesta]" value="<%= quiz.respuesta %>"/> <
5 <a href="/quizes"><button type="button">Cancelar</button></a>
6 <input type="submit" value="Salvar">
```

Name

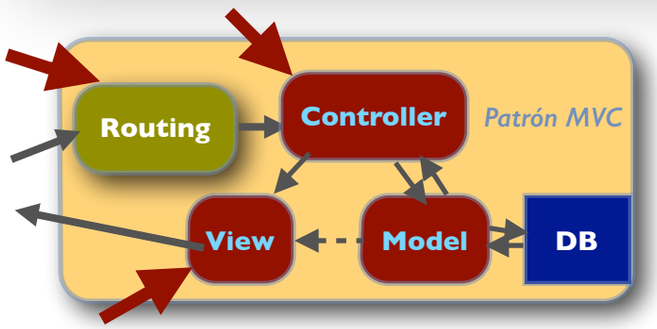
- app.js
- bin
- controllers
  - quiz_controller.js
- models
  - models.js
  - quiz.js
- node_modules
- package.json
- Procfile
- public
- quiz.sqlite
- routes
  - index.js
- views
  - error.ejs
  - index.ejs
  - layout.ejs
  - quizes
    - _form.ejs
    - answer.ejs
    - index.ejs
    - new.ejs
    - show.ejs

Los nombres de los parámetros del formulario

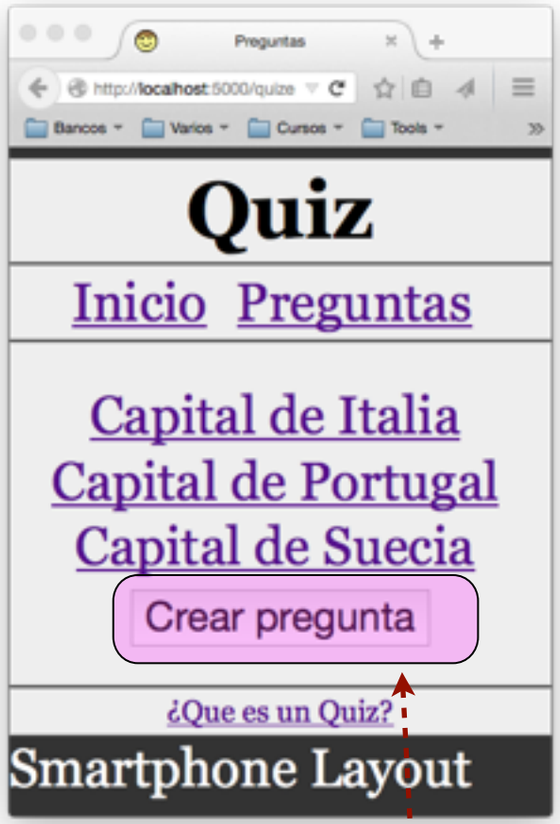
- name="quiz[pregunta]"
- name="quiz[respuesta]"

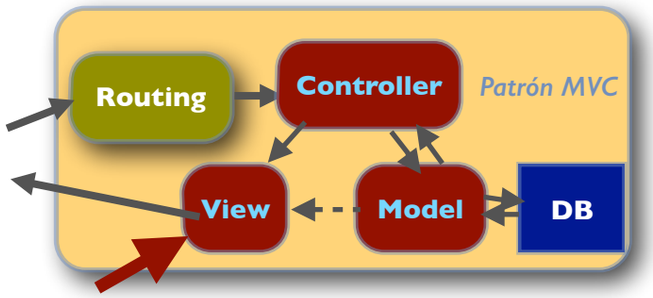
utilizan notación pseudo JSON que permite indicar que son propiedades de un objeto **quiz**. El middleware **bodyparser.urlencoded(..)** los analiza correctamente y genera el objeto **req.body.quiz**, si quitamos el parámetro de configuración {extended: false} que express-generator incluyo cuando generó el proyecto.

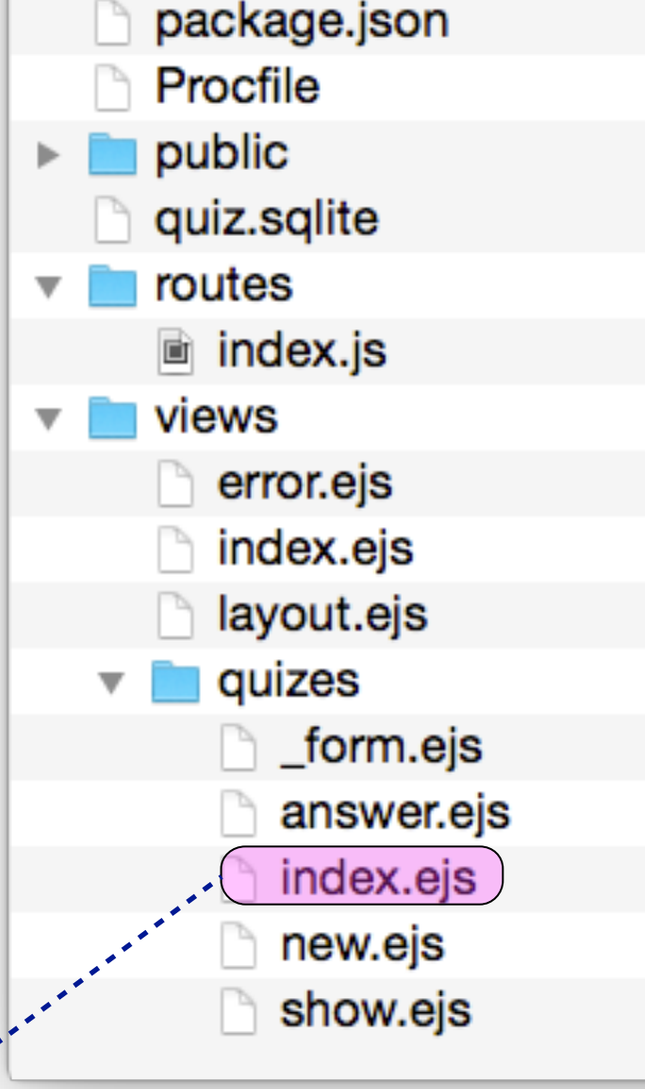
Mas info: https://github.com/expressjs/body-parser/blob/master/README.md



Paso 4: enlace a crear pregunta







index.ejs UNREGISTERED

```
1  <table>
2  <% var i; for (i=0; i < quizes.length; i++) { %>
3     <tr><td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td></tr>
4  <% } %>
5  </table><p>
6  <a href="/quizes/new"><button type="button">Crear pregunta</button></a>
```

views/quizes/index.js

Paso 4

Final del tema

Quiz 12: Validación de entradas

Juan Quemada, DIT - UPM

Quiz 12: Validación de entradas

Objetivo: Introducir en la aplicación Quiz comprobaciones de las preguntas/respuestas introducidas en el formulario para que ninguno de los dos campos sean un string vacía.

✦**Paso 1**: Definir validaciones en modeles/quiz.js

✦**Paso 2**: Validar entradas con **función validate()** en la acción **create** de **quiz_controller.js**

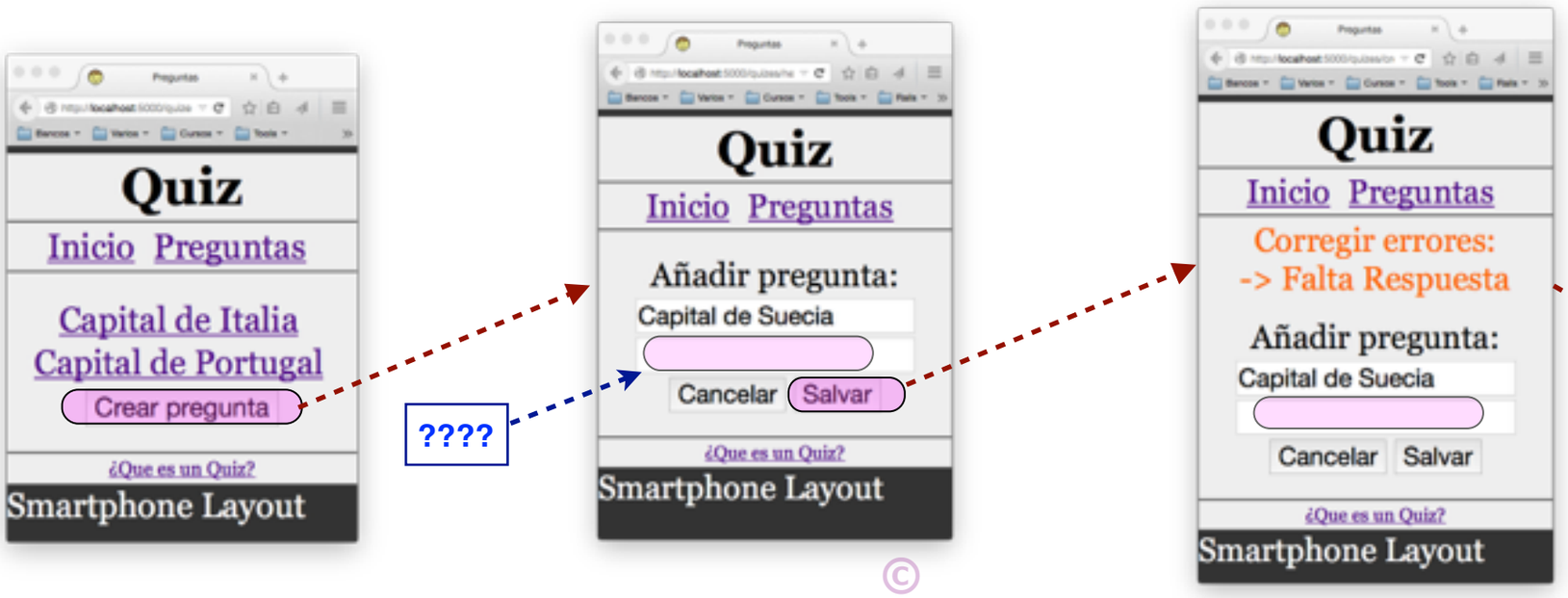
✦**Paso 3**: Añadir presentación de errores en vista **layout.ejs**

✦**Paso 4**: Aplicar color rojo a mensajes de error en **style.css**

✦**Paso 5**: Parámetro con errores vacíos ([]) en res.render(…) en todos los controladores

- a: Todos los controladores de **quiz_controller.js**, salvo **create**
- b: Filtro de página home en **routes/index**
- c: En res.render(…) de middlewares de error en **app.js**

✦**Paso 6**: Guardar **versión (commit)** git y subir a **Heroku**



id	pregunta	respuesta
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	Capital de Suecia	
4	……	……
5	……	……

©

Pasos 1 y 2

Sequelize incluye muchas funciones de validación ya definidas y además se pueden definir otras según necesite una aplicación. Ver: http://sequelize.readthedocs.org/en/latest/docs/models/#validations)

quiz.js UNREGISTERED

```
1  // Definicion del modelo de Quiz con validación
2
3  module.exports = function(sequelize, DataTypes) {
4    return sequelize.define(
5      'Quiz',
6      { pregunta: {
7          type: DataTypes.STRING,
8          validate: { notEmpty: {msg: "-> Falta Pregunta"}}
9        },
10       respuesta: {
11         type: DataTypes.STRING,
12         validate: { notEmpty: {msg: "-> Falta Respuesta"}}
13       }
14     }
15   );
16 }
```

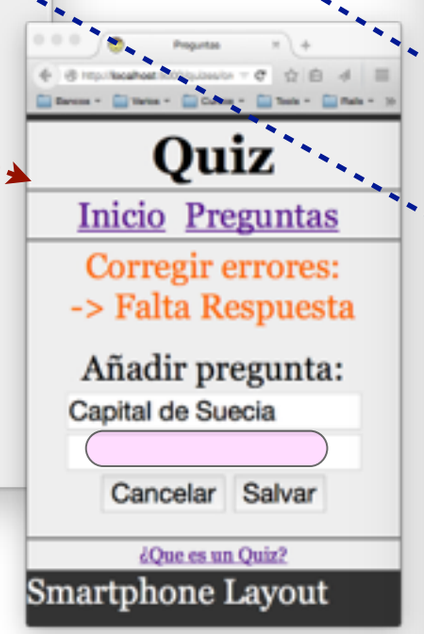
Paso 1: models/quiz.js

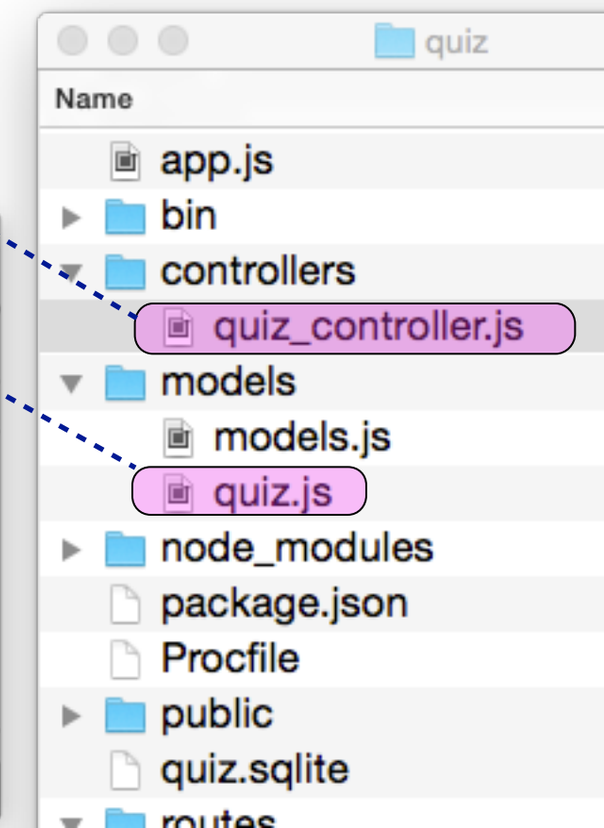
Sequelize.js permite añadir al modelo de una tabla restricciones en al contenido de los campos, de forma que se pueda comprobar con la **función validate()** de sequelize, si un objeto JavaScript de la clase asociada a los elementos de la tabla cumple las restricciones o no.

La restricción incluye el mensaje que deberá generarse en caso de no cumplirse. Los errores de validación están en la propiedad errores de **ValidationError**: http://sequelize.readthedocs.org/en/latest/api/errors/

```
53 // POST /quizes/create
54 exports.create = function(req, res) {
55   var quiz = models.Quiz.build( req.body.quiz );
56
57   quiz
58   .validate()
59   .then(
60     function(err){
61       if (err) {
62         res.render('quizes/new', {quiz: quiz, errors: err.errors});
63       } else {
64         quiz // save: guarda en DB campos pregunta y respuesta de quiz
65         .save({fields: ["pregunta", "respuesta"]})
66         .then( function(){ res.redirect('/quizes')})
67       }      // res.redirect: Redirección HTTP a lista de preguntas
68     }
69   );
70 };
```

Paso 2: controllers/quizController.js





Funciones de validación definidas en:
http://sequelize.readthedocs.org/en/latest/docs/models/#validations)

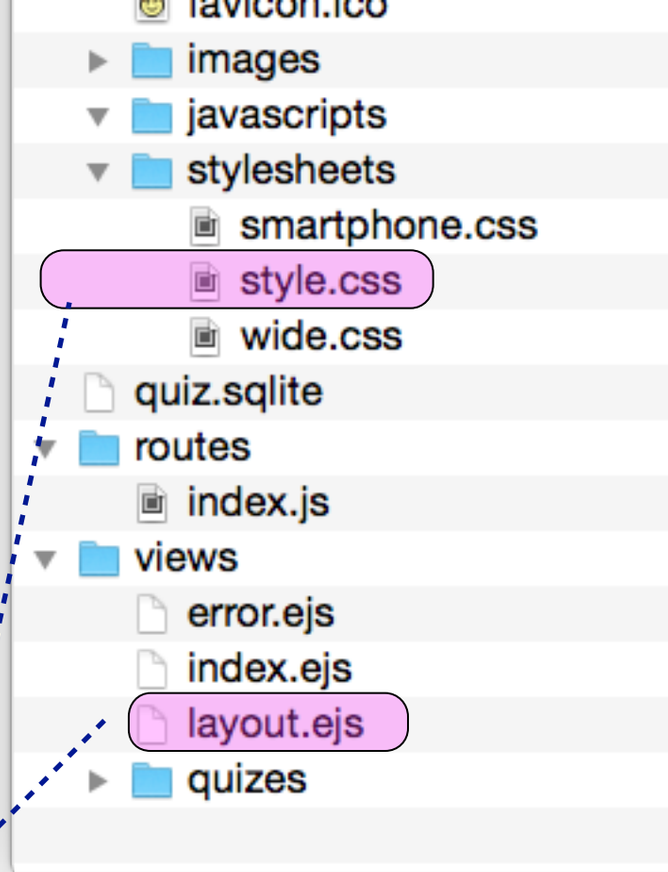
```
var ValidateMe = sequelize.define('Foo', {
  foo: {
    type: Sequelize.STRING,
    validate: {
      is: ["^[a-z]+$",'i'],       // will only allow letters
      is: /^[a-z]+$/i,            // same as the previous example using real RegExp
      not: ["[a-z]",'i'],         // will not allow letters
      isEmail: true,              // checks for email format (foo@bar.com)
      isUrl: true,                // checks for url format (http://foo.com)
      isIP: true,                 // checks for IPv4 (129.89.23.1) or IPv6 format
      isIPv4: true,               // checks for IPv4 (129.89.23.1)
      isIPv6: true,               // checks for IPv6 format
      isAlpha: true,              // will only allow letters
      isAlphanumeric: true        // will only allow alphanumeric characters, so "_abc" will fail
      isNumeric: true             // will only allow numbers
      isInt: true,                // checks for valid integers
      isFloat: true,              // checks for valid floating point numbers
      isDecimal: true,            // checks for any numbers
      isLowercase: true,          // checks for lowercase
      isUppercase: true,          // checks for uppercase
      notNull: true,              // won't allow null
      isNull: true,               // only allows null
      notEmpty: true,             // don't allow empty strings
      equals: 'specific value',   // only allow a specific value
      contains: 'foo',            // force specific substrings
      notIn: [['foo', 'bar']],    // check the value is not one of these
      isIn: [['foo', 'bar']],     // check the value is one of these
      notContains: 'bar',         // don't allow specific substrings
      len: [2,10],                // only allow values with length between 2 and 10
      isUUID: 4,                  // only allow uuids
      isDate: true,               // only allow date strings
      isAfter: "2011-11-05",      // only allow date strings after a specific date
      isBefore: "2011-11-05",     // only allow date strings before a specific date
      max: 23,                    // only allow values
      min: 23,                    // only allow values >= 23
      isArray: true,              // only allow arrays
      isCreditCard: true,         // check for valid credit card numbers
```

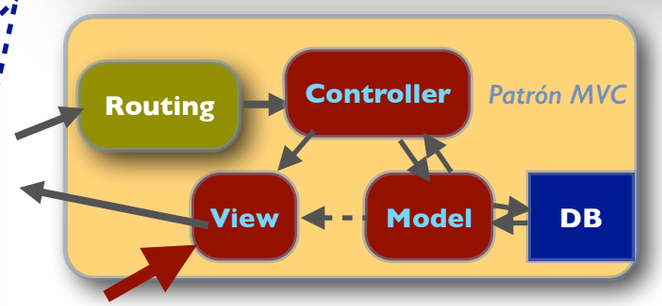
Validaciones soportados en sequelize.js

Pasos 3 y 4: Mostrar errores

layout.ejs UNREGISTERED

```
17  <body>
18  <div id="page-wrap">
19    <header class="main" id="h1">
20      <h2>Quiz<span>: el juego de las preguntas</span></h2>
21    </header>
22
23    <nav class="main" id="n1" role="navigation">
24      <span><a href="/">Inicio</a></span>
25      <span><a href="/quizes">Preguntas</a></span>
26    </nav>
27
28    <section class="main" id="s1">
29      <% if (errors.length) { %>
30        <span id='ErrorMsgs'>
31          Corregir errores:<br/>
32          <% for (var i in errors) { %>
33            <span> <%= errors[i].message %> </span><br/>
34          <% } %>
35        </span>
36      <% } %>
37
38      <div><%- body %></div>
39    </section>
40
41    <footer class="main" id="f1">
42      <a href="http://es.wikipedia.org/wiki/Quiz">
43        ¿Que es un Quiz?</a>
44    </footer>
45  </div>
46  </body>
47  </html>
```





views/layout.ejs

Los errores se pasan en el array **"errors: [mensajes de error]"** y se muestran en la parte alta de **<section>**. Se añaden directivas de stilo en **style.css** para que los errores se muestran en color rojizo.

```
......
nav span            { margin: 0px 10px 0px 0px;}

#ErrorMsgs          { color: #f61; font-size: smaler}
```

Paso 5: errors: [] en render(..)

```
// Autoload :id
exports.load = function(req, res, next, quizId) {
  models.Quiz.find(quizId).then(
    function(quiz) {
      if (quiz) {
        req.quiz = quiz;
        next();
      } else{next(new Error('No existe quizId=' + quizId))}
    }
  ).catch(function(error){next(error)});
};

// GET /quizes
exports.index = function(req, res) {
  models.Quiz.findAll().then(
    function(quizes) {
      res.render('quizes/index.ejs', {quizes: quizes, errors: []});
    }
  ).catch(function(error){next(error)});
};

// GET /quizes/:id
exports.show = function(req, res) {
  res.render('quizes/show', { quiz: req.quiz, errors: []});
};

// GET /quizes/:id/answer
exports.answer = function(req, res) {
  var resultado = 'Incorrecto';
  if (req.query.respuesta === req.quiz.respuesta) {
    resultado = 'Correcto';
  }
  res.render(
    'quizes/answer',
    { quiz: req.quiz,
      respuesta: resultado,
      errors: []
    }
  );
};

// GET /quizes/new
exports.new = function(req, res) {
  var quiz = models.Quiz.build( // crea objeto quiz
    {pregunta: "Pregunta", respuesta: "Respuesta"}
  );

  res.render('quizes/new', {quiz: quiz, errors: []});
};

// POST /quizes/create
exports.create = function(req, res) {
  var quiz = models.Quiz.build( req.body.quiz );
```

controllers/quizController.js

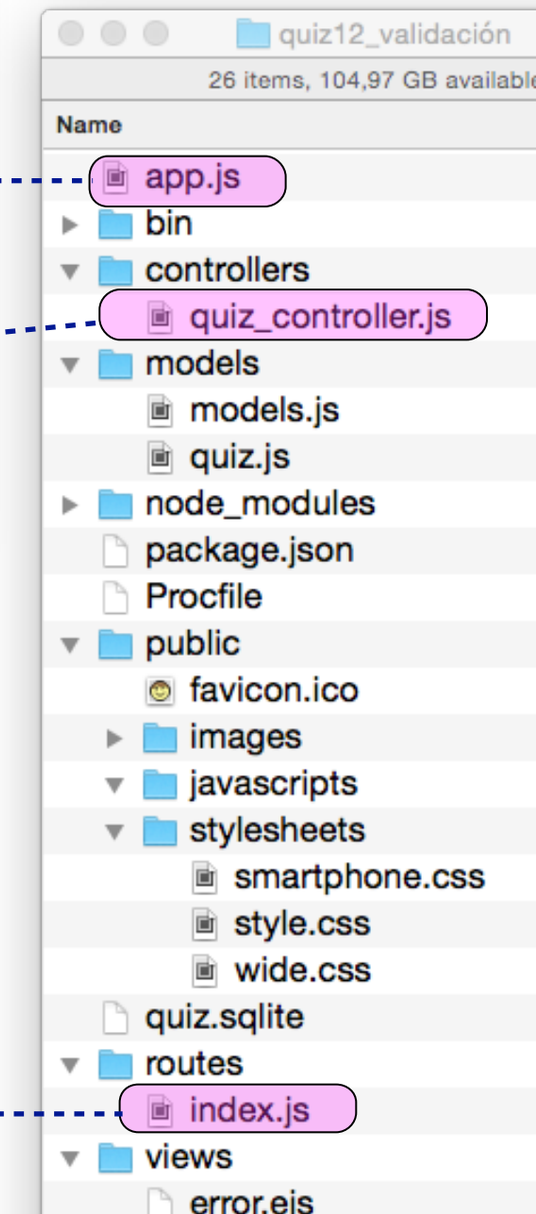
```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
});

// 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,
            errors: []
        });
    });
}

// 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: {},
        errors: []
    });
});

module.exports = app;
```

app.js

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

var quizController = require('../controllers/quiz_controller');

// Página de entrada (home page)
router.get('/', function(req, res) {
  res.render('index', { title: 'Quiz', errors: []});
});

// Autoload de comandos con :quizId
router.param('quizId', quizController.load);  // autoload :quizId
```

routes/index.js

La vista **layout.ejs** espera la variable **errors**. Hay que añadir el parámetro **errors:[]** al método res.render(..) para que renderice correctamente. Al pasar un array vacío no se muestran mensajes de error.



Final del tema

©

Quiz 13: Editar preguntas

Juan Quemada, DIT - UPM

Quiz 13: editar preguntas

Objetivo: Introducir en la aplicación Quiz un formulario que permita editar preguntas existentes, que permita cambiar o corregir su contenido.

✦**Paso 1**: Añadir formulario de editar preguntas en **GET /quizes/:quizId/edit**

- a: Añadir acción **edit** en **quiz_controller.js**
- b: Añadir en **routes/index.js** la ruta **GET /quizes/:quizId/edit**
- c: Añadir vista con formulario de editar pregunta: **views/quizes/edit.ejs**

✦**Paso 2**: Añadir MW **methodoverride()** para transformar **POST** en **PUT**

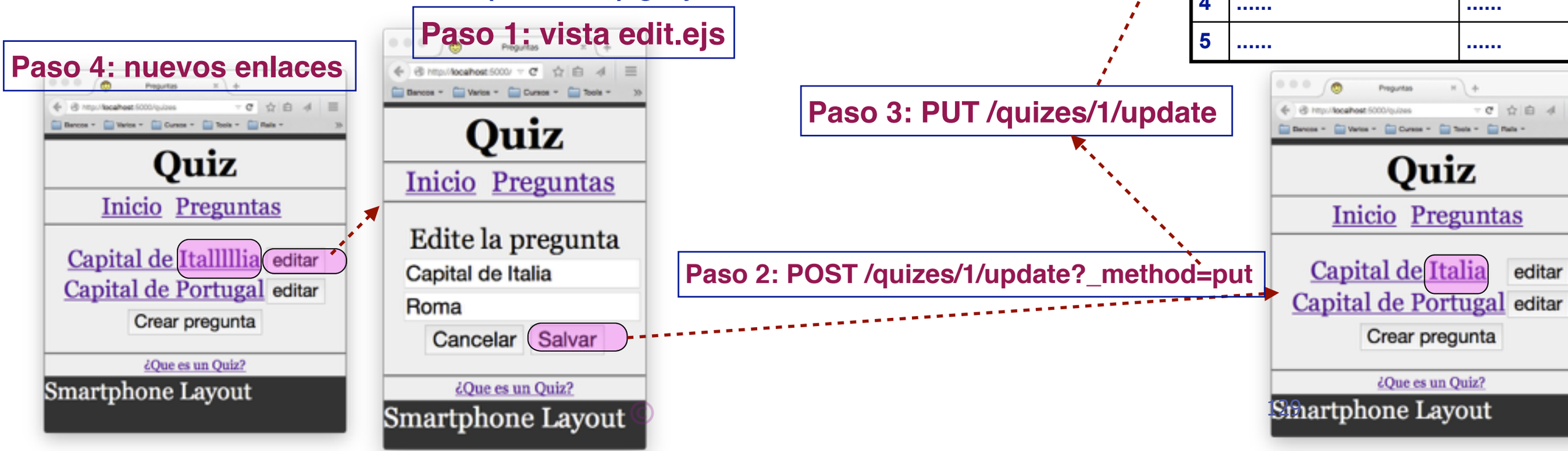
✦**Paso 3**: Añadir **PUT /quizes/:quizId** para modificar preguntas

- a: Añadir acción **update** a **quiz_controller.js** asociado a **/quizes/:quizId**
- b: Añadir ruta **POST /quizes/update** en **routes/index.js**

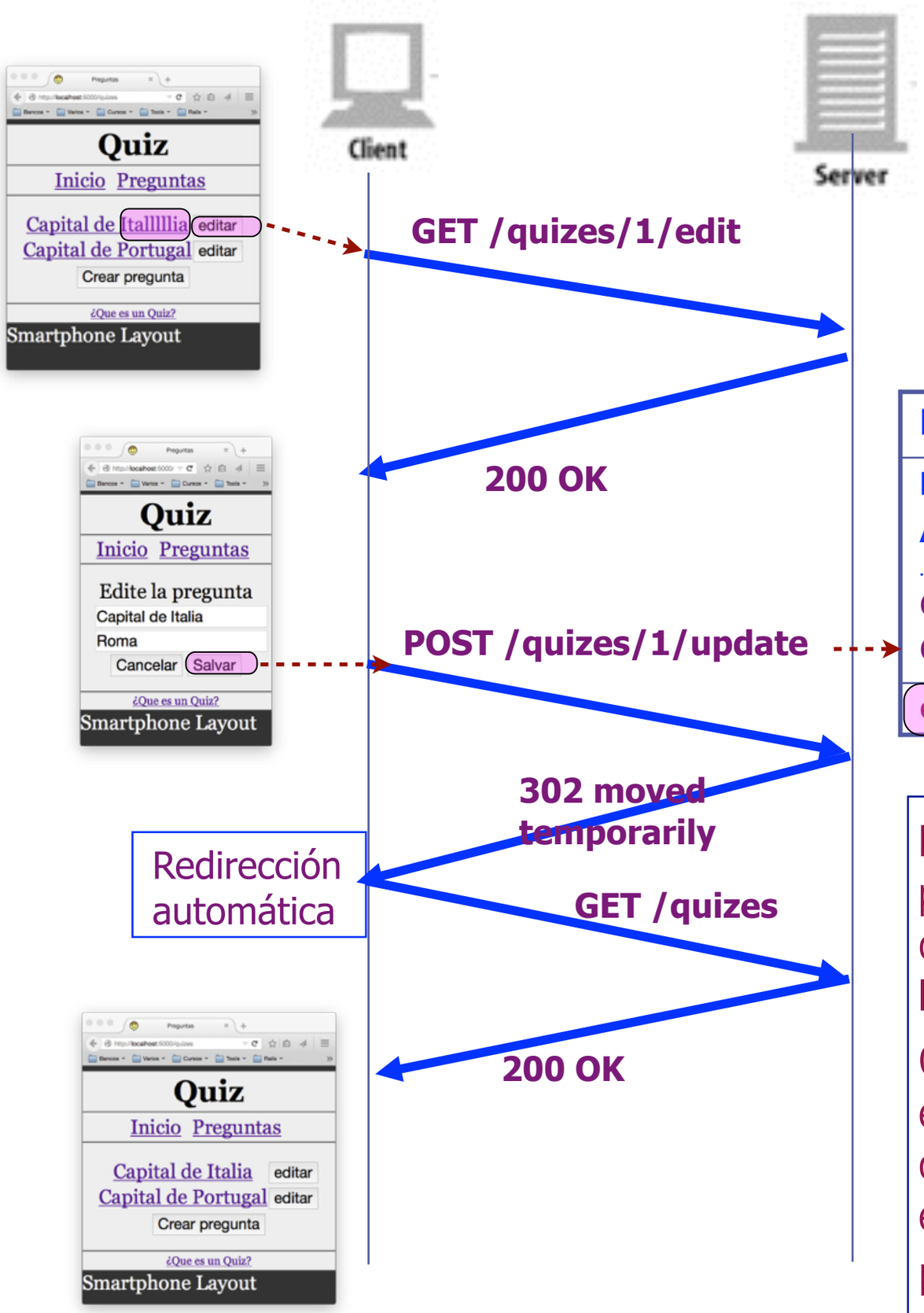
✦**Paso 4**: Añadir en vista **index.ejs** enlaces de edición a cada pregunta

✦**Paso 5**: Guardar **versión (commit)** git y subir a **Heroku**

id	pregunta	respuesta
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....
5	.....	.....



HTTP: Edición de un quiz




```
POST /quizes/1?_method=put HTTP/1.1
Host: localhost
Accept: text/*, image/*
.......
Content-type: application/x-www-form-urlencoded
Content-length: 17
quiz[pregunta]=Capital+de+Italia&quiz[respuesta]=Roma
```

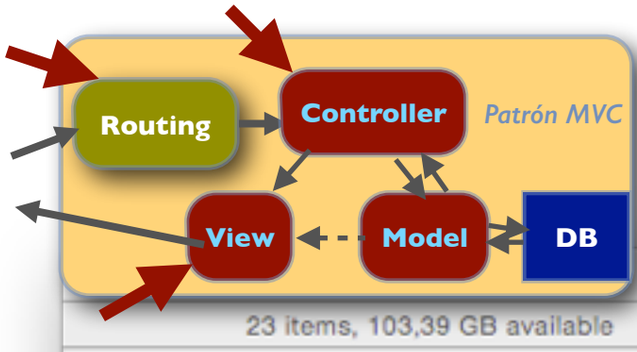
La primitiva **HTTP POST** para actualizar la DB con la pregunta corregida se invoca, igual que en la creación, con un formulario HTML. El formulario permite re-editar las preguntas e invocar HTTP POST para actualizar la DB.

Como las actualizaciones de un recurso deben realizarse en REST con el método PUT de HTTP, se utiliza el convenio conocido como **methodoverride** para encapsularlo como un parámetro oculto en el path.

Esta transacción (PUT) también redirecciona a la lista de preguntas donde ya aparecerá la pregunta creada.

GET /quizes/:quizId/edit accede al formulario de edición de preguntas.

El identificador de la pregunta en la tabla se extrae con (/**:quizId(\\d+)**/) para reconocer solo números. Como esta ruta incluye **:quizId**, autoload realiza la búsqueda en la tabla antes de invocar el controlador, dejando el objeto en **req.quiz**, para poder inicializar el formulario con la pregunta y respuesta guardada en la tabla.



23 items, 103,39 GB available

Name

- app.js
- bin
- controllers
  - quiz_controller.js
- models
- node_modules
- package.json
- Procfile
- public
- quiz.sqlite
- routes
  - index.js
- views
  - error.ejs
  - index.ejs
  - layout.ejs
  - quizes
    - _form.ejs
    - answer.ejs
    - edit.ejs
    - index.ejs
    - new.ejs
    - show.ejs

index.js — UNREGISTERED

Paso 1b: GET /quizes/:id/edit

```
// Definición de rutas de /quizes
router.get('/quizes',                          quizController.index);
router.get('/quizes/:quizId(\\d+)',            quizController.show);
router.get('/quizes/:quizId(\\d+)/answer',     quizController.answer);
router.get('/quizes/new',                      quizController.new);
router.post('/quizes/create',                  quizController.create);
router.get('/quizes/:quizId(\\d+)/edit',       quizController.edit);
router.put('/quizes/:quizId(\\d+)',            quizController.update);
```

quiz_controller.js — UNREGISTERED

Paso 1a: controlador **edit**

```
// GET /quizes/:id/edit
exports.edit = function(req, res) {
  var quiz = req.quiz; // autoload de instancia de quiz

  res.render('quizes/edit', {quiz: quiz, errors: []});
};
```

Quiz

Inicio Preguntas

Edite la pregunta

Capital de Italia

Roma

Cancelar Salvar

¿Que es un Quiz?

Smartphone Layout

edit.ejs — UNREGISTERED

Paso 1c: vista views/quizes/edit.ejs

```
Edite la pregunta <p>

<form method="post" action="/quizes/<%= quiz.id %>?_method=put" >

  <% include _form.ejs %>
</form>
```

_form.ejs — UNREGISTERED

```
<label for="preg" class="rem">Pregunta:</label>
<input type="text" id="preg" name="quiz[pregunta]" value="<%= quiz.pregunta %>"/> <p>
<label for="resp" class="rem">Respuesta:</label>
<input type="text" id="resp" name="quiz[respuesta]" value="<%= quiz.respuesta %>"/> <p>
<a href="/quizes"><button type="button">Cancelar</button></a>
<input type="submit" value="Salvar">
```

Paso 1c: vista views/quiz/_form.ejs

Paso 1

Paso 2: method override

app.js UNREGISTERED

app.js

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

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(partials());
// uncomment after placing your favicon in /public
// app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use(cookieParser());
app.use(methodOverride('_method'));
app.use(express.static(path.join(__dirname, 'public')));

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

```
  },
  "dependencies": {
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "debug": "~2.0.0",
    "ejs": "~0.8.5",
    "express": "~4.9.0",
    "express-partials": "^0.3.0",
    "method-override": "^2.3.1",
    "morgan": "~1.3.0",
    "pg": "^4.1.1",
    "sequelize": "^2.0.0-rc4",
    "serve-favicon": "~2.1.3"
  }
}
```

Se instala el MW **method-overrride**, tal y como hemos instalado otros MWs con

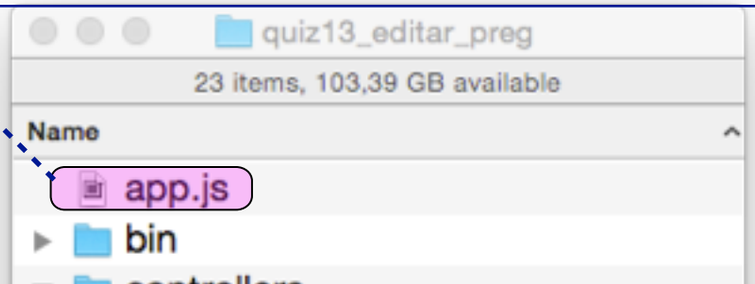
$ npm install --save method-override

para que se registre la instalación en **package.json**.

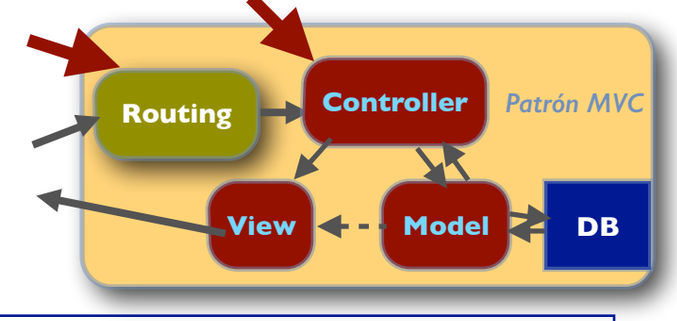
Importamos en app.js el paquete con **require('method-override'),** instalando el midlleware en nuestra aplicación con **app.use(methodOverride('_method'))**.

El parámetro '**_method**' indica el nombre utilizado para encapsular el método.

Información sobre MW method-overrride: **https://github.com/expressjs/method-override**



Paso 3: actualizar DB



PUT /quizes/:quizId actualiza la DB con la pregunta corregida. Como la ruta incluye el identificador (**:quizId**) de la pregunta en la tabla, autoload pre-carga el objeto en **req.quiz** antes de invocar el controlador **update**. El controlador actualiza las propiedades **pregunta** y **respuesta** de **req.quiz** con lo enviado desde el formulario, actualiza la DB y redirecciona a la lista de preguntas con la pregunta ya corregida.

index.js UNREGISTERED

Paso 2b: PUT /quizes/:quizId(\\d+)

```
14  // Definición de rutas de /quizes
15  router.get('/quizes',                        quizController.index);
16  router.get('/quizes/:quizId(\\d+)',          quizController.show);
17  router.get('/quizes/:quizId(\\d+)/answer',   quizController.answer);
18  router.get('/quizes/new',                    quizController.new);
19  router.post('/quizes/create',                quizController.create);
20  router.get('/quizes/:quizId(\\d+)/edit',     quizController.edit);
21  router.put('/quizes/:quizId(\\d+)',          quizController.update);
```

quiz_controller.js UNREGISTERED

Paso 1a: controlador update

```
79  // PUT /quizes/:id
80  exports.update = function(req, res) {
81    req.quiz.pregunta  = req.body.quiz.pregunta;
82    req.quiz.respuesta = req.body.quiz.respuesta;
83
84    req.quiz
85    .validate()
86    .then(
87      function(err){
88        if (err) {
89          res.render('quizes/edit', {quiz: req.quiz, errors: err.errors});
90        } else {
91          req.quiz     // save: guarda campos pregunta y respuesta en DB
92          .save( {fields: ["pregunta", "respuesta"]})
93          .then( function(){ res.redirect('/quizes');});
94        }     // Redirección HTTP a lista de preguntas (URL relativo)
95      }
96    );
97  };
```

id	pregunta	respuesta
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....

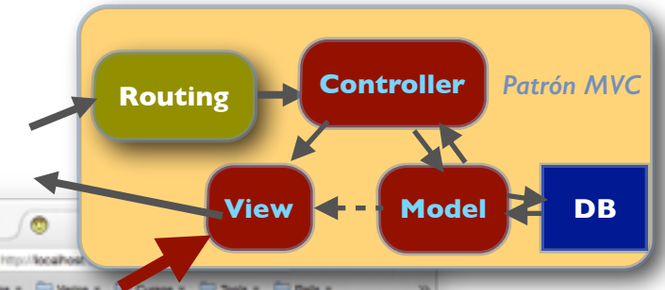
quiz13_editar_preg

23 items, 103,39 GB available

Name

- app.js
- bin
- controllers
  - quiz_controller.js
- models
- node_modules
- package.json
- Procfile
- public
- quiz.sqlite
- routes
  - index.js
- views
  - error.ejs

Paso 4: vista quizes/index.ejs



quiz_controller.js UNREGISTERED

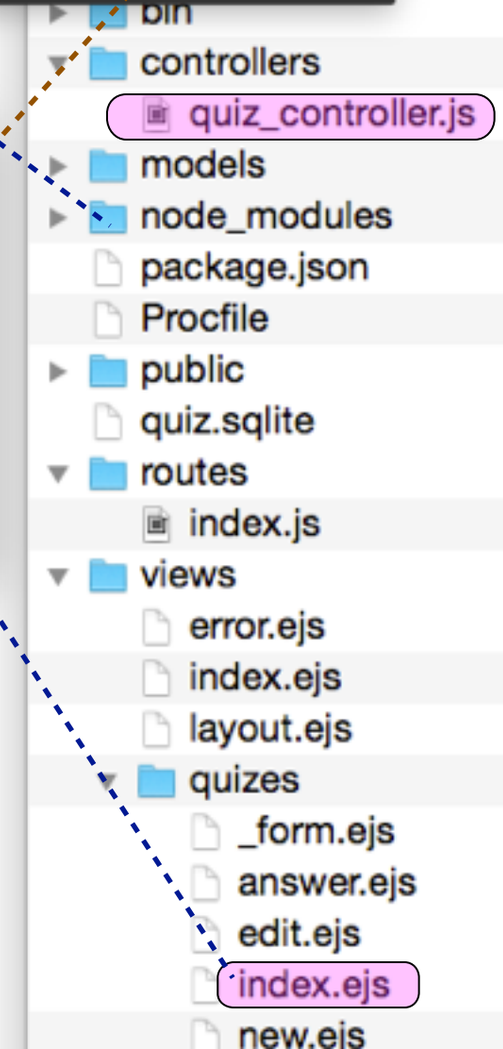
```
15 // GET /quizes
16 exports.index = function(req, res) {
17   models.Quiz.findAll().then(
18     function(quizes) {
19       res.render('quizes/index.ejs', {quizes: quizes, errors: []});
20     }
21   ).catch(function(error){next(error)});
22 };
```

index.ejs UNREGISTERED

```
1  <table>
2    <% var i; for (i=0; i < quizes.length; i++) { %>
3      <tr>
4        <td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td>
5        <td><a href="quizes/<%= quizes[i].id %>/edit"><button>editar</button></a></td>
6      </tr>
7    <% } %>
8  </table>
9  <p/>
10 <a href="/quizes/new"><button>Crear pregunta</button></a>
```

views/quizes/index.js

Paso 4



Se añade un **botón de editar** a cada pregunta en la vista de la lista de preguntas **quiz/index.ejs** para editar dicha pregunta.

El URL de acceso a la pregunta debe incluir el identificador :id de la pregunta para que al renderizar el formulario se inicialize con el contenido de la DB.

Final del tema

©

Quiz 14: Borrar preguntas

Juan Quemada, DIT - UPM

Quiz 14: borrar preguntas

Objetivo: Introducir en la lista de preguntas de la aplicación Quiz un botón que que permita borrar preguntas existentes

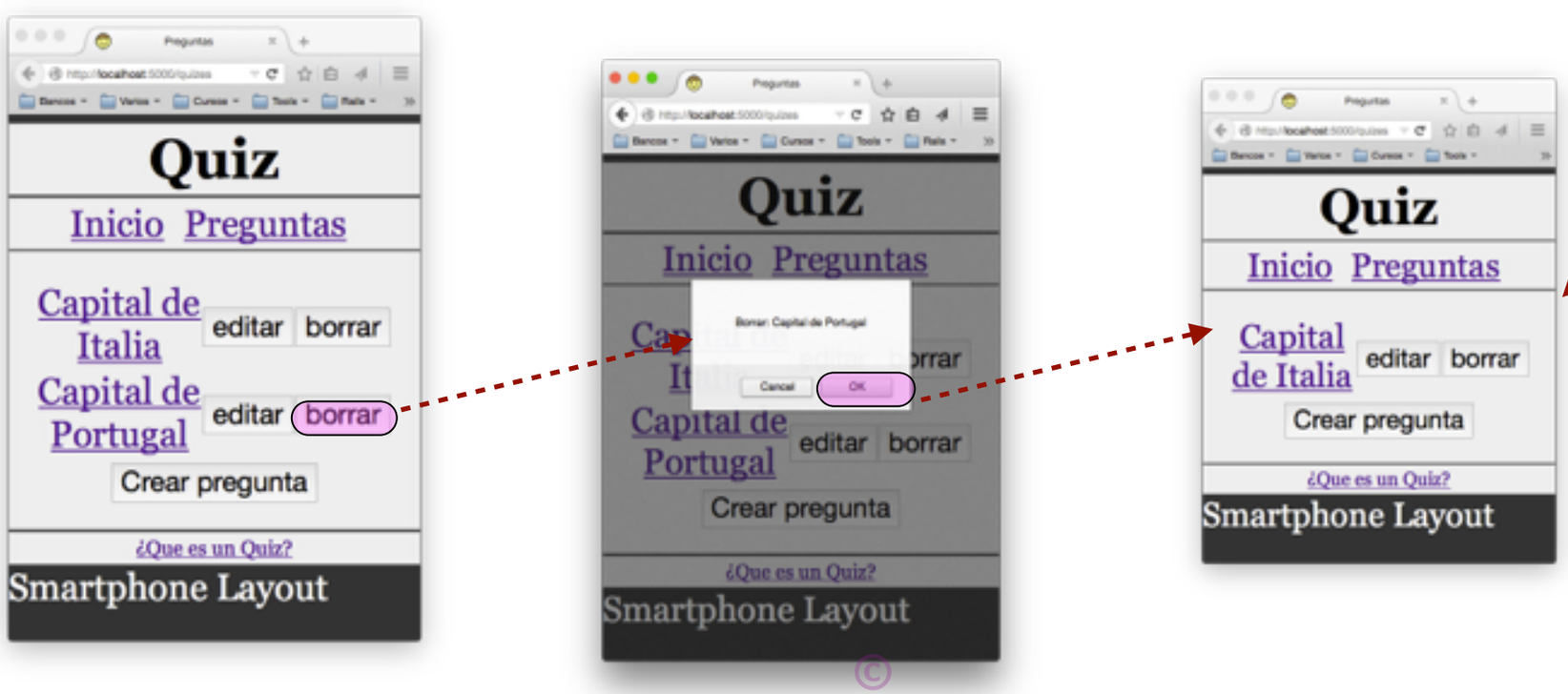
✦**Paso 1**: Añadir en vista **index.ejs** botones para borrar cada una de las preguntas

✦**Paso 2**: Añadir ruta **DELETE /quizes/:quizId** para borrar preguntas

- a: Añadir en **quiz_controller.js** la acción **destroy**
- b: Añadir filtro **DELETE /quizes/:quizID(\\d+)** en **routes/index.js**

✦**Paso 3**: Guardar **versión (commit)** git y subir a **Heroku**

id	pregunta	respuesta
1	**Capital de Italia**	Roma
2	.....	.....
3	.....	.....
4	.....	.....



Paso 1: vista quizes/index.ejs

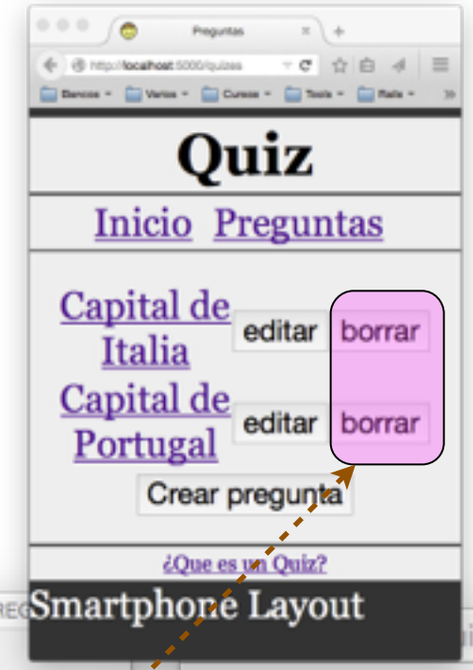
quiz_controller.js UNREGISTERED

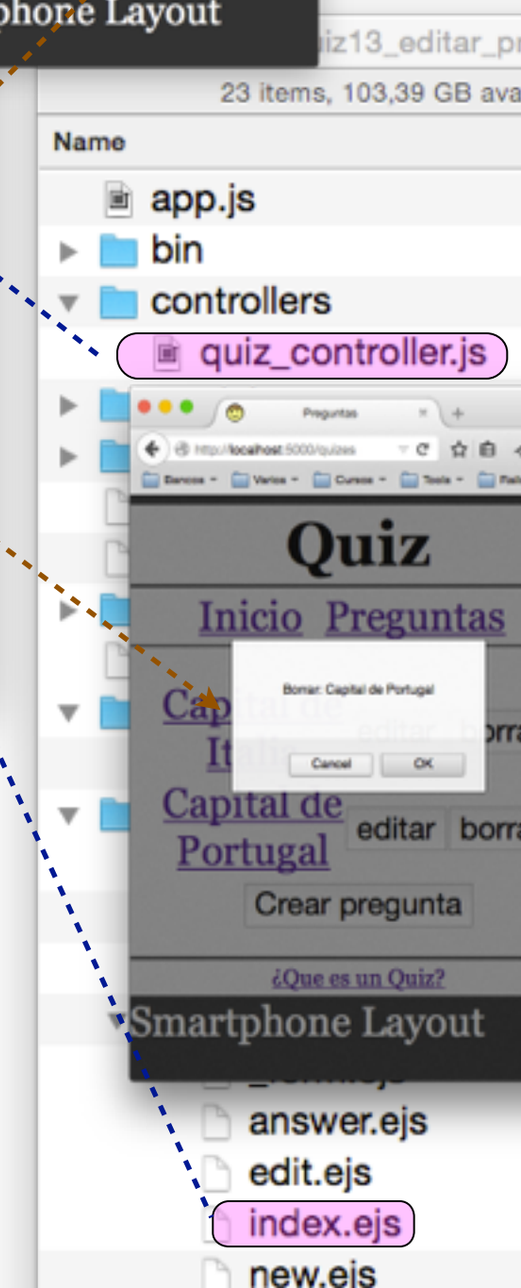
```
15  // GET /quizes
16  exports.index = function(req, res) {
17    models.Quiz.findAll().then(
18      function(quizes) {
19        res.render('quizes/index.ejs', {quizes: quizes, errors: []});
20      }
21    ).catch(function(error){next(error)});
22  };
```

index.ejs

```
1  <table>
2    <% var i; for (i=0; i < quizes.length; i++) { %>
3      <tr>
4        <td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td>
5        <td><a href="quizes/<%= quizes[i].id %>/edit"><button>editar</button></a></td>
6        <td>
7          <form method="post" action="quizes/<%= quizes[i].id %>?_method=delete">
8            <button type="submit" onClick="return confirm('Borrar: <%= quizes[i].pregunta %>');">
9              borrar
10           </button>
11         </form>
12       </td>
13     </tr>
14   <% } %>
15 </table>
16 <p/>
17 <a href="/quizes/new"><button>Crear pregunta</button></a>
```

views/quizes/index.js





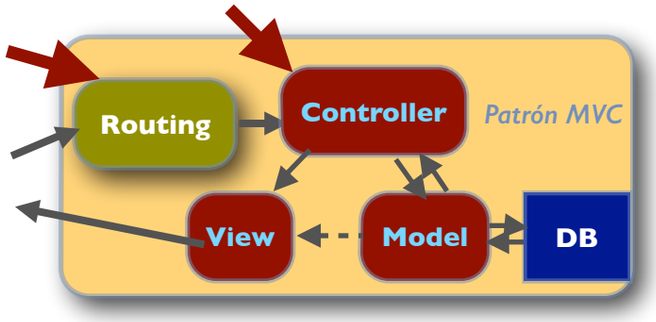
Se añade **botón de borrar** a cada pregunta en la lista de preguntas: **quiz/index.ejs**.

La operación **POST** de borrado de una pregunta se genera con un formulario que solo tiene un botón y que lleva el **:id** de la pregunta, además de **PUT encapsulado** (**con method override**) en el path de la solicitud HTTP.

El botón de submit del formulario lleva un manejador JavaScript que despliega un panel de confirmación antes de enviar la orden de borrado de la pregunta.

© Juan Quemada - UPM-DIT

Paso 2: borrar pregunta en DB



DELETE /quizes/:quizId borra la pregunta identificada por **:quizId** en la base de datos.

El controlador **destroy** se invoca al llegar **DELETE /quizes/:quizId.** Como la ruta incluye el identificador (**:quizId**) de la pregunta en la tabla, autoload pre-carga el objeto en **req.quiz** antes de invocar el controlador **update**.

El método **destroy()** ordena la destrucción de la entrada de la tabla identificada por **req.quiz.** Cuando dicha entrada se ha eliminado de la tabla se invoca el callback instalado con **then(..)** que redirecciona a la lista de preguntas.

Paso 2b: DELETE /quizes/:quizId(\\d+)

index.js UNREGISTERED

```
14 // Definición de rutas de /quizes
15 router.get('/quizes',                        quizController.index);
16 router.get('/quizes/:quizId(\\d+)',          quizController.show);
17 router.get('/quizes/:quizId(\\d+)/answer',   quizController.answer);
18 router.get('/quizes/new',                    quizController.new);
19 router.post('/quizes/create',                quizController.create);
20 router.get('/quizes/:quizId(\\d+)/edit',     quizController.edit);
21 router.put('/quizes/:quizId(\\d+)',          quizController.update);
22 router.delete('/quizes/:quizId(\\d+)',       quizController.destroy);
```

quiz_controller.js UNREGISTERED

```
99  // DELETE /quizes/:id
100 exports.destroy = function(req, res) {
101   req.quiz.destroy().then( function() {
102     res.redirect('/quizes');
103   }).catch(function(error){next(error)});
104 };
```

Paso 2a: controlador destroy

id	pregunta	respue
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....

quiz13_editar_preg

23 items, 103,39 GB available

Name

- app.js
- bin
- controllers
  - quiz_controller.js
- models
- node_modules
- package.json
- Procfile
- public
- quiz.sqlite
- routes
  - index.js
- views
  - error.ejs

Final del tema

Ejercicio: Selector de temática en Quiz

Añadir a la versión Quiz 14 “Borrar Preguntas” a las preguntas que se suben al portal Quiz un nuevo campo denominado temática, que indique el tema al que pertenece: geografía, historia, deportes, ocio, cultura, ciencia, o tecnología.

Para ello habrá que modificar el modelo, añadiendo a la tabla Quiz este nuevo campo, que podrá ser de tipo texto.

Además deberán habrá que hacer modificar las vistas new.ejs y edit.ejs añadiendo una entrada para incluir este nuevo campo. Se recomienda utilizar un desplegable de tipo

<select> <option>otros</option> <option>geografía</option> …. </select>

que despliegue las temáticas.

También habrá que mostrar este campo en la vista show.ejs. En la lista de preguntas (vista índex.ejs) habrá que añadir otro selector que permita filtrar las preguntas por temática.

Además habrá que modificar los controladores y demás elementos para tener en cuenta este nuevo elemento.

Quiz 15: Crear Comentario

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

©

Quiz 15: Crear comentario

Objetivo: Añadir a la aplicación Quiz la posibilidad de hacer comentarios a las preguntas, que se introducirán a través de un nuevo formulario y se verán en la vista de cada pregunta

✦**Paso 1**: Añadir nueva tabla en la DB para guardar los comentarios a las preguntas

- a: Añadir definición de la tabla de comentarios **models/comment.js**
- b: Importar tabla en **models/models.js** y definir la relación entre las tablas **quiz** y **comment**

✦**Paso 2**: Añadir formulario de crear comentarios: **GET /quizes/:quizId/comments/new**

- a: Añadir en **comments_controller.js** acción **new** asociada a ruta **/quizes/:quizId/comment/new**
- b: Añadir en **routes/index.js** ruta **GET /quizes/:quizId/comments/new**
- c: Añadir vista con formulario de crear comentario: **views/comments/new.ejs**

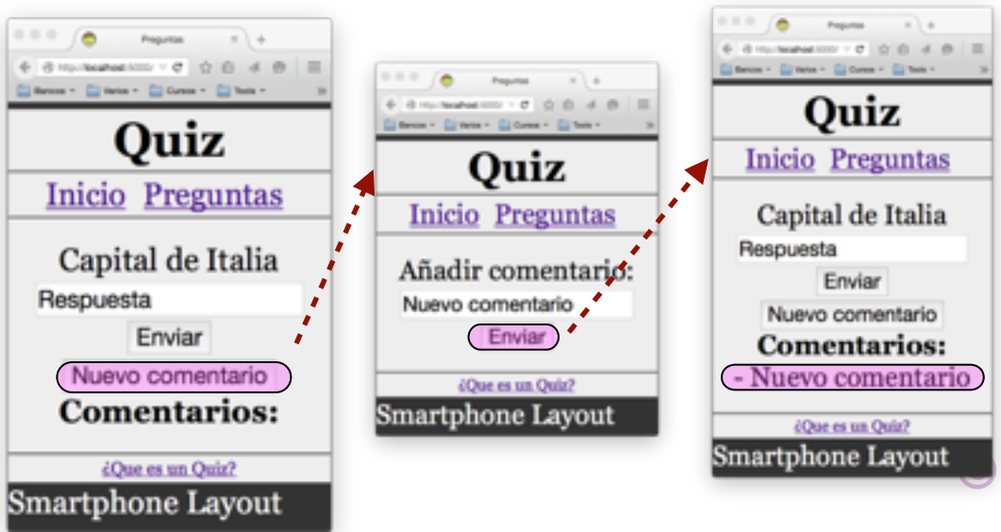
✦**Paso 3**: Añadir **POST /quizes/:quizId/comments** para introducir nuevos comentarios en DB

- a: Añadir en **comment_controller.js** acción **create**
- b: Añadir en **routes/index.js** ruta **POST /quizes/:quizId/comments**

✦**Paso 4**: Modificar vista **quizes/show** para incluir comentarios y botón de crear comentario

- a: Modificar **autoload** del controlador quiz**_controller.js** para que cargue también los comentarios del quiz
- b: Modificar vista **views/quizes/show** para incluir comentarios y botón de crearlos

✦**Paso 5**: Guardar **versión (commit)** git y subir a **Heroku**



id	pregunta	resp.
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....

id	texto
1	Nuevo comentario
2	Fue Sintra en el pasado
3	Rome en ingles
4	.....

Paso 1a: Tabla Comment

Fichero **quiz.sqlite** contiene los datos, en nuestro caso las preguntas.

quiz.sqlite se puede borrar para regenerar o copiar para hacer un backup.

comment.js define el formato de la tabla de comentarios.

Tiene solo 1 campo de tipo string:

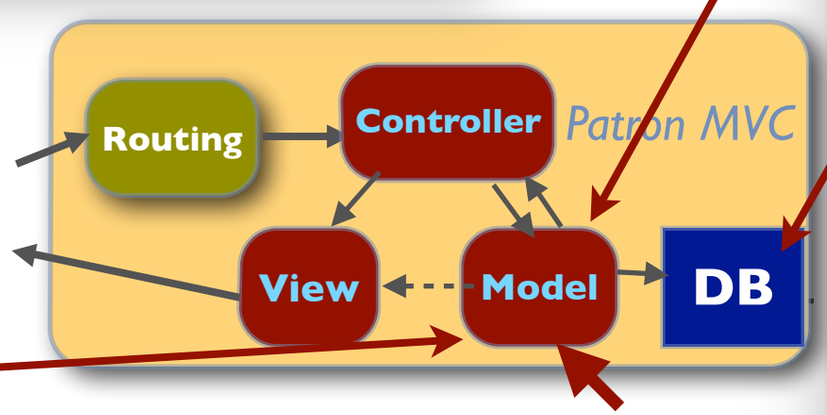
- texto: DataTypes.STRING

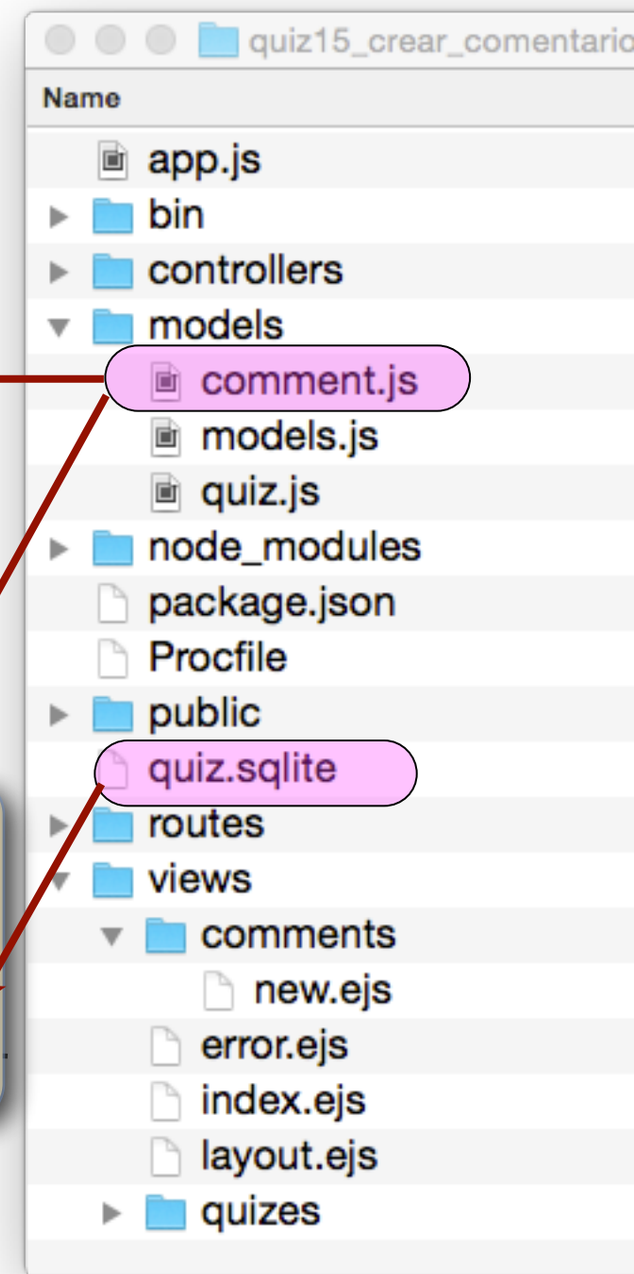
comment.js UNREGISTERED

```
1  // Definicion del modelo de Comment con validación
2
3  module.exports = function(sequelize, DataTypes) {
4    return sequelize.define(
5      'Comment',
6      { texto: {
7          type: DataTypes.STRING,
8          validate: { notEmpty: {msg: "-> Falta Comentario"}}
9        }
10     }
11   );
12 }
```

Tabla Comment

id	texto
1	Nuevo comentario
2	Fue Sintra en el pasado
3	Rome en ingles
4	……





Relaciones entre Modelos

- Las relaciones permiten asociar elementos de tablas diferentes
  - Hay tres tipos de relaciones básicas: **1-a-1, 1-a-N y N-a-N**
    - http://docs.sequelizejs.com/en/latest/docs/associations/
    - http://docs.sequelizejs.com/en/latest/api/associations/

- Relación **1-a-1**
  - **1** elemento de una tabla se asocia con 1 elemento en otra tabla
    - Por ejemplo, una pregunta tiene asociado un fichero
- Relación **1-a-N**
  - 1 elemento de una tabla se asocia con N elementos en otra tabla
    - Por ejemplo, una pregunta puede tener varios comentarios
- Relación **N-a-N**
  - N elementos de una tabla se asocian con N elementos en otra tabla
    - Por ejemplo, N preguntas gustan (like) a N usuarios

Paso 1b: Relación Quiz a Comment

La **relación de tipo 1-a-N** entre **Quiz** y **Comment** se define en **models.js** con

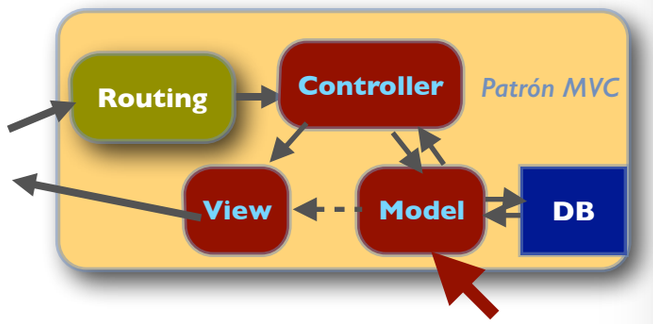
- **Comment.belongsTo(Quiz)**, que indica que los comentarios pertenecen a los quizes.
- **Quiz.hasMany(Comment)**, que indica que un quiz puede tener muchos comentarios.

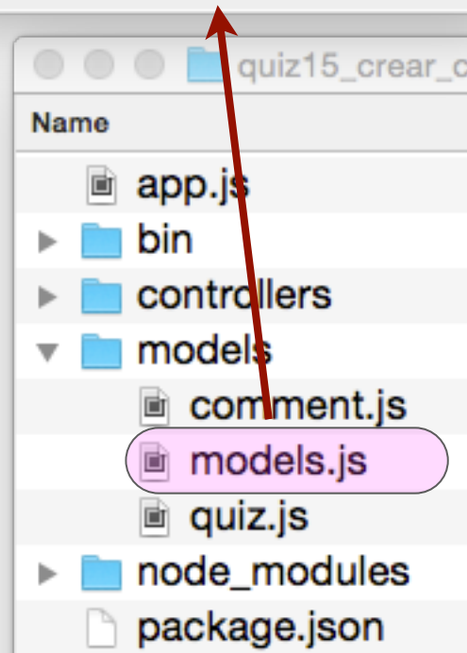
models.js

```
28  ......
29  // Importar definicion de la tabla Quiz
30  var quiz_path = path.join(__dirname,'quiz');
31  var Quiz = sequelize.import(quiz_path);
32
33  // Importar definicion de la tabla Comment
34  var comment_path = path.join(__dirname,'comment');
35  var Comment = sequelize.import(comment_path);
36
37  Comment.belongsTo(Quiz);
38  Quiz.hasMany(Comment);
39
40  exports.Quiz = Quiz; // exportar tabla Quiz
41  exports.Comment = Comment;
42  ......
```

id	pregunta	resp.
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....

id	texto
1	Nuevo comentario
2	Fue Sintra en el pasado
3	Rome en ingles
4	.....





Definición de relaciones

- Las relaciones se definen con métodos heredados del modelo
  - 1-a-1: belongsTo(…) y hasOne(…)
  - 1-a-N: belongsTo(…) y hasMany(…)
  - N-a-N: belongsToMany(…) y hasMany(…)
    - http://docs.sequelizejs.com/en/latest/api/associations/
    - http://docs.sequelizejs.com/en/latest/docs/associations/

id	pregunta	resp.
1	Capital de Italia	Roma
2	Capital de Portugal	Lisboa
3	.....	.....
4	.....	.....

id	comentario
1	Nuevo comentario
2	Fue Sintra en el pasado
3	Rome en ingles
4	……

Definimos las siguientes 2 nuevas primitivas en el interfaz REST:

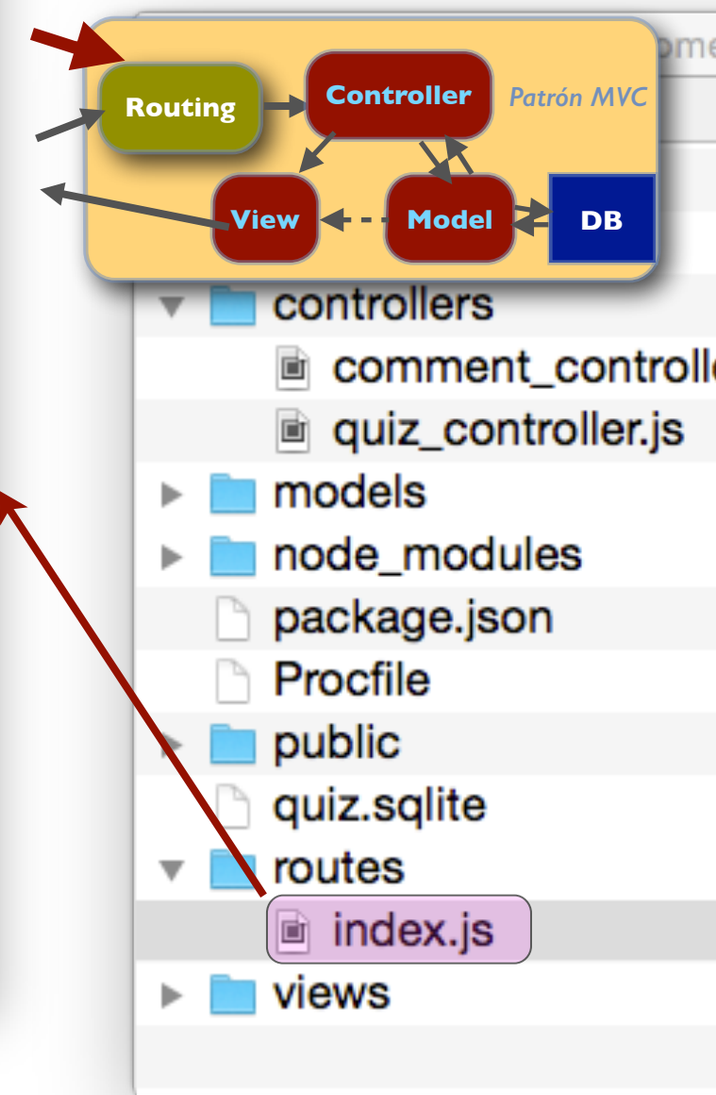
GET /quizes/:quizId/comments/new accede al formulario de crear comentario, asociado al **quiz :id.**

POST /quizes/:quizId/comments crea una entrada en la **tabla comments,** asociada a **:quizId** en Quiz

Pasos 2b y 3b: rutas

index.js UNREGISTERED

```
1  var express = require('express');
2  var router = express.Router();
3
4  var quizController = require('../controllers/quiz_controller');
5  var commentController = require('../controllers/comment_controller');
6
7  // Página de entrada (home page)
8  router.get('/', function(req, res) {
9    res.render('index', { title: 'Quiz', errors: []});
10 });
11
12 // Autoload de comandos con :quizId
13 router.param('quizId', quizController.load);  // autoload :quizId
14
15 // Definición de rutas de /quizes
16 router.get('/quizes',                        quizController.index);
17 router.get('/quizes/:quizId(\\d+)',          quizController.show);
18 router.get('/quizes/:quizId(\\d+)/answer',   quizController.answer);
19 router.get('/quizes/new',                    quizController.new);
20 router.post('/quizes/create',                quizController.create);
21 router.get('/quizes/:quizId(\\d+)/edit',     quizController.edit);
22 router.put('/quizes/:quizId(\\d+)',          quizController.update);
23 router.delete('/quizes/:quizId(\\d+)',       quizController.destroy);
24
25 router.get('/quizes/:quizId(\\d+)/comments/new', commentController.new);
26 router.post('/quizes/:quizId(\\d+)/comments',    commentController.create);
27
28 module.exports = router;
29
```



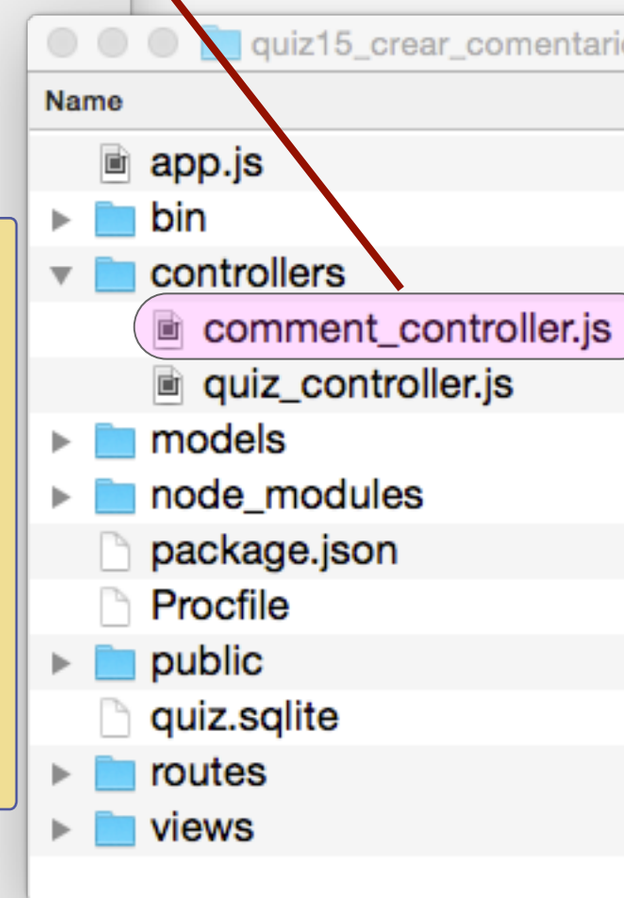
Creamos el controlador de comentarios solo con acciones new y create:

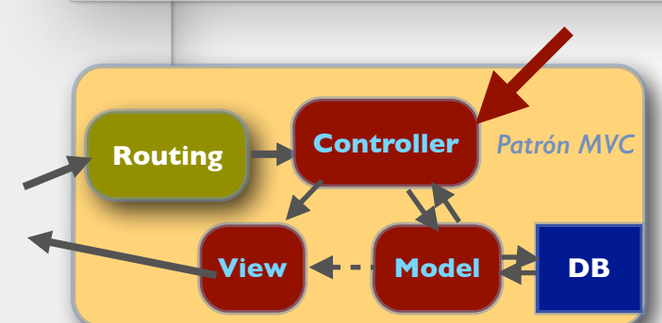
- Acción **new:** instancia el formulario de crear comentarios
- Acción **create:** guarda el comentario en la **tabla comment** de la base de datos y luego redirecciona a la vista con la lista de preguntas.

comment_controller.js UNREGISTERED

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

// GET /quizes/:quizId/comments/new
exports.new = function(req, res) {
  res.render('comments/new.ejs', {quizid: req.params.quizId, errors: []});
};

// POST /quizes/:quizId/comments
exports.create = function(req, res) {
  var comment = models.Comment.build(
      { texto: req.body.comment.texto,
        QuizId: req.params.quizId
        });

  comment
  .validate()
  .then(
    function(err){
      if (err) {
        res.render('comments/new.ejs', {comment: comment, errors: err.errors});
      } else {
        comment // save: guarda en DB campo texto de comment
        .save()
        .then( function(){ res.redirect('/quizes/'+req.params.quizId)})
      }      // res.redirect: Redirección HTTP a lista de preguntas
    }
  ).catch(function(error){next(error)});

};
```

La relación **belongsTo(...)** de Comment a Quiz añade un parámetro **:quizId** adicional en cada elemento de la **tabla Comments** que indica el quiz asociado. Se utiliza el nombre :quizId definido en la ruta en **routes/index.js**, salvo que se indique otro nombre.





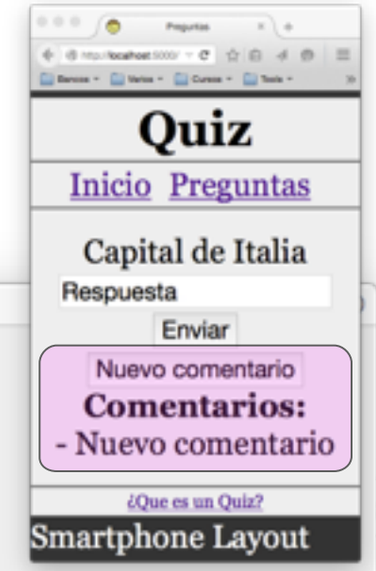
Pasos 2a y 3a: comment_controller

Pasos 2c y 4b: vistas

show.ejs

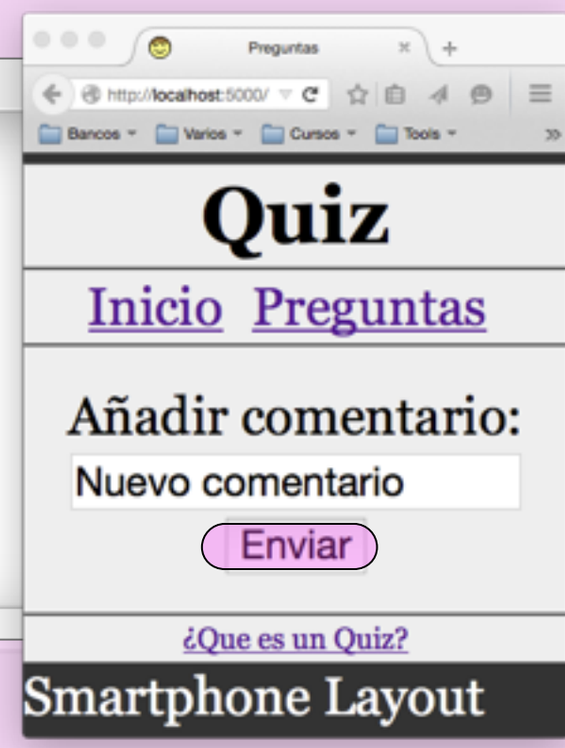
```
<form method="get" action="/quizes/<%= quiz.id %>/answer">
  <%= quiz.pregunta %> <p>
  <input type="text" name="respuesta" value="Respuesta"/>
  <input type="submit" value="Enviar">
</form>

<p><a href="/quizes/<%= quiz.id %>/comments/new"><button>Nuevo comentario</button></a></p>

<p><strong>Comentarios:</strong></p>
<%for(index in quiz.Comments){%>
    <p>- <%=quiz.Comments[index].texto%></p>
<%}%>
```

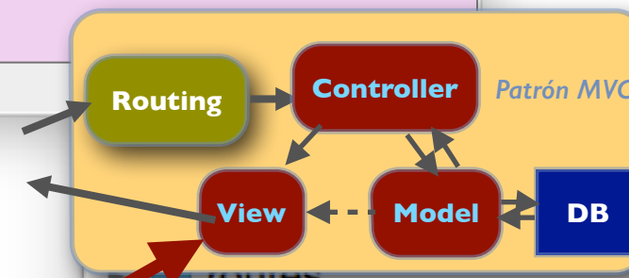
Paso 4b

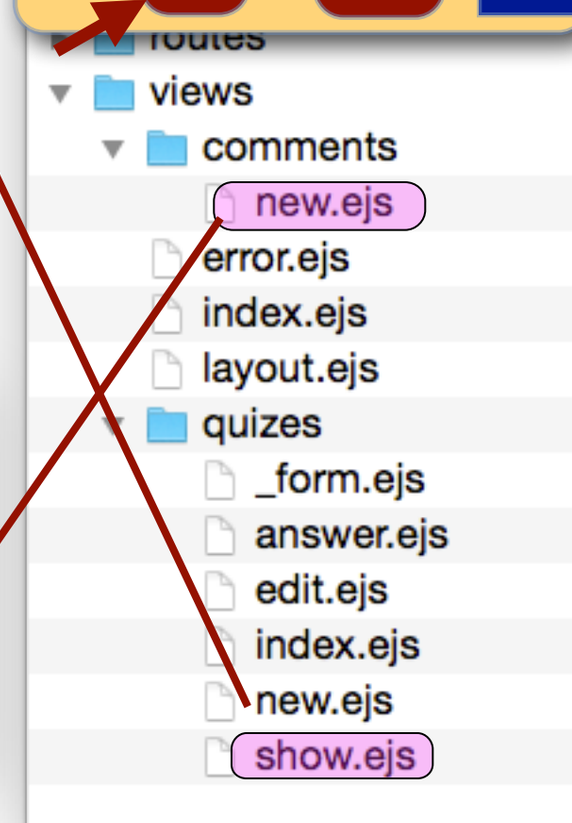


Se modifica la vista **view/quizes/show.ejs** para gestionar los comentarios de una pregunta.

Además se crea la vista **view/comments/new.ejs** para crear nuevos comentarios.







new.ejs

```
Añadir comentario: <p>

<form method="post" action="/quizes/<%=quizid%>/comments/" >
  <input type="text" id="comment" name="comment[texto]" value=""/> <p>
  <button type="submit">Enviar</button>
</form>
```

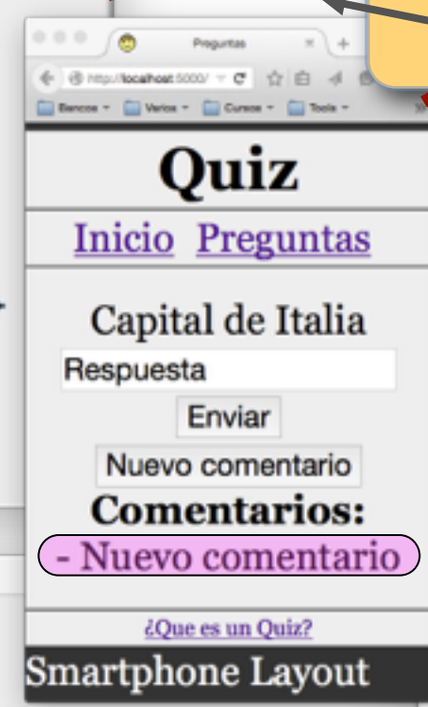
Paso 2c

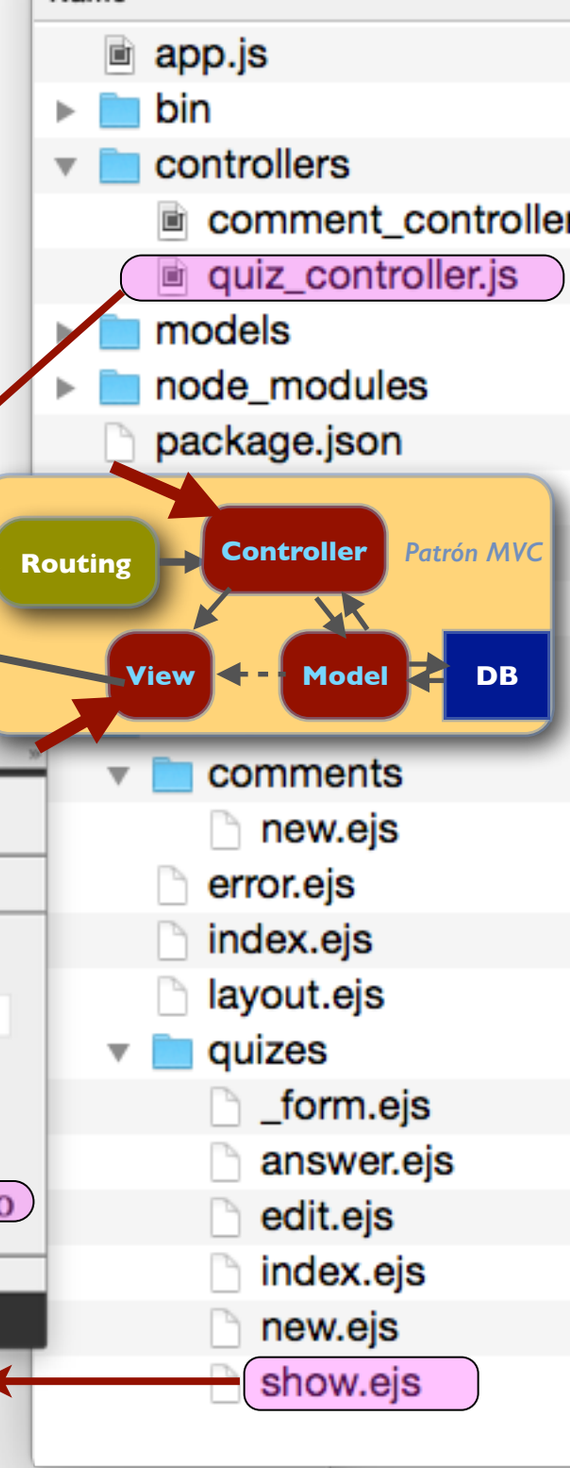
El parámetro "**where: {id: Number(quizId)}**" de **find(...)** indica buscar el quiz identificado por **quizId**. Pero además está el parámetro "**include: [{model: models.Comment}]** de **find(...)**, que solicita cargar en la propiedad **quiz.Comments**, los comentarios asociados al quiz a través de la **relación 1-N** entre **Quiz** y **Comment**.

quiz_controller.js UNREGISTERED

```
2
3  // Autoload :id
4  exports.load = function(req, res, next, quizId) {
5    models.Quiz.find({
6              where: { id: Number(quizId) },
7              include: [{ model: models.Comment }]
8          }).then(function(quiz) {
9        if (quiz) {
10         req.quiz = quiz;
11         next();
12       } else{next(new Error('No existe quizId=' + quizId))}
13     }
14   ).catch(function(error){next(error)});
15 };
16
```

Paso 4a





Paso 4b

show.ejs

```
1  <form method="get" action="/quizes/<%= quiz.id %>/answer">
2    <%= quiz.pregunta %> <p>
3    <input type="text" name="respuesta" value="Respuesta"/>
4    <input type="submit" value="Enviar">
5  </form>
6
7  <p><a href="/quizes/<%= quiz.id %>/comments/new"><button>Nuevo comentario</button></a></p>
8
9  <p><strong>Comentarios:</strong></p>
10 <%for(index in quiz.Comments){%>
11     <p>- <%=quiz.Comments[index].texto%></p>
12 <%}%>
13
```

Paso 4: visualizar comentarios

Final del tema

Quiz 16: Autenticación y sesión

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

©

Quiz 16: Autenticación y sesión

Objetivo: Añadir gestión de sesión a la aplicación Quiz para distinguir entre usuarios anónimos y usuarios que se autentican realizando **login**

✦**Paso 1**: Importar, instalar y configurar middleware de gestión de sesiones

- a: Importar e instalar middleware **express-session** en **app.js** y en **package.json**
- b: Hacer visible la sesión en las vistas **quizes** y **comment**
- c: guardar path de re-dirección después de login

✦**Paso 2**: Añadir 3 rutas de gestión de sesión en **routes/index.js**

- Cargar formulario de login: **GET /login**
- Crear sesión: **POST /login**
- Destruir sesión: **GET /logout** (funciona, pero debería ser DELETE)

✦**Paso 3**: Crear controlador de sesión **controllers/session_controller.js**

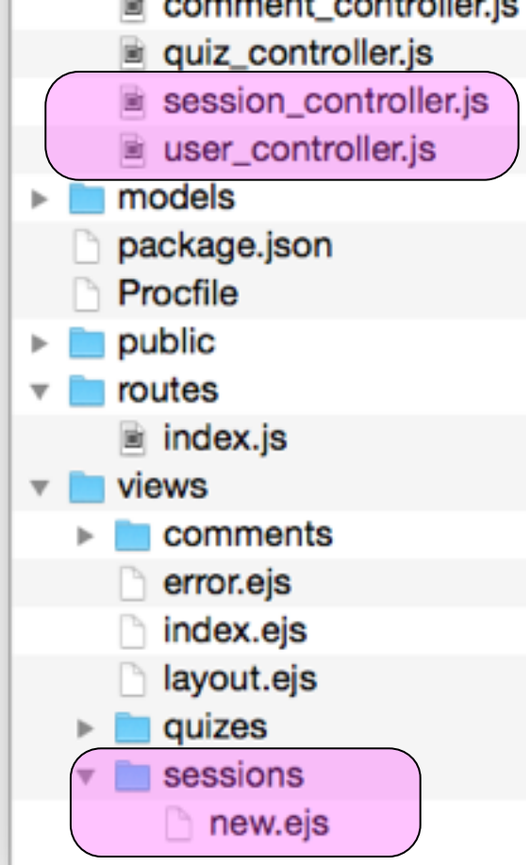
- a: Añadir acción **new** para la ruta **GET /login** para cargar formulario de login
- b: Añadir acción **create** para la ruta **POST /login** para crear la sesión de usuario
- **c**: Añadir acción **destroy** para a ruta **GET /logout** para destruir la sesión

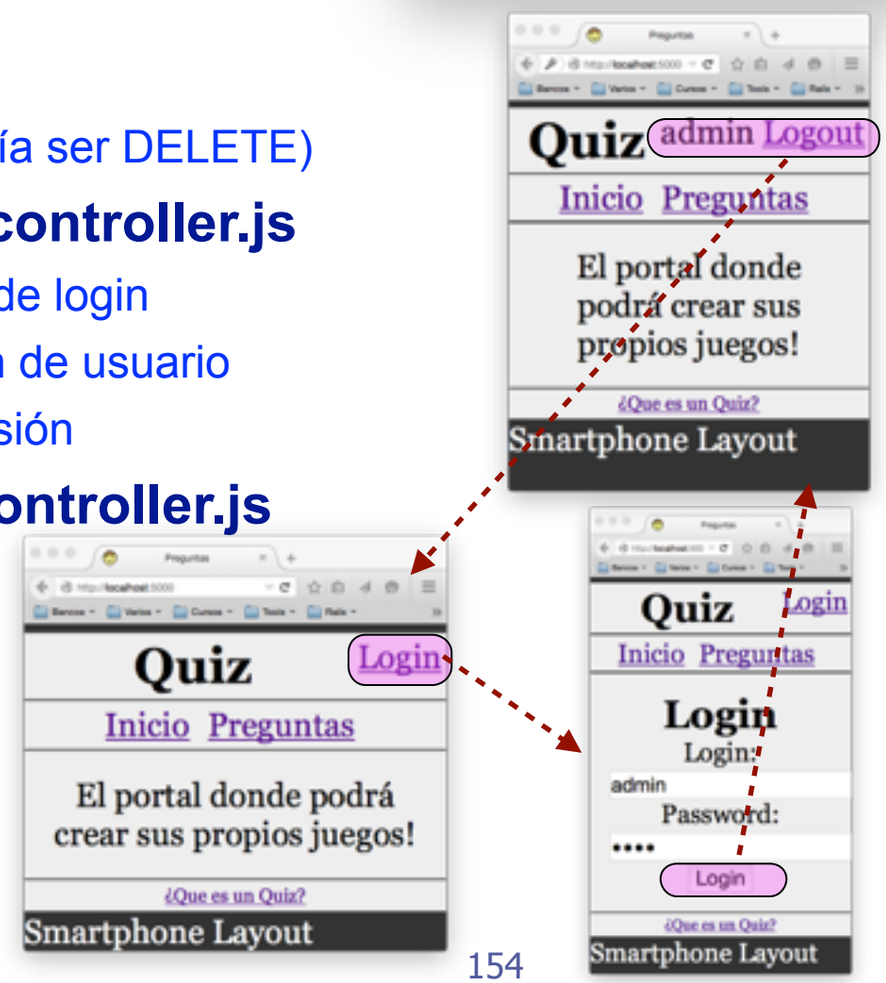
✦**Paso 4**: Crear controlador de usuarios: **controllers/user_controller.js**

✦**Paso 5**: Añadir **botón login/logout** en **views/layout.ejs**

✦**Paso 6**: Añadir vista de login **views/sessions/new.ejs**

✦**Paso 7**: Guardar **versión (commit)** git y subir a **Heroku**





Paso 1a

- **express-session**
  - middleware de gestión de sesiones
    - https://github.com/expressjs/session
- Identificador de sesión: **sid**
  - Se guarda en una cookie cifrada del cliente
    - La sesión está accesible en **req.session**

```
app.js                                                    UNREGISTERED
8  var methodOverride = require('method-override');
9  var session = require('express-session');
10
11 var routes = require('./routes/index');
12
13 var app = express();
14
15 // view engine setup
16 app.set('views', path.join(__dirname, 'views'));
17 app.set('view engine', 'ejs');
18
19 app.use(partials());
20 // uncomment after placing your favicon in /public
21 // app.use(favicon(__dirname + '/public/favicon.ico'));
22 app.use(favicon(__dirname + '/public/favicon.ico'));
23 app.use(logger('dev'));
24 app.use(bodyParser.json());
25 app.use(bodyParser.urlencoded());
26 app.use(cookieParser('Quiz 2015'));
27 app.use(session());
28 app.use(methodOverride('_method'));
```

Se importa el **paquete express-session** instalado con **npm**

```
6    "start": "node ./bin/www"
7  },
8  "engines": {
9    "node": "0.10.x",
10   "npm": "1.4.x"
11 },
12 "devDependencies": {
13   "sqlite3": "^3.0.4"
14 },
15 "dependencies": {
16   "body-parser": "~1.8.1",
17   "cookie-parser": "~1.3.3",
18   "debug": "~2.0.0",
19   "ejs": "~0.8.5",
20   "express": "~4.9.0",
21   "express-partials": "^0.3.0",
22   "express-session": "~1.0.3",
23   "method-override": "^2.3.1",
24   "morgan": "~1.3.0",
25   "pg": "^4.1.1",
26   "sequelize": "^2.0.0-rc4",
27   "serve-favicon": "~2.1.3"
28 }
29 }
```

// instalar el MW **express-session**
// de gestión de sesiones con **npm**

..$ npm install --save express-session

cookieParser: añadir semilla 'Quiz 2015' para cifrar cookie

Instalar MW session

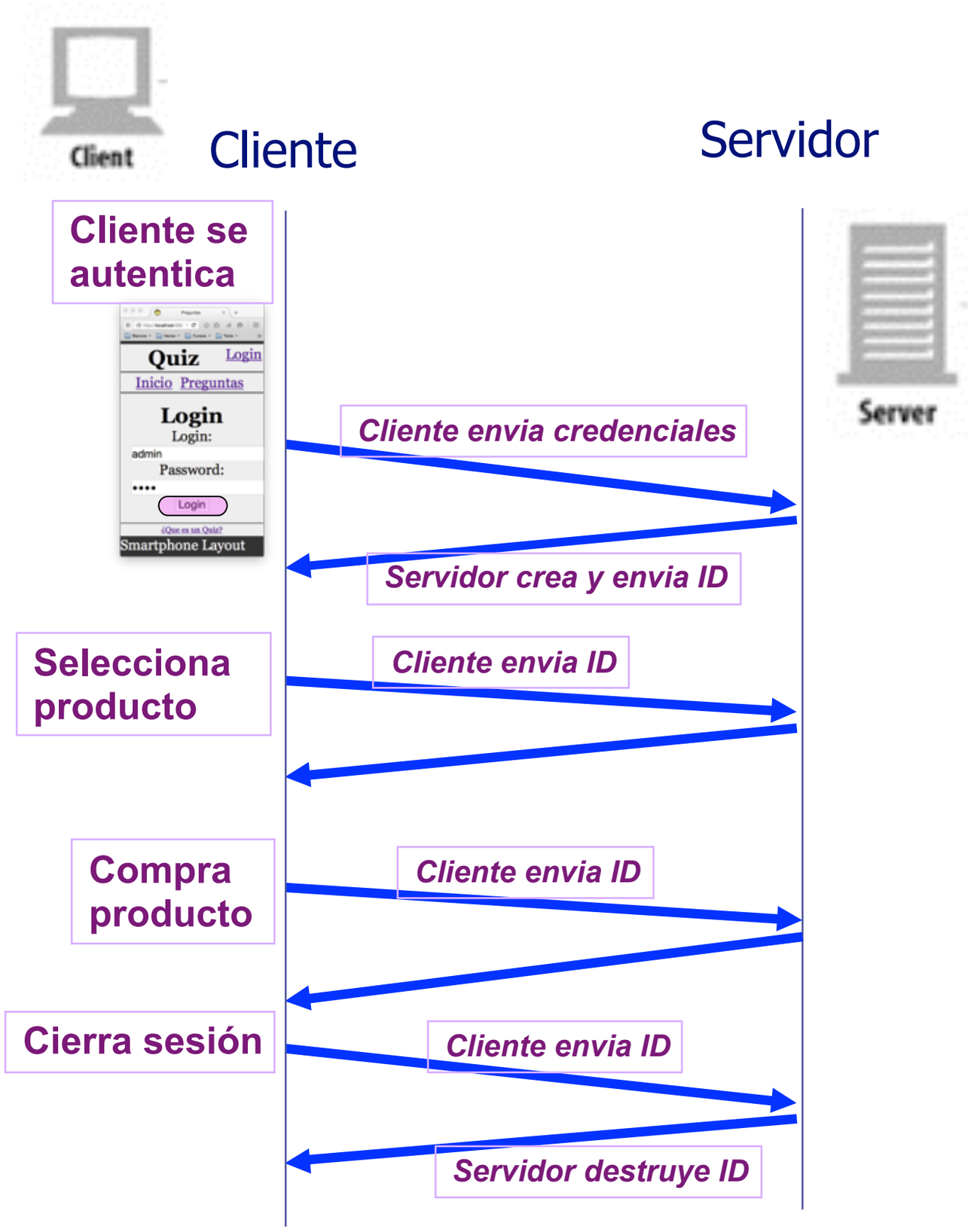
Sesiones

Sesiones HTTP es basan en un identificador **ID del cliente**:

- El servidor crea una sesión y un ID cuando el usuario se autentica.
- Durante toda la sesión el usuario utilizará el mismo ID para todas las transacciones.
- La sesión (y el ID) se liberan al acabar la sesión (al hacer logout o al vencer un temporizador).

El identificador de sesión se suele guardar en:

- **Cookie** (lo mas habitual).
- **Parámetro de ruta** (Fat URLs).
- **Parámetro oculto**



Cookies

Las cookies son como **variables del navegador,** accesibles solo desde el servidor que las configura.

Un servidor configura cookies en un cliente con el parámetro de la cabecera "**Set-Cookie: …**". Las cookies están asociadas a su **dominio/ruta (host/path)**, identificando de esta forma un conjunto de directorios o rutas asociadas con él. Una cookie tiene además un **plazo de expiración**.

El cliente siempre **devuelve las cookies** en las solicitudes al **dominio/ruta** al que están asociadas, en el parámetro de la cabecera **"Cookie:…"**, siempre que no hayan expirado.

Comienzo →

Cabecera →

Cuerpo

```
HTTP/1.1   200   OK
Content-type: text/html, …
Content-length: 567
Set-Cookie: Id=user675; Expires=Tue, 31 Mar 2008 11:30:12 GMT; Path=/
Set-Cookie: User=Pepe+Perez; Expires=Tue, 31 Mar 2008 11:30:12 GMT; Path=/
……
```

Comienzo →

Cabecera →

Cuerpo →

```
GET   /lib/.html   HTTP/1.1
Accept: text/*
Accept-language: en, sp
Cookie: Id=user675; User=Pepe+Perez
```

Sesión: Cookie connect.sid

Bancos Varios Cursos Tools Rails HTML5 HTML5-RTC MiTwitter CTING

Quiz: el juego de las preguntas Login

Inicio
Preguntas

Bienvenido a Quiz
El portal donde podrá crear sus propios juegos!

Inspector | Console | Debugger | Style Editor | Performance | Network

Net | CSS | JS | Security | Logging | Clear | Filter output

```
GET http://localhost:5000/                                  [HTTP/1.1 200 OK 49ms]
GET http://localhost:5000/stylesheets/style.css             [HTTP/1.1 200 OK 16ms]
GET http://localhost:5000/stylesheets/wide.css              [HTTP/1.1 200 OK 17ms]
GET http://localhost:5000/stylesheets/smartphone.css        [HTTP/1.1 200 OK 17ms]
```

Inspect Networ

Request URL: http://localhost:5000/
Request Method: GET
Status Code: HTTP/1.1 200 OK

Request Headers 07:49:48.000

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:35.0) Gecko/20100101 Firefox/35.0
Host: localhost:5000
Connection: keep-alive
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Sent Cookie

connect.sid: s:QMhcWQGLZO9mUpnA6mGY8Sk7.E6I3SkclbXP0OiEyXIpjSmVfW/DBI78txrJ8JCuItd4

Response Headers Δ49ms

X-Powered-By: Express
Etag: W/"QU5f194RarWuwg9RhAhiDA=="
Date: Fri, 27 Feb 2015 06:49:48 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1113
Connection: keep-alive

Received Cookie

connect.sid: s:fi5faTyb2oUUuZJGeLZpe5es.9JO6zvDgzvgC1nSGL4DwOLrufpYWfR+TNvOa7rmpa+o

El identificador de sesión se guarda en la cookie cifrada:

connect.sid

Los parámetros se guardan en el servidor, asociados al sid de cada sesión.

Además definimos en app.js un middleware que realiza 2 funciones:

- **1.b**: copia la sesión que está accesible en **req.session** en **res.locals.session** para que esté accesible en las vistas.
- **1.c**: guarda la ruta de cada solicitud HTTP en la variable **session.redir** para poder redireccionar a la vista anterior después de hacer login o logout.

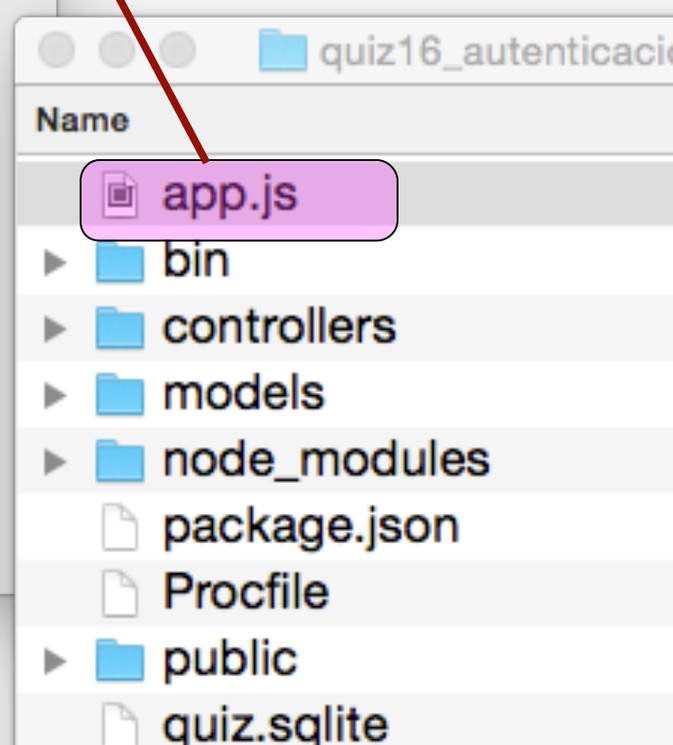
app.js UNREGISTERED

```
26  app.use(cookieParser('Quiz 2015'));
27  app.use(session());
28  app.use(methodOverride('_method'));
29  app.use(express.static(path.join(__dirname, 'public')));
30
31  // Helpers dinamicos:
32  app.use(function(req, res, next) {
33
34    // guardar path en session.redir para despues de login
35    if (!req.path.match(/\/login|\/logout/)) {
36      req.session.redir = req.path;
37    }
38
39    // Hacer visible req.session en las vistas
40    res.locals.session = req.session;
41    next();
42  });
43
44  app.use('/', routes);
```

Paso 1c

Paso 1b

Pasos 1b y 1c



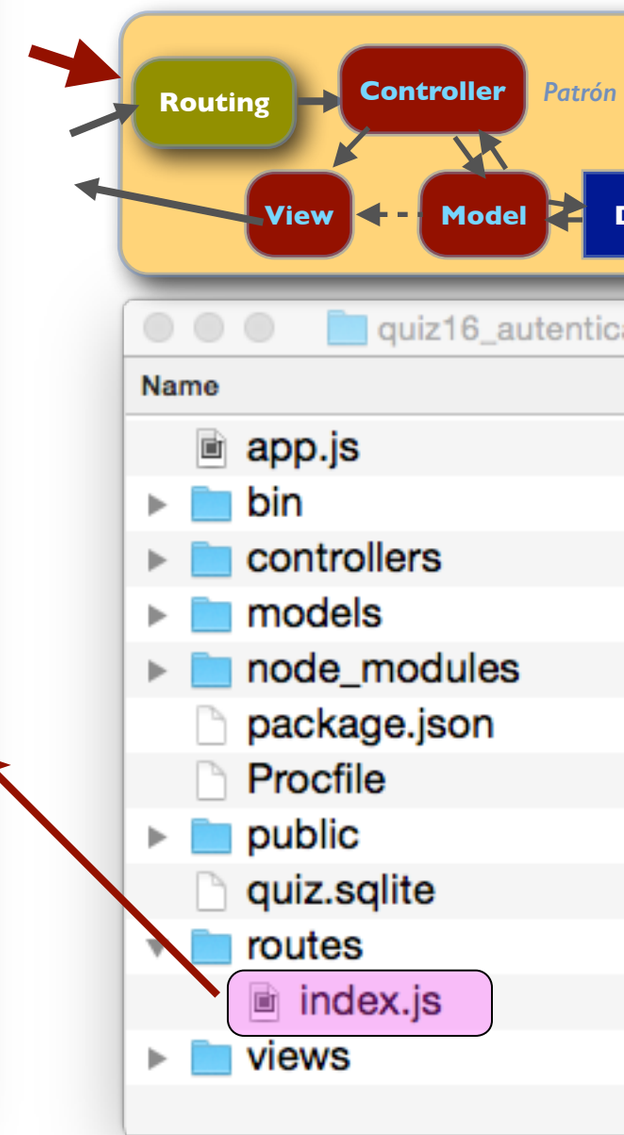
Paso 2

El enrutador importa el controlador de sesiones y define las rutas 3 rutas asociadas:

GET /login // formulario
POST /login // Crear sesión
GET /logout // Destruir ses.

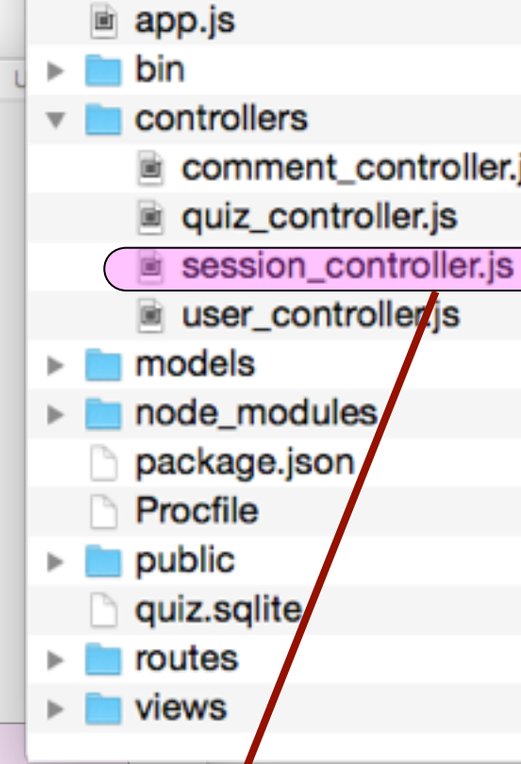
index.js

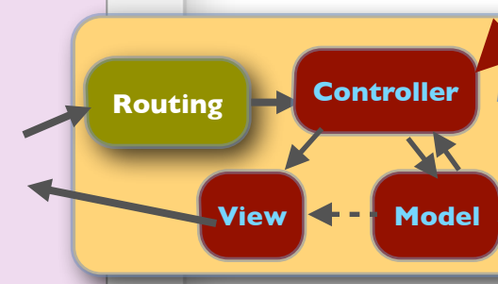
```js
var express = require('express');
var router = express.Router();

var quizController = require('../controllers/quiz_controller');
var commentController = require('../controllers/comment_controller');
var sessionController = require('../controllers/session_controller');

// Página de entrada (home page)
router.get('/', function(req, res) {
  res.render('index', { title: 'Quiz', errors: []});
});

// Autoload de comandos con :quizId
router.param('quizId', quizController.load);  // autoload :quizId

// Definición de rutas de sesion
router.get('/login',  sessionController.new);     // formulario login
router.post('/login', sessionController.create);  // crear sesión
router.get('/logout', sessionController.destroy); // destruir sesión

// Definición de rutas de /quizes
router.get('/quizes',                       quizController.index);
router.get('/quizes/:quizId(\\d+)',         quizController.show);
router.get('/quizes/:quizId(\\d+)/answer', quizController.answer);
router.get('/quizes/new',                   quizController.new);
router.post('/quizes/create',               quizController.create);
router.get('/quizes/:quizId(\\d+)/edit',    quizController.edit);
router.put('/quizes/:quizId(\\d+)',         quizController.update);
router.delete('/quizes/:quizId(\\d+)',      quizController.destroy);

// Definición de rutas de comentarios
router.get('/quizes/:quizId(\\d+)/comments/new', commentController.new);
router.post('/quizes/:quizId(\\d+)/comments',    commentController.create);

module.exports = router;
```



```
// Get /login   -- Formulario de login
exports.new = function(req, res) {
    var errors = req.session.errors || {};
    req.session.errors = {};

    res.render('sessions/new', {errors: errors});
};

// POST /login   -- Crear la sesion
exports.create = function(req, res) {

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

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

        if (error) {  // si hay error retornamos mensajes de error de sesión
            req.session.errors = [{"message": 'Se ha producido un error: '+error}];
            res.redirect("/login");
            return;
        }

        // Crear req.session.user y guardar campos   id  y  username
        // La sesión se define por la existencia de:    req.session.user
        req.session.user = {id:user.id, username:user.username};

        res.redirect(req.session.redir.toString());// redirección a path anterior a login
    });
};

// DELETE /logout   -- Destruir sesion
exports.destroy = function(req, res) {
    delete req.session.user;
    res.redirect(req.session.redir.toString()); // redirect a path anterior a login
};
```

Al crear y destruir la sesión se redirecciona a la ruta guardada en **req.sesion.redir**, de forma que se vuelva a la página desde la que se realiza el login o el logout.





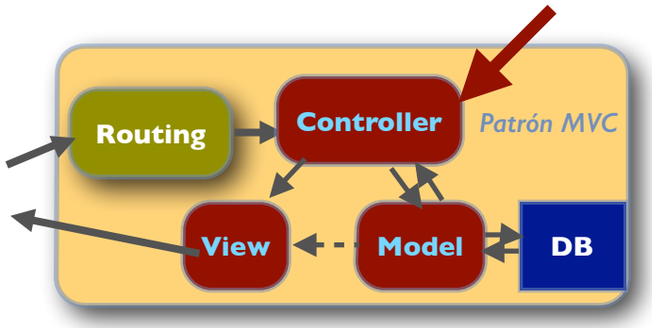
Pasos 3a, 3b y 3c

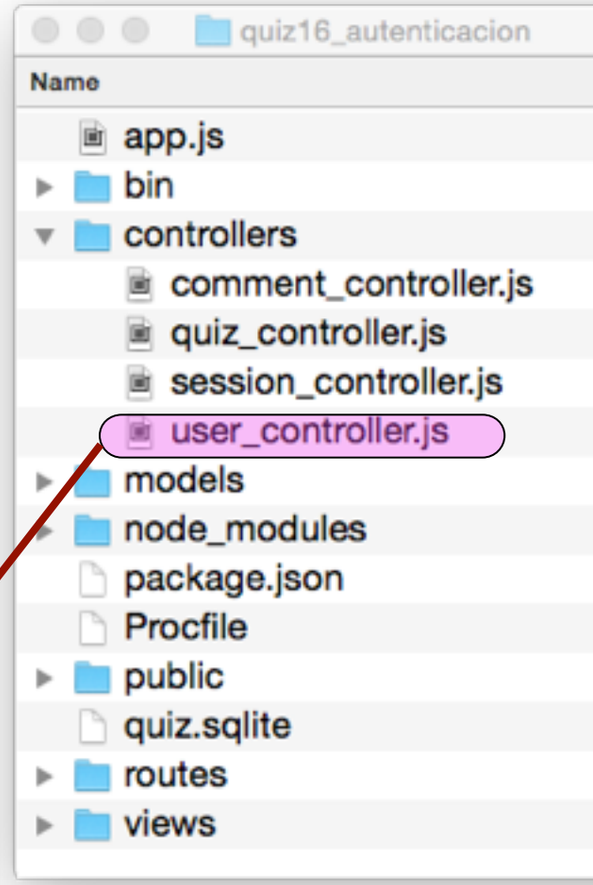
Paso 4: controlador autenticar

Creamos el controlador **autenticar** en el nuevo fichero **controllers/user_controller.js** para el control de usuarios.

La gestión de usuarios es muy simplista y se basa en tener **2 usuarios** registrados predefinidos en la **variable users**. En un portal real suele haber una tabla de gestión de usuarios.

El controlador autenticar comprueba si el usuario está incluido en el objeto de la variable users y si el password es correcto.





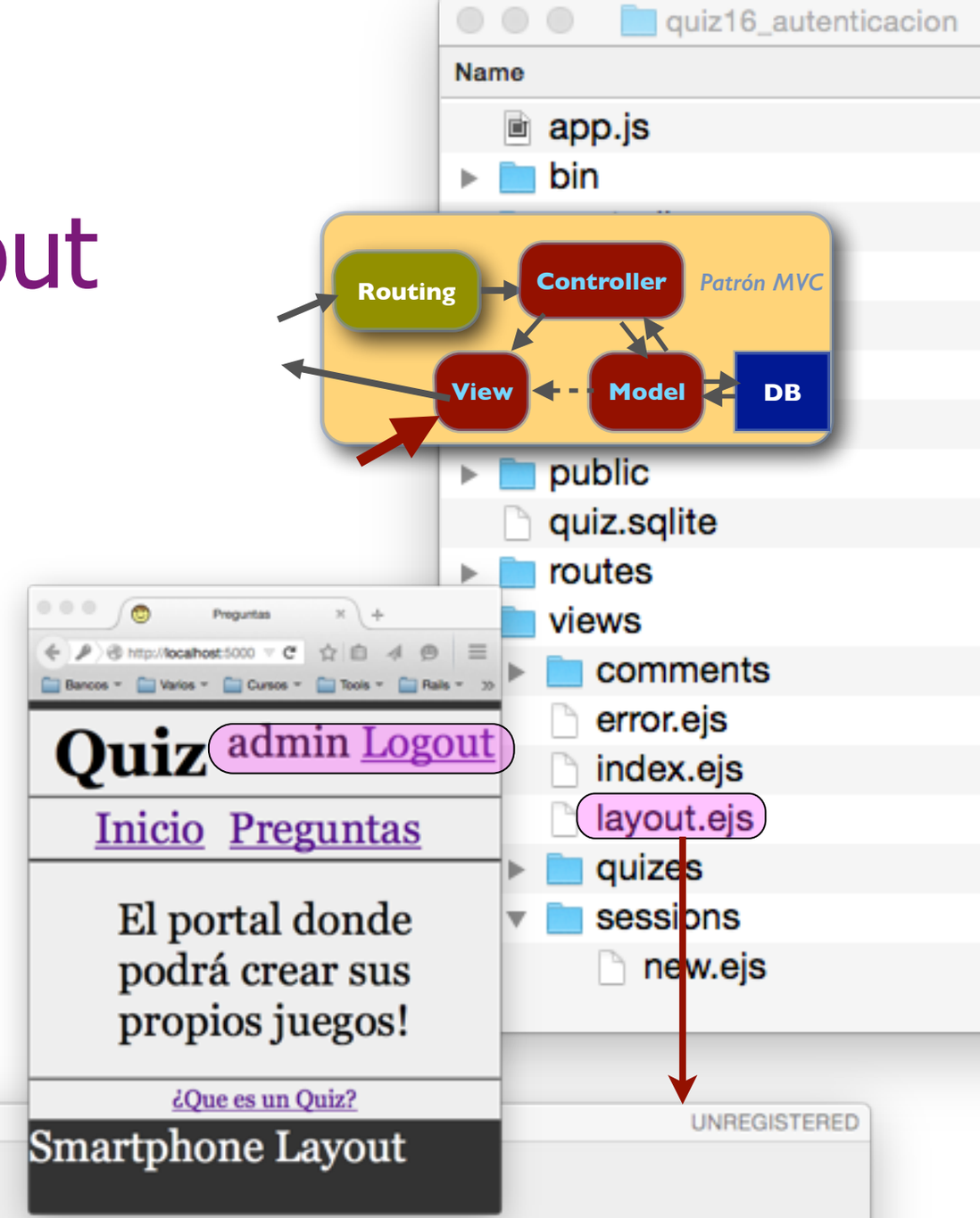
user_controller.js UNREGISTERED

```
var users = { admin: {id:1, username:"admin", password:"1234"},
              pepe:  {id:2, username:"pepe",  password:"5678"}
            };

// Comprueba si el usuario esta registrado en users
// Si autenticación falla o hay errores se ejecuta callback(error).
exports.autenticar = function(login, password, callback) {
    if(users[login]){
        if(password === users[login].password){
            callback(null, users[login]);
        }
        else { callback(new Error('Password erróneo.')); }
    } else { callback(new Error('No existe el usuario.'));}
};
```

Paso 5: botón de login/logout

El botón de login y logout se incluye en el header de **views/layout.ejs** en la misma linea donde se sitúa el título en la cabecera de la página. El botón se asocia a **class="right"** para colocarlo en el extremo derecho con el comando CSS añadido a **style.css**: **.right {float: right;}**

La propiedad **sesión.user** que indica si la sesión está establecida o no, se utiliza para decidir si se muestra el botón de login o el de logout con el nombre del usuario al lado.



layout.ejs UNREGISTERED

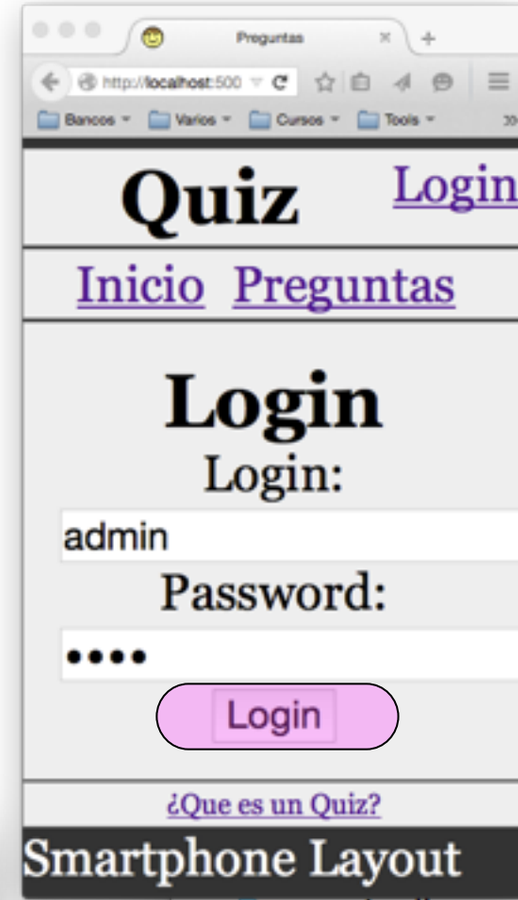
```
18 <div id="page-wrap">
19   <header class="main" id="h1">
20     <%if(!session.user){%>
21       <span class="right"><a href="/login">Login</a></span>
22     <%}else{%>
23       <span class="right"> <%=session.user.username%> <a href="/logout">Logout</a></span>
24     <%}%>
25
26     <h2>Quiz<span>: el juego de las preguntas</span></h2>
27   </header>
28
```

Paso 5

Paso 6: formulario de registro

El formulario de autenticación es la unica vista que se crea asociada a las sesiones.

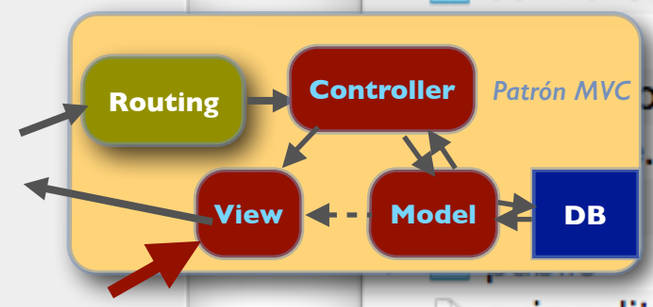
Es un formulario convencional, cuya única particularidad es que para introducir el password utiliza un elemento <input type= “password “ ….> para que el navegador no muestre lo que se teclea, ni guarde el password tecleado.

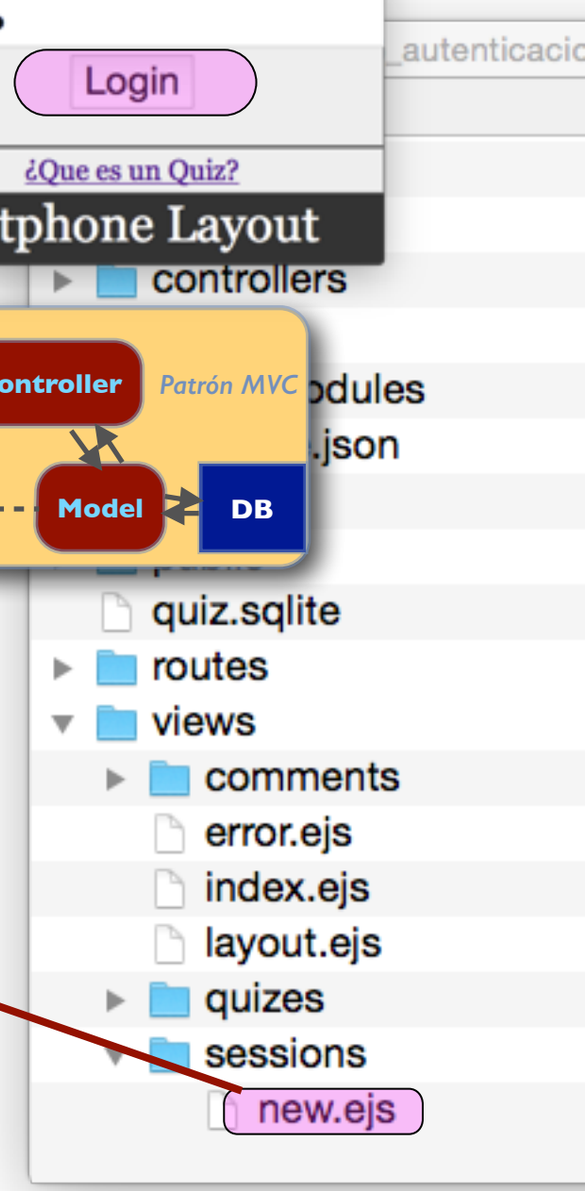


new.ejs UNREGISTERED

```
1  <h2>Login</h2>
2
3  <form method='POST' action='/login'>
4
5    <div class="field">
6      <label for="login">Login:</label><br />
7      <input type="text" id="login" name="login" size="30" />
8    </div>
9
10   <div class="field">
11     <label for="password">Password:</label><br />
12     <input type="password" id="password" name="password" size="30" />
13   </div>
14
15   <div class="actions">
16     <input name="commit" type="submit" value="Login" />
17   </div>
18
19 </form>
```

Paso 6





Final del tema

©

Quiz 17: Autorización

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

©

Quiz 17: Autorización

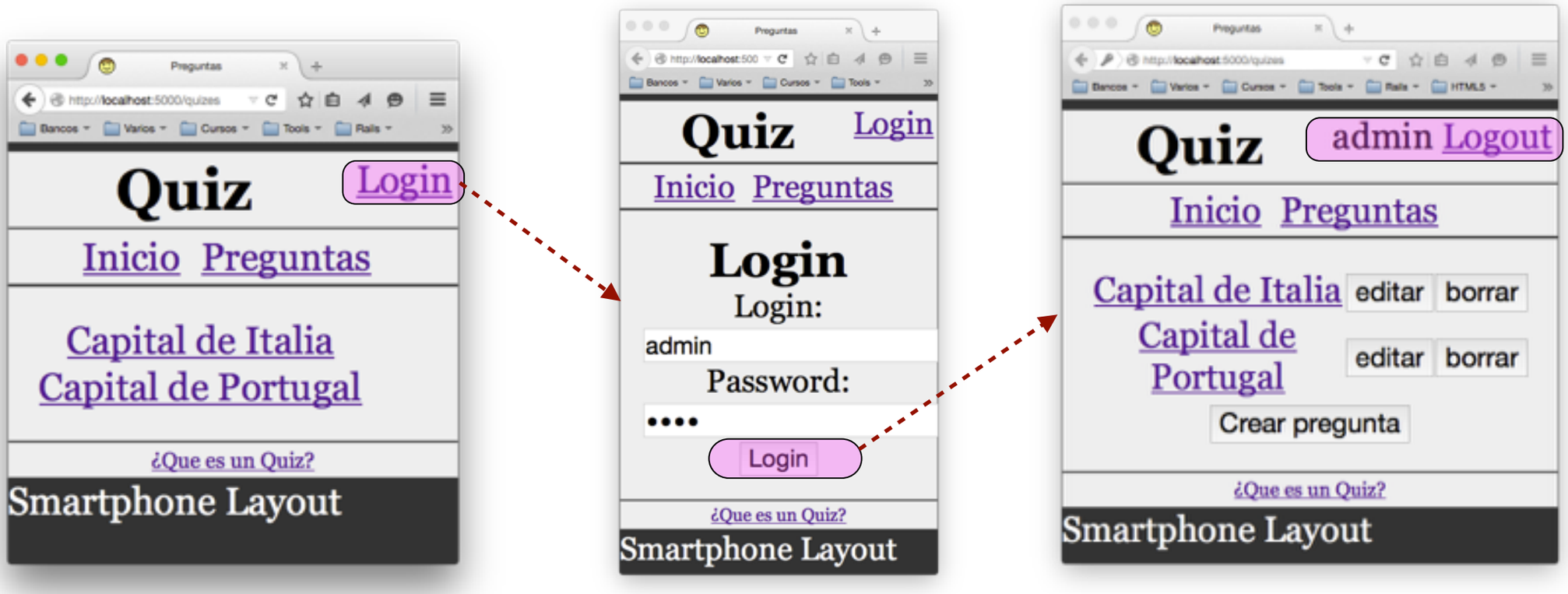
Objetivo: Añadir control de acceso a la aplicación Quiz para que solo los usuarios autenticados este autorizados para crear, editar o borrar preguntas.

✦**Paso 1**: Crear un middleware de autorización en **controllers/session_controller.js**

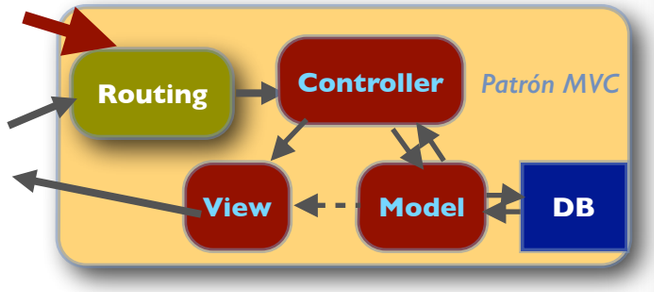
✦**Paso 2**: Añadir MW de autorización en **routes/index.js** a rutas de creación, edición y borrado

✦**Paso 3**: Modificar vista **views/quizes/index.ejs** para quitar botones a usuarios anónimos

✦**Paso 4**: Guardar **versión (commit)** git y subir a **Heroku**



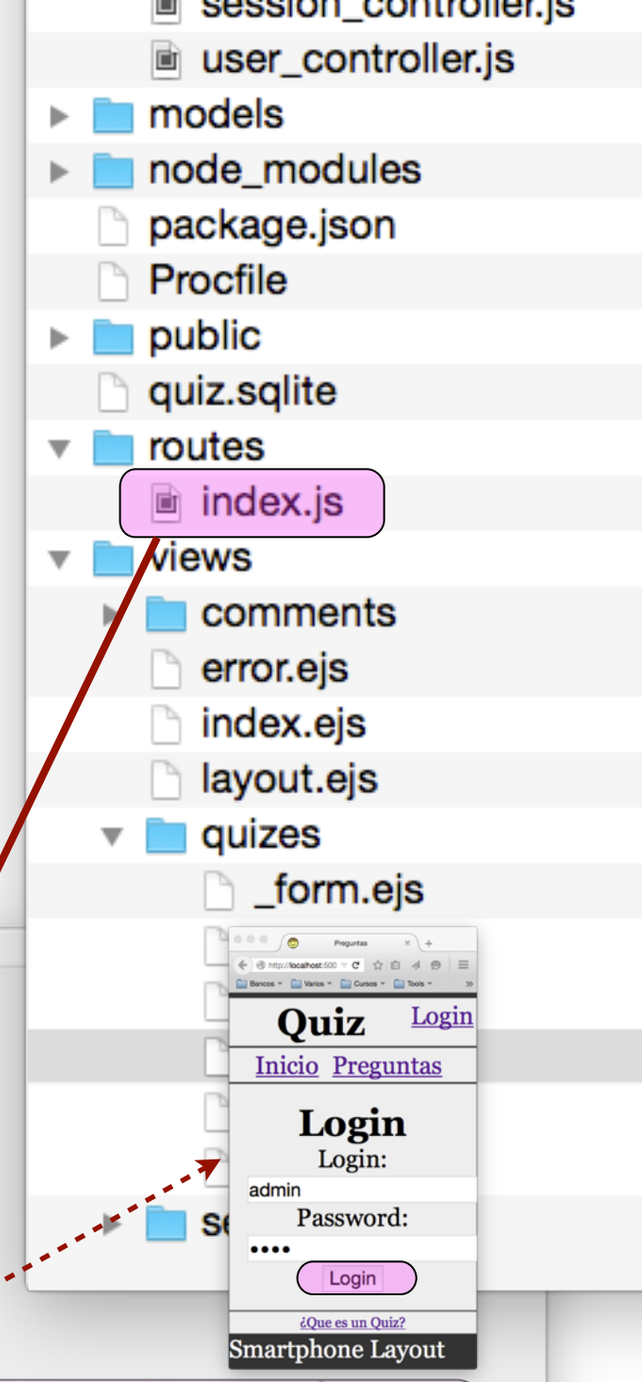
Paso 2



Una ruta puede invocarse con varios MWs en serie: "**get('/quizes/new', MW1, MW2)**". **MW1** y **MW2** se ejecutan en serie, de forma que si **MW1** no pasa control a **MW2** con **next()**, **MW2** no llegará a ejecutarse.

Si añadimos **sessionController.loginRequired** delante de los controladores de accesos que necesiten autenticación, se impide que usuarios sin sesión ejecuten operaciones de crear, editar o borrar recursos.

Aunque los botones para realizar dichas operaciones se han quitado, hay que evitar también que se puedan realizar de otras formas.



index.js

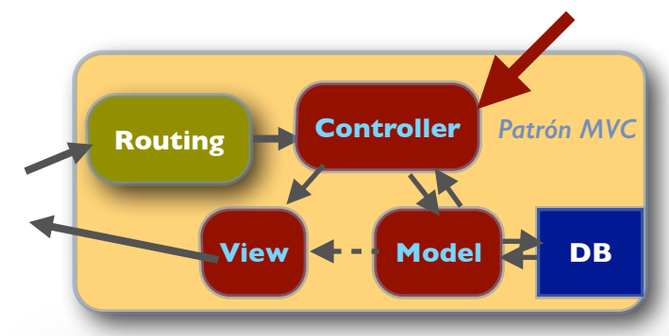
```
// Definición de rutas de sesion
router.get('/login',  sessionController.new);     // formulario login
router.post('/login', sessionController.create);  // crear sesión
router.get('/logout', sessionController.destroy); // destruir sesión

// Definición de rutas de /quizes
router.get('/quizes',                       quizController.index);
router.get('/quizes/:quizId(\\d+)',         quizController.show);
router.get('/quizes/:quizId(\\d+)/answer', quizController.answer);
router.get('/quizes/new',                   sessionController.loginRequired, quizController.new);
router.post('/quizes/create',               sessionController.loginRequired, quizController.create);
router.get('/quizes/:quizId(\\d+)/edit',    sessionController.loginRequired, quizController.edit);
router.put('/quizes/:quizId(\\d+)',         sessionController.loginRequired, quizController.update);
router.delete('/quizes/:quizId(\\d+)',      sessionController.loginRequired, quizController.destroy);

// Definición de rutas de comentarios
router.get('/quizes/:quizId(\\d+)/comments/new', commentController.new);
router.post('/quizes/:quizId(\\d+)/comments',    commentController.create);
```

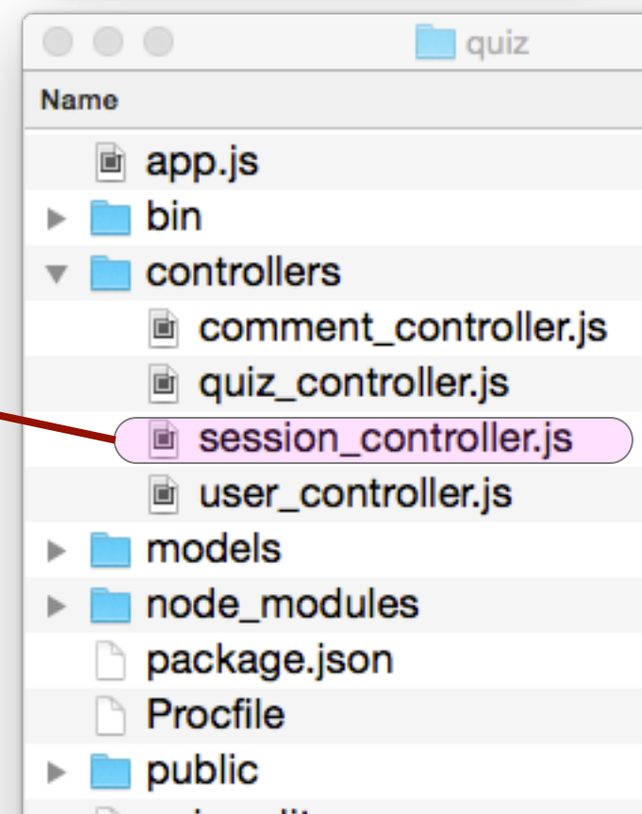
Paso 1: MW de control

Para evitar la ejecución de los middlewares de atención a las operaciones que necesitan autenticación, se añade a **session_controller.js** un nuevo middleware que solo deja continuar al usuario si está autenticado. Son solo le da acceso a la pantalla de login.

session_controller.js UNREGISTERED

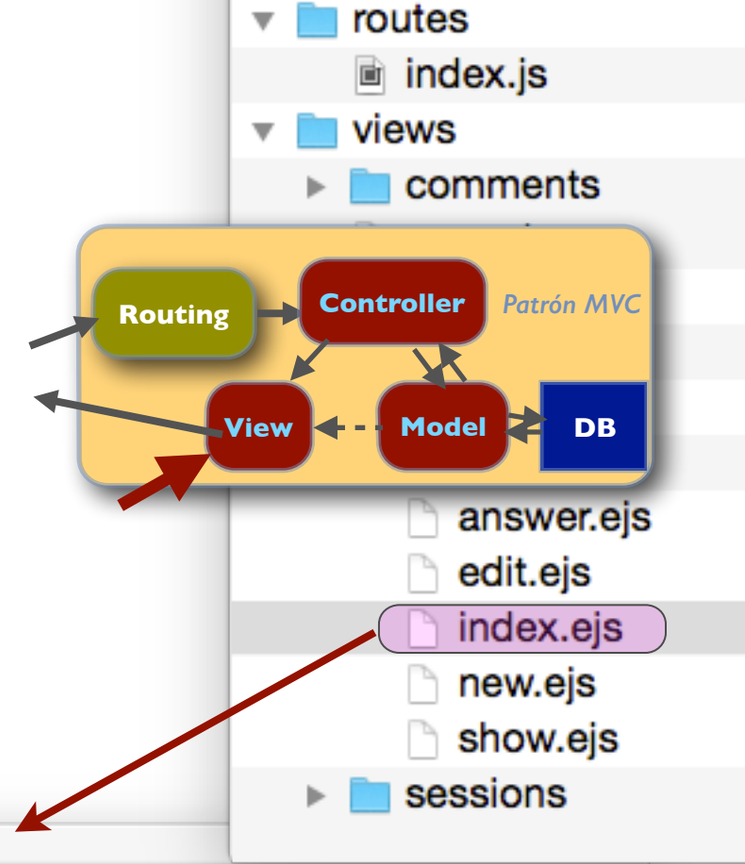
```
1  // MW de autorización de accesos HTTP restringidos
2  exports.loginRequired = function(req, res, next){
3      if (req.session.user) {
4          next();
5      } else {
6          res.redirect('/login');
7      }
8  };
9
10 // Get /login   -- Formulario de login
11 exports.new = function(req, res) {
12     var errors = req.session.errors || {};
13     req.session.errors = {};
14
15     res.render('sessions/new', {errors: errors});
16 };
17 ......
```





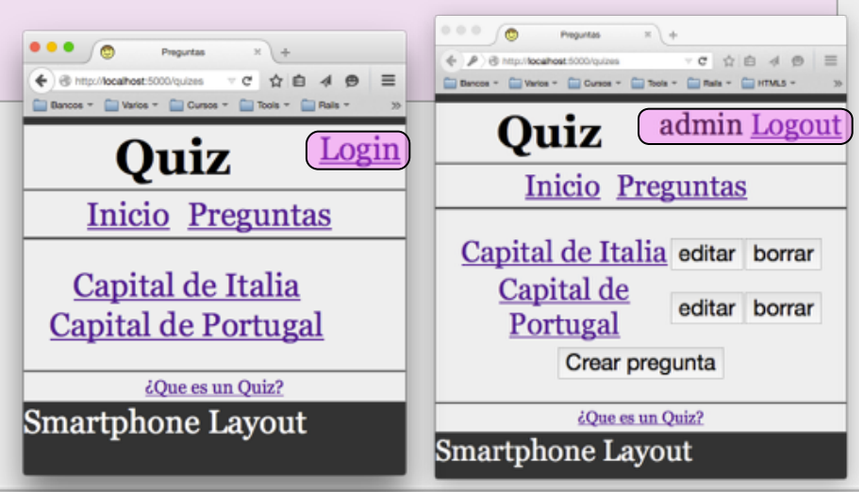
Paso 3: quitar botones

En la vista index.js con la lista de preguntas, se el código HTML de los botones se genera solo si el usuario ha establecido la sesión. Esto se consigue incluyendo dicho código en el bloque de una sentencia **if (session.user) { .... }** de EJS, tal y como se muestra.



index.ejs

```
1  <table>
2    <% var i; for (i=0; i < quizes.length; i++) { %>
3      <tr>
4        <td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td>
5        <%if(session.user){%>
6          <td><a href="quizes/<%= quizes[i].id %>/edit"><button>editar</button></a></td>
7          <td>
8            <form method="post" action="quizes/<%= quizes[i].id %>?_method=delete">
9              <button type="submit" onClick="return confirm('Borrar: <%= quizes[i].pregunta %>');">
10               borrar
11             </button>
12           </form>
13         </td>
14       <%}%>
15     </tr>
16   <% } %>
17 </table>
18 <p/>
19 <%if(session.user){%>
20   <a href="/quizes/new"><button>Crear pregunta</button></a>
21 <%}%>
```



Final del tema

©

Quiz 18: Moderación de Comentarios

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

Quiz 18: Moderación de Comentarios

Objetivo: Añadir moderación de comentarios a la aplicación Quiz de forma que los comentarios no se publiquen hasta que un usuario autenticado lo autorice.

✦**Paso 1**: Añadir campo de **publicado** en tabla **models/comment.js** de la DB

✦**Paso 2**: Añadir autoload y acción publish en **controllers/comment_controller.js**

- a: Añadir método de Autoload de comentarios en controlador
- b: Añadir acción **publish** para publicar comentarios una vez autorizados

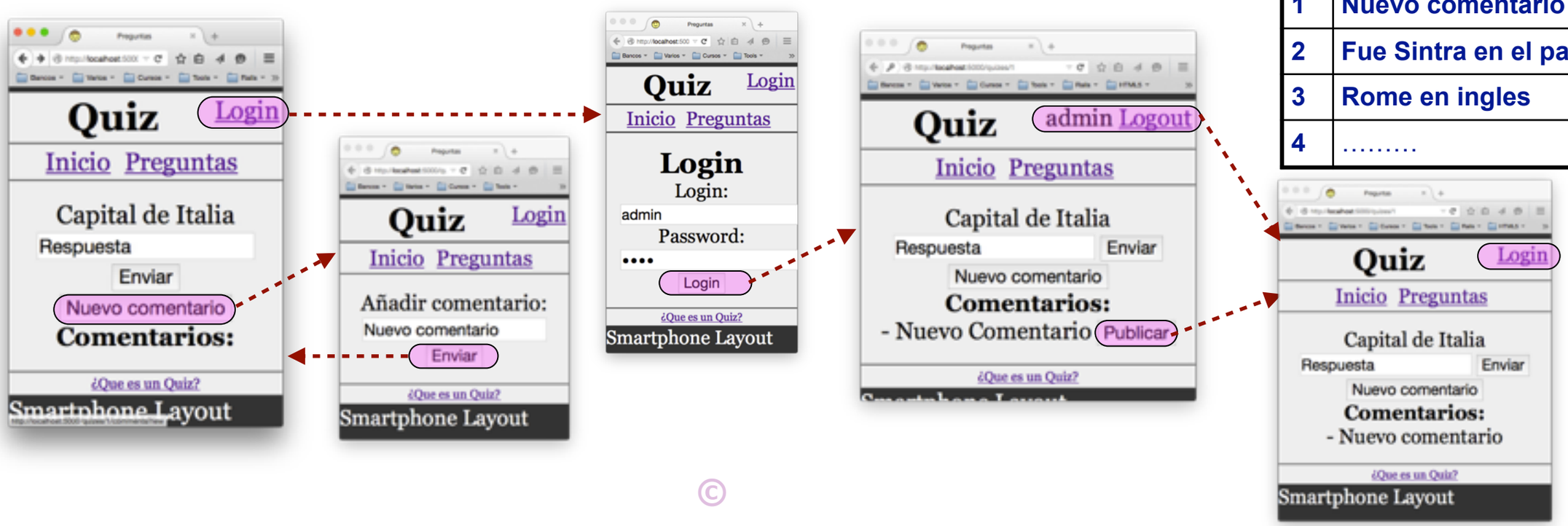
✦**Paso 3**: Instalar middleware de autoload y nueva ruta en **routes/index.js**

- a: Instalar middleware de Autoload de comentarios en **routes/index.js**
- b: Añadir en **routes/index.js** ruta **GET /quizes/:quizId/comments/:id/publish** para autorizar comentario

✦**Paso 4**: Modificar **views/quizes/:quizId/show** para mostrar solo comentarios autorizados

✦**Paso 6**: Guardar **versión (commit)** git y subir a **Heroku**

id	texto	publicar
1	**Nuevo comentario**	TRUE
2	**Fue Sintra en el pasado**	FALSE
3	**Rome en ingles**	TRUE
4	……	



©

Paso 1: Tabla Comment

Fichero **quiz.sqlite** contiene los datos, en nuestro caso las preguntas.

quiz.sqlite se puede borrar para regenerar o copiar para hacer un backup.

comment.js define el formato de la tabla de comentarios.

Tiene solo 1 campo de tipo string:

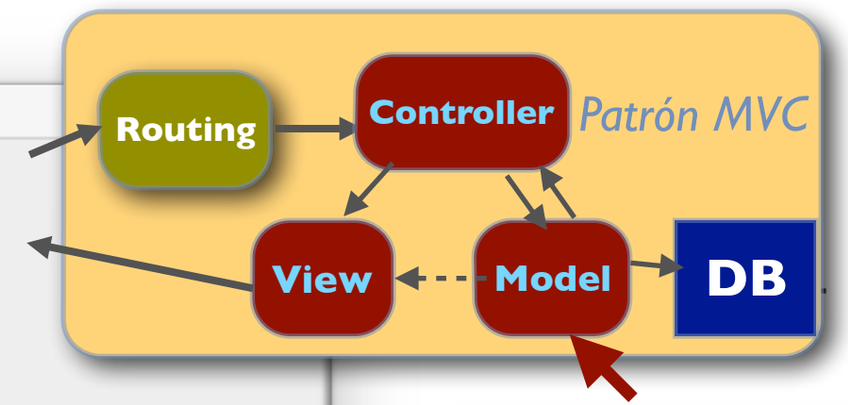
- texto: DataTypes.STRING

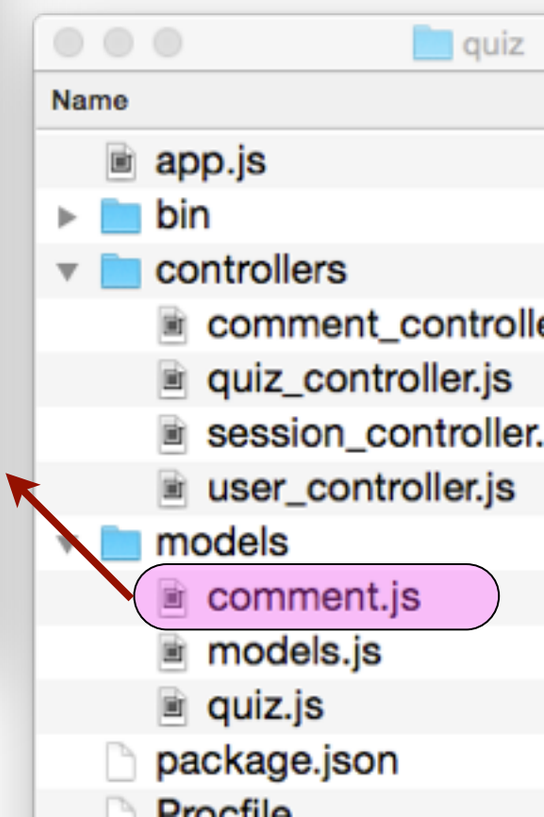
comment.js

```
1  // Definicion del modelo de Quiz con validación
2
3  module.exports = function(sequelize, DataTypes) {
4    return sequelize.define(
5      'Comment',
6      { texto: {
7          type: DataTypes.STRING,
8          validate: { notEmpty: {msg: "-> Falta Comentario"}}
9        },
10       publicado: {
11         type: DataTypes.BOOLEAN,
12         defaultValue: false
13       }
14     }
15   );
16 }
```

Tabla Comment

id	texto	publicar
1	Nuevo comentario	TRUE
2	Fue Sintra en el pasado	FALSE
3	Rome en ingles	TRUE
4	.....	

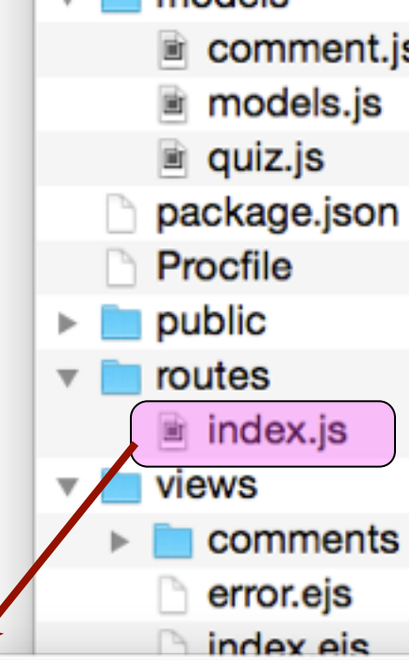




Pasos 3a y 3b

Autoload se instala con **router.param('commentId', commentController.load)**.
El comentario estará accesible cuando cuando la accón publish se ejecute:

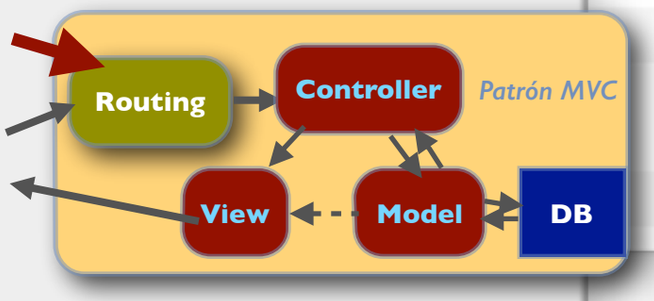
GET /quizes/:quizId/comments/:commentId/publish
-> GET funciona, pero el interfaz uniforme indica usar PUT en este caso



index.js UNREGISTERED

Paso 3a

```
13 // Autoload de comandos con :quizId
14 router.param('quizId', quizController.load);  // autoload :quizId
15 router.param('commentId', commentController.load);  // autoload :commentId
16
17 // Definición de rutas de sesion
18 router.get('/login',  sessionController.new);     // formulario login
19 router.post('/login', sessionController.create);  // crear sesión
20 router.get('/logout', sessionController.destroy); // destruir sesión
21
22 // Definición de rutas de /quizes
23 router.get('/quizes',                      quizController.index);
24 router.get('/quizes/:quizId(\\d+)',        quizController.show);
25 router.get('/quizes/:quizId(\\d+)/answer', quizController.answer);
26 router.get('/quizes/new',                  sessionController.loginRequired, quizController.new);
27 router.post('/quizes/create',              sessionController.loginRequired, quizController.create);
28 router.get('/quizes/:quizId(\\d+)/edit',   sessionController.loginRequired, quizController.edit);
29 router.put('/quizes/:quizId(\\d+)',        sessionController.loginRequired, quizController.update);
30 router.delete('/quizes/:quizId(\\d+)',     sessionController.loginRequired, quizController.destroy);
31
32 // Definición de rutas de comentarios
33 router.get('/quizes/:quizId(\\d+)/comments/new', commentController.new);
34 router.post('/quizes/:quizId(\\d+)/comments',    commentController.create);
35 router.get('/quizes/:quizId(\\d+)/comments/:commentId(\\d+)/publish',
36                                  sessionController.loginRequired, commentController.publish);
37
38 module.exports = router;
```



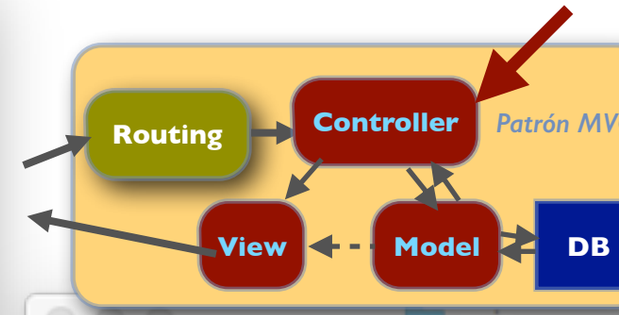
Paso 3b

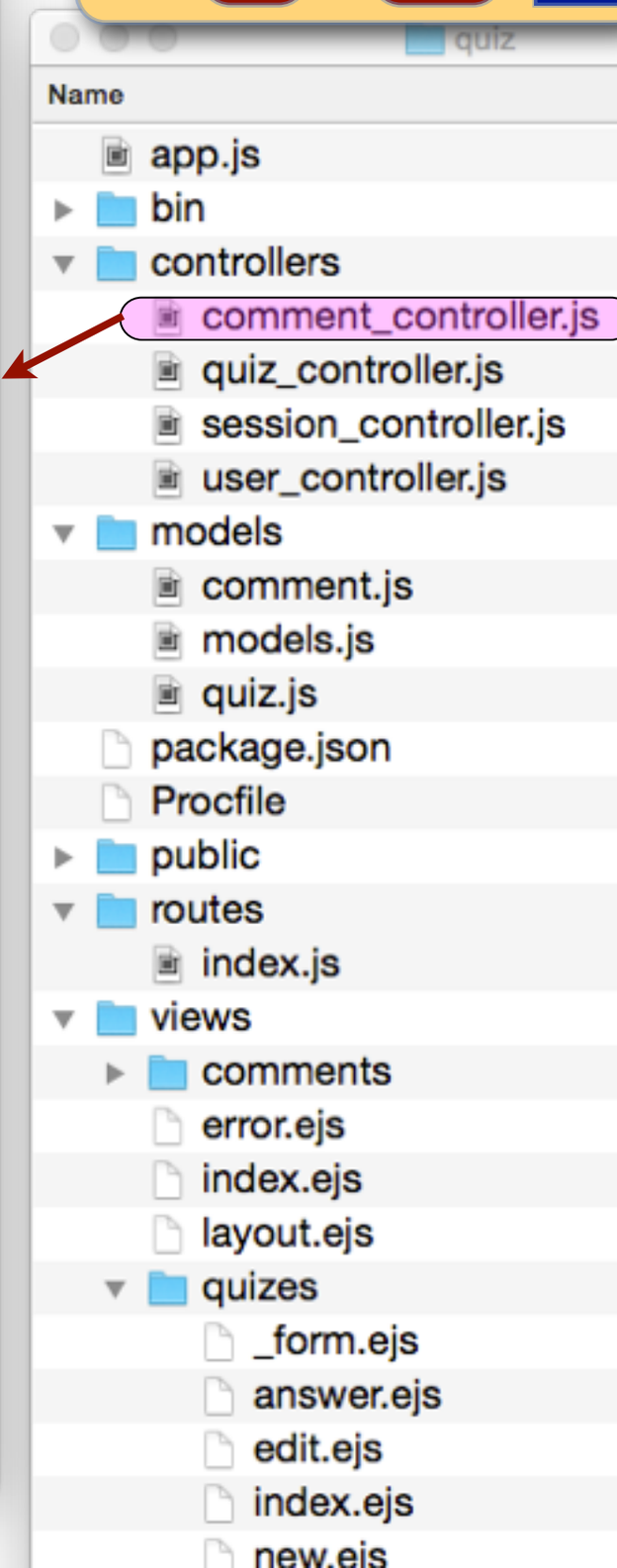
Pasos 2a y 2b

comment_controller.js UNREGISTERED

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

// Autoload :id de comentarios
exports.load = function(req, res, next, commentId) {
  models.Comment.find({
            where: {
                id: Number(commentId)
            }
        }).then(function(comment) {
      if (comment) {
        req.comment = comment;
        next();
      } else{next(new Error('No existe commentId=' + commentId))}
    }
  ).catch(function(error){next(error)});
};

// GET /quizes/:quizId/comments/new
exports.new = function(req, res) {
  res.render('comments/new.ejs', {quizid: req.params.quizId, errors: []});
};

// POST /quizes/:quizId/comments
exports.create = function(req, res) {
  .........
};

// GET /quizes/:quizId/comments/:commentId/publish
exports.publish = function(req, res) {
  req.comment.publicado = true;

  req.comment.save( {fields: ["publicado"]})
    .then( function(){ res.redirect('/quizes/'+req.params.quizId);} )
    .catch(function(error){next(error)});

  };
```

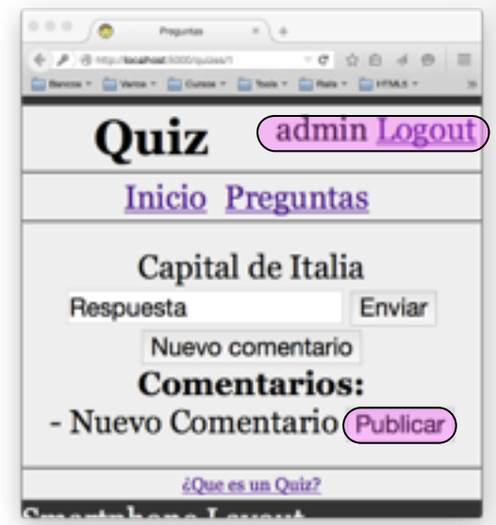
La ruta de la acción publish lleva :comentId y necesita autoload

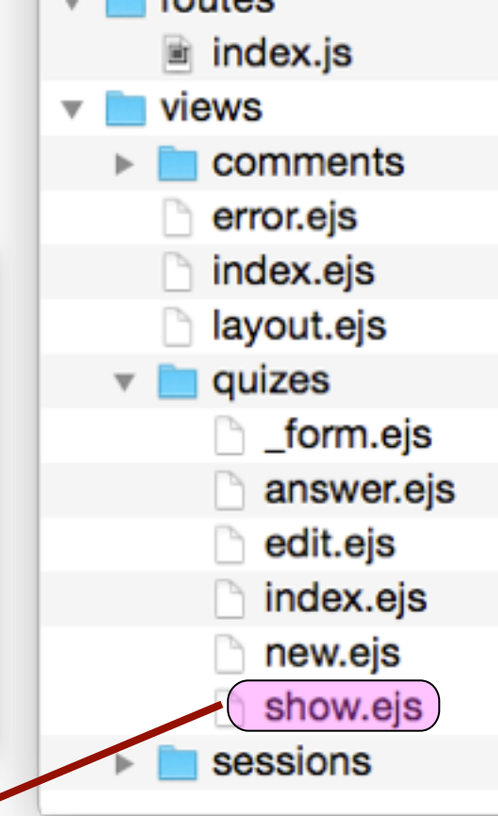




Paso 4

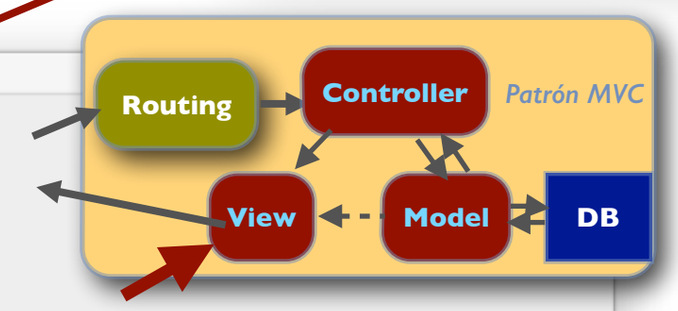
Por último hay que modificar la vista views/quizes/show.ejs para que solo muestre los comentarios publicados si el usuario no está autenticado.
Si está autenticado mostrará también los comentarios sin autorizar, con el botón de publicar al lado.





show.ejs

```
<form method="get" action="/quizes/<%= quiz.id %>/answer">
  <%= quiz.pregunta %> <p>
  <input type="text" name="respuesta" value="Respuesta"/>
  <input type="submit" value="Enviar">
</form>

<p><a href="/quizes/<%= quiz.id %>/comments/new"><button>Nuevo comentario</button></a></p>

<p><strong>Comentarios:</strong></p>
<%for(index in quiz.Comments){%>
    <%if(quiz.Comments[index].publicado || session.user){%>
        <p>- <%=quiz.Comments[index].texto%>
            <%if(session.user && !quiz.Comments[index].publicado){%>
                <a href="/quizes/<%= quiz.id %>/comments/<%=quiz.Comments[index].id%>/publish">
                                                    <button>Publicar</button></a>
            <%}%>
        </p>
    <%}%>
<%}%>
```



Final del tema

©

Quiz 19: HTTPS - HTTP Seguro

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

Quiz 19: HTTPS - HTTP Seguro

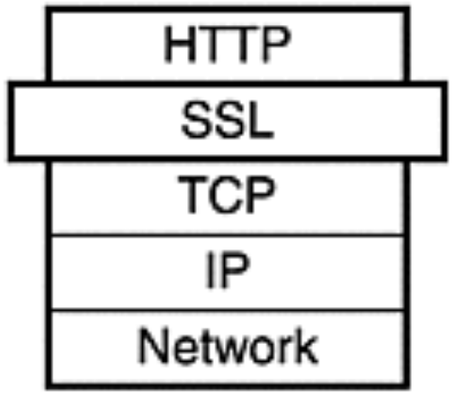
Objetivo: Añadir acceso HTTP Seguro a la aplicación Quiz para evitar que las contraseñas de acceso se envíen en claro y añadir así mayor seguridad al uso del portal.

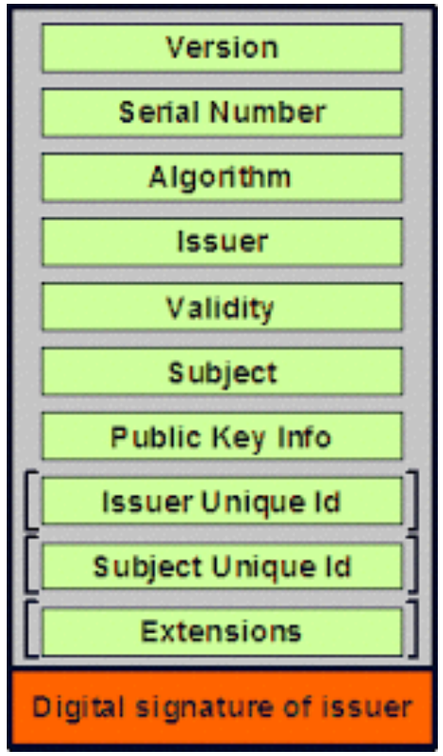
✦**Paso 1**: Añadir acceso por HTTPS al arrancar el servidor de node.js en **bin/www**

✦**Paso 2**: Crear e instalar claves y certificados digitales para acceso seguro HTTPS

- a: Crear script de creación e instalación de certificados digitales
- b: Crear e instalar certificados

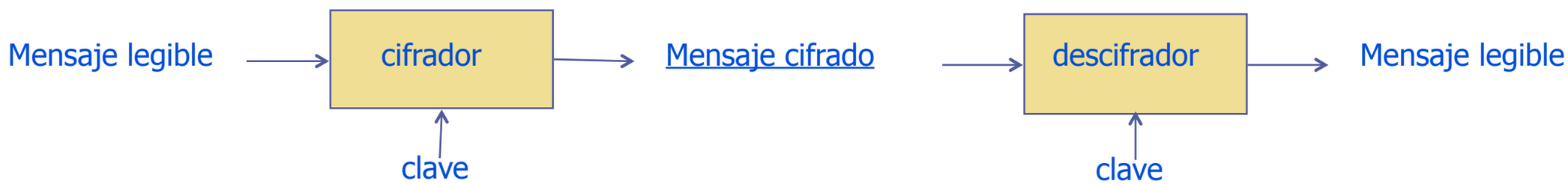
✦**Paso 3**: Guardar **versión (commit)** git y subir a **Heroku**



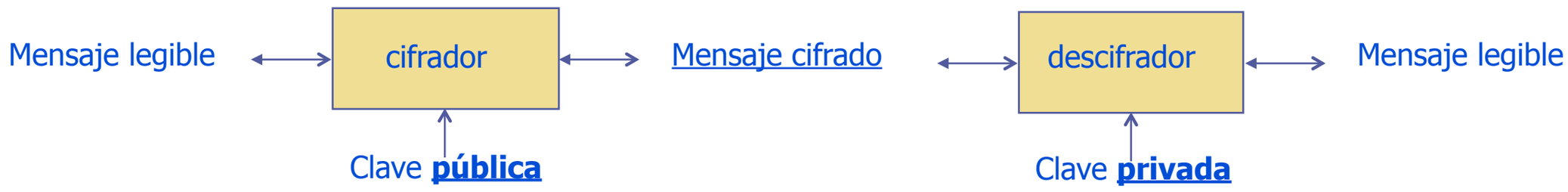


Clave pública y clave privada

- Clave simétrica
  - Emisor y receptor necesitan clave para cifrar y descifrar

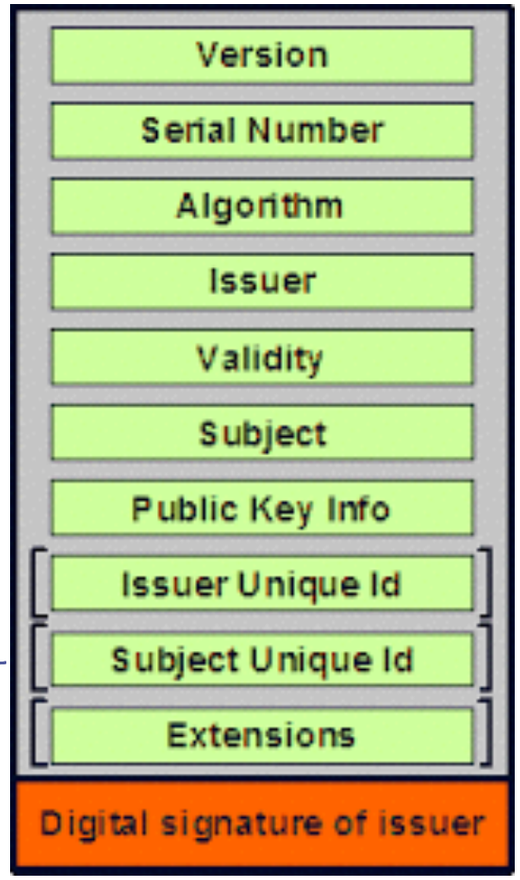


- Clave asimétrica (publica + privada)
  - Todos conocen la clave pública
    - Cualquiera puede enviarle un mensaje encriptado
  - El receptor conoce solamente la clave privada
    - Solamente el receptor puede desencriptar los mensajes



Certificado y autoridad certificadora

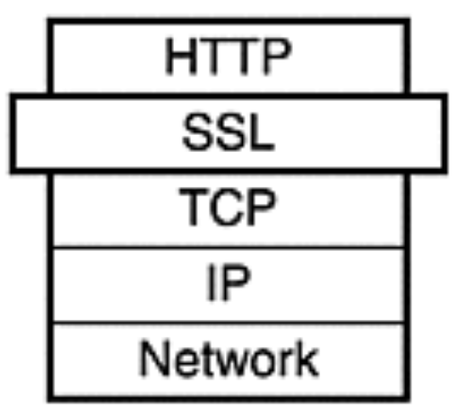
- Las **infraestructuras de claves públicas X509** se basan en
  - **Autoridades Certificadoras** (**AC**s) que garantizan la autenticidad de un sujeto
    - Emiten **certificados X509** con una clave pública que permiten comprobar autenticidad
  - La configuración de la clave pública necesita 3 ficheros: **Clave**, **CSR** y **Certificado X509**
- **Clave:** clave privada para cifrar y firmar -> **quiz-2015-key.pem**
  - Debe ser guardado celosamente por el sujeto o la AC
- **CSR** (Certificate signing request): -> **quiz-2015-csr.pem**
  - Solicitud a AC de un certificado X509 temporal firmado
- **Certificado X509:** para cifrar o autenticar -> **quiz-2015-cert.pem**
  - Certificado con clave pública emitido por AC o "auto-firmado"
- En Quiz utilizamos un **certificado "auto-firmado"** de 1 año de validez
  - Los navegadores lo detectarán y notificarán al usuario

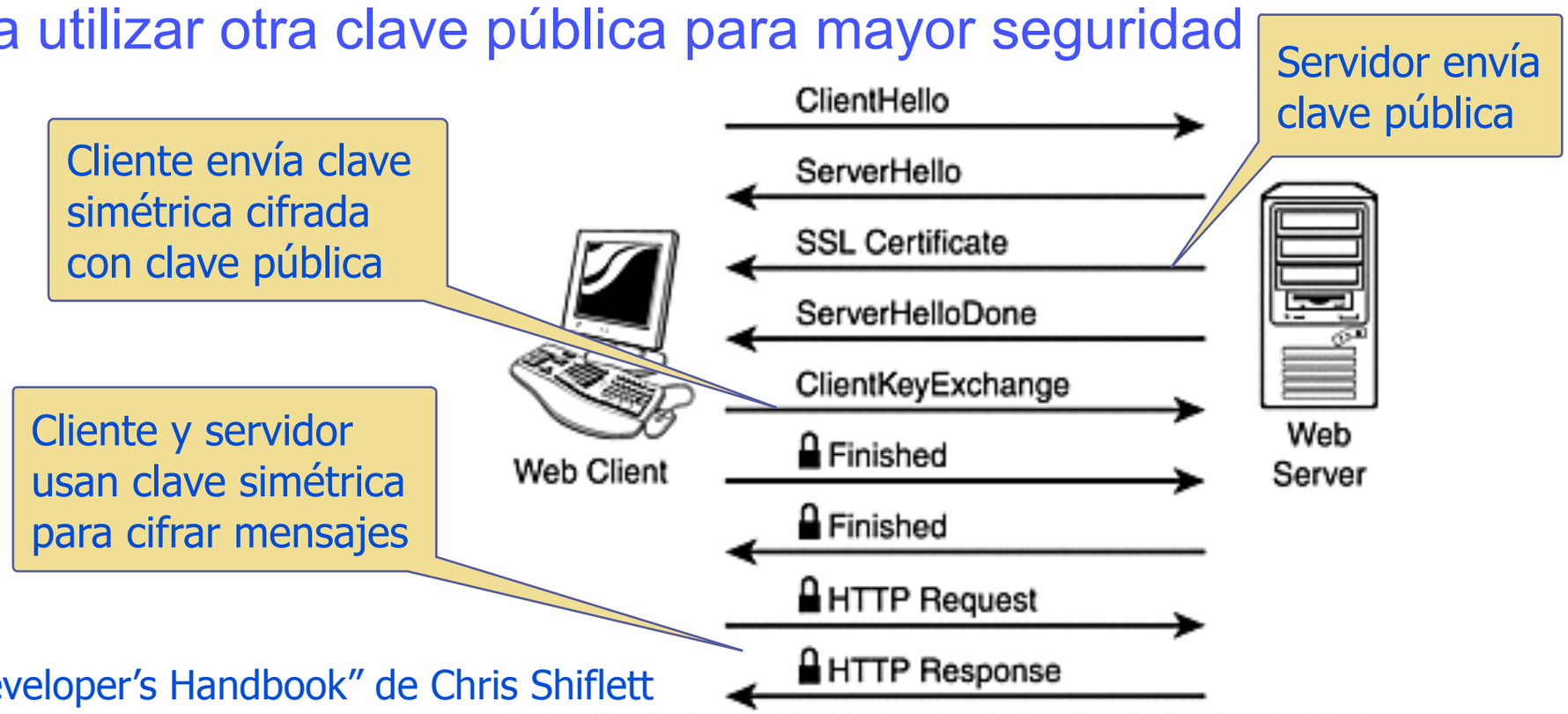


Estructura de clave púbica X509

HTTPS: Conexiones seguras

- HTTPS introduce el nivel SSL de cifrado entre HTTP y Sockets
  - Este nivel cifra la información que intercambian cliente y servidor
    - HTTPS se activa automaticamente si el URL lo indica: **https://dit.upm.es/……**
- HTTPS se suele utilizar solo con la clave pública del servidor
  - El **servidor envía su certificado al cliente**, que incluye clave pública
    - El **cliente crea una clave simétrica** y la envía al servidor, cifrada con la **clave pública**
      - La clave simétrica es mucho mas eficiente y se utiliza para cifrar la información
  - El cliente podría utilizar otra clave pública para mayor seguridad

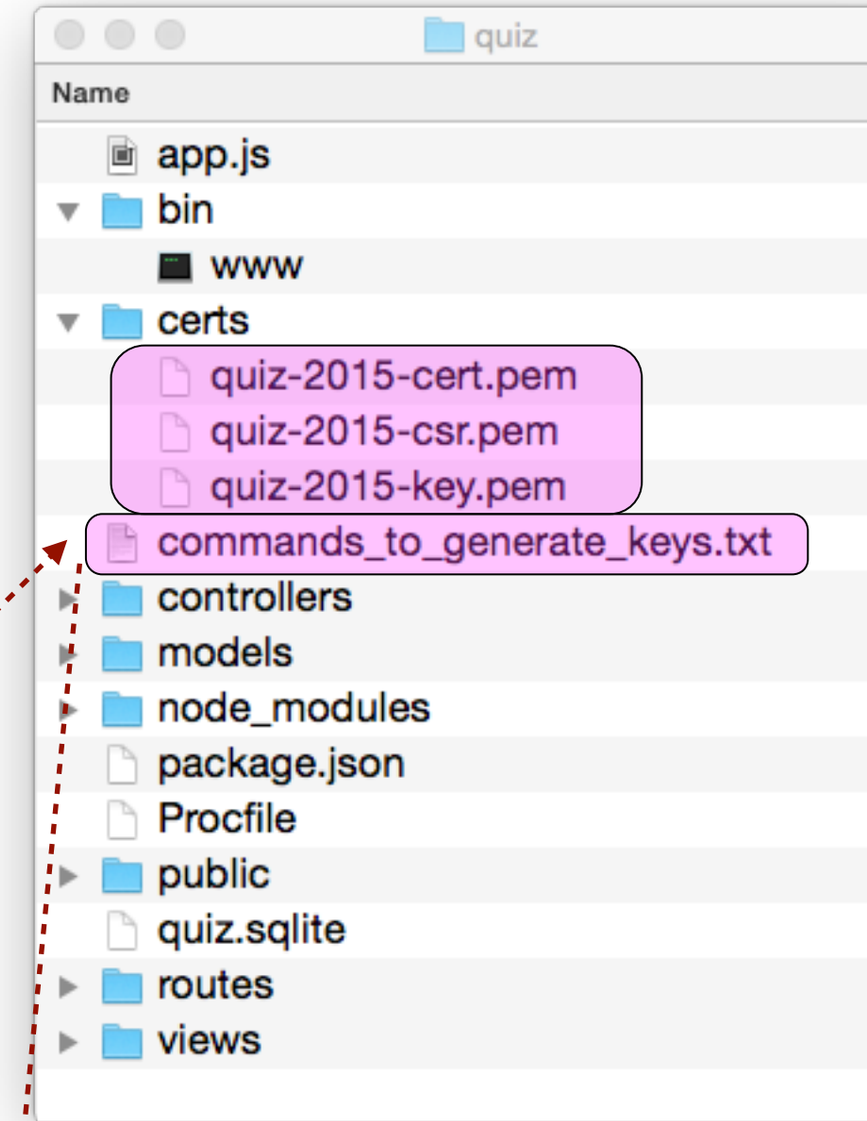




Figuras tomada de "HTTP Developer's Handbook" de Chris Shiflett

Generación del certificado

- **OpenSSL** es una herramienta criptográfica muy potente
  - Implementa clientes SSL y TLS
    - además de otras funciones y librerías criptográficas
- Incluimos comando shell de UNIX (genera los 3 ficheros)
  - **Clave privada** para cifrar y firmar: **quiz2015-key.pem**
  - **Solicitud** de firma de certificado: **quiz2015-csr.pem**
    - OpenSSL pide datos al generar CSR "auto-firmado"
      - Poner datos ficticios
  - **Certificado** en formato X509: **quiz2015-cert.pem**



commands_to_generate_keys.txt UNREGISTERED

```
mkdir cert
cd cert
openssl genrsa -out quiz-2015-key.pem 2048
openssl req -new -sha256 -key quiz-2015-key.pem -out quiz-2015-csr.pem
openssl x509 -req -in quiz-2015-csr.pem -signkey quiz-2015-key.pem -out quiz-2015-cert.pem
```

Programa de arranque

Una vez creadas las claves y los certificados, se modifica el fichero **bin/www** para que use HTTPS.

Se importan los módulos **fs** y **http** de El módulo **fs** permute leer los ficheros asociados a la variable options con la clave privada y el certificado auto-firmado. **http** permite instalar HTTPS.
Documentación módulo https: http://nodejs.org/api/https.html
Instalación HTTPS en express.js: http://expressjs.com/4x/api.html#app.listen

Por último se instala HTTPS en **app**, incluyendo clave privada y certificado, y se conecta al puerto 8443.

www UNREGISTERED

```
1  #!/usr/bin/env node
2  var debug = require('debug')('quiz');
3  var app = require('../app');
4  var fs = require("fs");
5  var https = require("https");
6
7  var options = {
8      key: fs.readFileSync('certs/quiz-2015-key.pem').toString(),
9      cert: fs.readFileSync('certs/quiz-2015-cert.pem').toString()
10 };
11
12 app.set('port', process.env.PORT || 3000);
13
14 var server = app.listen(app.get('port'), function() {
15   debug('Express server listening on port ' + server.address().port);
16 });
17
18 var serverSecure = https.createServer(options, app);
19 serverSecure.listen(8443, function() {
20   debug('Express server listening on port ' + server.address().port);
21 });
22
```

quiz

Name

- app.js
- bin
  - www
- certs
  - quiz-2015-cert.pem
  - quiz-2015-csr.pem
  - quiz-2015-key.pem
- commands_to_generate_keys.txt
- controllers
- models
- node_modules
- package.json
- Procfile
- public
- quiz.sqlite
- routes
- views

Final del tema

Ejercicio: Añadir autologout

Añadir a la versión Quiz 19 “HTTPS” una funcionalidad de “autologout”. Esta nueva funcionalidad debe desconectar la sesión del usuario si este no realiza ninguna actividad durante un periodo de 2 minutos.

Se recomienda añadir los siguiente en en el máximo nivel de la aplicación (app.js)

- Un nuevo evento que ejecute periodicamente
- un middleware que cuente los hay que añadir un reloj que cuente el tiempo desde periódico que La página se enlazara desde la barra de navegación con un enlace adicional, de nombre autor.

La página incluirá el nombre de los 2 autores (miembros de la pareja), una foto y un pequeño video de presentación del portal.

Desplegar dicha versión en Heroku y entregar el enlace a dicha versión por Moodle.

Recomendación: Hacer una rama en un clon del proyecto Quiz desde Quiz 6, que añada dicha funcionalidad y desplegar dicha rama en Heroku.

Ejercicio Quiz

Añadir Para esta entrega se debe añadir a la versión 7 de la aplicación Quiz la capacidad de borrar preguntas de la base de datos. La version 7 es la que permite responder a preguntas, crear nuevas preguntas y editarlas.

La funcionalidad hay que añadirla conectando al enlace "Borrar” ya incluido en la barra de navegación (delimitada con la marca “<nav>” en el marco o layout).

“Borrar” nos llevará a una primera pantalla que nos permite seleccionar la pregunta que queremos borrar, de forma que una vez seleccionada borre la pregunta de la base de datos cuando se pulse un botón de “Borrar” que debe existir en dicha vista.

La implementación deberá pedir confirmación antes de borrar con un pequeño desplegable. En caso de confirmar debe borrar la pregunta y volver a la vista “index”. En caso de no confirmar, deberá volver a index solamente.

Debe entregarse un fichero zip con todo el desarrollo realizado. Por supuesto deben borrarse los directorios y ficheros de instalaciones y datos como: node_modules, quiz.sqlite, ...

Quiz 20: Tabla de usuarios

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

Quiz 20: Tabla de Usuarios

Objetivo: Añadir la **tabla User** que almacenará el nombre de cada usuario, su palabra de paso (password) y si es administrador o no.

La **palabra de paso** deberá guardarse **cifrada** en la DB para **evitar robo de identidad**.

La **tabla User** tendra una **relación 1-N** con la **tabla Quiz**

- **Paso 1**: Definir la tabla User
- **Paso 2:** Definir la relación 1-N entre la tabla User y la tabla Quiz
- **Paso 3:** Inicializar la base de datos con usuarios y quizes actualizados
- **Paso 4:** Rehacer **user_controller.js** para que autentique desde la DB
- **Paso 4:** Rehacer **quiz_controller.js** para gestionar los enlaces de la relación
- **Paso 5**: Guardar versión (commit) con git

id	username	password	isAdmin
1	admin	r34et5690	TRUE
2	pepe	56gh90op	FALSE
3	.....	.....	……

id	pregunta	resp.	UserId
1	Capital de Italia	Roma	2
2	Capital de Portu.	Lisboa	2
3	.....	.....	
4	.....	.....	

id	texto	publicar	QuizId
1	Nuevo coment	TRUE	1
2	Fue Sintra en el pasado	FALSE	2
3	Inglés: Rome	TRUE	1
4	……		

©

```
1  // Modelo de User con validación y encriptación de passwords
2  var crypto = require('crypto');
3  var key = process.env.PASSWORD_ENCRYPTION_KEY;
4
5  module.exports = function(sequelize, DataTypes) {
6    var User = sequelize.define(
7      'User',
8      {
9        username: {
10         type: DataTypes.STRING,
11         unique: true,
12         validate: { ⊟
26           .....
27         }
28       },
29       password: {
30         type: DataTypes.STRING,
31         validate: { notEmpty: {msg: "-> Falta password"}},
32         set: function (password) {⊟
40           .....
41       }
42       },
43       isAdmin: {
44         type: DataTypes.BOOLEAN,
45         defaultValue: false
46       }
47     },
48     {
49       instanceMethods: {
50         verifyPassword: function (password) {
51           var encripted = crypto⊟
55           ....
56           return encripted === this.password;
57         }
58       }
59     }
60   );
61   return User;
62 }
```

.env — quiz20_tabla_usuarios

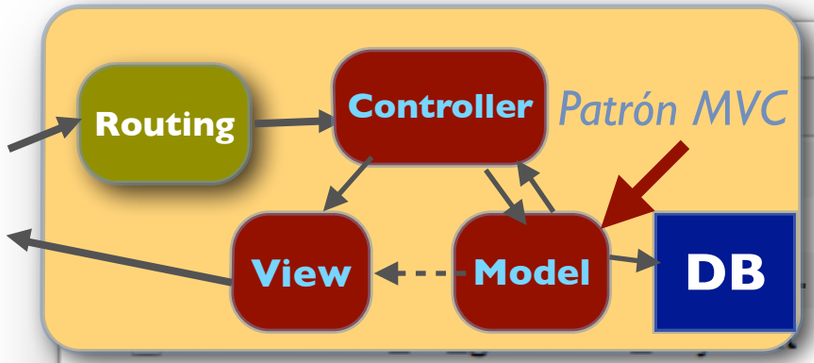
.env

```
1  DATABASE_URL=sqlite://:@:/
2  DATABASE_STORAGE=quiz.sqlite
3  PASSWORD_ENCRYPTION_KEY=asdfghjklzxcvbnmqwert
```

Campos de la tabla de usuarios:
- username
- password
- isAdmin

id	username	password	isAdmin
1	admin	r34et5690y	TRUE
2	pepe	56gh90op5	FALSE
3	.....	.....	.....

Método de instancia verifyPassword (pwd) usado en user_controller.js



- controllers
  - comment_controller.js
  - quiz_controller.js
  - session_controller.js
  - user_controller.js
- models
  - comment.js
  - models.js
  - quiz.js
  - user.js
- package.json
- Procfile
- public
- routes
- views

Paso 1: tabla User

© Juan Quemada - UPM DIT

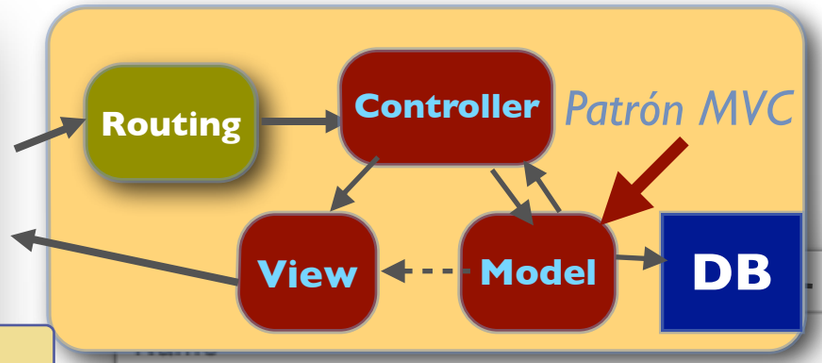
Tabla User: campo username

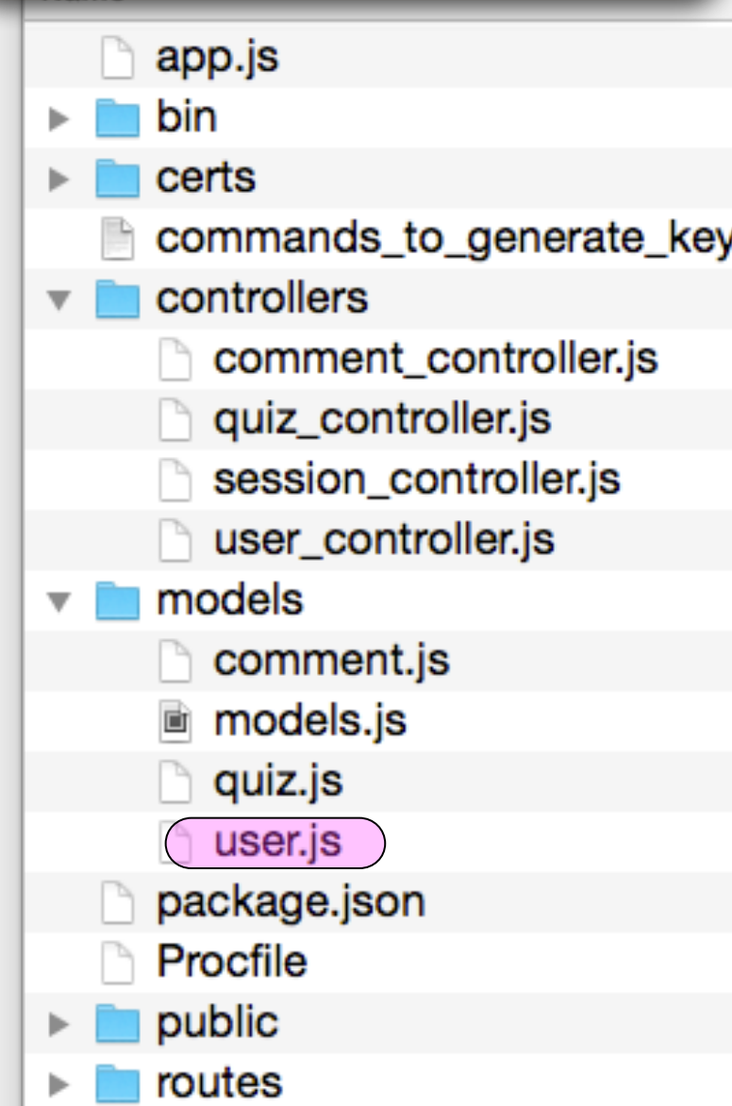
id	username	password	isAdmin
1	admin	r34et5690y	TRUE
2	pepe	56gh90op5	FALSE
3	.....	.....	.....

user.js UNREGISTERED

```
1  // Modelo de User con validación y encriptación de passwords
2  var crypto = require('crypto');
3  var key = process.env.PASSWORD_ENCRYPTION_KEY;
4
5  module.exports = function(sequelize, DataTypes) {
6    var User = sequelize.define(
7      'User',
8      {
9        username: {
10         type: DataTypes.STRING,
11         unique: true,
12         validate: {
13           notEmpty: {msg: "-> Falta username"},
14           // -> devuelve mensaje de error si username ya existe
15           isUnique: function (value, next) {
16             var self = this;
17             User
18             .find({where: {username: value}})
19             .then(function (user) {
20               if (user && self.id !== user.id) {
21                 return next('Username ya utilizado');
22               }
23               return next();
24             }).catch(function (err) {return next(err);});
25           }
26         }
27       },
28       password: {
```

Campo único, el método **isUnique (value, next)** comprueba que no hay otro usuario con el mismo id





© Juan Quemada - UPM-DIT

Tabla User: campos password e isAdmin

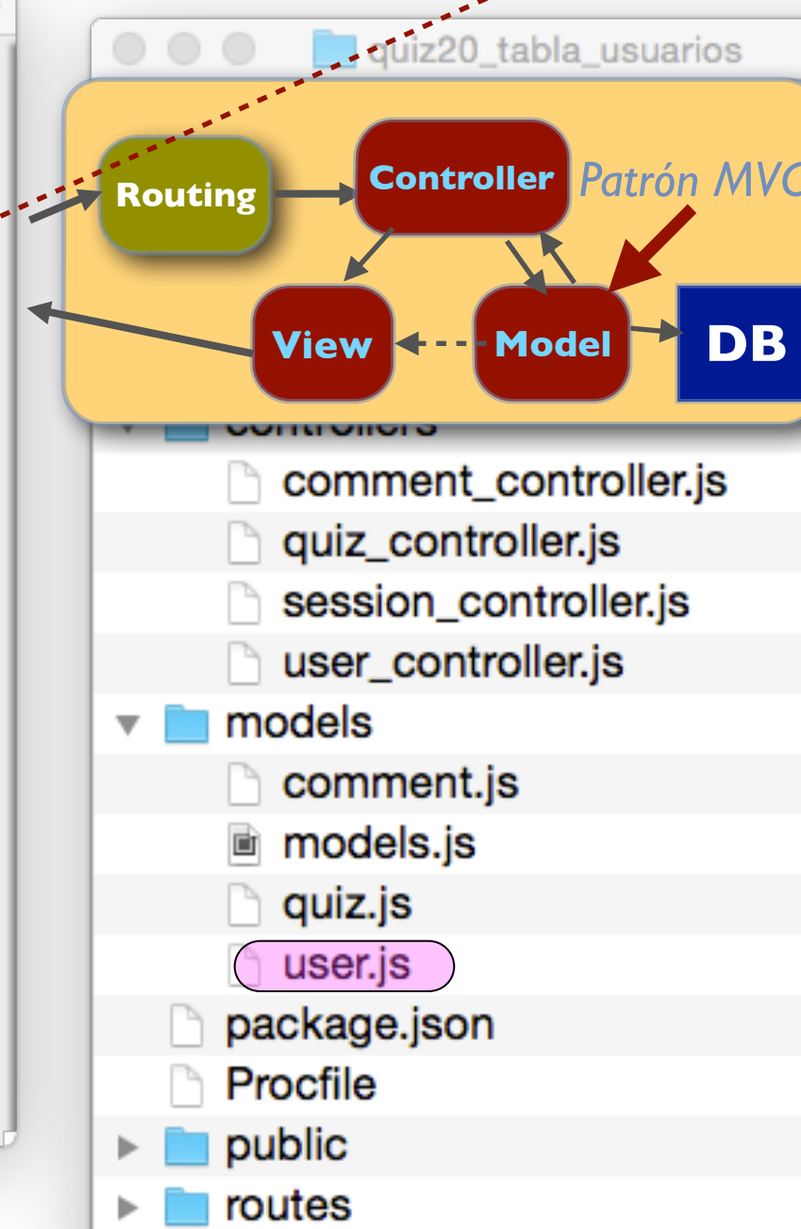
id	username	password	isAdmin
1	admin	r34et5690y	TRUE
2	pepe	56gh90op5	FALSE
3	.....	.....	.....

El campo **password** se cifra con un algoritmo normalizado utilizando la función set(password).

user.js UNREGISTERED

```
 password: {
    type: DataTypes.STRING,
    validate: { notEmpty: {msg: "-> Falta password"}},
    set: function (password) {
      var encripted = crypto
                      .createHmac('sha1', key)
                      .update(password)
                      .digest('hex');
      // Evita passwords vacíos
      if (password === '') {encripted = '';}
      this.setDataValue('password', encripted);
    }
  },
  isAdmin: {
    type: DataTypes.BOOLEAN,
    defaultValue: false
  }
},
```

El campo **isAdmin** índica si el usuario es administrador. Este campo se rellena solo al **inicializar la DB**.



© Juan Quemada - UPM-DIT

Paso 2: relación User - Quiz

id	username	password	isAdmin
1	admin	r34et5690	TRUE
2	pepe	56gh90op	FALSE
3	.....	.....	.....

id	pregunta	resp.	UserId
1	Capital de Italia	Roma	2
2	Capital de Portu.	Lisboa	2
3	.....	.....	
4	.....	.....	

id	texto	publicar	QuizId
1	Nuevo coment	TRUE	1
2	Fue Sintra en el pasado	FALSE	2
3	Inglés: Rome	TRUE	1
4	.....		

models.js

```
28
29  // Importar definicion de la tabla Quiz
30  var quiz_path = path.join(__dirname,'quiz');
31  var Quiz = sequelize.import(quiz_path);
32
33  // Importar definicion de la tabla Comment
34  var comment_path = path.join(__dirname,'comment');
35  var Comment = sequelize.import(comment_path);
36
37  // Importar definicion de la tabla Comment
38  var user_path = path.join(__dirname,'user');
39  var User = sequelize.import(user_path);
40
41  Comment.belongsTo(Quiz);
42  Quiz.hasMany(Comment);
43
44  // los quizes pertenecen a un usuario registrado
45  Quiz.belongsTo(User);
46  User.hasMany(Quiz);
47
48  // exportar tablas
49  exports.Quiz = Quiz;
50  exports.Comment = Comment;
51  exports.User = User;
52
```

models.js

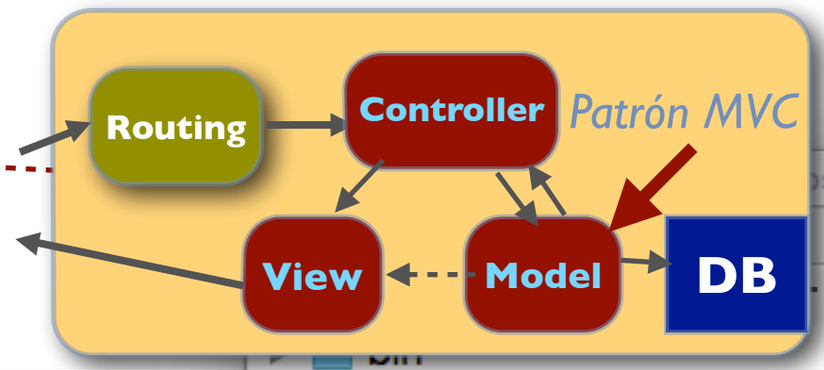
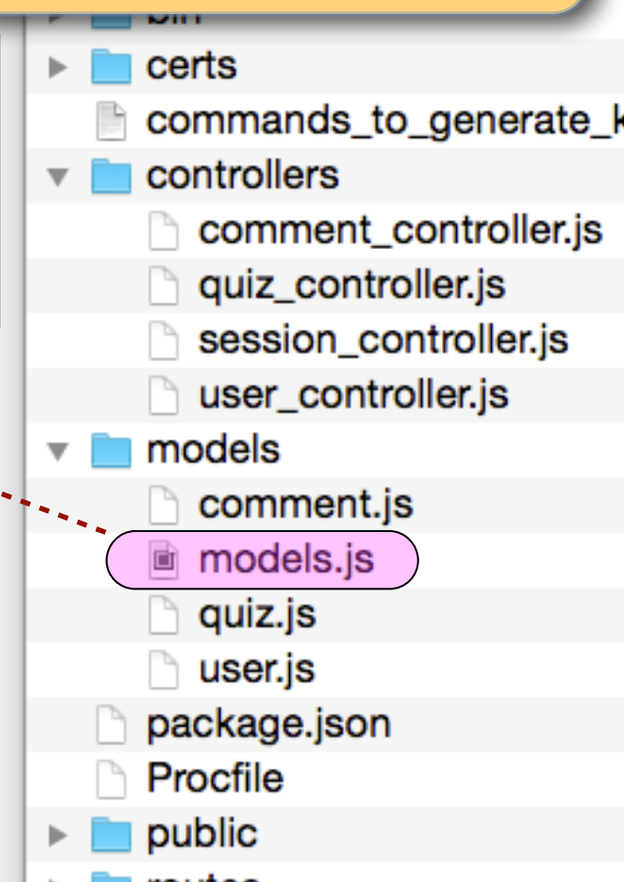


Tabla User:
- username
- password
- isAdmin

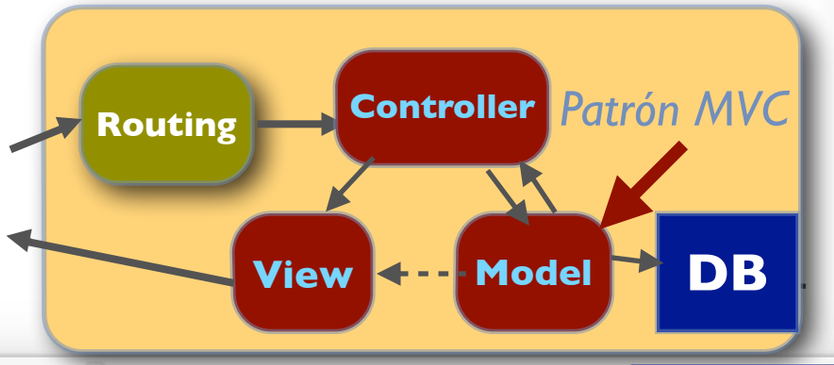
Relación 1-N entre Tablas User y Quiz

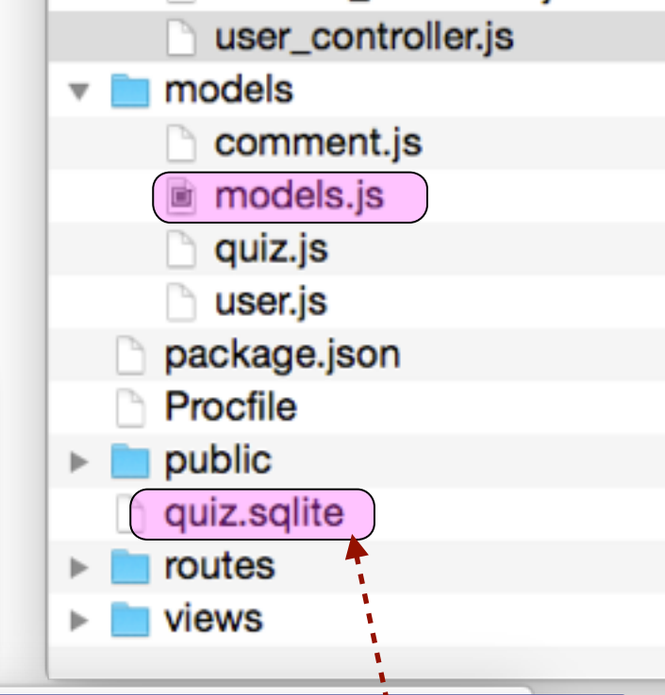


©

Paso 3: inicialización de la DB

id	username	password	isAdmin
1	admin	r34et5690y	TRUE
2	pepe	56gh90op5	FALSE
3	.....	.....	.....





models.js

```
// sequelize.sync() inicializa tabla de preguntas en DB
sequelize.sync().then(function() {
  // then(..) ejecuta el manejador una vez creada la tabla
  User.count().then(function (count){
    if(count === 0) {   // la tabla se inicializa solo si está vacía
      User.bulkCreate(
        [ {username: 'admin',   password: '1234', isAdmin: true},
          {username: 'pepe',   password: '5678'} // isAdmin por defecto: 'false'
        ]
      ).then(function(){
        console.log('Base de datos (tabla user) inicializada');
        Quiz.count().then(function (count){
          if(count === 0) {   // la tabla se inicializa solo si está vacía
            Quiz.bulkCreate(   // estos quizes pertenecen al usuario pepe (2)
              [ {pregunta: 'Capital de Italia',   respuesta: 'Roma', UserId: 2},
                {pregunta: 'Capital de Portugal', respuesta: 'Lisboa', UserId: 2}
              ]
            ).then(function(){console.log('Base de datos (tabla quiz) inicializada')});
          };
        });
      });
    };
  });
});
```

Si la DB (**quiz.sqlite**) no existe, **sequelize.sync()** la crea.

El administrador solo se puede crear así.

Se inicializa con 2 usuarios

Añadir UserIds de la relación

Paso 4: user_controller.js

El **método autenticar** busca el usuario en la base de datos y verifica la palabra de paso con el método **verifyPassword (pwd)** definido en **models/user.js**

user.js — UNREGISTERED

```
46      {
47        instanceMethods: {
48          verifyPassword: function (password) {
49            var encripted = crypto
50                            .createHmac('sha1', key)
51                            .update(password)
52                            .digest('hex');
53            return encripted === this.password;
54          }
55        }
56      }
57    );
58    return User;
59  }
```

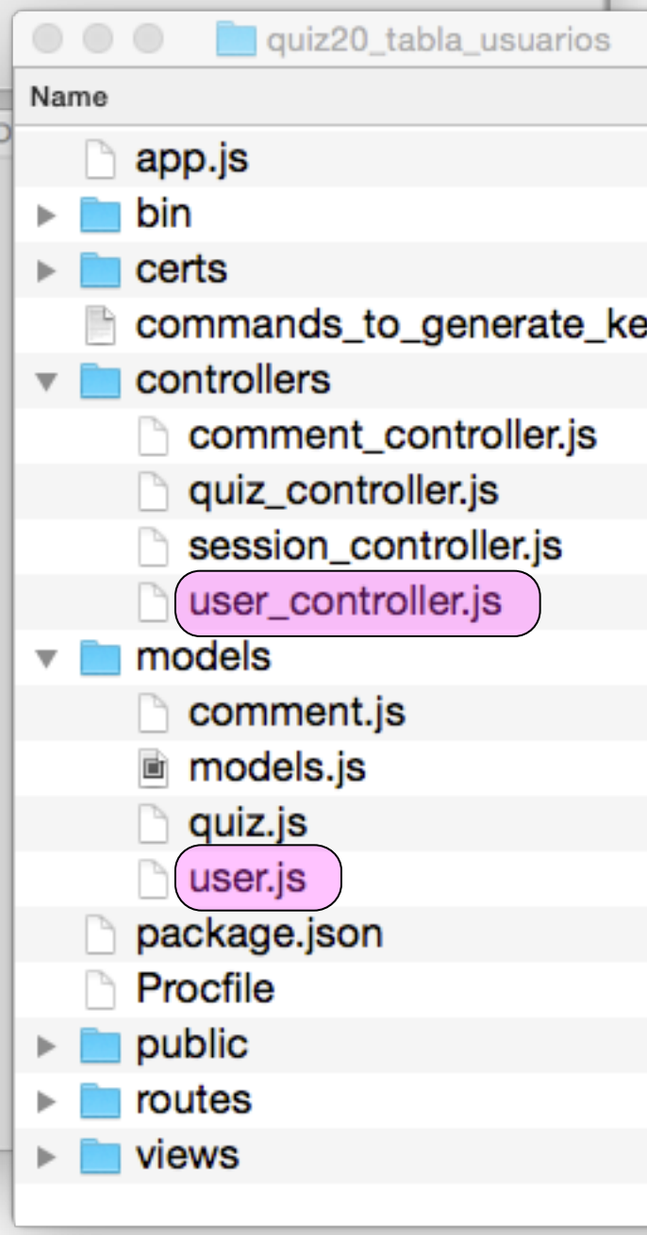
user.js

user_controller.js — quiz20_tabla_usuarios — UNREGISTERED

```
1  var models = require('../models/models.js');
2
3  // Comprueba si el usuario esta registrado en users
4  // Si autenticación falla o hay errores se ejecuta callback(error).
5  exports.autenticar = function(login, password, callback) {
6      models.User.find({
7          where: {
8              username: login
9          }
10     }).then(function(user) {
11         if (user) {
12             if(user.verifyPassword(password)){
13                 callback(null, user);
14             }
15             else { callback(new Error('Password erróneo.')); }
16         } else { callback(new Error('No existe user=' + login))}
17     }).catch(function(error){callback(error)});
18 };
```

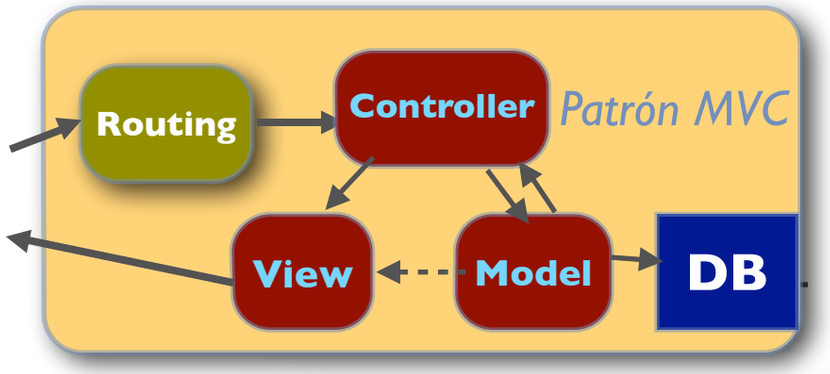
user_controler.js

Método **verifyPassword (pwd)** definido en **models/user.js**



© Juan Quemada - UPM-DIT

Paso 5: quiz_controller.js



Se añade el campo **UserId**, guardado en la sesión, al objeto **req.body.quiz** del formulario de creación de quizes

quiz_controller.js — quiz20_tabla_usuarios

```
// POST /quizes/create
exports.create = function(req, res) {
  req.body.quiz.UserId = req.session.user.id;
  var quiz = models.Quiz.build( req.body.quiz );

  quiz
  .validate()
  .then(
    function(err){
      if (err) {
        res.render('quizes/new', {quiz: quiz, errors: err.errors});
      } else {
        quiz // save: guarda en DB campos pregunta y respuesta de quiz
        .save({fields: ["pregunta", "respuesta", "UserId"]})
        .then( function(){ res.redirect('/quizes')})
      }      // res.redirect: Redirección HTTP a lista de preguntas
    }
  ).catch(function(error){next(error)});
};
```

quiz_controler.js

Ahora se debe guardar también el campo UserId de cada quiz.

quiz20_tabla_usuari

Name

- app.js
- bin
- certs
- commands_to_generate
- controllers
  - comment_controller.j
  - quiz_controller.js
  - session_controller.js
  - user_controller.js
- models
  - comment.js
  - models.js
  - quiz.js
  - user.js
- package.json
- Procfile
- public
- routes
- views

© Juan Quemada - UPM-DIT

Final del tema

Quiz 21: Registro de usuarios y gestión de cuentas

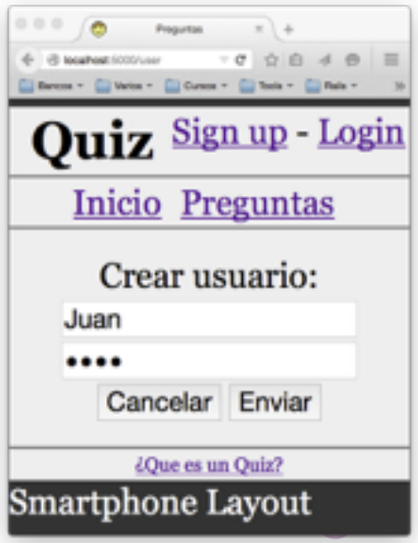
Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

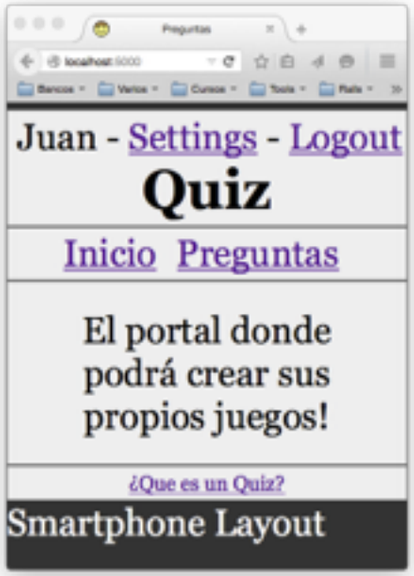
©

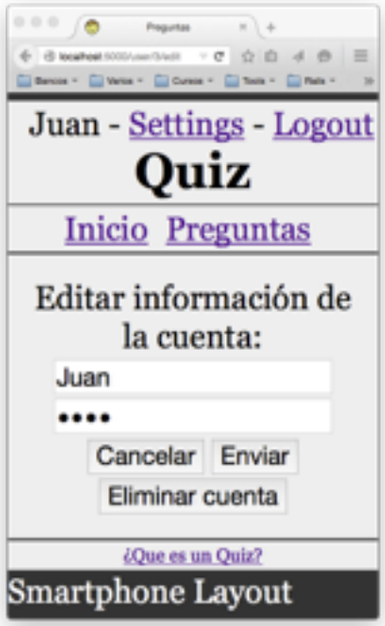
Quiz 21: Registro y gestión de usuarios

Objetivo: Añadir el interfaz REST de acceso para gestionar usuarios permitiendo que se registren en el portal, configuren sus preferencias y borren sus cuentas cuando no las necesitan. Este interfaz renderizará recursos HTML que permiten a los usuarios gestionar el registro desde un navegador.

- **Paso 1**: Crear interfaz de gestión de cuentas de usuarios
- **Paso 2:** Completar **user_controller.js** con acciones del interfaz de gestión de cuentas
- **Paso 3:** Completar middleware de redirección en **app.js**
- **Paso 4:** Actualizar el marco **layout.ejs** incluyendo acceso al registro
- **Paso 4:** Crear las vistas **new.ejs** (Sign up) y **edit.ejs** (Settings)
- **Paso 5**: Guardar versión (commit) con git

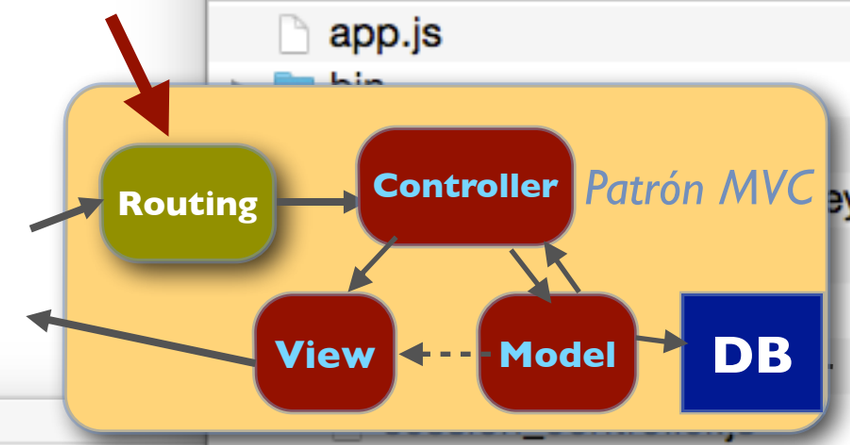


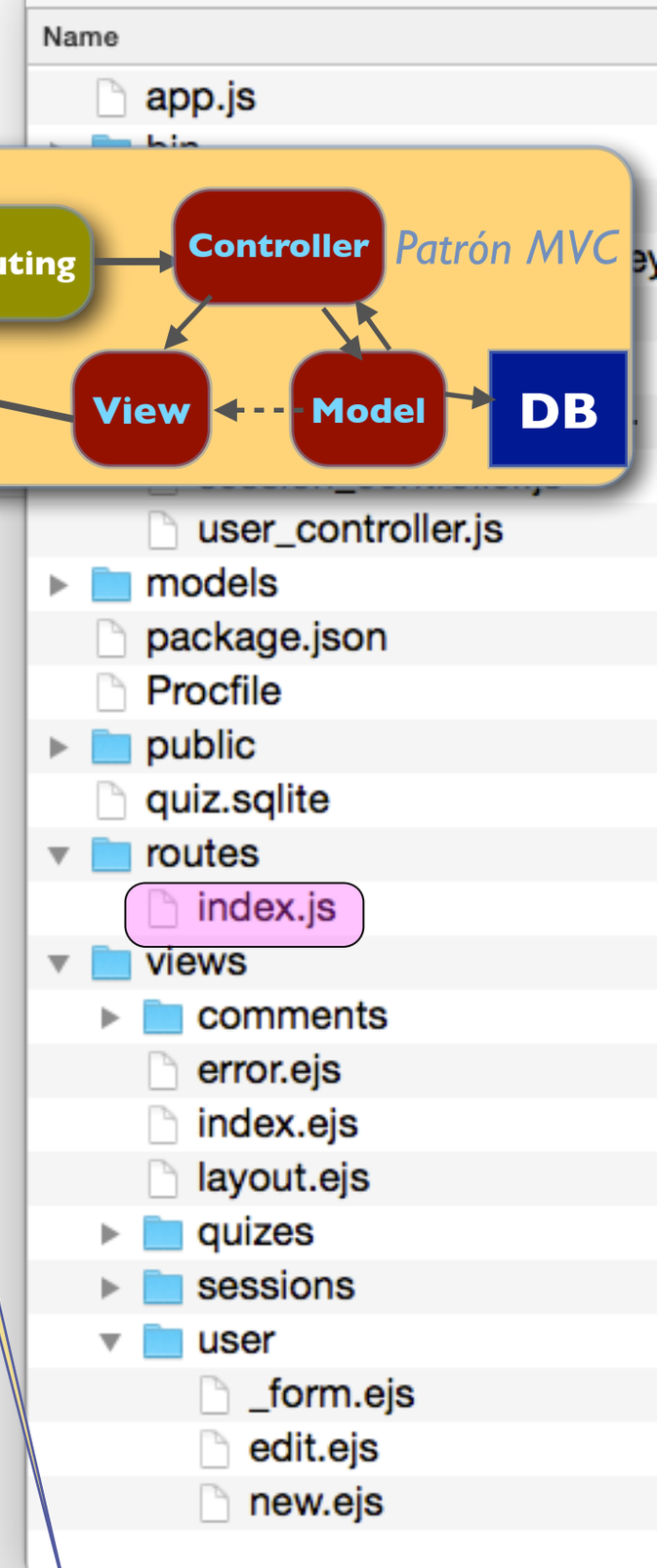




Paso 1: Interfaz

Interfaz de gestión de usuarios:
- Creación de cuenta de usuario
- Edición de preferencias
- Borrado de cuenta de usuario





index.js

```
1  var express = require('express');
2  var router = express.Router();
3
4  var quizController = require('../controllers/quiz_controller');
5  var commentController = require('../controllers/comment_controller');
6  var sessionController = require('../controllers/session_controller');
7  var userController = require('../controllers/user_controller');
8
9  // Página de entrada (home page)
10 router.get('/', function(req, res) {
11   res.render('index', { title: 'Quiz', errors: []});
12 });
13
14 // Autoload de comandos con ids
15 router.param('quizId', quizController.load);        // autoload :quizId
16 router.param('commentId', commentController.load);  // autoload :commentId
17 router.param('userId', userController.load);        // autoload :userId
18
19 // Definición de rutas de sesion
20 router.get('/login',  sessionController.new);     // formulario login
21 router.post('/login', sessionController.create);  // crear sesión
22 router.get('/logout', sessionController.destroy); // destruir sesión
23
24 // Definición de rutas de cuenta
25 router.get('/user',  userController.new);          // formulario sign un
26 router.post('/user',  userController.create);      // registrar usuario
27 router.get('/user/:userId(\\d+)/edit',  sessionController.loginRequired, userController.edit);
28 router.put('/user/:userId(\\d+)',  sessionController.loginRequired, userController.update);
29 router.delete('/user/:userId(\\d+)',  sessionController.loginRequired, userController.destroy);
30
```

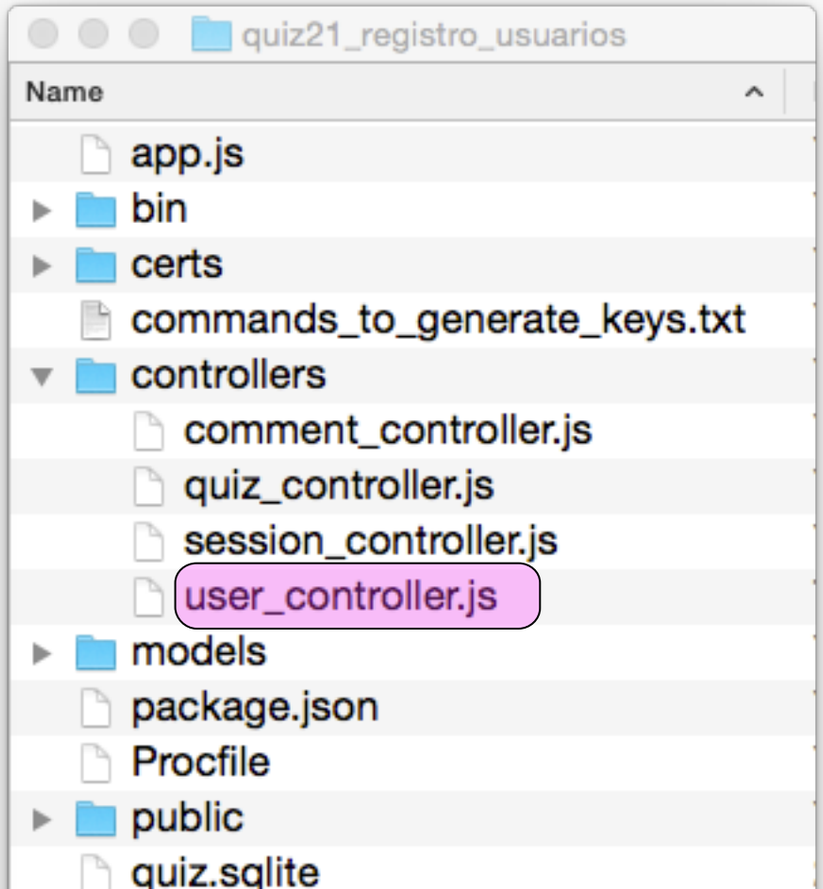
Paso 2: user controller.js

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

// Autoload :userId
exports.load = function(req, res, next, userId) {
  models.User.find({ where: { id: Number(userId)}})
    .then(function(user){
      if (user) {
        req.user = user;
        next();
      } else{next(new Error('No existe userId=' + userId))}
    }
  ).catch(function(error){next(error)});
};

// Comprueba si el usuario esta registrado en users
// Si autenticación falla o hay errores se ejecuta callback(error).
exports.autenticar = function(login, password, callback) {
    .......
};

// GET /user/:id/edit
exports.edit = function(req, res) {
  res.render('user/edit', { user: req.user, errors: []});
};            // req.user: instancia de user cargada con autoload

// GET /user
exports.new = function(req, res) {
    .......
};

// POST /user
exports.create = function(req, res) {
    .......
};

// PUT /user/:id
exports.update = function(req, res, next) {
};

// DELETE /user/:id
exports.destroy = function(req, res) {
    .......
};
```

user_controller.js

Función de auto-load y acciones del controlador de gestión de usuarios:

- Creación de cuenta de usuario
- Edición de preferencias
- Borrado de cuenta de usuario



Preferencias de usuario y auto-load

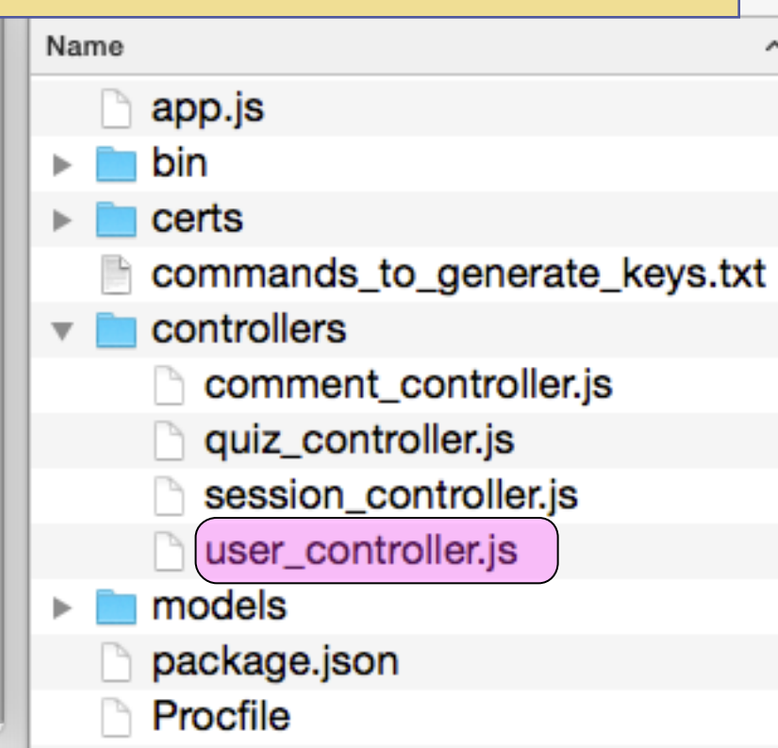
user_controller.js

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

// Autoload :userId
exports.load = function(req, res, next, userId) {
  models.User.find({ where: { id: Number(userId)}})
    .then(function(user){
      if (user) {
        req.user = user;
        next();
      } else{next(new Error('No existe userId=' + userId))}
    }
  ).catch(function(error){next(error)});
};

// Comprueba si el usuario esta registrado en users
// Si autenticación falla o hay errores se ejecuta callback(error).
exports.autenticar = function(login, password, callback) {
      ............
};

// GET /user/:id/edit
exports.edit = function(req, res) {
  res.render('user/edit', { user: req.user, errors: []});
};            // req.user: instancia de user cargada con autoload

// PUT /user/:id
exports.update = function(req, res, next) {
  req.user.username  = req.body.user.username;
  req.user.password  = req.body.user.password;

  req.user
  .validate()
  .then(
    function(err){
      if (err) {
        res.render('user/' + req.user.id, {user: req.user, errors: err.errors});
      } else {
        req.user     // save: guarda campo username y password en DB
        .save( {fields: ["username", "password"]})
        .then( function(){ res.redirect('/');});
      }     // Redirección HTTP a /
    }
  ).catch(function(error){next(error)});
};
```

La función de auto-load carga el registro de usuarios de la tabla User para las primitivas de gestión de preferencias de usuario:

GET /user/:userId/edit
-> carga el formulario de edición

PUT /user/:userId
-> Actualiza actualiza preferencias



Creación y borrado de cuentas

user_controller.js UNREGISTERED

```
// GET /user
exports.new = function(req, res) {
    var user = models.User.build( // crea objeto user
        {username: "", password: ""}
    );
    res.render('user/new', {user: user, errors: []});
};

// POST /user
exports.create = function(req, res) {
  var user = models.User.build( req.body.user );

  user
  .validate()
  .then(
    function(err){
      if (err) {
        res.render('user/new', {user: user, errors: err.errors});
      } else {
        user // save: guarda en DB campos username y password de user
        .save({fields: ["username", "password"]})
        .then( function(){
          // crea la sesión con el usuario ya autenticado y redirige a /
          req.session.user = {id:user.id, username:user.username};
          res.redirect('/');
        });
      }
    }
  ).catch(function(error){next(error)});
};

// DELETE /user/:id
exports.destroy = function(req, res) {
  req.user.destroy().then( function() {
    // borra la sesión y redirige a /
    delete req.session.user;
    res.redirect('/');
  }).catch(function(error){next(error)});
};
```

user_controller.js

quiz21_registro_usuarios

Name

- app.js
- bin
- certs
- commands_to_generate_keys.txt
- controllers
  - comment_controller.js
  - quiz_controller.js
  - session_controller.js
  - user_controller.js
- models
- package.json
- sessions

Las primitivas de creación y borrado de cuentas son:

GET /user
-> carga formulario de creación

POST /user
-> Crea cuenta de usuario

DELETE /user/:userId
-> Destruye cuenta de usuario

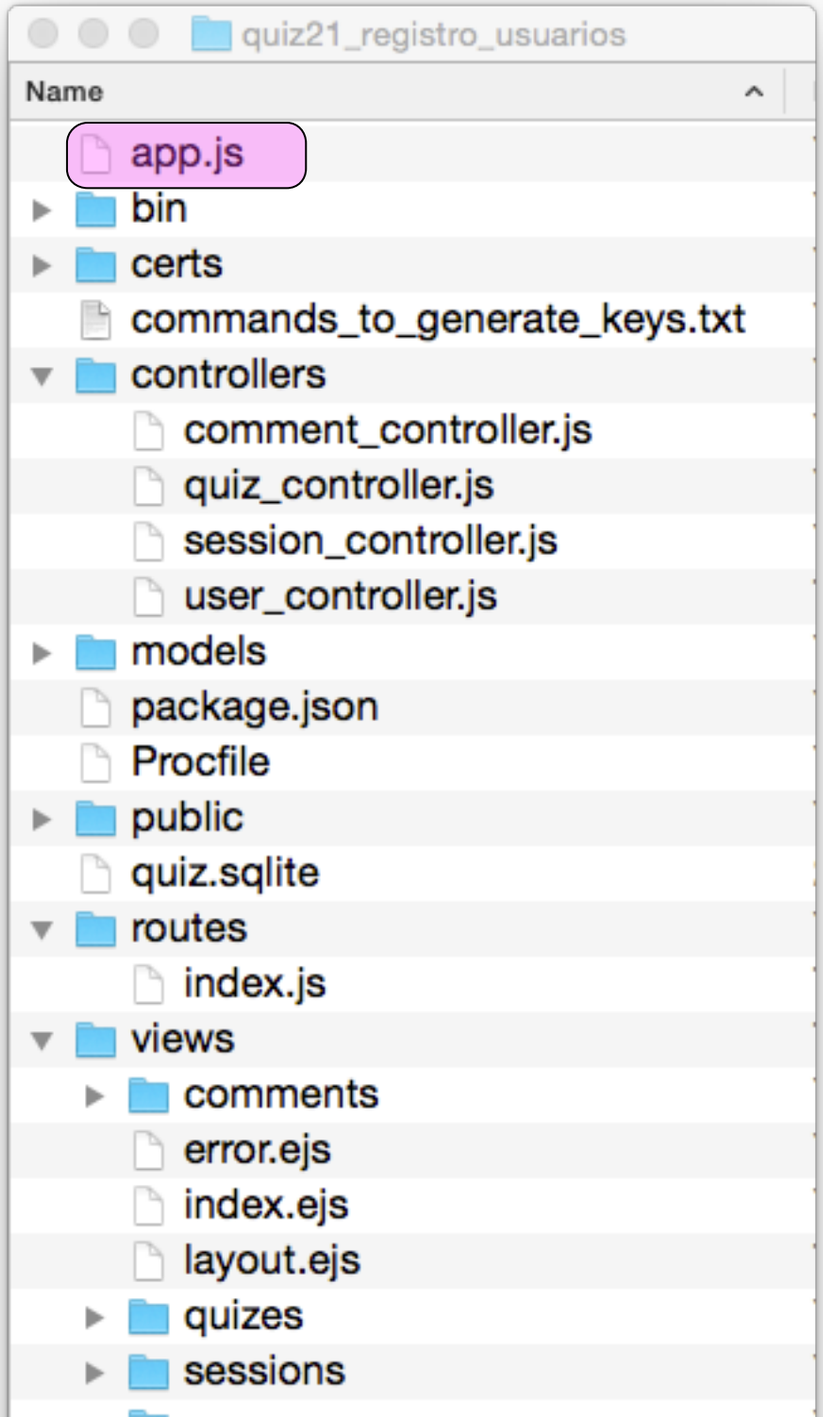
Paso 3: Middleware de redirección

Se actualiza el middleware de gestión de redirección con inicialización y no guardando la ruta tampoco en accesos a **/user**.

app.js UNREGISTERED

```
32  // Helpers dinamicos:
33  app.use(function(req, res, next) {
34
35    // si no existe lo inicializa
36    if (!req.session.redir) {
37      req.session.redir = '/';
38    }
39    // guardar path en session.redir para despues de login
40    if (!req.path.match(/\/login|\/logout|\/user/)) {
41      req.session.redir = req.path;
42    }
43
44    // Hacer visible req.session en las vistas
45    res.locals.session = req.session;
46    next();
47  });
48
49  app.use('/', routes);
50
```

app.js



© Juan Quemada - UPM-DIT

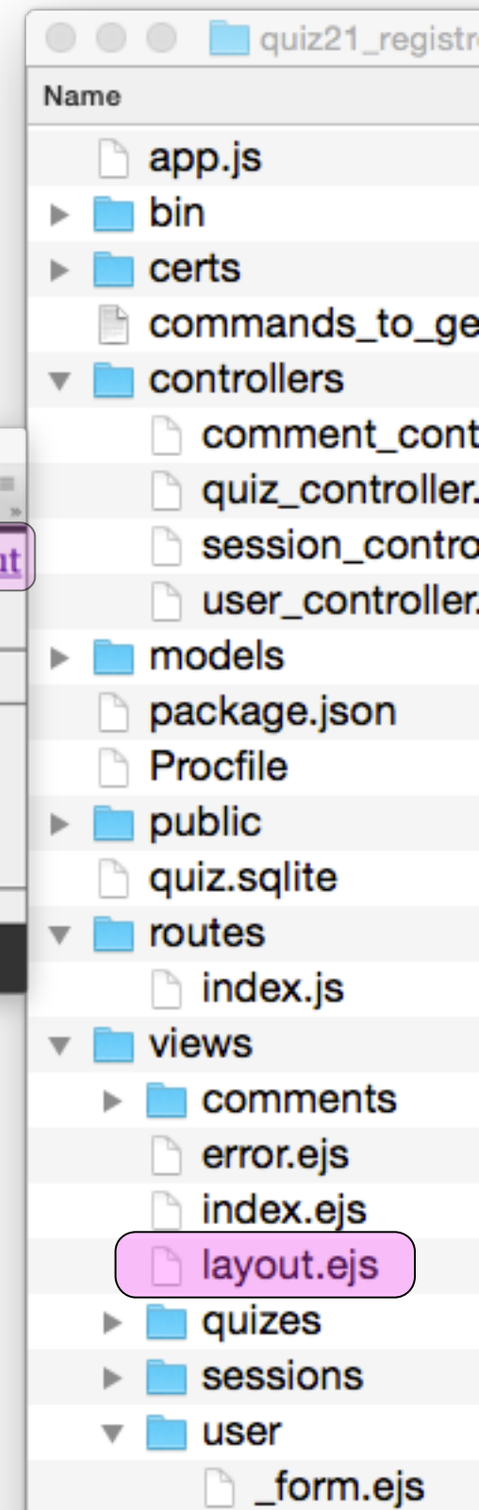
Paso 4: Enlaces "Sign up" y "Settings"

Se añaden nuevos enlaces al marco de las vistas (**layout.ejs**):

- **Sign up**: para crear cuenta si usuario no está autenticado
- **Settings**: para editar cuenta si usuario está autenticado

```
17  <body>
18  <div id="page-wrap">
19    <header class="main" id="h1">
20      <%if(!session.user){%>
21        <span class="right">
22            <a href="/user">Sign up</a>
23          - <a href="/login">Login</a>
24        </span>
25      <%}else{%>
26        <span class="right">
27            <%=session.user.username%>
28          - <a href="/user/<%=session.user.id%>/edit">Settings</a>
29          - <a href="/logout">Logout</a>
30        </span>
31      <%}%>
32
33      <h2>Quiz<span>: el juego de las preguntas</span></h2>
```

layout.ejs



© Juan Quemada - UPM-DIT

Paso 4:
Vistas "new.ejs" y "edit.ejs"

edit.ejs

```
<p>Editar información de la cuenta:</p>

<form method="post" action="/user/<%= user.id %>?_method=put" >
  <% include _form.ejs %>
</form>

<form method="post" action="/user/<%= user.id %>?_method=delete">
    <button type="submit" onClick="return confirm('Borrar cuenta: <%= user.username %>');">
    Eliminar cuenta
    </button>
</form>
```

edit.ejs

Los formularios **new.ejs** y **edit.ejs** reutilizan los cajetines input y el boton de submit al ser comunes.

new.ejs

```
Crear usuario: <p>

<form method="post" action="/user" >

  <% include _form.ejs %>
</form>
```

new.ejs

_form.ejs

```
<label for="preg" class="rem">Username:</label>
<input type="text" id="username" name="user[username]" value="<%= user.username %>"/><p>
<label for="resp" class="rem">Password:</label>
<input type="password" id="pass" name="user[password]"/> <p>
<a href="/"><button type="button">Cancelar</button></a>
<input type="submit" value="Enviar">
```

_form.ejs

Final del tema

Quiz 22: Roles y permisos

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

Quiz 22: Roles y permisos

Objetivo: Añadir el rol de administrador del sistema y definir los permisos adecuadamente:

- **Admin**: puede hacer cualquier operación sobre cualquier recurso
- **Usuario registrado**: Es propietario de los recursos que ha creado (cuenta y quizes) y puede hacer cualquier operación sobre ellos. Los demás recursos solo puede verlos.
- **Anónimo**: Solo puede ver los recursos del portal.

✦**Paso 1**: Añadir controles de **propiedad** (ownershipRequired) a rutas en **routes/index.js**

✦**Paso 2**: Añadir MW **ownershipRequired()** a **quiz_controller**

✦**Paso 3**: Añadir MW **ownershipRequired()** a **user_controller**

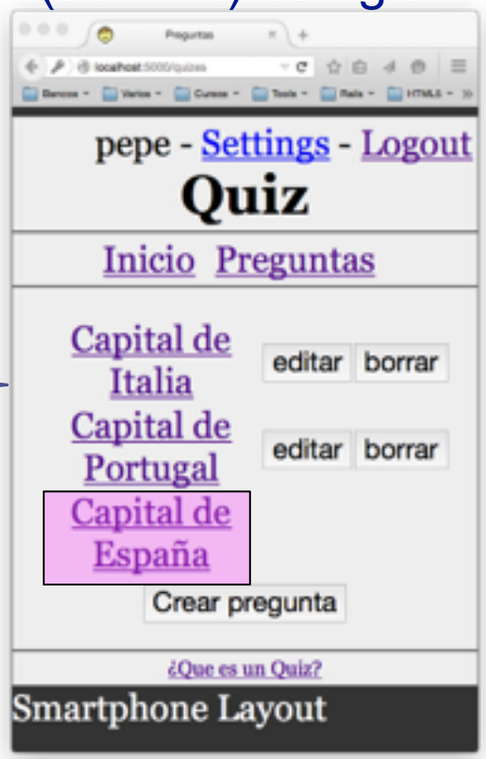
✦**Paso 4**: Añadir MW **ownershipRequired()** a **comment_controller**

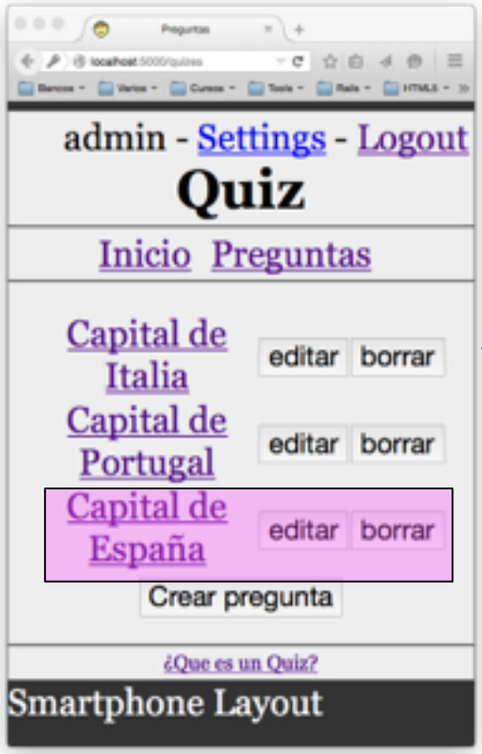
✦**Paso 5**: Adaptar **session_controller** al modelo con administrador

✦**Paso 6**: Modificar vista **index.ejs** y **show.ejs** de **quizes** para mostrar botones adecuadamente

✦**Paso 7**: Guardar versión (commit) con git

Usuario autenticado: solo puede realizar las operaciones sobre sus recursos



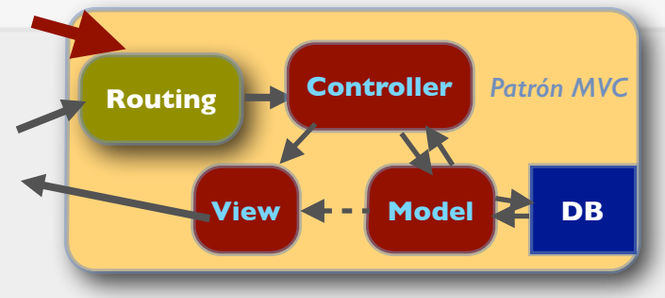


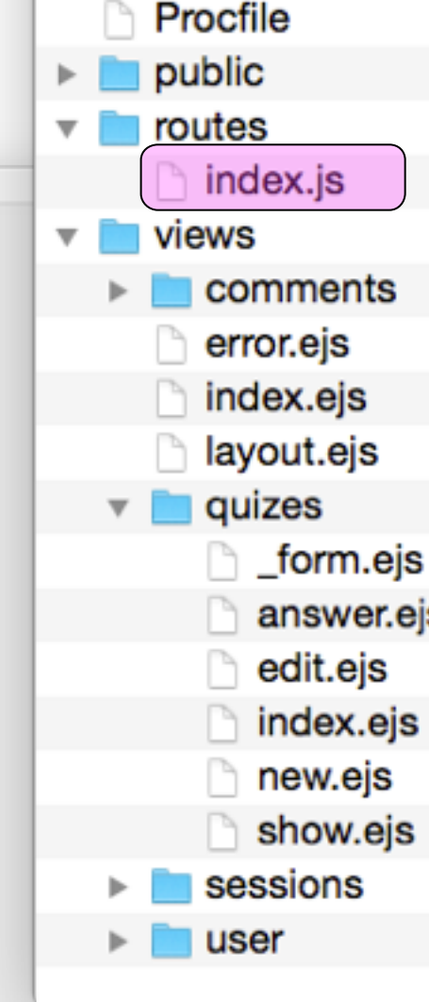
Administrador: puede realizar cualquier operación

Paso 1

index.js

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

var quizController = require('../controllers/quiz_controller');
var commentController = require('../controllers/comment_controller');
var sessionController = require('../controllers/session_controller');
var userController = require('../controllers/user_controller');

// Página de entrada (home page)
router.get('/', function(req, res) {
  res.render('index', { title: 'Quiz', errors: []});
});

// Autoload de comandos con ids
router.param('quizId', quizController.load);        // autoload :quizId
router.param('commentId', commentController.load);  // autoload :commentId
router.param('userId', userController.load);        // autoload :userId

// Definición de rutas de sesion
router.get('/login',  sessionController.new);     // formulario login
router.post('/login', sessionController.create);  // crear sesión
router.get('/logout', sessionController.destroy); // destruir sesión

// Definición de rutas de cuenta
router.get('/user',  userController.new);         // formulario sign un
router.post('/user',  userController.create);     // registrar usuario
router.get('/user/:userId(\\d+)/edit',  sessionController.loginRequired, userController.ownershipRequired, userController.edit);
router.put('/user/:userId(\\d+)',       sessionController.loginRequired, userController.ownershipRequired, userController.update);
router.delete('/user/:userId(\\d+)',    sessionController.loginRequired, userController.ownershipRequired, userController.destroy);

// Definición de rutas de /quizes
router.get('/quizes',                      quizController.index);
router.get('/quizes/:quizId(\\d+)',        quizController.show);
router.get('/quizes/:quizId(\\d+)/answer', quizController.answer);
router.get('/quizes/new',                  sessionController.loginRequired, quizController.new);
router.post('/quizes/create',              sessionController.loginRequired, quizController.create);
router.get('/quizes/:quizId(\\d+)/edit',   sessionController.loginRequired, quizController.ownershipRequired, quizController.edit);
router.put('/quizes/:quizId(\\d+)',        sessionController.loginRequired, quizController.ownershipRequired, quizController.update);
router.delete('/quizes/:quizId(\\d+)',     sessionController.loginRequired, quizController.ownershipRequired, quizController.destroy);

// Definición de rutas de comentarios
router.get('/quizes/:quizId(\\d+)/comments/new', commentController.new);
router.post('/quizes/:quizId(\\d+)/comments',    commentController.create);
router.get('/quizes/:quizId(\\d+)/comments/:commentId(\\d+)/publish',
                                   sessionController.loginRequired, commentController.ownershipRequired, commentController.publish)

module.exports = router;
```





ownershipRequired: MV que comprueba si el usuario que accede tiene derechos para realizar la operación.

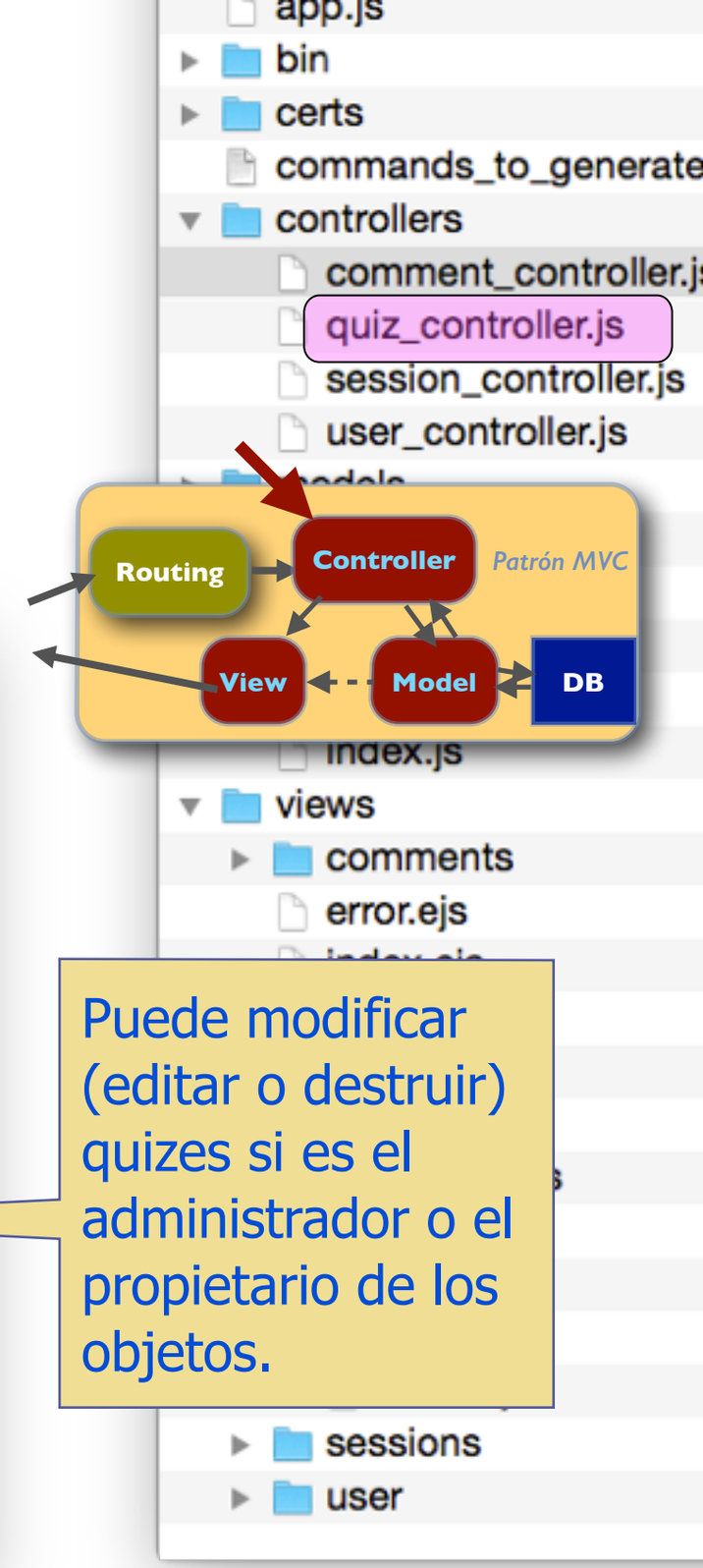
Paso 2: quiz_controller.js

quiz_controller.js UNREGISTERED

```
1  var models = require('../models/models.js');
2
3  // MW que permite acciones solamente si el quiz objeto
4  // pertenece al usuario logeado o si es cuenta admin
5  exports.ownershipRequired = function(req, res, next){
6      var objQuizOwner = req.quiz.UserId;
7      var logUser = req.session.user.id;
8      var isAdmin = req.session.user.isAdmin;
9
10     if (isAdmin || objQuizOwner === logUser) {
11         next();
12     } else {
13         res.redirect('/');
14     }
15 };
16
```

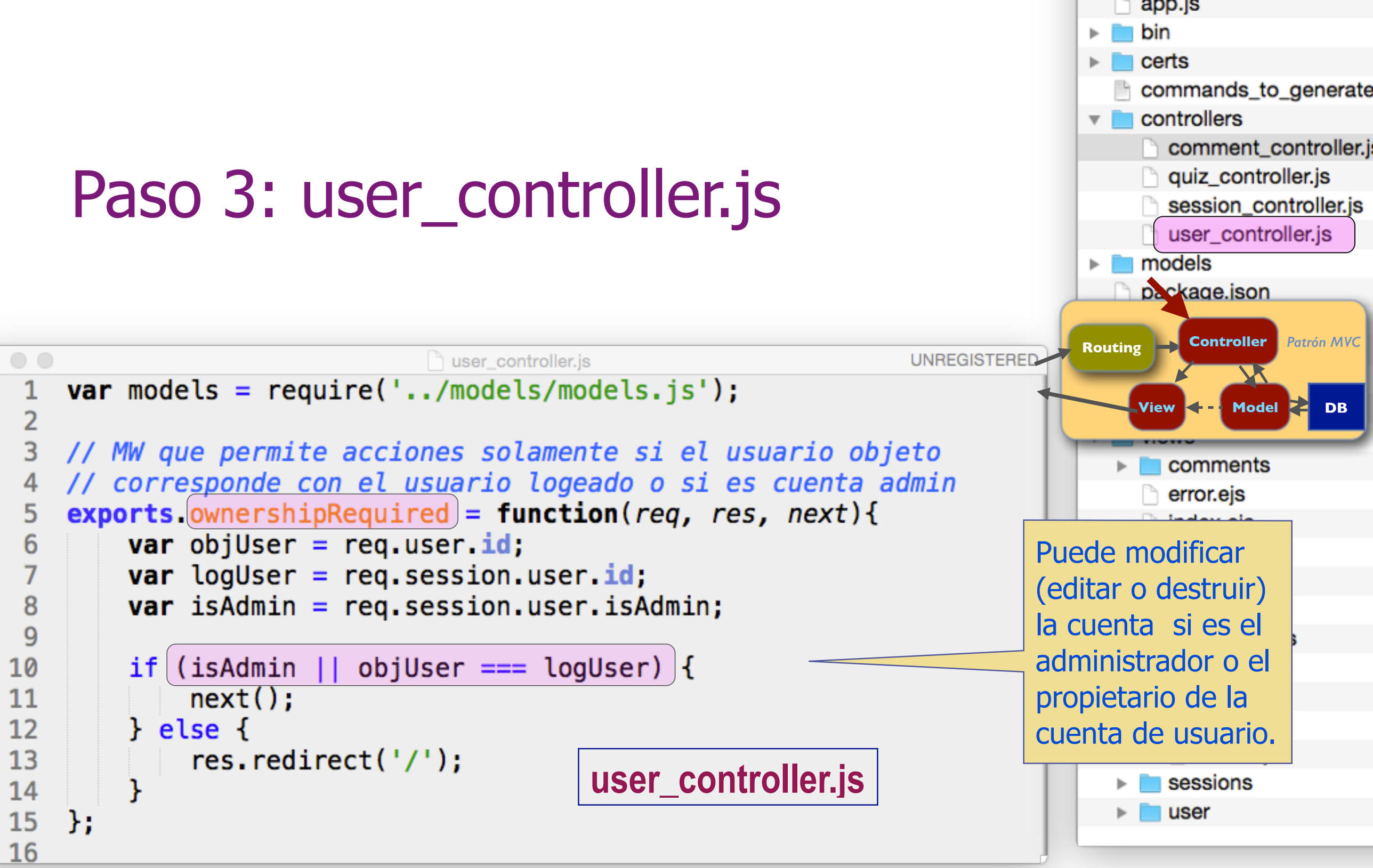
quiz_controller.js

Puede modificar (editar o destruir) quizes si es el administrador o el propietario de los objetos.



© Juan Quemada - UPM-DIT

Paso 3: user_controller.js

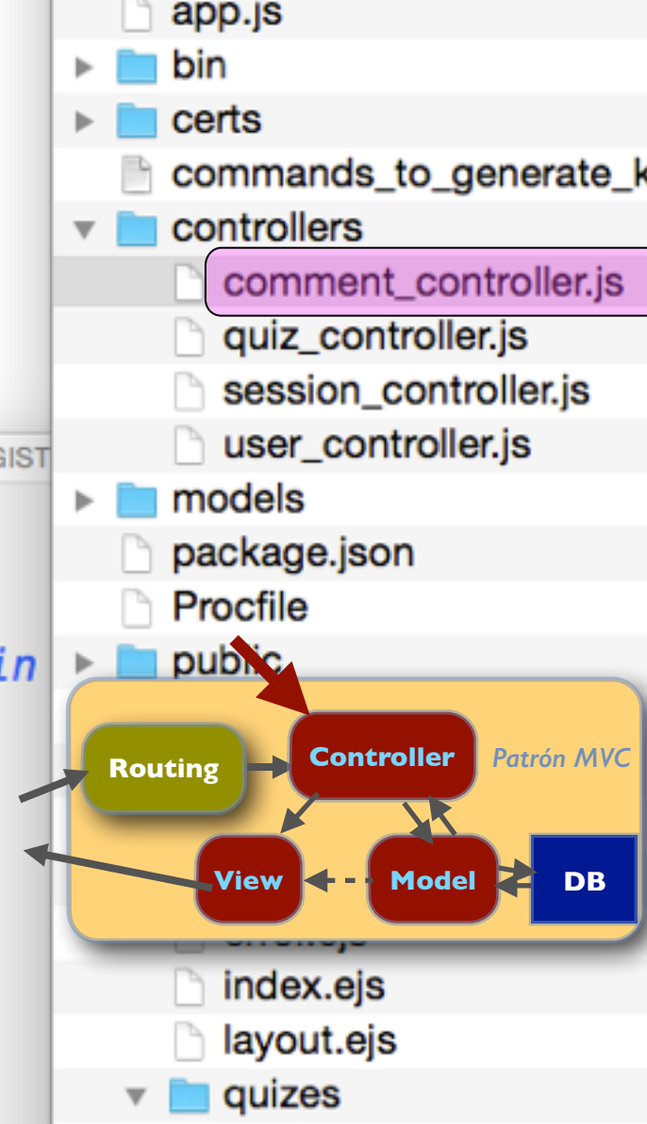


© Juan Quemada - UPM-DIT

Paso 4: comment_controller.js

comment_controller.js

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

// MW que permite acciones solamente si el quiz al que pertenece
// el comentario objeto pertenece al usuario logeado o si es cuenta admin
exports.ownershipRequired = function(req, res, next){
    models.Quiz.find({
            where: {
                id: Number(req.comment.QuizId)
            }
        }).then(function(quiz) {
            if (quiz) {
                var objQuizOwner = quiz.UserId;
                var logUser = req.session.user.id;
                var isAdmin = req.session.user.isAdmin;

                console.log(objQuizOwner, logUser, isAdmin);

                if (isAdmin || objQuizOwner === logUser) {
                    next();
                } else {
                    res.redirect('/');
                }
            } else{next(new Error('No existe quizId=' + quizId))}
        }
    ).catch(function(error){next(error)});
};
```



comment_controller.js

Puede publicar un comentario si es el administrador o el propietario del quiz al que se ha enviado el comentario.

© Juan Quemada - UPM-DIT

Paso 5: session_controller.js

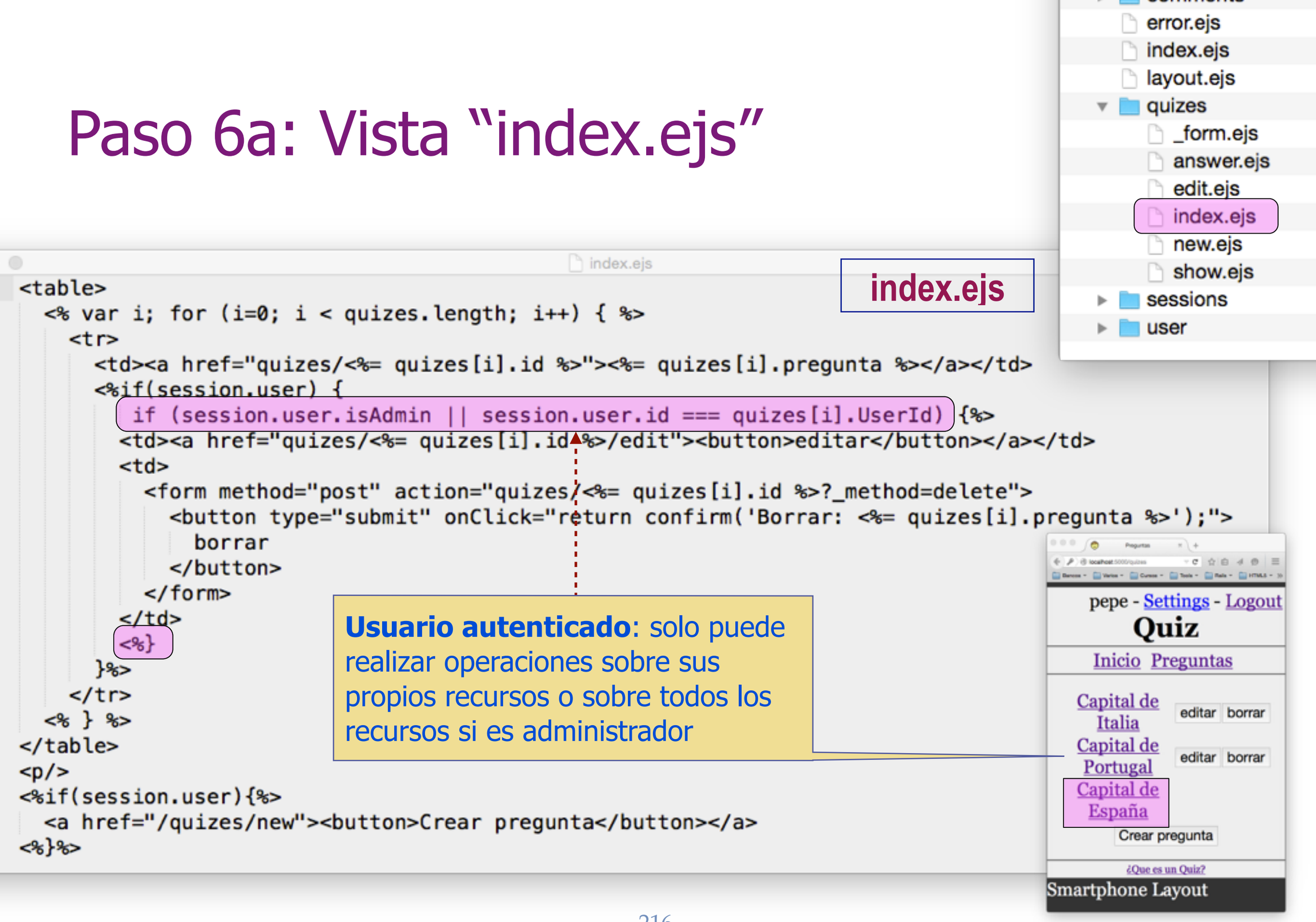
Al autenticar un usuario y crear el objeto user en la sesión, hay que guardar también información de si es administrador o no.

```
// POST /login   -- Crear la sesion si usuario se autentica
exports.create = function(req, res) {

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

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

        if (error) {  // si hay error retornamos mensajes de error de sesi n
            req.session.errors = [{"message": 'Se ha producido un error: '+error}];
            res.redirect("/login");
            return;
        }

        // Crear req.session.user y guardar campos   id  y  username
        // La sesión se define por la existencia de:    req.session.user
        req.session.user = {id:user.id, username:user.username, isAdmin:user.isAdmin};

        res.redirect(req.session.redir.toString());// redirección a path anterior a login
    });
};
```

session_controller.js

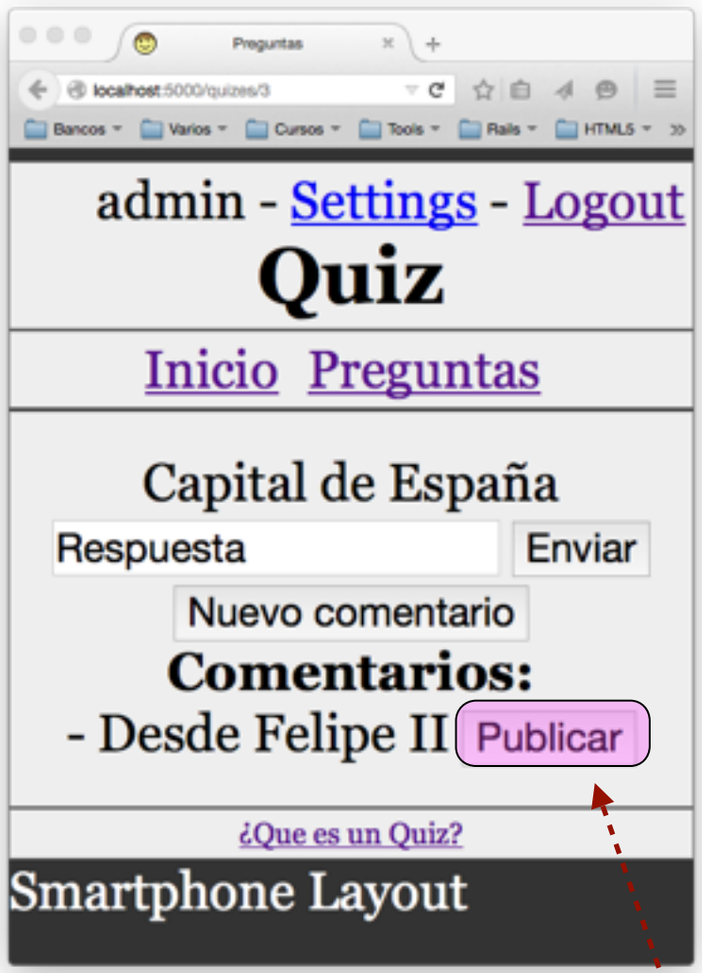
© Juan Quemada - UPM-DIT

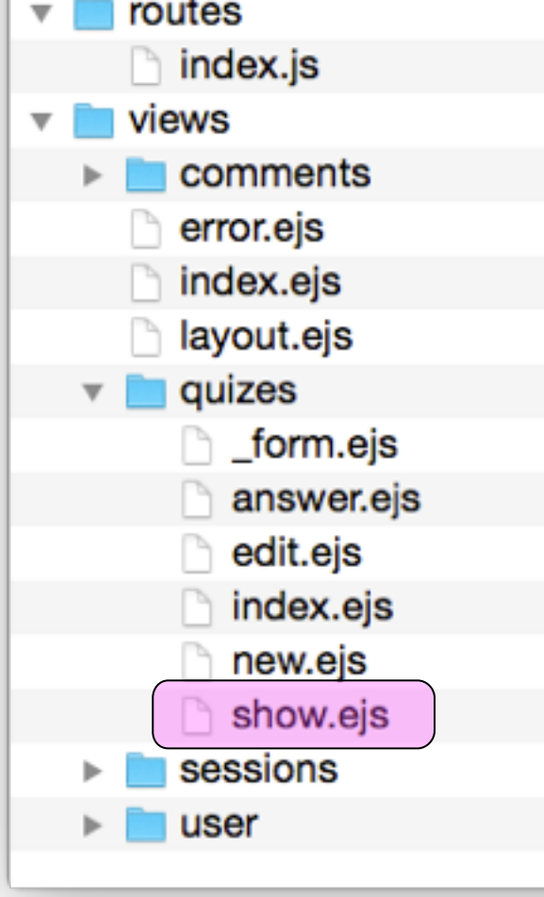
Paso 6a: Vista "index.ejs"



© Juan Quemada - UPM-DIT

Paso 6b: Vista "show.ejs"





Un comentario publicado puede verse sin ninguna restricción

Usuario autenticado: solo el administrador o el propietario del quiz pueden publicar un comentario a ese quiz

show.ejs

```
<form method="get" action="/quizes/<%= quiz.id %>/answer">
  <%= quiz.pregunta %> <p>
  <input type="text" name="respuesta" value="Respuesta"/>
  <input type="submit" value="Enviar">
</form>

<p><a href="/quizes/<%= quiz.id %>/comments/new"><button>Nuevo comentario</button></a></p>

<p><strong>Comentarios:</strong></p>
<%for(index in quiz.Comments){%>
    <%if(quiz.Comments[index].publicado){%>
        <p>- <%=quiz.Comments[index].texto%></p>
    <%} else if(session.user && (session.user.isAdmin || (session.user && quiz.UserId === session.user.id))){%>
            <p>- <%=quiz.Comments[index].texto%>
            <a href="/quizes/<%= quiz.id %>/comments/<%=quiz.Comments[index].id%>/publish"><button>Publicar</button></a></p>
    <%}%>
<%}%>
```

© Juan Quemada - UPM-DIT

Final del tema

Quiz 23: Subir imagenes

Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

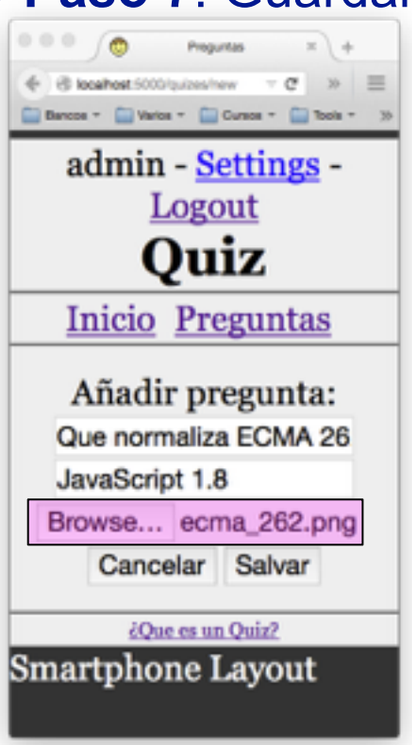
©

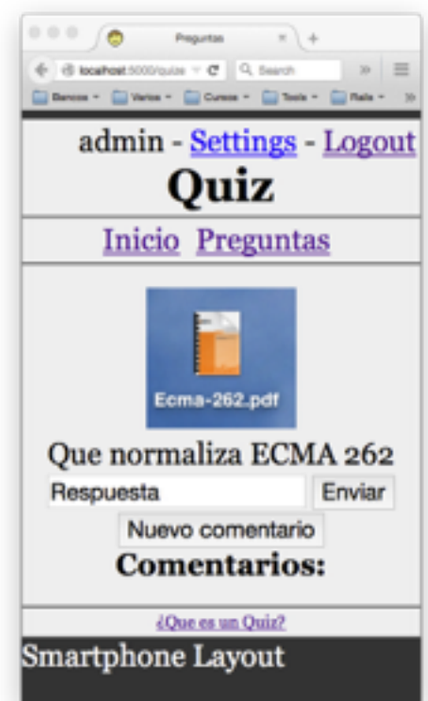
Quiz 23: Añadir una imagen

Objetivo: Añadir una imagen a los quizes sobre la que hacer preguntas.

La imagen se subira con un elemento <input type="file" ….> y se codificará e formato multipart/form-data para enviarla al servidor en el body de la solicitud

- ✦**Paso 1**: Añadir el middleware multer en **routes/index.js** para gestionar **multipart/form-data**
- ✦**Paso 2**: Añadir el directorio **public/media** para guardar imagenes
- ✦**Paso 3**: Adaptar acciones **create** y **update** de **quiz_controller**
- ✦**Paso 4**: Añadir campo **image** a la tabla **quiz.js** del modelo
- ✦**Paso 5**: Modificar los formularios **new.ejs** y **edit.ejs** de **quizes** para que suban ficheros
- ✦**Paso 6**: Modificar vista **show.ejs** de **quizes** para mostrar la imagen
- ✦**Paso 7**: Guardar versión (commit) con git y desplegar en Heroku





Paso 1

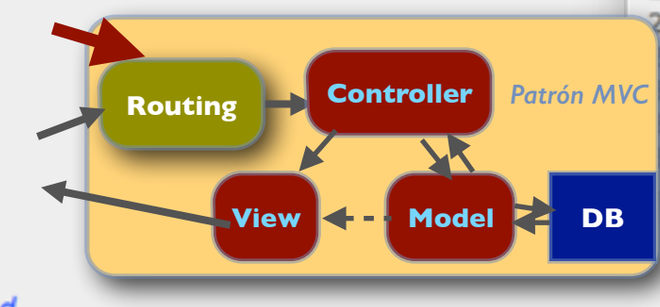
Se importa el MW **multer** instalado con **npm,** que permite extraer ficheros enviados en formato MIME **multipart/form-data** y guardarlos en un fichero en el servidor.

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

var quizController = require('../controllers/quiz_controller');
var commentController = require('../controllers/comment_controller');
var sessionController = require('../controllers/session_controller');
var userController = require('../controllers/user_controller');

// Página de entrada (home page)
router.get('/', function(req, res) {
  res.render('index', { title: 'Quiz', errors: []});
});

// Autoload de comandos con ids
router.param('quizId', quizController.load);        // autoload :quizId
router.param('commentId', commentController.load);  // autoload :commentId
router.param('userId', userController.load);        // autoload :userId

// Definición de rutas de sesion
router.get('/login',  sessionController.new);     // formulario login
router.post('/login', sessionController.create);  // crear sesión
router.get('/logout', sessionController.destroy); // destruir sesión

// Definición de rutas de cuenta
router.get('/user',  userController.new);         // formulario sign un
router.post('/user',  userController.create);     // registrar usuario
router.get('/user/:userId(\\d+)/edit',  sessionController.loginRequired, userController.ownershipRequired, userController.edit);
router.put('/user/:userId(\\d+)',  sessionController.loginRequired, userController.ownershipRequired, userController.update);
router.delete('/user/:userId(\\d+)',  sessionController.loginRequired, userController.ownershipRequired, userController.destroy);

// Definición de rutas de /quizes
router.get('/quizes',                      quizController.index);
router.get('/quizes/:quizId(\\d+)',        quizController.show);
router.get('/quizes/:quizId(\\d+)/answer', quizController.answer);
router.get('/quizes/new',                  sessionController.loginRequired, quizController.new);
router.post('/quizes/create',              sessionController.loginRequired,
                                           multer({ dest: './public/media/'}),
                                           quizController.create);
router.get('/quizes/:quizId(\\d+)/edit',   sessionController.loginRequired, quizController.ownershipRequired, quizController.edit);
router.put('/quizes/:quizId(\\d+)',        sessionController.loginRequired,
                                           quizController.ownershipRequired,
                                           multer({ dest: './public/media/'}),
                                           quizController.update);
router.delete('/quizes/:quizId(\\d+)',     sessionController.loginRequired, quizController.ownershipRequired, quizController.destroy);
```

MW **multer** de gestión de multipart

MW **multer** de gestión de multipart

```
  "sqlite3": "^3.0.4"
  },
  "dependencies": {
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "debug": "~2.0.0",
    "ejs": "~0.8.5",
    "express": "~4.9.0",
    "express-partials": "^0.3.
    "express-session": "~1.0.3
    "method-override": "^2.3.1
    "morgan": "~1.3.0",
    "pg": "^4.1.1",
    "pg-hstore": "2.3.2",
    "sequelize": "^2.0.0-rc4",
    "serve-favicon": "~2.1.3",
    "multer": "~0.1.8"
  }
}
```

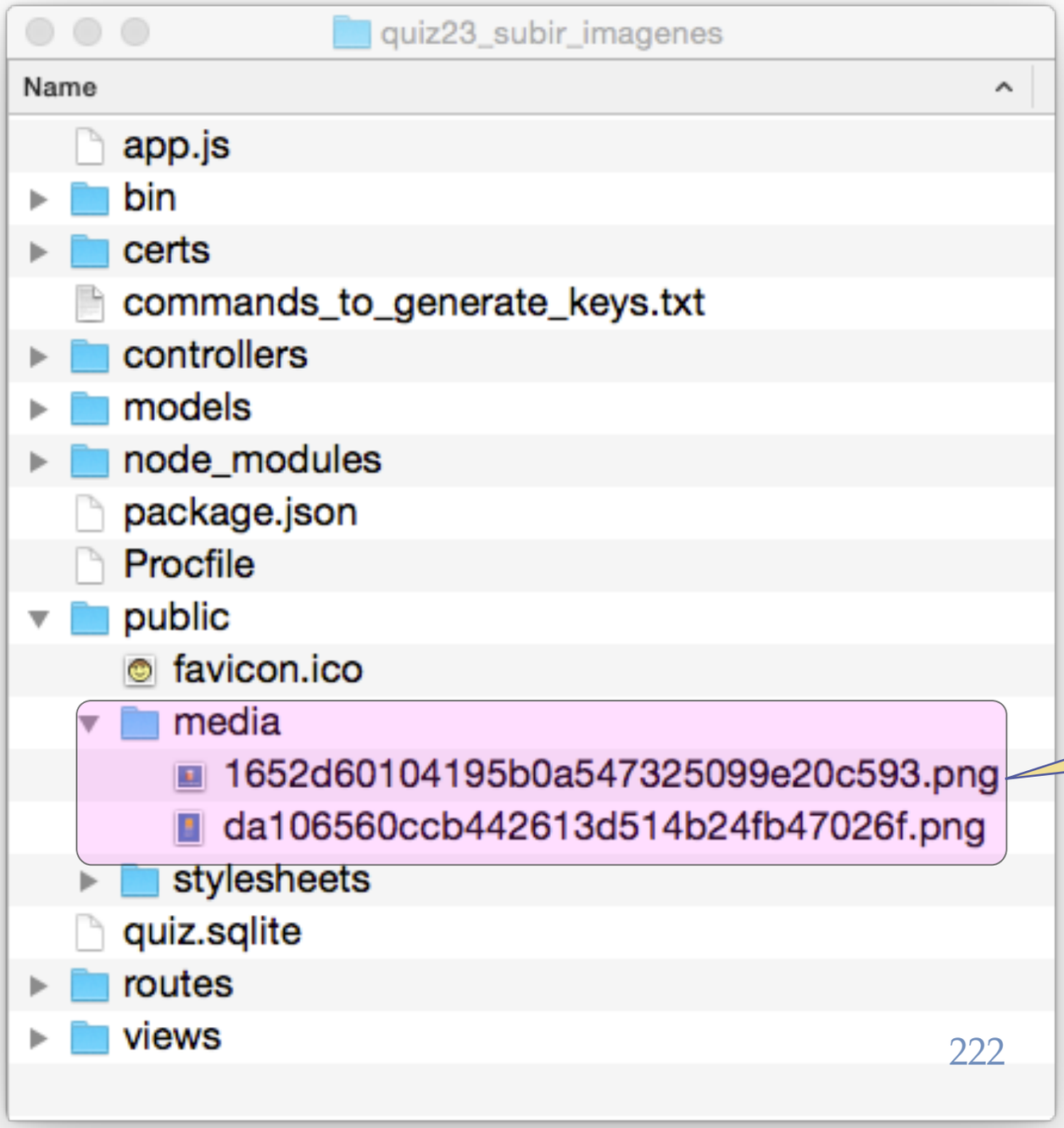


// instalar MW **multer** de gestión
// de **multipart/form-data** en
// **routes/index.js**

..$ npm install --save multer

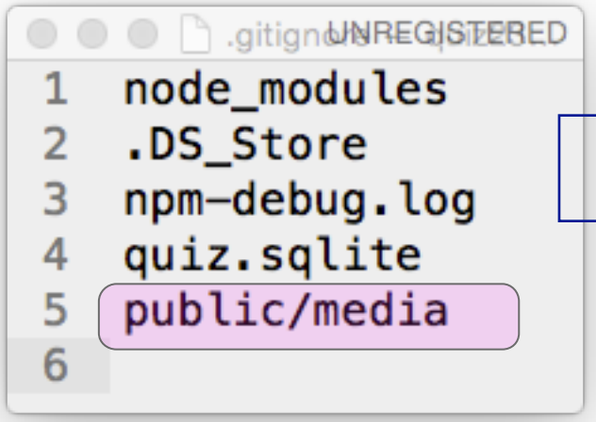
routes/index.js

Paso 3: Directorio public/media



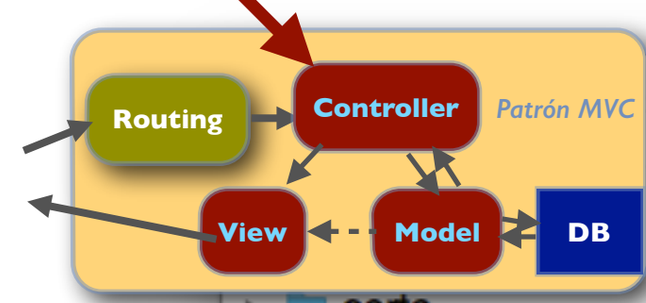
Las imágenes se guardan como ficheros con un nombre único en el directorio public/media.

En Heroku las imágenes se guardan en un disco virtual, por lo que si se reinicia la másuina virtual (Dynos), los fichero guardados se perderán. Para guardarlas permanentemente hay que útilizar servicios como cloudinary (http://cloudinary.com) que permiten guardar información permanentemente.



.gitignore

© Juan Quemada - UPM-DIT



Paso 3a: quiz_controller - create

quiz_controller.js — quiz23_subir_imagenes

quiz_controller.js

```
72  // POST /quizes/create
73  exports.create = function(req, res) {
74    req.body.quiz.UserId = req.session.user.id;
75    if(req.files.image){
76      req.body.quiz.image = req.files.image.name;
77    }
78    var quiz = models.Quiz.build( req.body.quiz );
79
80    quiz
81    .validate()
82    .then(
83      function(err){
84        if (err) {
85          res.render('quizes/new', {quiz: quiz, errors: err.errors});
86        } else {
87          quiz // save: guarda en DB campos pregunta y respuesta de quiz
88          .save({fields: ["pregunta", "respuesta", "UserId", "image"]})
89          .then( function(){ res.redirect('/quizes')})
90        }      // res.redirect: Redirección HTTP a lista de preguntas
91      }
92    ).catch(function(error){next(error)});
93  };
94
```

si el quiz enviado desde el formulario incluye una imagen, su referencia se añade también al objeto quiz que se va a guardar en la DB.

- certs
- commands_to_gene
- controllers
  - comment_control
  - quiz_controller.js
  - session_controlle
  - user_controller.js
- models
- package.json
- Procfile
- public
- routes
  - index.js
- views
  - comments
  - error.ejs
  - index.ejs
  - layout.ejs
  - quizes
    - _form.ejs
    - answer.ejs
    - edit.ejs
    - index.ejs
    - new.ejs
    - show.ejs
  - sessions
  - user

© Juan Quemada - UPM-DIT

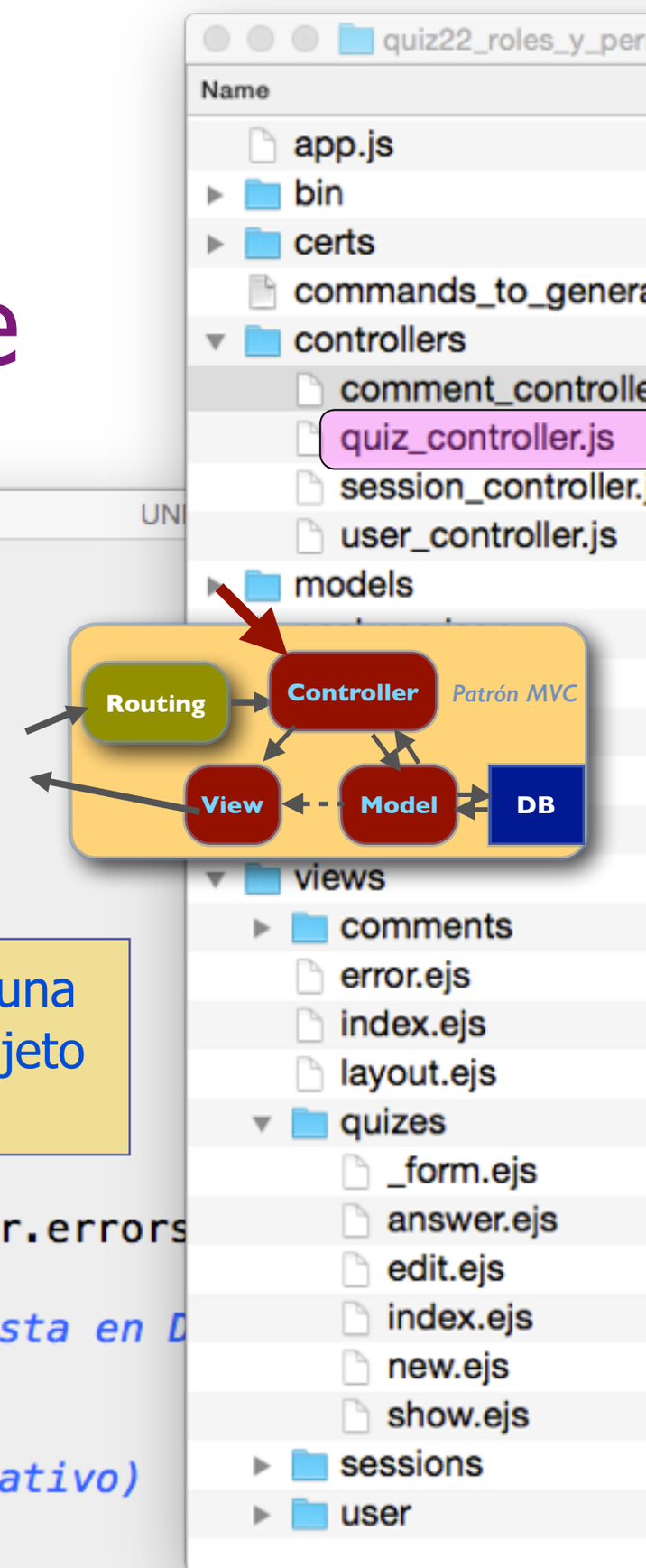
Paso 3b: quiz_controller - update

quiz_controller.js

quiz_controller.js — quiz23_subir_imagenes

```
102 // PUT /quizes/:id
103 exports.update = function(req, res) {
104   if(req.files.image){
105     req.quiz.image = req.files.image.name;
106   }
107   req.quiz.pregunta  = req.body.quiz.pregunta;
108   req.quiz.respuesta = req.body.quiz.respuesta;
109
110   req.quiz
111   .validate()
112   .then(
113     function(err){
114       if (err) {
115         res.render('quizes/edit', {quiz: req.quiz, errors: err.errors
116       } else {
117         req.quiz     // save: guarda campos pregunta y respuesta en D
118         .save( {fields: ["pregunta", "respuesta", "image"]})
119         .then( function(){ res.redirect('/quizes');});
120       }     // Redirección HTTP a lista de preguntas (URL relativo)
121     }
122   ).catch(function(error){next(error)});
123 };
124
```

si el quiz enviado desde el formulario incluye una imagen, su referencia se añade también al objeto quiz que se va a guardar en la DB.



© Juan Quemada - UPM-DIT

Paso 4: tabla Quiz

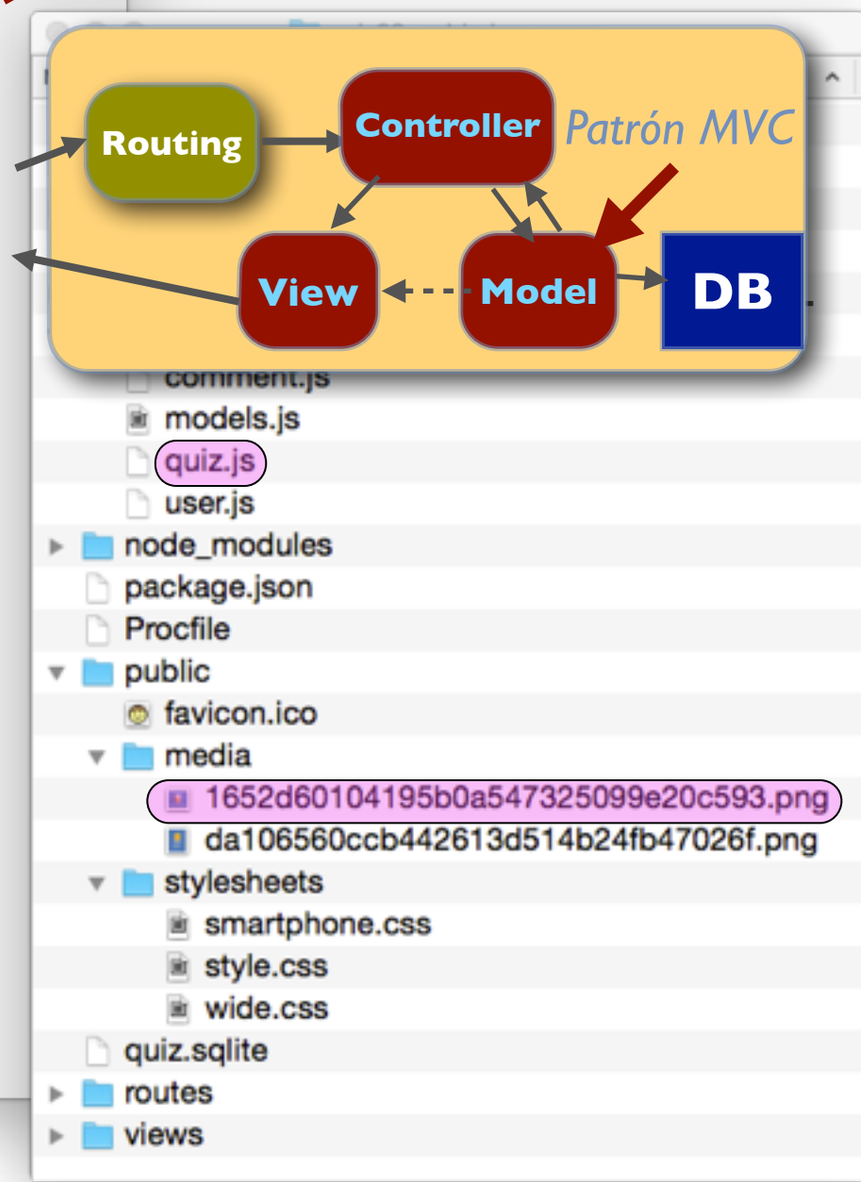
id	pregunta	resp.	UserId	image
1	Capital de Italia	Roma	2	1652d60104195b0a547325099.png
2	Capital de Portu.	Lisboa	2	
3	.....	.....		

quiz.js — quiz23_subir_imagenes UNREGISTERED

```
1  // Definicion del modelo de Quiz con validación
2
3  module.exports = function(sequelize, DataTypes) {
4    return sequelize.define(
5      'Quiz',
6      { pregunta: {
7          type: DataTypes.STRING,
8          validate: { notEmpty: {msg: "-> Falta Pregunta"}}
9        },
10       respuesta: {
11         type: DataTypes.STRING,
12         validate: { notEmpty: {msg: "-> Falta Respuesta"}}
13       },
14       image: {
15         type: DataTypes.STRING
16       }
17     }
18   );
19 }
```

quiz.js

Nuevo campo guarda el nombre del fichero en el directorio public/media



© Juan Quemada - UPM-DIT

Paso 5:"new.ejs" y "edit.ejs"

<input type="file" .....>
se visualiza así

multipart/form-data
codifica y agrupa varios formatos en un único bloque.

quizes/edit.ejs

```
edit.ejs — quiz23_subir_imagenes                UNREGISTERE
1  Edite la pregunta <p>
2
3  <form method="post"
4        action="/quizes/<%= quiz.id %>?_method=put"
5        enctype="multipart/form-data">
6    <% include _form.ejs %>
7  </form>
```

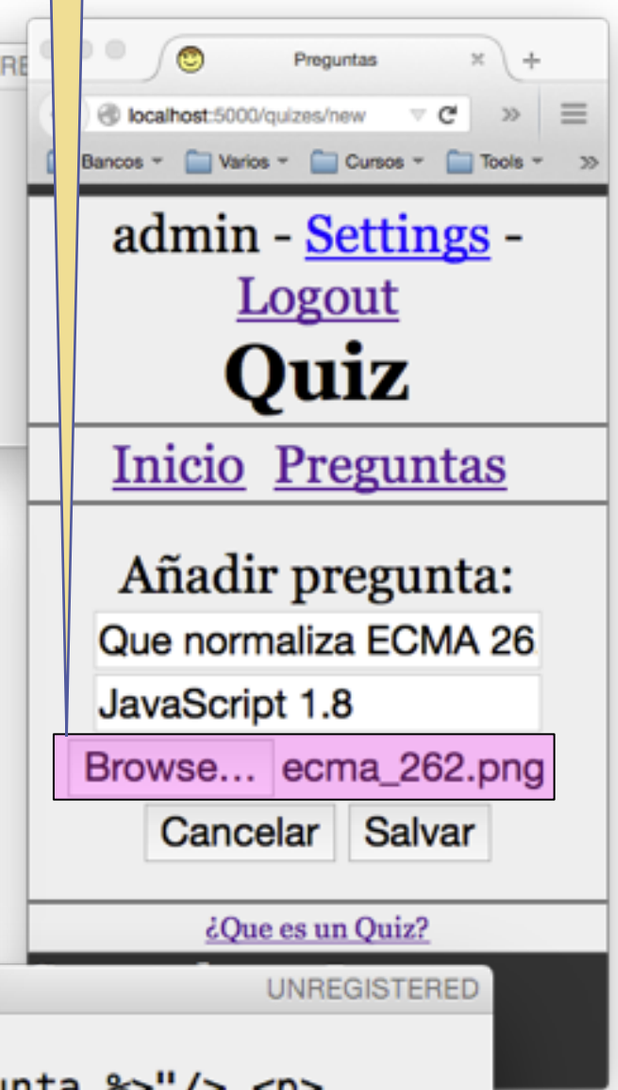
quizes/new.ejs

```
new.ejs — quiz23_subir_imagenes                 UNREGISTERED
1  Añadir pregunta: <p>
2
3  <form method="post"
4        action="/quizes/create"
5        enctype="multipart/form-data">
6    <% include _form.ejs %>
7  </form>
```

<input type="file" .....>
permite subir ficheros a un servidor codificados en formato multipart/form-data

quizes/_form.ejs

```
_form.ejs — quiz23_subir_imagenes               UNREGISTERED
1  <label for="preg" class="rem">Pregunta:</label>
2  <input type="text" id="preg" name="quiz[pregunta]" value="<%= quiz.pregunta %>"/> <p>
3  <label for="resp" class="rem">Respuesta:</label>
4  <input type="text" id="resp" name="quiz[respuesta]" value="<%= quiz.respuesta %>"/> <p>
5  <label for="image" class="rem">Imagen:</label>
6  <input type="file" accept="image/*" id="image" name="image"/><p>
7  <a href="/quizes"><button type="button">Cancelar</button></a>
8  <input type="submit" value="Salvar">
```



© Juan Quemada - UPM-DIT

Paso 6: Vista "show.ejs"

La imagen se muestra con los quizes para poder realizar preguntas sobre la imagen

smartphone.css — quiz23_subir_imagenes UNREGISTERED

```
1  body                { background: #333; }
2
3  #page-wrap:after { content: "Smartphone Layout"; }
4
5  h2 span, section h3, label.rem
6                      { display: none; }
7  body                { font-size: xx-large; }
8  input, button       { font-size: x-large; }
9  footer              { font-size: large; }
10 .quiz_img           { float: none; }
```

style.css UNREGISTERED

```
11 nav span            { margin: 0px 10px 0px 0px;}
12
13 #ErrorMsgs          { color: #f61; font-size: smaler}
14 .right              {float: right;}
15
16 .quiz_img           { height: 150px; float: left;}
```

show.ejs — quiz23_subir_imagenes UNREGISTERED

```
1  <%if (quiz.image){%>
2  <p><img class="quiz_img" src="/media/<%= quiz.image %>"></p>
3  <%}%>
4  <form method="get" action="/quizes/<%= quiz.id %>/answer">
5    <%= quiz.pregunta %> <p>
6    <input type="text" name="respuesta" value="Respuesta"/>
7    <input type="submit" value="Enviar">
8  </form>
9  ................
```

show.ejs

© Juan Quemada - UPM-DIT

Final del tema

©

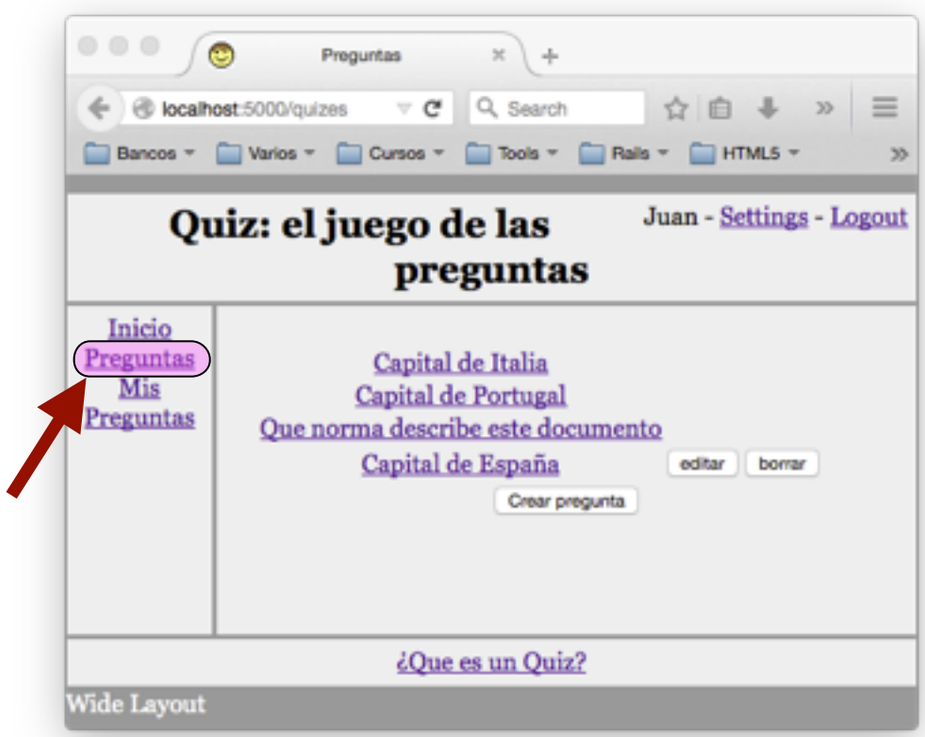
Quiz 24: Mis preguntas

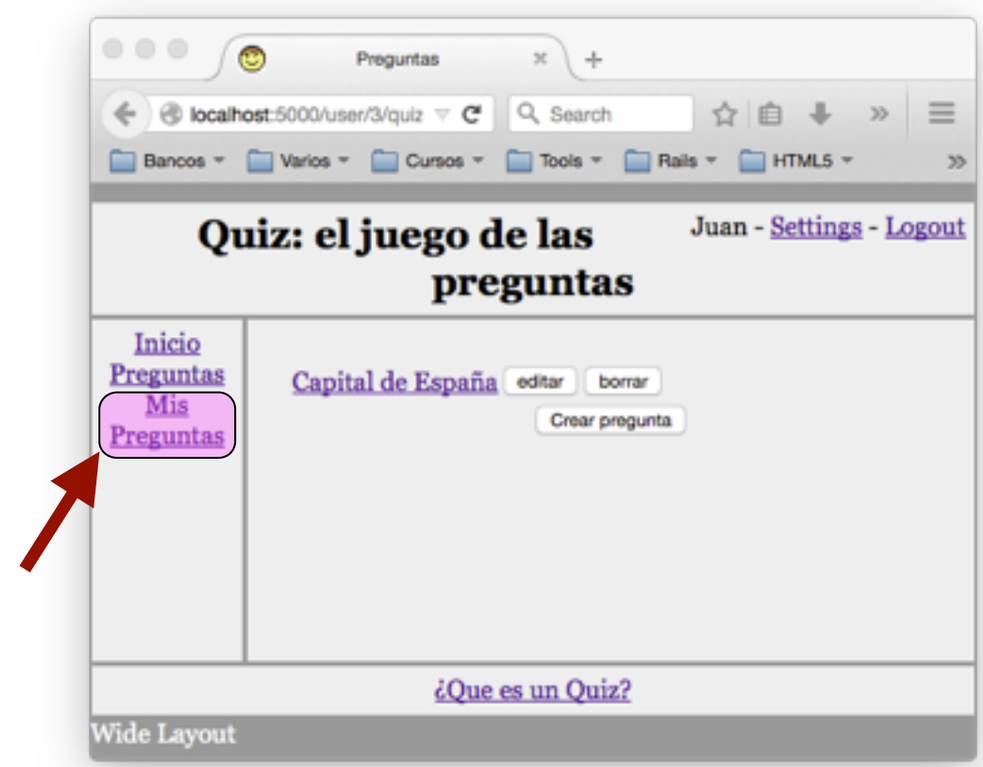
Juan Quemada, DIT - UPM
Enrique Barra, DIT - UPM
Alvaro Alonso, DIT - UPM

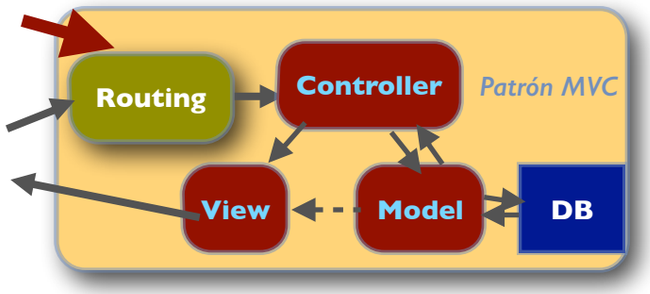
Quiz 24: Mis preguntas

Objetivo: Añadir una nueva vista de los quizes accesible desde la barra de navegación, donde un usuario autenticado puede ver la lista de sus preguntas

- ✦**Paso 1**: Crear ruta **GET /user/1/quizes** para acceso a vista con quizes del usuario
- ✦**Paso 2**: Adaptar acción **index** de **quiz_controller.js**
- ✦**Paso 3**: Añadir enlace **mis_preguntas** a barra de navegación
- ✦**Paso 4**: Transformar la **rutas** de los enlaces de **quizes/index.ejs** en absolutas (corregir error)
- ✦**Paso 5**: Guardar versión (commit) con git y desplegar en Heroku







Paso 1: Añadir GET /user/:userId/quizes

Se añade la primitiva del interfaz

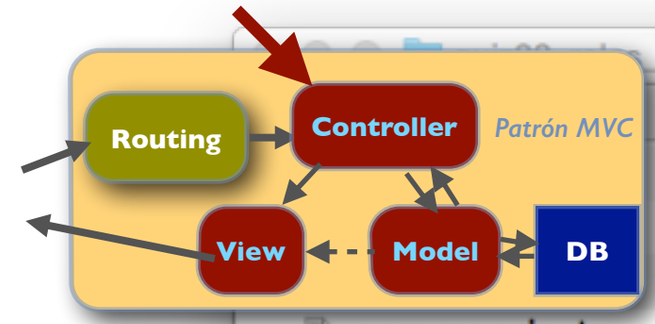
GET /user/:userId/quizes

que invoca la acción **index** de **quiz_controler.js** que ha sido modificada para que atienda tanto a la lista de **Mis_Preguntas** como a la lista con todas las **Preguntas**.

routes/index.js

```
// Definición de rutas de cuenta
router.get('/user',  userController.new);     // formulario sign un
router.post('/user',  userController.create);     // registrar usuario
router.get('/user/:userId(\\d+)/edit',  sessionController.loginRequired, userCon
router.put('/user/:userId(\\d+)',  sessionController.loginRequired, userControll
router.delete('/user/:userId(\\d+)',  sessionController.loginRequired, userContr
router.get('/user/:userId(\\d+)/quizes',  quizController.index);     // ver las
```

Paso 2: quiz_controller.js



La acción index de quiz_controller atiende ahora a 2 primitivas y decide si debe buscar **Mis_Preguntas** o todas las preguntas. Buscará Mis_Preguntas cuando exista la variable **req.user** creada por el **autoload** de usuarios cuando la ruta lleva el parámetro **:userId**,

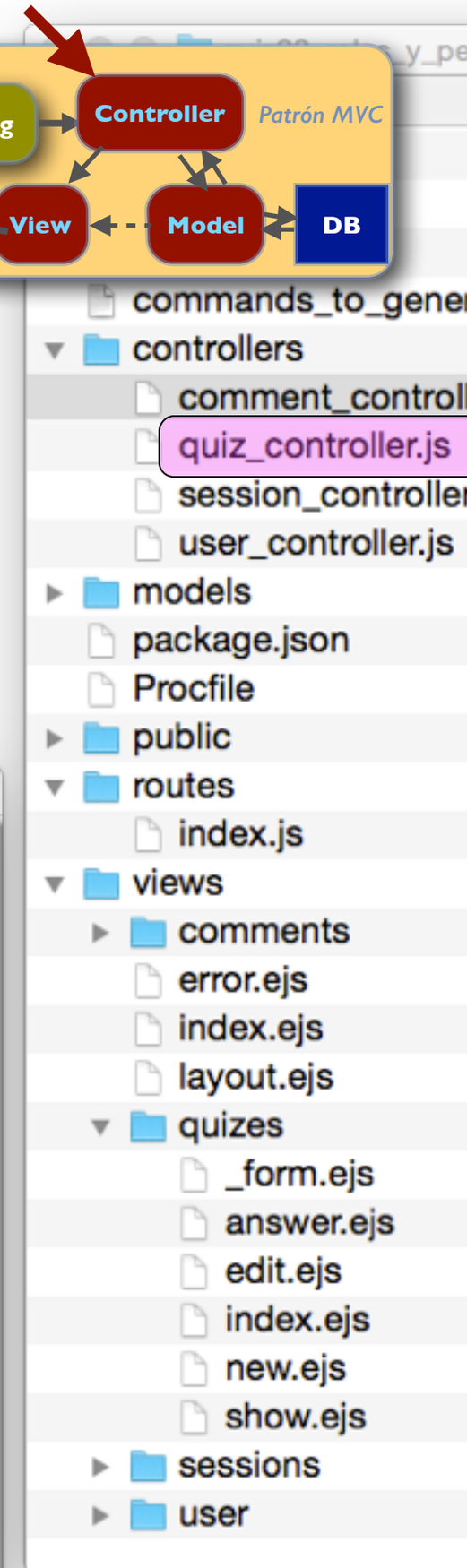
GET /quizes -> req.user no se ha creado
GET /user/:userId/quizes -> autoload de usuarios crea req.user

quiz_controller.js

UNREGISTERED

quiz_controller.js

```
33
34  // GET /quizes
35  // GET /users/:userId/quizes
36  exports.index = function(req, res) {
37    var options = {};
38    if(req.user){   // req.user es creado por autoload de usuario
39                    // si la ruta lleva el parámetro .quizId
40      options.where = {UserId: req.user.id}
41    }
42
43    models.Quiz.findAll(options).then(
44      function(quizes) {
45        res.render('quizes/index.ejs', {quizes: quizes, errors: []});
46      }
47    ).catch(function(error){next(error)});
48  };
49
```



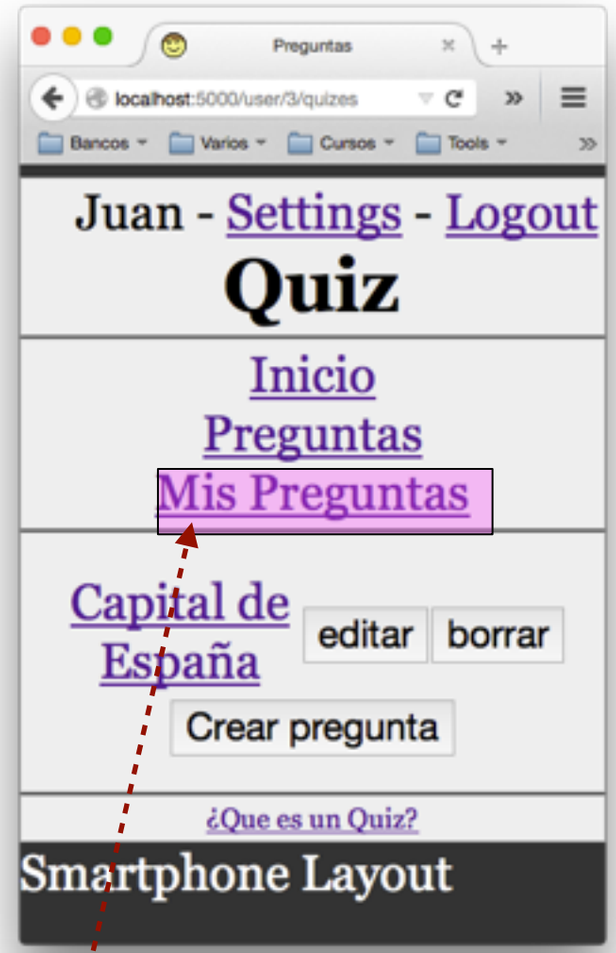
© Juan Quemada - UPM-DIT

Paso 3: Enlace Mis_Preguntas

Se incluye el nuevo enlace en la barra de navegación para acceder a **Mis_Preguntas**

style.css

```
11  nav span                { margin: 0px 10px 0px 0px;
12                            width: 100%;
13                            display: inline-block;}
14
15  #ErrorMsgs              { color: #f61; font-size: smaler}
16  .right                  {float: right;}
```



layout.ejs

```
29  <nav class="main" id="n1" role="navigation">
30    <span><a href="/">Inicio</a></span>
31    <span><a href="/quizes">Preguntas</a></span>
32    <%if(session.user){%>
33      <span><a href="/user/<%=session.user.id%>/quizes">Mis Preguntas</a></span>
34    <%}%>
35  </nav>
```


© Juan Quemada - UPM-DIT

Paso 4: Corregir rutas en “index.ejs”

Las rutas de enlaces y acciones de formulario (**quizes/:quizId** y **quizes/:quizId/edit**) de **quizes/index.ejs** eran rutas relativas (por error). Al incluir la ruta **/user/:userId/quizes** han dejado de funcionar y han tenido que transformarse en rutas absolutas.

index.ejs UNREGISTERED

```
1  <table>
2    <% var i; for (i=0; i < quizes.length; i++) { %>
3      <tr>
4        <td><a href="/quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td>
5        <%if(session.user) {
6          if (session.user.isAdmin || session.user.id === quizes[i].UserId) {%>
7        <td><a href="/quizes/<%= quizes[i].id %>/edit"><button>editar</button></a></td>
8        <td>
9          <form method="post" action="/quizes/<%= quizes[i].id %>?_method=delete">
10           <button type="submit" onClick="return confirm('Borrar: <%= quizes[i].pregunta %>');">
11             borrar
12           </button>
13         </form>
14       </td>
15       <%}
16     }%>
17     </tr>
18   <% } %>
19 </table>
20 <p/>
21 <%if(session.user){%>
22   <a href="/quizes/new"><button>Crear pregunta</button></a>
23 <%}%>
```

index.ejs

Final del tema

©