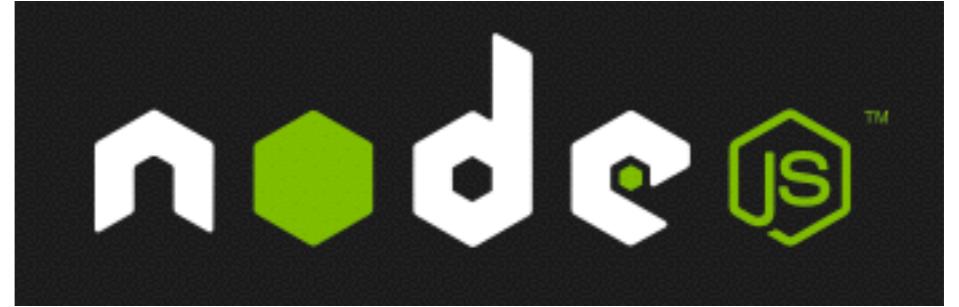




JavaScript



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**

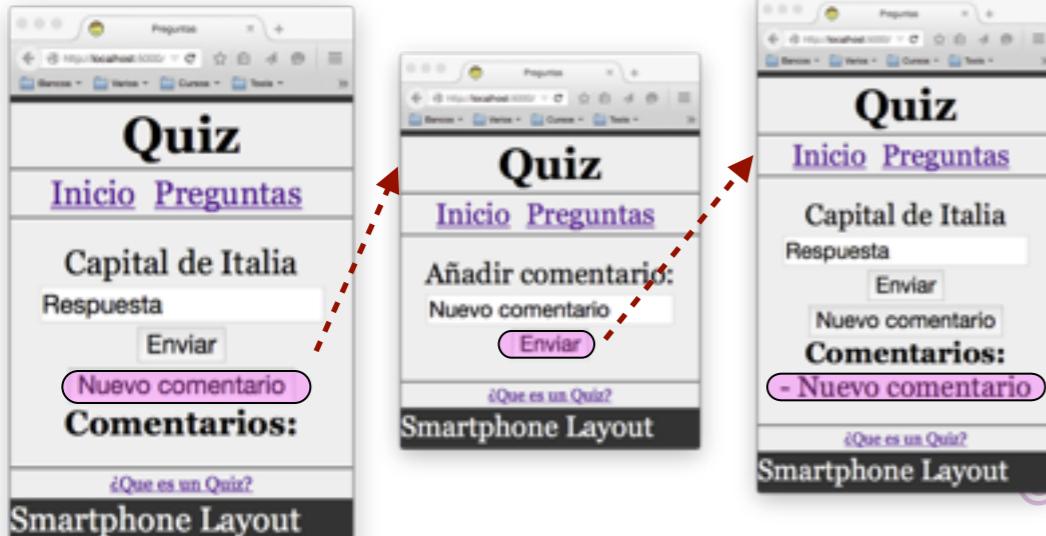
◆ Paso 3: Añadir **POST /quizes/:quizId/comments** para introducir nuevos comentarios en DB

- 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
```

```
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 }
```

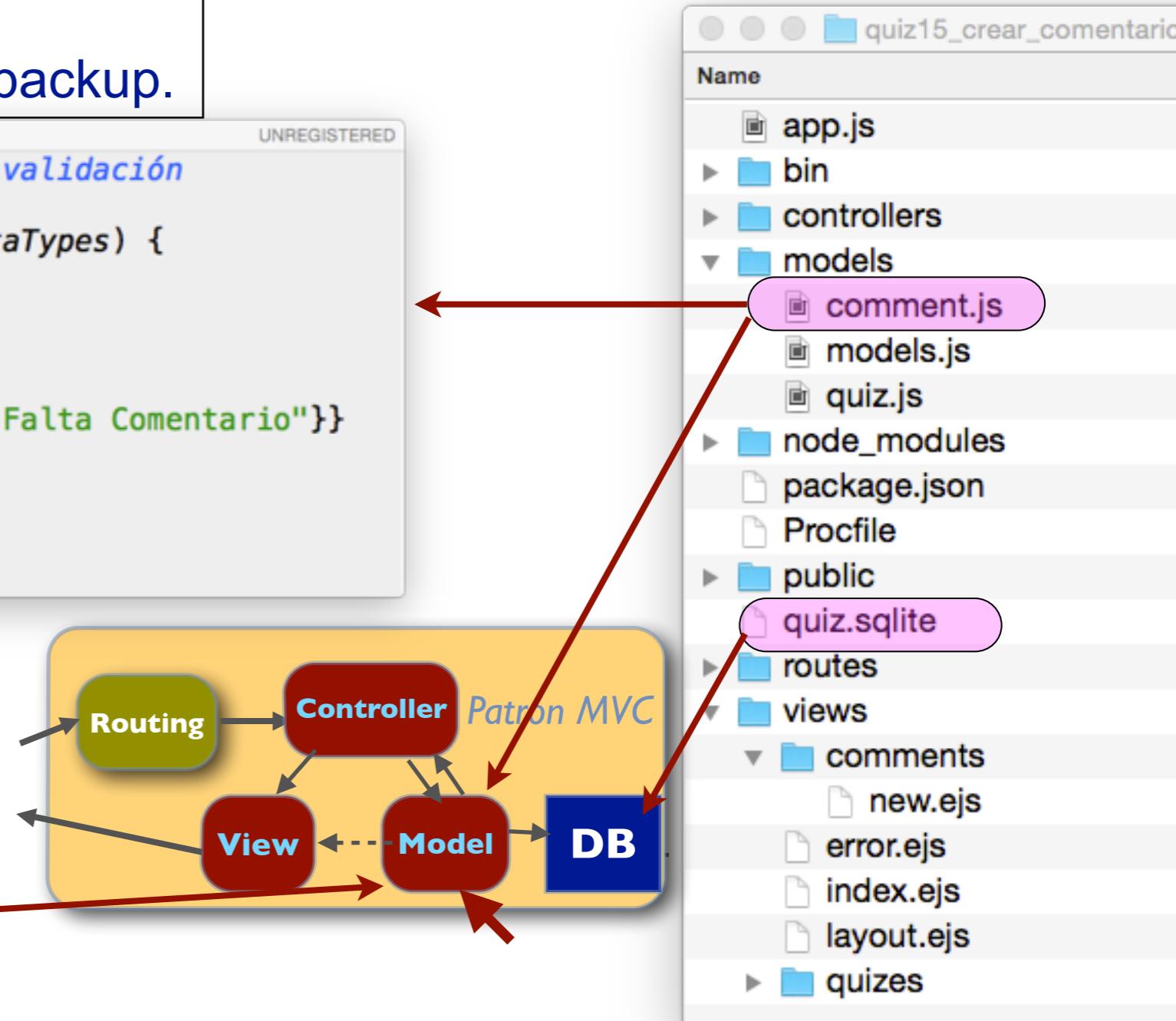
Tabla Comment

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

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

Tiene solo 1 campo de tipo string:

- **texto: DataTypes.STRING**



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**: asocia 1 elemento con otro elemento en otra tabla
 - Por ejemplo, **una pregunta tiene asociado un fichero**
- ◆ Relación **1-a-N**: asocia 1 elemento con N elementos en otra tabla
 - Por ejemplo, **una pregunta puede tener asociados varios comentarios**
- ◆ Relación **N-a-N**: asocia N elementos con N elementos en otra tabla
 - Por ejemplo, **N preguntas gustan (se asocian) a N usuarios**
- ◆ 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 `belongsToMany(...)`

Paso 1b: Relación Quiz a Comment

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

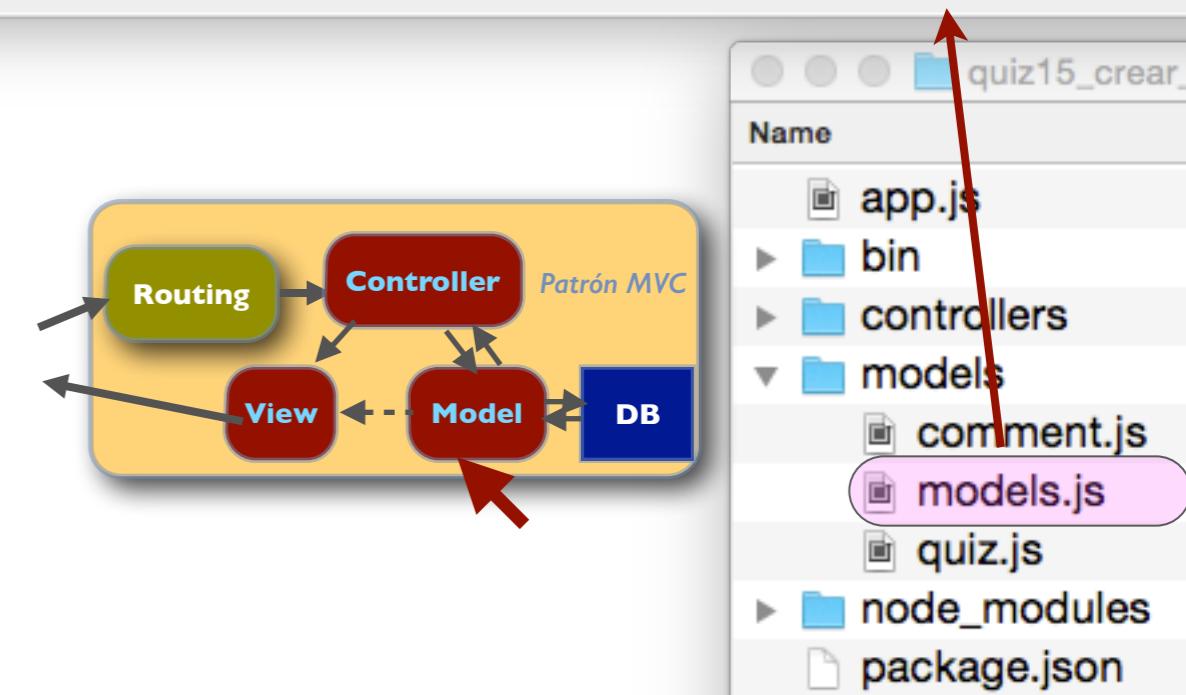
- **Comment.belongsTo(Quiz)**, indica que un comment pertenece a un quiz.
- **QuizhasMany(Comment)**, indica que un quiz puede tener muchos comments.

La relación añade la columna “**QuizId**” en la tabla “**Comment**” que contiene la **clave externa (foreign key)**, que indica que quiz esta asociado al comentario.

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

id	texto	QuizId
1	Nuevo coment	1
2	Fue Sintra en el pasado	2
3	Inglés: Rome	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 Comment.belongsTo(Quiz);  
38 Quiz.hasMany(Comment);  
39  
40 exports.Quiz = Quiz; // exportar tabla Quiz  
41 exports.Comment = Comment;  
42 .....
```



Definición de relaciones

◆ Sequelize crea mecanismos para gestionar una relación

- Ver más información en:

- <http://docs.sequelizejs.com/en/latest/api/associations/>
- <http://docs.sequelizejs.com/en/latest/docs/associations/>

◆ Por ejemplo, La relación 1-a-N entre Quiz y Comment define

- El método: `quiz.addComment(comment)`

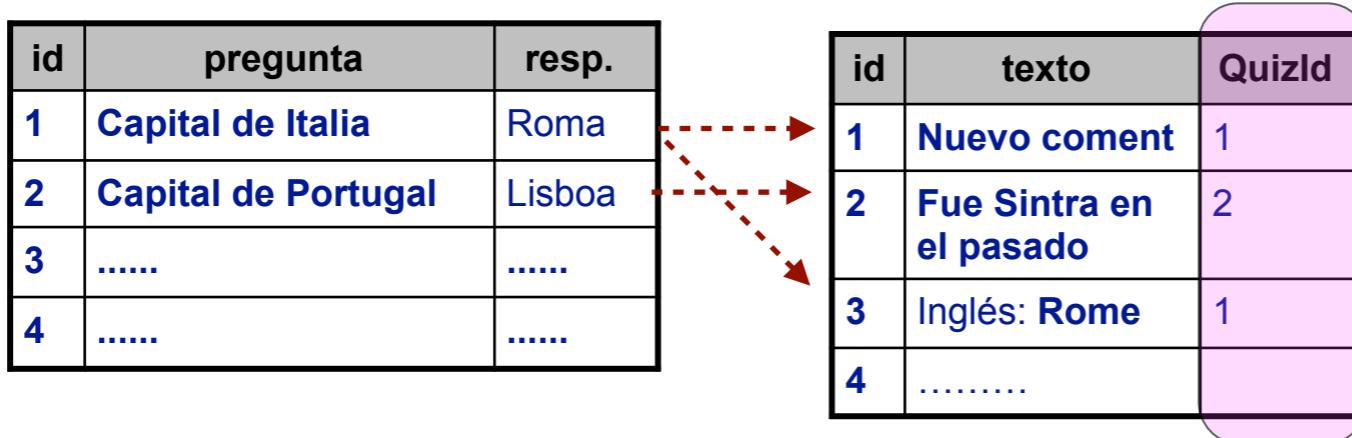
- Permite **añadir** el comentario “comment” a “quiz”

- El método: `quiz.hasComment(comment)`

- Indica si “comment” pertenece a “quiz”

- Añade propiedad **Comentarios** (con array de comentarios) al objeto buscado

- `Quiz.find({ where: { id: Number(QuizId)}}, include: [{ model: models.Comment }])`



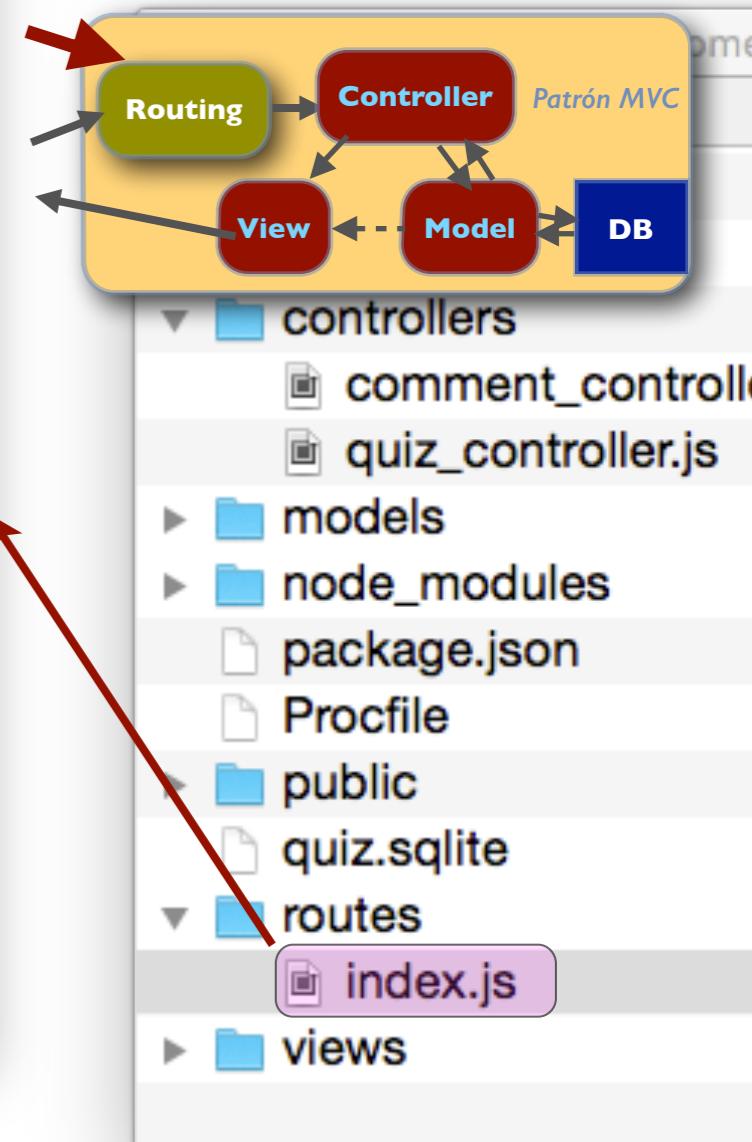
Definimos las siguientes 2 nuevas primitivas en el interfaz REST:

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

```
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',
17           quizController.index);
18 router.get('/quizes/:quizId(\d+)',      quizController.show);
19 router.get('/quizes/:quizId(\d+)/answer', quizController.answer);
20 router.get('/quizes/new',                quizController.new);
21 router.post('/quizes/create',           quizController.create);
22 router.get('/quizes/:quizId(\d+)/edit',  quizController.edit);
23 router.put('/quizes/:quizId(\d+)',     quizController.update);
24 router.delete('/quizes/:quizId(\d+)',   quizController.destroy);
25
26 router.get('/quizes/:quizId(\d+)/comments/new', commentController.new);
27 router.post('/quizes/:quizId(\d+)/comments',    commentController.create);
28
29 module.exports = router;
```

Pasos 2b y 3b: rutas

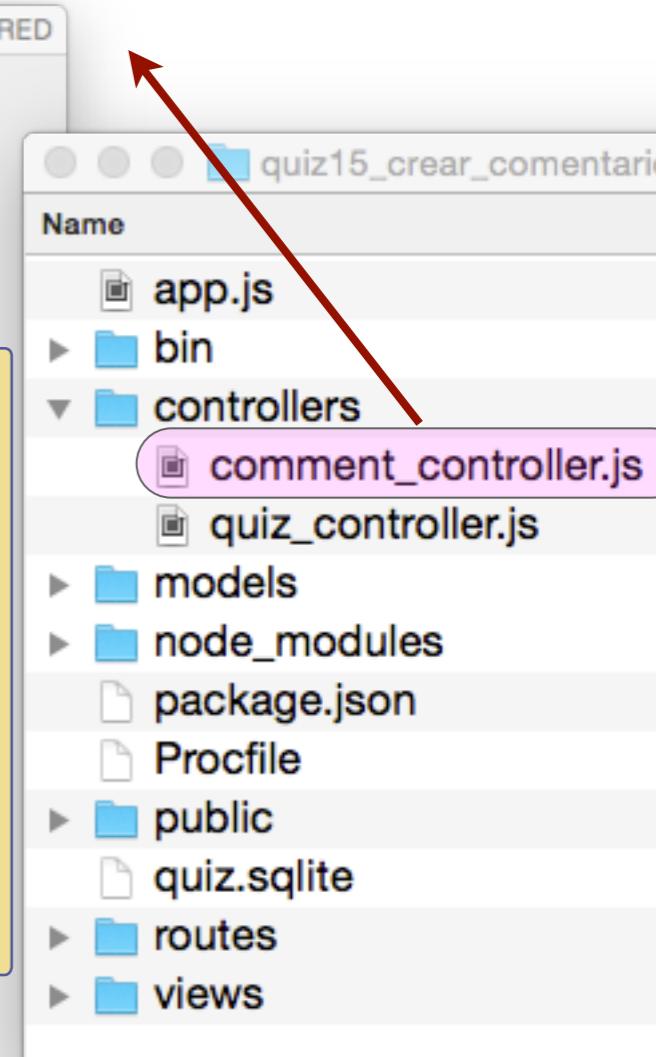


Creamos el controlador de comentarios solo con acciones new y create:

- 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.

```
1 var models = require('../models/models.js');
2
3 // GET /quizes/:quizId/comments/new
4 exports.new = function(req, res) {
5   res.render('comments/new.ejs', {quizid: req.params.quizId, errors: []});
6 };
7
8 // POST /quizes/:quizId/comments
9 exports.create = function(req, res) {
10   var comment = models.Comment.build(
11     { texto: req.body.comment.texto,
12       QuizId: req.params.quizId
13     });
14
15   comment
16     .validate()
17     .then(
18       function(err){
19         if (err) {
20           res.render('comments/new.ejs',
21             {comment: comment, quizid: req.params.quizId, errors: err.errors});
22         } else {
23           comment // save: guarda en DB campo texto de comment
24             .save()
25             .then( function(){ res.redirect('/quizes/'+req.params.quizId)})
26           } // res.redirect: Redirección HTTP a lista de preguntas
27         }
28       ).catch(function(error){next(error)});
29     };
30   
```

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.



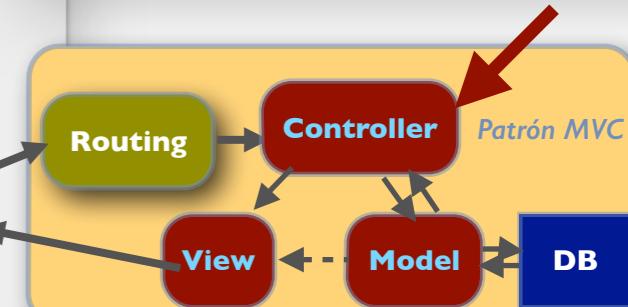
UNREGISTERED

comment_controller.js

quiz15_crear_comentario

Name

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



Patrón MVC

Pasos 2a y 3a: comment_controller

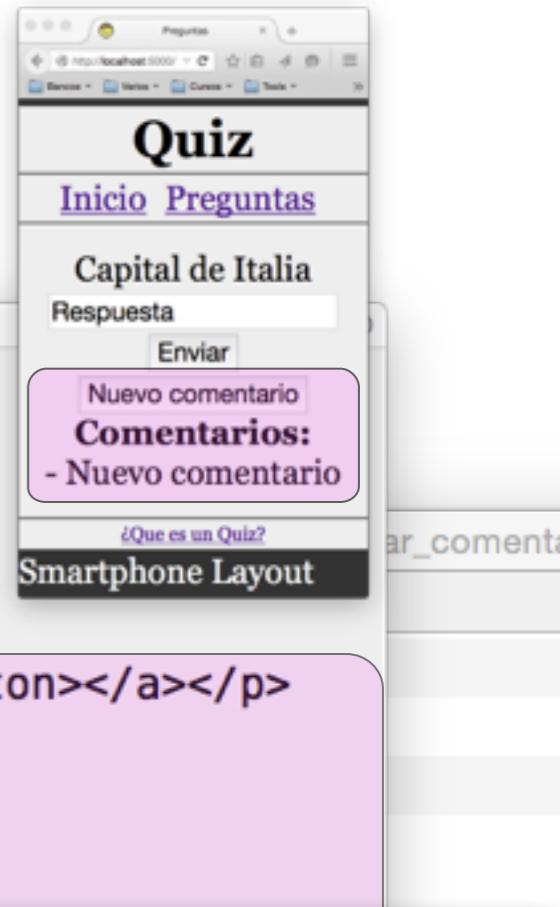
8

© Juan Quemada - UPIM-DTI

Pasos 2c y 4b: vistas

```
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 4b

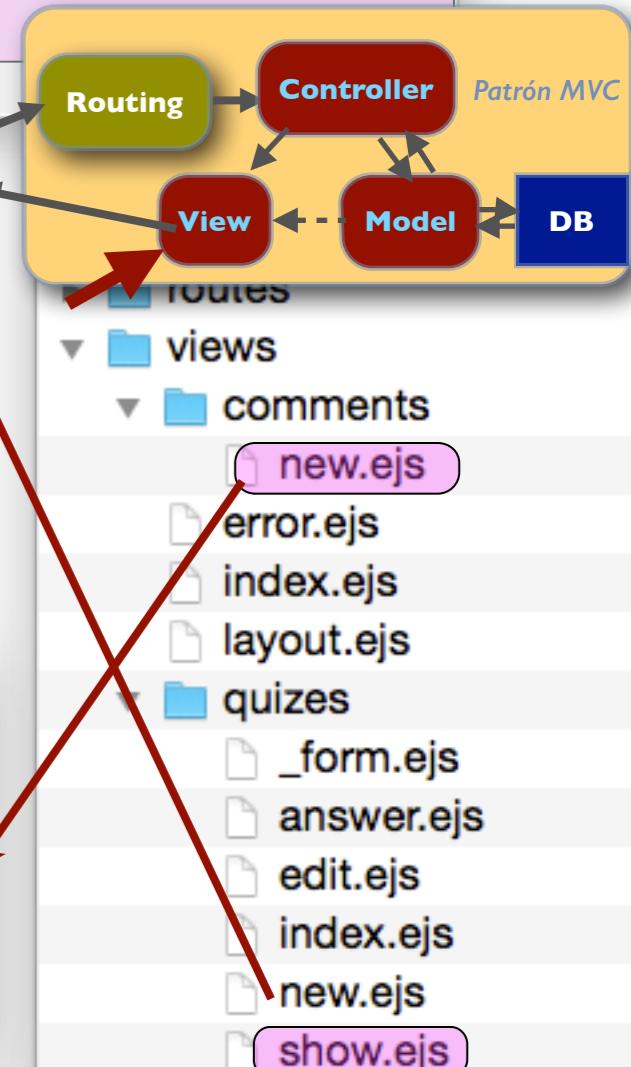
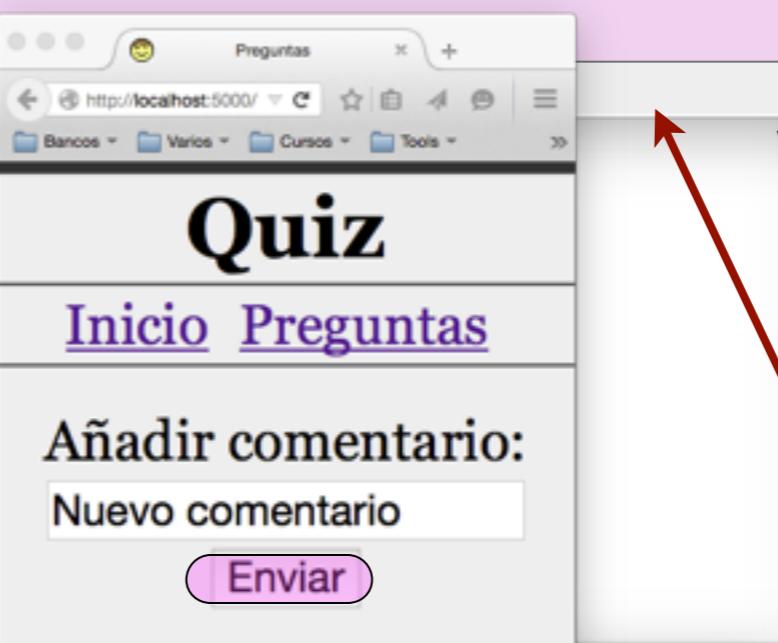


Se modifica la vista **view/quizes/show.ejs** para gestionar los comentarios de una pregunta.

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

```
1 Añadir comentario: <p>
2
3   <form method="post" action="/quizes/<%=quizid%>/comments/" >
4     <input type="text" id="comment" name="comment[texto]" value="" /> <p>
5       <button type="submit">Enviar</button>
6     </form>
7
```

Paso 2c



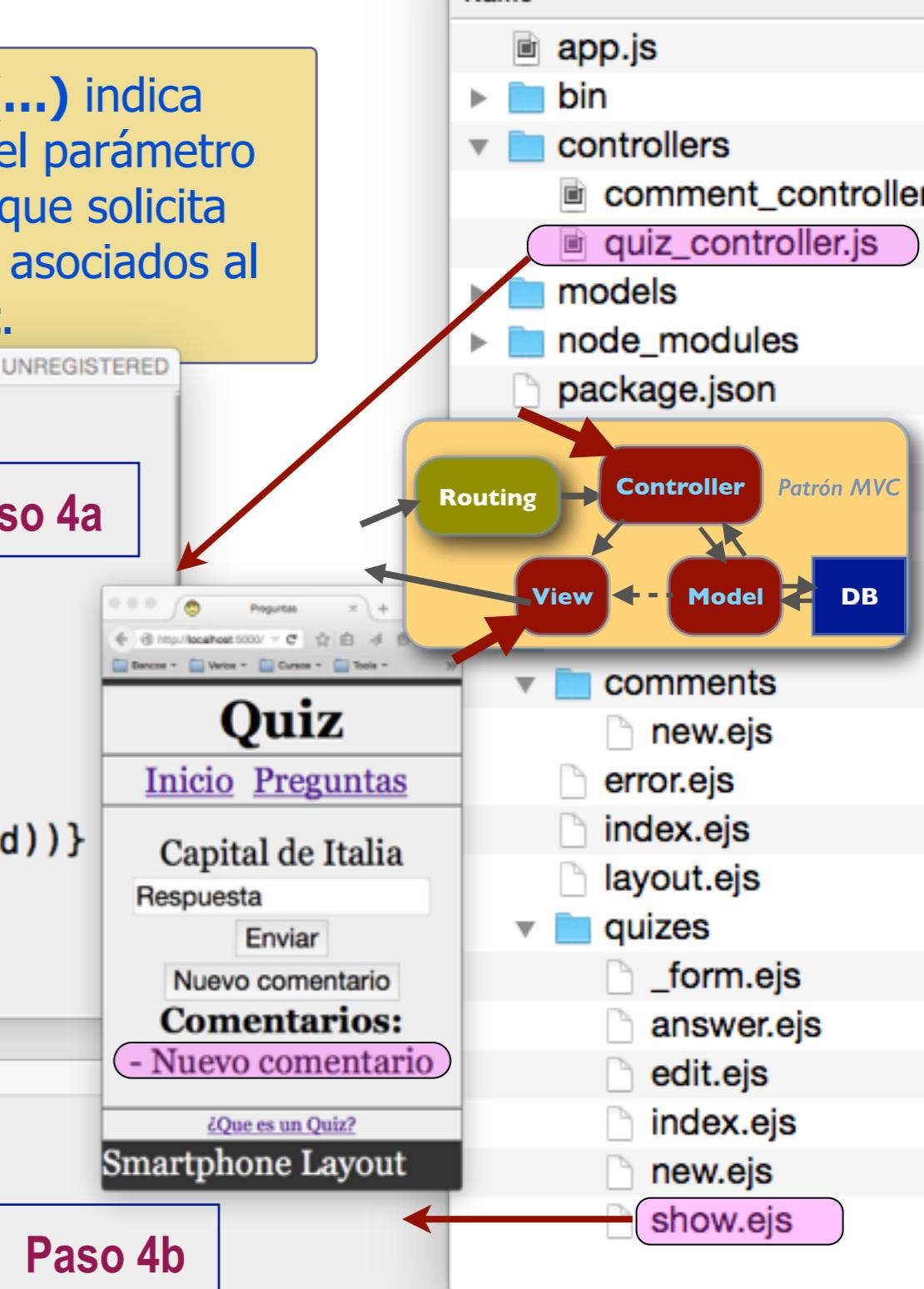
El parámetro “**where: { id: Number(quizId) }**” de **find(...)** indica buscar el quiz identificado por **quizId**. Pero además está el parámetro “**include: [{model: models.Comment}]** de **find(...)**, que solicita cargar en la propiedad **quiz.Comments**, los comentarios asociados al quiz a través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

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

```
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  <%}%>
```

Paso 4a

Paso 4b



Paso 4: visualizar comentarios

Reiniciar la DB

◆ Si se modifica el modelo

- Debe migrarse la DB
 - En este curso no se ven migraciones,
 - los interesados pueden buscar en
 - <http://sequelize.readthedocs.org/en/latest/docs/migrations/>

◆ Si no se utilizan migraciones

- lo mas sencillo es reiniciar o borrar la DB
 - aunque se perderán los datos guardados

◆ Para borrar sqlite

- Debemos borrar el fichero “**quiz.sqlite**”

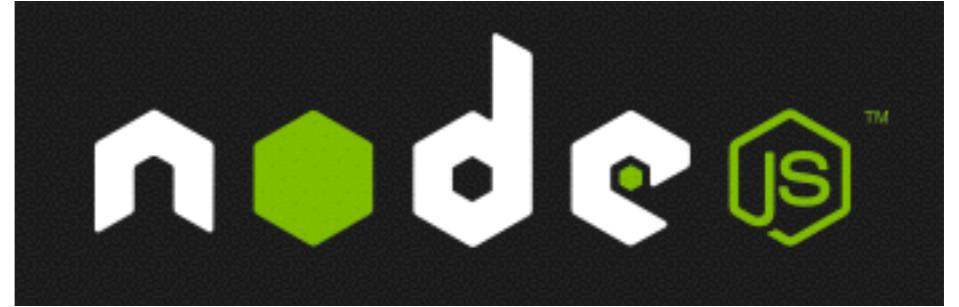
◆ Para borrar postgres en Heroku

- Ir al Dashboard y borrarla
- También se puede borrar con comandos

The screenshot shows the Heroku Dashboard for the application 'quiz-2015'. The top navigation bar includes tabs for Resources, Code, Metrics, Activity, Access, and Settings. Below the tabs, the URL <https://dashboard.heroku.com> is displayed. A red dashed arrow points from the 'Resources' tab down to the 'Add-ons' section. In the 'Add-ons' section, there is one entry for 'Heroku Postgres :: Gold' which is currently set to 'Dev' mode. A yellow callout box highlights this entry with the text: 'Borrar la DB y crear una desde cero para que al nueva versión se configure correctamente.' (Delete the DB and create a new one from scratch so that when the new version is deployed, it is configured correctly). At the bottom of the dashboard, there are links for heroku.com/home, Blog, Careers, Privacy Policy, and Feedback.



JavaScript



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**

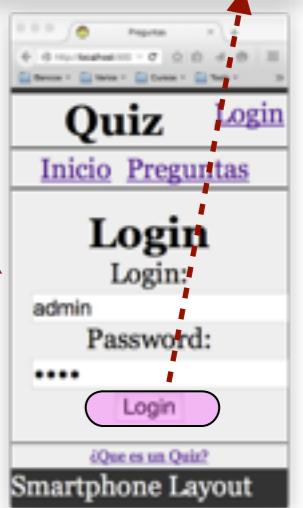
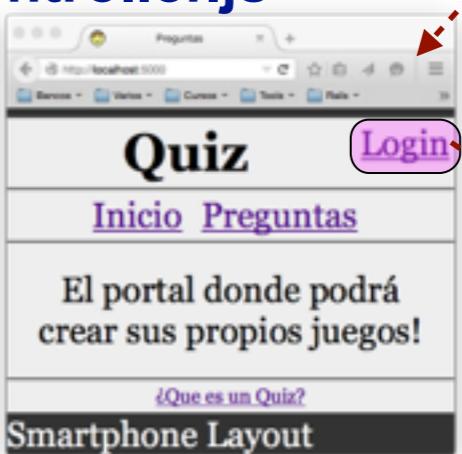
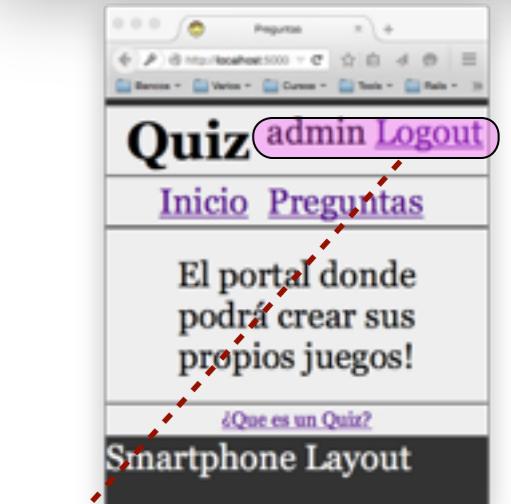
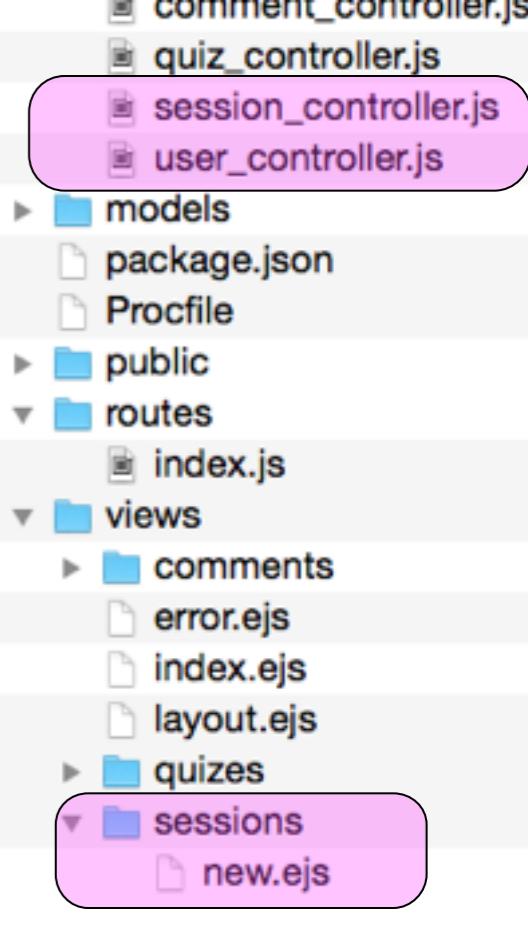
- a: Añadir acción **new** para la ruta **GET /login** para cargar formulario de login
- b: Añadir acción **create** para la ruta **POST /login** para crear la sesión de usuario
- c: Añadir acción **destroy** para a ruta **GET /logout** para destruir la sesión

◆ 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 var routes = require('./routes/index');
11
12 var app = express();
13
14 // view engine setup
15 app.set('views', path.join(__dirname, 'views'));
16 app.set('view engine', 'ejs');

17 app.use(partials());
18
19 // uncomment after placing your favicon in /public
20 // app.use(favicon(__dirname + '/public/favicon.ico'));
21 app.use(cookieParser('Quiz 2015'));
22 app.use(logger('dev'));
23 app.use(bodyParser.json());
24 app.use(bodyParser.urlencoded());
25
26 app.use(session());
27
28 app.use(methodOverride('_method'));
```

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

// 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

```
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",
  "express-session": "~1.0.3",
  "method-override": "^2.3.1",
  "morgan": "~1.3.0",
  "pg": "^4.1.1",
  "sequelize": "^2.0.0-rc4",
  "serve-favicon": "~2.1.3"
}
```

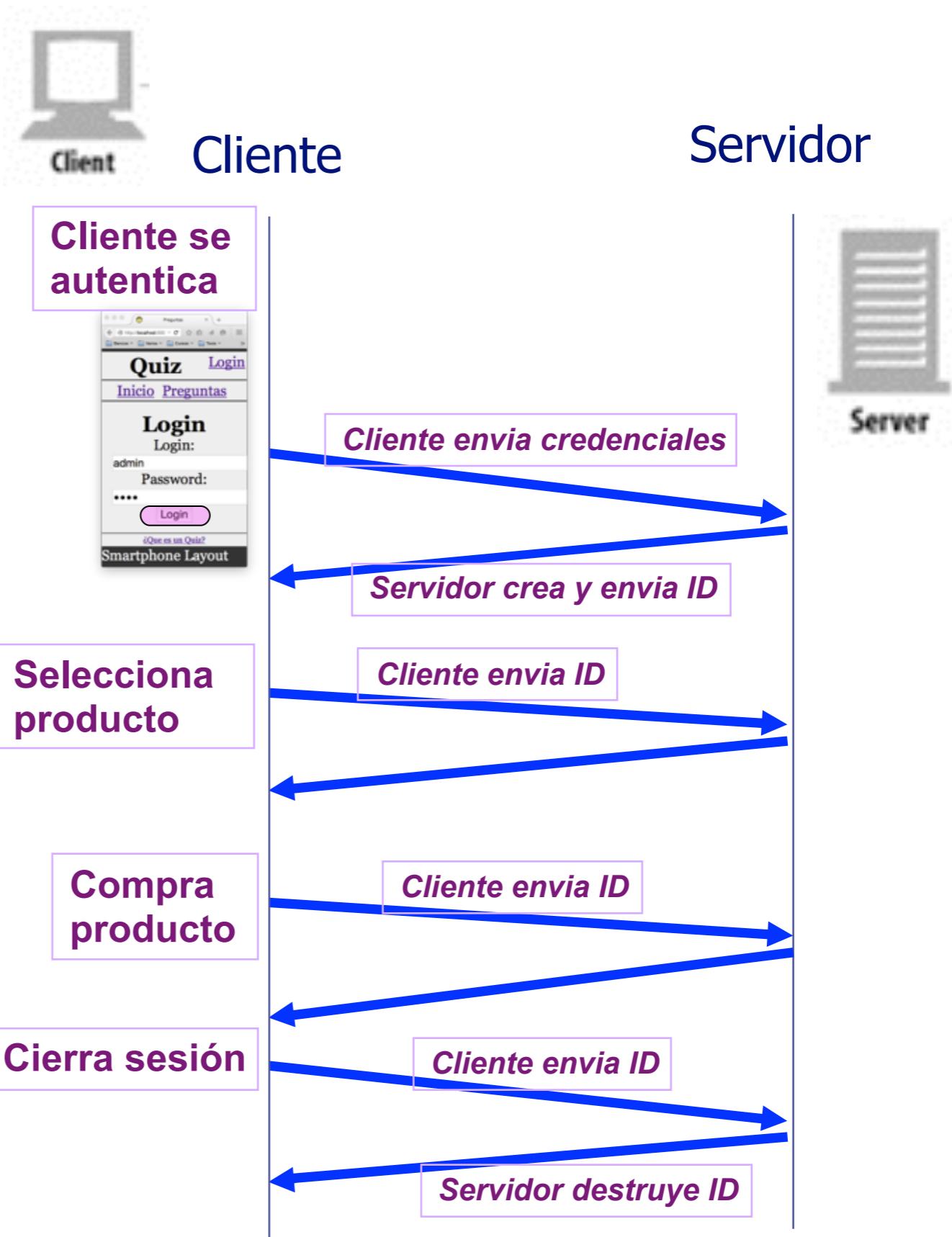
Sesiones

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).

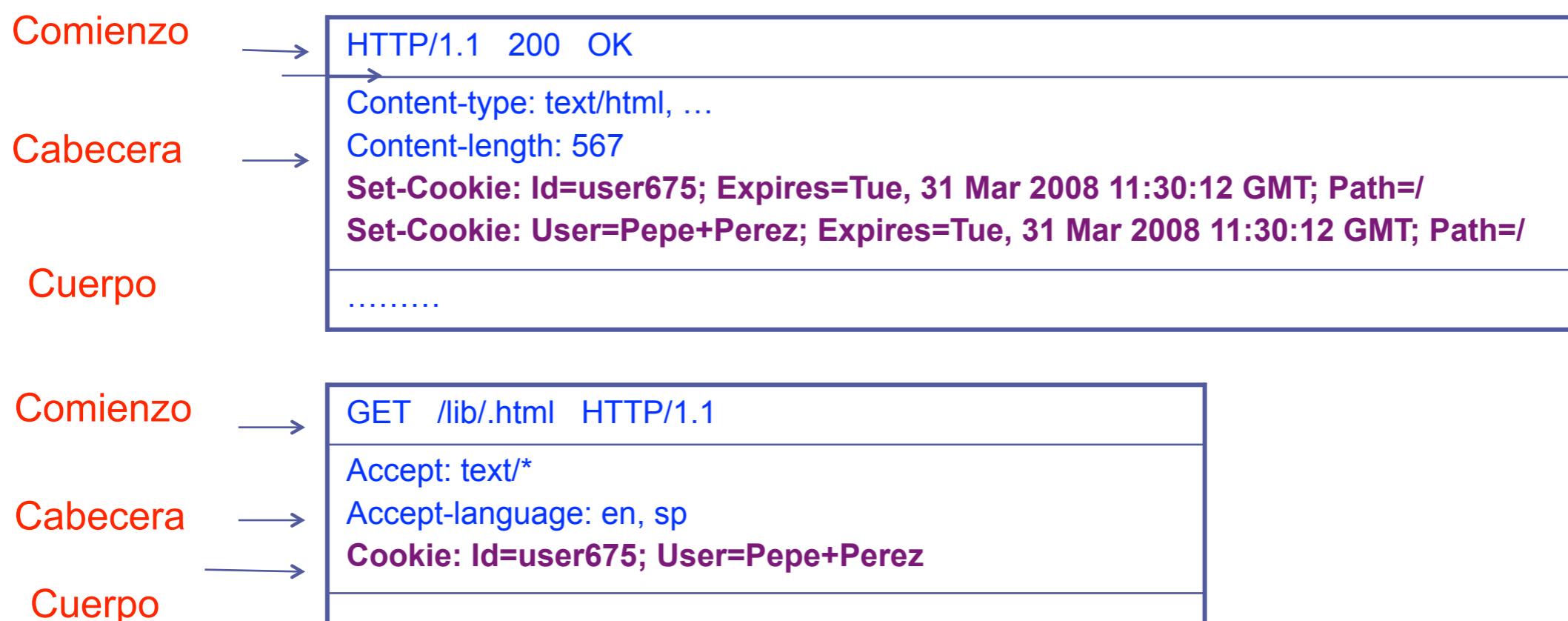
El identificador de sesión se suele guardar en:

- **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.



Sesión: Cookie connect.sid

Bancos Varios Cursos Tools Rails HTML5 HTML5-RTC MiTwitter CTING Login

Quiz: el juego de las preguntas

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

Inspector Console Debugger Style Editor Performance Network Filter output

Net CSS JS Security Logging Clear

GET http://localhost:5000/
GET http://localhost:5000/stylesheets/style.css
GET http://localhost:5000/stylesheets/wide.css
GET http://localhost:5000/stylesheets/smartphone.css

HTTP/1.1 200 OK 49ms
HTTP/1.1 200 OK 16ms
HTTP/1.1 200 OK 17ms
HTTP/1.1 200 OK 17ms

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/DBI78txrJ8JCultd4

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.

A screenshot of the Firefox developer tools Network tab. It shows a request from the client to the server at 'http://localhost:5000/' and several responses from the server back to the client. One of the responses contains a 'connect.sid' cookie. A red arrow points from the text 'El identificador de sesión se guarda en la cookie cifrada:' to this cookie in the response. Below the screenshot, there are two boxes containing explanatory text: one about the 'connect.sid' cookie and another about session parameters being stored on the server.

17

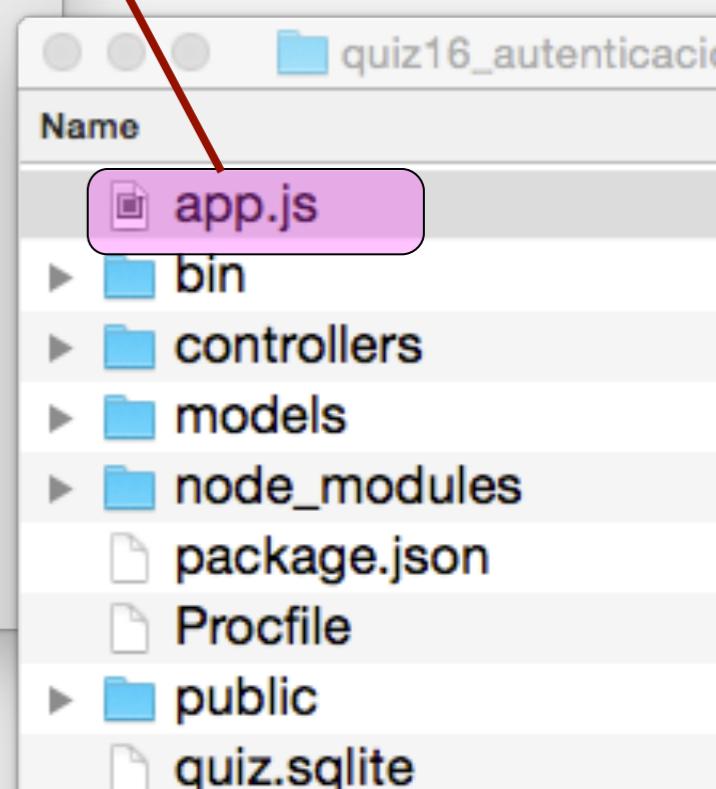
Además definimos en app.js un middleware que realiza 2 funciones:

- **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.redirect** para poder redireccionar a la vista anterior después de hacer login o logout.

Pasos 1b y 1c

```
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.redirect para despues de login
35     if (!req.path.match(/\/login|\/logout/)) {
36         req.session.redirect = req.path;
37     }
38
39     // Hacer visible req.session en las vistas
40     res.locals.session = req.session;    Paso 1b
41     next();
42 });
43
44 app.use('/', routes);
```

Paso 1c

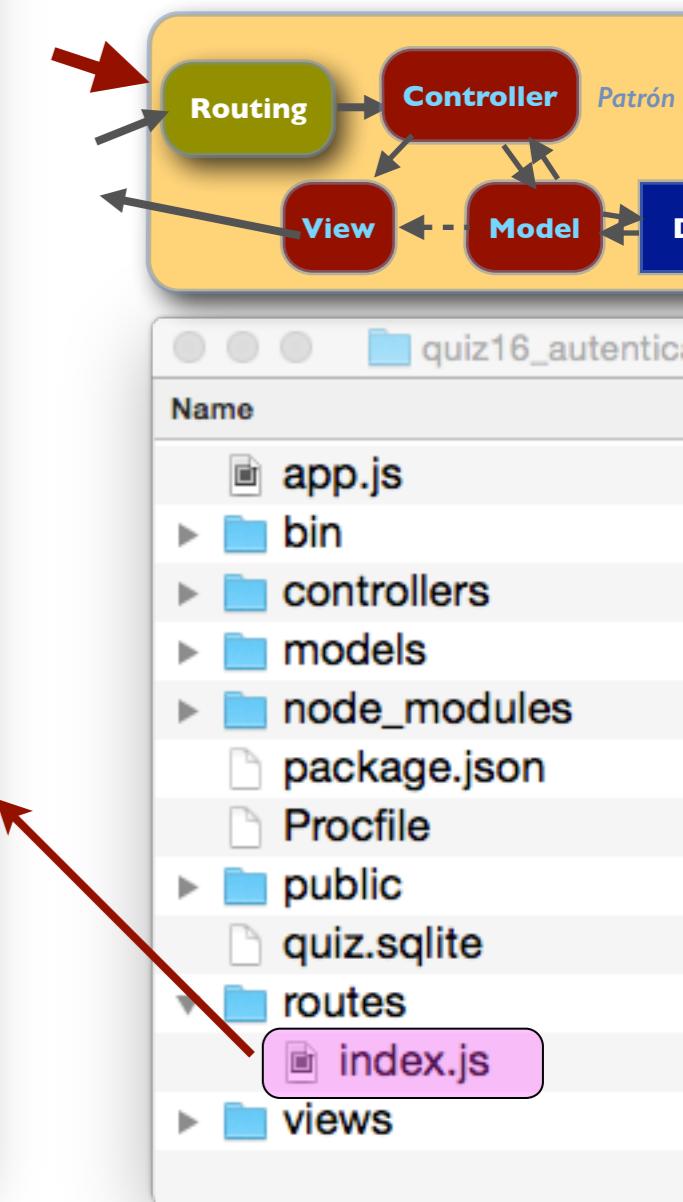


Paso 2

```
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 var sessionController = require('../controllers/session_controller');  
7  
8 // Página de entrada (home page)  
9 router.get('/', function(req, res) {  
  res.render('index', { title: 'Quiz', errors: []});  
11});  
12  
13 // Autoload de comandos con :quizId  
14 router.param('quizId', quizController.load); // autoload :quizId  
15  
16 // Definición de rutas de sesión  
17 router.get('/login', sessionController.new); // formulario login  
18 router.post('/login', sessionController.create); // crear sesión  
19 router.get('/logout', sessionController.destroy); // destruir sesión  
20  
21 // Definición de rutas de /quizes  
22 router.get('/quizes', quizController.index);  
23 router.get('/quizes/:quizId(\d+)', quizController.show);  
24 router.get('/quizes/:quizId(\d+)/answer', quizController.answer);  
25 router.get('/quizes/new', quizController.new);  
26 router.post('/quizes/create', quizController.create);  
27 router.get('/quizes/:quizId(\d+)/edit', quizController.edit);  
28 router.put('/quizes/:quizId(\d+)', quizController.update);  
29 router.delete('/quizes/:quizId(\d+)', quizController.destroy);  
30  
31 // Definición de rutas de comentarios  
32 router.get('/quizes/:quizId(\d+)/comments/new', commentController.new);  
33 router.post('/quizes/:quizId(\d+)/comments', commentController.create);  
34  
35 module.exports = router;  
36
```

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.

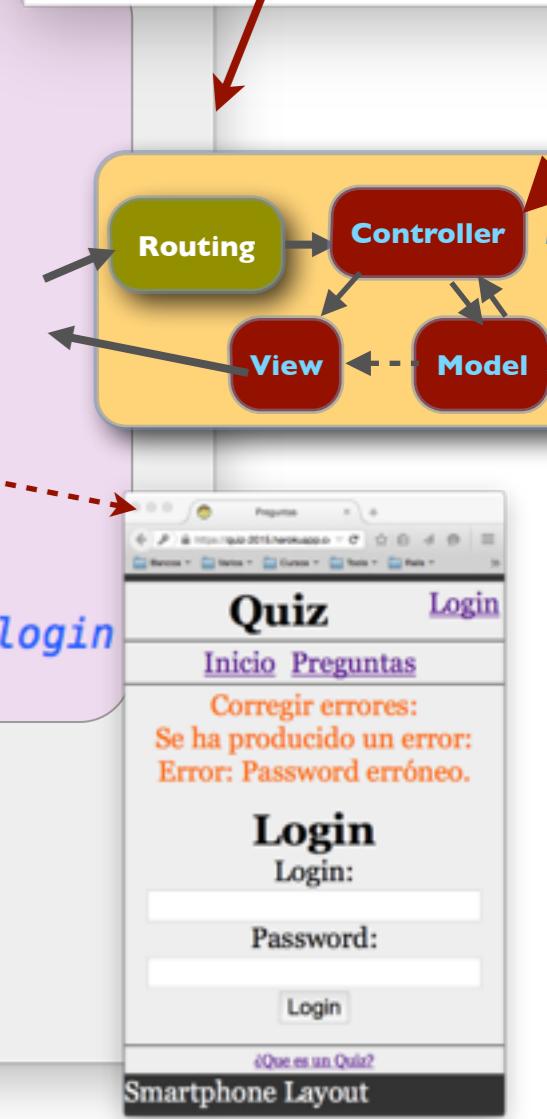
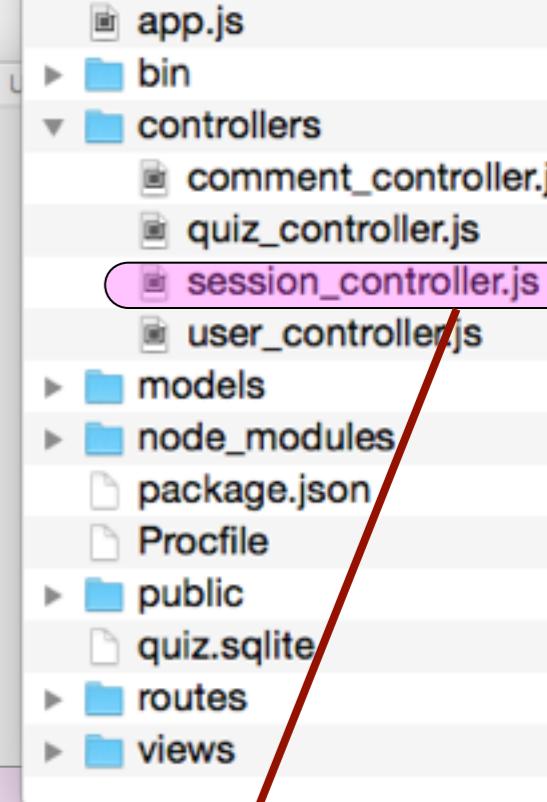


```

1 // Get /login -- Formulario de login
2 exports.new = function(req, res) {
3   var errors = req.session.errors || {};
4   req.session.errors = {};
5
6   res.render('sessions/new', {errors: errors});
7 };
8
9 // POST /login -- Crear la sesion
10 exports.create = function(req, res) {
11
12   var login = req.body.login;
13   var password = req.body.password;
14
15   var userController = require('./user_controller');
16   userController.autenticar(login, password, function(error, user) {
17
18     if (error) { // si hay error retornamos mensajes de error de sesion
19       req.session.errors = [{"message": 'Se ha producido un error: '+error}];
20       res.redirect("/login");
21       return;
22     }
23
24     // Crear req.session.user y guardar campos id y username
25     // La sesion se define por la existencia de: req.session.user
26     req.session.user = {id:user.id, username:user.username};
27
28     res.redirect(req.session.redir.toString()); // redireccion a path anterior a login
29   });
30
31
32 // DELETE /logout -- Destruir sesion
33 exports.destroy = function(req, res) {
34   delete req.session.user;
35   res.redirect(req.session.redir.toString()); // redirect a path anterior a login
36 };

```

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



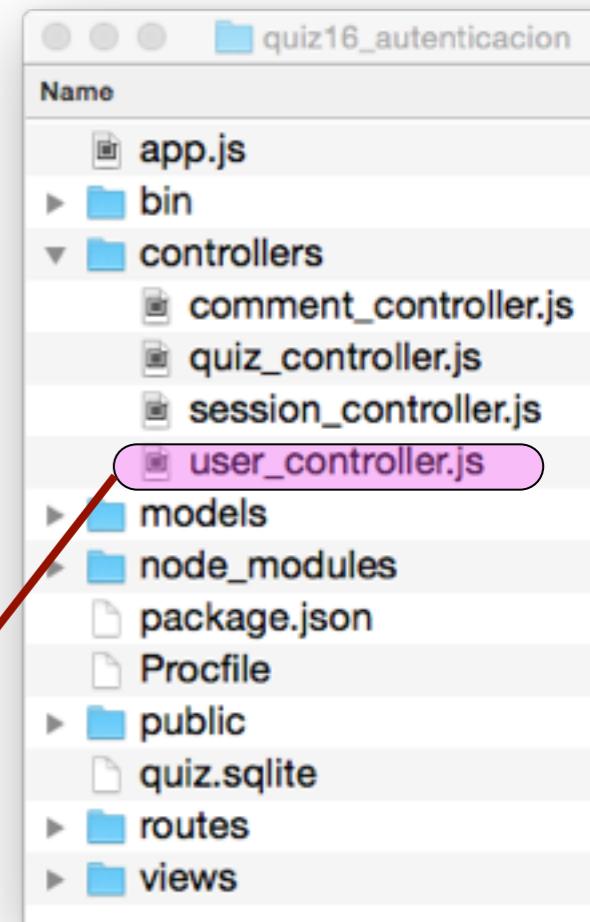
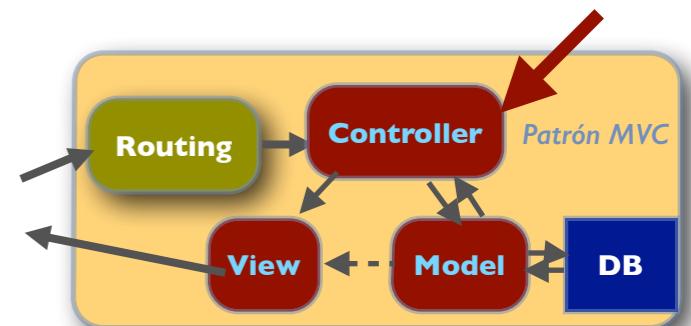
Pasos 3a, 3b y 3c

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.

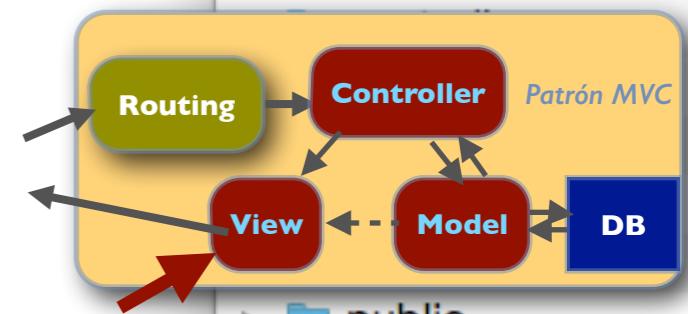


```
user_controller.js
UNREGISTERED
1 var users = { admin: {id:1, username:"admin", password:"1234"}, 
2             pepe: {id:2, username:"pepe", password:"5678"} 
3 };
4 
5 // Comprueba si el usuario esta registrado en users
6 // Si autenticación falla o hay errores se ejecuta callback(error).
7 exports.autenticar = function(login, password, callback) {
8     if(users[login]){
9         if(password === users[login].password){
10             callback(null, users[login]);
11         }
12         else { callback(new Error('Password erróneo.')); }
13     } else { callback(new Error('No existe el usuario.'));}
14 }
```

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.



Quiz admin Logout

Inicio Preguntas

El portal donde podrá crear sus propios juegos!

¿Qué es un Quiz?

Smartphone Layout

```

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

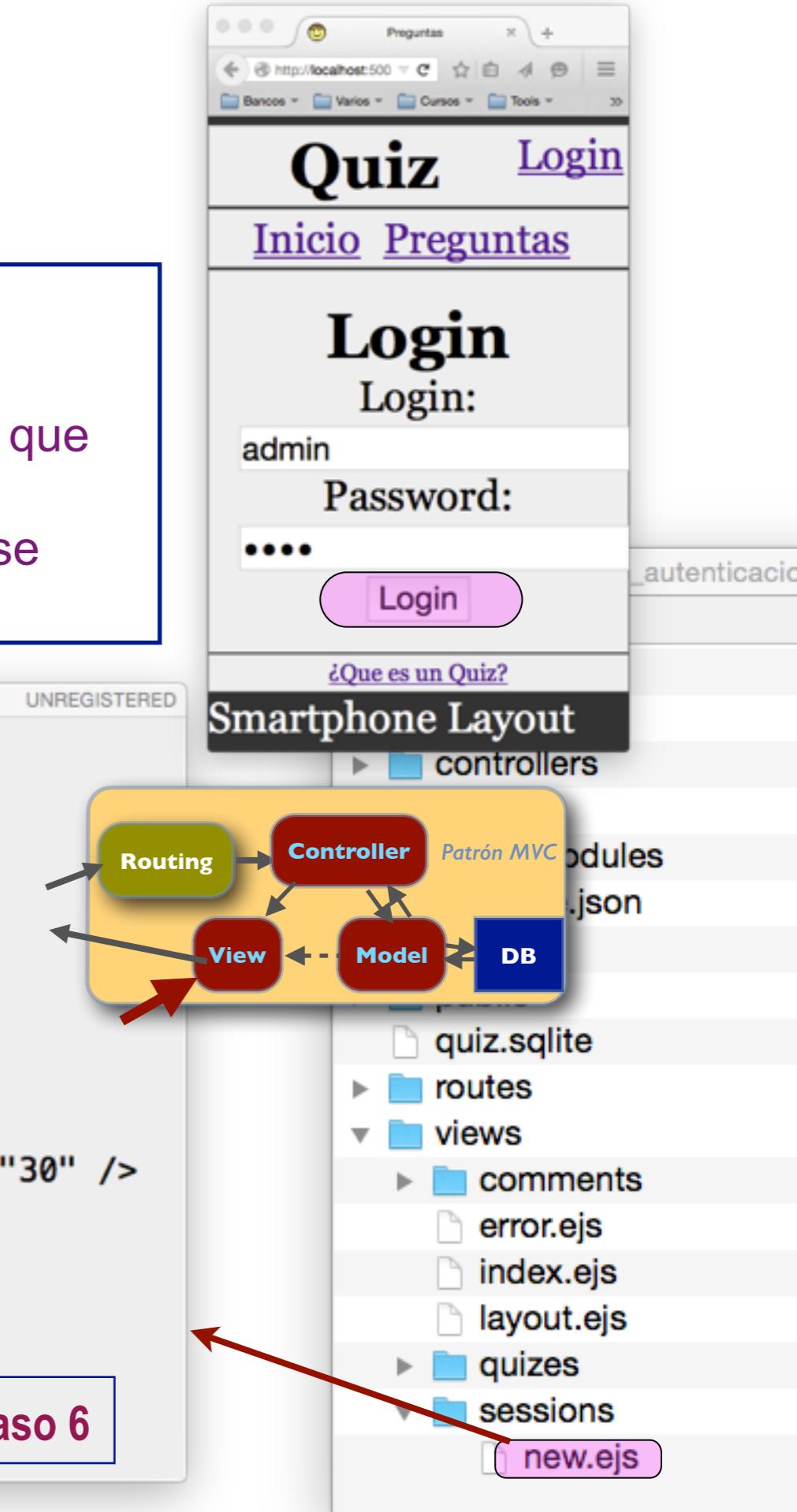
El formulario de autenticación es la única vista que se crea asociada a las sesiones.

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
```

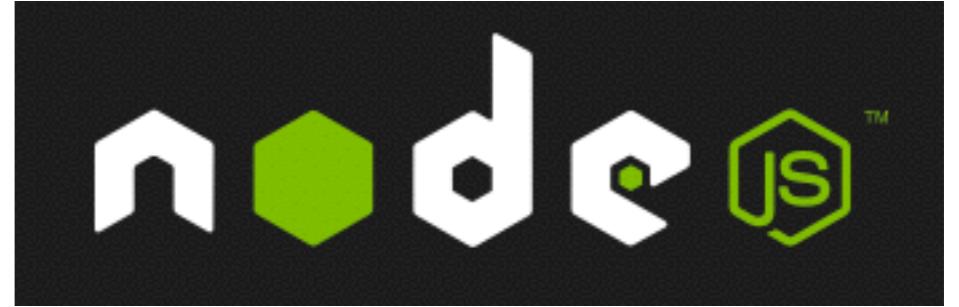
```
1 <h2>Login</h2>
2
3 <form method='POST' action='/login'>
4
5   <div class="field">
6     <label for="login">Login:</label><br />
7     <input type="text" id="login" name="login" size="30" />
8   </div>
9
10  <div class="field">
11    <label for="password">Password:</label><br />
12    <input type="password" 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





JavaScript



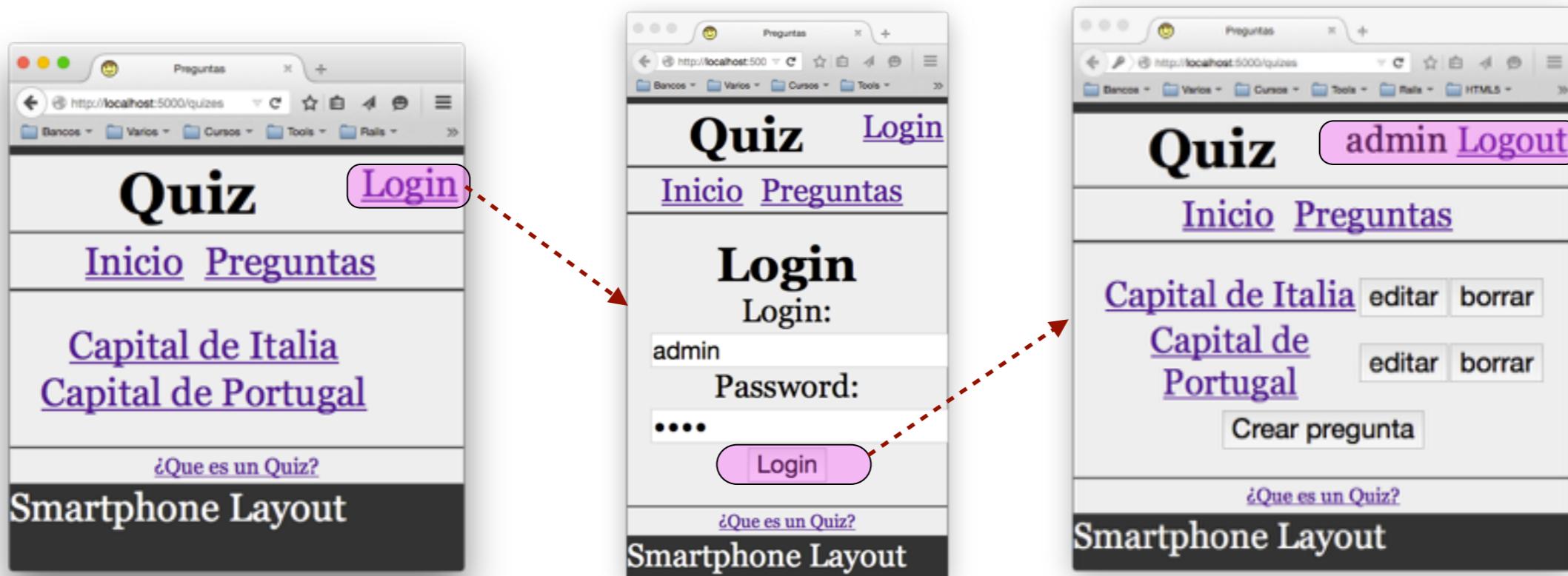
Quiz 17: Autorización

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

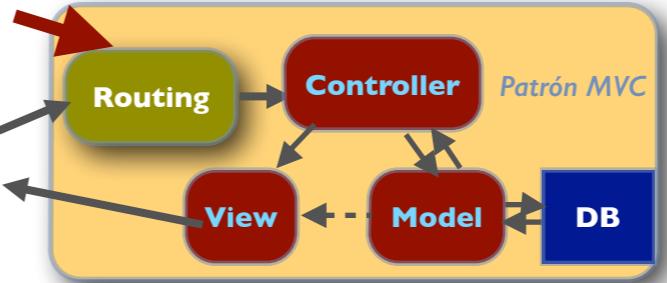
Quiz 17: Autorización

Objetivo: Añadir control de acceso a la aplicación Quiz para que solo los usuarios autenticados estén 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**



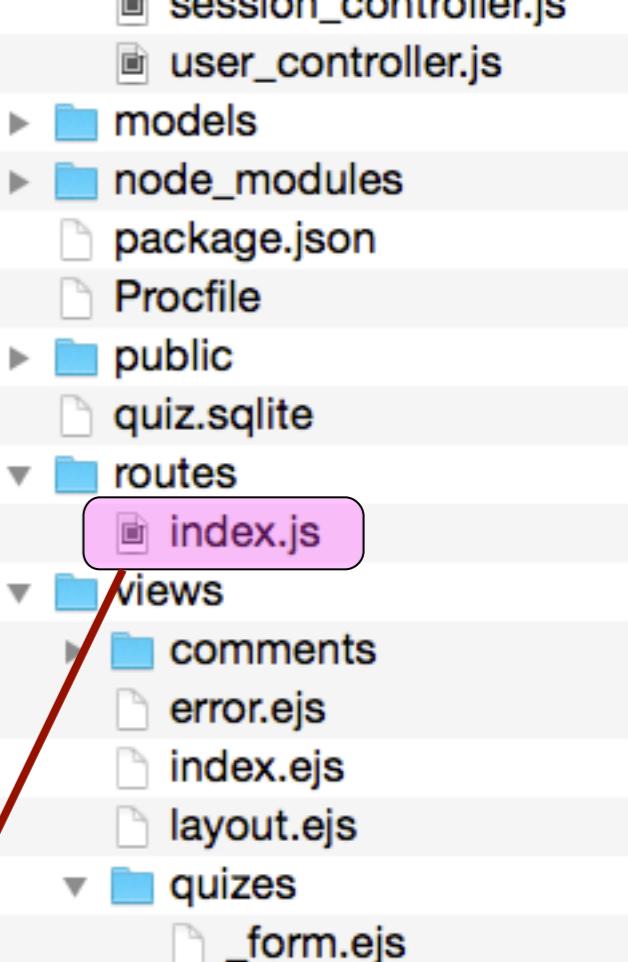
Paso 2



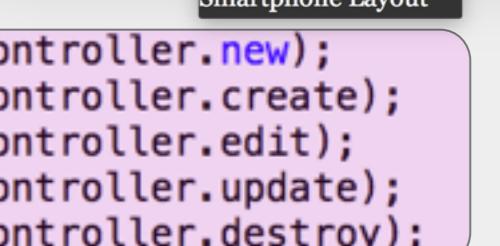
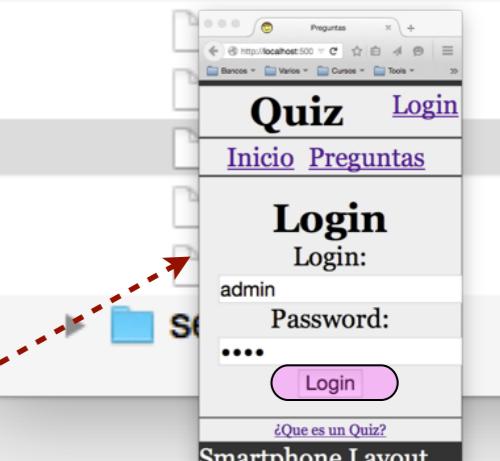
Una ruta puede invocarse con varios MWs en serie: “`get('/quizes/new', MW1, MW2)`”. **MW1** y **MW2** se ejecutan en serie, de forma que si **MW1** no pasa control a **MW2** con `next()`, **MW2** no llegará a ejecutarse.

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.



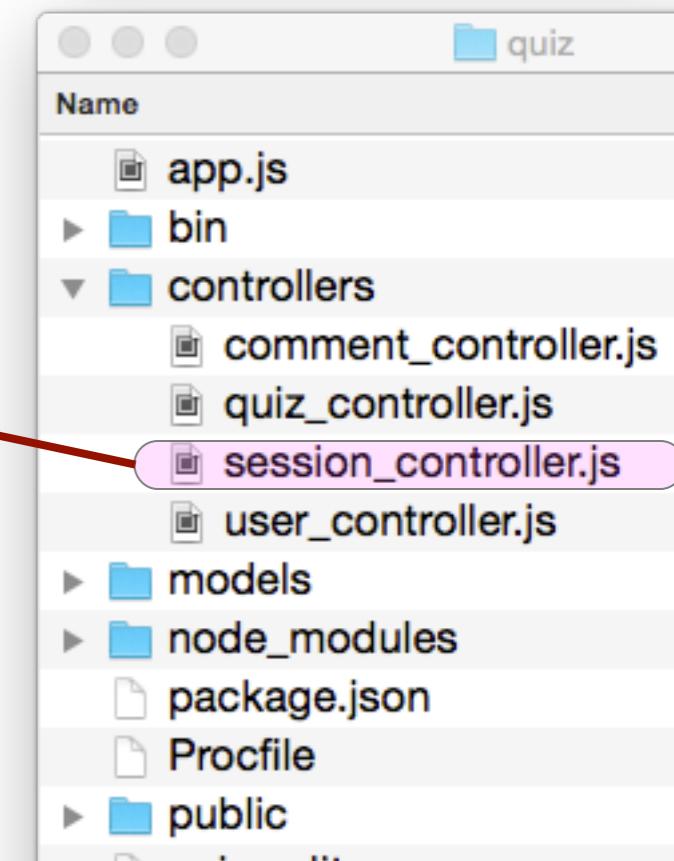
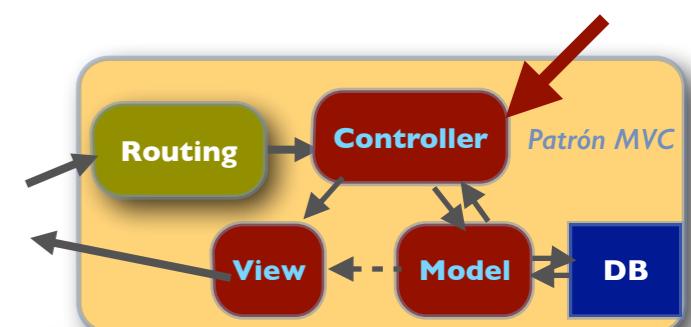
```
index.js
16 // Definición de rutas de sesión
17 router.get('/login', sessionController.new);      // formulario login
18 router.post('/login', sessionController.create);   // crear sesión
19 router.get('/logout', sessionController.destroy); // destruir sesión
20
21 // Definición de rutas de /quizes
22 router.get('/quizes', quizController.index);
23 router.get('/quizes/:quizId(\d+)', quizController.show);
24 router.get('/quizes/:quizId(\d+)/answer', quizController.answer);
25 router.get('/quizes/new', sessionController.loginRequired, quizController.new);
26 router.post('/quizes/create', sessionController.loginRequired, quizController.create);
27 router.get('/quizes/:quizId(\d+)/edit', sessionController.loginRequired, quizController.edit);
28 router.put('/quizes/:quizId(\d+)', sessionController.loginRequired, quizController.update);
29 router.delete('/quizes/:quizId(\d+)', sessionController.loginRequired, quizController.destroy);
30
31 // Definición de rutas de comentarios
32 router.get('/quizes/:quizId(\d+)/comments/new', commentController.new);
33 router.post('/quizes/:quizId(\d+)/comments', commentController.create);
```



Paso 1: MW de control

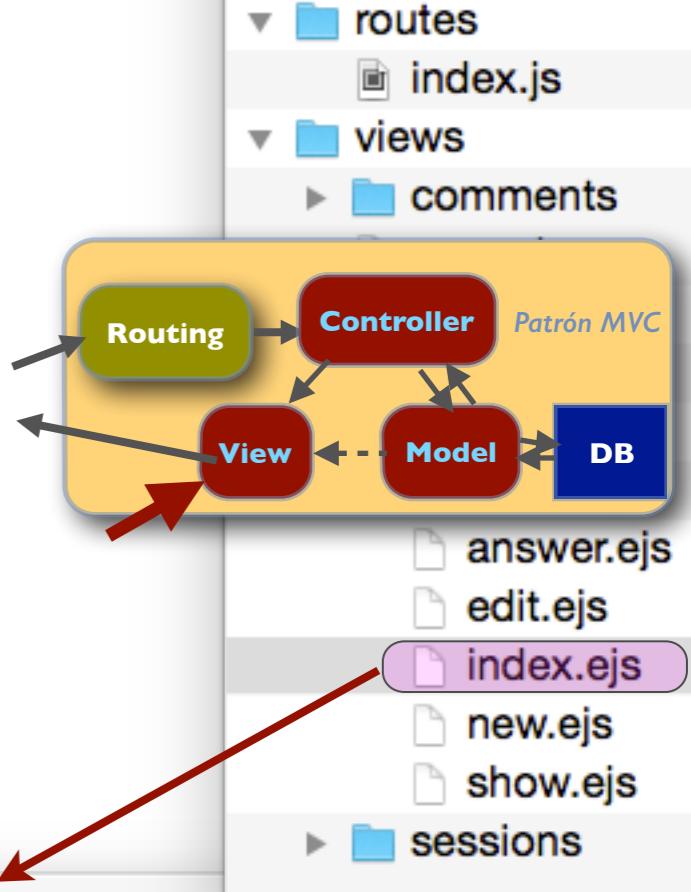
Para evitar la ejecución de los middlewares de atención a las operaciones que necesitan autenticación, se añade a **session_controller.js** un nuevo middleware que solo deja continuar al usuario si está autenticado. Son solo le da acceso a la pantalla de login.

```
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.

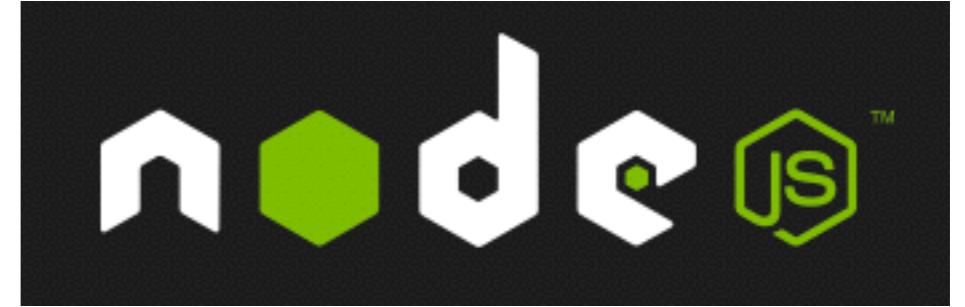


```
index.ejs
1 <table>
2   <% var i; for (i=0; i < quizzes.length; i++) { %>
3     <tr>
4       <td><a href="quizes/<%= quizzes[i].id %>"><%= quizzes[i].pregunta %></a></td>
5       <%if(session.user){%>
6         <td><a href="quizes/<%= quizzes[i].id %>/edit"><button>editar</button></a></td>
7         <td>
8           <form method="post" action="quizes/<%= quizzes[i].id %>?_method=delete">
9             <button type="submit" onClick="return confirm('Borrar: <%= quizzes[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 <%}%>
```





JavaScript



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

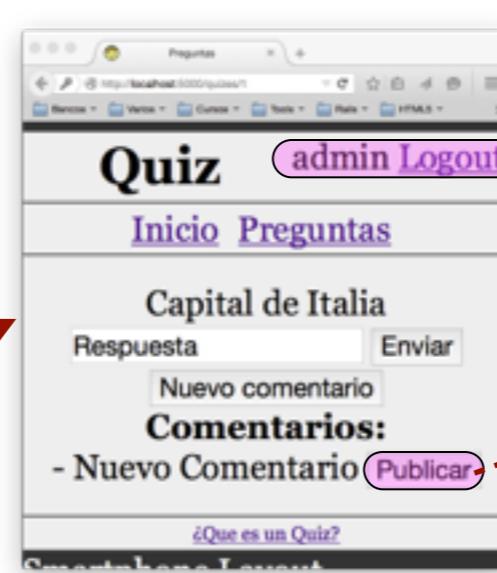
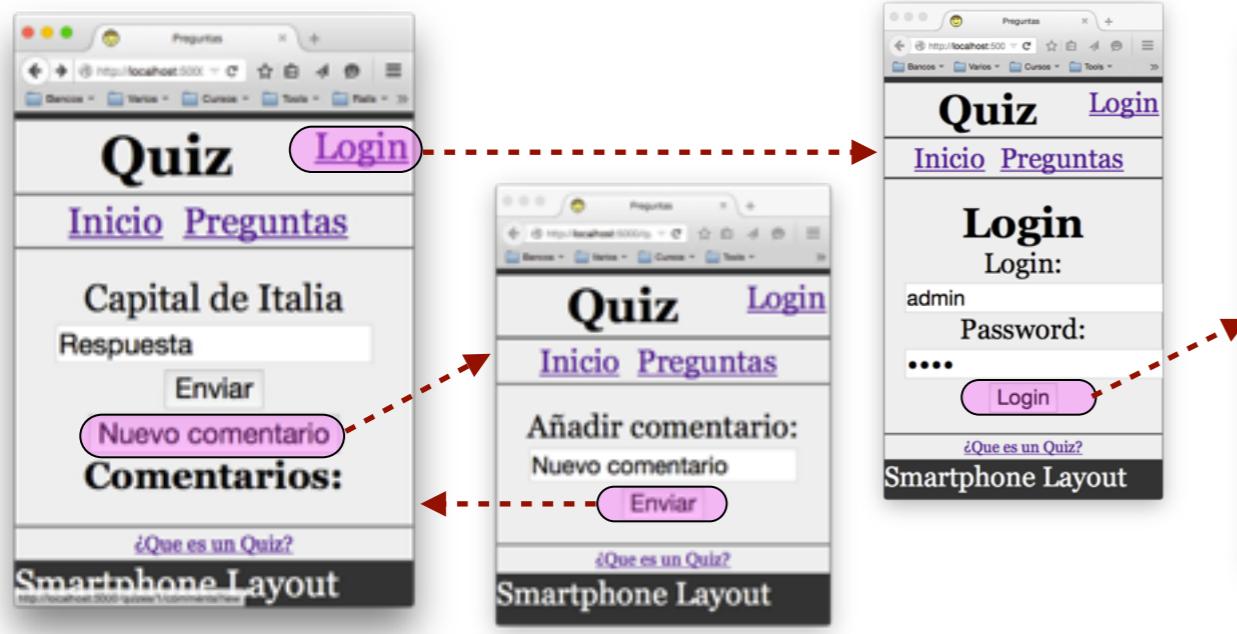
◆ **Paso 3:** Instalar middleware de autoload y nueva ruta en **routes/index.js**

- 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

- Se debe volver a reinicializar la base de datos al haberse modificado las tablas



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

A screenshot of a browser showing a table of comments. The first comment, "Nuevo comentario", has a value of "TRUE" in the "publicar" column, indicating it is published. The second comment, "Fue Sintra en el pasado", has a value of "FALSE" in the "publicar" column, indicating it is not published yet. The third comment, "Rome en ingles", also has a value of "TRUE". This table likely represents the database structure with an additional column for moderation status.

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.

```
// Definicion del modelo de Quiz con validación
module.exports = function(sequelize, DataTypes) {
  return sequelize.define(
    'Comment',
    { texto: {
        type: DataTypes.STRING,
        validate: { notEmpty: {msg: "-> Falta Comentario"}},
      },
      publicado: {
        type: DataTypes.BOOLEAN,
        defaultValue: false
      }
    }
  );
}
```

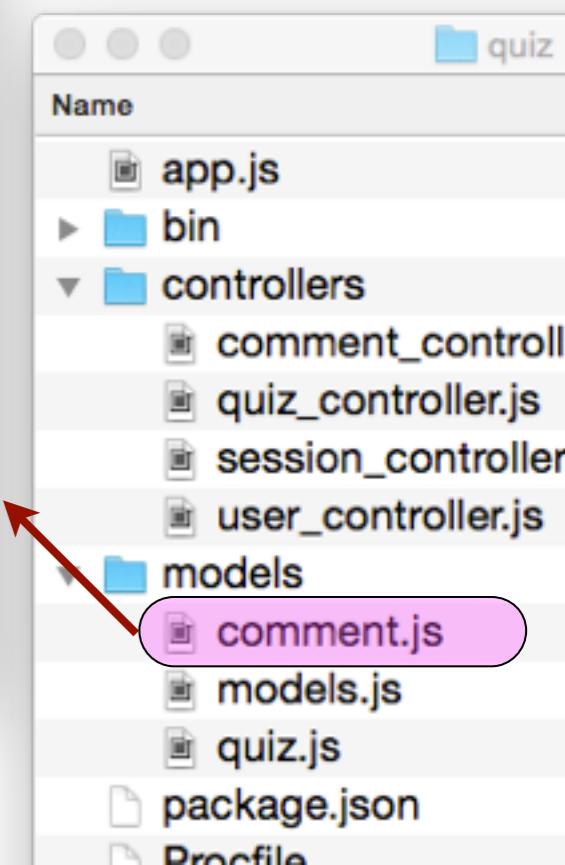
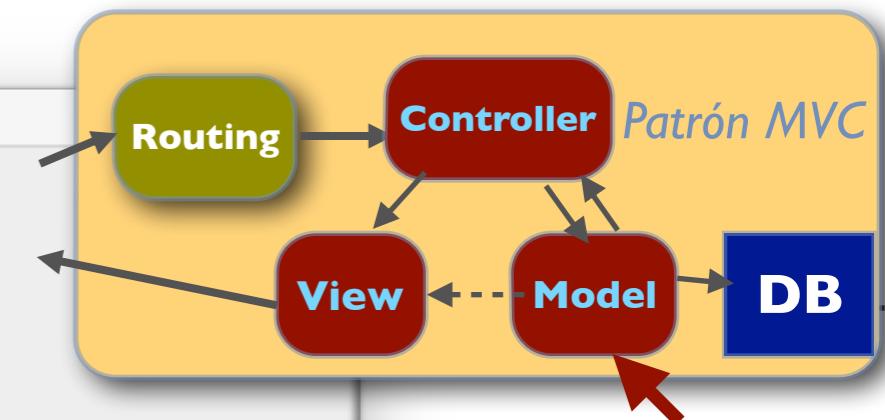
Tabla Comment

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

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

Tiene solo 1 campo de tipo string:

- **texto: DataTypes.STRING**

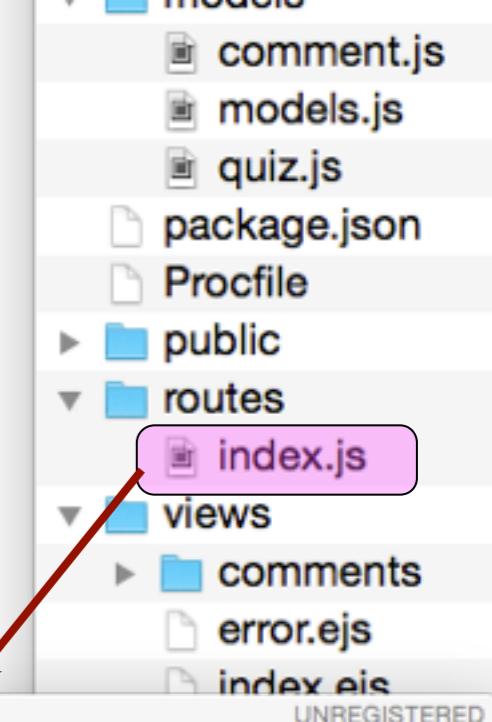


Pasos 3a y 3b

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

GET /quizes/:quizId/comments/:commentId/publish

-> GET funciona, pero el interfaz uniforme indica usar PUT en este caso



```
index.js
```

UNREGISTERED

```
13 // Autoload de comandos con :quizId
14 router.param('quizId', quizController.load); // autoload :quizId
15 router.param('commentId', commentController.load); // autoload :commentId Paso 3a
16
17 // Definición de rutas de sesión
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',
24   router.get('/quizes/:quizId(\d+)', quizController.index);
25   router.get('/quizes/:quizId(\d+)/answer', quizController.show);
26   router.get('/quizes/new', quizController.answer);
27   router.post('/quizes/create', sessionController.loginRequired, quizController.new);
28   router.get('/quizes/:quizId(\d+)/edit', sessionController.loginRequired, quizController.create);
29   router.put('/quizes/:quizId(\d+)', sessionController.loginRequired, quizController.edit);
30   router.delete('/quizes/:quizId(\d+)', sessionController.loginRequired, quizController.update);
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', sessionController.loginRequired, commentController.publish); Paso 3b
36
37
38 module.exports = router;
```

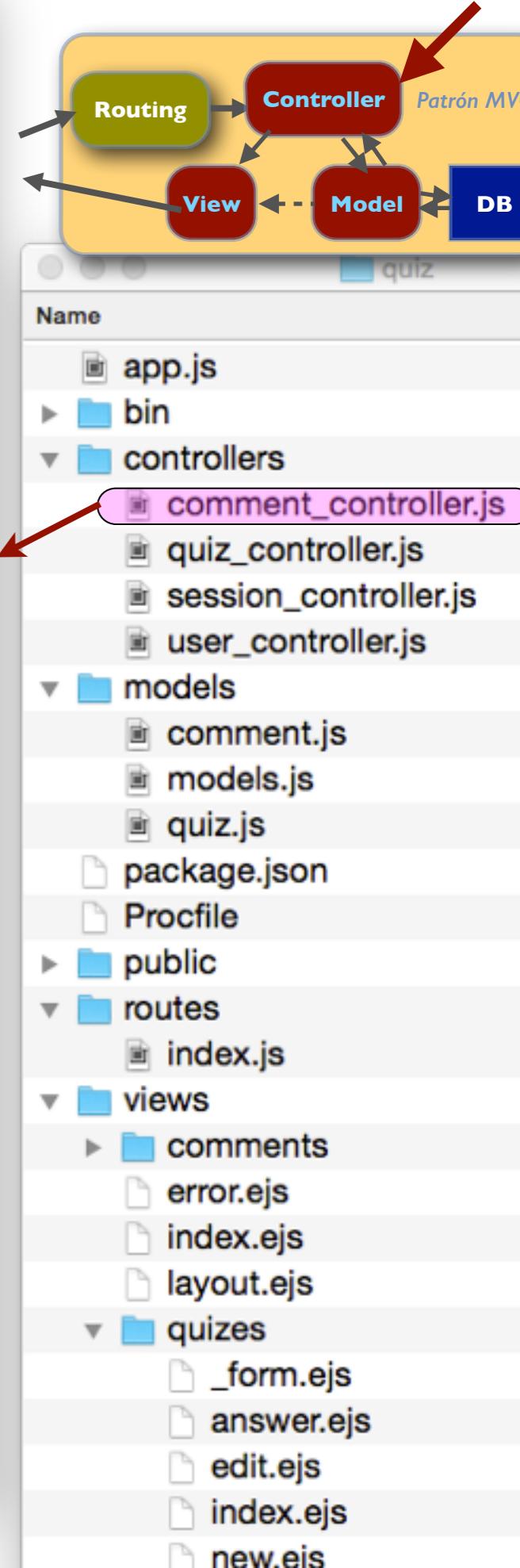
Patrón MVC

The diagram illustrates the MVC (Model-View-Controller) pattern. It features a central yellow rounded rectangle labeled 'Patrón MVC'. Inside, four components are arranged: 'Routing' (green oval at the top left), 'Controller' (red rounded rectangle at the top right), 'View' (red rounded rectangle at the bottom left), and 'Model' (blue rounded rectangle at the bottom right). Arrows indicate interactions: a solid arrow from 'Routing' to 'Controller', a dashed arrow from 'View' to 'Model', and a double-headed arrow between 'Controller' and 'Model'.

Pasos 2a y 2b

```
1 var models = require('../models/models.js');
2 // Autoload :id de comentarios
3 exports.load = function(req, res, next, commentId) {
4   models.Comment.find({
5     where: {
6       id: Number(commentId)
7     }
8   }).then(function(comment) {
9     if (comment) {
10       req.comment = comment;
11       next();
12     } else{next(new Error('No existe commentId=' + commentId))}
13   }
14   .catch(function(error){next(error)});
15 };
16
17 // GET /quizzes/:quizId/comments/new
18 exports.new = function(req, res) {
19   res.render('comments/new.ejs', {quizid: req.params.quizId, errors: []});
20 };
21
22 // POST /quizzes/:quizId/comments
23 exports.create = function(req, res) {
24   .....
25 };
26
27 // GET /quizzes/:quizId/comments/:commentId/publish
28 exports.publish = function(req, res) {
29   req.comment.publicado = true;
30
31   req.comment.save( {fields: ["publicado"]})
32     .then( function(){ res.redirect('/quizzes/'+req.params.quizId);} )
33     .catch(function(error){next(error)});
34 };
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54 }
```

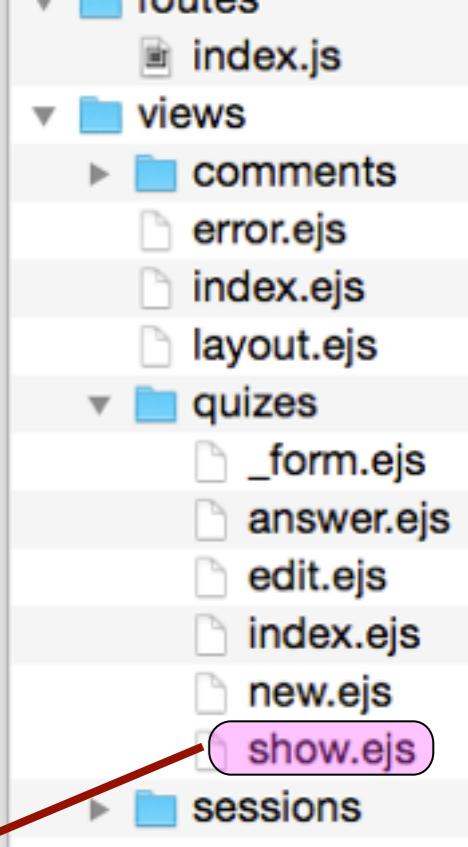
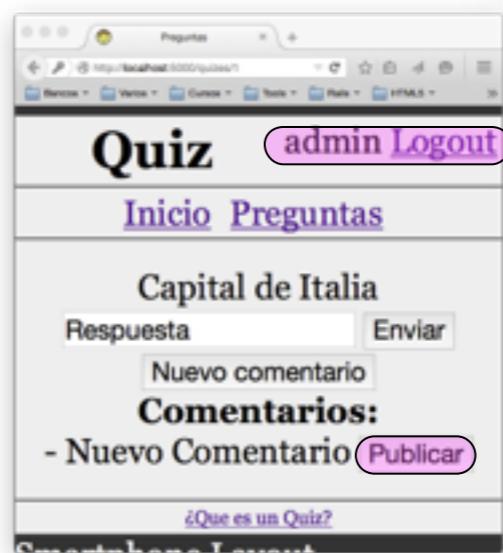
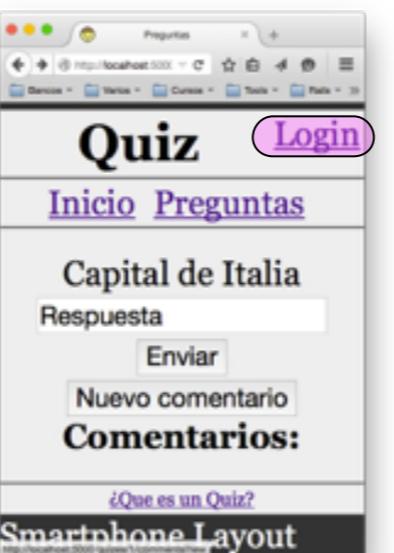
La ruta de la acción
publish lleva :commentId
y necesita autoload



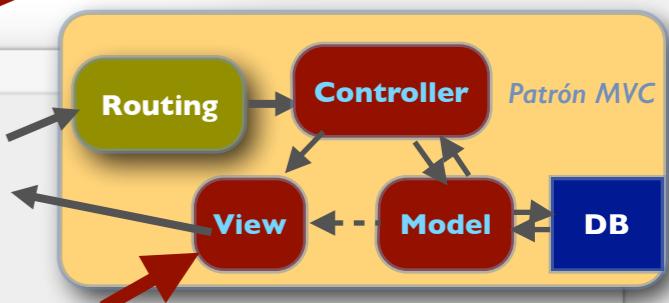
Paso 4

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.

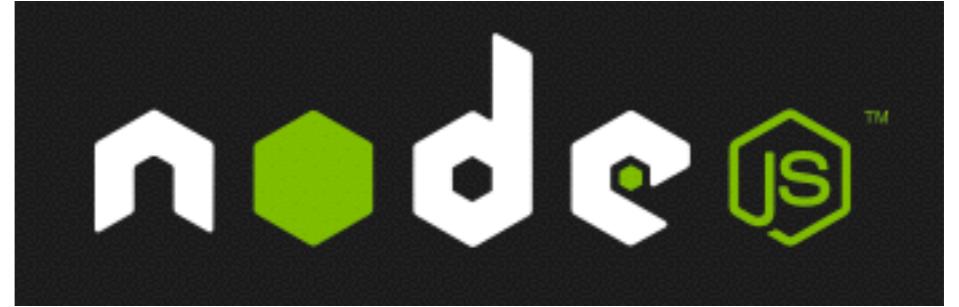


```
1 <form method="get" action="/quizes/<%= quiz.id %>/answer">
2   <%= quiz.pregunta %> <p>
3     <input type="text" name="respuesta" value="Respuesta"/>
4     <input type="submit" value="Enviar">
5   </form>
6
7   <p><a href="/quizes/<%= quiz.id %>/comments/new"><button>Nuevo comentario</button></a></p>
8
9   <p><strong>Comentarios:</strong></p>
10  <%for(index in quiz.Comments){%>
11    <%if(quiz.Comments[index].publicado || session.user){%>
12      <p>- <%=quiz.Comments[index].texto%>
13      <%if(session.user && !quiz.Comments[index].publicado){%>
14        <a href="/quizes/<%= quiz.id %>/comments/<%=quiz.Comments[index].id%>/publish">
15          <button>Publicar</button></a>
16        <%}%>
17      </p>
18    <%}%>
19  <%}%>
```





JavaScript



Quiz 19: HTTPS - HTTP Seguro

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

Quiz 19: HTTPS - HTTP Seguro

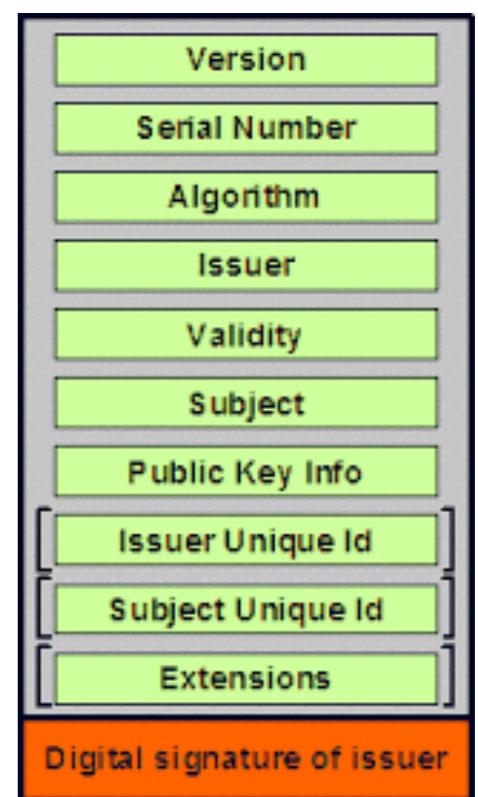
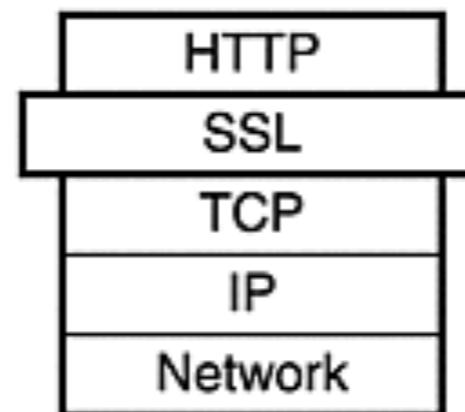
Objetivo: Añadir acceso HTTP Seguro a la aplicación Quiz para evitar que las contraseñas de acceso se envíen en claro y añadir así mayor seguridad al uso del portal.

◆ **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

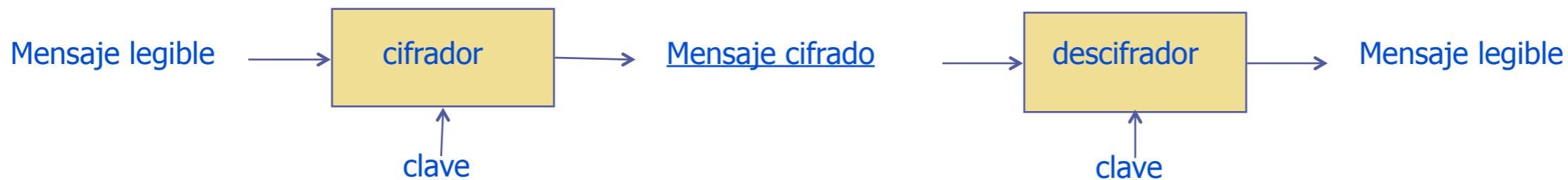
◆ **Paso 3:** Guardar **versión (commit)** git y subir a **Heroku**



Clave pública y clave privada

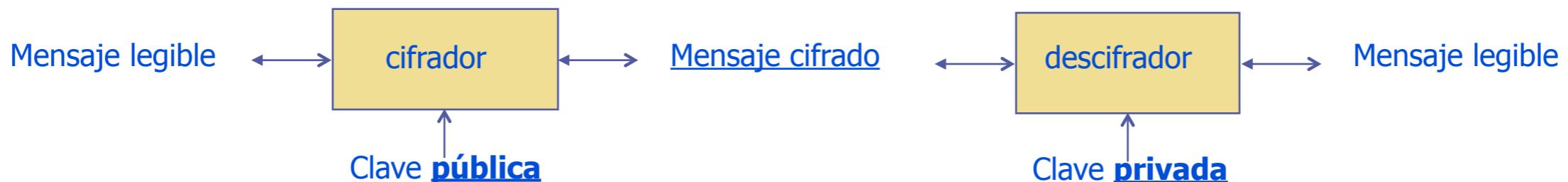
◆ Clave simétrica

- Emisor y receptor necesitan clave para cifrar y descifrar



◆ Clave asimétrica (pública + 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 (ACs) 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

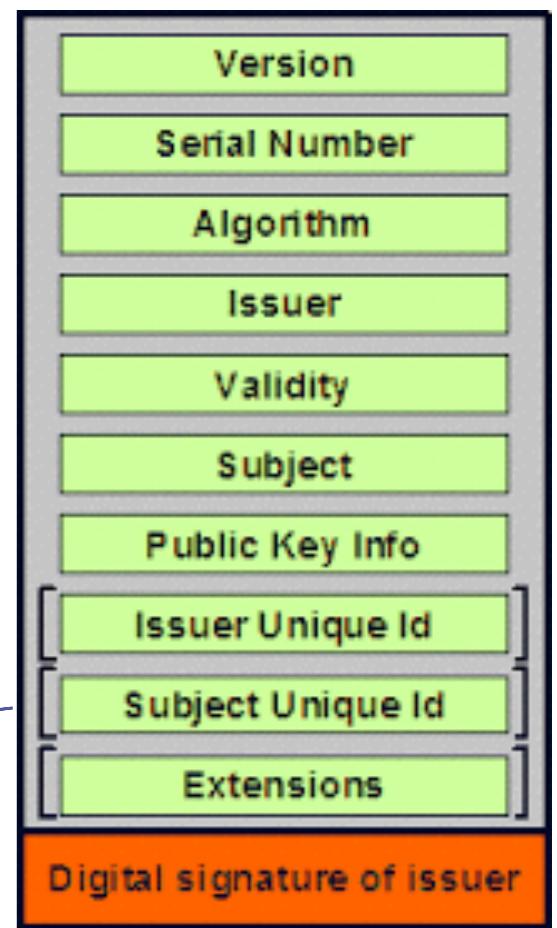
◆ Certificado X509: para cifrar o autenticar → quiz-2015-cert.pem

- Certificado con clave pública emitido por AC o “auto-firmado”

◆ En Quiz utilizamos un certificado “auto-firmado” de 1 año de validez

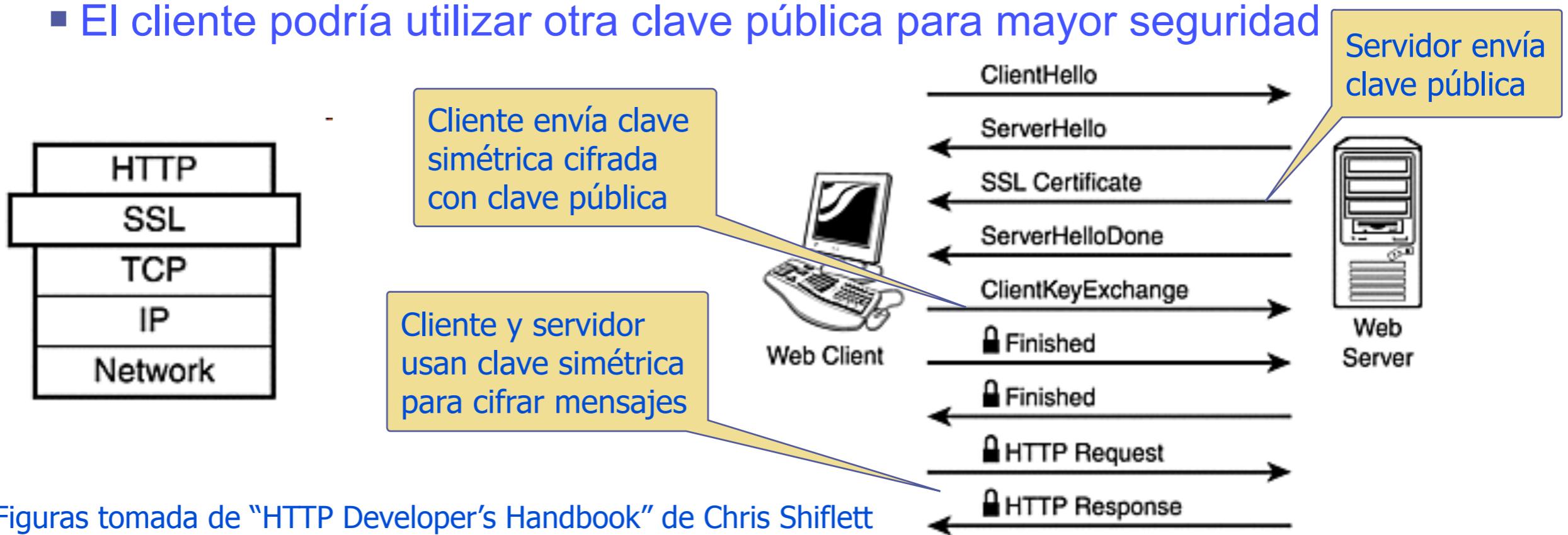
- Los navegadores lo detectarán y notificarán al usuario

Estructura de clave pública X509



HTTPS: Conexiones seguras

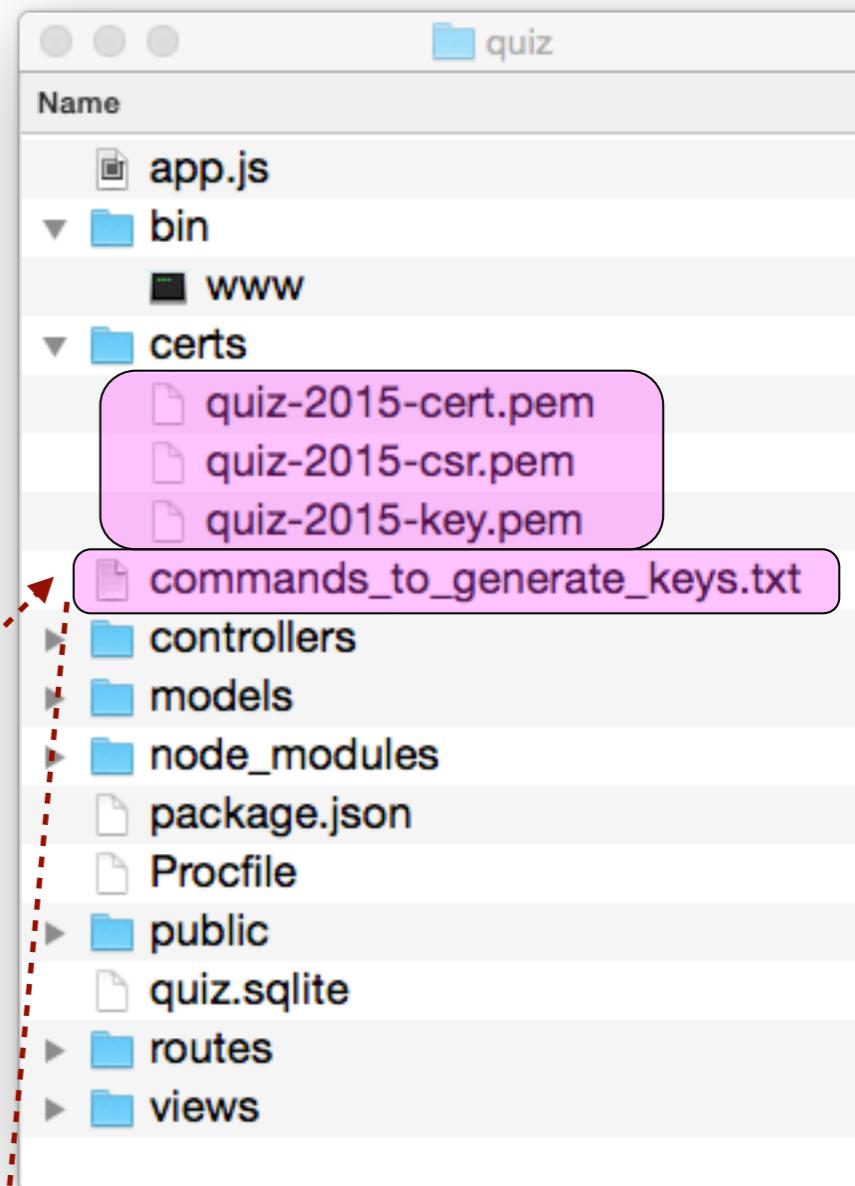
- ◆ HTTPS introduce el nivel SSL de cifrado entre HTTP y Sockets
 - Este nivel cifra la información que intercambian cliente y servidor
 - ◆ HTTPS se activa automáticamente si el URL lo indica: [https://dit.upm.es/.....](https://dit.upm.es/)
- ◆ HTTPS se suele utilizar solo con la clave pública del servidor
 - 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 tomadas 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**



```
1 mkdir certs
2 cd certs
3 openssl genrsa -out quiz-2015-key.pem 2048
4 openssl req -new -sha256 -key quiz-2015-key.pem -out quiz-2015-csr.pem
5 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** permite 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.

The terminal window on the left displays the code for the 'www' file. A red dashed arrow points from the 'www' file in the terminal to the 'www' folder in the file browser on the right. The file browser shows the directory structure of the 'quiz' project, including 'app.js', 'bin', 'certs' (containing 'quiz-2015-cert.pem', 'quiz-2015-csr.pem', and 'quiz-2015-key.pem'), 'controllers', 'models', 'node_modules', 'package.json', 'Procfile', 'public', 'quiz.sqlite', 'routes', and 'views'.

```
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 });
```



Final del tema