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**

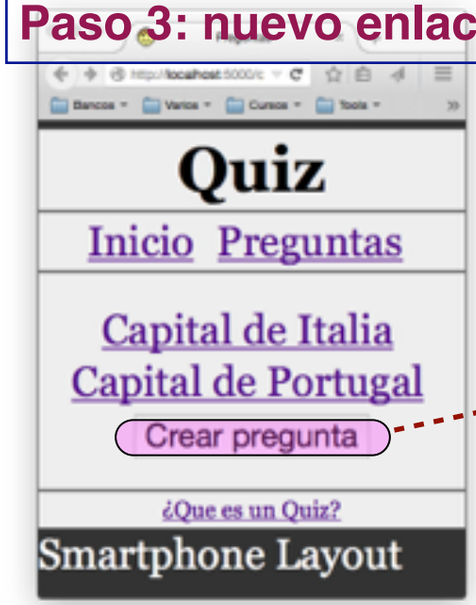
✦**Paso 3**: Cambiar config. **bodyParser.urlencoded()** en **app.js**

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

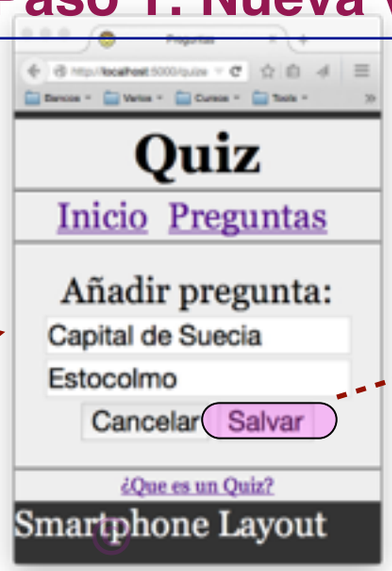
✦**Paso 5**: Guardamos **versión (commit)** git y subir a **Heroku**

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

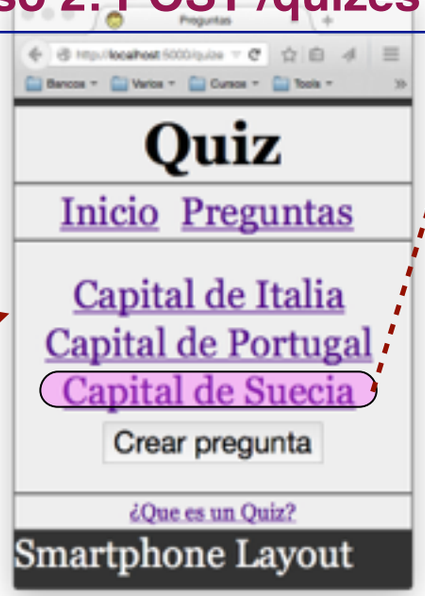
Paso 3: nuevo enlace



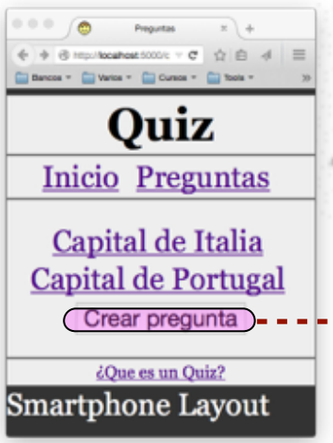
Paso 1: Nueva vista



Paso 2: POST /quizes

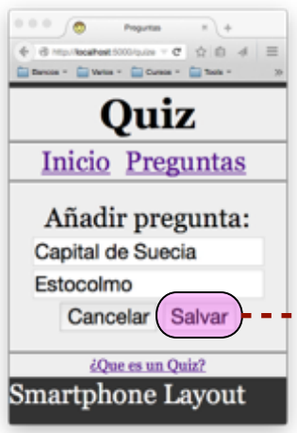


HTTP: Creación de nuevo quiz



GET /quizes/new

200 OK



POST /quizes/create

302 moved temporarily

Redirección automática

GET /quizes

200 OK

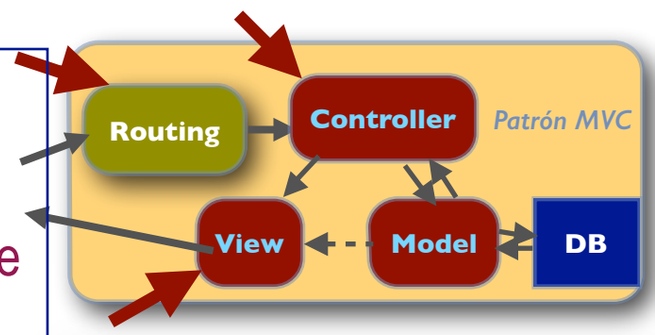
POST /quizes/create HTTP/1.1
Host: localhost
Accept: text/*, image/*
.....
Content-type: application/x-www-form-urlencoded
Content-length: 17
quiz[pregunta]=Capital+de+Suecia&quiz[respuesta]=Estocolmo

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.

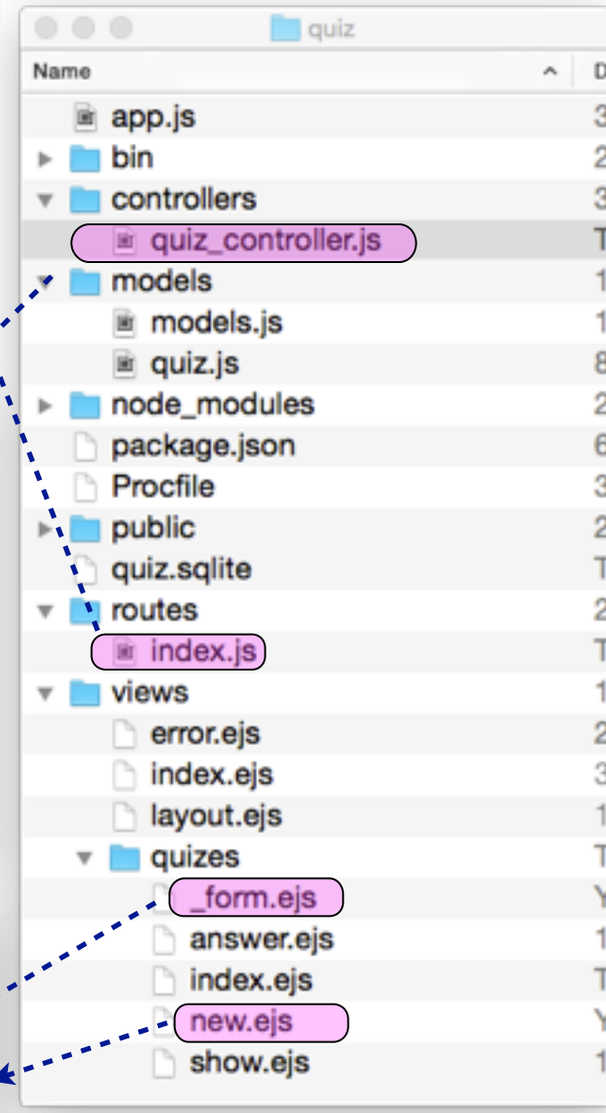
El método **build(..)** de sequelize (http://sequelize.readthedocs.org/en/latest/docs/instances/) creo un objeto no persistente asociado a la tabla Quiz, con las propiedades inicializadas. Este objeto se utiliza aquí solo para renderizar las vistas.

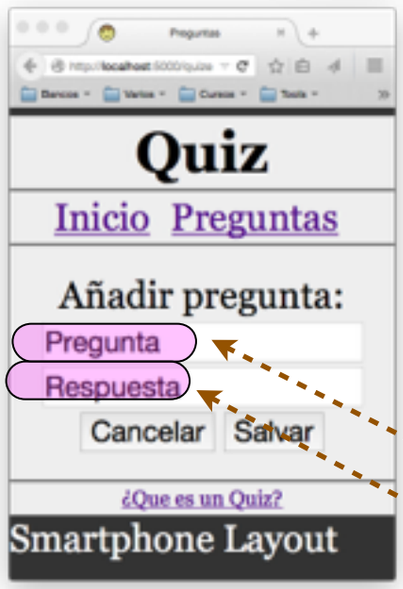


Paso 1

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);
```





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

```
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);
```

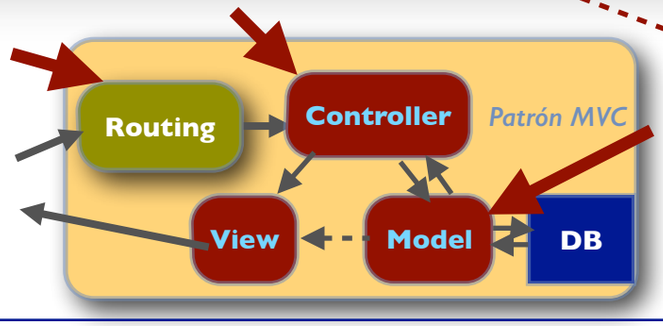
Paso 2a: controlador de **create**

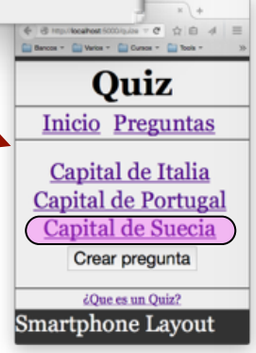
quiz_controller.js — UNREGISTERED

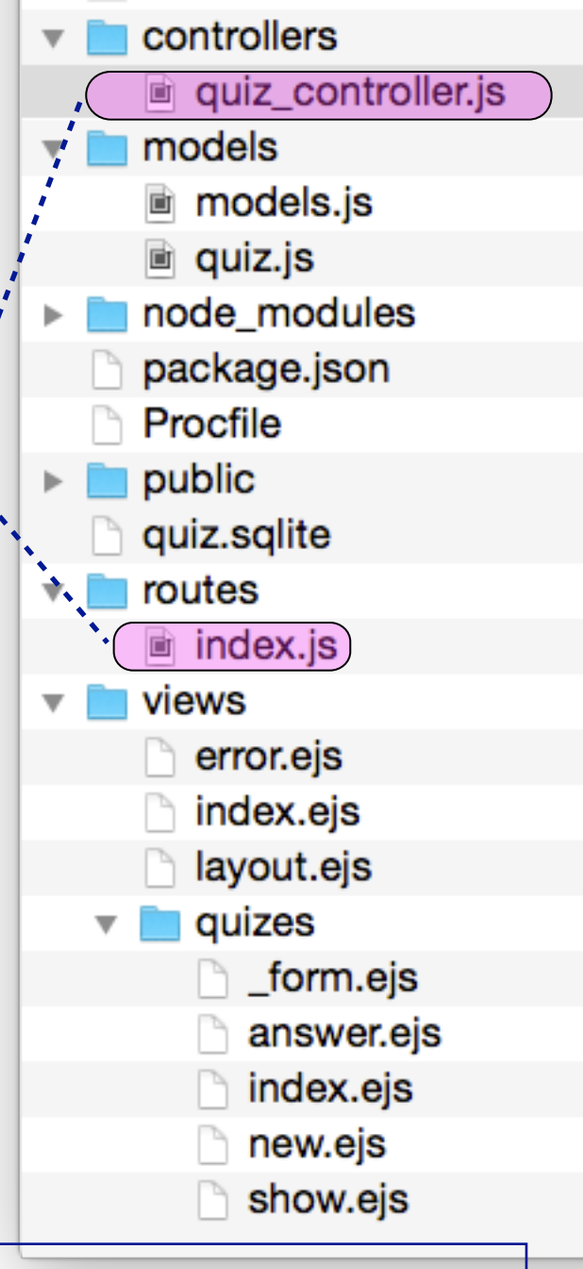
```
// POST /quizes/create
exports.create = function(req, res) {
  var quiz = models.Quiz.build( req.body.quiz );

// 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

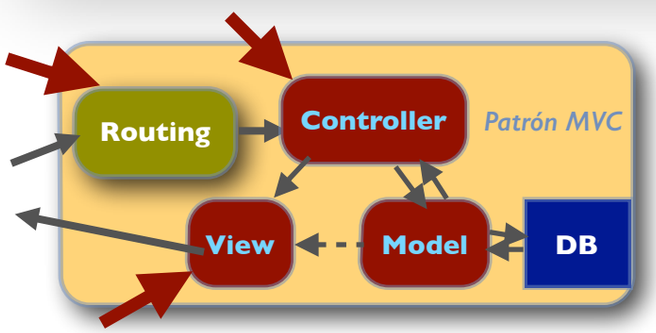
- app.js
- bin
- controllers
  - quiz_controller.js
- models
  - models.js
  - quiz.js
- node_modules
- package.json
- Procfile
- public
- quiz.sqlite
- routes
  - index.js
- views
  - error.ejs
  - index.ejs
  - layout.ejs
  - quizes
    - _form.ejs
    - answer.ejs
    - index.ejs
    - new.ejs
  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

Los nombres de los parámetros del formulario

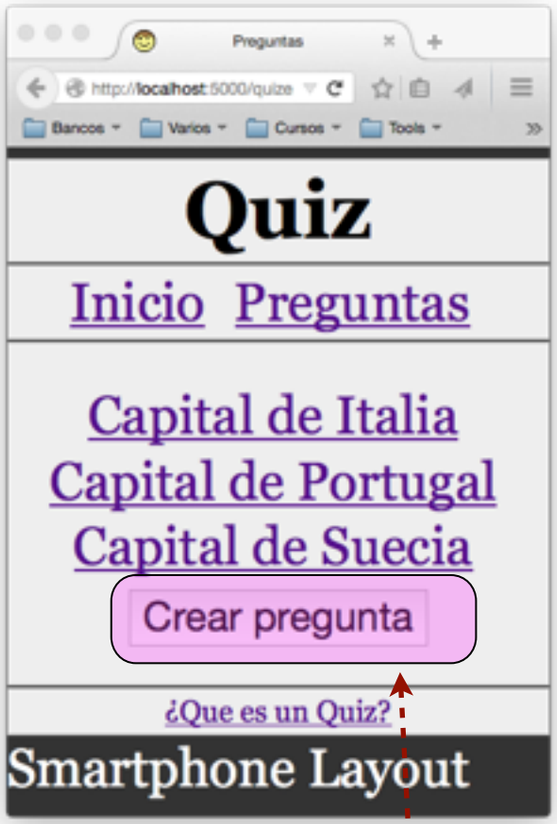
- name="quiz[pregunta]"
- name="quiz[respuesta]"

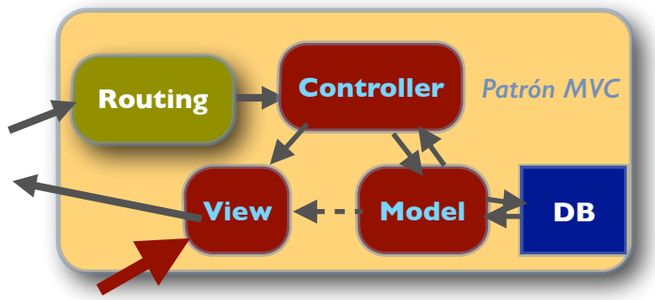
utilizan notación pseudo JSON que permite indicar que son propiedades de un objeto **quiz**. El middleware **bodyparser.urlencoded(..)** los analiza correctamente y genera el objeto **req.body.quiz**, si quitamos el parámetro de configuración {extended: false} que express-generator incluyo cuando generó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

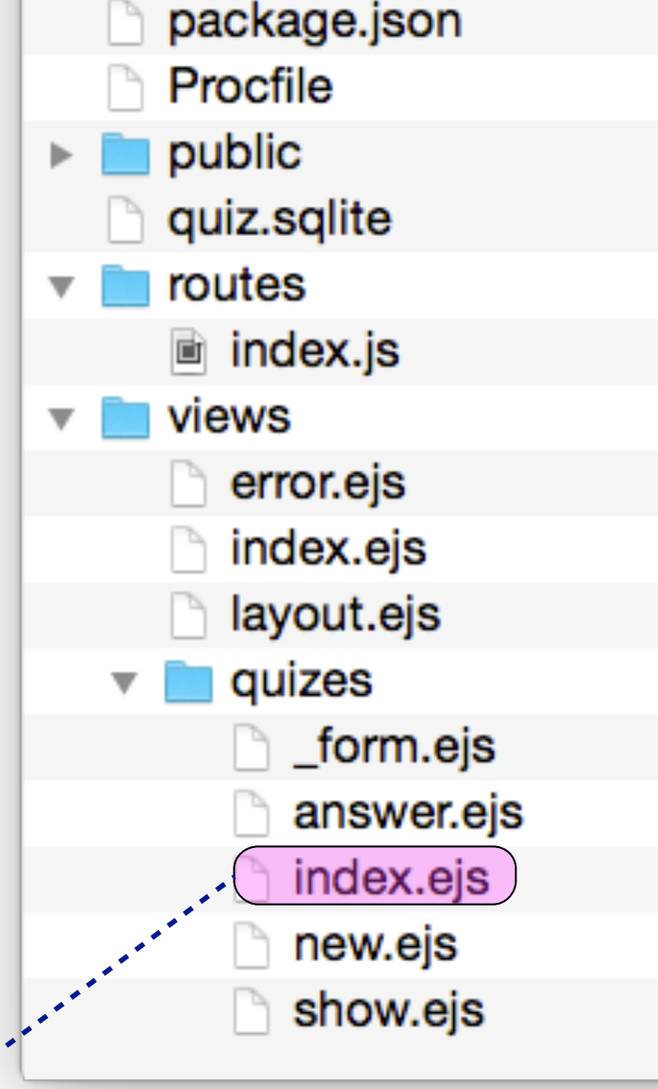
Mas info: https://github.com/expressjs/body-parser/blob/master/README.md



Paso 4: enlace a crear pregunta







views/quizes/index.js

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>
```

Paso 4

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 }
```

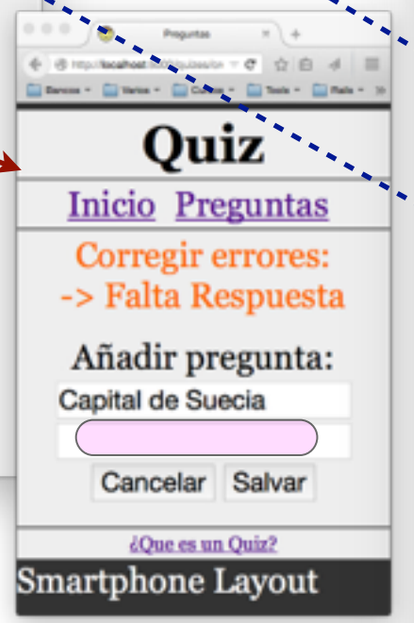
Paso 1: models/quiz.js

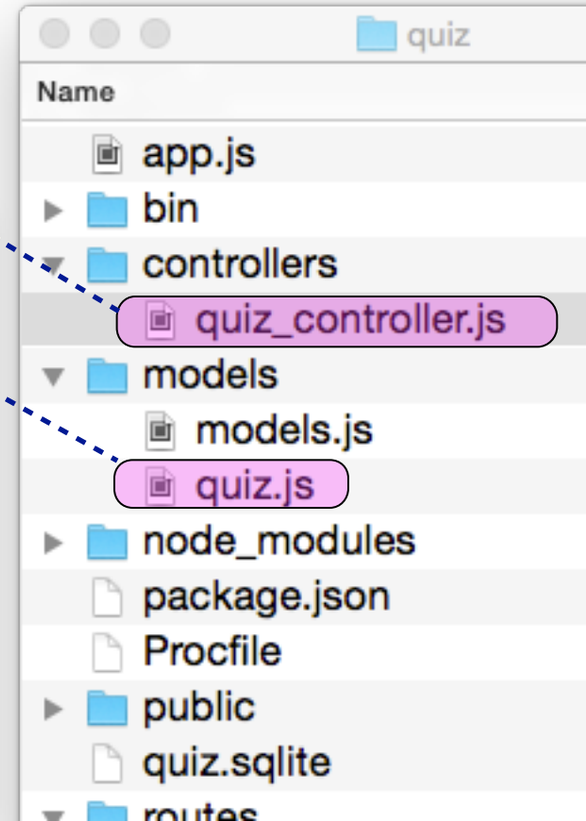
Sequelize.js permite añadir al modelo de una tabla restricciones en al contenido de los campos, de forma que se pueda comprobar con la **función validate()** de sequelize, si un objeto JavaScript de la clase asociada a los elementos de la tabla cumple las restricciones o no.

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)

Validaciones soportados en sequelize.js

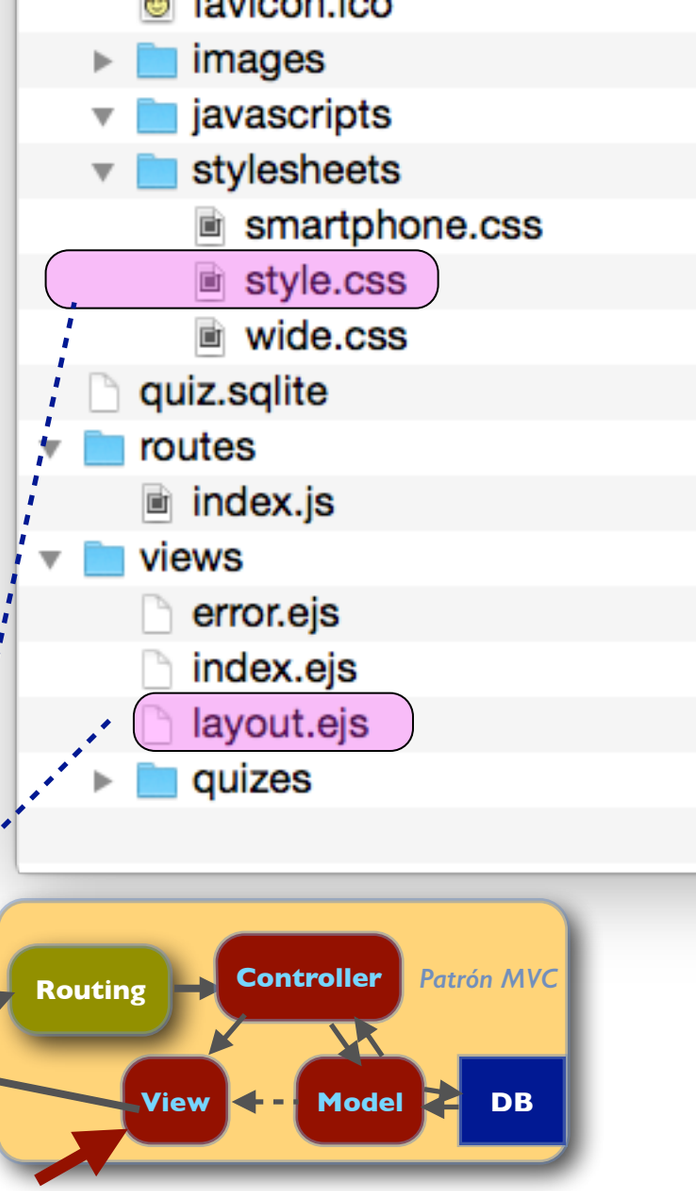
```
var ValidateMe = sequelize.define('Foo', {
  foo: {
    type: Sequelize.STRING,
    validate: {
      is: ["^[a-z]+$",'i'],      // will only allow letters
      is: /^[a-z]+$/i,           // same as the previous example using real RegExp
      not: ["[a-z]",'i'],        // will not allow letters
      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
```

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

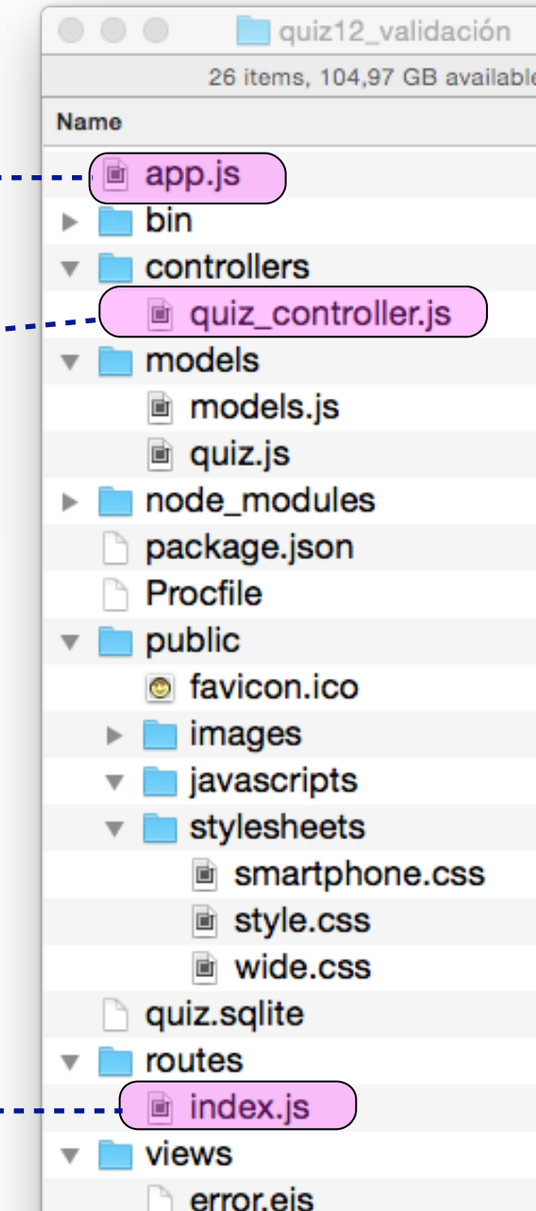
```
var express = require('express');
var router = express.Router();

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.

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

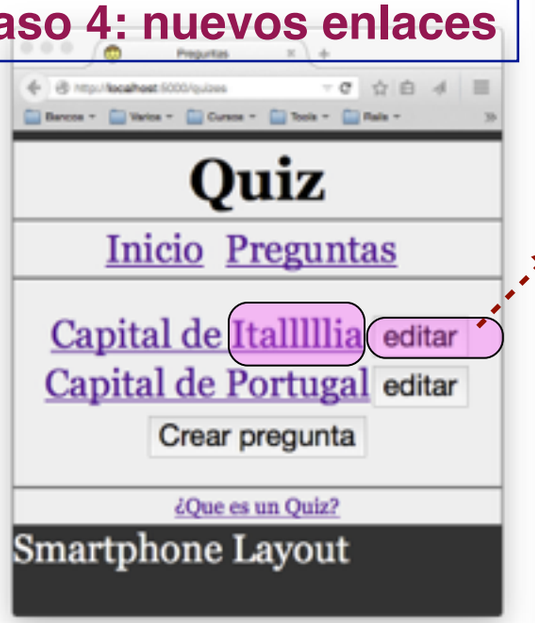
- a: Añadir acción **update** a **quiz_controller.js** asociado a **/quizes/:quizId**
- b: Añadir ruta **POST /quizes/update** en **routes/index.js**

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

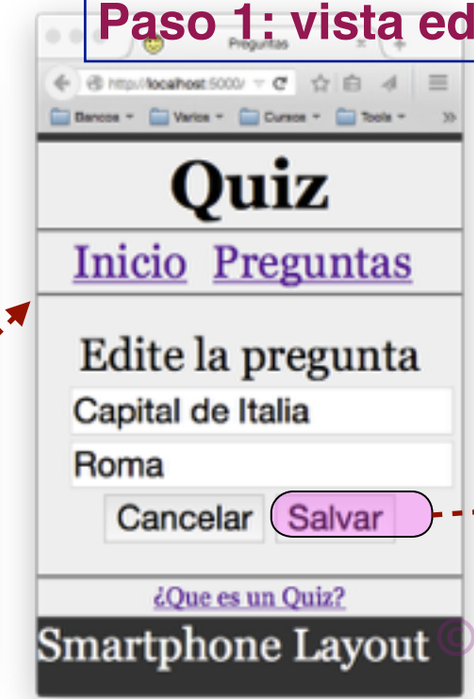
✦**Paso 5**: Guardar **versión (commit)** git y subir a **Heroku**

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

Paso 4: nuevos enlaces

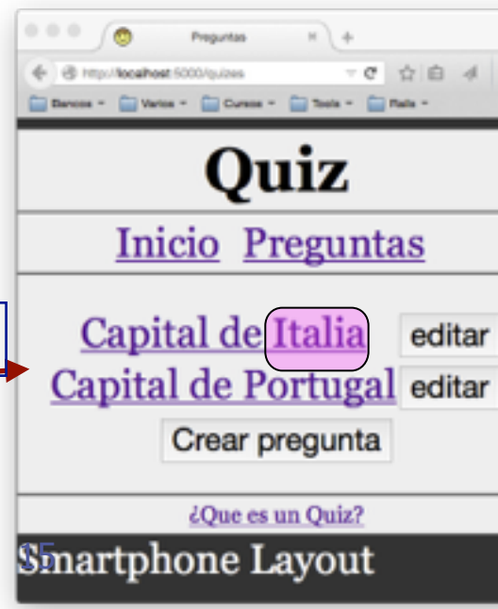


Paso 1: vista edit.ejs

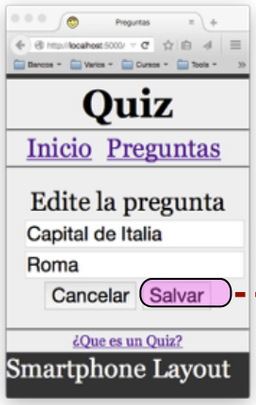


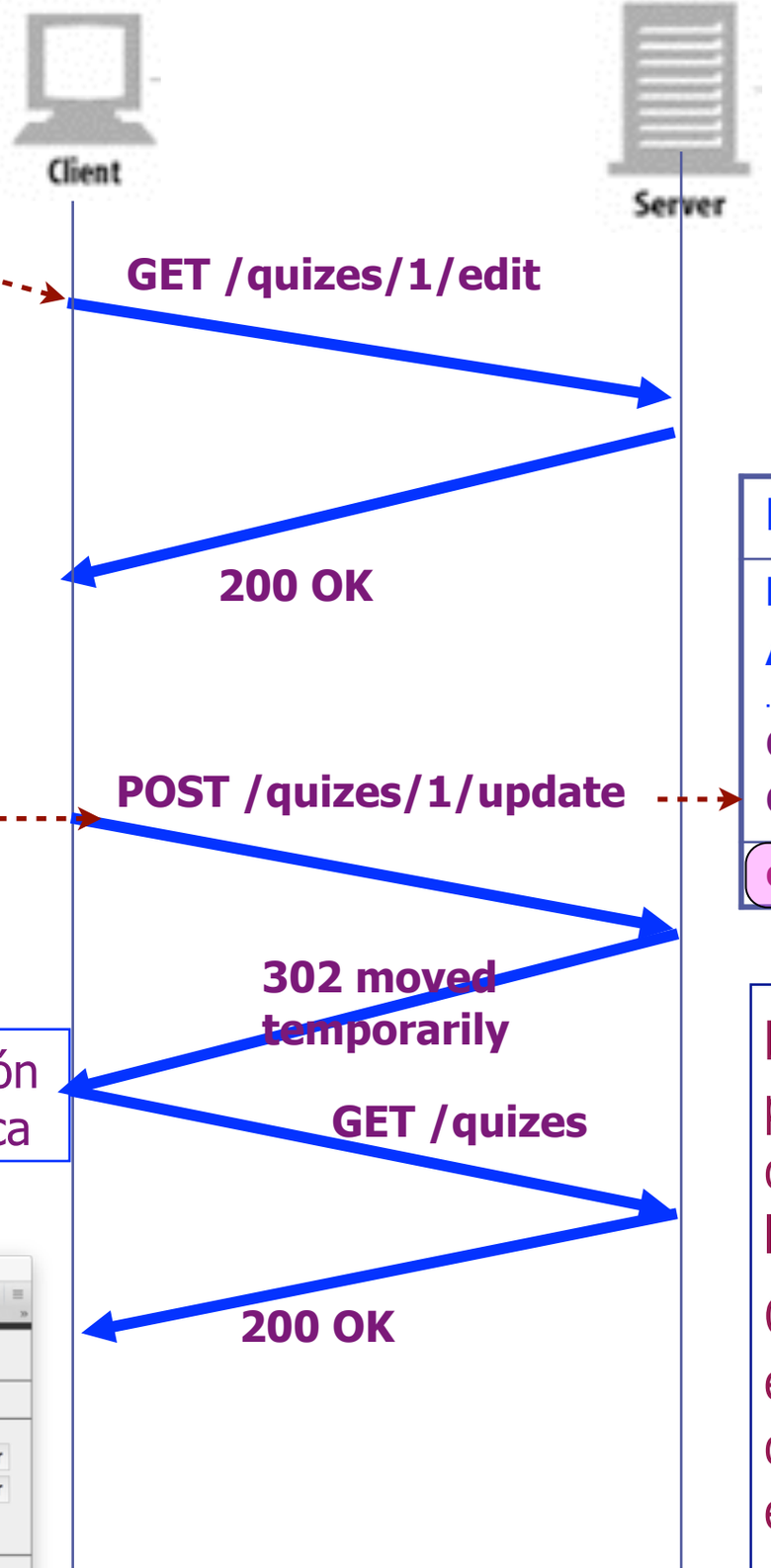
Paso 3: PUT /quizes/1/update

Paso 2: POST /quizes/1/update?_method=put



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
```

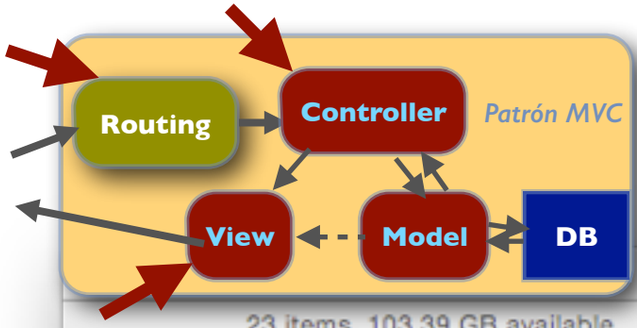
La primitiva **HTTP POST** para actualizar la DB con la pregunta corregida se invoca, igual que en la creación, con un formulario HTML. El formulario permite re-editar las preguntas e invocar HTTP POST para actualizar la DB.

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

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

index.js

```
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);
```

Paso 1a: controlador **edit**

quiz_controller.js

```
72  // GET /quizes/:id/edit
73  exports.edit = function(req, res) {
74    var quiz = req.quiz; // autoload de instancia de quiz
75
76    res.render('quizes/edit', {quiz: quiz, errors: []});
77  };
```


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

edit.ejs

```
1  Edite la pregunta <p>
2
3  <form method="post" action="/quizes/<%= quiz.id %>?_method=put" >
4    <% include _form.ejs %>
5  </form>
```

_form.ejs

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

Paso 1

Paso 2: method override

app.js

```
1  var express = require('express');
2  var path = require('path');
3  var favicon = require('serve-favicon');
4  var logger = require('morgan');
5  var cookieParser = require('cookie-parser');
6  var bodyParser = require('body-parser');
7  var partials = require('express-partials');
8  var methodOverride = require('method-override');
9
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
18 app.use(partials());
19 // uncomment after placing your favicon in /public
20 // app.use(favicon(__dirname + '/public/favicon.ico'));
21 app.use(favicon(__dirname + '/public/favicon.ico'));
22 app.use(logger('dev'));
23 app.use(bodyParser.json());
24 app.use(bodyParser.urlencoded());
25 app.use(cookieParser());
26 app.use(methodOverride('_method'));
27 app.use(express.static(path.join(__dirname, 'public')));
28
29 app.use('/', routes);
30 .........
```

```
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     "method-override": "^2.3.1",
23     "morgan": "~1.3.0",
24     "pg": "^4.1.1",
25     "sequelize": "^2.0.0-rc4",
26     "serve-favicon": "~2.1.3"
27   }
28 }
```

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

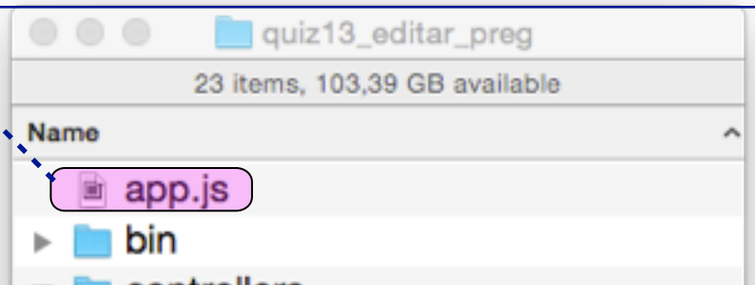
$ npm install --save method-override@2.3.1

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

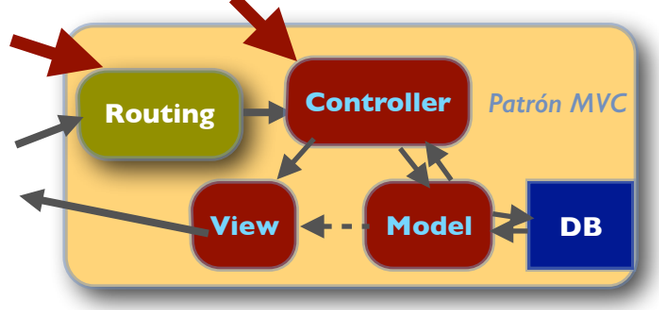
Importamos en app.js el paquete con **require('method-override'),** instalando el midlleware en nuestra aplicación con **app.use(methodOverride('_method'))**.

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	.....	.....

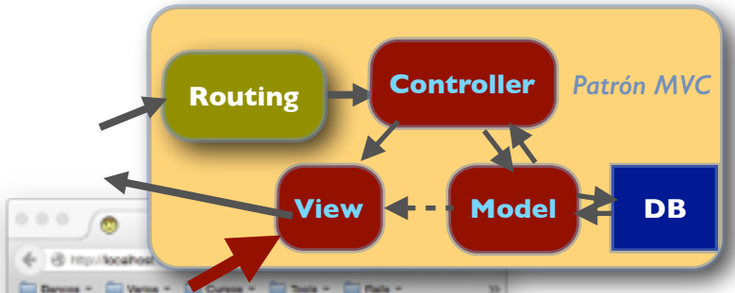
quiz13_editar_preg

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



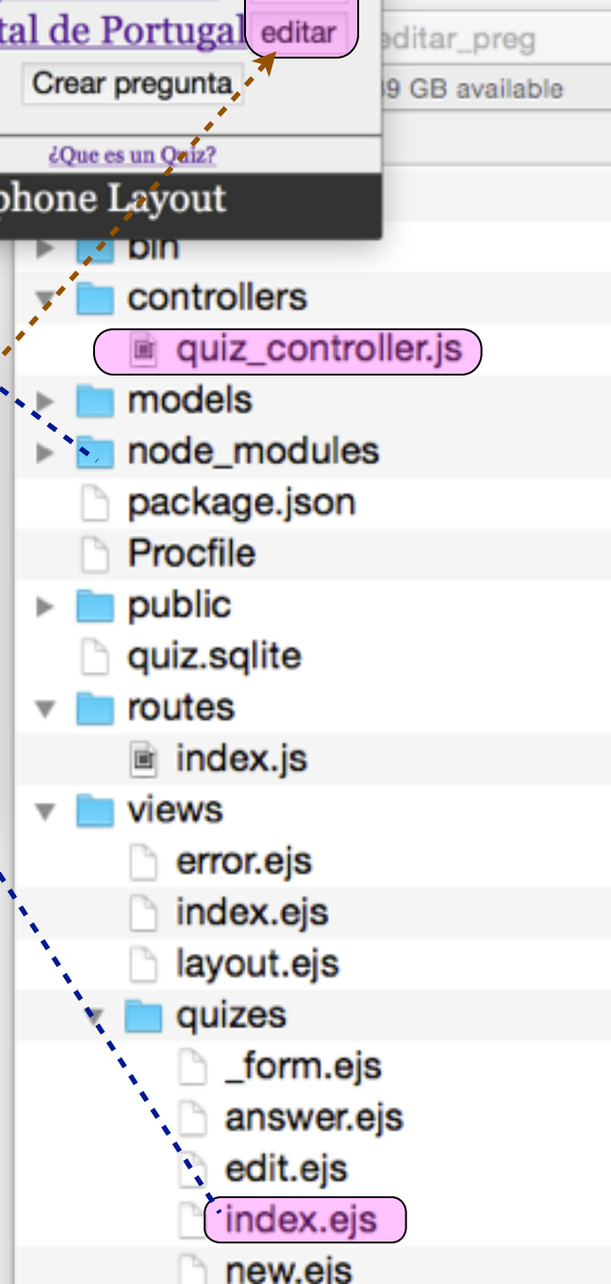
quiz_controller.js — UNREGISTERED

```
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.

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

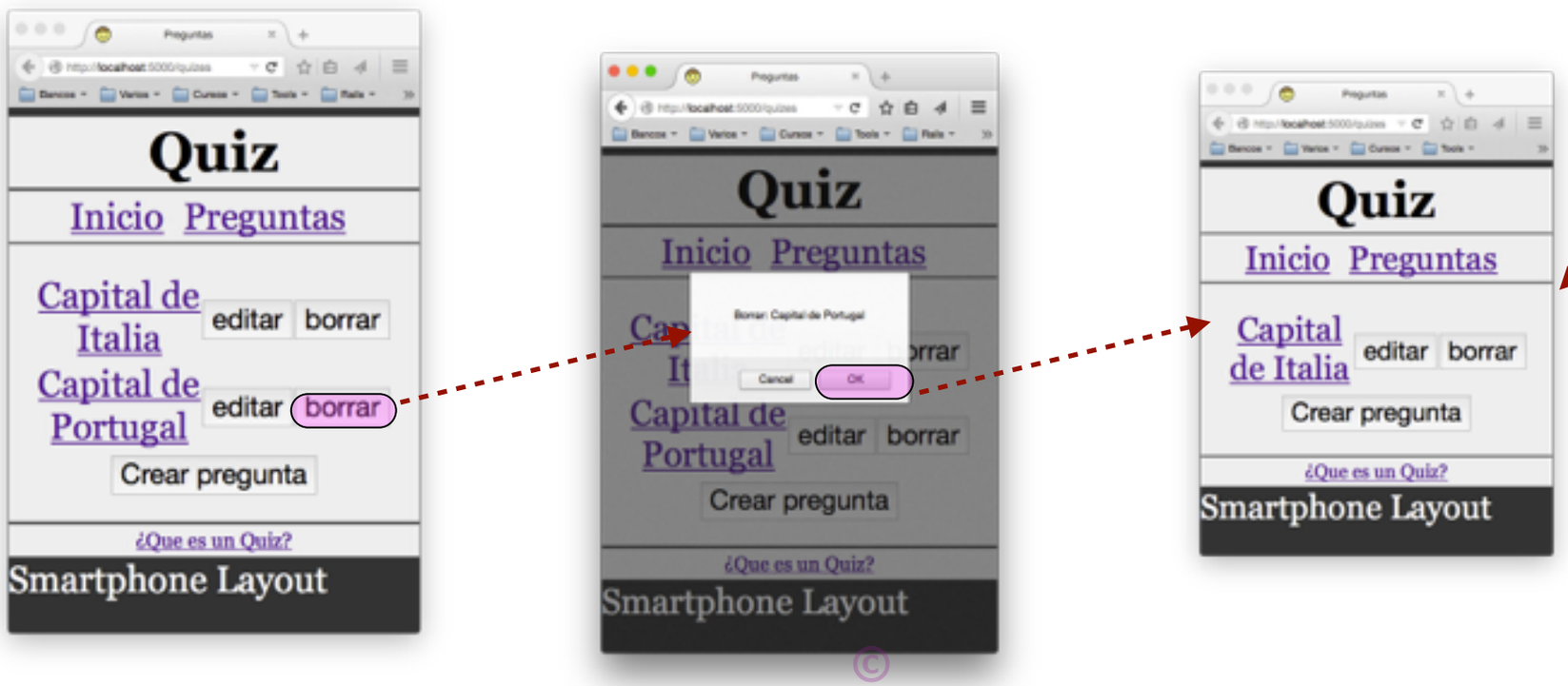
✦**Paso 1**: Añadir en vista **index.ejs** botones para borrar cada una de las preguntas

✦**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

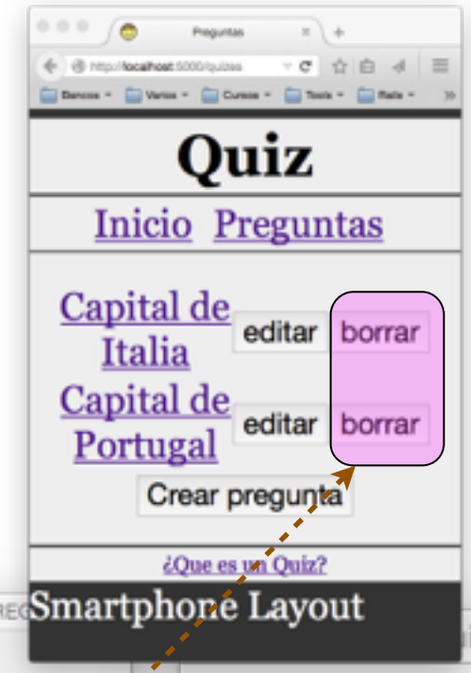
quiz_controller.js — UNREGISTERED

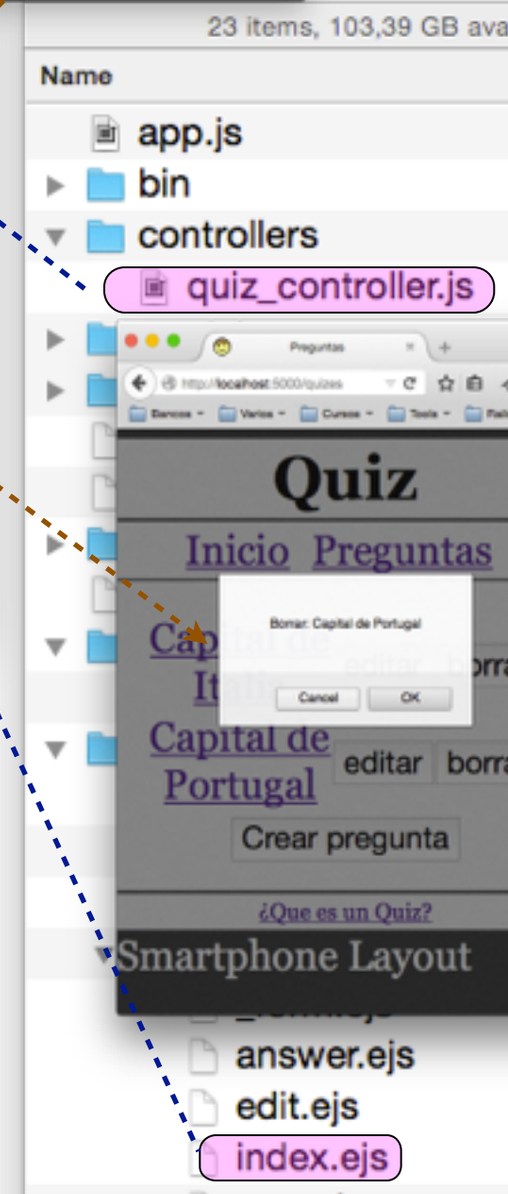
```
15  // GET /quizes
16  exports.index = function(req, res) {
17    models.Quiz.findAll().then(
18      function(quizes) {
19        res.render('quizes/index.ejs', {quizes: quizes, errors: []});
20      }
21    ).catch(function(error){next(error)});
22  };
```

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



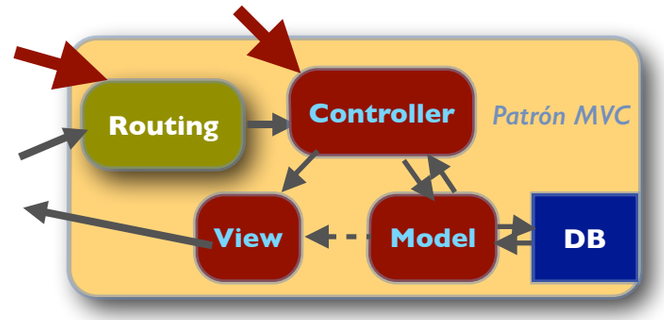


Se añade **botón de borrar** a cada pregunta en la lista de preguntas: **quiz/index.ejs**.

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.

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