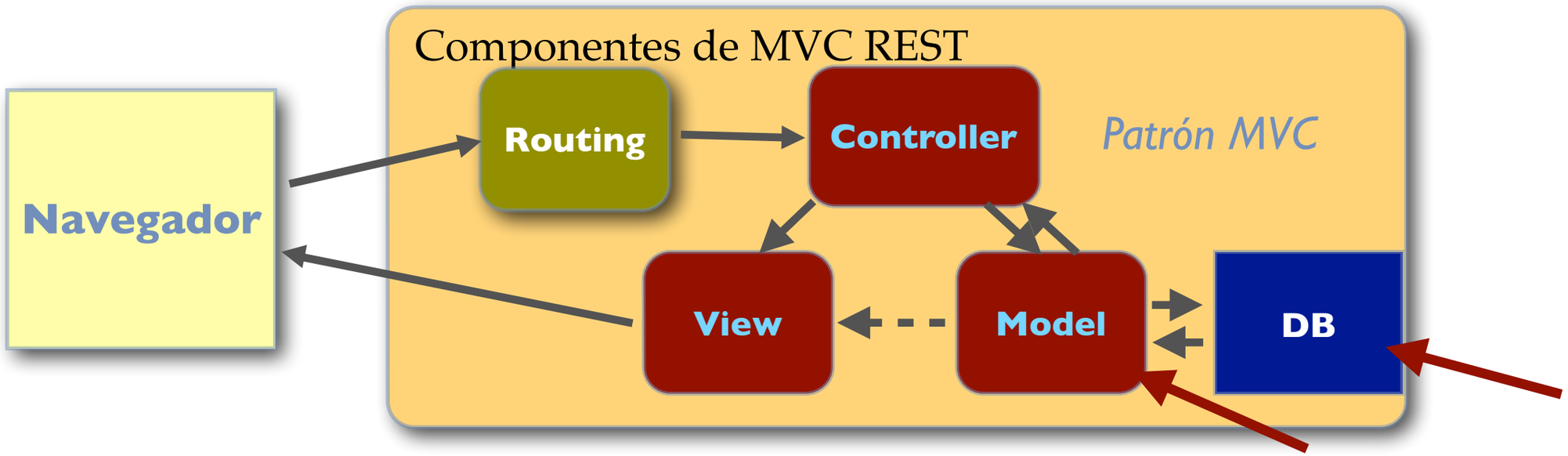
Quiz 7a: La base de datos

Juan Quemada, DIT - UPM

Quiz 7: Introducción de una base de datos

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

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



Bases de Datos

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

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

Tablas de una Base de Datos

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

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

Relaciones entre tablas

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

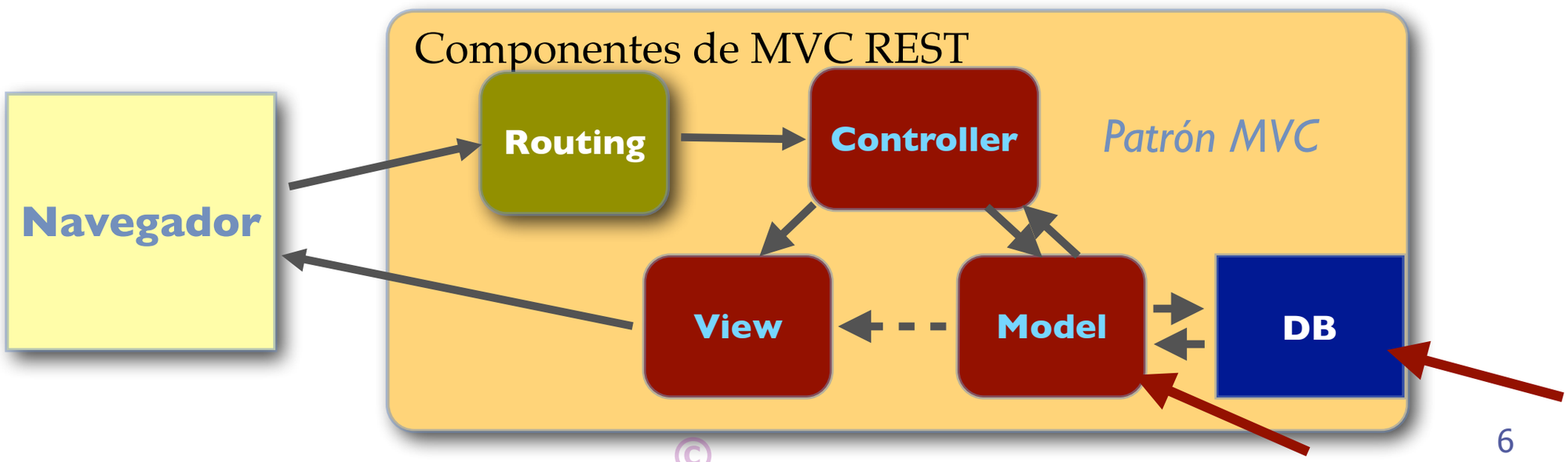
id	name	email	....	....	....	....	....	....	....
1	Juan López	jlopez@gmail.com	....	....	....	....	....	....	....
2	Javier Tello	javiertello@upm.es	....	....	....	....	....	....	....
3	.....	.....	....	....	....	....	....	....	....
4	Gabriel Tomé	gabit@dit.upm.es	....	....	....	....	....	....	....
5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

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

Modelo y DB

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

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



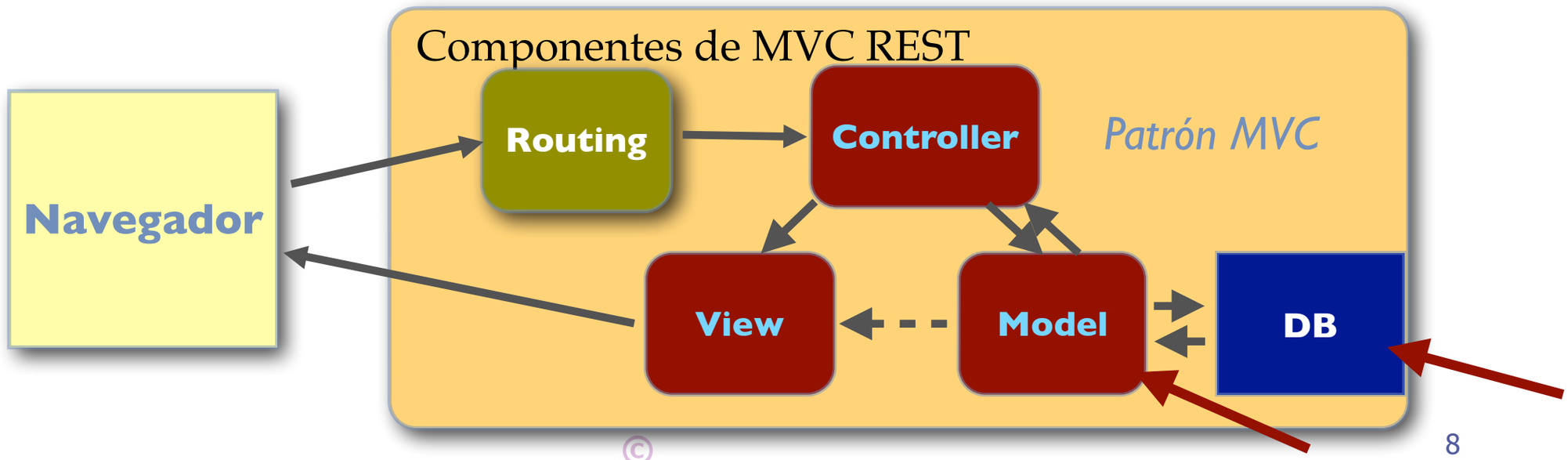
Quiz 7b: sequelize.js y SQLite

Juan Quemada, DIT - UPM

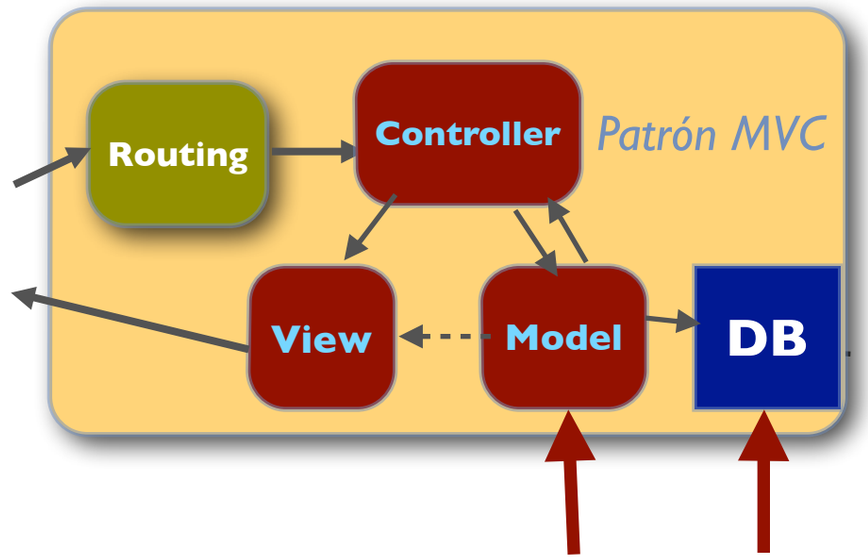
Quiz 7: sequelize.js y SQLite

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

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



Paso 1: Sequelize y SQLite3



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

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

..$ npm install --save sequelize@1.7.0

..$ npm install --save sqlite3@2.2.0

package.json

```
{
  "name": "application-name",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~3.4.8",
    "static-favicon": "~1.0.0",
    "morgan": "~1.0.0",
    "cookie-parser": "~1.0.1",
    "body-parser": "~1.0.0",
    "debug": "~0.7.4",
    "ejs": "~0.8.5",
    "express-partials": "~0.2.0",
    "sequelize": "~1.7.0",
    "sqlite3": "~2.2.0"
  }
}
```

Paso 2a: tabla Quiz

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

$ npm start

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

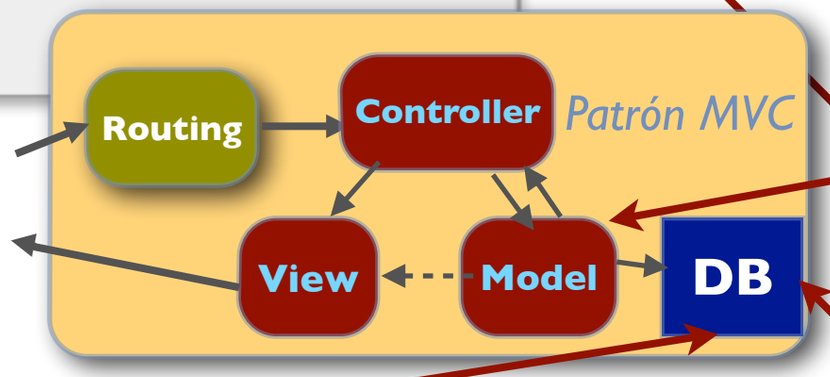
El fichero **quiz.js** es parte del modelo (**directorio models**) y define la estructura de la tabla de quizes (preguntas) con 2 campos (tipo string):

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

quiz.js

```
// Definicion del modelo de Quiz

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

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



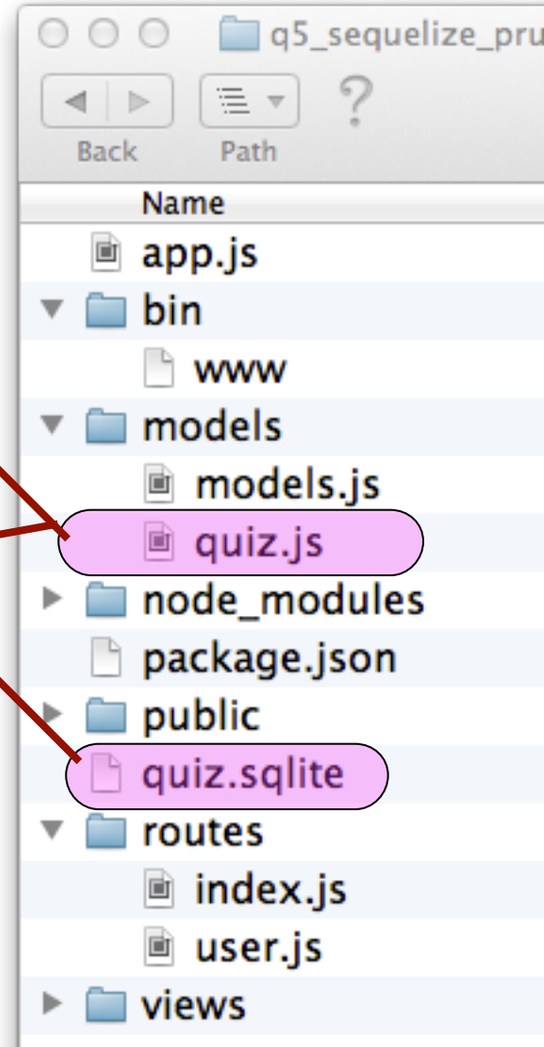


Tabla Quiz

Tipos de datos soportados en sequelize.js

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

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

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

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

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

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

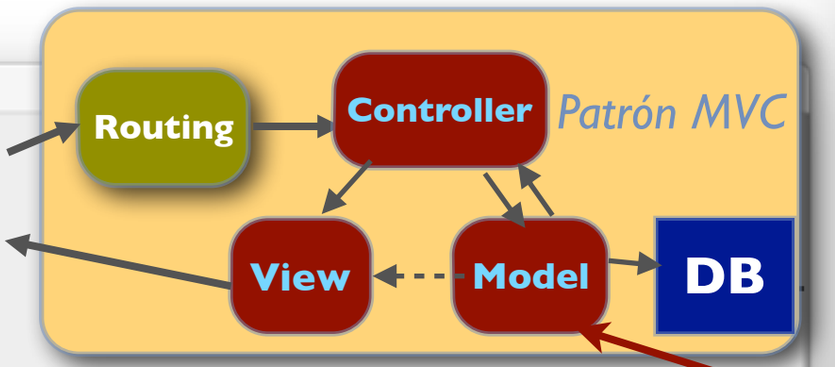
id	pregunta	respuesta
1	¿Cual es la Capital de Italia?	Roma
2	¿Quien descubrió América?	Cristobal Colón
3	¿Capital de Portugal?	Lisboa
4	.....	.....
5	.....	.....

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

El tipo STRING almacena texto.

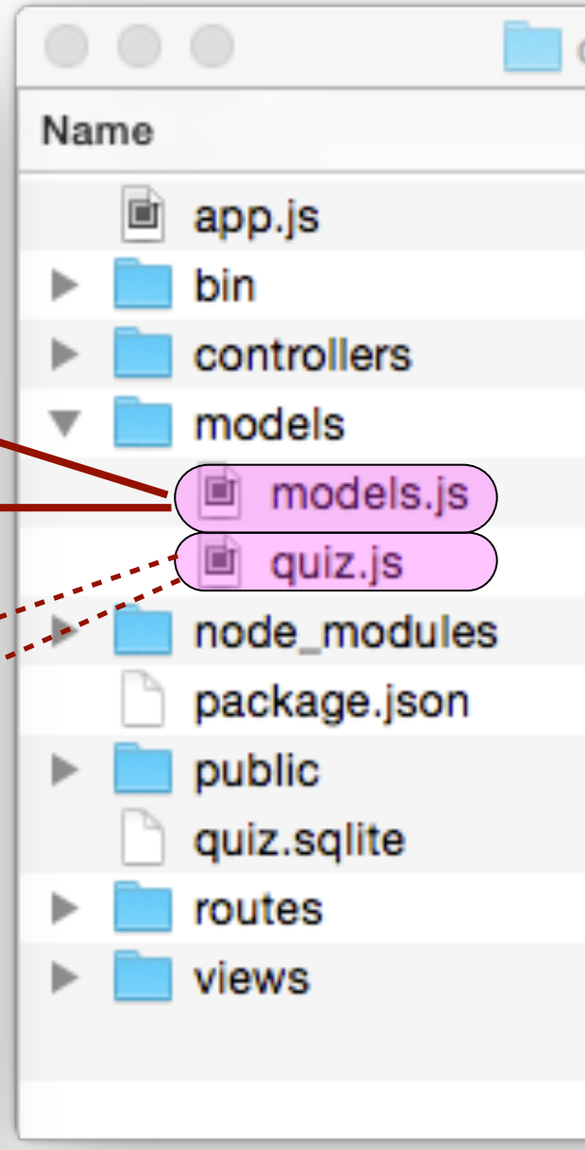
Paso 2b: models.js

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



models.js

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

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

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

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

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

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



Paso 2b: inicializar la DB

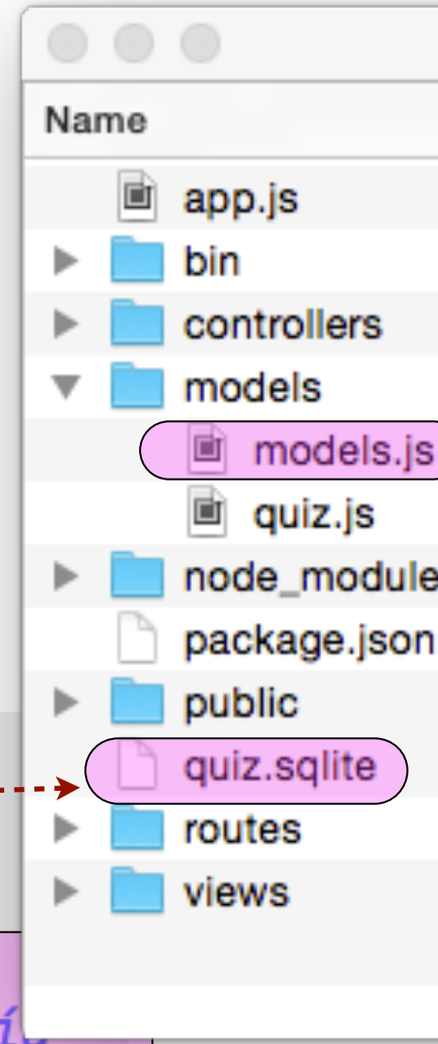
id	pregunta	respuesta
1	**¿Cual es la Capital de Italia?**	Roma
2		

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

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

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

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

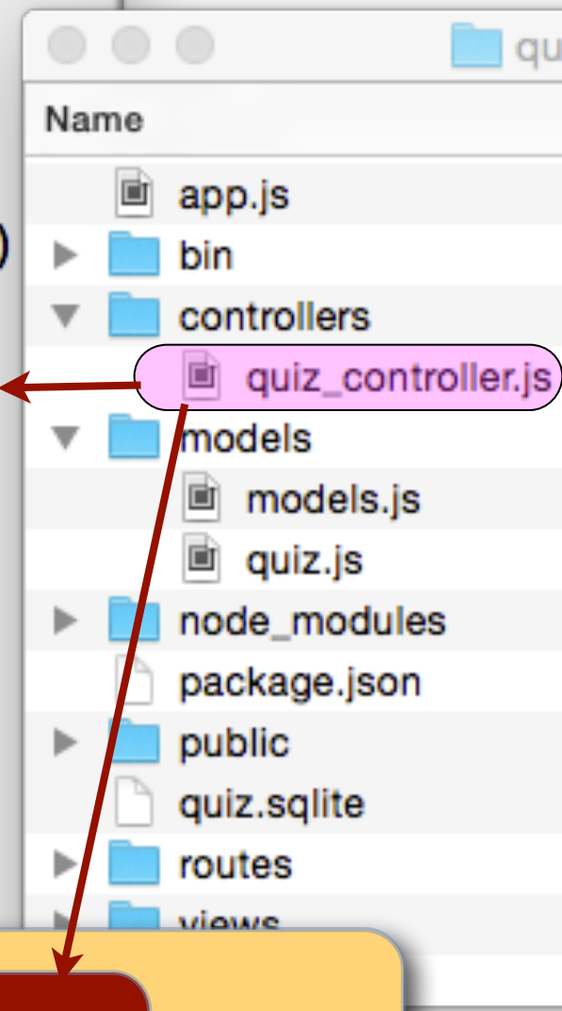



```
// sequelize.sync() crea e inicializa tabla de preguntas en DB
sequelize.sync().success(function() {
  // success(..) ejecuta el manejador una vez creada la tabla
  Quiz.count().success(function (count){
    if(count === 0) {   // la tabla se inicializa solo si está vacía
      Quiz.create({ pregunta: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

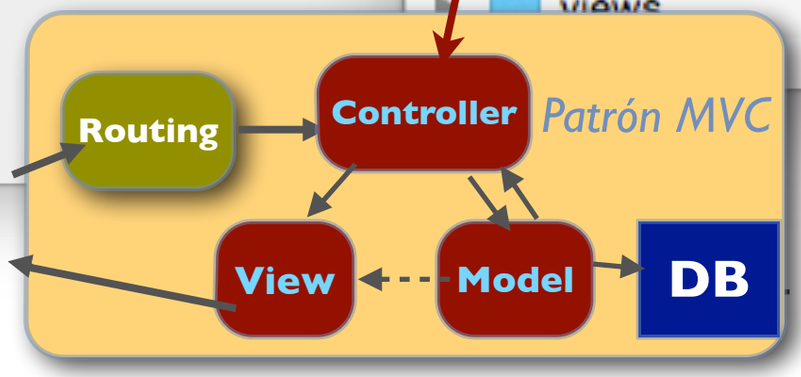
- El controlador importa el modelo para poder acceder a DB.
- Con los métodos **models.Quiz.findAll()** o **find()** buscamos los datos en la tabla **Quiz** y los procesamos en el callback del método **success(..)**.
- En el ejemplo usamos **findAll()** para buscar el array de elementos de la tabla **Quiz**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

quiz_controller.js — UNREGISTERED

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

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

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



Paso 3: controlador



Quiz 8: Despliegue DB en Heroku

Juan Quemada, DIT - UPM

Quiz 6: Despliegue DB en Heroku

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

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

Nota: Heroku se ha **actualizado** en **mayo de 2015** y **el Paso 2** puede **no necesitarse**, porque configura en el Paso 1 la variable DATABASE_URL con el URL a la base de datos.

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

Pasos 1 y 2: Añadir DB Postgress a Heroku

✦P1: desplegar DB Postgres en Heroku

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

✦P1a: desplegar desde el Dashboard

- Debe añadirse como un addon al despliegue

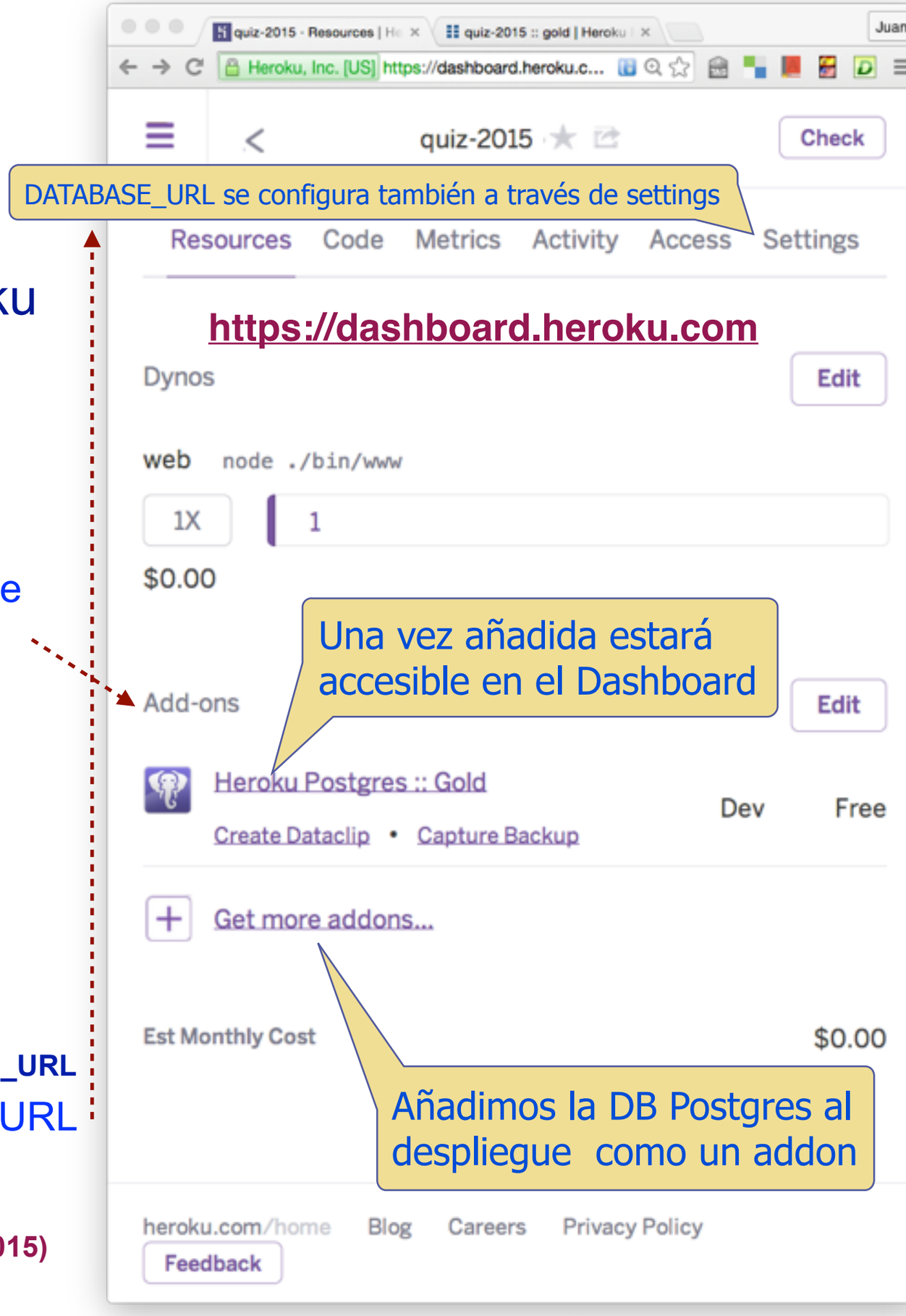
✦P1b: desplegar con Heroku Toolbelt

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

✦P2: Inicializar DATABASE_URL*

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

***Nota: Heroku** puede **no necesitar el Paso 2 (actualiz. mayo-2015)**



Resources Deploy Metrics Activity Access Settings

Name

quiz-2015

Rename

Nombre de la aplicación

Configuración de quiz-2015

Variables de entorno: Se debe inicializar DATABASE_URL con el contenido de HEROKU_POSTGRESQL_GOLD_URL

Config Variables

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

Config Vars

Edit

DATABASE_URL	postgres://tfewgztclgzzpw:G93ci
HEROKU_POSTGRESQL_GOLD_URL	postgres://tfewgztclgzzpw:G93ci
NODE_ENV	production

Las variables de entorno de ejecución de nuestra aplicación en Heroku, también se pueden editar desde el dashboard.

quiz-2015 en dashboard

Info

Region	US
Stack	cedar-14
Framework	Node.js
Git URL	git@heroku.com:quiz-2015.git

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

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

quiz-2015 :: gold

Base de datos Postgres desplegada para Quiz en Heroku

Connection Settings

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

Parámetros individuales extraidos del URL

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

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

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

$heroku pg:promoteb HEROKU_POSTGRESQL_GOLD_URL

Statistics

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

Estadísticas de uso de la base de datos en Heroku

DB en Dashboard de Heroku

Modificar proyecto Quiz

- Paso 3: crear parámetro "**devDependencies**" en package.json
  - Pasar "**sqlite3**" de "**dependencies**" a "**devDependencies**"
    - "**devDependencies**" se instalarán solo cuando instalemos el paquete en el entorno local
- Paso 4: añadir dependencia a Postgres a package.json
  - Comando "**npm install --save ..**" (guarda instalación en package.json)

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

$ npm install --save pg@4.1.1
$ npm install --save pg-hstore@2.3.2  //Ojo! en proyecto GITHUB no esta instalado
```

- Paso 5: Añadir variables de entorno a despliegue local
  - DATABASE_URL y DATABASE_STORAGE se crean en fichero .env
    - El fichero .env se creará en el directorio raíz del proyecto con el siguiente contenido

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

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

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

```
package.json                            UNREGISTERED
{
  "name": "quiz",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "engines": {
    "node": "0.10.x",
    "npm": "1.4.x"
  },
  "devDependencies": {
    "sqlite3": "^3.0.4"
  },
  "dependencies": {
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "debug": "~2.0.0",
    "ejs": "~0.8.5",
    "express": "~4.9.0",
    "express-partials": "^0.3.0",
    "morgan": "~1.3.0",
    "pg": "^4.1.1",
    "pg-hstore": "2.3.2",
    "sequelize": "^2.0.0-rc4",
    "serve-favicon": "~2.1.3"
  }
}
```

Algunos comandos útiles de Heroku Toolbelt

// Una vez creada la cuenta en Heroku e instalado Heroku Toolbelt en nuestro ordenador local,
// el comando local "heroku" invoca comandos remotamente en la consola de la máquina
// virtual donde hemos desplegado quiz-2015 en Heroku

..$ heroku login // Conectar a nuestra cuenta en Heroku con nuestras credenciales
Enter your Heroku credentials.
Email: jquemada@dit.upm.es // **Contestar que si** cuando nos
Password (typing will be hidden): ……. // pregunte si crea claves u otros
Authentication successful. // elementos necesarios.

..$ heroku create // crea una aplicación en heroku con un nombre aleatorio: **pure-reaches-3917**

..$ heroku apps:rename quiz-2015

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

..$ heroku logs // muestra los comandos invocados en heroku (consola remota) y su resultado
// **permite ver errores que han ocurrido** al desplegar y ejecutar y corregirlos

..$ heroku stop // Interrumpe la ejecución de quiz-2015 (stop es equivalente a ps:stop)
// es conveniente parar la máquina antes de desplegar una nueva versión

..$ heroku restart // Rearranca quiz-2015 (restar es equivalente a ps:restart)

..$ heroku apps // Interrumpe la ejecución de quiz-2015 (stop es equivalente a ps:stop)

Paso 6: adaptar modelo a despliegue en Heroku

models.js

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

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

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

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

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

exports.Quiz = Quiz; // exportar tabla Quiz

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

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

Ejecución local, versión y despliegue en Heroku

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

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

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

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

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

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

Quiz 9: Lista de preguntas

Juan Quemada, DIT - UPM

Quiz 9: Múltiples preguntas

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

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

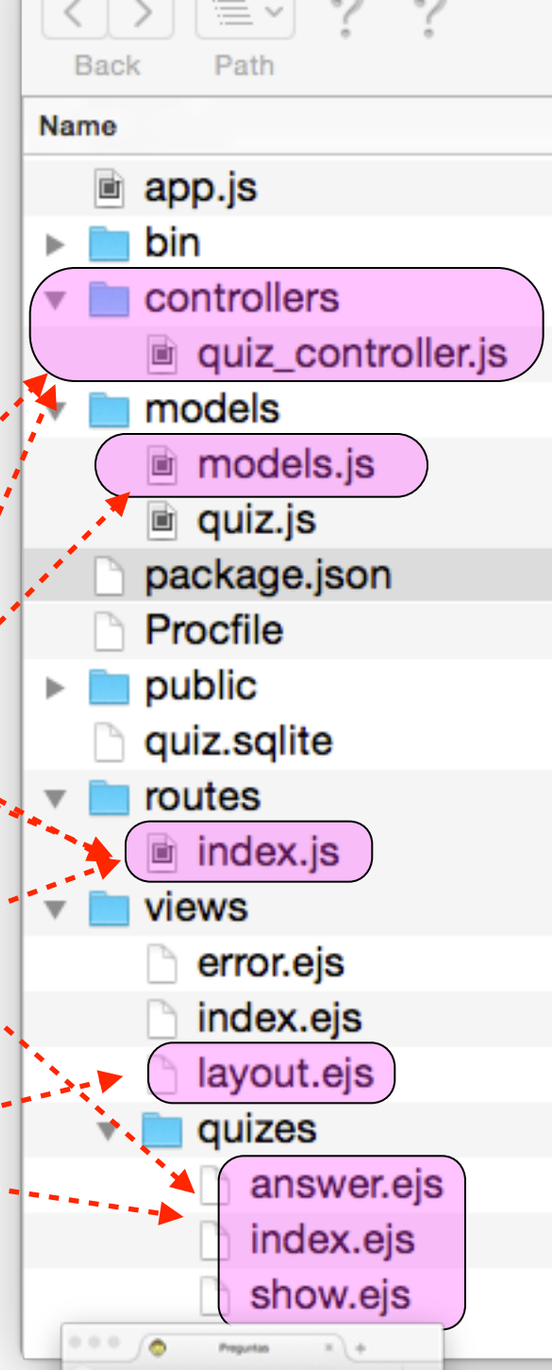
- a: Cambiar en controlador **quiz_controller.js**: **question** por **show** y modificar **answer**
- b1: Cambiar en **index.js**: **GET /quizes/question** por **GET /quizes/:quizId**
- b1: Cambiar en **index.js**: **GET /quizes/answer** por **GET /quizes/:quizId/answer**
- c: Adaptar vistas cambiando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

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

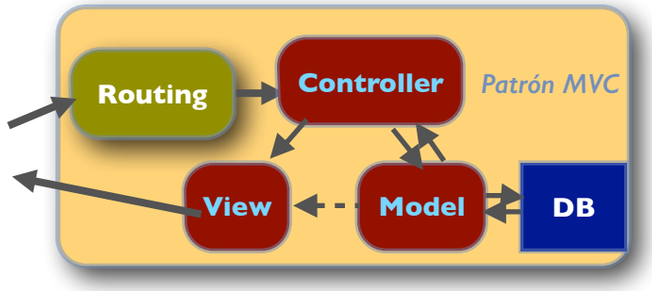
- a: Añadir al controlador **quiz_controller.js** la acción **index** asociada a la ruta **/quizes**
- b: Añadir otra pregunta al inicializar la base de datos en **models/models.js**
- c: Añadir la ruta **/quizes** (lista de preguntas) al enrutador **routes/index.js**
- d: Crear nueva vista con lista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

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

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



Recursos y nombres



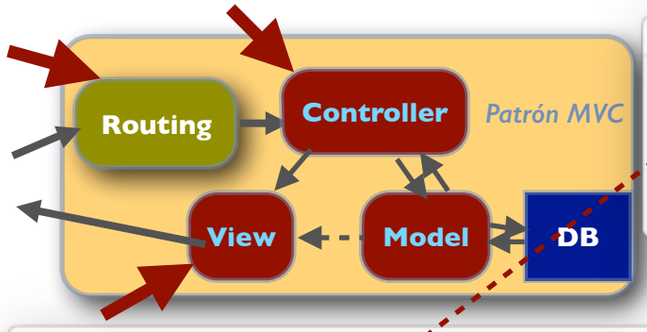
Cuando se diseña según el patrón MVC una tabla de una base de datos suele contener una lista de recursos. Para facilitar el diseño se siguen convenios de nombres similares para los elementos:

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

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

index.js — UNREGISTERED

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



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

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

routes/index.js

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

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

controllers/quiz_controller.js

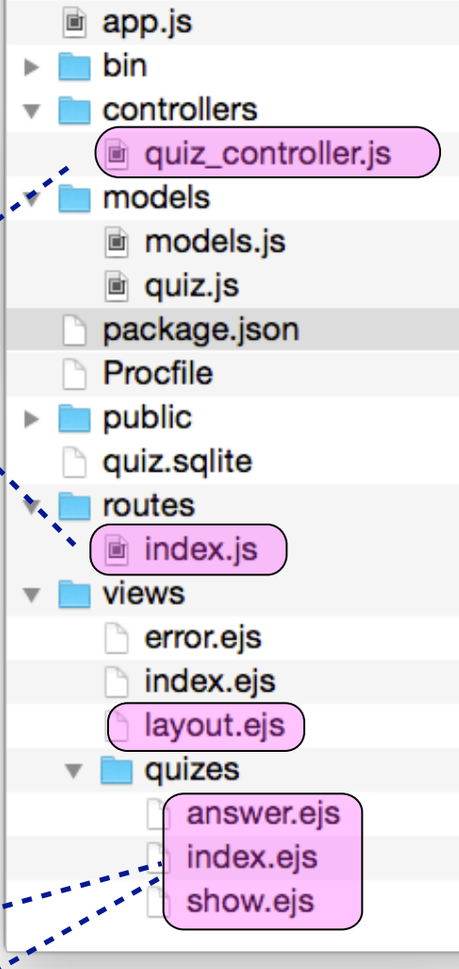
Paso 1a: controladores de **show** y **answer**

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

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

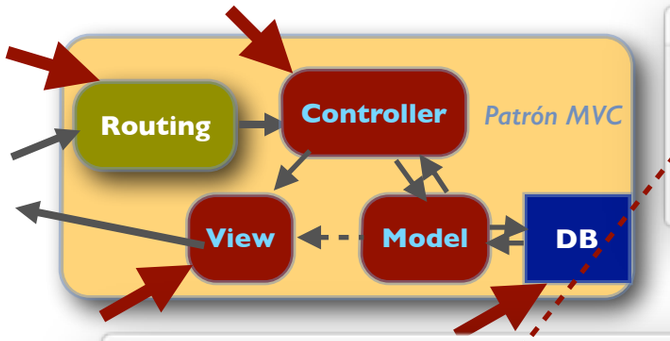
Paso 1c: vista views/quizes/show.ejs

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

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



Paso 1



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

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

routes/index.js

Paso 2a: controlador de index

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

controllers/quiz_controller.js

Paso 2

models/models.js

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

Paso 2b: inicializar DB

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

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

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

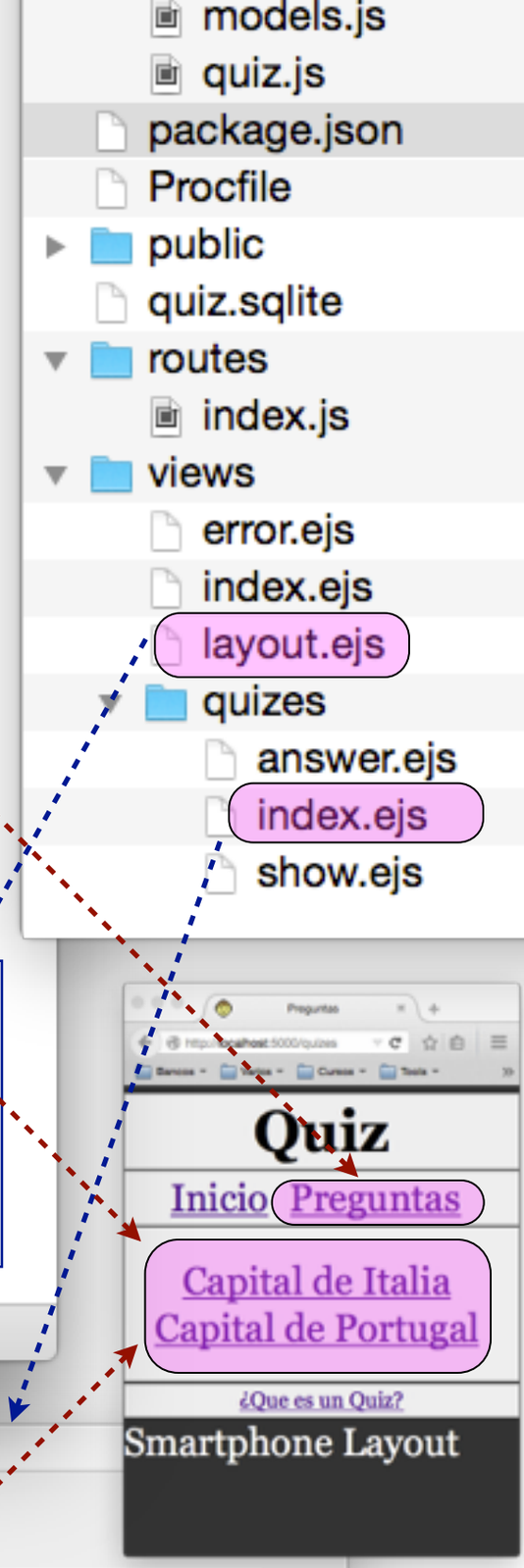
```
<table>
<% var i; for (i=0; i < quizes.length; i++) { %>
   <tr><td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td></tr>
<% } %>
</table>
```

Paso 3

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

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

  <section class="main" id="s1">
    <div>  <table>

     <tr><td><a href="quizes/1">Capital de Italia</a></td></tr>

     <tr><td><a href="quizes/2">Capital de Portugal</a></td></tr>

  </table>

</div>
  </section>

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

Paso 3: enlaze a **/quizes**

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



Vista views/quizes/index.ejs

index.ejs

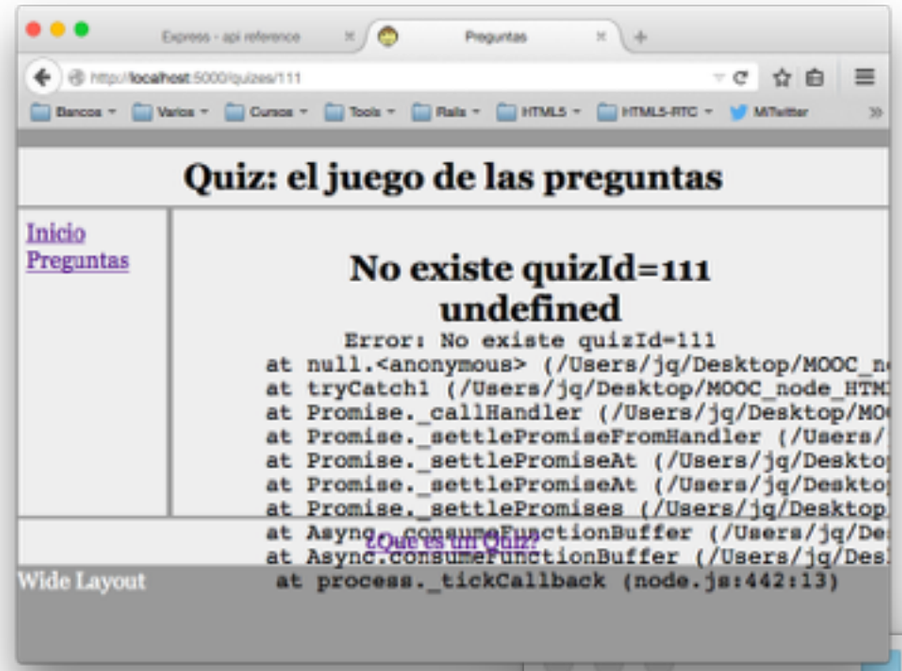
```
<table>
<% var i; for (i=0; i < quizes.length; i++) { %>
    <tr><td><a href="quizes/<%= quizes[i].id %>"><%= quizes[i].pregunta %></a></td></tr>
<% } %>
</table>
```

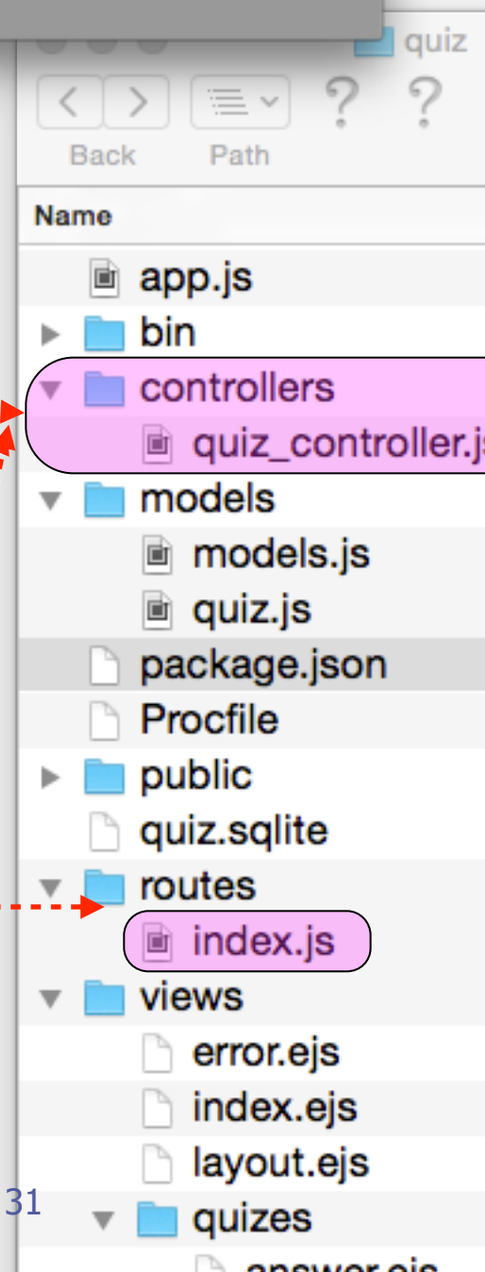
Bucle-for: renderíza el código HTML del bloque en cada iteración. Generará una fila de la tabla HTML por cada elemento del array quizes, que contiene todos los objetos de la tabla Quiz de la DB,

Quiz 10: Autoload

Juan Quemada, DIT - UPM

Quiz 10: Autoload





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

- **Paso 1**: Introducir gestión de errores en **GET /quizes**
  - La gestión de errores se introduce directamente en la primitiva
- **Paso 2**: Definir la función **load()** de lectura de un quiz identificado por **:quizId**
  - Gestiona también los errores de **GET /quizes/:quizId** y **GET /quizes/:quizId/answer**
- **Paso 3**: Adaptarlas siguientes rutas de acceso a preguntas para usar **load():**
  - **GET /quizes/:quizId**
  - **GET /quizes/:quizId/answer**
- **Paso 4**: Instalar el middleware load() en **router/index.js**
  - Instalada antes de **index.js** para que **load()** cargue quiz **antes** de invocar la ruta asociada
- **Paso 4**: Guardar **versión (commit)** git y subir a **Heroku**

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

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

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

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

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

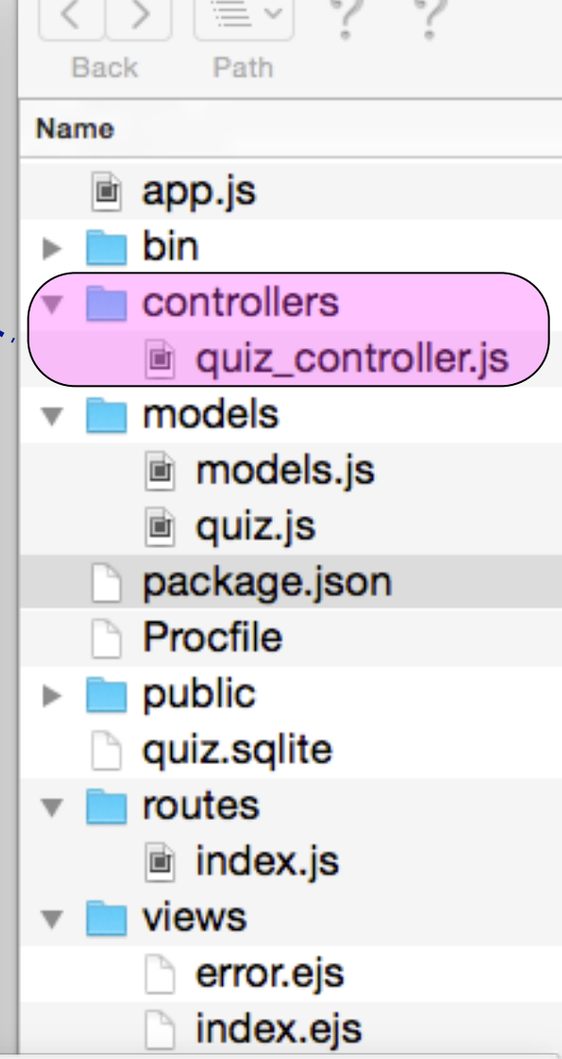
controllers/quiz_controller.js

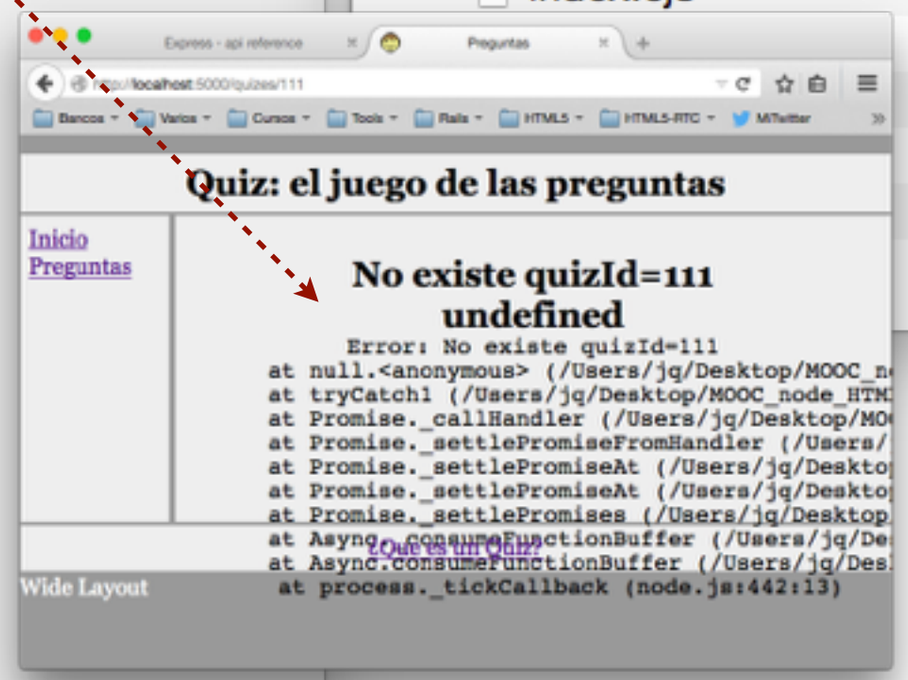
Paso 2

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

Paso 1

Paso 3



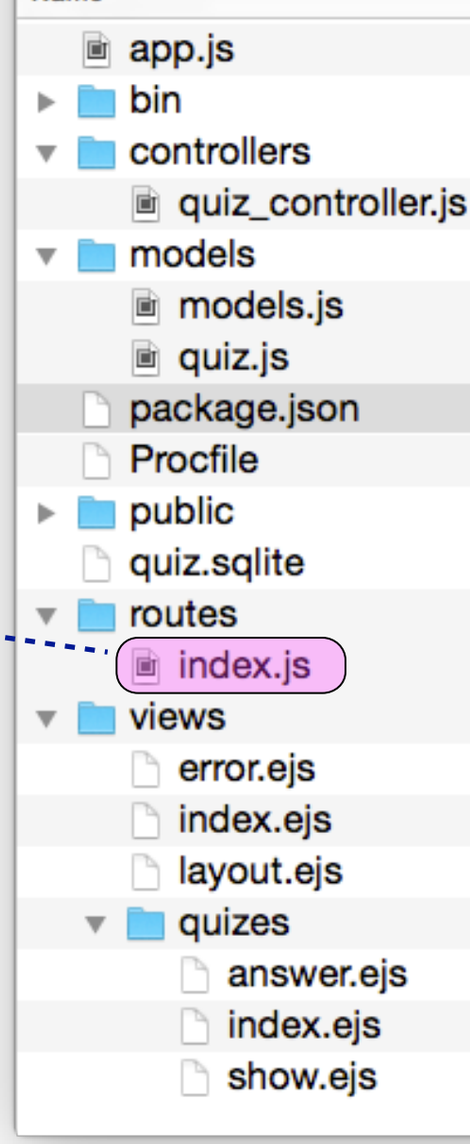


Ejecución de load()

index.js — UNREGISTERED

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

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

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

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

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

module.exports = router;
```

Paso 4



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

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

Final del tema