Proyecto de la asignatura CORE

Desarrollo del Servidor Quiz

Sequelize

CORE 2018-2019
Santiago Pavón

ver: 2019-02-28

© Santiago Pavón - UPM-DIT

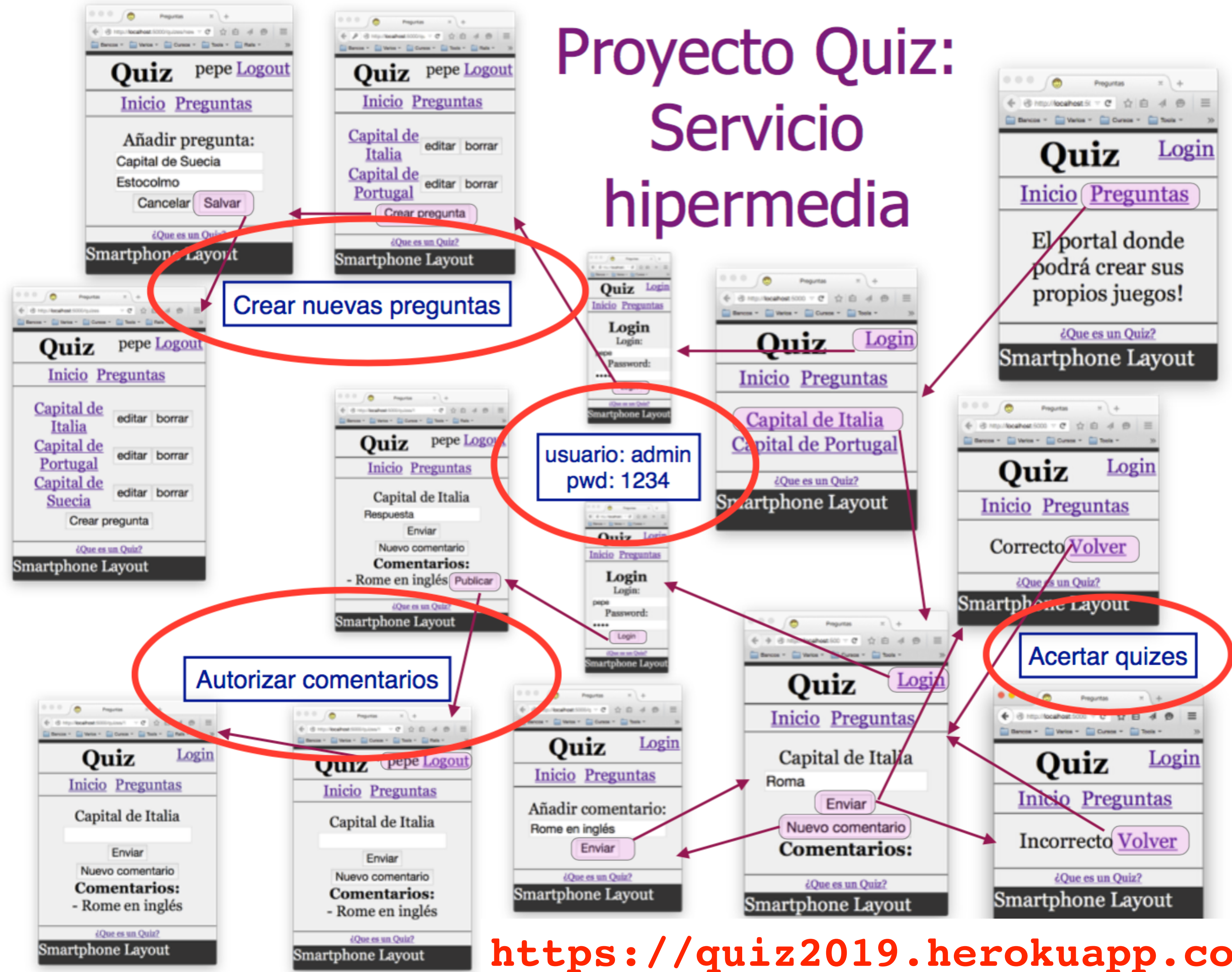
La Funcionalidad de Quiz

© Santiago Pavón - UPM-DIT

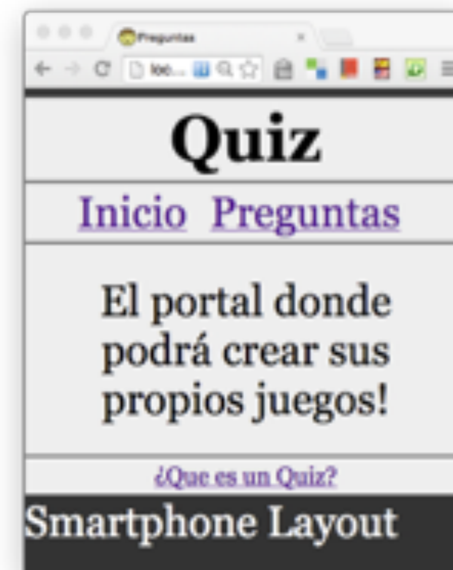
Ideas Generales

- Servicio web de publicación de Preguntas con su Respuesta (quiz).
- Cualquier usuario puede leer los Quizzes.
- Para publicar un quiz hay que registrarse y logearse.
- Para publicar una pista no hay que registrarse ni logearse.
  - Pero un administrador debe autorizar su publicación para que sea visible.
- Los quizzes pueden llevar imágenes adjuntas.
- Responsive design.
- Despliegue en la nube.
- . . .

© Santiago Pavón - UPM-DIT



Proyecto Quiz: Servicio hipermedia
Crear nuevas preguntas
usuario: admin
pwd: 1234
Autorizar comentarios
Acertar quizes
Quiz
Login
pepe Logout
Inicio Preguntas
Añadir pregunta:
Capital de Suecia
Estocolmo
Cancelar Salvar
Capital de Italia
Capital de Portugal
editar borrar
Crear pregunta
El portal donde podrá crear sus propios juegos!
¿Que es un Quiz?
Smartphone Layout
Correcto Volver
Incorrecto Volver
Roma
Enviar
Nuevo comentario
Comentarios:
- Rome en inglés
Publicar
Añadir comentario:
Rome en inglés
https://quiz2019.herokuapp.com



© Santiago Pavón - UPM-DIT

quiz2017.herokuapp.com

Preguntas | Management Console - Media library | Cloudinary Status

spg Logout

Quiz: el juego de las preguntas

Inicio
Preguntas
Mis Preguntas
Mis Favoritos
Créditos
Usuarios

pregunta a buscar | Buscar

Preguntas:

¿Qué serie es esta?
(by pepe)

Capital de Portugal
(by Anómino)

Capital de Francia
(by Anómino)

Capital de Italia
(by Anómino)

Capital de España
(by Anómino)

¿Cuantos centimos hay en la mitad de medio euro?
(by spg)
ver editar borrar

¿Cuál es el siguiente número en esta secuencia? 1, 10, 11, 100, 101, 110
(by spg)
ver editar borrar

¿Cómo se llama este meneito?
(by spg)
ver editar borrar

Crear pregunta

1 2

¿Qué es un Quiz?

Abrir "https://res.cloudinary.com/core-upm/image/upload/v1490...7/attachments/dvx95ogrgsyueciqpncc.jpg" en una pestaña nueva

quiz2017.herokuapp.com

Preguntas | Cloudinary Management Console - Media library | Cloudinary Status

spg Logout

Quiz: el juego de las preguntas

Inicio
Preguntas
Mis Preguntas
Mis Favoritos
Créditos
Usuarios

Jugar:

Capital de Francia
Respuesta | Comprobar
by Anómino

Pistas:

Nueva pista: Pista | Añadir

- La respuesta lleva un acento.

Volver

¿Qué es un Quiz?

Wide Layout

quiz2017.herokuapp.com

Preguntas | Cloudinary Management Con... | Cloudinary Status

spg Logout

Quiz: el juego de las preguntas

Inicio Preguntas Mis Preguntas Mis Favoritos Créditos Usuarios

Jugar:

Capital de Francia
Respuesta | Comprobar
by Anómino

Pistas:

Nueva pista: Pista | Añadir

- La respuesta lleva un acento.

Volver

¿Qué es un Quiz?

quiz2017.herokuapp.com

Preguntas | Cloudinary Manageme... | Cloudinary Status

spg Logout

Quiz

Inicio Preguntas Mis Preguntas Mis Favoritos Créditos Usuarios

Jugar:

Capital de Francia
Respuesta | Comprobar
by Anómino

Pistas:

Nueva pista: Pista | Añadir

- La respuesta lleva un acento.

Volver

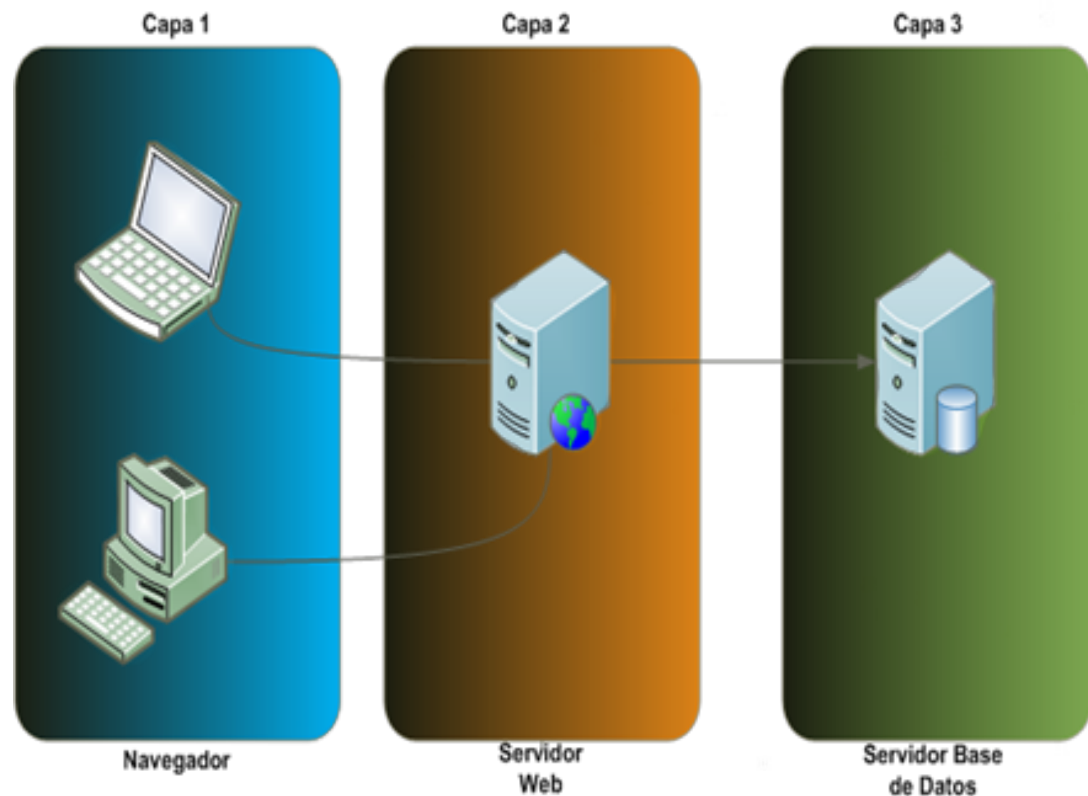
¿Qué es un Quiz?

Smartphone Layout

© Santiago Pavón - UPM-DIT

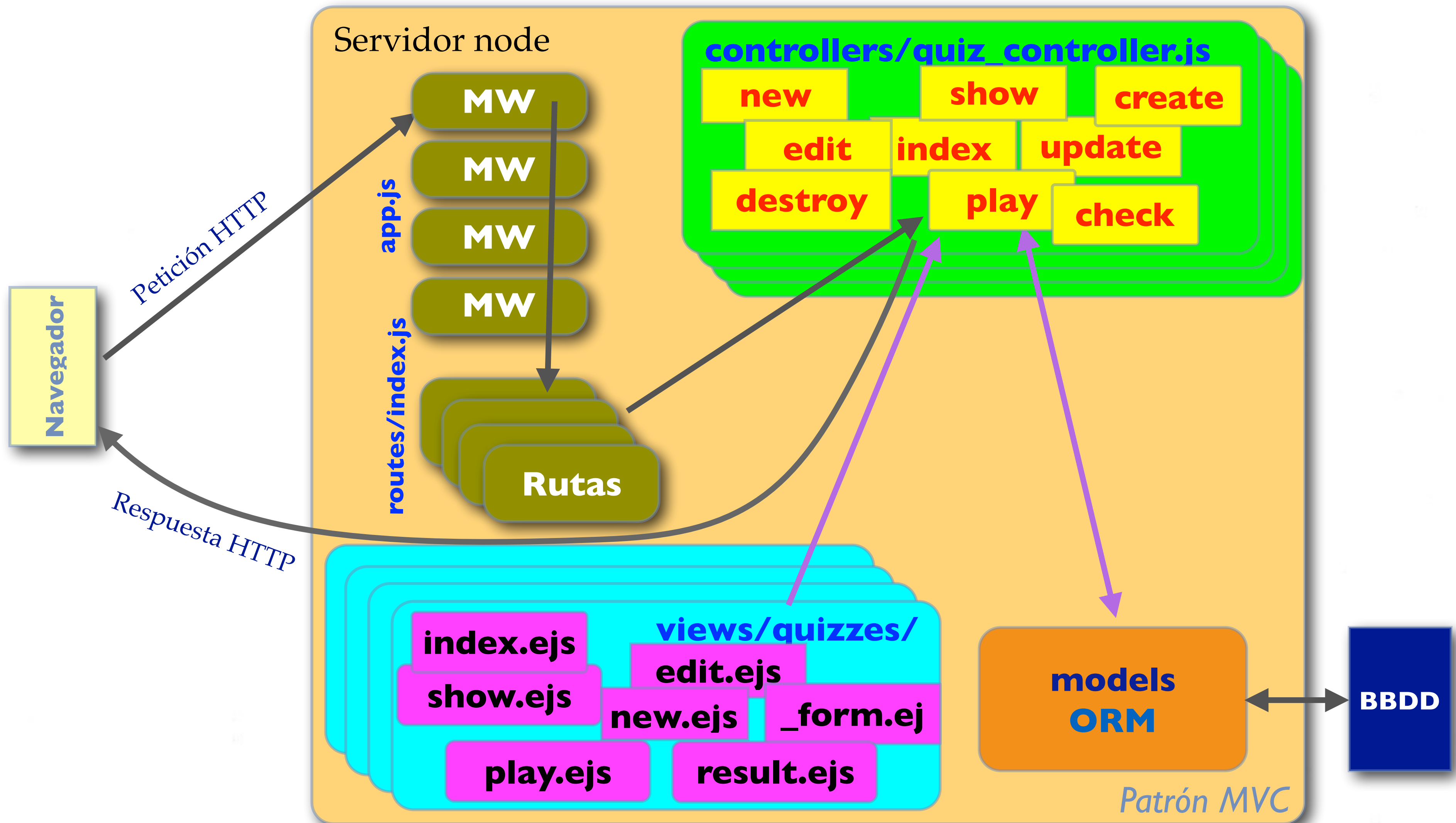
Arquitectura en Tres Capas

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



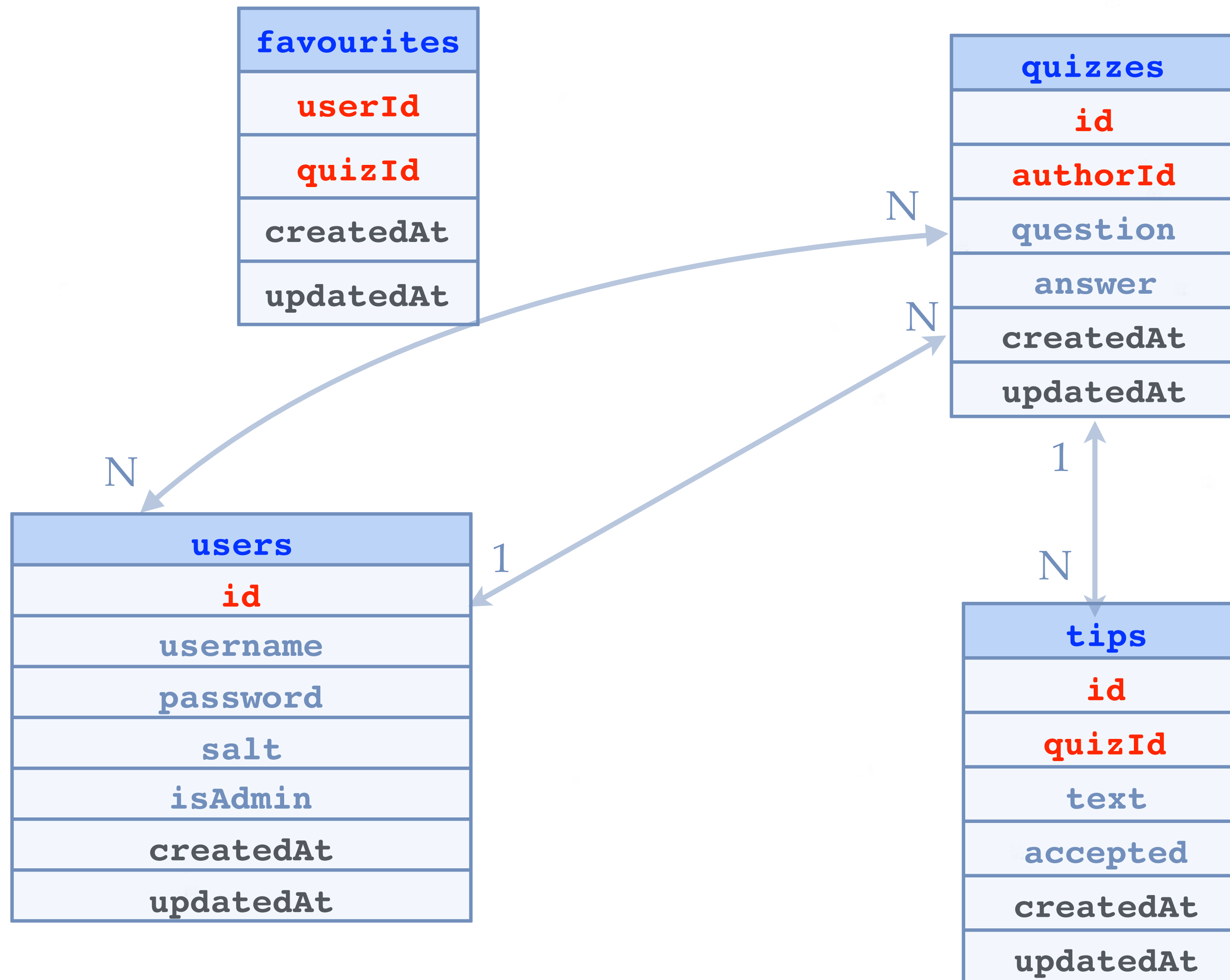
© Santiago Pavón - UPM-DIT

MVC



© Santiago Pavón - UPM-DIT

Base de Datos



© Santiago Pavón - UPM-DIT

El Modelo

SQLite y Sequelize

© Santiago Pavón - UPM-DIT

El Modelo

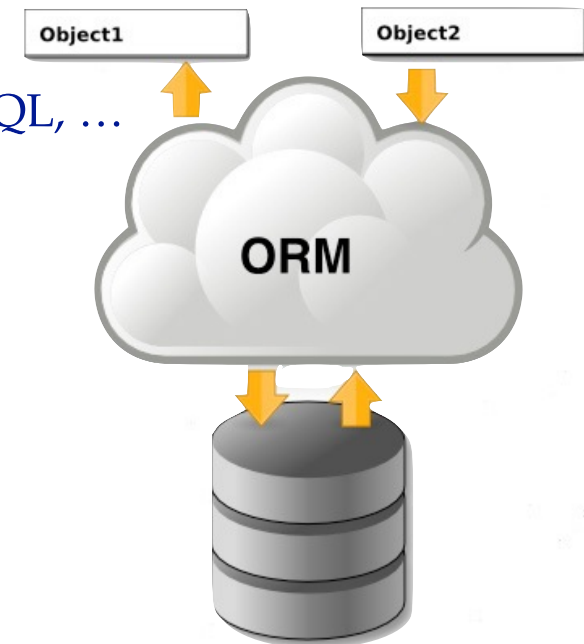
- Necesitamos dos cosas:

1. Una base de datos para almacenar los datos.
   - En local usaremos **SQLite**.
   - En Heroku usaremos **Postgres**.
2. Un **ORM** (Object-Relational Mapping) para acceder a la base de datos.
   - Usaremos **sequelize**.
     - Oculta los detalles de uso de la base de datos.
     - Sólo manejaremos objetos javascript.

© Santiago Pavón - UPM-DIT

Sequelize

- Documentación:
  - `http://www.sequelizejs.com`
- Es un módulo que proporciona ORM (Object-Relational Mapping)
  - Se crea una correspondencia entre los datos y las tablas de la base de datos, y objetos y clases en javascript.
    - Usamos la base de datos a través de estos objetos y clases.
- Características:
  - Soporta varias bases de datos: MySQL, SQLite, PostgreSQL, ...
  - Definición de esquemas.
  - Sincronización con la base de datos,
  - Validaciones
  - Métodos CRUD
  - Relaciones 1-a-1, 1-a-N, N-a-N.
  - Migraciones
  - . . .



© Santiago Pavón - UPM-DIT

Detalles

Para usar Sequelize hay que crear un objeto Sequelize describiendo cómo se accede a la base de datos.

Una vez creado el objeto Sequelize, definiremos los modelos de datos, es decir, cómo son los objetos javascripts que tienen su almacenamiento en la base de datos. Crearemos así los modelos quiz, user, tip, etc.

Después definiremos las relaciones entre estos modelos: 1 a 1, 1 a N, y N a N.

Ahora crearemos la tablas necesarias de la base de datos con sync o con migraciones.

Ya podemos crear objetos nuevos (create, build), buscar objetos en la base de datos (find, findAll), modificar los objetos, salvarlos (save) en la base de datos, borrar (destroy) los objetos, etc.

© Santiago Pavón - UPM-DIT

Instalar Sequelize y SQLite

- Nota: Para usar **npm** en el desarrollo de una aplicación nueva, hay que inicializar **npm** para que cree el fichero **package.json**. Para ello ejecutamos:

```
$ npm init
```

  - En el caso del proyecto **Quiz**, no es necesario ya que se generó al usar express-generator.

- Para instalar el módulo **Sequelize** ejecutamos:

```
$ npm install sequelize
```

- Para usar **Sequelize** con **SQLite,** hay que instalar el módulo **sqlite3:**

```
$ npm install sqlite3
```

  - Añade a la sección de dependencias las líneas:

```
"sequelize": "^4.42.0"
"sqlite3": "^4.0.6"
```

© Santiago Pavón - UPM-DIT

Usar Sequelize

- Hay que configurar **Sequelize** para indicar como se accede a la base de datos:

```
// Cargamos el modulo Sequelize.
const Sequelize = require('sequelize');

// Configurar Sequelize para usar SQLite.
// El fichero con la BBDD es quiz.sqlite.
const sequelize = new Sequelize("sqlite:quiz.sqlite");
```

© Santiago Pavón - UPM-DIT

Definir los Modelos

- Hay que definir cómo son los modelos **user**, **quiz**, **tip**, …
  - Se crea una clase para cada modelo definido.
  - Se especifica el nombre de la tabla de la bases de datos asociada, sus campos o atributos, sus tipos, las validaciones, etc.
    - Estos modelos están respaldados por las tablas **users**, **quizzes**, **tips**, … de la base de datos.
- También hay que definir cuales son las relaciones entre los modelos.
  - Para indicar quien es el autor de cada quiz se creará una relación 1 a N entre users y quizzes.
  - Para indicar a que quiz pertenece un comentario se creará una relación 1 a N entre quizzes y tips.
  - Para saber cuáles son los quizzes favoritos de cada usuario se creará una relación N a N entre users y quizzes.

© Santiago Pavón - UPM-DIT

Quizzes

- La tabla **quizzes** de la base de datos almacena todos los quizzes.
  - Esta tabla necesita tener las siguientes columnas:
    - **`id`**
      - Clave primaria. En un entero que se autoincrementa automáticamente.
    - **`authorId`**
      - id del autor del post. Es una clave externa a la tabla de usuarios.
    - **`question`**
      - String con la pregunta.
    - **`answer`**
      - String con la respuesta.
    - **`createdAt`**
      - Fecha de creación del quiz.
    - **`updatedAt`**
      - Fecha de actualización del quiz.

- Al definir el modelo **quiz**:
  - No hay que declarar los atributos **id**, **createdAt** y **updatedAt**. Se crean automáticamente.
  - Tampoco hay que crear el atributo **userId**. Se añade automáticamente al definir la asociación **1aN** entre **quiz** y **user**.

© Santiago Pavón - UPM-DIT

```
// Definicion del modelo Quiz:

let quiz = sequelize.define('quiz',
      { question: {
            type: Sequelize.STRING,
            validate: {
               notEmpty: {msg: "Falta Pregunta"}
            }
        },
        answer: {
            type: Sequelize.STRING,
            validate: {
               notEmpty: {msg: "Falta Respuesta"}
            }
        }
      });
```

© Santiago Pavón - UPM-DIT

Users

- La tabla **users** de la base de datos almacena los datos de todos los usuarios.
  - Esta tabla necesita tener las siguientes columnas:
    - `id`
      - Clave primaria. En un entero que se autoincrementa automáticamente.
    - `username`
      - String con el login del usuario.
    - `password`
      - String con la contraseña de acceso.
    - `salt`
      - Salt para cifrar la contraseña *(aunque no lo usaremos en este tema)*.
    - `isAdmin`
      - Booleano que indica si el usuario es un administrador.
    - `createdAt`
      - Fecha de creación del usuario.
    - `updatedAt`
      - Fecha de actualización del usuario.
- Al definir el modelo **user**:
  - No hay que declarar los atributos **id**, **createdAt** y **updatedAt**.

© Santiago Pavón - UPM-DIT

```
// Definicion del modelo User:

let user = sequelize.define('user',
      { username: {
           type: Sequelize.STRING,
           unique: true,
           validate: {
              notEmpty: {msg: "Falta username"}
           }
        },
        password: {
           type: Sequelize.STRING,
           validate: {
              notEmpty: {msg: "Falta password"}
           }
        },
        salt: {
           type: Sequelize.STRING
        },
        isAdmin: {
           type: Sequelize.BOOLEAN,
           defaultValue: false
        }
      });
```

© Santiago Pavón - UPM-DIT

Tips

- La tabla **tips** de la base de datos almacena las pistas de cada quiz.
  - Esta tabla necesita tener las siguientes columnas:
    - **`id`**
      - Clave primaria. En un entero que se autoincrementa automáticamente.
    - **`quizId`**
      - id del quiz al que se refiere la pista. Es una clave externa a la tabla de quizzes.
    - **`text`**
      - String con la pista.
    - **`accepted`**
      - Booleano que indica si se permite la visualización de la pista.
    - **`createdAt`**
      - Fecha de creación de la pista.
    - **`updatedAt`**
      - Fecha de actualización de la pista.

- Al definir el modelo **tip**:
  - No hay que declarar los atributos **id**, **createdAt** y **updatedAt**. Se crean automáticamente.
  - Tampoco hay que crear el atributo **quizId**. Se añade automáticamente al definir la asociación **1aN** entre **quiz** y **tip**.

© Santiago Pavón - UPM-DIT

```
// Definicion del modelo Tip:

let tip = sequelize.define('tip',
      { text: {
           type: Sequelize.STRING,
           validate: {
              notEmpty: {msg: "Falta el texto de la pista"}
           }
        },
        accepted: {
           type: Sequelize.BOOLEAN,
           defaultValue: false
      }
      });
```

© Santiago Pavón - UPM-DIT

Relaciones entre Modelos

- Entre usuarios y quizzes hay una relación **1 a N**.
  - un quiz pertenece o ha sido creado por un sólo usuario (autor).
  - un usuario puede haber creado varios quizzes.

- Sequelize permite crear esta relación añadiendo:

```
user.hasMany(quiz, {foreignKey: 'authorId'});
quiz.belongsTo(user, {as: 'author',
                      foreignKey: 'authorId'});
```

  - El módelo que invoca el método **belongsTo** es el modelo en el que se crea la clave externa que apunta al modelo relacionado.
    - Es decir, la tabla **quizzes** tiene la columna **authorId**.

© Santiago Pavón - UPM-DIT

- Sequelize define las relaciones **1 a 1**, **1 a N** y **N a N** entre los modelos usando los métodos **belongsTo**, **hasOne**, **hasMany** y **belongsToMany**.

- Al definir estas relaciones se crean automáticamente atributos y métodos que permiten acceder, añadir, borrar o consultar los datos de las clases asociadas con estas relaciones.

  - Por ejemplo, la llamada **user.hasMany(quiz)**
    - Crea un atributo llamado **userId** en el modelo **quiz.**
      - Nota: En nuestro caso hemos usado la opción **foreignKey** para que la clave externa en el modelo **quiz** se llame **authorId**.
    - Crea los métodos **getQuizzes**, **setQuizzes**, **addQuizzes, addQuiz, createQuiz, removeQuiz**, **removeQuizzes**, **hasQuizzes**, **hasQuiz y countQuizzes** en el prototipo de la clase del modelo **user.**
      - Estos métodos devuelven promesas de los valores manejados.

  - Por ejemplo, la llamada **quiz.belongsTo(user)**
    - Crea un atributo llamado **userId** en el modelo **Quiz**.
      - Nota: En nuestro caso hemos usado la opción **foreignKey** para que la clave externa en el modelo **quiz** se llame **authorId**.
    - Crea los métodos **getUser**, **setUser** y **createUser** en el prototipo de la clase del modelo **quiz**.
      - Nota: En nuestro caso hemos usado la opción **as** para que la relación se llame **author**, y por tanto los métodos de acceso creados se llaman **setAuthor**, **getAuthor** y **createAuthor**.
      - Estos métodos devuelven promesas de los valores manejados.

© Santiago Pavón - UPM-DIT

Detalles

Opciones usadas al especificar la asociación entre los modelos user y quiz:

- **Sin opciones**:

```
user.hasMany(quiz);
quiz.belongsTo(user);
```

  - Con estas declaraciones, el campo de la tabla **quizzes** usado como clave externa que apunta a la tabla **users** debe llamarse **userId**.
  - En este caso, al definir el modelo **quiz** (con el método **sequelize.define**) no es necesario escribir explícitamente el campo **userId**.

  - El campo usado como clave externa podría llamarse **user_id** si se hubiera usado la opción **underscored** al crear el objeto **sequelize**.

  - El valor del primer parámetro se usa para crear los nombres de los métodos de acceso de la asociación:
    - **addQuiz**, **removeQuiz**, **hasQuiz**, **hasQuizzes**, . . .
    - **getUser**, **setUser**, **createUser**
  - Para cambiar el nombre de la asociación y de los métodos de acceso creados, se usa la opción **as**.

- **Opción foreignKey**:

```
user.hasMany(quiz);
quiz.belongsTo(user, {foreignKey: 'authorId'});
```

  - Si el campo de la tabla **quizzes** usado como clave externa que apunta a la tabla **users** no se llama **userId**, entonces debe indicarse su nombre con la opción foreignKey.

- **Opción as**:

```
user.hasMany(quiz);
quiz.belongsTo(user, {as: 'author', foreignKey: 'authorId'});
```

  - Con la opción as el nombre de la asociación es **author**, y los métodos de acceso creados son **getAuthor**, **setAuthor**, **createAuthor**.

© Santiago Pavón - UPM-DIT

Relaciones entre Modelos

- Entre pistas y quizzes hay una relación **1 a N**.
  - una pista pertenece o aclara un sólo quiz.
  - un quiz puede tener varias pistas.

- Sequelize permite crear esta relación añadiendo:

```
quiz.hasMany(tip);
tip.belongsTo(quiz);
```

  - El módelo que invoca el método **belongsTo** es el modelo en el que se crea la clave externa que apunta al modelo relacionado.
    - Es decir, la tabla **tips** tiene la columna **quizId**.

© Santiago Pavón - UPM-DIT

Favoritos

- Favoritos es una relación **N a N** entre usuarios y quizzes.
  - Un usuario puede tener varios quizzes que le gusten mucho, y los marcará como sus favoritos.
  - Un mismo quiz puede haber sido marcado como favorito por varios usuarios.

- No hay que crear un nuevo modelo para implementar esta relación.

- Esta relación se define añadiendo:

```
user.belongsToMany(quiz, {as: 'favouriteQuizzes', through: 'favourites',
                          foreignKey: 'userId', otherKey: 'quizId'});
quiz.belongsToMany(user, {as: 'fans', through: 'favourites',
                          foreignKey: 'quizId', otherKey: 'userId'});
```

  - La opción **as** define cual el nombre del atributo que conecta los modelos, y el nombre de los métodos de acceso creados: **setFavouriteQuizzess**, **getFans**, …
  - La opción **through** indica cuál es el nombre de la tabla join creada.
  - Las opciones **foreignKey** y **otherKey** lindican el nombre de las claves usadas en la tabla join

- Se crea una tabla **join**, llamada **favourites**, que no tiene la columna id.
  - Solo tiene las columnas: **userId**, **quizId**, **createdAt** y **updatedAt**.
  - Cada registro de esta tabla relaciona un usuario y un quiz.

© Santiago Pavón - UPM-DIT

Crear la Tablas

- Para que la aplicación funcione es necesario que la base de datos y sus tablas estén ya creadas.
- Crearemos la base de datos y todas las tablas que se necesitan invocando **sequelize.sync()**.
  - Este método crea todas las tablas que necesitan y que aún no existen.
    - Las tablas que ya existen no se vuelven a crear.
  - Hay que tener en cuenta que si se modifica la definición de un modelo, no se rehace automáticamente la tabla asociada.
    - Para rehacer una tabla se debe pasar la opción **{force: true}** en la llamada a **sequelize.sync()**.
      - Se borrará la tabla actual y se creará una nueva tabla vacía (perdiendo los datos existentes).
- Nota: En la práctica, la creación y modificación de las tablas debe hacerse usando **migraciones**. Son programas creados especialmente para hacer evolucionar la base de datos, creando nuevas tablas, índices, campos, etc.
  - Veremos cómo se usan más adelante.

© Santiago Pavón - UPM-DIT

- En la tabla quizzes debe existir la columna **authorId** para implementar su relación con users.
  - Esta columna se crea automáticamente al invocar **sequelize.sync()** ya que hemos definido previamente la relación entre los modelos user y quiz.

- Si la tabla quizzes se hubiera creado antes de definir la relación con el modelo user, la columna **authorId** no existiría y deberíamos crearla.
  - Algunas opciones para añadir la columna **authorId**:
    1. Borrar el fichero con la base de datos **quiz.sqlite**.
       - La próxima vez que ejecutemos el servidor, **sequelize.sync()** creará otra vez el fichero de la base de datos **quiz.sqlite**, y la tabla **quizzes** con la columna **authorId**.
       - Con esta opción perderemos todos los datos existentes.
    2. Añadir la sentencia `quiz.sync({force: true});`
       - Destruye la tabla **quizzes** y se vuelve a crear.
         - Una vez creada la nueva tabla, borrar esta sentencia.
       - Con esta opción perderemos todos los datos de la tabla **quizzes**.
    3. Usar una **migración** que cree la columna **authorId**.
       - Con esta opción no se perderá ningún dato.
       - Las migraciones las estudiaremos más adelante.

- El mismo razonamiento aplica para la columna **quizId** de la tabla **tips**.

© Santiago Pavón - UPM-DIT

Crear un Módulo

- Todo el código usado para inicializar Sequelize, definir los modelos, las relaciones entre ellos, y crear las tablas puede ser muy largo.
- Vamos a reestructurarlo para que quede más claro y cómodo de usar.
  - Crearemos un módulo llamado **models** que contenga todo el código anterior.
    - Este módulo exportará el objeto **sequelize** creado.
    - El objeto sequelize tiene una propiedad llamada **models** que contiene referencias a todos los modelos definidos/creados.
  - Y la definición de cada modelo se realizará en un fichero independiente
    - que importaremos con el método **sequelize.import**.
- Así, bajo el directorio **models** tendríamos los siguiente ficheros **index.js**, **quiz.js**, **user.js** y **tip.js**.
- Para cargar el módulo ejecutaremos:

```
const sequelize = require("../models");
const models = sequelize.models;
```

  - Otra forma equivalente de cargar el módulo:

```
const {models} = require("../models");
```

  - y accederemos a los modelos con **models.quiz**, **models.user** y **models.tip**.

© Santiago Pavón - UPM-DIT

```
const path = require('path');
const Sequelize = require('sequelize');

// Configurar Sequelize para usar SQLite.
// El fichero con la BBDD es quiz.sqlite.
const sequelize = new Sequelize("sqlite:quiz.sqlite");

// Importar modelos.
sequelize.import(path.join(__dirname,'quiz'));
sequelize.import(path.join(__dirname,'user'));
sequelize.import(path.join(__dirname,'tip'));

// Relaciones entre modelos
const {quiz, tip, user} = sequelize.models;

// Relacion 1 a N entre User y Quiz:
user.hasMany(quiz, {foreignKey: 'authorId'});
quiz.belongsTo(user, {as: 'author', foreignKey: 'authorId'});

// Relacion 1 a N entre Tip y Quiz:
quiz.hasMany(tip);
tip.belongsTo(quiz);

// Relacion N a N para los favoritos:
quiz.belongsToMany(user, {as: 'fans', through: 'favourites',
                          foreignKey: 'quizId', otherKey: 'userId'});

user.belongsToMany(quiz, {as: 'favouriteQuizzes', through: 'favourites',
                          foreignKey: 'userId', otherKey: 'quizId'});

// Crear tablas pendientes:
sequelize.sync()

// Exportar modelos:
module.exports = sequelize;
```

models/index.js

© Santiago Pavón - UPM-DIT

```
// Definicion del modelo Quiz:

module.exports = function(sequelize, DataTypes) {
  return sequelize.define('quiz',
      { question: {
           type: DataTypes.STRING,
           validate: {
              notEmpty: {msg: "Falta Pregunta"}
           }
        },
        answer: {
           type: DataTypes.STRING,
           validate: {
              notEmpty: {msg: "Falta Respuesta"}
           }
        }
      });
};
```

quiz.js

© Santiago Pavón - UPM-DIT

```
// Definicion del modelo Tip:

module.exports = function(sequelize, DataTypes) {
  return sequelize.define('tip',
      { text: {
            type: DataTypes.STRING,
            validate: {
               notEmpty: {msg: "Falta el texto de la pista"}
            }
         },
         accepted: {
            type: DataTypes.BOOLEAN,
            defaultValue: false
         }
      });
};
```

tip.js

© Santiago Pavón - UPM-DIT

```
// Definicion del modelo User:

module.exports = function(sequelize, DataTypes) {
  return sequelize.define('user',
      { username: {
           type: DataTypes.STRING,
           unique: true,
           validate: {
              notEmpty: {msg: "Falta username"}
           }
        },
        password: {
           type: DataTypes.STRING,
           validate: {
              notEmpty: {msg: "Falta password"}
           }
        },
        salt: {
           type: DataTypes.STRING
        },
        isAdmin: {
           type: DataTypes.BOOLEAN,
           defaultValue: false
        }
      });
};
```

user.js

© Santiago Pavón - UPM-DIT

Carga Ansiosa

- Una característica muy cómoda es la carga ansiosa de asociaciones.
  - Consiste en recuperar datos de varias tablas con una única llamada a **findOne** o **findAll**.
    - Se hace usando la opción **include**.
      - Se indican los modelos relaciones de los que hay que cargar objetos.
      - Esta opción provoca que se realice una petición SQL de tipo JOIN.
  - También se usa para crear varios objetos asociados.

© Santiago Pavón - UPM-DIT

Ejemplos de Uso
CRUD

© Santiago Pavón - UPM-DIT

Crear y Salvar un User

versión: 1.1-crear_user.js

```
const Sequelize = require("sequelize");

const {models} = require("../models");

const user = models.user.build( { username: "Pepe",  // No es persistente
                                  isAdmin: true
                                });

user.password = "1234"; // Lo edito

user.save()  // Persistente
.then(user => {
    console.log('Usuario ' +  user.username + ' creado con éxito.');
})
.catch(Sequelize.ValidationError, error => {
    console.log("Errores de validación:");
    error.errors.forEach(({message}) => console.log('Error:', message));
})
.catch(error => {
    console.log("Error:", error);
});
```

- Uso build para crear una instancia no persistente de user.
- Después la edito y la salvo en la BBDD.
- .catch(ErrorType, cb) solo soportado por las promesas bluebird usadas por Sequelize.

© Santiago Pavón - UPM-DIT

versión: 1.2-crear_user.js

```
const Sequelize = require("sequelize");

const {models} = require("../models");

models.user.create( { username: "Pepe",      // Persistente
                      password:  "1234",
                      isAdmin: true
                    })
.then(user => {
    console.log('Usuario ' +  user.username + ' creado con éxito.');
})
.catch(Sequelize.ValidationError, error => {
    console.log("Errores de validación:");
    error.errors.forEach(({message}) => console.log('Error:', message));
})
.catch(error => {
    console.log("Error:", error);
});
```

- Uso create para crear una instancia persistente de user.
- Al crearla ya está guardada en la BBDD.

© Santiago Pavón - UPM-DIT

Buscar un User

versión: 2-buscar_user.js

```
const {models} = require("../models");

models.user.findOne( { where: {username: "Pepe"}} )
.then(user => {
    if (user) {
        console.log('Usuario encontrado:', user.username);
    } else {
        throw new Error("No existe el Usuario");
    }
})
.catch(error => {
    console.log("Error:", error);
});
```

- Busco un usuario por su nombre.

© Santiago Pavón - UPM-DIT

Crear Varios Quizzes

versión: 3.1-crear_3_quizzes.js

```
const {models} = require("../models");

models.user.findOne( { where: {username: "Pepe"}} )
.then(user => {
    if (user) {
        console.log('Usuario encontrado.');

        return models.quiz.bulkCreate(
            [ {question: '1+2', answer: '3', authorId: user.id},
              {question: '3*2', answer: '6', authorId: user.id},
              {question: '5-2', answer: '3', authorId: user.id}
            ])
            .then(quizzes => {
                console.log('Creados varios quizzes.');
            });
    } else {
        throw new Error("No existe el Usuario");
    }
})
.catch(error => {
    console.log("Error:", error);
});
```

- Busco un usuario por su nombre y le añado tres quizzes..

© Santiago Pavón - UPM-DIT

versión: 3.2-crear_3_quizzes.js

```
const {models} = require("../models");

models.user.findOne( { where: {username: "Pepe"}} )
.then(user => {
    if (!user) {
        throw new Error("No existe el Usuario");
    }
    console.log('Usuario encontrado.');
    return user.id;
})
.then(userid => {

    return models.quiz.bulkCreate(
        [ {question: '1+2', answer: '3', authorId: userid},
          {question: '3*2', answer: '6', authorId: userid},
          {question: '5-2', answer: '3', authorId: userid}
        ])
        .then(quizzes => {
            console.log('Creados varios quizzes.');
        });

})
.catch(error => {
    console.log("Error:", error);
});
```

- Busco un usuario por su nombre y le añado tres quizzes..

versión 2

© Santiago Pavón - UPM-DIT

versión: 3.3-crear_3_quizzes.js

```
const Sequelize = require("sequelize");

const {models} = require("../models");

Sequelize.Promise.all([ models.user.findOne( { where: {username: "Pepe"}} )
                        .then(user => {
                            if (!user) {
                                throw new Error("No existe el Usuario");
                            }
                            return user;
                        }),
                        models.quiz.bulkCreate([ {question: '1+2', answer: '3'},
                                                 {question: '3*2', answer: '6'},
                                                 {question: '5-2', answer: '3'}
                                               ])
])
.spread((user, quizzes) => {
    return user.addQuizzes(quizzes);
})
.then(quizzes => {
    console.log('Creados varios quizes.');
})
.catch(error => {
    console.log("Error:", error);
});
```

- Busco un usuario por su nombre y le añado tres quizzes.
- Soportado por bluebird:
  .spread((user,quizzes) => ....
  - es igual a
    .then(([user,quizzes]) => ....
  - es igual a
    .then(array => {
      const user = array[0];
      const quizzes = array[1]; ....

© Santiago Pavón - UPM-DIT

Buscar Todos los Quizzes

versión: 4.1-buscar_quizzes.js

```
const {models} = require("../models");

models.quiz.findAll()
.then(quizzes => {  // arg: array con todos los quizzes
    quizzes.forEach(quiz => {
        console.log("Autor:", quiz.authorId);
        console.log("  Preg:", quiz.question, "Resp:", quiz.answer);
    })
})
.catch(error => {
    console.log("Error:", error);
});
```

- Busco todos los quizzes existentes en la BBDD.
- Esto explota si hay muchos quizzes el la BBDD.

© Santiago Pavón - UPM-DIT

versión: 4.2-buscar_quizzes.js

```
const {models} = require("../models");

models.quiz.findAll()
.each(quiz => {  // Iterar por todos los quizzes. De uno en uno.
    console.log("Autor:", quiz.authorId);
    console.log("  Preg:", quiz.question, "Resp:", quiz.answer);
})
.catch(error => {
     console.log("Error:", error);
});
```

- Busco todos los quizzes existentes en la BBDD.
- Extension de bluebird:
  - Sustituir
    .then(quizzes => { quizzes.forEach(quiz => ....
  - por
    .each(quiz => ....

© Santiago Pavón - UPM-DIT

versión: 4.3-buscar_quizzes.js

```
const {models} = require("../models");

models.quiz.findAll()
.map(quiz => {  // Crea muchas Promesasa para
                // procesar los quizzes en paralelo.
        console.log("Autor:", quiz.authorId);
        console.log("  Preg:", quiz.question, "Resp:", quiz.answer);
     },
     { concurrency: 3 // Limita el numero maximo de promesas creadas para
     }                // que no haya mas de tres pendientes simultaneamente.
)
.catch(error => {
    console.log("Error:", error);
});
```

- Busco todos los quizzes existentes en la BBDD.
- Extension de bluebird:
  - Sustituir
    .then(quizzes => { quizzes.forEach(quiz => ....
  - por
    .map(quiz => ....

© Santiago Pavón - UPM-DIT

Quizzes con Nombre del Autor

versión: 5-quizzes_con_autor.js

```
const {models} = require("../models");

models.quiz.findAll({ order: [['updatedAt','DESC']],
                      include: [ { model: models.user,
                                   as: 'author' }
                               ]
})
.then(quizzes => {
    quizzes.forEach(quiz => {
        console.log("Preg:", quiz.question,
                    "Resp:", quiz.answer,
                    "By:", quiz.author.username);
    });
})
.catch(error => {
    console.log("Error:", error);
});
```

- Listar todos los quizzes existentes en la BBDD con el nombre del autor.
- Y ordenados descendientemente por fecha de modificación.

© Santiago Pavón - UPM-DIT

El 2 es Favorito de Todos

versión: 6-dos_favorito.js

```
const Sequelize = require("sequelize");
const Op = Sequelize.Op;

const {models} = require("../models");

models.quiz.findAll({ where: { question: { [Op.like]: '%2%' }}})
.then(quizzes => {

    return models.user.findAll()
    .then(users => {
          users.forEach(user => {
             return user.addFavouriteQuizzes(quizzes);
          });
    });
})
.catch(error => {
    console.log("Error:", error);
});
```

- Las preguntas que contienen un 2 se añaden favoritas de todo el mundo.

© Santiago Pavón - UPM-DIT

Crear con una Asociación

versión: 7-crear_con_una_asociacion.js

```
const {models} = require("../models");

models.user.create( { username: "Juan",
                      password: "1234",
                      quizzes: [ { question: "12345", answer: "6" },
                                 { question: "13579", answer: "11" },
                                 { question: "11235", answer: "8" }
                               ]
                    },
                    { include: [ models.quiz ]
                    }
                  )
.then(user => {
    console.log('Creado Usuario con varios quizzes.');
})
.catch(error => {
    console.log("Error:", error);
});
```

- Crear un usuario y con varios quizzes usando una asociación.

© Santiago Pavón - UPM-DIT

Crear con varias Asociaciones

versión: 8-crear_con_varias_asociaciones.js

```
const {models} = require("../models");

models.user.create( { username: "Juan",
                      password: "1234",
                      quizzes: [ { question: "12345",
                                   answer: "6"
                                 },
                                 { question: "13579",
                                   answer: "11",
                                   tips: [ {text: "Impares"} ]
                                 },
                                 { question: "11235",
                                   answer: "8",
                                   tips: [ {text: "Fibonacci"} ]
                                 }
                               ]
                    },
                    { include: [ { model: models.quiz,
                                   include: [ { model: models.tip } ]
                                 }
                               ]
                    }
                  )
.then(function(user) {
    console.log('Creado Usuario con varios quizzes y pistas.');
})
.catch(error => { console.log("Error:", error); });
```

- Crear un usuario y varios quizzes y comentarios usando varias asociaciones.

© Santiago Pavón - UPM-DIT

Listado Total

versión: 9.1-lista_todo.js

```
const {models} = require("../models");

models.user.findAll()
.then(users => {
     users.forEach(user => {
     console.log("User:", user.username);

     return models.quiz.findAll({where: {authorId: user.id}})
        .then(quizzes => {
            quizzes.forEach(quiz => {
                console.log("   * Preg:", quiz.question, "Resp:", quiz.answer);

                return models.tip.findAll({where: {quizId: quiz.id,
                                                   accepted: true}
                })
                .then(tips => {
                    tips.forEach(tip => {
                        console.log("       > ", tip.text);
                    });
                });
            });
        });
    });
})
.catch(error => {
    console.log("Error:", error);
});
```

- Se muestran todos los usuarios.
- Se muestran todos los quizzes.
- Se muestran solo las pistas con accepted igual a true.

© Santiago Pavón - UPM-DIT

versión: 9.2-lista_todo.js

- Se muestran todos los usuarios.
- Se muestran todos los quizzes.
- Se muestran solo las pistas con accepted igual a true.

```
const {models} = require("../models");

models.user.findAll()
.each(user => {
    console.log("User:", user.username);

    // return evita entrelazado
    return models.quiz.findAll({where: {authorId: user.id}})
    .each(quiz => {
        console.log("  * Preg:", quiz.question, "Resp:", quiz.answer);

        // return evita entrelazado
        return models.tip.findAll({where: {quizId: quiz.id,
                                           accepted: true}})
            .each(tip => {
                console.log("       > ", tip.text);
            })
    })
})
.catch(error => {
    console.log("Error:", error);
});

// Nota: .each devuelve promesa que se resuelve con el array original sin
modificar.
```

© Santiago Pavón - UPM-DIT

versión: 9.3-lista_todo.js

```
const {models} = require("../models");

models.user.findAll({ include: [ { model: models.quiz,
                                   include: [ models.tip ] } ]
                    })
.then(users => {
    users.forEach(user => {
     console.log("User:", user.username);

        user.quizzes.forEach(quiz => {
            console.log("  * Preg:", quiz.question, "Resp:", quiz.answer);

            quiz.tips.forEach(tip => {
                if (tip.accepted) {
                    console.log("       > ", tip.text);
                }
            });
        });
    });
})
.catch(error => {
    console.log("Error:", error);
});
```

- Se muestran todos los usuarios.
- Se muestran todos los quizzes.
- Se muestran solo las pistas con accepted igual a true.

© Santiago Pavón - UPM-DIT

```
const {models} = require("../models");

models.user.findAll({ include: [ { model: models.quiz,
                                   include: [ models.tip ] } ]
                    })
.each(user => {
    console.log("User:", user.username);

    user.quizzes.forEach(quiz => {
        console.log("  * Preg:", quiz.question, "Resp:", quiz.answer);

        quiz.tips.forEach(tip => {
            if (tip.accepted) {
                console.log("       > ", tip.text);
            }
        });
    });
})
.catch(error => {
    console.log("Error:", error);
});
```

- Se muestran todos los usuarios.
- Se muestran todos los quizzes.
- Se muestran solo las pistas con accepted igual a true.

© Santiago Pavón - UPM-DIT

Listar si hay Pistas Aceptadas

versión: 10-lista_no_aceptados.js

```
const {models} = require("../models");

models.user.findAll({ include: [{ model: models.quiz,
                                  include: [ { model: models.tip,
                                               where: {accepted: false} }]}]
                    })
.then(users => {
    users.forEach(user => {
     console.log("User:", user.username);

        user.quizzes.forEach(quiz => {
            console.log("  * Preg:", quiz.question, "Resp:", quiz.answer);

            quiz.tips.forEach(tip => {
                console.log("       > ", tip.text);
            });
        });
    });
})
.catch(error => {
    console.log("Error:", error);
});
```

- Se muestran las pistas no aceptadas (y sus quizzes) de todos los usuarios.

© Santiago Pavón - UPM-DIT

Listar Favoritos

versión: 11-favoritas_user.js

```
const {models} = require("../models");

models.user.find({ where: { username: "Juan" },
                   include: [ { model: models.quiz,
                                as: 'favouriteQuizzes' } ] })
.then(user => {

    if (!user) { throw new Error("No existe el Usuario");}

    user.favouriteQuizzes.forEach(quiz => {
        console.log("Preg:", quiz.question,
                    "Resp:", quiz.answer);
    });
})
.catch(error => {
    console.log("Error:", error);
});
```

- Muestra todos los favoritos de un usuario.

© Santiago Pavón - UPM-DIT

Busca o Crea Usuario

versión: 12-busca_o_crea_user.js

```
const {models} = require("../models");

models.user.findOrCreate({ where: { username: "Pepe" }    // Buscar
                          ,
                           defaults: { password: "1234" }    // Crear
                         })
.then(([user, created]) => {
    if (created) {
        console.log('El usuario no existia y se ha creado.');
    }
    console.log("UN:",user.username,"P:",user.password)
})
.catch(error => {
    console.log("Error:", error);
});
```

- Busca un usuario y si no existe lo crea.

© Santiago Pavón - UPM-DIT

Borrar

versión: 13-borrar_un_user.js

```
const {models} = require("../models");


models.user.find({ where: { username: "Pepe" } })
.then(user => {

    if (user) {
       return user.destroy();
    }
})
.then(() => {
    console.log("Ya no existe en la BBDD.");
})
.catch(error => {
    console.log("Error:", error);
});
```

- Borrar un usuario de la BBDD.

© Santiago Pavón - UPM-DIT

Ejemplos Pendientes

- Al método save se le puede pasar un array con nombres de campos.
    - Solo se salvarán en la tabla los campos indicados.
- Usar las opciones limit y offset en las llamadas a findAll para limitar e indicar los resultados obtenidos.
    - Útil para hacer paginación.
- Incrementar un campo de forma atómica.
- Transacciones.
- Ganchos.
- ...

© Santiago Pavón - UPM-DIT

© Santiago Pavón - UPM-DIT

Sequelize: Migraciones

- Sequelize proporciona un mecanismo llamado **migraciones** para transformar la base de datos.
- Cada migración es un fichero javascript donde se programan dos cosas:
  - Los cambios que hay que hacer en la base de datos para que evolucione y soporte una versión nueva de la aplicación.
    - Las evoluciones consisten en crear nuevas tablas, crear nuevos campos, cambiar campos ya existentes, etc.
  - Los cambios que hay que hacer para deshacer los cambios realizados en el punto anterior, y volver así al estado anterior.
- Estas tareas son las que realizan las funciones **up** y **down** de los ficheros de migración.

© Santiago Pavón - UPM-DIT

Comando: `sequelize`

- Primero hay que instalar el paquete **sequelize-cli**

```
$ npm install sequelize-cli
```

- Proporciona el comando **sequelize** para gestionar la aplicación de las migraciones.
  - Su path es `./node_modules/.bin/sequelize`
- Opciones:
  - **help** : Muestra ayuda.
  - **init** : Crea directorios y ficheros migrations, config y models.
  - **--env [entorno]** : Especificar el entorno. (defecto = **'development'**).
  - **db:migrate** : Ejecutar todas las migraciones pendientes.
  - **db:migrate:undo** : Deshacer las últimas migraciones aplicadas.
  - **migration:generate --name [nombre]** : Crea un fichero de migración llamado **'fecha'+'nombre'**.
  - . . .

© Santiago Pavón - UPM-DIT

Inicialización

- El primer comando que hay que ejecutar es:

```
$ ./node_modules/.bin/sequelize init
```

  - Crea el fichero de configuración **config/config.json** con los parámetros de acceso a la base de datos.
  - Crea el directorio **migrations** para guardar los ficheros de migración.
  - Crear **models/index.js**, para los modelos de datos.
    - Este paso fallará en nuestro caso porque ya hemos creado manualmente este fichero.
      - Nos quedaremos con nuestro **models/index.js**.
  - Crea el directorio **seeders** para guardar los ficheros de generación de datos.

- Existen otras opciones para inicializar solo una parte:
  - **init:config**, **init:migrations**, **init:models** e **init:seeders**.

© Santiago Pavón - UPM-DIT

Configurar el Acceso a la BBDD

- El comando **sequelize** necesita saber cómo se accede a la base de datos.
  - Por defecto, la configuración de acceso se lee del fichero **`config/config.json`**.
    - Puede usarse otro fichero con la opción **`--config`** de la línea de commandos
  - También puede pasarse con la opción **`--url`** indicando la URL completa de acceso.

```
$ sequelize db:migrate
    --url sqlite:///path_al_proyecto/quiz.sqlite
```

    - Nota: Hay que indicar el path completo en la URL.

© Santiago Pavón - UPM-DIT

Usar Migraciones

- Los ficheros de migración se crean con el comando

```
$ sequelize migration:generate --name nombre_de_la_migración
```

  - Se genera un fichero plantilla que debemos editar.

- Para aplicar la migración ejecutaremos:

```
$ sequelize db:migrate
```

  - Se aplican todas las migraciones pendientes de ser aplicadas.

- Para deshacer migraciones:

```
$ sequelize db:migrate:undo
```

  - Se deshacen todas las migraciones que se aplicaron juntas al ejecutar el comando sequelize db:migrate.

- NO OLVIDAR ELIMINAR LA SENTENCIA **sequelize.sync()**
  - Al usar migraciones hay que eliminar la sentencia **sequelize.sync()** de **models/index.js**.

© Santiago Pavón - UPM-DIT

Migración para Crear Tabla **Users**

- Crear la migración que crea la tabla **users**:

```
$ npx sequelize migration:generate
                  --name CreateUsersTable
```

  - Se crea el fichero:

```
migrations/20190228140322-CreateUsersTable.js
```

    - Lo editamos para programar las modificaciones que queremos realizar en la base de datos.

- Para aplicar la migración ejecutamos:

```
$ npx sequelize db:migrate
             --url sqlite://$(pwd)/quiz.sqlite
```

© Santiago Pavón - UPM-DIT

```
'use strict';

module.exports = {
  up: function (queryInterface, Sequelize) {
    /*
      Add altering commands here.
      Return a promise to correctly handle asynchronicity.

      Example:
      return queryInterface.createTable('users', { id: Sequelize.INTEGER });
    */
  },

  down: function (queryInterface, Sequelize) {
    /*
      Add reverting commands here.
      Return a promise to correctly handle asynchronicity.

      Example:
      return queryInterface.dropTable('users');
    */
  }
};
```

Original: 20190228140322-CreateUsersTable.js

© Santiago Pavón - UPM-DIT

Hay que especificar todos los campos de la BBDD.

```
'use strict';

module.exports = {
  up: function (queryInterface, Sequelize) {

      return queryInterface.createTable(
           'users',
           { id:         { type: Sequelize.INTEGER,  allowNull: false,
                           primaryKey: true,         autoIncrement: true,
                           unique: true },
             username:   { type: Sequelize.STRING,   unique: true,
                           validate: { notEmpty: {msg: "Falta username"} } },
             password:   { type: Sequelize.STRING,
                           validate: { notEmpty: {msg: "Falta password"} } },
             salt:       { type: Sequelize.STRING },
             isAdmin:    { type: Sequelize.BOOLEAN,  defaultValue: false },
             createdAt:  { type: Sequelize.DATE,     allowNull: false },
             updatedAt:  { type: Sequelize.DATE,     allowNull: false }
           },
           { sync: {force: true}
           }
      );
  },

  down: function (queryInterface, Sequelize) {
      return queryInterface.dropTable('users');
  }
};
```

Editado: 20190228140322-CreateUsersTable.js

© Santiago Pavón - UPM-DIT

Migración para Crear Tabla **`Quizzes`**

- Crear la migración que crea la tabla **`quizzes`**:

```
$ node_modules/.bin/sequelize migration:generate
                --name CreateQuizzesTable
```

- Se crea el fichero:

```
migrations/20190228140344-CreateQuizzesTable.js
```

- Lo editamos para programar las modificaciones que queremos realizar en la base de datos.

- Para aplicar la migración ahora (podemos esperar) ejecutamos:

```
$ npx sequelize db:migrate
               --url sqlite://$(pwd)/quiz.sqlite
```

© Santiago Pavón - UPM-DIT

```
'use strict';

module.exports = {
  up: function (queryInterface, Sequelize) {

      return queryInterface.createTable(
           'quizzes',
           { id:         { type: Sequelize.INTEGER,  allowNull: false,
                           primaryKey: true,         autoIncrement: true,
                           unique: true },
             question:   { type: Sequelize.STRING,
                           validate: { notEmpty: {msg: "Falta Pregunta"} } },
             answer:     { type: Sequelize.STRING,
                           validate: { notEmpty: {msg: "Falta Respuesta"} } },
             createdAt: { type: Sequelize.DATE,     allowNull: false },
             updatedAt: { type: Sequelize.DATE,     allowNull: false }
           },
           { sync: {force: true}
           }
      );
  },

  down: function (queryInterface, Sequelize) {
      return queryInterface.dropTable('quizzes');
  }
};
```

Hay que especificar todos los campos de la BBDD.

Editado: 20190228140344-CreateQuizzesTable.js

© Santiago Pavón - UPM-DIT

Migración para Crear Tabla **`Tips`**

- Crear la migración que crea la tabla **`tips`**:

```
$ node_modules/.bin/sequelize migration:generate
                --name CreateTipsTable
```

  - Se crea el fichero:

```
migrations/20190228140358-CreateTipsTable.js
```

    - Lo editamos para programar las modificaciones que queremos realizar en la base de datos.

- Para aplicar la migración ahora (podemos esperar) ejecutamos:

```
$ npx sequelize db:migrate
             --url sqlite://$(pwd)/quiz.sqlite
```

© Santiago Pavón - UPM-DIT

Hay que especificar todos los campos de la BBDD.

```
'use strict';

module.exports = {
  up: function (queryInterface, Sequelize) {

      return queryInterface.createTable(
           'tips',
           { id:         { type: Sequelize.INTEGER,  allowNull: false,
                           primaryKey: true,         autoIncrement: true,
                           unique: true },
             text:       { type: Sequelize.STRING,   unique: true,
                           validate: { notEmpty: {msg: "Falta pista"} } },
             accepted:   { type: Sequelize.BOOLEAN,  defaultValue: false },
             createdAt: { type: Sequelize.DATE,     allowNull: false },
             updatedAt: { type: Sequelize.DATE,     allowNull: false }
           },
           { sync: {force: true}
           }
      );
  },

  down: function (queryInterface, Sequelize) {
      return queryInterface.dropTable('tips');
  }
};
```

Editado: 20190228140358-CreateTipsTable.js

© Santiago Pavón - UPM-DIT

Migración para Crear Claves Externas

- Crear la migración que crea las claves externas que implementan las relaciones 1 a N entre las tablas:

```
$ npx sequelize migration:generate
                --name CreateForeignKeys
```

  - Se crea el fichero:

    `migrations/20190228140947-CreateForeignKeys.js`

    - Lo editamos para programar las modificaciones que queremos realizar en la base de datos.

- Para aplicar la migración ahora (podemos esperar) ejecutamos:

```
$ npx sequelize db:migrate
             --url sqlite://$(pwd)/quiz.sqlite
```

© Santiago Pavón - UPM-DIT

```
'use strict';

module.exports = {
  up: function (queryInterface, Sequelize) {

      return Sequelize.Promise.all([
          queryInterface.addColumn('tips',
                                   'quizId',
                                   { type: Sequelize.INTEGER }
                                  ),
          queryInterface.addColumn('quizzes',
                                   'authorId',
                                   { type: Sequelize.INTEGER }
                                  )
     ]);
  },

  down: function (queryInterface, Sequelize) {
      return Sequelize.Promise.all([
          queryInterface.removeColumn('tips','quizId'),
          queryInterface.removeColumn('quizzes','authorid')
     ]);
  }
};
```

Editado: 20190228140947-CreateForeignKeys.js

© Santiago Pavón - UPM-DIT

Migración para Crear Tabla **favourites**

- Crear la migración que crea la tabla join **favourites**:

```
$ npx sequelize migration:generate
                --name CreateFavouritesTable
```

  - Se crea el fichero:

```
migrations/20190228141254-CreateFavouritesTable.js
```

    - Lo editamos para programar las modificaciones que queremos realizar en la base de datos.

- Para aplicar esta migración y las que estén pendientes ejecutamos:

```
$ npx sequelize db:migrate
              --url sqlite://$(pwd)/quiz.sqlite
```

© Santiago Pavón - UPM-DIT

```
'use strict';

module.exports = {
  up: function (queryInterface, Sequelize) {

      return queryInterface.createTable(
           'favourites',
           { QuizId:    { type: Sequelize.INTEGER,  allowNull: false,
                           primaryKey: true, unique: "compositeKey" },
             UserId:    { type: Sequelize.INTEGER,  allowNull: false,
                           primaryKey: true, unique: "compositeKey" },
             createdAt: { type: Sequelize.DATE,     allowNull: false },
             updatedAt: { type: Sequelize.DATE,     allowNull: false }
           },
           { sync: {force: true}
           }
      );
  },

  down: function (queryInterface, Sequelize) {
      return queryInterface.dropTable('favourites');
  }
};
```

Editado: 20190228141254-CreateFavouritesTable.js

© Santiago Pavón - UPM-DIT

Métodos para cambiar el Esquema de la BBDD

- **createTable**(tableName, attributes, options)
- **dropTable**(tableName)
- **dropAllTables**()
- **renameTable**(before, after)
- **showAllTables**()
- **describeTable**(tableName)
- **addColumn**(tableName, attributeName, dataTypeOrOptions)
- **removeColumn**(tableName, attributeName)
- **changeColumn**(tableName, attributeName, dataTypeOrOptions)
- **renameColumn**(tableName, attrNameBefore, attrNameAfter)
- **addIndex**(tableName, attributes, options)
- **removeIndex**(tableName, indexNameOrAttributes)

© Santiago Pavón - UPM-DIT

© Santiago Pavón - UPM-DIT

Comandos

- Ayuda:

```
$ npx sequelize help
$ npx sequelize help:tarea
```

- Crear un nuevo fichero seed:

```
$ npx sequelize seed:generate --name filename
```

- Ejecutar el seed especificado:

```
$ npx sequelize db:seed --seed filename
```

- Ejecutar todos los seed:

```
$ npx sequelize db:seed:all
```

- Deshacer el seed especificado:

```
$ npx sequelize db:seed:undo --seed filename
```

- Deshacer todos los seed:

```
$ npx sequelize db:seed:undo:all
```

© Santiago Pavón - UPM-DIT

Más Detalles

- Consultar en el capítulo de migraciones:
  - Cómo se inicializa sequelize.
  - Cómo se configura el acceso a la base de datos.
  - . . .

© Santiago Pavón - UPM-DIT

© Santiago Pavón - UPM-DIT