Proyecto de la asignatura CORE

Desarrollo de un Blog

Tema 9: Migraciones.

CORE 2013-2014

ver: 2014-05-17

Índice

- Migraciones
- Integración en el proyecto.
- Documentación y tutoriales de Cloudinary

  `http://sequelizejs.com`

- El ejemplo está disponible en la rama tema9.

  `http://github.com/CORE-UPM/blog_2014`

Sequelize: Migraciones

- Sequelize proporciona un mecanismo llamado migraciones para transformar la base de datos.
  - Permiten crear nuevas tablas, borrarlas, crear nuevas columnas, cambiar su tipo, etc.
- Estas transformaciones se escriben en ficheros de migración, especificando:
  - cómo se realizan las transformaciones para pasar a un nuevo estado,
  - y cómo se deshacen para volver al estado anterior.
- Usando funciones javascript.

Comando: `sequelize`

- El paquete **Sequelize** también proporciona el comando **sequelize** para gestionar la aplicación de las migraciones.
  - Su path es **`./node_modules/sequelize/bin/sequelize`**
- Opciones:
  - **-h** : Muestra ayuda.
  - **-i** : Crea los directorios migrations y config.
  - **-e [entorno]** : Especificar el entorno. (defecto = **'development'**).
  - **-m** : Ejecutar todas las migraciones pendientes.
  - **-m -u** : Deshacer las últimas migraciones aplicadas (con -m).
  - **-c [nombre]** : Crea un fichero de migración llamado **'fecha'+'nombre'**.
  - . . .

Inicialización

- El primer comando que hay que ejecutar es:

```
$ sequelize -i
```

  - Crea el fichero de configuración **`config/config.json`** con los parámetros de acceso a la base de datos.
  - Crea el directorio **`migrations`** donde se guardarán los ficheros de migración que crearemos en un futuro.

config/config.json

- El fichero de configuración generado por defecto tiene este contenido:
  - Contiene secciones para varios entorno.

Nota: El entorno de ejecución se especifica asignando el valor adecuado a la variable de entorno ***NODE_ENV****.*

```
{
  "development": {
    "username": "root",
    "password": null,
    "database": "database_development",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": null,
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": null,
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

- En CoreBlog crearemos un fichero de configuración **config/config.json** sin secciones para varios entornos, y con el siguiente contenido:

```
{
    "dialect":  "DATABASE_DIALECT",
    "protocol": "DATABASE_PROTOCOL",
    "username": "DATABASE_USER",
    "password": "DATABASE_PASSWORD",
    "database": "DATABASE_NAME",
    "host":     "DATABASE_HOST",
    "port":     "DATABASE_PORT",
    "omitNull": true,
    "storage":  "blog.sqlite"
}
```

  - Hay que sustituir los valores en **morado** por los valores reales que use cada alumno en su propio proyecto.

- Este fichero no debe meterse en git. (*añadirle a* ***.gitignore***)
  - Cada alumno tendrá valores diferentes.
  - Su contenido es distinto para ejecutar en local y en Heroku.
  - Por motivos de seguridad.

- Crearemos un script para crear este fichero automáticamente: **config/mkconfig.js**
  - Este script lo meteremos en git.
  - Hay que ejecutarlo antes de aplicar migraciones, tanto en local, como en Heroku.

config/mkconfig.js

```
#!/usr/bin/env node

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

// Ruta al fichero de configuracion
var config_path = path.dirname(process.argv[1]) + '/config.json';

if (! process.env.DATABASE_URL) {
    console.log('ERROR: No existe la variable de entorno DATABASE_URL.')
    console.log('En local, ejecute: "foreman run node config/mkconfig.js"');
    process.exit(1);
}

var vals = process.env.DATABASE_URL.match(/(.*)\:\/\/(.*?)\:(.*)@(.*)\:(.*)\/(.*)/);

var config_value = {
    "dialect":  vals[1],
    "protocol": vals[1],
    "username": vals[2],
    "password": vals[3],
    "host":     vals[4],
    "port":     vals[5],
    "database": vals[6],
    "omitNull": true,
    "storage":  process.env.DATABASE_STORAGE
};

// Crear fichero de configuracion
fs.writeFileSync(config_path, JSON.stringify(config_value,null,2));
```

- Para crear **config/config.json** en la máquina local de desarrollo:

```
$ foreman run node config/mkconfig.js
```

- Para crear **config/config.json** en la máquina de Heroku:

```
$ heroku run bash
(estoy en heroku)$ node config/mkconfig.js
(estoy en heroku)$ otros comandos
(estoy en heroku)$ exit
$
```

Notas

Recordatorio:
En la máquina local la configuración para acceder a la base de datos está guardada en el fichero .env.
En Heroku la configuración está guardada en la variable de entorno DATABASE_URL.

Migraciones

- Cada migración es un fichero javascript donde se programan dos tareas:
  - Los cambios hay que hacer en la base de datos para que evolucione y soporte una versión nueva de la aplicación. Normalmente, las evoluciones consisten en crear nuevas tablas, o campos en las tablas ya existentes.
  - Los cambios 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 asignadas a los atributos **up** y **down** de los ficheros de migración.
- Los ficheros de migración se crean con el comando

```
$ sequelize -c nombre_de_la_migración
```

  - Se genera un fichero plantilla que debemos editar.
- Para aplicar la migración ejecutaremos:

```
$ sequelize -m
```

  - En realidad se aplican todas las migraciones pendientes de ser aplicadas.

Notas

Sobre la aplicación de migraciones:

- Para aplicar todas las migraciones pendientes: **`sequelize -m`**
- Para deshacer migraciones: **`sequelize -m -u`**
  - Se deshacen todas las migraciones que se aplicaron juntas al ejecutar el comando `sequelize -m`

Las funciones asignadas a los atributos **up** y **down** tienen tres argumentos:

- **migration**
  - Permite acceder a las funciones que modifican la base de datos (createTable, dropTable, etc...).
- **DataTypes**
  - Acceso a los tipos para los campos de las tablas de la base de datos.
- **done**
  - Las funciones que aplican o deshacen la migración son asíncronas, y **done** apunta a una función que hay que llamar para indicar que ya se ha aplicado o deshecho la migración.

NO OLVIDAR ELIMINAR LA SENTENCIA **sequelize.sync()**

- Al usar migraciones, estas gestionan la creación de las tablas, luego hay que eliminar la sentencia **sequelize.sync()** del fichero **models/index.js**.

Funciones

- **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)

Migraciones de cada Tema

Borrar sequelize.sync()

- Ahora la creación de las tablas la gestionaremos con migraciones.
- Por tanto, hay que eliminar del fichero **models/index.js** la sentencia:

```
sequelize.sync()
```

Tema 2: Crear tabla **`Posts`**

- Migración para crear la tabla **`Posts`**:
- Ejecutar:

```
$ ./node_modules/.bin/sequelize -c CreatePostsTable
```

  - Se crea el fichero:

    `migrations/20140517140802-CreatePostsTable.js`

  - Lo editamos dejando su contenido como se muestra en la siguiente transparencia.

  - Y meterlo en git.

- Para aplicar la migración en local ejecutamos:

```
$ foreman run node config/mkconfig.js
$ ./node_modules/sequelize/bin/sequelize -m
```

- Para aplicar la migración en local ejecutamos:

```
(local)  $ heroku run bash
         Running `bash` attached to terminal... up, run.6006
(heroku) $ node mkconfig.js
(heroku) $ cat config/config.json
         { "dialect": "postgres",
           "protocol": "postgres",
           "username": "aaaaaaaaaaaa",
           "password": "bbbbbbbbbbbb",
           "database": "ccccccccccc",
           "host": "ddddddd.amazonaws.com",
           "port": "5432",
           "omitNull": true,
           "storage": null
         }
(heroku) $ ./node_modules/sequelize/bin/sequelize -m
         Executing migration: 20140517140802-CreatePostsTable.js
         Executed migration: 20140517140802-CreatePostsTable.js
(heroku) $ exit

(local)  $
```

```
module.exports = {
  up: function(migration, DataTypes, done) {
      migration.createTable(
            'Posts', {  id: { type: DataTypes.INTEGER,
                              allowNull: false,
                              primaryKey: true,
                              autoIncrement: true,
                              unique: true },
                     title: { type: DataTypes.STRING,
                              allowNull: false,
                              defaultValue:'Título del Posts' },
                      body: { type: DataTypes.TEXT,
                              allowNull: false },
                 createdAt: { type: DataTypes.DATE,
                              allowNull: false },
                 updatedAt: { type: DataTypes.DATE,
                              allowNull: false }
            },
            { sync: {force:true} })
      .complete(done);
  },
  down: function(migration, DataTypes, done) {
       migration.dropTable('Posts')
                .complete(done);
  }
}
```

Crear la tabla.

Subir a la siguiente versión.

Retroceder a la versión anterior.

Fuerza a que se cree de nuevo la tabla.

Migración terminada.

Destruir la tabla.

20140517140802-CreatePostsTable.js

Tema 4: Crear tabla **`Users`**

- Migración para crear la tabla **`Users`**:
- Ejecutar:

```
$ ./node_modules/.bin/sequelize -c CreateUsersTable
```

- Se crea el fichero:

  `migrations/20140517150611-CreateUsersTable.js`

- Lo editamos dejando su contenido como se muestra en la siguiente transparencia.

- Y meterlo en git.

- Para aplicar la migración en local ejecutamos:

```
$ foreman run node config/mkconfig.js

$ ./node_modules/sequelize/bin/sequelize -m
```

- Para aplicar la migración en local ejecutamos:

```
(local)  $ heroku run bash
(heroku) $ node mkconfig.js
(heroku) $ ./node_modules/sequelize/bin/sequelize -m
(heroku) $ exit
```

```
module.exports = {
  up: function(migration, DataTypes, done) {

     migration.createTable(
          'Users',
          {  id: {
                  type: DataTypes.INTEGER,
                  allowNull: false,
                  primaryKey: true,
                  autoIncrement: true,
                  unique: true
             },
             login: {
                  type: DataTypes.STRING,
                  notEmpty: true,
                  unique: true
             },
             name: {
                  type: DataTypes.STRING,
                  allowNull: false,
                  defaultValue:'John Smith'
             },
             email: {
                  type: DataTypes.STRING,
                  notEmpty: true
             },
             hashed_password: {
                  type: DataTypes.STRING,
                  notEmpty: true,
                  allowNull: false
             },
             salt: {
                  type: DataTypes.STRING,
                  notEmpty: true,
                  allowNull: false
             };
             createdAt: {
                  type: DataTypes.DATE,
                  allowNull: false
             },
             updatedAt: {
                  type: DataTypes.DATE,
                  allowNull: false
             }
          },
          { sync: {force:true}
          })
        .complete(done);
  },

  down:

     function(migration,DataTypes,done) {
          migration.dropTable('Users')
                .complete(done);
     }
}
```

20140517150611-CreateUsersTable.js

Tema 6: Añadir **AuthorId** a **Posts**

- Migración para crear el campo **AuthorId** en la tabla **`Posts`**:
- Ejecutar:

```
$ ./node_modules/.bin/sequelize -c AddAuthorIdToPostsTable
```

  - Se crea el fichero:

    `migrations/20140517151251-AddAuthorIdToPostsTable.js`

  - Lo editamos dejando su contenido como se muestra en la siguiente transparencia.

  - Y meterlo en git.

- Para aplicar la migración en local ejecutamos:

```
$ foreman run node config/mkconfig.js

$ ./node_modules/sequelize/bin/sequelize -m
```

- Para aplicar la migración en local ejecutamos:

```
(local)  $ heroku run bash
(heroku) $ node mkconfig.js
(heroku) $ ./node_modules/sequelize/bin/sequelize -m
(heroku) $ exit
```

```
module.exports = {
  up: function(migration, DataTypes, done) {

        migration.addColumn('Posts',
                            'AuthorId',
                            DataTypes.INTEGER
                           )
             .complete(done);
  },
  down: function(migration, DataTypes, done) {

        migration.removeColumn('Posts', 'AuthorId')
             .complete(done);
  }
}
```

20140517151251-AddAuthorIdToPostsTable.js

Tema 7: Crear tabla **Comments**

- Migración para crear la tabla **Comments**:
- Ejecutar:

```
$ ./node_modules/.bin/sequelize -c CreateCommentsTable
```

- Se crea el fichero:

  migrations/20140517154521-CreateCommentsTable.js

- Lo editamos dejando su contenido como se muestra en la siguiente transparencia.

- Y meterlo en git.

- Para aplicar la migración en local ejecutamos:

```
$ foreman run node config/mkconfig.js

$ ./node_modules/sequelize/bin/sequelize -m
```

- Para aplicar la migración en local ejecutamos:

```
(local)  $ heroku run bash
(heroku) $ node mkconfig.js
(heroku) $ ./node_modules/sequelize/bin/sequelize -m
(heroku) $ exit
```

```
module.exports = {
  up: function(migration, DataTypes, done) {

    migration.createTable(
        'Comments',
        {
            id: {
                type: DataTypes.INTEGER,
                allowNull: false,
                primaryKey: true,
                autoIncrement: true,
                unique: true
            },
            AuthorId: {
                type: DataTypes.INTEGER,
                allowNull: false
            },
            PostId: {
                type: DataTypes.INTEGER,
                allowNull: false
            },
            body: {
                type: DataTypes.TEXT,
                allowNull: false
            },
            createdAt: {
                type: DataTypes.DATE,
                allowNull: false
            },
            updatedAt: {
                type: DataTypes.DATE,
                allowNull: false
            }
        },
        { sync: {force:true}
        })
     .complete(done);

  },
  down: function(migration, DataTypes, done) {

     migration.dropTable('Comments')
         .complete(done);

  }
}
```

20140517154521-CreateCommentsTable.js

Tema 8: Crear tabla **`Attachments`**

- Migración para crear la tabla **`Attachments`**:
- Ejecutar:

```
$ ./node_modules/.bin/sequelize -c CreateAttachmentsTable
```

  - Se crea el fichero:

    `migrations/20140517155153-CreateAttachmentsTable.js`

  - Lo editamos dejando su contenido como se muestra en la siguiente transparencia.

  - Y meterlo en git.

- Para aplicar la migración en local ejecutamos:

```
$ foreman run node config/mkconfig.js

$ ./node_modules/sequelize/bin/sequelize -m
```

- Para aplicar la migración en local ejecutamos:

```
(local)  $ heroku run bash
(heroku) $ node mkconfig.js
(heroku) $ ./node_modules/sequelize/bin/sequelize -m
(heroku) $ exit
```

```
module.exports = {
  up: function(migration, DataTypes, done) {

    migration.createTable(
        'Attachments', {
            id: {
                type: DataTypes.INTEGER,
                allowNull: false,
                primaryKey: true,
                autoIncrement: true,
                unique: true
            },
            PostId: {
                type: DataTypes.INTEGER,
                allowNull: false
            },
            public_id: {
                type: DataTypes.STRING,
                allowNull: false
            },
            url: {
                type: DataTypes.STRING,
                allowNull: false
            },
            filename: {
                type: DataTypes.STRING,
                allowNull: false
            },
            mime: {
                type: DataTypes.STRING,
                allowNull: false
            },
            createdAt: {
                type: DataTypes.DATE,
                allowNull: false
            },
            updatedAt: {
                type: DataTypes.DATE,
                allowNull: false
            }
        },
        { sync: {force:true}
        })
      .complete(done);

  },
  down: function(migration, DataTypes, done) {

    migration.dropTable('Attachments')
          .complete(done);
  }
}
```

20140517155153-CreateAttachmentsTable.js