Módulo del proyecto DAW

David Jiménez González

Objetivos

Desarrollar una plataforma para la gestión de información. El sistema será capaz de almacenar, clasificar y gestionar dicha información. Para que esto sea posible, la información necesitará algún tipo de estructura.

La primera separación lógica de información será la biblioteca. Cada usuario será dueño de una librería. Esto no es mas que una referencia a un documento en la base de datos.

Cada librería contendrá una serie de libros. Cada libro será una estructura de datos definida por el usuario. Para ello cada libro contendrá una serie de campos.

Los campos definen el esquema básico del libro. El usuario podrá decidir que campos contiene cada libro. Para que la información tenga una estructura y se pueda gestionar los posibles campos estarán limitados a unos tipos definidos previamente.

Los tipos básicos de los campos serán: Number, String, Date, Boolean, Array, File y Libro.

Los tipos básicos solo podrán contener sus correspondientes valores. En cambio los tipos file podrán almacenar cualquier tipo de fichero, véase audio, imagen, video…

Los campos de tipo libro serán una referencia hacia otro libro creado previamente o a un campo especifico de este.

Para contener los datos de cada libro, estos contaran con paginas. Cada una de estas guarda un registro definido por los campos.

Recursos hardware y software

Para la realización de este proyecto utilizaremos, a nivel de hardware, un Mac Book Pro.

A nivel de software tenemos:

DB: Para la gestión de los datos usaremos un servidor MongoDb. Para la comunicación de este con el servidor principal de la aplicación usaremos una librería para NodeJs llamada Moongose. Moongose nos provee de un modelo, definido por un schema, que es el encargado de interactuar con la DB y ejecutar las querys.

Servidor: Dado que el servidor tendrá una arquitectura REST y toda la aplicación será realizada en javascript, el servidor será NodeJs. Para la gestión de las rutas usaremos la librería ExpressJs.

Una serie de librerías serán añadidas para la gestión de algunas partes del servidor. Para el manejo de sesiones utilizaremos ExpressSession, una librería que añade esto al modulo de express que ya poseemos.

Para validar las peticiones de usuario usaremos tokens basados en Auth0. La librería encargada de gestionar dichos token será JsonWebToken.

Cors, BodyParser y Multer se encargaran de parsear las peticiones que lleguen al servidor a una estructura json para su futuro procesamiento.

Transpilado: Dado que los navegadores actuales no soportan es6 en adelante, el uso de un transpilador resulta esencial. Para este propósito utilizaremos Babelify, una librería basada en Babel, que se integra con Browserify, y nos gestiona todas las dependencias en el código dando como resultado un solo archivo final en es5, mitificado y preparado para correr en cualquier navegador.

Gestión de Dependencias: Para este punto usaremos Browserify, una librería que se encarga de gestionar los import, require.. el la aplicación cliente.

Gestor de Tareas: Este punto será gestionado por gulp. Gracias a NPM podemos correrlo en el servidor y programar una serie de tareas automatizadas. Por una parte tendremos un comando que se encargara de llamar a Browserify, y gracias a Babelify, compilar el código en tiempo de ejecución para hacer mas fácil el desarrollo. Programaremos también otro tipo de tares tales como poblar la db, pasar de un ambiente a otro (producción a desarrollo y viceversa), así como el lanzamiento del servidor.

Cliente: El cliente será una Single Page App. Para la gestión de las rutas usaremos una librería llamada Page. Esta librería estructura las rutas de la misma forma que express, pero en el cliente. También provee una serie de funcionalidades, tales como midelwares, re direcciones, window state…

Toda la app estará diseñada con el patrón flux. Para integrarlo utilizaremos Redux, una librería que nos provee de un state global para toda la aplicación, facilitando así la gestión de toda la información.

La vista estará definida por componentes. Para ello usaremos ReactJs, una librería para crearlos y gestionarlos.

Para la integración del state global con los componentes usaremos ReacRedux, una librería que se encarga de comunicar el state global con el state de cada componente, convirtiendo estos en reactivos.

Para facilitar el manejo del state global contamos con ImmutableJs, una librería para crear estructura de datos inmutables haciendo así que cada cambien en el state se refleje en la vista haciendo que este vuelva a renderizarse.

Para dar estilos a la aplicación usaremos Materialize, una librería basada en Material Design propuesto por google. Esta librería nos provee tanto un sistema de rejilla parecido al de Boostrap, así como una serie de funcionalidad tales como tarjetas, modal, buttons, inputs… definidos previamente.

Desarrollo: La aplicación utilizada para escribir el código será SublimeText 3. Para la monitorización usaremos NodeLogg en el servidor. En el cliente instalaremos una serie de plugins en el navegador. React para la monitorizacion de los componentes. Redux Logre para monitorizar el state global de la aplicación.

Fases

Diseño

La aplicación estará distribuida en 2 servidores, uno servidor rest puro y un servidor encargado de servir la aplicación cliente y todas sus depencias.

El servidor rest contara con un router provisto por Express. Aquí será donde se definirán todas las rutas encargadas de la gestión de la información. Tendremos 2 rutas principales: /users y /librari.

La primera se encargara de gestionar todo lo relacionado con los usuarios de la aplicación. Véase creación, modificación, roles, login…

La segunda se encargara de gestionar las bibliotecas de los usuarios. Creación, borrado, actualización…

En niveles inferiores de esta se definirán las rutas encargadas de gestionar los libros, los campos y las paginas de cada libro. Definiremos 3 sub-rutas:

/library/book: En esta ruta se definirán las rutas encargadas de crear, borrar, editar y leer los libros.

/library/book/fields: En esta ruta se definirán las rutas encargadas de crear, borrar, editar y leer los campos.

/library/book/page: En esta ruta se definirán las rutas encargadas de crear, borrar, editar y leer las páginas.

La base de datos estará distribuida en 2 tipos de documentos. Unos definirá los usuarios (Users) y otro definirá las bibliotecas (Librarys).

La relación entre ellos será atreves de relaciones débiles mediante el ObjectId del usuario en su librería.

El schema de cada documento será el siguiente:

Users:

```
{
        name: {
                type: String,
                required: true
        },
        secondName: {
                type: String
        },
        password: {
                type: String,
                required: true,
                set: require(__root + 'middleware/SerializePassword')
        },
        email: {
                type: String,
                required: true
        },
        createdAt: {
                type: Date,
                default: Date.now,
        },
        role: {
                type: String,
                default: 'user'
        }
}
```

Librarys:

```
{
    user: {
        type: ObjectId,
        required: true
    },
    name: {
        type: String,
        required: true
    },
    createdAt: {
        type: Date,
        default: Date.now,
    },
    books: {
        type: Array,
        default: { [
            fields: {
                type: Array,
                default: { [
                    name: {
                        type: String,
                        required: true
                    },
                    type: {
                        type: String
                        required: true
                    }
                ] }
            },
            pages: {
                type: Array,
                default: {
                    field: {
                        type: null,
                        value: null
                    }
                }
            } ]
        }
```

El servidor del cliente tendrá varios propósitos. Por una parte será el encargado de servir un archivo llamado index.html en cualquier llamada recibida sin importar la ruta solicitada. Se encargara también de redireccionar al raíz cualquier petición recibida. Por otra parte gestionara todo el contenido estático de la aplicación, haciendo este accesible para todas las llamadas dirigidas a /public/*.

El cliente contara con un archivo llamado app.js que será el encargado de montar la aplicación. En este archivo se definirá el componente raíz, que contendrá tanto el state global, como el router, las llamadas ajax… Este será un buen punto para añadir los reducers de Redux, que se encargaran de gestionar el state y sus cambios.

En un nivel inferior se definira un archivo llamado Router.js. Este archivo será el componente encargado de la navegación en la página. También será el encargado de gestionar los permisos de acceso a ciertas rutas y controlar las re direcciones.

Dentro del componente anteriormente mencionado, se situaran 4 componentes.

Landing: Index de la aplicación.

Register: Registro de usuarios.

Login: Inicio de sesión.

Home: Gestion de la libreria de cada usuario.

Estos componentes irán en sus respectivas rutas, a excepción de home que contara con una segunda ruta que englobara todas las peticiones inferiores a el (/home/*).

Definiremos una ultima ruta (/*) a la cual serán redirigidas todas aquellas peticiones que no puedan ser procesadas.

Dentro de el componente Home definiremos una serie de componentes para la gestión de la librería.

Desarrollo

Deploy

Desarrollo de las fases

Preguntas relacionadas con el módulo de FCT

Bases de datos

¿Se utiliza Access en la empresa? Si es así ¿qué tipo de tareas se realizan con dicho software?

Si, pero solo Excel para general informes.

¿Qué otros SGBD relacionales utilizan? ¿qué tipo de tareas se realizan?

Sql Lite, para almacenamiento temporal de información antes de ser transmitida a Eslastic Search.

Construyes

Sistemas Operativos

¿Qué Sistemas Operativos utilizan (Windows, Linux, otros)? Indica también la versión y/o distribución.

Windows 10 y Linux bajo una distribución de Ubunto.

¿Qué tareas relacionadas con los Sistemas Operativos has realizado?

Gestión de los servidores.

Has configurado algún servicio de red ¿Cuál?

No

Lenguajes de marcas

¿Utilizan XML (XML Schema, XPath, XQuery?

Si

¿Actualizan la página web de la empresa con alguna base de datos utilizando XML?

No

Programación en entorno servidor (PHP, JSP, ASP .NET…)

¿Qué entorno de desarrollo se utiliza?: Visual Studio, NetBeans, Eclipse,…

Eclipse y Visual Estudio

En caso de emplear Visual Studio utilizan el patrón de diseño MVC o Web Forms? Y ¿qué lenguaje de programación?: PHP, Visual Basic, C#, Java….

Java y Visual Basic.

En el caso de programar en PHP, ¿realizan POO? ¿Con algún Framework: Laravel, Symfony, Yii, Zend..?

Solo se maneja java para el servidor.

¿Se trabaja con o sobre algún gestor de contenidos tipo Wordpress (desarrollo de plugins o de temas), Prestashop…..como usuario, administrador o desarrollador?

Se utiliza RedMine para la gestión de incidencias y proyectos con plugins reinstalados.

¿Qué servidores utilizan: web, FTP, DNS,…?

Web, DNS y FTP

¿Qué servidores de aplicaciones utilizan?

Programación en entorno cliente

¿Utilizan JavaScript? Con qué entorno se trabaja?

Si, toda la aplicación cliente esta escrita en AngularJS. Con Eclipse.

Se utilizan librerías como JQuery con Javascript. ¿Cuáles?

Las 2 librerías principal sen AngularJs y Jquery. También se usa un serie de librerías para la gestión de timepicker, slider…