

**UNIVERSIDAD DE EXTREMADURA**



**PI- Programación en Internet**

# **ENTREGA 2**

Web App using REST and AngularJS

## **RedNotes**

**AUTOR:**

Francisco Javier Rojo Martín  
(DNI:76042262-F)

# **ÍNDICE**

<b>1- Introducción</b>	<b>2</b>
<b>2- Requisitos obligatorios</b>	<b>3</b>
<b>3- Extras</b>	<b>6</b>
3.1- Lista de extras	6
3.2- Modificaciones en la Base de Datos	13
3.3- Futuros extras interesantes	14
<b>4- Anexo I: Contraseñas de usuarios</b>	<b>14</b>

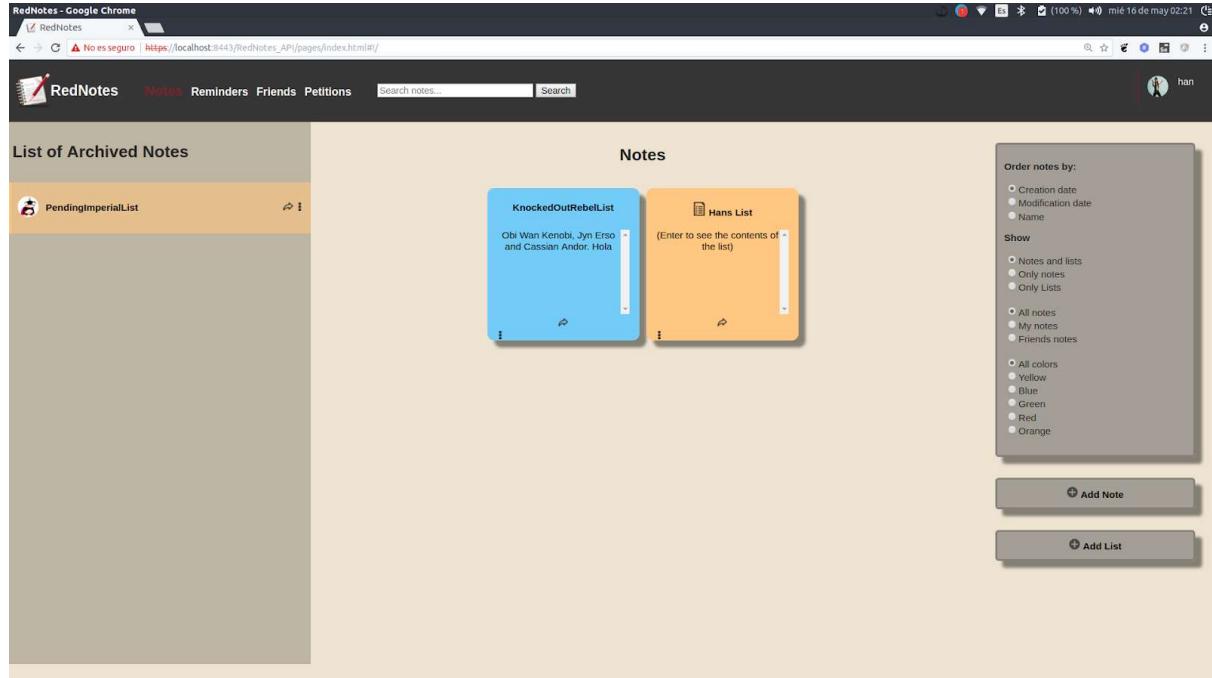
# 1- Introducción

El proyecto presentado consiste en una aplicación web de notas online (similar a la aplicación *Google Keep*), a la que se le ha denominado **RedNotes**.

Dicha aplicación, ofrece la posibilidad de tener una gestión de notas y listas, incluyendo funciones como recordatorios, gestión de amigos,... Todo ello, con el requisito de haberse registrado previamente en la aplicación.

Además, el usuario podrá proporcionar información como su país, nombre o ciudad, que ayudará a los usuarios a buscar nuevos amigos. También podrá poner su número de teléfono, de tal forma que sus amigos se puedan comunicar con él fuera de la aplicación. Incluso podrá personalizar su ícono con una serie de imágenes predefinidas en la aplicación.

Junto a la aplicación, se entrega una base de datos: *RedNotes.sqlite*. Esta base de datos está adaptada para la aplicación Web en cuestión.



## 2- Requisitos obligatorios

En la lista siguiente, encontramos como se han conseguido cada uno de los requisitos de la aplicación, marcados en la entrega (la enumeración coincide con la marcada en la entrega).

- **Requisito 1: HTML & CSS are valid.**

Este requisito se cumple mediante el uso adecuado de las etiquetas HTML y CSS. Existen Warnings relacionados con las variables de color, en el CSS. Así mismo, debido a que el CSS no se incluye de forma explícita en cada plantilla, no reconocerá las clases especificadas en el HTML.

Es interesante hablar de que, **se han incluido dos librerías de AngularJS**, que nos permiten usar la **propiedad ‘css’ dentro del ngRoute** de “app.js”, especificando, de esta forma, que plantilla CSS queremos incluir en cada una de las plantillas que complementan a “index.html”, según la ruta.

Estas librerías resultan muy útiles pues, sino, no podría usarse la propiedad “css” (tan útil si tienes varias plantillas CSS, una para cada tipo de página de la web).

Las librerías, las podemos encontrar en un proyecto GitHub abierto de *castillo-io*: <https://github.com/castillo-io/angular-css>

- **Requisito 2: Security.**

Se implementa la seguridad marcada mediante:

- Protección, mediante LoginFilter, de que un usuario no logueado, tan solo pueda acceder al LoginServlet, o realizar peticiones POST a una cierta URL (para registrarse).
- Un usuario no logueado no solo no podrá realizar peticiones REST, sino que no podrá acceder a los archivos “.html” de la página web. Esto, se controla en el LoginFilter.
- En el archivo “web.xml”, se ha añadido una parte para el uso de HTTPS, mediante el uso de TomCat configurado para SSL (siguiendo el guión de la práctica 6).
- La API, controlará que la información solicitada por un usuario, es información a la que debería tener acceso ese usuario. Si no es así, devolverá un mensaje de error.
- La API, nunca devolverá información “sensible” de otro usuario, que no sea nuestro amigo (más allá de la esencial para buscar al usuario, como es el nombre, username, ciudad o país).

- **Requisito 3: Good practices.**

Se han seguido las buenas prácticas estudiadas en la asignatura en todos los archivos “.js”, teniendo en cuenta siempre el patrón Singleton que cumplen las factorías, pero no los controladores (permitiendo hacer código de mejor calidad en muchas ocasiones, aprovechandonos de esta característica).

Por ejemplo, en las factorías tan solo se han definido variables que necesitemos que conserven el valor aunque naveguemos por otros sitios de la web, pero no aquello que no necesitemos conservar. Esto último, se ha puesto en los controladores (pues, además, serán elementos que necesitemos para el “binding” entre la vista y el controlador, en la mayoría de casos).

- **Requisito 4: Any person can register as user of the web app, edit and delete his account.**

Se ha añadido, en la esquina superior derecha, un menú para las opciones del usuario logueado. Desde ahí, se podrá consultar la información de la cuenta y modificarla, así como borrar la cuenta.

*Implementación:* Para implementar estas funciones, se ha creado un controlador: *PerfilCtrl*, que será el encargado de mostrarnos, editar y borrar el perfil, si así lo desea el usuario.

- **Requisito 5: Any user can create, see and edit notes.**

Desde la página principal o página de Notas, se incorpora un botón para añadir nuevas notas. También se puede abrir cada nota, así como editarla y eliminarla desde dentro de la página de la propia nota.

*Implementación:* Para implementar estas funciones, se han creado dos controlador: *NotesCtrl*, que será el encargado de mostrarnos todas las notas del usuario; y *NoteCtrl*, que será el encargado de mostrar la información de una nota específica, así como de editarla y borrarla.

- **Requisito 6: Any user can share any of her note.**

En la página de la Nota, así como desde el menú de Notas, se pueden compartir las notas con los amigos. Para cada nota, se muestra la lista de amigos del usuario, excepto aquellos con los que ya está compartida la nota.

*Implementación:* Se ha incluido un método en *FriendsCtrl*, que nos permite compartir una nota. Este método, será llamado desde “ShareNote.html”.

Si la nota que se pretende compartir no pertenece al usuario, aparecerá un alert indicándonos que solo se pueden compartir notas donde seamos el dueño de la nota.

- **Requisito 7: Any user can edit notes that were shared with them.**

Para todas las notas, sean propias o no, se pueden editar tal y como se explicó en el requisito 5.

*Implementación:* Esto, como ya se explicó en el Requisito 5, se realiza en *NoteCtrl*, de la misma forma que se edita una nota propia.

- **Requisito 8: Any user can “delete” her notes or notes that were shared with her.**

Para eliminar las notas, se comprueba quien es el dueño de la nota. Si la nota es del dueño, además de eliminarse la entrada en *UsersNotes* para el usuario, se eliminan todas las entradas de esta nota en *UsersNotes* (para todos los usuarios), así como se elimina definitivamente la nota.

*Implementación:* Esto, independientemente de si es una nota o una lista, se realizará siempre desde *NoteCtrl*. Este, usará un método de la factoría de notas, que tan solo realizará una llamada a la API para eliminar la nota. Será en la API donde se decida, según si el usuario logueado es el dueño o no, si se debe eliminar la nota para todos los usuarios o solo para él.

- **Requisito 9: Pinned notes.**

Se encuentran dos botones: pinned y unpinned. Tanto en el menú desplegable de cada nota, dentro de la lista de notas; o en el menú de opciones inferior, en la vista en detalle de una nota. Se mostrará solo uno u otro, según el estado actual de la nota. Cuando una nota está fijada, se muestra en primer lugar, junto con las notas fijadas, en la página de Notas.

*Implementación:* Para mostrar solo una de las opciones (fijar o desfijar), se usa la directiva “ng-show” de AngularJS, tanto en “Note.html” como en “Notes.html”. Así mismo, la funcionalidad para fijar o desfijar notas, la encontramos tanto en *NotesCtrl* como en *NoteCtrl*. Podría haberse creado un tercer controlador, que se incluyese sólo en esa zona del template, y que se encargará de fijar y archivar notas (y así no duplicar código en los dos primeros controladores), pero debido a la simpleza del método (una llamada a un método de una factoría), se ha optado por no realizarlo.

- **Requisito 10: Archived notes.**

Se realiza de igual forma que para fijar una nota. Cuando una nota está archivada, se muestra en la lista lateral izquierda de la página de notas.

Implementación: Se realiza de igual forma que para fijar una nota.

- **Requisito 11: Search.**

Se pueden buscar notas, siempre que ponga “Search note...” en la barra de búsqueda superior (pues, en otros casos, puede aparecer “Search friend...”, al ser una página relacionada con la ampliación de amigos).

La búsqueda de notas, se realiza, tanto por título como por contenido. Si el texto buscado se encuentra en el título o en el contenido de una nota, se muestra esta nota, en la lista de notas buscadas.

Implementación: Se ha incluido un método en *HeaderCtrl*, que se encarga de realizar las búsquedas según el parámetro “search” introducido en la barra. Con esto, se nos redirigirá a una página nueva con el resultado. Se ha hecho así para poder entrar en las notas, y volver atrás en el navegador en cualquier momento, pudiendo acceder a los resultados de la búsqueda de forma sencilla.

## **3- Extras**

### **3.1- Lista de extras**

En la lista siguiente, se pueden ver la lista de extras que se han incluido a la aplicación web, y que ya se incluyeron en la Entrega 1 de esta práctica (**se han conservado todos y muchos se han mejorado, en funcionalidad o estética**).

Estos extras, no se encuentran completamente en la lista de la entrega. Igualmente, hay otros extras de la lista que se han implementado de una manera ligeramente distinta de la que se decía en la entrega.

- **Extra 1: Filtrado y ordenación de notas.**

Se añade un menú en la página principal de Notas, a la derecha. Desde este menú, se puede decidir que tipo de notas mostrar (listas o notas), así como que colores mostrar, y en qué orden.

Cuando iniciamos sesión, se ponen unas opciones de filtrada y ordenación por defecto. Sin embargo, mientras tengamos iniciada la sesión, los cambios que

se produzcan en estas opciones se guardarán durante toda la sesión (mediante el uso de atributos en la sesión), hasta que las cambiemos o cerremos la sesión.

**Implementación:** Se han añadido tres parámetros a *NotesCtrl*, uno para cada tipo de filtrado, así como otro para la ordenación. Estos parámetros se sincronizan con los de una factoría, donde los almacenaremos “persistentemente”. Después, con el filtro “orderBy” de AngularJS, ordenaremos las notas, al mismo tiempo que filtramos con el filtro “filterNotes”, creado por nosotros.

### **Mejoras:**

- Ahora, cuando un usuario no tiene notas, se oculta el menú de filtrado y ordenación de notas. Esto se ha conseguido con un “ng-show”.
- Ahora, ya no se necesita hacer uso de un botón “Enviar”, sino que, al pulsar sobre las distintas opciones, se irán actualizando automáticamente. Esto se debe a que, el filtrado y la ordenación, las hacen filtros de AngularJS (uno definido por mí, y otro genérico de AngularJS), trabajando con los datos de la factoría de parámetros o con los del controlador (que estarán sincronizados en todo momento).
- Ahora, cuando nos vamos a otro sitio de la web, y volvemos a la vista “Notes”, las opciones de filtrado y ordenación se conservan como las dejamos. Esto se consigue con la factoría “filterNotesParametersFactory”, que almacenará los valores seleccionados con su patrón Singleton.

### ● **Extra 2: Listas.**

Se ha añadido la posibilidad de crear listas, además de notas, en la aplicación. Dichas listas, se crean desde el menú de Notas, como las notas normales. Se diferencian de las notas normales, en el hecho de que el contenido es una lista de elementos, en lugar de un texto plano. Sobre una lista creada, podremos modificar su estado, añadiendo (uno a uno) y eliminando elementos (tantos como se quieran de golpe), así como cambiar el estado de cada elemento (mediante un checkbox asociado a dicho elemento). Aparte de esto, las listas permiten una gestión normal de cualquier nota: modificar título, color, compartir,...

**Implementación:** Se ha creado un controlador *ListCtrl*, donde cargaremos la lista que quiere ver el usuario. Al cargar esta lista, se deberán partir el campo contenido en los distintos elementos de la lista.

La implementación completa de las listas, es bastante compleja, por lo que se recomienda ver el controlador, en lugar de explicarse todo aquí.

### **Mejoras:**

- Ahora, el usuario solo tendrá una vista para editar la nota. Desde esta vista, y de forma dinámica, se podrán añadir y eliminar elementos. Cada vez que eliminamos un conjunto de elementos, se crea una nueva versión de la lista, pero, cuando añadimos elementos, solo crea una versión al “Guardar los cambios”.
- Ahora, el botón de “Guardar los cambios” solo funcionará si se ha cambiado algo en el título o contenido de la lista (sino, no nos dejará usarlo, teniendo que usar el “Cancelar”).
- Ahora, durante la edición de las listas, se podrá editar sobre una versión vieja de la nota, en lugar de sobre la actual, si así se desea. Esta me ha parecido una característica muy interesante, por lo que la he incluido. Esta mejora, se explicará mejor en el Extra dedicado a *Control de Versiones*.

### **• Extra 3: Imágenes de los usuarios.**

Se ha añadido información a los usuarios, de tal forma que cada usuario tenga una imagen de perfil. Por defecto, se asigna una predeterminada. El usuario puede modificar su imagen de sesión, cogiendo una de las que se le ofrece desde un menú de imágenes. Este menú, es accesible desde el Perfil del usuario (no desde el Menú de Edición del Perfil). Todas las modificaciones realizadas, se guardan en la base de datos, para posteriores veces que el usuario inicie sesión.

Estas imágenes, resultan muy útiles para mostrar junto con el nombre de usuario en las notas (para mostrar el usuario dueño de la nota), así como para mostrar en las distintas versiones de una nota (dejándonos ver quien ha hecho cada edición en una nota).

Dichas imágenes, se guardan en una tabla diccionario, en forma de URLs.

**Implementación:** Se ha introducido una tabla diccionario para las URLs en la Base de Datos, así como un campo para la imagen en los usuarios. Se implementa la funcionalidad en el *PerfilCtrl*.

### **• Extra 4: Recordatorios.**

Esta ampliación, nos permite que, un usuario, pueda asociar múltiples recordatorios a una nota (podrá asociar varios recordatorios a una misma nota, siempre y cuando la fecha del recordatorio no coincida en varias).

En el recordatorio, se añade la nota, la fecha y hora de recordatorio, y una breve descripción (para adjuntar información relevante). También, podemos eliminar los recordatorios que queramos, antes de que llegue su fecha.

Una vez la fecha del recordatorio es más vieja que la fecha actual, el recordatorio desaparece del menú de recordatorios para siempre.

Cada usuario tendrá sus propios recordatorios sobre sus notas (mostrándoles estas todas cuando vayan a crear el recordatorio, siendo mucho más fácil crearlo).

**Implementación:** Se ha creado un controlador *RemindersCtrl*, encargado de crear un recordatorio nuevo (mostrándonos la lista de notas y listas que tenemos, para crear el recordatorio sobre una de esas notas o listas). Igualmente, se ha definido el filtro “*filterReminders*”, que se encargará de solo mostrar las recordatorios posteriores a la fecha actual. Los recordatorios, se mostrarán ordenados por la fecha.

#### **Mejoras:**

- Ahora, el usuario puede ir a la nota o lista asociada al recordatorio directamente desde el mismo recordatorio, clickeando sobre él.

#### **• Extra 5: Gestión de amigos.**

La gestión de amigos, se complementa con el siguiente Extra. Nos permite tener amigos en la aplicación. Estos amigos, serán con los que podamos compartir las notas (mediante un menú donde aparecen los amigos a los que aún no hemos compartido la nota). El usuario, tendrá un menú para ver sus amigos, así como puede buscarlos con la barra de búsqueda u ordenarlos con las opciones del lateral izquierdo.

En la página de amigos, se muestra una barra, a la izquierda, donde vemos todos los amigos o los coincidentes con la búsqueda realizada en la barra superior (coincidentes en *username* o *name*). En la parte derecha, la información del amigo, así como un apartado para acceder a las notas que nos ha compartido ese amigo. También hay un botón para añadir amigos.

A la hora de añadir amigos, nos encontraremos con una página similar a la anterior, donde tenemos una lista de todos los usuarios de la base de datos que aún no son amigos, así como una barra de búsqueda avanzada (se explica en el siguiente Extra). Cuando pulsamos sobre un usuario, nos muestra una pequeña cantidad de información y nos da la opción de añadirlo. Si aceptamos, se le envía una petición de amistad, que el otro usuario podrá aceptar o rechazar. Igualmente, si nos arrepentimos de haberla mandado, podemos borrar la petición nosotros mismos. Todo ello desde la página Peticiones de nuestro sitio web.

Se recomienda encarecidamente probar esta parte de la aplicación, pues ha sido muy laboriosa, y resulta complejo resumirla aquí, sin una muestra.

**Implementación:** Se ha creado un controlador *FriendsCtrl*, encargado de todo lo relativo a amistades entre usuarios (amigos actuales, búsqueda de nuevos amigos y gestión de peticiones de amistad). Al ser esta ampliación, al igual que la de las listas, una ampliación muy extensa, es mejor ver el controlador, pues resulta difícil explicar aquí todo el funcionamiento de este.

Tan solo decir que, para ordenar los amigos por “username” o “fecha de amistad”, se hace uso del filtro “orderBy” de AngularJS.

#### ***Mejoras:***

- Ahora, el usuario debe introducir algún tipo de información en la búsqueda de un nuevo amigo. Si no se introduce nada, no se mostrarán todos los usuarios de la aplicación (como se hacía antes).
- **Extra 6: Búsqueda avanzada de nuevos amigos.**  
Se añade un menú en la página de búsqueda de amigos, que nos permite filtrar la búsqueda de nuevos amigos. Podremos buscar amigos con la combinación de información de los campos: Name, Username, Country y City. Resulta interesante decir que se forma el WHERE de la consulta SQL para obtener los resultados mediante funciones del tipo String. Esto último, se realiza en el lado del servidor, con la información que le llega a partir de la API.

**Implementación:** Se ha creado un controlador *newFriendsCtrl*, que se encarga de realizar la petición a la API. Con todos los usuarios coincidentes que le devolverá la API, hace un filtrado, consultando los amigos actuales del usuario, y eliminando de la lista primera aquellos usuarios que ya son amigos del usuario en cuestión.

#### ***Mejoras:***

- Ahora, el usuario debe introducir algún tipo de información en la búsqueda de un nuevo amigo. Si no se introduce nada, no se mostrarán todos los usuarios de la aplicación (como se hacía antes).
- **Extra 7: Colores.**  
Se añade la opción de cambiar el color de todos los tipos de notas (notas y listas), permitiéndonos poner estas de 5 colores distintos. Además, a la hora de mostrar los colores de las notas, se comprueba si la nota está archivada o no, cambiando ligeramente el tono del color según la situación. Además, este color se guarda en la base de datos para cada usuario. Es decir, dos usuarios distintos, pueden tener, para una misma nota, dos colores distintos. Por defecto, el color inicial de una nota es el Amarillo.

**Implementación:** Desde el controlador *NoteCtrl* y *ListCtrl*, podremos editar el color de la nota o lista en cuestión. Para ello, se modifica el valor de la propiedad desde el HTML, llamando a una función ‘*updateColor()*’, con el nuevo color solicitado como parámetro.

Desde ahí, ya será donde llamemos al método de la factoría que actualiza la información de la entrada en *UsersNote*.

### **Mejoras:**

- Ahora, el filtrado por color, como pasaba con los otros filtrados, explicados en el Extra 1, “persiste” aunque cambiemos de vista en la aplicación web, mediante el uso de la factoría de parámetros.
- Ahora, el filtrado de color, como explicamos en el Extra 1 para los otros parámetros, se realiza de forma dinámica, al pulsar el botón con el color deseado.

### ● **Extra 8: Control de versiones.**

Se añade un control de versiones de las notas. Se permite, por cada nota:

- Ver versiones viejas de la nota.
- Establecer una de las versiones viejas como la actual (y eliminar las posteriores)
- Eliminar una versión determinada de la nota, mientras no sea la única que queda (en cuyo caso, se deberá eliminar directamente la nota). Si la versión a eliminar es la actual, el efecto será el mismo a volver a la versión anterior de la nota.

Para ayudar a la navegación entre versiones, se han añadido aspectos como colorear de un color más fuerte, la versión en la que nos encontramos dentro de una nota.

Al crear la nota o lista, se crea la versión inicial de la nota.

Cuando modificamos el título o el contenido de la nota o lista (en esta última, añadiendo, eliminando o modificando el estado de los elementos), se modifica la fecha de modificación de la nota, y se añade una versión nueva a la lista de versiones de la nota.

Se podrá ver quien fue el usuario que editó esa versión de la nota (usuario que provocó que se crease esa nueva versión), mediante su imagen de perfil y su username (cómo “*label*” de la imagen).

**Implementación:** Se ha creado una factoría, que se encargará de gestionar las peticiones relativas a las versiones de notas y listas. Dicha interfaz, será muy simple, y será aprovechada por *NoteCtrl* y *ListCtrl*, para mostrar las versiones de la lista, así como todas las otras funciones asociadas a esta.

Podríamos haber creado un nuevo controlador que nos permitiese gestionar las versiones de las notas de forma independiente a *NoteCtrl* y *ListCtrl*, pero esto

habría imposibilitado la mejora que expongo a continuación (y, bajo mi punto de vista, una de las mejores funciones que ofrece este control de versiones).

### **Mejoras:**

- Ahora, el usuario, mientras está editando una nota o lista, podrá seleccionar una versión antigua de la nota, y actualizar la nota sobre esa versión. Incluso podrá seleccionar la versión antigua y darle a “Guardar cambios”, consiguiendo el mismo resultado que volviendo a una versión antigua, pero sin eliminar las versiones posteriores. **Resulta muy interesante decir que esto se ha conseguido implementar incluso para las listas, combinando ambas Ampliaciones, por tanto, de una manera un tanto compleja, pero con un muy buen resultado.**

- **Extra 9: Inicio de sesión con usuario y email.**

Se permite iniciar sesión mediante el email o username. Así, el usuario podrá optar por cualquiera de estos dos campos, acompañados de la contraseña, para iniciar sesión en la aplicación web. Para esto, nos ayudamos del hecho de que los *usernames* no permiten el carácter “@”, mientras que el email obliga a tenerlo.

**Implementación:** Esto se implementa en la parte del servidor, tal y como se hacía en los Servlets.

- **Extra 10: Encriptado de contraseñas en Base de Datos.**

Se añade ha añadido un algoritmo de encriptado al proyecto, permitiéndonos encriptar las contraseñas de los usuarios al almacenarlas en la Base de Datos (tanto al crear la cuenta como al editar la contraseña). Se ha considerado una buena ampliación, al no controlarse los SQL Injection. Gracias a esto, aunque se hiciese un “SELECT” de la tabla User, no se obtendrían la contraseñas de forma sencilla.

**Implementación:** Esto se implementa en la API, concretamente, en el POST del *UsersResource*, encriptando la contraseña del nuevo usuario, antes de añadirlo a la Base de Datos.

También se han llevado a cabo cambios menores, aparte de los mencionados en las mejoras de cada extra, que se espera que se tengan en cuenta, en conjunto, como otro extra:

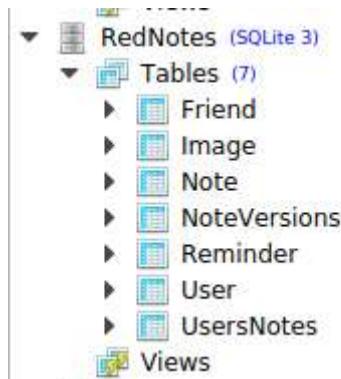
- **Extra 11: Detalles varios.**

- Ahora, el botón de “Guardar los cambios” solo funcionará si se ha cambiado algo en el título o contenido de la nota o lista(sino, no nos dejará usarlo, teniendo que usar el “Cancelar”).

- Se ha generado documentación para la API REST que se emplea en la aplicación. Dicha documentación, se ha creado con JavaDoc, y nos indica el comportamiento de cada método, para cada recurso, así como aspectos de estos como:
  - Propiedades obligatorias en el envío de un POST o PUT, así como las que no es obligatorio que aparezcan.
  - Restricciones en los parámetros del JSON.
  - ...
- Se ha conseguido un estilo más unificado y más bonito, al usar enlaces “<a>” para funciones donde antes se usaban “<button>”, que se mezclaban con otros enlaces y conseguían “romper” el estilo que se pretendía conseguir.

### 3.2- Modificaciones en la Base de Datos

Téngase en cuenta que, para estas ampliaciones, **se han creado varias tablas dependientes (de Note o UsersNote) en la base de datos**. En todo momento, se gestiona que las tuplas dependientes se eliminan de estas nuevas tablas, al eliminar las notas o cuentas de usuarios a las que están asociadas. Podríamos dejar este trabajo a la base de datos, mediante el “ON DELETE CASCADE”, pero se ha optado por controlarlo nosotros en la aplicación, dándonos mayor seguridad de que todo será consistente.



Como se puede ver, se han incluido las tablas:

- **Friend** (*almacena amistades entre usuarios de User*)
- **Image** (*diccionario de imágenes de perfil de usuarios*)
- **NoteVersions** (*almacena informaciones de cada versión de una nota*)
- **Reminder** (*almacena información de los recordatorios*)

### 3.3- Futuros extras interesantes

Existen otras muchas mejoras que me gustaría haber implementado, pero, debido al limitado tiempo, no se han podido implementar. Ejemplos de estas posibles mejoras, serían:

- Poder editar los recordatorios ya creados.
- Crear un menú para ver los recordatorios de una nota concreta.
- Poder crear notas públicas, accesibles a cualquier usuario (amigo o no), desde la ventana de búsqueda. Además, que solo sean leídas (ni editadas, ni borradas).
- Dar solución a los posibles SQLInjection e inyecciones de Scripts (más allá de encriptar las contraseñas de la Base de Datos).
- Crear una sección para mostrar todos los enlaces que se encuentren en una nota.
- Crear una sección para mostrar los videos de YouTube, cuyo enlace fuese incluido en la nota.
- Poder añadir una ubicación con Google Maps a un recordatorio.
- ...

### 4- Anexo I: Contraseñas de usuarios

Se añade este anexo a la documentación, con el fin de poder saber la contraseña de cada usuario, pues, al estar encriptadas en la Base de Datos, no se pueden saber al abrir la tabla simplemente.

idu	username	password	email
0	darth	vadER99	darth.vader@darksideoftthe.org
1	emperor	palpatiNE99	emperor.palpatine@darksideoftthe.org
2	leia	orgaNA99	leia.organa@lightsideoftthe.org
3	luke	skywalkER99	luke.skywalker@lightsideoftthe.org
4	han	soloEL54	han.solo@lightsideoftthe.org
9	lopez	pruebaAA11	esto@es.unaprueba
10	userPrueba	pruebaAA12	prueba@prueba.com
11	pepe	holaM9	pepe@pep.e
12	javi97	Prueba11	javier@rojo.com
14	javiER	rojoBB99	javier.rojo@unex.es

Se añade esta imagen entre los archivos entregados también.