Ayuda y Gestión Psicológica

Trabajo Fin de Ciclo - IES Alfonso X El Sabio

Realizado por: Nicolás Gabarrón Blaya
Tutora: Marioly Vivancos Abad
Fecha de inicio: 19 de marzo de 2022

Índice

Desarrollo de la idea

La idea de realización de este proyecto reside en resolver una necesidad propia de acordarme de aquellos acontecimientos que me ocurren a lo largo de las dos semanas que transcurren entre sesión y sesión con mi psicóloga.
Con esa necesidad en la cabeza, se me ocurrió desarrollar una aplicación para el Proyecto Fin de Ciclo, con la que se facilitara a cualquier usuario de una forma totalmente gratuita la "gestión" de su Salud Mental. Entrecomillo gestión ya que no se trata de gestionar de una forma directa la Salud Mental, ya que de eso se encargan los psicólogos; se trata de que el usuario final tenga centralizada toda su información de forma sencilla y sea capaz de llevar un control de la misma.
Esto es útil para todas las personas, estén siguiendo terapia o no, ya que también puede servir como un método de alivio y escape donde "desahogarse" diariamente.
Además, puede también puede ayudar a llevar un seguimiento de metas personales que el usuario quiera conseguir a lo largo del tiempo.
Entre las funcionalidades principales están:

- Diario personal.
- Lista de hábitos.
- Lista de propósitos.
- Registro diario del estado anímico.
- Registro diario de sucesos clave.
- Evaluación diaria y semanal.
- Solicitar ayuda.
- Integración con widgets, mostrando frases motivacionales.

(algunas de estas funcionalidades estarán disponibles en un futuro)

En cuanto a visión de negocio sobre el producto final, tengo intención de que esta misma aplicación pueda ser vendida a gabinetes psicológicos de forma en que los profesionales de dicho gabinete tengan acceso a la información redactada por sus pacientes, sirviendo esto para diagnosticar de una forma más eficiente y poder evaluar la evolución de los mismos con el paso del tiempo.

Además, se podrían fechar citas, programar tests, evaluaciones o cualquier evento o actividad que se desee, cuyos eventos se asociarían a un paciente concreto el cual recibiría los recordatorios en su teléfono móvil.

Arquitectura de la aplicación

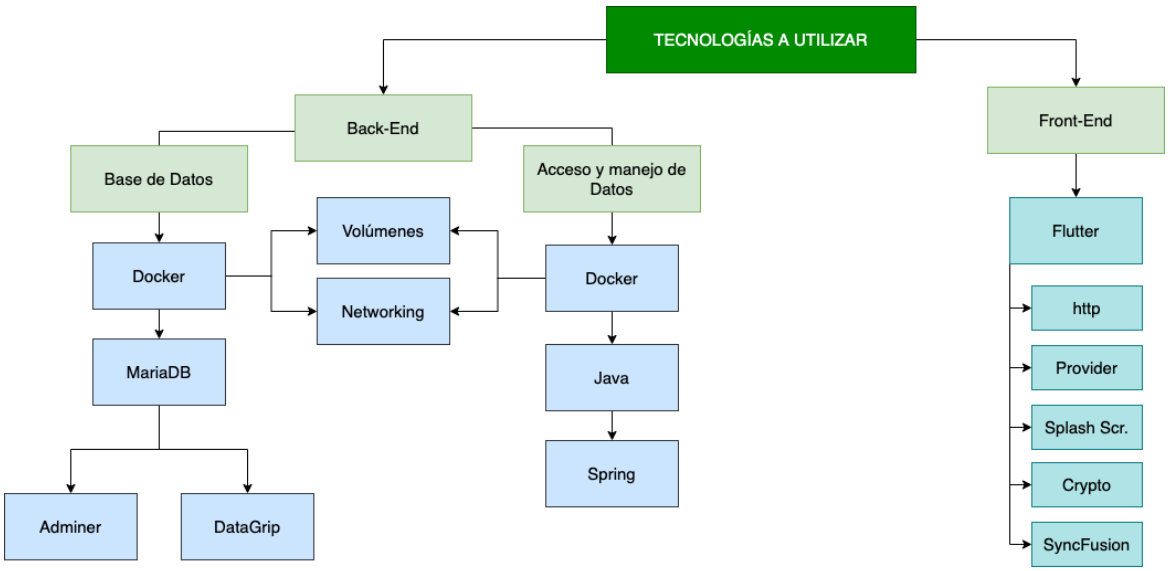
El producto final tendrá una arquitectura "cliente-servidor" en 3 capas, separando así el *front-end* (cliente) de toda la lógica y trabajo con los datos, tareas las cuales residirán en el *back-end*.

El cliente se comunicará con el servidor a través de peticiones *GET* y *POST*, dependiendo de si quiere obtener o enviar datos. Esta decisión es tomada debido a que delegando el trabajo de esta manera, se asegura la integridad de los datos, aislando la aplicación de todo código de acceso directo a la Base de Datos, lo cual aumenta exponencialmente la seguridad de la misma, ya que toda operación que se quiera realizar, tendrá que ser vía petición HTTP(s).

En caso de que alguna de estas peticiones fallase, el servidor devolvería un código de error y la tarea no se realizaría, capturando dicho código en el cliente.

De hacerlo todo en el cliente, esto podría ocasionar también que la aplicación se detuviera de forma inesperada, empeorando así la experiencia de usuario y la seguridad de los datos.

Tecnologías utilizadas.



Desarrollado por Nicolás Gabarrón Blaya.

Las tecnologías que se han decidido usar para este proyecto son:

Front-End:

Para el *frontend* se ha decidido utilizar el *framework Cross-Platform* Flutter. Esta decisión ha sido tomada ya que se pretende que la aplicación sea multiplataforma, es decir, esté disponible para Android e iOS.

¿Por qué este framework y no otro como por ejemplo React Native?

La respuesta está en el rendimiento. Flutter ha demostrado tener un rendimiento casi nativo, factor que React Native no (o no al mismo nivel). Además, el peso final de las aplicaciones es superior en React, por lo que la relación "peso/potencia" no es tan favorable.

Otro motivo de peso es la popularidad. Flutter es un framework en crecimiento exponencial entre las empresas y los desarrollos móviles.

A su vez este desarrollo del front se complementará con paquetes “pub.dev” que añaden ciertas funcionalidades a nuestro proyecto.
Algunos de ellos lo son *http, provider, SplashScreen, crypto, syncfusion*, entre otros.

Back-End:

Para el *backend* se ha decidido hacer en dos capas separadas. Esta decisión ha sido tomada para favorecer la seguridad y la abstracción de los datos.
Se pretende utilizar el software de virtualización *Docker*, mediante el cual los servicios estarán montados sobre una capa de abstracción, el *Docker Engine*.
Esto es realmente útil, ya que todo servicio será totalmente independiente a la máquina dónde se encuentra ejecutando.
Especificando más la forma en que se montarán estos servicios, será mediante *docker-compose*.
Se creará un *stack*, el cual contendrá tanto la base de datos como el servidor de acceso a dichos datos (*API-REST*).
Este *stack* dispondrá de una red privada y volúmenes privados para hacer persistentes dichos datos.
Estos volúmenes se verán involucrados copias de seguridad diarias en el servidor donde se encuentren almacenados.
Se ha decidido utilizar un Motor de Bases de Datos SQL (*Structured Query Language*) para un mayor manejo y permitir relaciones entre entidades dentro de nuestro modelo de negocio.
Esta decisión ha sido tomada en el último momento, debido a diferentes dificultades que se han encontrado a la hora de desarrollar todo el *backend*.
Como lenguaje elegido para el desarrollo del propio *backend* se ha elegido Java, concretamente con el framework Spring.
A este framework se le han añadido diferentes dependencias, tales como *Spring Web*, *Spring Security*, *JSON Web Token*, y los controladores de la propia Base de Datos (*JDBC Drivers*).

A cada petición, se comprobará el rol del usuario (usuario normal, especialista o administrador) y se le otorgará acceso a los recursos según éste.

Metodología de desarrollo.

Se ha decidido utilizar metodologías ágiles para la ejecución de este proyecto.
En concreto *scrum*, con el software de gestión *Jira* de *Atlassian*.
En éstos métodos Agile se utilizan iteraciones sucesivas las cuales tienen el siguiente funcionamiento:
Al inicio, se definen los *requisitos* que se abordarán dentro de cada *sprint*.
En segundo lugar, hay una fase de *diseño* en la que se define cómo se le da solución a los requisitos propuestos.
En penúltimo lugar, está la fase de *desarrollo*, en la que los desarrolladores trabajan en desarrollar la funcionalidad objetivo del *sprint*.
En último lugar, se realizan pruebas al *software* desarrollado para comprobar que nada es erróneo y cumple con los estándares de calidad propuestos.
Tras un número determinado de *sprints*, se obtendrá el *producto final*.

A la hora del desarrollo de *software*, se pretende hacer uso de la herramienta de control de versiones *Git* con su flujo de trabajo *flow*.
Esta decisión es tomada ya que si se estructura correctamente el proyecto desde el inicio, a posteriori, cuando haya que realizar mantenimiento, se hará una tarea mucho más sencilla.
Utilizando *git-flow* tenemos dos ramas destacadas: *main* y *development*.
La rama *main* es la destinada a *producción*. Aquí estarán aquellas versiones (o *releases*) que consigan superar todos los test y tengan las implementaciones que se estimen oportunas.

La rama *development* su mismo nombre indica para que se utiliza. Sirve para separar todo el desarrollo de producción, evitando conflictos de código, funcionalidades incompletas y desorganización.
De la rama *development* surgen ramas hijas llamadas *features*, que van siendo creadas e integradas en *development* según van siendo desarrolladas dichas funcionalidades.
También es importante conocer que entre la rama *development* y la rama *main* hay dos ramas *intermedias* muy importantes: *releases* y *hotfixes*.
La rama *releases* sirve para que cuando en *development* tenemos una versión lista (según el desarrollador), pase a ser probada por el equipo de calidad. Si el equipo de calidad da el *OK*, pasa a producción (es decir, a la rama *main*) y también es integrada de nuevo en la rama *development*, por si ha sufrido algún cambio el código desde que entro en *releases*.
La rama *hotfixes* está dedicada a solucionar problemas urgentes en producción. Normalmente estos problemas son causados por vulnerabilidades de alguna dependencia, pues el software debe haber sido probado antes de que alcance la propia rama main. No obstante está ahí para situarnos en ella siempre que queramos solucionar rápidamente cualquier error de producción.

Evidentemente, en este escenario de proyecto, todo este trabajo de equipo y distintas “secciones” serán realizadas por mí.

Secciones (screens) de la aplicación.

- **Login**: pantalla que se desplegará nada más iniciar la aplicación. En ella se pedirá que el usuario acceda con sus claves (e-mail y contraseña). En caso de no tener cuenta habrá un botón "registrar" que cargará un formulario mediante el cual el usuario podrá registrarse en el sistema.
  Esta pantalla comprobará también si previamente existe un token de acceso para evitar el acceso con claves cada vez que se entra en la aplicación.

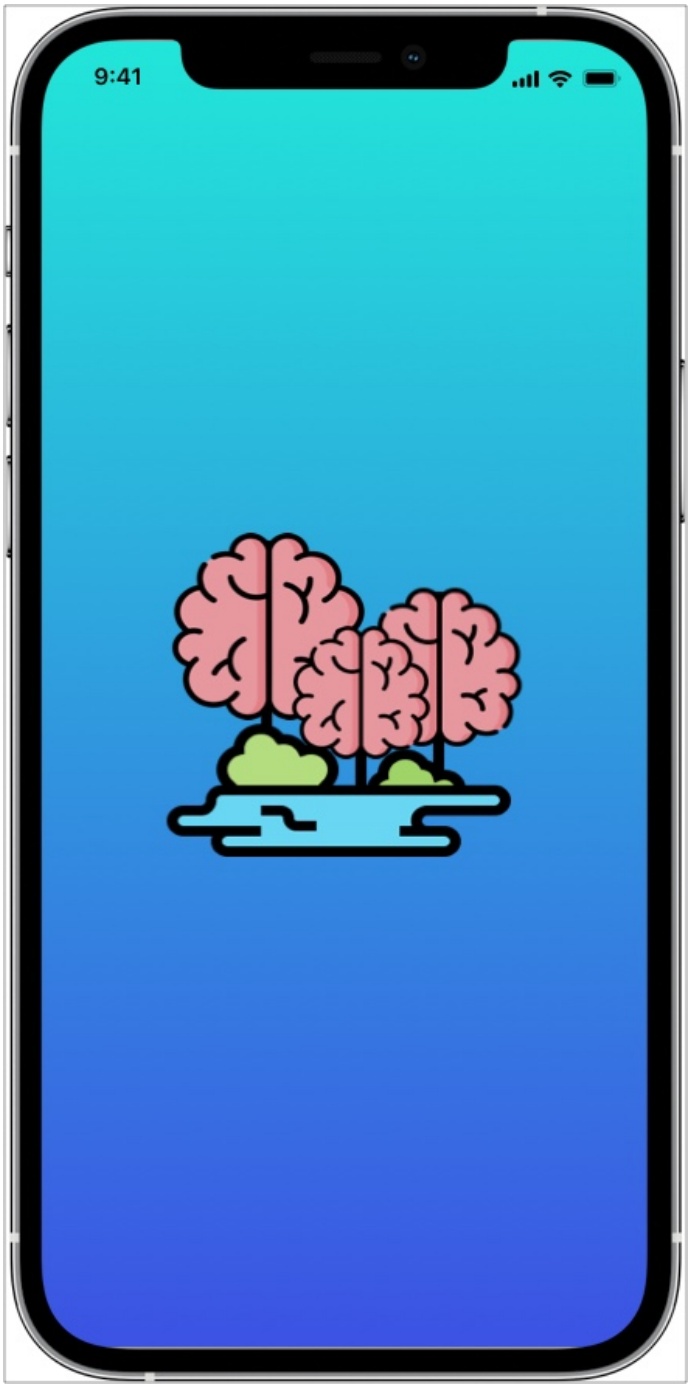
- **Pantalla de bienvenida**: pantalla principal en la cual se le da la bienvenida al usuario, mostrando alguna frase de motivación aleatoria con una estadística sobre el estado anímico semanal (media).
  Habrán enlaces con *routing* a las demás interfaces de la aplicación, así como a los ajustes.

- **Diario**: pantalla en la cual el usuario cada día podrá hacer una entrada y redactar qué tal ha ido su día a modo diario.
  Se permitirá establecer una hora a la cual el usuario reciba una notificación para que recuerde escribirlo todos los días.

- **Sucesos clave diarios**: pantalla en la cual el usuario podrá registrar entradas anotando sucesos clave, catalogándolos de positivos a negativos, para posteriormente visualizarlos en forma de lista general. Se podrá filtrar por fechas.

- **Habit Tracker**: pantalla en la cual el usuario podrá establecer unos hábitos a seguir y cada día ir marcando si los ha realizado o no.
  Además, se podrá establecer una hora a la cual el usuario recibirá una notificación recordándole si ha realizado dicho hábito.

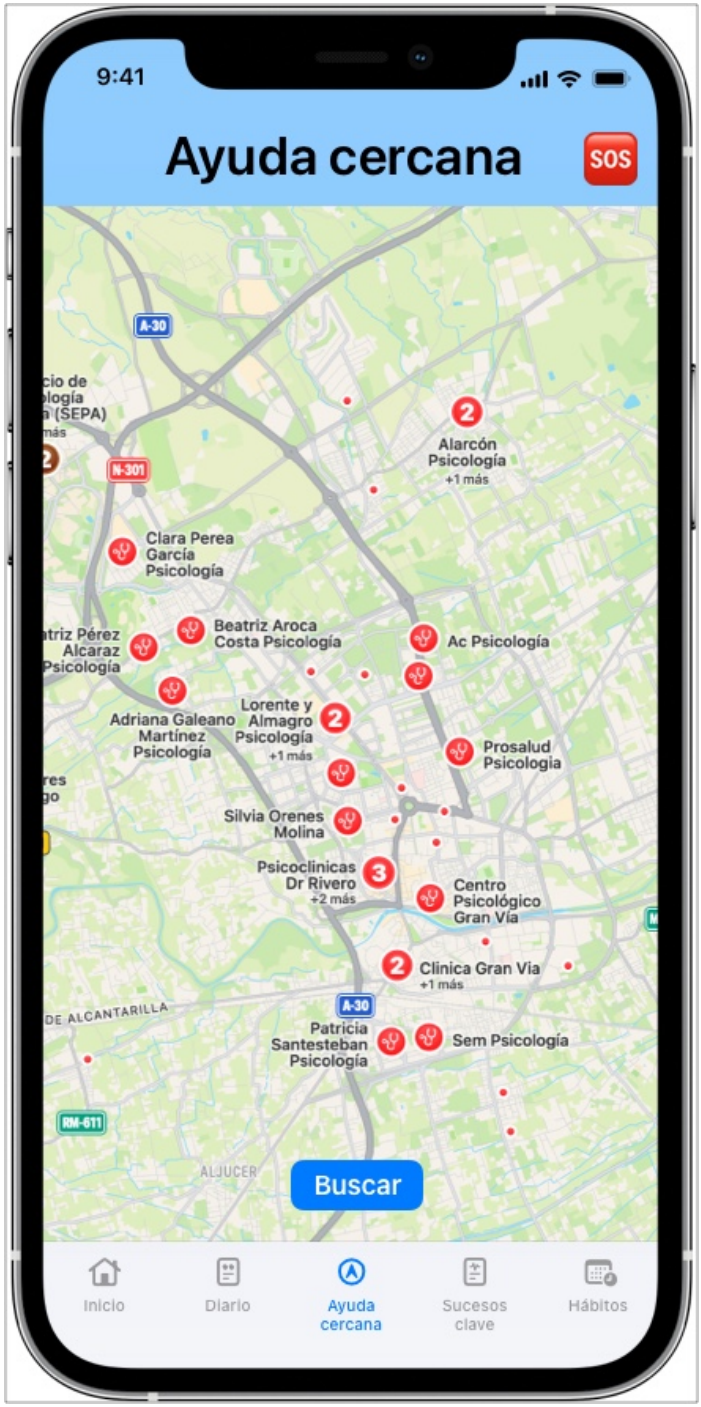
Prototipos.



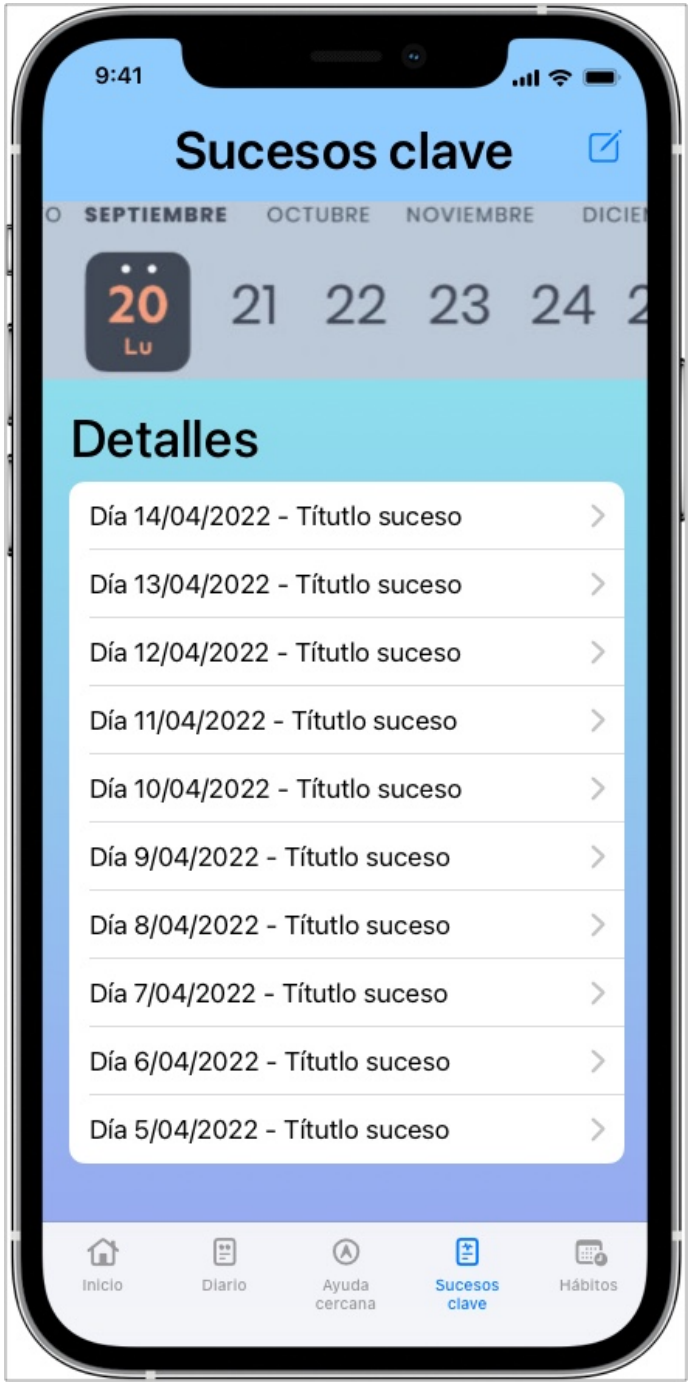
Prototipo 1. Splash Screen.

Prototipo 2. Pantalla principal.

Prototipo 3. Diario.



Prototipo 4. Ayuda cercana.



Prototipo 5. Sucesos clave.

Prototipo 6. Hábitos.

Estructura de modelos en el Back-End.

Los datos de la aplicación persistirán en una Base de Datos MongoDB. Estos datos están contenidos en modelos dentro de nuestro Front-End.

Estos modelos son los siguientes:

- Usuario.
  - id: Long.
  - username: String.
  - email : String.
  - password : String (no se almacenará en el dispositivo).
  - nombre : String.
  - apellidos : String.
  - ciudadNacimiento : String.
  - rol : Set de Rol.
- Rol.
  - id : Integer.
  - nombre : ERol.
- Entrada de diario.
  - id : Long.
  - usuario : Usuario (relación many to one).
  - titulo : String.
  - contenido : String.
  - fechaCreacion : Date.
- Suceso Clave.
  - id : Long.
  - usuario : Usuario (relación many to one).
  - titulo : String.
  - contenido : String.
  - fechaCreacion : Date.

    - valoracion: Integer.
- Recordatorios.
    - id : Long.
    - usuario : Usuario (relación many to one).
    - titulo : String.
    - detalle: String.
    - fechaCreacion : Date.
    - fechaRecordatorio: Date.
    - realizado: Boolean.

Endpoints del servidor.

El servidor ha sido organizado siguiendo una estructura hexagonal por lo cual ha sido muy sencillo organizar los diferentes *endpoints* a través de los cuales posteriormente se obtendrán los datos deseados.

Ruta TESTING en Postman: https://www.getpostman.com/collections/3b68a56a5de96bad7eef

Estos *endpoints* son los siguientes:

- **/api/auth** : Rutas de acceso al servidor.
    - /signin (POST): Intenta dar acceso al servidor al usuario con las credenciales enviadas en el cuerpo de la petición.
    - /signup (POST): Da de alta un nuevo usuario en el servidor.
    - /logout (POST): Envia una cookie vacía limpiando así los datos de acceso al servidor del usuario el cual lanza la petición.
- **/api/admin**: Ruta de recursos de administración.
    - /todosUsuarios (GET): Obtiene los datos de todos los usuarios. Se valora devolver DTOs simplificados para no recuperar datos confidenciales.
    - /usuario/{usuarioId} (GET): Obtiene los datos del usuario con ID especificado en la ruta.

    - /usuario/{usuarioId} (DELETE): Elimina de Base de Datos el usuario con ID especificado.
- **/api/test** : Rutas de test para diferentes aspectos del servidor.
    - /todos (GET): Todos los usuarios pueden entrar. Estén o no logueados.
    - /usuarios (GET): Usuarios con rol USUARIO / ESPECIALISTA / ADMINISTRADOR podrán acceder a este recurso.
    - /especialistas (GET): Usuarios con rol ESPECIALISTA / ADMINISTRADOR podrán acceder a este recurso.
    - /administradores (GET): Usuarios con rol ADMINISTRADOR podrán acceder a este recurso.
- **/api/diario** : Entradas de diario.
    - /entradas (GET): Obtiene todas las entradas de diario del usuario logueado que lanza dicha petición.
    - /entradas/{idEntrada} (GET): Obtiene la entrada de diario con ID especificado siempre y cuando pertenezca al usuario.
    - /entradas/nueva (POST): Crea una nueva entrada de diario en el usuario que lanza la petición.
    - /entradas/modificar/{idEntrada} (PATCH): Modifica la entrada de diario con ID especificado siempre y cuando pertenezca al usuario.
    - /entradas/eliminar/{idEntrada} (DELETE): Elimina la entrada de diario con ID especificado siempre y cuando pertenezca al usuario.
- **/api/sclave** : Sucesos clave.
    - /sucesos (GET) : Obtiene todos los sucesos clave del usuario logueado que lanza dicha petición.
    - /sucesos/{idSuceso} (GET): Obtiene el suceso con ID especificado siempre y cuando pertenezca al usuario.
    - /sucesos (POST): Crea un nuevo suceso clave en el usuario que lanza la petición.

- /sucesos/{idSuceso} (DELETE): Elimina el suceso con ID especificado siempre y cuando pertenezca al usuario.

- **/api/recordatorios** : Recordatorios.
  - /todos (GET): Obtiene todos los recordatorios del usuario fogueado que lanza dicha petición.
  - /{idRecordatorio} (GET): Obtiene el recordatorio con ID especificado siempre y cuando pertenezca al usuario.
  - /nuevo (POST): Crea un nuevo recordatorio en el usuario que lanza la petición.
  - /modificar/{idRecordatorio} (PATCH): Modifica el recordatorio con ID especificado siempre y cuando pertenezca al usuario.
  - /realizado/{idRecordatorio} (PATCH): Cambia el estado (realizado false a realizado true / realizado true a realizado false) del recordatorio con ID especificado siempre y cuando pertenezca al usuario.
  - /eliminar/{idRecordatorio} (DELETE): Elimina el recordatorio con ID especificado siempre y cuando pertenezca al usuario.

Posibles mejoras.

Backend.

- Mejorar la modularización de algunas funciones tales como obtención de Usuarios para su posterior trabajo con otras entidades (Entradas de diario, Sucesos Clave, Habitos).
- Crear endpoints específicos para especialistas a los que acceder a una colección de Usuarios y consultar sus diferentes datos. Posible mejora de modelos con una nueva incorporación de Tareas (usuario)?