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 transcurren entre sesión y sesión con mi psicóloga.

Con esa necesidad en la cabeza, se me ocurrió desarrollar una aplicación para este Proyecto Fin de Ciclo, con la que se facilitara a cualquier usuario de una forma completamente gratuita la "gestión" de su Salud Mental. Entrecomillo gestión ya que no se trata de gestionar de una forma directa la Salud Mental; de eso se encargan los psicólogos. Se trata de que el usuario final tenga centralizada toda su información de una forma sencilla y sea capaz de llevar un control de la misma.

Esto es útil para todas las personas, estén siguiendo una 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 (próximamente).
- Lista de recordatorios.
- Registro diario del estado anímico (próximamente).
- Registro diario de sucesos clave.
- Evaluación diaria y semanal.
- Solicitar ayuda (próximamente).
- Integración con widgets, mostrando frases motivacionales (próximamente).

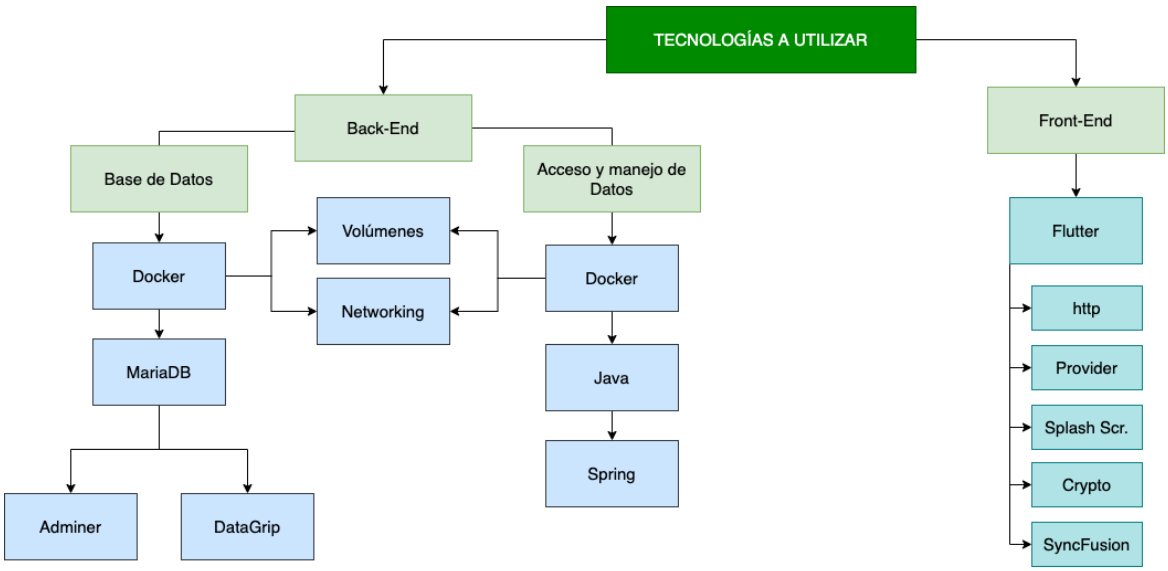
En cuanto a visión de negocio sobre el producto final, existe la intención de que esta misma aplicación pueda ser vendida a gabinetes psicológicos de

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 *HTTP (GET, POST, PATCH, DELETE)*, dependiendo de qué tipo de operaciones se desee hacer con los 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 (*front-end*).
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, 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.

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 este.

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.

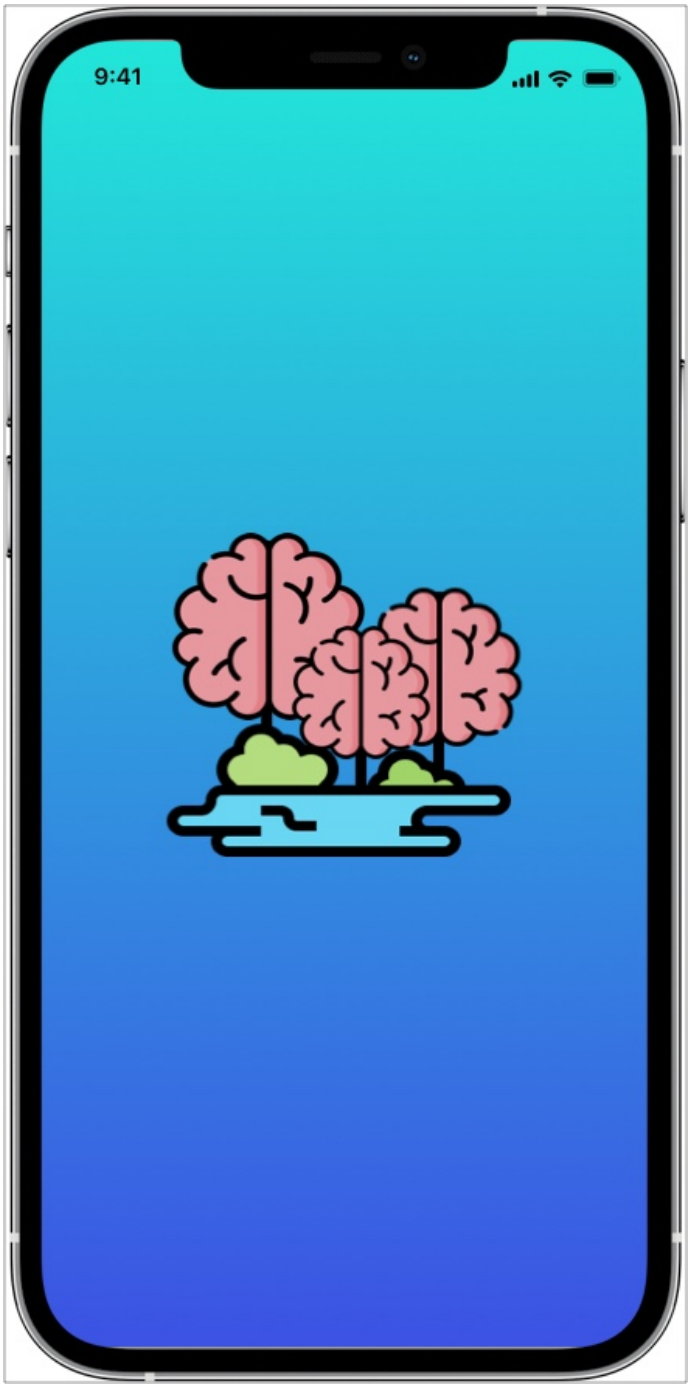
- **Pantalla de bienvenida**: pantalla principal en la cual se le da la bienvenida al usuario, mostrando alguna frase de motivación aleatoria con una estadística sobre el estado anímico semanal (media).
  Habrán enlaces con *routing* a las demás interfaces de la aplicación, así como a los ajustes.

- **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.

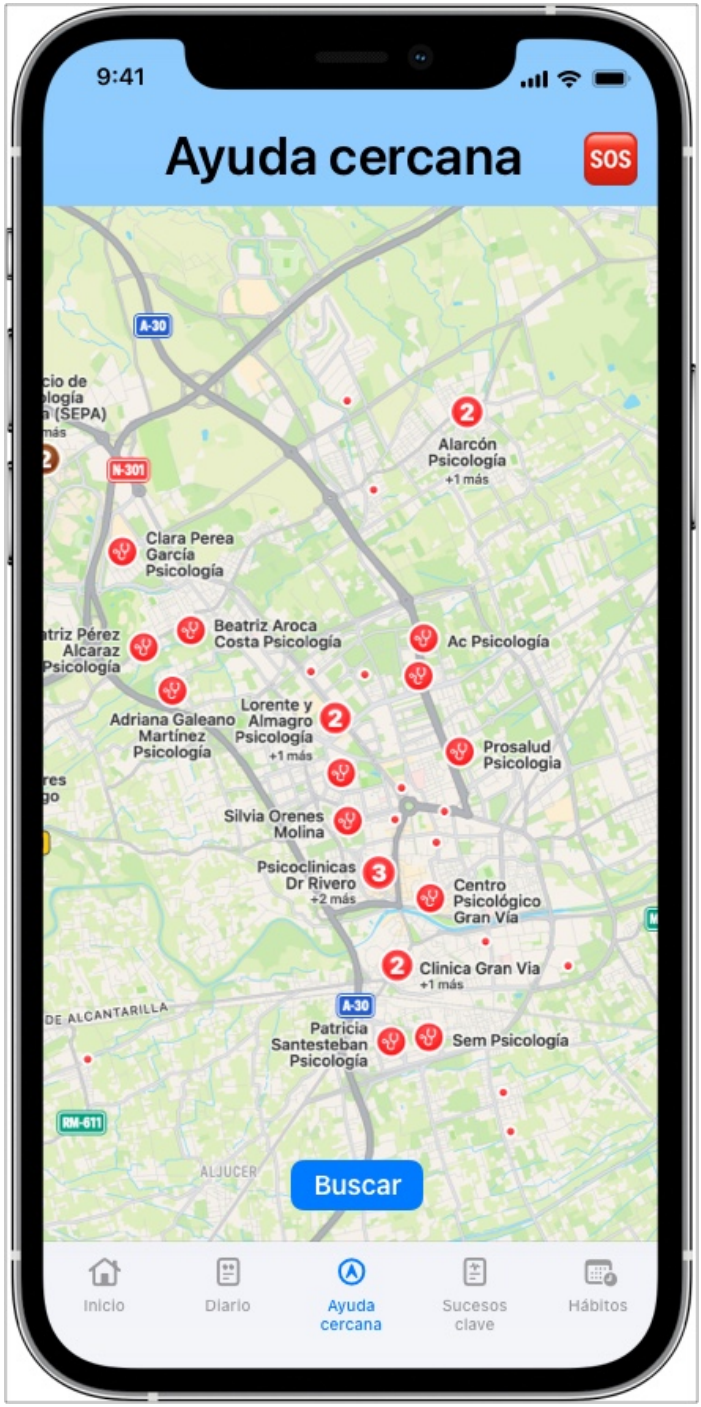
Prototipos.



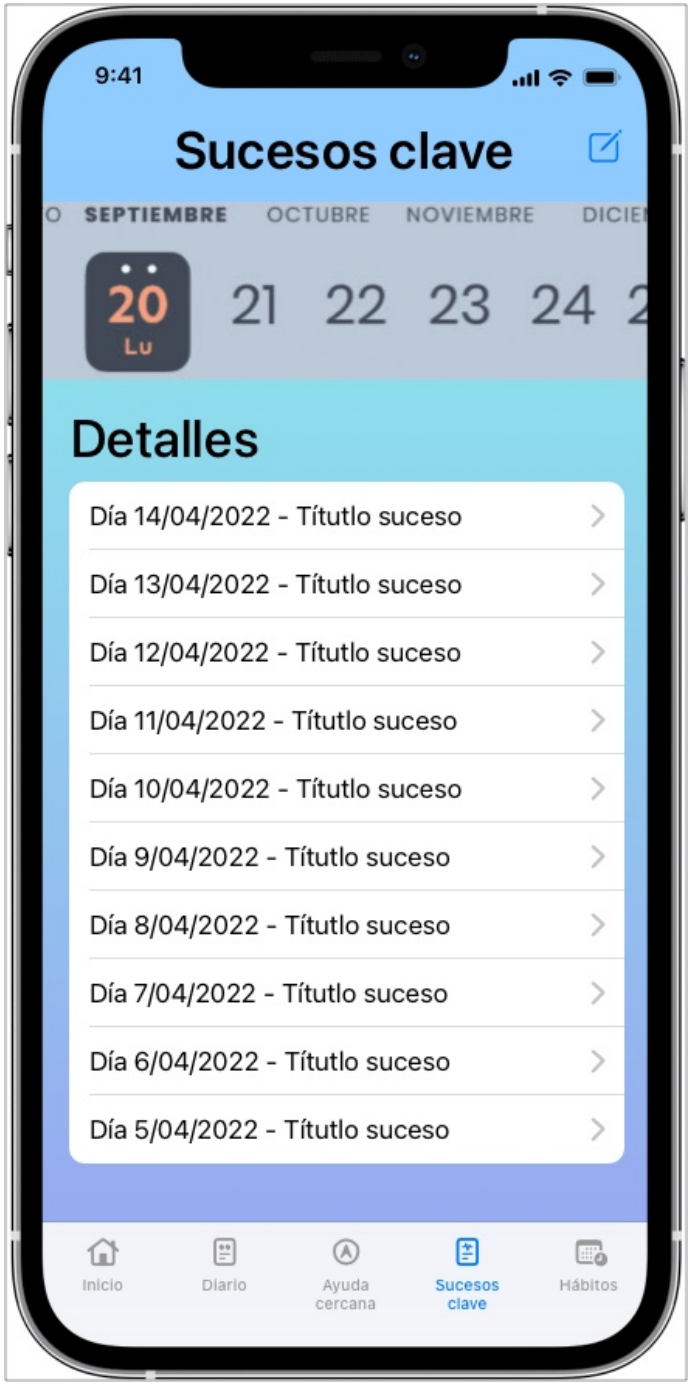
Prototipo 1. Splash Screen.

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.
  - edad : Integer.
  - ciudadNacimiento : String.
  - rol : Set de Rol.
- Rol.
  - id : Integer.
  - nombre : ERol.
- Entrada de diario.
  - id : Long.
  - usuario : Usuario (relación many to one).
  - titulo : String.
  - fechaCreacion : Date.
  - contenido : String.
- Suceso Clave.
  - id : Long.
  - usuario : Usuario (relación many to one).
  - titulo : String.
  - fechaCreacion : Date.

    - contenido : String.
- Hábito.
    - id : Long.
    - usuario : Usuario (relación many to one).
    - titulo : String.
    - fechaCreacion : String.
    - dias : Array de Date (?).

Endpoints del Servidor.

El servidor está estructurado de la manera en que para obtener los datos se ha de lanzar peticiones HTTP. Estas son:

Paquete de testing.

- Todos los usuarios (no registrados inclusive): /api/test/todos (GET)
- Usuarios (Registrados): /api/test/usuarios (GET)
- Especialistas: /api/test/especialistas (GET)
- Administrador: /api/test/administrador (GET)

Paquete de autenticación:

- signIn (Login): /api/auth/login (POST)
- signUp (Registro): /api/auth/signup (POST)
- logOut (Cerrar sesión): /api/auth/logout (POST)

Paquete de administración.

- Obtener datos de todos los usuarios: /api/admin/todosUsuarios (GET)
- Obtener usuario por ID: /api/admin/usuario/{idUsuario} (GET)
- Eliminar usuario por ID: /api/admin/usuario/{idUsuario} (DELETE)

Paquete entradas de diario (requiere autenticación).

- Obtener las entradas de diario: /api/diario/entradas (GET)
- Obtener entrada por ID: /api/diario/entradas/{idEntrada} (GET)
- Nueva entrada: /api/entradas/nueva (POST)

- Modificar entrada: /api/entradas/modificar/{idEntrada} (PATCH)
- Eliminar entrada: /api/entradas/eliminar/{idEntrada} (DELETE)

Paquete sucesos clave (requiere autenticación).

- Obtener los sucesos clave: /api/sclave/sucesos (GET)
- Obtener suceso clave por ID: /api/sclave/sucesos/{idSuceso} (GET)
- Crear suceso clave: /api/sclave/sucesos (POST)
- Modificar suceso clave: /api/sclave/sucesos/{idSuceso} (PATCH)
- Eliminar suceso clave: /api/sclave/sucesos/{idSuceso} (DELETE)

Paquete recordatorios (requiere autenticación).

- Obtener todos los recordatorios: /api/recordatorios/todos (GET)
- Obtener recordatorio por ID: /api/recordatorios/{idRecordatorio} (GET)
- Crear recordatorio: /api/recordatorios/nuevo (POST)
- Modificar recordatorio: /api/recordatorios/modificar/{idRecordatorio} (PATCH)
- Cambiar estado no-realizado/realizado: /api/recordatorios/realizado/{idRecordatorio} (PATCH)
- Eliminar recordatorio: /api/recordatorios/eliminar/{idRecordatorio} (DELETE)

Desarrollo del Back-end

El Backend ha sido desarrollado utilizando el lenguaje de programación Java, concretamente con el framework Spring.

Adicionalmente se han añadido las siguientes librerías:

- JPA: estas librerías (Java Persistance Api) son necesarias para poder utilizar los estándares de Jakarta y poder utilizar el ORM de Java por excelencia, Hibernate.

  Facilita enormemente la labor de desarrollo en el sentido de persistir los datos, siendo prescindible el crear DAO's y el manejo de consultas a Base de Datos.

- Spring WEB: esta librería es necesaria para poder exponer finalmente los endpoints al exterior de una manera sencilla.
  Además proporciona herramientas de gran utilidad a la hora de crear servlets y portlets.
- Spring Security: librería imprescindible para poder hacer filtrado de peticiones y gestión de roles dentro de la aplicación.
  A través de diferentes paquetes que contiene, se puede utilizar un filtrado dinámico a cada petición.
- Validation: librería que permite validar diferentes tipos de datos a nuestra propia conveniencia. Entre sus utilidades está el comprobar longitud de cadenas, tipos de datos, parseos automáticos desde JSON, etc.
- Json Web Token: método de encriptación de sesiones y usuarios utilizado globalmente en esta aplicación.
  En este caso, utilizo el algoritmo de encriptación HS256 (HMAC con SHA-256).
- MapStruct: librería que proporciona herramientas de mapeo para aplicar el patrón de diseño (o buena práctica) de utilizar DTO's (Data Transfer Objects).
  Mediante etiquetas, permite el cambio de entidad nativa a DTO y viceversa.
- JDBC: librería que implementa drivers de conexión a bases de datos desde dentro de nuestra aplicación.
  Permite comunicación con los principales motores de bases de datos, en este caso MySql con MariaDB.

Dicho desarrollo se ha llevado a cabo mediante el IDE (Integrated Development Enviroment) de la firma americana Jetbrains, Intelli-J.
Se ha elegido este IDE gracias a la numerosa cantidad de atajos y facilidades que proporciona al desarrollador, facilitando las labores de desarrollo al máximo y evitando perdidas de tiempo.

La estrategia de desarrollo del mismo ha consistido en tres fases; una primera centrada en la parte de usuarios, con sus respectivos login y asegurando el filtrado correcto de las peticiones, una segunda en el que se ha planteado el desarrollo de los diferentes endpoints a los que acceder de manera individualizada por usuario, y por último, una tercera donde se ha testado todo.
Se ha usado una estructura hexagonal para la organización de paquetes y clases Java.
Como hecho destacable del desarrollo de la parte del servidor, he de destacar la eficiencia y seguridad del mismo.
Utilizando Spring Security en adición a JWT, se ha creado un sistema muy eficiente de filtrado de peticiones en la que el usuario final nada más que tiene que consultar el endpoint deseado en función a los datos que quiera obtener, y el servidor es el encargado de devolver dichos datos pertenecientes al usuario gracias al análisis en tiempo real del JWT.

Desarrollo del Front-end

El front-end ha sido desarrollado utilizando el lenguaje de programación Dart, propietario de Google, con su framework de interfaces gráficas, Flutter.
Esta decisión ha sido tomada debido a que Flutter es un framework con una gran cuota de mercado para crear aplicaciones móviles multiplataforma (para Android e iOS) mediante la cual se puede crear una misma base de código para todas las plataformas, compartiendo controles y apariencia general.
Adicionalmente se han añadido algunas librerías:

- Provider: librería que permite la gestión de estados entre diferentes pantallas y escenarios dentro de la misma aplicación.
  Utilizados para traspaso de datos entre pantallas y servicios globales necesarios para el propio funcionamiento de la aplicación.

- SharedPreferences: librería que permite el almacenamiento de diferentes datos que precisemos en tiempo de almacenamiento en forma “key-value” dentro del almacenamiento del dispositivo.
- Intl: librería que permite transformar diferentes datos, tales como JSON’s, fechas, etc.
- Http: librería que nos permite lanzar peticiones HTTP a un servidor, en este caso, nuestro backend.
- FlutterSecureStorage: librería que permite almacenar datos de una forma segura en el almacenamiento de nuestro dispositivo, pero a diferencia de SharedPreferences, se utiliza mecanismos de criptografía para encriptar los datos que se pretenden guardar.
  Para iOS, utiliza KeyChain; para Android, utiliza AES con claves RSA.

Dicho desarrollo se ha llevado a cabo en el editor de texto plano Visual Studio Code, con diferentes extensiones que permiten la programación de una forma eficiente en este framework.
La estrategia de desarrollo ha consistido en dos fases; una primera donde se ha creado la parte relacionada con los usuarios (login, almacenamiento de propiedades) y ajustes, y una segunda, relacionada con la funcionalidad propia de la aplicación, permitiendo las diferentes operaciones con Entradas de Diario, Sucesos clave y Recordatorios.
Como hecho destacable del desarrollo me gustaría remarcar la creación dinámica de objetos transformados desde el JSON de entrada. Se ha precisado desarrollar manualmente ciertos mareos ya que los métodos “decode” y “encode” del propio lenguaje no propiciaban las características necesarias para este escenario.
Además, en alguna circustancia (p.e. el dialogo de eliminación de entradas de diario) se han desarrollado controles adaptativos en función de la plataforma sobre la que se esté ejecutando la aplicación.