

# 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

Índice	2
Desarrollo de la idea	3
Arquitectura de la aplicación	4
Tecnologías utilizadas.	5
Metodología de desarrollo.	7

## 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.

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 (teniendo que conceder permiso cada paciente), 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.

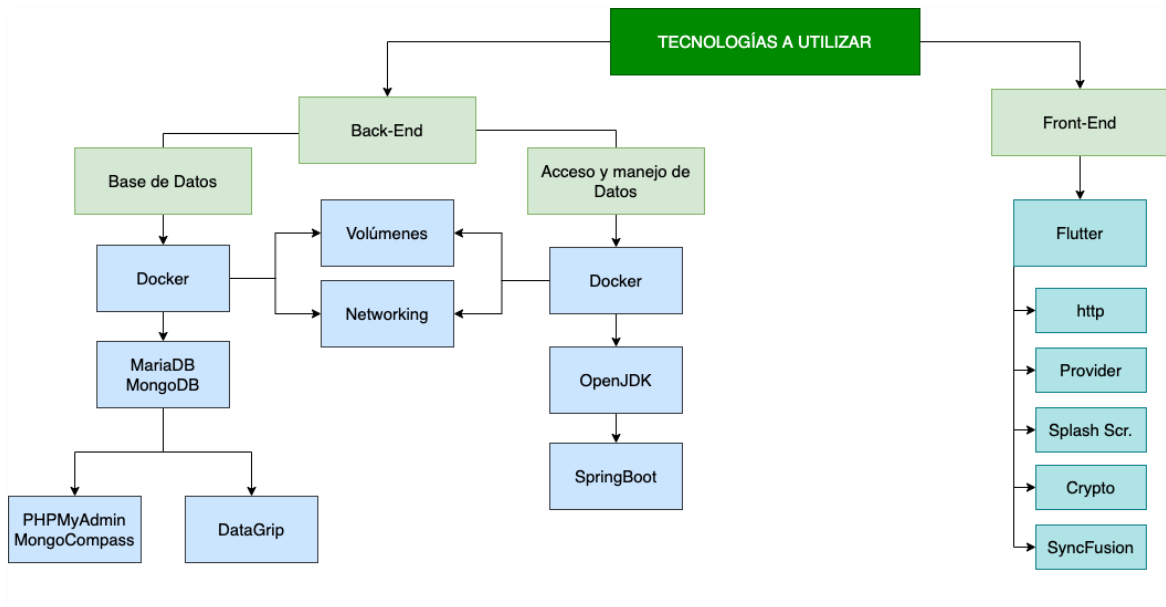
## Arquitectura de la aplicación

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.



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.

La decisión de usar *SQL* o *No-SQL* aún no ha sido tomada, pues se está analizando minuciosamente los requisitos para ver qué servicio se adapta mejor a las características del proyecto, pero en caso de ser *SQL*, será montado con el SGBD *PHPMyAdmin*; y en caso de ser *No-SQL* se utilizará con el SGBD *MongoDB Compass*.

Ambas opciones tendrán como cliente de monitorización y utilización *Datagrip*, de la compañía *JetBrains*.

Posteriormente, para la parte encargada del trato con los datos, parte la cual será accedida por el *frontend*, se creará con *Java SpringBoot*, un *framework* web el cual permite crear servicios *API-REST* y devolver los datos de la manera en que establezcamos.

Dicho servidor *Springboot* será *dockerizado*, creando una imagen expresa y propia. Versión del *JDK (Java Development Kit)* aún por determinar.

## 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.