***GestY***

***Por: Javier Garcia Ruiz***

[1. ÁNALISIS DEL PROBLEMA 3](#_Toc134729310)

[1.1 INTRODUCCION 3](#_Toc134729311)

[1.2 OBJETIVOS 3](#_Toc134729312)

[1.3 FUNCIONES Y RENDIMIENTOS DESEADOS 3](#_Toc134729313)

[1.4 PLANTEAMIENTO Y EVALUACION DE DIVERSAS SOLUCIONES 4](#_Toc134729314)

[1.5 JUSTIFICACION DE LA SOLUCION ELEGIDA 4](#_Toc134729315)

[1.6 MODEELADO DE LA SOLUCION 4](#_Toc134729316)

[1.7 PLANIFICACION TEMPORAL 5](#_Toc134729317)

[2. DISEÑO E IMPLEMENTACION DEL PROYECTO 5](#_Toc134729318)

[3. FASE DE PRUEBAS 12](#_Toc134729319)

[4. DOCUMENTACION DE LA APLICACIÓN 12](#_Toc134729320)

[4.1 INTRODUCCIÓN A LA APLICACIÓN 12](#_Toc134729321)

[4.2 MANUAL DE INSTALACIÓN 12](#_Toc134729322)

[4.3 MANUAL DE USUARIO 12](#_Toc134729323)

[4.4 MANUAL DE ADMINSTRACIÓN 12](#_Toc134729324)

[5. CONCLUSIONES FINALES 12](#_Toc134729325)

[6. BIBLIOGRAFIA 12](#_Toc134729326)

# ÁNALISIS DEL PROBLEMA

## INTRODUCCION

***GestY*** es una aplicación de gestión de empresas, donde podrás gestionar todo aquello necesario para organización de los quehaceres de un empleado dentro de su empresa. En *GestY* podrás gestionar: *Proyectos, Tareas y Empleados*, así como un apartado de mensajería instantánea entre empleados de la misma empresa, integrado dentro de la misma aplicación. También cuenta con un panel administrativo situado en la página oficial de *GestY.*

## OBJETIVOS

Con *GestY* buscamos la integración de todas las herramientas de gestión necesarias con el fin de facilitar el trabajo y la gestión de los empleados de las empresas asociadas a *GestY*. La integración de la mensajería instantánea dentro de la aplicación aporta un valor añadido a la aplicación, gracias a la falta de necesidad de alternar de aplicaciones para la comunicación con los empleados de la empresa.

## FUNCIONES Y RENDIMIENTOS DESEADOS

*GestY* cuenta con dos utilidades principales:

* Servidor: Situado en <https://gesty.devf6.es>, *GestY* cuenta con un panel administrativo desarrollado en el Framework web *Laravel* usando *PHP*, en el cual podremos encontrar a la empresa a la que pertenecemos, asi como un listado de tareas, empleados y la configuración de correo electrónico. Situado en el servidor *cloud* de la empresa *HiveMQ,* <https://www.hivemq.com/mqtt-cloud-broker/>, está alojado el servicio ***Mosquitto***(MQTT), servicio encargado de la gestión de la mensajería instantánea, concretamente en la *URL:*

<e7fa393ea4af4647a2482dffccd1d654.s2.eu.hivemq.cloud>

* Cliente: *GestY* cuenta con una versión para Ordenadores, desarrollado en *Java* con una interfaz grafica en Java Swift, así como una aplicación para Móviles Android, desarrollado en el Framework *Flutter* usando el lenguaje de programación *Dart.* Los clientes se conectan con una *API REST* situada en el servidor web de *GestY*, concretamente <https://gesty.devf6.es/api>.

## PLANTEAMIENTO Y EVALUACION DE DIVERSAS SOLUCIONES

A la hora de plantear el desarrollo de la aplicación, se buscó la forma de ampliar el rango de posibles problemas a solucionar, haciendo asi una más potente aplicación/infraestructura la cual presentase una fácil implementación en las empresas asociadas y fuera *User-Friendly* a la hora del uso de cualquiera de sus integraciones.

Se buscó una forma de centralización de servicios y datos, para realizar una aplicación con posible uso en grupos con datos disponibles para los usuarios de la aplicación.

## JUSTIFICACION DE LA SOLUCION ELEGIDA

Se valoró la idea de hacer en Java/Android la aplicación movil, pero se descartó definitivamente al ser Flutter una tecnología más actual y, a la vez, más potente con un desarrollo menor.

Para la aplicación de escritorio, se valoró la idea de desarrollarla junto a Flutter, gracias a que dispone de desarrollo multiplataforma, pero acabó desechándose por ser Flutter un lenguaje orientado a móviles, aunque disponga de desarrollo multiplataforma.

La idea inicial de centralizar todos los servicios en un servidor externo usando la tecnología Docker, creando contenedores propios y realizando una clusterización con Kubernetes (K8s), ha sido desechada para Producción de la aplicación ya que podría generar ralentización y fallas de seguridad de carácter grave o críticas. Sin embargo, se ha mantenido una dockerización los servicios necesarios para desarrollo y realización de pruebas en un entorno local, juntando PHP, MySQL y MQTT.

## MODEELADO DE LA SOLUCION

##### 1.6.1 RECURSOS HUMANOS

##### 1.6.2 RECURSOS HARDWARE

##### 1.6.3 RECURSOS SOFTWARE

## PLANIFICACION TEMPORAL

# DISEÑO E IMPLEMENTACION DEL PROYECTO

***Introducción a GestY***

*GestY* está montado sobre una ***infraestructura técnica robusta y muy potente***, buscando la ***solidez*** del traspaso de datos y almacenamientos de los mismos, asi como una ***usabilidad lo más sencilla pero potente posible***.

La infraestructura consta de:

* ***Servidor central / WebService***:

*Explicación Teórica:*

El servidor central es donde está corriendo el servidor web junto a la base de datos, siendo este servidor la pieza maestra de la infraestructura. Está montado sobre el dominio ***devf6.es*** gracias a que la empresa ***Área F5***, donde estoy cursando la Formación en Centros de Trabajo me ha cedido su espacio de desarrollo para el alojamiento de los servicios principales.

Dentro de este servidor estan corriendo los siguientes servicios:

* + ***Apache2 + PHP*** como Servidor Web:
  + ***MariaDB*** (v10.3.38) como Base de datos:

El servicio web está ejecutando una ***plataforma desarrollada en Laravel***, ya que Laravel tiene un funcionamiento muy robusto y potente a la vez, asi como una gran diversidad de librerías y utilidades que lo hacen un gran gestor de contenidos web (CMS). ***Laravel trabaja excelentemente con APIs***, tanto de cliente como de servidor.

El ***uso principal de Laravel*** en este proyecto ***es hacer de intermediario entre la aplicación cliente y la base de datos***; obteniendo, ordenando y preparando la información necesaria para la aplicación cliente.

Una de las principales ventajas de usar Laravel es su ***sistema de migraciones***. Este permite crear, de forma sencilla, la base de datos.

Laravel ***también hace de gestor de correos***, permitiendo crear, preparar y enviar correos gracias a su integración con el servicio SMTP y la facilidad de configuración del mismo.

Detallado de código

Laravel se instala a través de la herramienta *Composer*, un gestor de paquetes y librerías para PHP, el cual facilita en gran manera la instalación, tanto de Laravel como de librerías externas que necesitemos en nuestro proyecto.

*Composer* está basado en la Herramienta ***NodeJS***, la cual es necesaria para la instalación de *composer*, asi como de algunos paquetes que vamos a necesitar de ellos mas adelante en el desarrollo de esta aplicación.

Para descargar NodeJS, podremos hacerlo desde la página oficial de [*NodeJS*](https://nodejs.org/en), descargando también ***npm***, el gestor de paquetes de NodeJS.

Con *NodeJS* instalado, vamos a instalar [*composer*](https://getcomposer.org/) desde su página oficial.

GestY esta basado en la última versión de **Laravel 9** (Concretamente la versión **9.52.4**), siendo esta la última versión disponible a la hora del inicio del proyecto. Se ha desestimado actualizar a Laravel 10, la ultima versión, porque se valora que no merece la pena los cambios integrados en la nueva versión a cambio del tiempo invertido en el proceso de actualización.

El detallado de la instalación a fondo se encuentra en la [documentación de Laravel 9](https://laravel.com/docs/9.x/installation)

Para la instalación de esta versión de Laravel, vamos a usar el siguiente comando de Composer

**composer create-project laravel/laravel:^9.0 gesty**

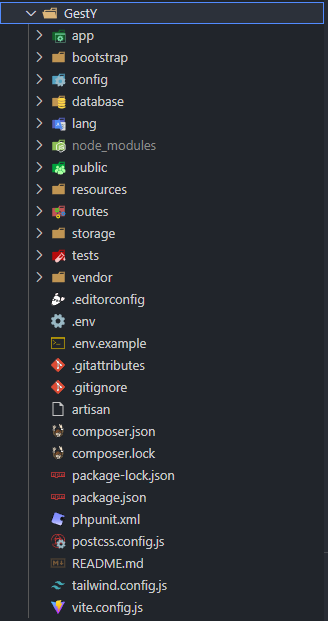
Este comando nos descargará una versión limpia de Laravel 9.

Junto a Laravel base, vamos a instalar dos paquetes más, ***TailwindCSS*** y ***Breeze.***

*TailwindCSS* es un Framework de CSS el cual nos permitirá desarrollar nuestras vistas de forma cómoda y sencilla, haciendo mucho más cómodo nuestro trabajo.

*Breeze* es un paquete desarrollado por el propio Laravel, el cual, al instalarlo, nos crea una estructura de usuarios básica, creando registros, inicio de sesión y toda una estructura de páginas en torno a un ámbito de usuarios.

Laravel cuenta con un complejo sistema de ficheros, los cuales necesitan unos de otros para el correcto funcionamiento de la web.

  
Este es el árbol de archivos de Laravel. Para que la aplicación funcione correctamente requiere de todos ellos. Dentro de este árbol nos encontramos el fichero principal, llamado ***artisan***, el cual será indispensable para el funcionamiento de nuestro proyecto.

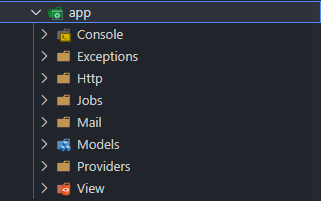
También cuenta con el fichero ***.env***, en el cual se encuentran variables de entorno, tales como credenciales a base de datos, configuración de correo, etc.

Para lanzar nuestro Laravel, únicamente tendremos que usar el siguiente comando:

***php artisan serve***

Para el desarrollo, nos centramos en las siguientes carpetas:

* + - * App
      * Database
      * Resources
      * Routes



La carpeta app cuenta con los ficheros núcleos de la aplicación. Aquí vamos a desarrollar todo aquello que tenga que ver con el funcionamiento de la aplicación, como controladores, modelos, peticiones, etc.

Las carpetas donde vamos almacenar nuestro desarrollo serán:

- Models

- Http

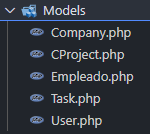
- Controllers

- Middlewares

- Requests

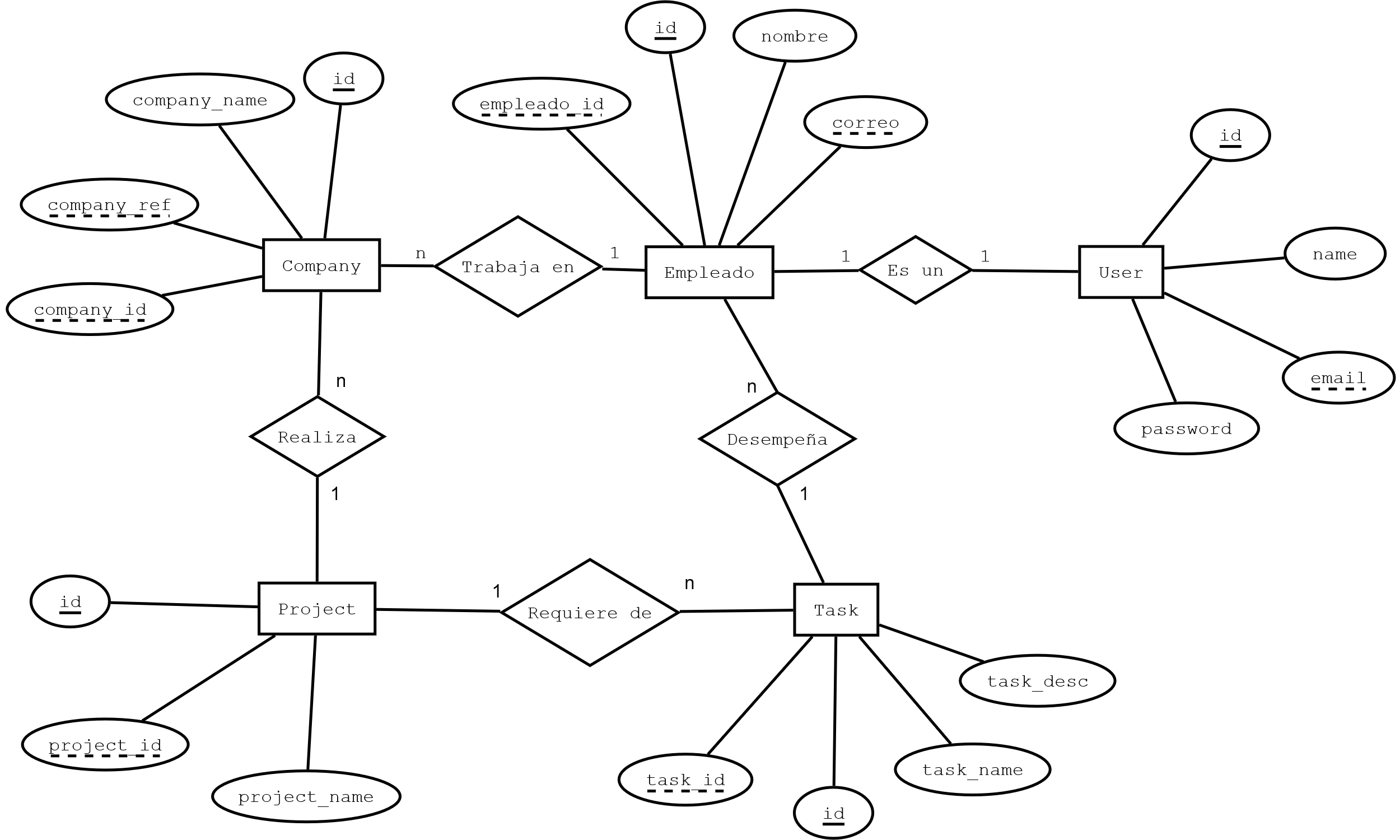
¿Qué es un modelo en Laravel?

Los modelos son clases que representan la interacción con la base de datos. Proporcionan una capa de abstracción entre la base de datos y la lógica de la aplicación, lo que permite realizar operaciones de creación, lectura, actualización y eliminación (CRUD) de datos de una manera fácil y estructurada.

Vamos a necesitar los siguientes modelos:

* Company
* Project (Llamado CProject por problema con una clase previamente creada)
* Empleado
* Task
* User (Creado por defecto)

Vamos a replicar el siguiente diagrama entidad/relación:



Todos los modelos desarrollados contienen una variable llamada *$fillable*, la cual contiene los campos que va a tener este modelo en base de datos que, junto a un fichero de ***migración***, situado en ***database/migrations***,generan la tabla pertinente en la base de datos.

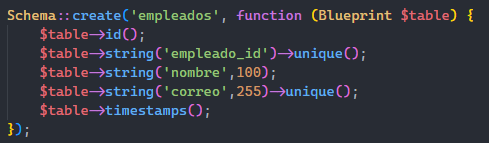
Para crear nuevas migraciones, vamos a hacerlo a través de un comando de artisan:

***php artisan make:migration CreateCompanyTask***

* Empleado:

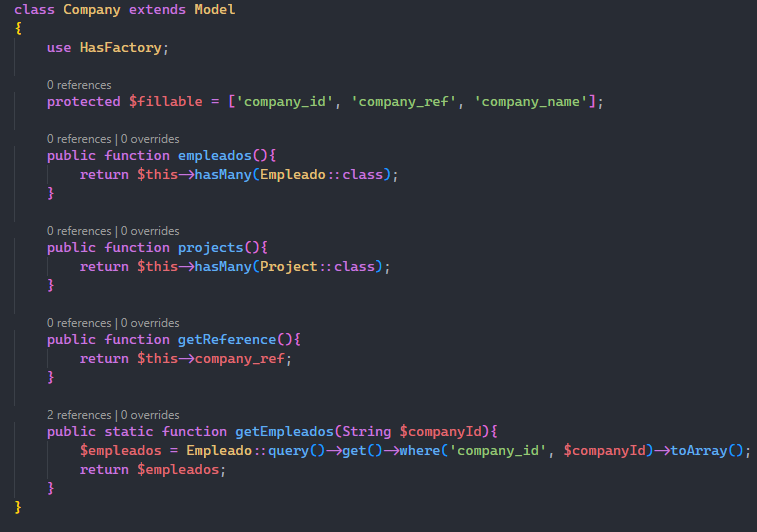


* Las funciones user() y company() devuelven el usuario y la empresa a la que pertenece el usuario.
* La función estática isEmpty() verifica si la colección pasa por parámetro esta vacía.



Esta es la migración para la creación de la tabla.

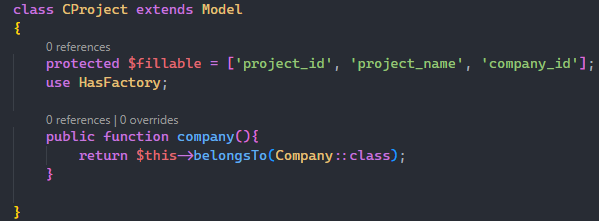
* Company:

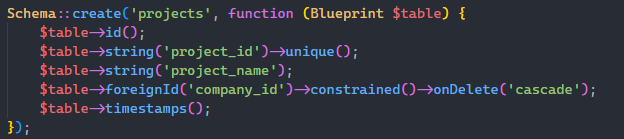


* Las funciones empleados() y projects() devuelven los empleados y los proyectos de una empresa.
* La función estática getEmpleados() devuelve los empleados de la empresa pasada como parámetro usando *company\_id*.

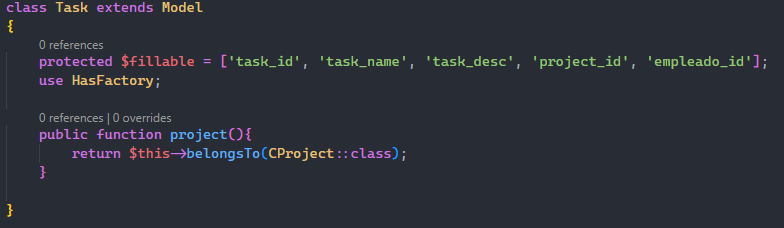


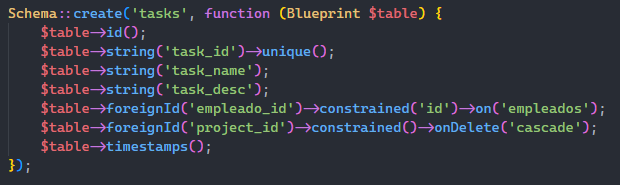
Esta es la migración de companies, junto a la creación de una fila por defecto.

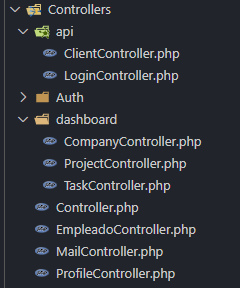
* Project: 
* La función company() devuelve la compañía a la que pertenece la empresa



* Task:







¿Qué es un controlador en Laravel?

Un controlador ayuda a la definición de toda la lógica de gestión de solicitudes, agrupando toda la lógica en una única clase. Estos controladores se crean con un comando de *artisan:*

***php artisan make:controller CompanyController***

Los controladores que se usa en el funcionamiento son los siguientes:

* *Api/ClientController.php*
* *Api/AdminController.php*
* *Dashboard/CompanyController.php*
* *Dashboard/ProjectController.php*
* *Dashboard/TaskController.php*
* *EmpleadoController.php*

Los controladores situados en la carpeta *api/* son aquellos que se encargan de las peticiones realizadas por las aplicaciones.

Imágenes

* ***Servidor cloud MQTT***:

*Explicación Teórica:*

***MQTT*** es un ***protocolo de mensajería*** basado en estándares, o un conjunto de reglas, la que se utiliza para comunicación de un equipo a otro. ***MQTT*** ***admite la mensajería entre dispositivos*** a la nube y la nube al dispositivo.

La idea inicial fue alojar este servicio en el servidor central, para poder asi reunir, dentro del mismo servidor, todos los servicios necesarios para el funcionamiento de la aplicación. Esto fue llevado a cabo en un primer momento, pero a la hora de investigar el funcionamiento de MQTT, fue desestimada, ya que necesitaba una infraestructura específica para el buen funcionamiento de esta herramienta.

Investigando como solucionar este problema, he decidido alojarlo en un servidor cloud, concretamente en <https://console.hivemq.cloud/>, el cual proporciona, de forma sencilla y gratuita, una infraestructura ideal para este servicio. Esto garantizará una mayor velocidad y seguridad a nuestros datos.

*HiveMQ* pondrá a nuestra disposición un clúster único para nuestro propósito, el cual contará con dos puertos abiertos, 8883 y 8884, los cuales usaremos para conectarnos al servicio. También usa un protocolo seguro para la conexión, el cual puede variar entre TLS y SSL según conveniencia. En nuestro caso usaremos SSL, al ser el más cómodo de usar.

Para la conexión deberemos de formular una URL válida para MQTT, contando con el protocolo, la dirección del clúster y el puerto de acceso, quedando algo asi:

*ssl://e7fa393ea4af4647a2482dffccd1d654.s2.eu.hivemq.cloud:8883*

Junto a esto, necesitaremos una cuenta registrada en el clúster.

MQTT fue diseñado para el envío de datos entre un usuario y un dispositivo inteligente, o denominado IoT (Internet of Things). La idea principal de MQTT consta en un envió unidireccional, el cual se pueda crear un flujo de datos que pudiera leer el dispositivo inteligente y usarlos. Los canales de datos se denominan *Topics*.

Para ello, MQTT posee un sistema de permisos único, el cual cuenta con 3 tipos:

* + *Solo publicación*
  + *Solo suscripción*
  + *Publicación y suscripción*

Estos permisos se asemejan al funcionamiento de un periódico convencional, siendo los *topics* los periódicos. Los usuarios con el permiso de *Solo publicación*, siendo la imprenta de nuestro periódico convencional, dispondrán del poder escribir dentro del *topic*, pero sin el poder de leer el contenido del mismo. Los usuarios con permiso *Solo suscripción* podrán leer el contenido del *topic*, pero sin poder publicar en el mismo.

Entendiendo este funcionamiento de los permisos de MQTT, y aprovechando que existe un permiso de *Publicación y suscripción*, si se crea un *topic* en el cual los usuarios tengan permisos de *Publicación y Suscripción*, creamos un chat bidireccional en grupo para los usuarios que esten dentro de una empresa.

* ***Cliente***:

*Explicación Teórica*

La aplicación cliente consta de dos versiones:

* + PC:

La aplicación para Ordenadores está basada en Java, usando como interfaz gráfica Java Swing.

Esta aplicación será la usada por los empleados, la cual constará con un inicio de sesión, asi como con un registro de empresa en el caso de que no tengas ninguna asignada con anterioridad.

La funcionalidad de la aplicación cliente en Java consta de una serie de peticiones HTTP, usando la librería ***Apache HttpClient*** a una API REST montada en el [***WebService***](#_DISEÑO_E_IMPLEMENTACION) la cual hará de intermediario entre la base de datos y la aplicación, ocupándose de obtener, organizar y preparar la información, devolviéndola en formato JSON, la cual será formateada de vuelta, usando la librería ***Google Gson***.

La aplicación cuenta con la librería ***Eclipse Paho MqttV5***, la cual nos permite, de forma sencilla, la integración y comunicación de MQTT con el servidor cloud.

Para que la aplicaicon pudiera ser exportada a un fichero ***.jar***, el cual se pudiera crear un ejecutable .exe, ha sido necesario añadir la librería ***maven-assembly-plugin***, la cual nos permite añadir, dentro del mismo fichero compilado, las dependencias necesarias para el correcto funcionamiento de la aplicación.

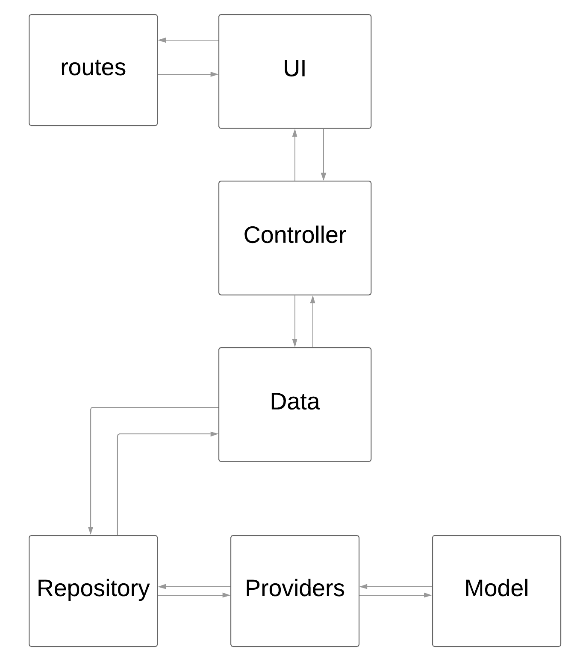
Para la creación del ejecutable .exe, se ha usado la versión gratuita de ***VMware InstallBuilder***.

* + Android:

La aplicación para móviles Android está basada en Dart, usando su propio Framework, Flutter, ambos desarrollados por ***Google***.

Esta aplicación está orientada a la administración de la misma, asi como para los jefes de la empresa. En ella podremos encontrar el listado de empresas, proyectos y tareas, asi como el listado de empleados que trabajan dentro de la misma.

La funcionalidad es similar a la aplicación Java, usando peticiones HTTP, usando la librería ***HTTP*** integrada en Flutter, a la API REST de nuestro [***WebService***](#_DISEÑO_E_IMPLEMENTACION) a través del cual obtendremos un listado de información en formato JSON, el cual formatearemos dentro de la aplicación usando el flujo de trabajo de Flutter.



Usando la librería ***GetX*** de Dart dispondremos de una serie de ***Providers, Consumers y Services***, los cuales nos facilitaran el desarrollo de la aplicación, asi como la obtención y utilización de la información obtenida a través de las peticiones a nuestro WebService.

Detallado de código

Imágenes

# FASE DE PRUEBAS

# DOCUMENTACION DE LA APLICACIÓN

## INTRODUCCIÓN A LA APLICACIÓN

## MANUAL DE INSTALACIÓN

## 4.3 MANUAL DE USUARIO

## 4.4 MANUAL DE ADMINSTRACIÓN

# CONCLUSIONES FINALES

# BIBLIOGRAFIA

1. <https://trello.com/b/L432pNQ3/gesty> -------> Pizarra digital de quehaceres del proyecto
2. <https://laravel.com/docs/9.x/>
3. <https://pub.dev/packages/mqtt_client>
4. <https://www.emqx.com/en/blog/how-to-use-mqtt-in-java>
5. <https://community.jitsi.org/t/integrate-jitsi-with-a-java-application/20351>
6. <https://pub.dev/packages/jitsi_meet>
7. <https://hub.docker.com/_/eclipse-mosquitto>
8. <https://www.php.net/manual/es/>
9. <https://stackoverflow.com/>
10. <https://chat.openai.com/>
11. <https://pub.dev/packages/get>
12. <https://console.hivemq.cloud/clients/java-hivemq?uuid=e7fa393ea4af4647a2482dffccd1d654>
13. <https://aws.amazon.com/es/what-is/mqtt/>
14. <https://es.wikipedia.org/wiki/Flutter_(software)>
15. <https://tailwindcss.com/docs/installation>
16. <https://www.udemy.com/course-dashboard-redirect/?course_id=2311106>
17. <https://www.udemy.com/course-dashboard-redirect/?course_id=2306140>
18. <https://www.udemy.com/course-dashboard-redirect/?course_id=1813098>
19. <https://dart.dev/>
20. <https://hc.apache.org/httpcomponents-client-5.2.x/>
21. <https://laracasts.com/>
22. <https://medium.com/>
23. <https://slidesgo.com/es/tema/propuesta-de-proyecto-web?variant=365#search-tecnologia&position-1&results-9&rs=search>
24. <https://www.emqx.io/docs/en/v4.4/development/java.html>
25. <https://community.openhab.org/t/solved-mqtt-connection-lost-with-mosquitto/32162/4>
26. <https://stackoverflow.com/questions/58360371/mqttasyncclient-client-is-not-connected-32104>
27. <https://www.hivemq.com/article/connecting-eclipse-paho-mqtt-java-client-hivemq-cloud-broker/>
28. <https://github.com/eclipse/paho.mqtt.java/blob/master/MQTTv5.md>
29. <https://github.com/eclipse/paho.mqtt.java/issues/679>
30. <https://app.quicktype.io/?l=dart>
31. <https://techglimpse.com/create-maven-jar-file-netbeans-tutorial/>
32. <https://howtodoinjava.com/maven/executable-jar-with-dependencies/>
33. <https://www.adictosaltrabajo.com/2009/11/12/maven-assembly-plugin-batch-process/>
34. <https://www.adictosaltrabajo.com/2009/03/29/maven-standalone-app/>
35. <https://www.redhat.com/es/topics/cloud-computing/what-are-cloud-services>
36. <https://nodejs.org/en>
37. <https://getcomposer.org/>
38. <https://talently.tech/blog/que-es-laravel/>