GestY (Gestión Ya)

PROYECTO FIN DE CICLO

Javier García Ruiz – 2º GSDAM

1. ÁNALISIS DEL PROBLEMA

1.1 INTRODUCCIÓN

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.*

1.2 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.

1.3 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.

1.4 PLANTEAMIENTO Y EVALUACIÓN 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.

1.5 JUSTIFICACIÓN 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.

1.6 MODELADO DE LA SOLUCIÓN

1.6.1 RECURSOS HUMANOS

He necesitado de 4 meses para el desarrollo de la aplicación, estimando 5/6 horas diarias para el desarrollo de la aplicación.

1.6.2 RECURSOS HARDWARE

He requerido de un portátil con S.O. Windows 11, asi como de un servidor web con S.O. Debian 10

1.6.3 RECURSOS SOFTWARE

He requerido de las siguientes aplicaciones para el desarrollo de la aplicación:

- Visual Studio Code
- Apache NetBeans
- Docker

- Composer
- NodeJS
- PHP
- HeidiSQL
- Laragon
- Firefox
- Postman
- MQTT.fx
- Dia
- Github Desktop
- Git
- VMware InstallBuilder
- Microsoft Publisher
- Microsoft Visio
- WireGuard

1.7 PLANIFICACION TEMPORAL

- Funcionalidad Java→ 100 hora/s
- Funcionalidad Flutter→ 60 hora/s
- Funcionalidad Laravel→ 80 hora/s
- Dockerización→ 5 hora/s
- Visual Java→ 35 hora/s
- Visual Flutter→ 1 hora/s
- Integración MQTT en Java→ 15 hora/s
- Montaje de servicio web→ 8 hora/s
- Montaje de servicio MQTT → 2 hora/s

2. DISEÑO E IMPLEMENTACIÓN 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*, descargando también ***npm***, el gestor de paquetes de NodeJS.

Con *NodeJS* instalado, vamos a instalar *composer* 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

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

composer create-project laravel/laravel:^9.0 gesty

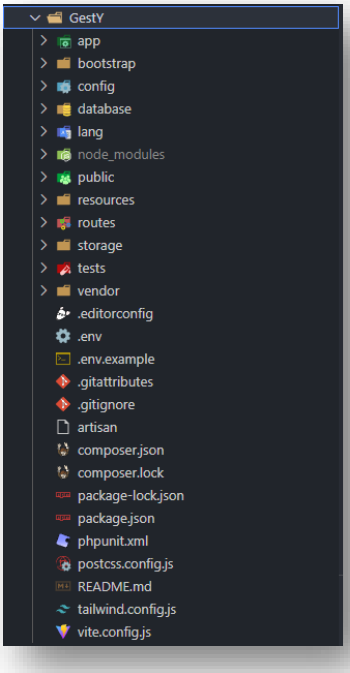
Este comando nos descargará una versión limpia de Laravel 9.

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.

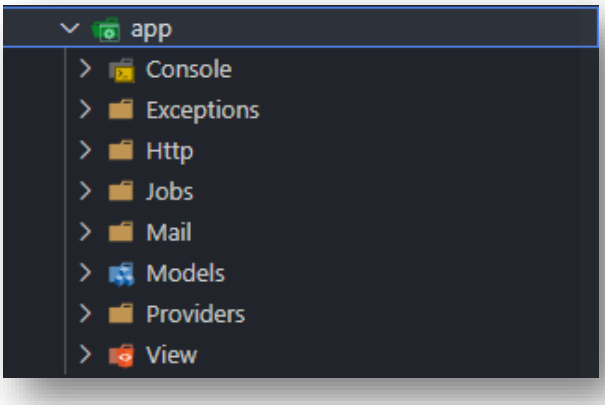
También cuenta con el fichero ***.env***, en el cual se encuentran variables de entorno, tales como credenciales a base de datos, configuración de correo, etc.
Para lanzar nuestro Laravel, únicamente tendremos que usar el siguiente comando:

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.



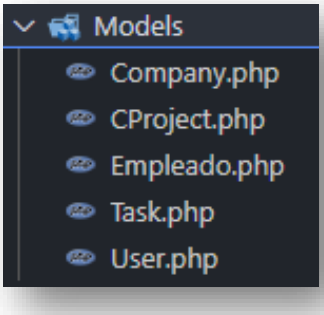
Las carpetas donde vamos almacenar nuestro desarrollo serán:

- Models
- Http
  - Controllers
  - Middlewares

¿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

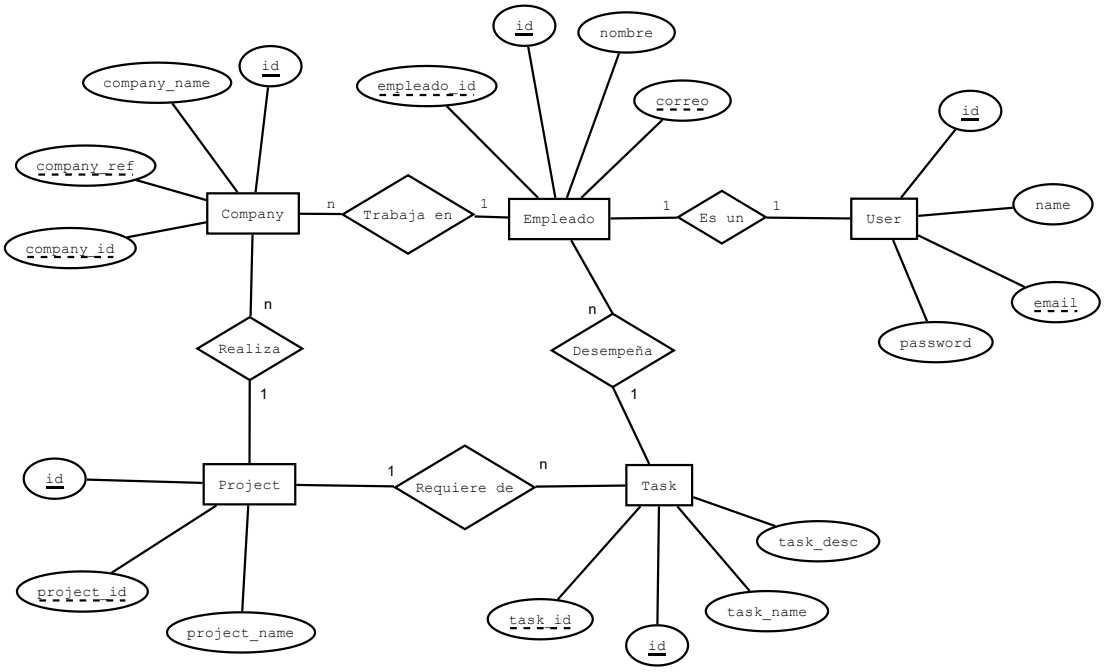
operaciones de creación, lectura, actualización y eliminación (CRUD) de datos de una manera fácil y estructurada.



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:

```php
class Empleado extends Model
{
    0 references
    protected $fillable = ['empleado_id', 'nombre', 'correo', 'company_id'];
    use HasFactory;

    0 references | 0 overrides
    public function user(){
        return $this->belongsTo(User::class);
    }

    0 references | 0 overrides
    public function company(){
        return $this->belongsTo(Company::class);
    }

    0 references | 0 overrides
    public function getId(){
        return $this->id;
    }

    2 references | 0 overrides
    public static function isEmpty(Collection $empleados){
        return $empleados->toArray() == [];
    }
}
```

- 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.

```php
Schema::create('empleados', function (Blueprint $table) {
    $table->id();
    $table->string('empleado_id')->unique();
    $table->string('nombre',100);
    $table->string('correo',255)->unique();
    $table->timestamps();
});
```

Esta es la migración para la creación de la tabla.

- Company:

```php
class Company extends Model
{
    use HasFactory;

    0 references
    protected $fillable = ['company_id', 'company_ref', 'company_name'];

    0 references | 0 overrides
    public function empleados(){
        return $this->hasMany(Empleado::class);
    }

    0 references | 0 overrides
    public function projects(){
        return $this->hasMany(Project::class);
    }

    0 references | 0 overrides
    public function getReference(){
        return $this->company_ref;
    }

    2 references | 0 overrides
    public static function getEmpleados(String $companyId){
        $empleados = Empleado::query()->get()->where('company_id', $companyId)->toArray();
        return $empleados;
    }
}
```

- 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*.

```php
Schema::create('companies', function (Blueprint $table) {
    $table->id();
    $table->string('company_id')->unique();
    $table->string('company_ref')->unique();
    $table->string('company_name');
    $table->timestamps();
});

DB::table('companies')->insert([
    'company_id' => 1,
    'company_ref' => 'cvs34s',
    'company_name' => 'Gesty Admin'
]);
```

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

- Project:

```php
class CProject extends Model
{
    0 references
    protected $fillable = ['project_id', 'project_name', 'company_id'];
    use HasFactory;

    0 references | 0 overrides
    public function company(){
        return $this->belongsTo(Company::class);
    }

}
```

  - La función company() devuelve la compañía a la que pertenece la empresa

```php
Schema::create('projects', function (Blueprint $table) {
    $table->id();
    $table->string('project_id')->unique();
    $table->string('project_name');
    $table->foreignId('company_id')->constrained()->onDelete('cascade');
    $table->timestamps();
});
```

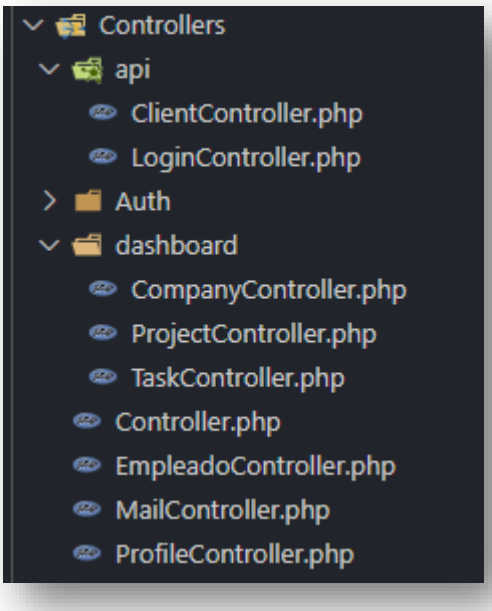
- Task:

```php
class Task extends Model
{
    0 references
    protected $fillable = ['task_id', 'task_name', 'task_desc', 'project_id', 'empleado_id'];
    use HasFactory;

    0 references | 0 overrides
    public function project(){
        return $this->belongsTo(CProject::class);
    }

}
```

  - La función project() devuelve el proyecto al que pertenece

```php
Schema::create('tasks', function (Blueprint $table) {
    $table->id();
    $table->string('task_id')->unique();
    $table->string('task_name');
    $table->string('task_desc');
    $table->foreignId('empleado_id')->constrained('id')->on('empleados');
    $table->foreignId('project_id')->constrained()->onDelete('cascade');
    $table->timestamps();
});
```

¿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.

Comenzando por los controladores que manejan modelos, estos cuentan con 7 funciones principales (***index, create, store, show, edit, update, destroy***), de las cuales vamos a usar dos o tres, asi como unas funciones que vamos a crear. Para no extender la documentación, no se incluirá las funciones vacías.

- CompanyController:

```php
public function index(User $user)
{

    if(Auth::user()->isAdmin()){
        $empresas = Company::all();
        return view('dashboard.companies.index', compact('empresas'));
    }else{
        $empleado = Empleado::query()->get()->where('id', Auth::user()->empleado_id)->first();
        $empresas = Company::query()->get()->where('id', $empleado->company_id)->first();
        return view('dashboard.companies.index', compact('empresas'));
    }
}
```

La función ***index*** verifica si el usuario autentificado es administrador o no, en el caso que sea, mostrará una vista de todas las empresas. Si no es administrador, se mostrará una vista con la empresa a la que el usuario es empleado.

```php
public function store(StoreRequest $request)
{
    $name = $request->input('company_name');
    $id = Random::generate(9, '0-9');
    $ref = Random::generate(6);

    $comp = Company::create([
        'company_id' => $id,
        'company_name' => $name,
        'company_ref' => $ref
    ]);

    return to_route('company.index');

}
```

La función ***store*** nos ayudará a crear una nueva empresa y, a su vez, una nueva fila en nuestra tabla en la base de datos. Esta función recoge como parámetro una petición *POST*, de la cual obtendremos el nombre de una nueva empresa. Se generará un id al azar de 9 números, asi como un patrón de 6 caracteres para la referencia.

- ProjectController:

```php
public function store(StoreRequest $request, String $companyId)
{
    $name = $request->input('project_name');
    $id = Random::generate(9, '0-9');

    $proj = DB::table('projects')->updateOrInsert([
        'project_id' => $id,
        'project_name' => $name,
        'company_id' => $companyId
    ]);

    return redirect()->back();

}
```

La función ***store*** ayuda a crear un nuevo proyecto, obteniendo de la petición POST el nombre del proyecto. Generaremos un identificador aleatorio de 9 números. Seguidamente, se creará un nuevo proyecto y se almacenará en la base de datos.

```php
public function show(string $companyId, string $projectId)
{
    $tasks = Task::query()->get()->where('project_id', $projectId);
    return view('dashboard.companies.projects.tasks.index', compact('tasks', 'projectId', 'companyId'));
}
```

La función ***show*** buscará todas las tareas que tenga el proyecto pasado por parámetro., mostrando una vista con una lista de todas ellas.

```php
public function getProjects(Request $request, String $companyId){
    //$companyId = $company->id;
    $projects = CProject::query()->from('projects')->get()->where('company_id', $companyId)->toArray();

    return view('dashboard.companies.projects.index', compact('companyId', 'projects'));
}
```

La función ***getProjects*** devuelve una vista con todos proyectos creados por una empresa.

- TaskController:

```php
public function store(StoreRequest $request, string $companyId, string $projectId)
{
    $name = $request->input('task_name');
    $desc = $request->input('task_desc');
    $proj = $request->input('project_id');
    $emp = $request->input('empleado_id');
    $id = Random::generate(9, '0-9');

    $task = Task::create([
        'task_id' => $id,
        'task_name' => $name,
        'task_desc' => $desc,
        'project_id' => $proj,
        'empleado_id' => $emp,
    ]);

    $tasks = Task::query()->get()->where('project_id', $projectId);

    return view('dashboard.companies.projects.tasks.index', compact('companyId', 'projectId', 'tasks'));

}
```

La función ***store*** ayuda a crear una nueva tarea, asi como una nueva fila en la correspondiente base de datos, obteniendo de la petición POST el nombre y la descripción de la nueva tarea, asi como el identificador del proyecto y del empleado asignado a la tarea.

```php
public function show(string $companyId, string $projectId, Task $Task)
{
    return view('dashboard.companies.projects.tasks.show', compact('companyId', 'projectId', 'Task'));
}
```

La función ***show*** mostrará una vista con la información de la tarea.

- EmpleadoController:

```php
public function index(int $companyId)
{
    $empleados = Empleado::query()->get()->where('company_id', $companyId);
    echo view('dashboard.companies.empleados.index', compact('companyId', 'empleados'));
}
```

La función ***index*** nos mostrará una vista con todos los empleados de una empresa.

¿Qué es un Middleware en Laravel?

Los *middlewares* son una parte fundamental del flujo de solicitud y respuesta de una aplicación.

Un *middleware* actúa como un filtro o una capa intermedia entre la solicitud HTTP entrante y la lógica de la aplicación, permitiendo realizar tareas específicas antes o después de que una solicitud alcance su destino final.

Para el correcto funcionamiento de nuestras aplicaciones, vamos a desarrollar 2 middleware, uno para el cliente Java denominado ***ClientMiddleware***, para empleados de las empresas, y uno para la administración de GestY para el cliente Movil denominado ***AdminMiddleware***.

Para la creación de nuevos *middlewares*, *artisan* nos proporciona un comando que nos creará un nuevo *middleware* con las funciones básicas.

php artisan make:middleware ClientMiddleware

- ClientMiddleware:

  Usará el siguiente token de autentificación:

  3sJak0orV9ysH0GBV2PZBDPqilBroEZI

```php
public function handle(Request $request, Closure $next)
{
    $token = env('CLIENT_API_TOKEN');

    if ($request->header('Authorization') !== $token) {
        return response()->json(['message' => 'Token inválido'], 401);
    }

    $response = $next($request);

    return $response;
}
```

Obtendremos de nuestro fichero ***.env*** el token de autentificación para los clientes. En el caso que la petición entrante no cuente con el token válido, se obtendrá un error de conexión, con un codigo HTTP 401 (*Unauthorized*), con el mensaje '*Token inválido*'

En caso de que el token sea válido, obtendrá un codigo HTTP 200, con mensaje 'OK' junto a los datos correspondientes.

- AdminMiddleware:

Usará el siguiente token de autentificación:

jklsdfias842nf9823nr893qr832rn89f

```php
public function handle(Request $request, Closure $next)
{
    $token = env('ADMIN_API_TOKEN');

    if ($request->header('Authorization') !== $token) {
        return response()->json(['message' => 'Token inválido'], 401);
    }

    $response = $next($request);

    return $response;
}
```

Mantiene el mismo funcionamiento que el *ClientMiddleware*, pero especializado en la aplicación administradora para Movil.

Para poder usar nuestros *middlewares*, deberemos de incorporarlos a la lista de middlewares usados por Laravel, situado en el fichero ***/app/Http/Kernel.php***:

```php
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'client' => \App\Http\Middleware\ClientMiddleware::class,
    'admin' => \App\Http\Middleware\AdminMiddleware::class,
];
```

Vistas y enrutamientos

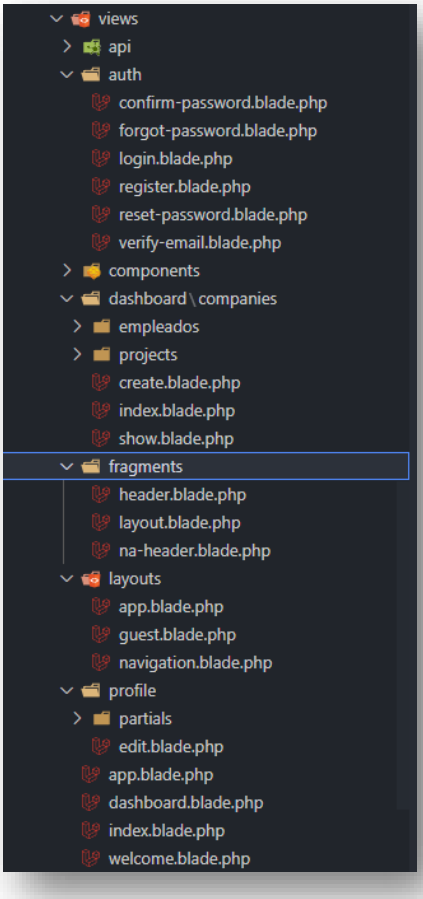
Tanto las vistas como las rutas son las partes principales para que la parte visual de nuestra página sea accesible.

Las vistas son una parte fundamental del sistema de plantillas que permite separar la lógica de presentación de la lógica de la aplicación. Proporcionan una forma estructurada y flexible de generar la salida HTML o cualquier otro tipo de respuesta que se envíe al cliente.

Las vistas en Laravel son ficheros *php*, pero utilizando un pequeño motor de vistas llamado Blade, el cual nos permite obtener, iterar y usar datos provenientes del *framework*. También cuenta con un complejo sistema de componentes, característica poderosa que te permite crear bloques de código reutilizables y modulares en tus vistas.

Para la creación de la página, han sido requeridos un total de 28 vistas, las cuales, 12 han sido creadas automáticamente por *Laravel Breeze*, siendo modificadas para ajustar tamaños, colores, tipografias, etc.

Este es el árbol de ficheros de las vistas, con aquellas que han sido creada o modificadas:



Gracias a las rutas, nuestra página puede seguir un flujo cómodo a la hora de ser utilizada, permiten definir cómo se manejan y responden las diferentes solicitudes HTTP que llegan a la aplicación.

Las rutas estan localizadas en la carpeta */routes/*, concretamente en los siguientes ficheros:

- api.php:

  Este fichero gestionará el enrutado de nuestra página sobre la dirección https://gesty.devf6.es/api, la cual es donde situaremos nuestros controladores.

```php
Route::middleware('client')->group( function() {
    Route::post('login-user', [LoginController::class, 'loginUser']);
    Route::post('register-user', [LoginController::class, 'registerUser']);
    Route::post('check-ref', [ClientController::class, 'checkRef']);
    Route::post('check-user-ref', [ClientController::class, 'checkUserRef']);
    Route::put('set-user-ref', [ClientController::class, 'setUserRef']);
    Route::post('get-company', [ClientController::class, 'getCompany']);
    Route::post('get-projects', [ClientController::class, 'getProjects']);
    Route::post('get-tasks', [ClientController::class, 'getTasks']);
});

Route::middleware('admin')->group( function() {
    Route::post('companies', [ClientController::class, 'companies']);
    Route::post('projects', [ClientController::class, 'projects']);
    Route::post('tasks', [ClientController::class, 'tasks']);
});
```

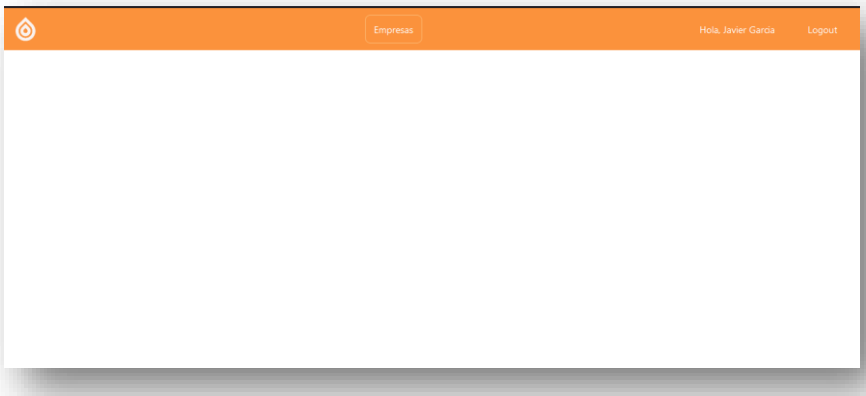
Aquí será donde entren en juego los *middlewares*, eligiendo la ruta a acceder y la función que se va a ejecutar.

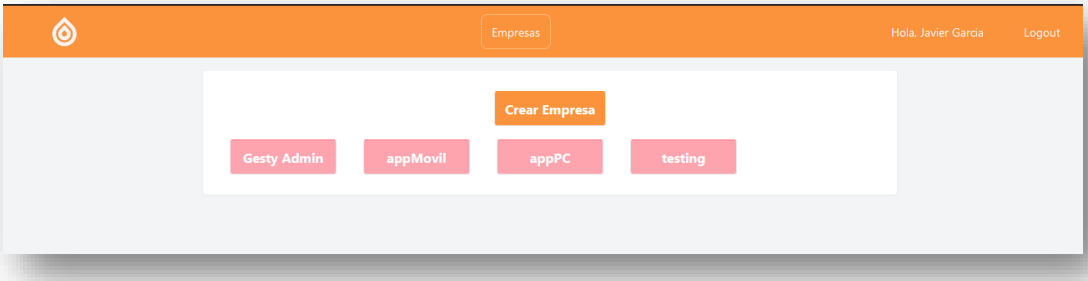
- web.php:

  Este fichero es el principal del enrutamiento, situando aquí las rutas usadas por el usuario o la propia página.

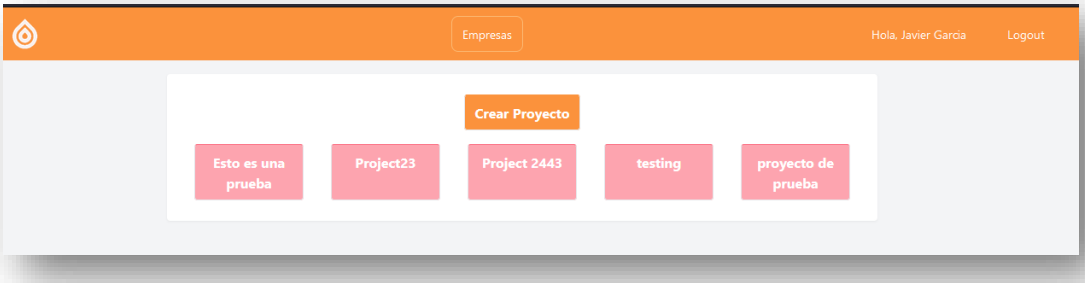
```php
Route::get('/', function () {
    return view('index');
});

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

Route::group(['prefix' => 'dashboard', 'middleware' => ['auth']], function() {
    Route::resources([
        'company' => CompanyController::class,
        'company/{company_id}/empleados' => EmpleadoController::class,
    ]);
    Route::resource('company/{company_id}/projects', ProjectController::class)->except('index');
    Route::post('company/{company_id}/projects/', [ProjectController::class, 'getProjects'])->name('projects.projects');
    Route::post('company/{company_id}/projects/create', [ProjectController::class, 'store'])->name('projects.store');
    Route::resource('company/{company_id}/projects/{project_id}/tasks', TaskController::class);
    Route::get('company/{company_id}/empleados/exportpdf', [EmpleadoController::class, 'getpdf']);
    Route::get('company/{company_id}/empleados/viewpdf', [EmpleadoController::class, 'streampdf']);
    Route::get('company/{company_id}/mail/sendmail', [MailController::class, 'sendmail']);
    Route::resource('mail', MailController::class)->only(['index']);
});

Route::middleware('auth')->group(function () {      Javier 'Glyaxz' García, hace 2 meses • ADD new Structure …
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});

require __DIR__.'/auth.php';
```

Imágenes


Empresas
Hola, Javier Garcia
Logout


Empresas
Hola, Javier Garcia
Logout
Crear Empresa
Gesty Admin
appMovil
appPC
testing


Empresas
Hola, Javier Garcia
Logout
IDENTIFICADOR
NOMBRE DE EMPRESA
CÓDIGO DE REFERENCIA
ACCIONES
1
Gesty Admin
cvs34s
Ver Projectos
Ver Empleados


Empresas
Hola, Javier Garcia
Logout
Crear Proyecto
Esto es una prueba
Project23
Project 2443
testing
proyecto de prueba

- ***Servidor cloud MQTT***:

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*** 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 aplicación 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***.

Detallado de código

El cliente desarrollado en Java está basado en un proyecto Maven.

En el fichero *pom.xml* se ha añadido las dependencias necesitadas en el proyecto, asi como un bloque necesario para la exportación

del proyecto hacia un ejecutable que incluya las dependencias dentro de él.

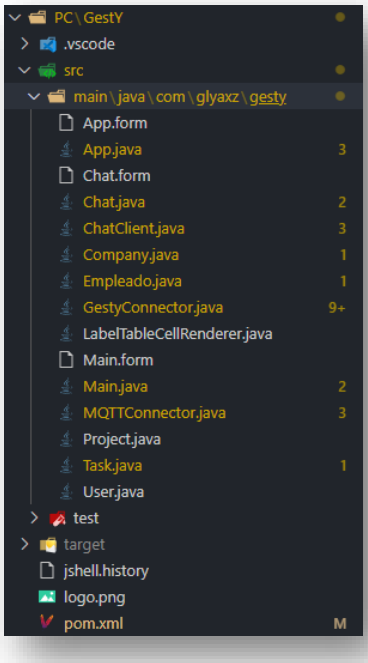
```xml
<dependencies>
    <dependency>
        <groupId>org.jdesktop</groupId>
        <artifactId>swing-layout</artifactId>
        <version>1.0.2</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.10.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.httpcomponents.client5</groupId>
        <artifactId>httpclient5</artifactId>
        <version>5.2.1</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>io.github.vincenzopalazzo</groupId>
        <artifactId>material-ui-swing</artifactId>
        <version>1.1.4</version>
    </dependency>
    <dependency>
        <groupId>org.eclipse.paho</groupId>
        <artifactId>org.eclipse.paho.mqttv5.client</artifactId>
        <version>1.2.5</version>
    </dependency>
</dependencies>
```

En esta sección de nuestro fichero *pom.xml*, en el cual contamos con las siguientes dependencias:

- *Swing-layout*, desde un repositorio externo.
- *GSON*
- *HTTPClient5*
- *SLF4J*
- *Material UI Swing,* siendo este el Look&Feel usado en la aplicación.
- *Paho MQTTv5 Client*, desde un repositorio externo.

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>3.3.0</version>
            <configuration>
                <descriptorRefs>
                    <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
                <archive>
                    <manifest>
                        <mainClass>com.glyaxz.gesty.Main</mainClass>
                    </manifest>
                </archive>
            </configuration>
            <executions>
                <execution>
                    <id>make-assembly</id>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Esta sección del fichero *pom.xml* configuraremos el empaquetamiento de la aplicación a un ejecutable externo incluyendo todas las dependencias necesarias, para ello vamos a usar un plugin de *Maven.*
Este es el árbol de ficheros de nuestro cliente Java:



Dentro del paquete, disponemos de 3 ficheros con paneles:

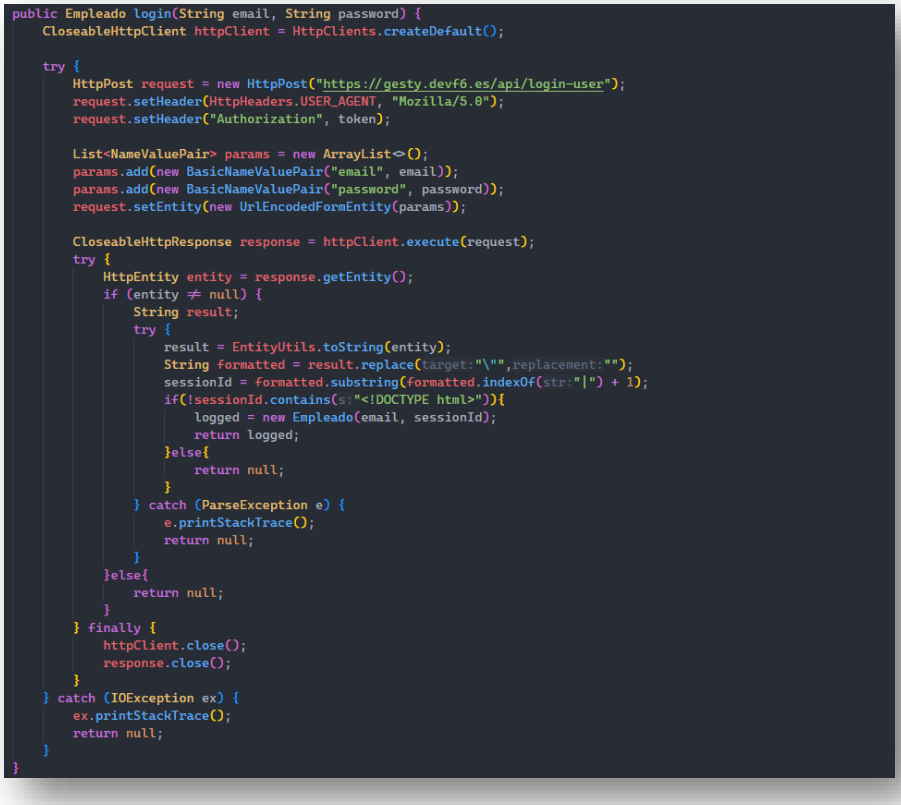
- *Main*
- *App*
- *Chat*

Junto a los ficheros con una parte visual, disponemos de los siguientes ficheros:

- *Company*
- *Project*
- *Task*
- *User*
- *Empleado*
- *GestyConnector*
- *MQTTConnector*
- *ChatClient*
- *LabelTableCellRenderer*

Junto a los objetos básicos, la aplicación obtiene los datos desde el servidor principal gracias a la clase ***GestyConnector***. Este gestiona las peticiones HTTP para la obtención, filtrado y procesado de los datos pertinentes.

En esta clase, se usará la librería ***HTTPClient5***, la cual nos permitirá hacer peticiones complejas de una forma sencilla e intuitiva, pudiendo añadir valores en el cuerpo de la petición, *tokens* de autentificación, etc.

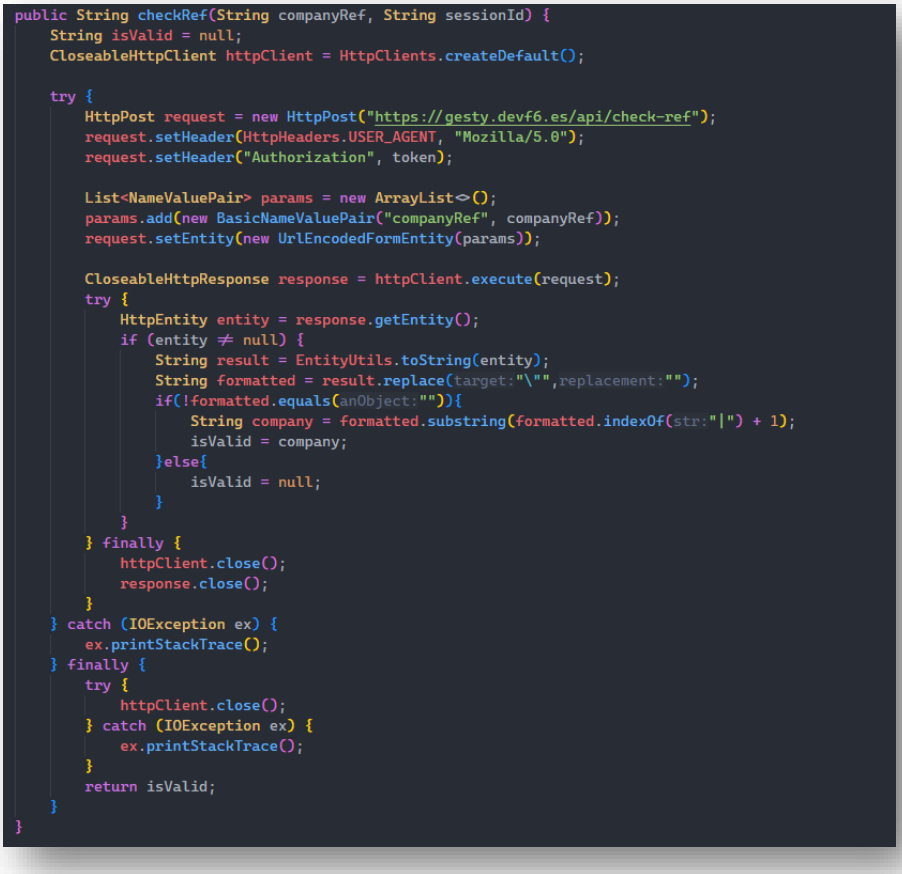


```java
public Empleado login(String email, String password) {
    CloseableHttpClient httpClient = HttpClients.createDefault();

    try {
        HttpPost request = new HttpPost("https://gesty.devf6.es/api/login-user");
        request.setHeader(HttpHeaders.USER_AGENT, "Mozilla/5.0");
        request.setHeader("Authorization", token);

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("email", email));
        params.add(new BasicNameValuePair("password", password));
        request.setEntity(new UrlEncodedFormEntity(params));

        CloseableHttpResponse response = httpClient.execute(request);
        try {
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                String result;
                try {
                    result = EntityUtils.toString(entity);
                    String formatted = result.replace(target:"\"",replacement:"");
                    sessionId = formatted.substring(formatted.indexOf(str:"|") + 1);
                    if(!sessionId.contains(s:"<!DOCTYPE html>")){
                        logged = new Empleado(email, sessionId);
                        return logged;
                    }else{
                        return null;
                    }
                } catch (ParseException e) {
                    e.printStackTrace();
                    return null;
                }
            }else{
                return null;
            }
        } finally {
            httpClient.close();
            response.close();
        }
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
}
```

La función login nos permite verificar si el usuario es válido. Para ello realiza una petición por POST a https://gesty.devf6.es/api/login-user, adjuntando el token de autentificación en la cabecera de la petición.

En el cuerpo de la petición, se añade el correo y la contraseña.

En el caso que las credenciales introducidas sean válidas, devuelve un Empleado con el correo previamente introducido y un identificador de sesión único como comprobante de un correcto inicio de sesión, en caso contrario, se devolverá un Empleado nulo.

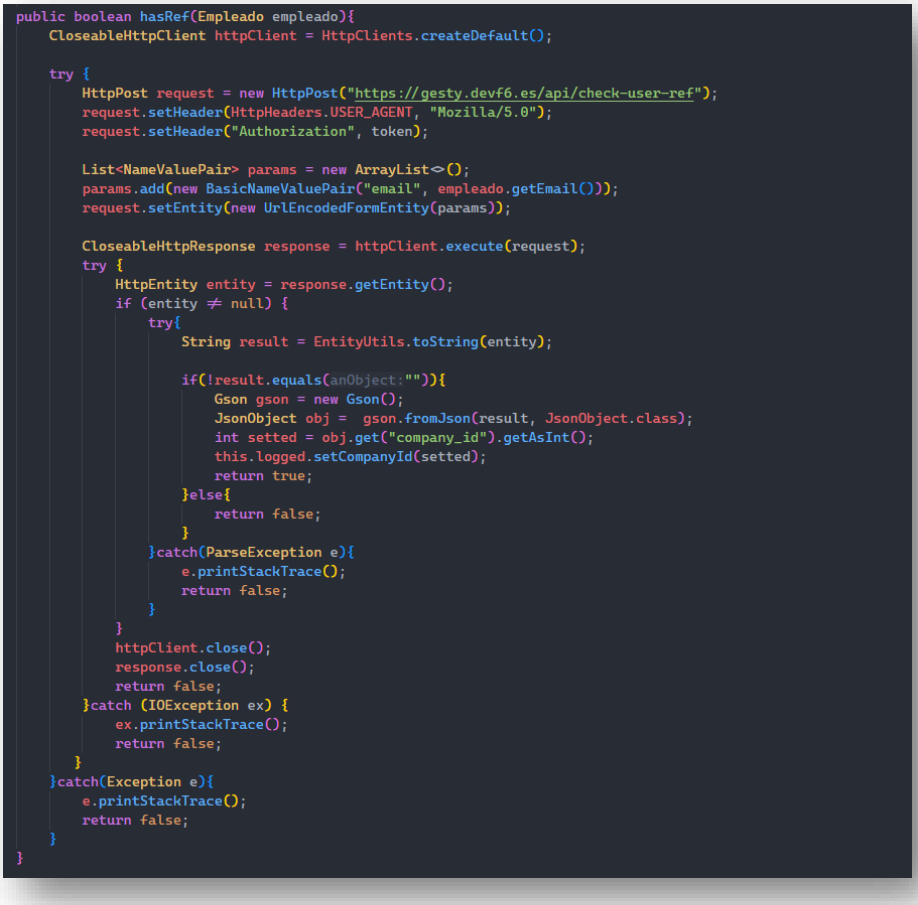


```java
public String checkRef(String companyRef, String sessionId) {
    String isValid = null;
    CloseableHttpClient httpClient = HttpClients.createDefault();

    try {
        HttpPost request = new HttpPost("https://gesty.devf6.es/api/check-ref");
        request.setHeader(HttpHeaders.USER_AGENT, "Mozilla/5.0");
        request.setHeader("Authorization", token);

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("companyRef", companyRef));
        request.setEntity(new UrlEncodedFormEntity(params));

        CloseableHttpResponse response = httpClient.execute(request);
        try {
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                String result = EntityUtils.toString(entity);
                String formatted = result.replace(target:"\"",replacement:"");
                if(!formatted.equals(anObject:"")){
                    String company = formatted.substring(formatted.indexOf(str:"|") + 1);
                    isValid = company;
                }else{
                    isValid = null;
                }
            }
        } finally {
            httpClient.close();
            response.close();
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    } finally {
        try {
            httpClient.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return isValid;
    }
}
```

Esta función nos permite verificar si una referencia de compañía existe y es válida, para ello realiza una petición por POST a https://gesty.devf6.es/api/check-ref, adjuntando el token de autentificación en la cabecera de la petición.

En el cuerpo de la petición, se añade la referencia de la compañía. Se añade también el identificador de sesión, aunque no se use, ya que puede llegar a ser útil en una futura mejora de la función, en búsqueda de una superior seguridad.

Si la referencia es válida, se devuelve el nombre de la compañía como una cadena de caracteres a la que pertenece, en caso contrario devuelve una cadena vacía.

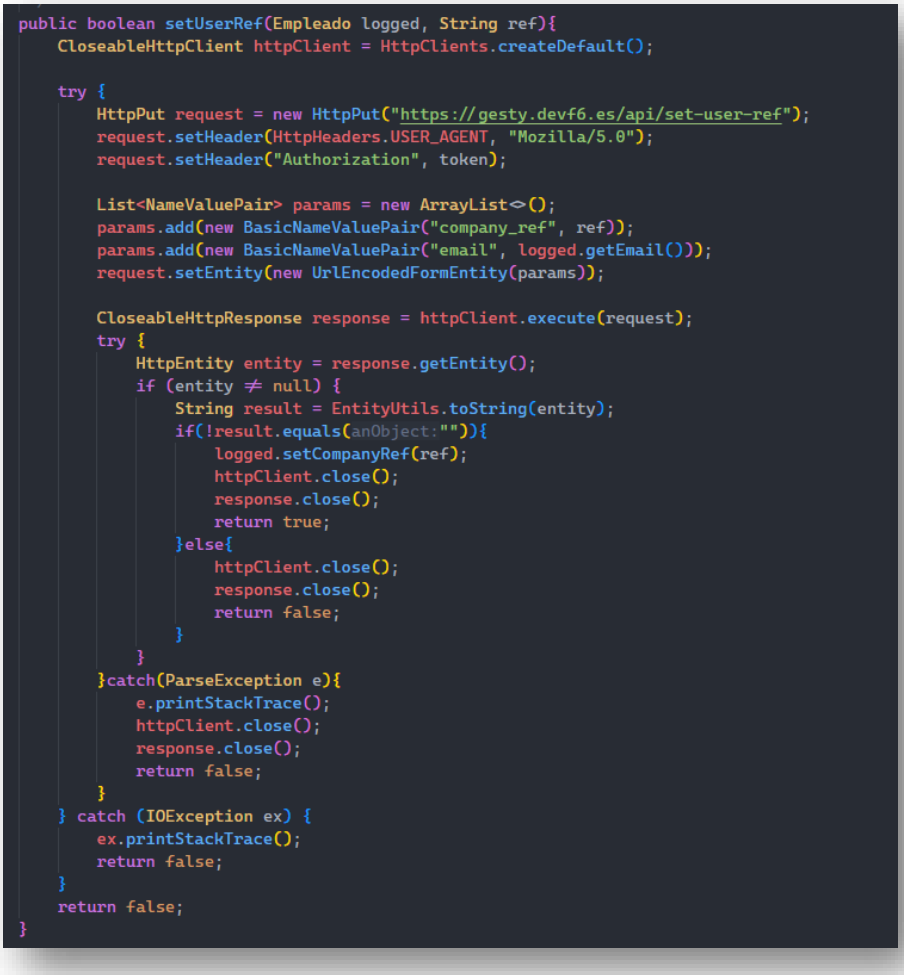


```java
public boolean hasRef(Empleado empleado){
    CloseableHttpClient httpClient = HttpClients.createDefault();

    try {
        HttpPost request = new HttpPost("https://gesty.devf6.es/api/check-user-ref");
        request.setHeader(HttpHeaders.USER_AGENT, "Mozilla/5.0");
        request.setHeader("Authorization", token);

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("email", empleado.getEmail()));
        request.setEntity(new UrlEncodedFormEntity(params));

        CloseableHttpResponse response = httpClient.execute(request);
        try {
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                try{
                    String result = EntityUtils.toString(entity);

                    if(!result.equals(anObject:"")){
                        Gson gson = new Gson();
                        JsonObject obj =  gson.fromJson(result, JsonObject.class);
                        int setted = obj.get("company_id").getAsInt();
                        this.logged.setCompanyId(setted);
                        return true;
                    }else{
                        return false;
                    }
                }catch(ParseException e){
                    e.printStackTrace();
                    return false;
                }
            }
            httpClient.close();
            response.close();
            return false;
        }catch (IOException ex) {
            ex.printStackTrace();
            return false;
        }
    }catch(Exception e){
        e.printStackTrace();
        return false;
    }
}
```

Esta función nos permite verificar si un usuario cuenta con una referencia de compañía ya configurada. Para ello, realiza una petición por POST a https://gesty.devf6.es/api/check-user-ref, adjuntando el token de autentificación en la cabecera de la petición.

En el cuerpo de la petición, se añade el correo del empleado a verificar.

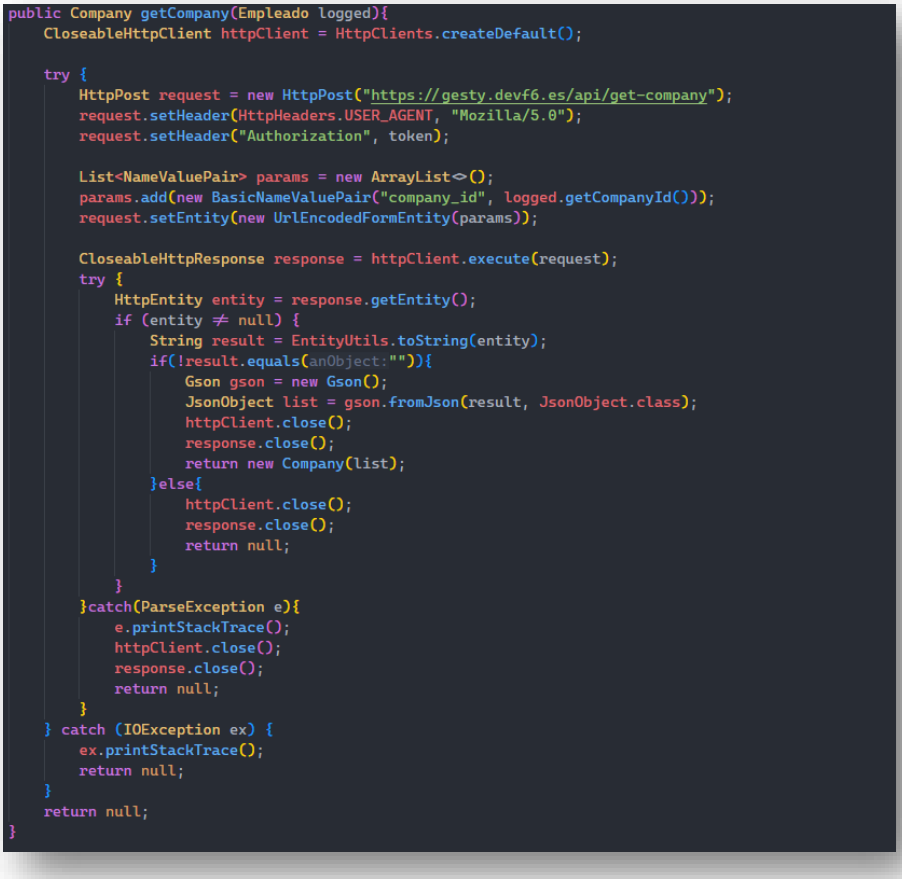
En el caso que tenga una referencia asignada, devuelve *true*, en caso contrario, devuelve *false*.



```java
public boolean setUserRef(Empleado logged, String ref){
    CloseableHttpClient httpClient = HttpClients.createDefault();

    try {
        HttpPut request = new HttpPut("https://gesty.devf6.es/api/set-user-ref");
        request.setHeader(HttpHeaders.USER_AGENT, "Mozilla/5.0");
        request.setHeader("Authorization", token);

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("company_ref", ref));
        params.add(new BasicNameValuePair("email", logged.getEmail()));
        request.setEntity(new UrlEncodedFormEntity(params));

        CloseableHttpResponse response = httpClient.execute(request);
        try {
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                String result = EntityUtils.toString(entity);
                if(!result.equals(anObject:"")){
                    logged.setCompanyRef(ref);
                    httpClient.close();
                    response.close();
                    return true;
                }else{
                    httpClient.close();
                    response.close();
                    return false;
                }
            }
        }catch(ParseException e){
            e.printStackTrace();
            httpClient.close();
            response.close();
            return false;
        }
    } catch (IOException ex) {
        ex.printStackTrace();
        return false;
    }
    return false;
}
```

Esta función nos permite configurarle una referencia de empresa a un empleado. Para ello, realiza una petición por POST a https://gesty.devf6.es/api/set-user-ref, adjuntando el token de autentificación en la cabecera de la petición.

En el cuerpo de la petición, se añade el correo del empleado a configurar, asi como la referencia asignada.

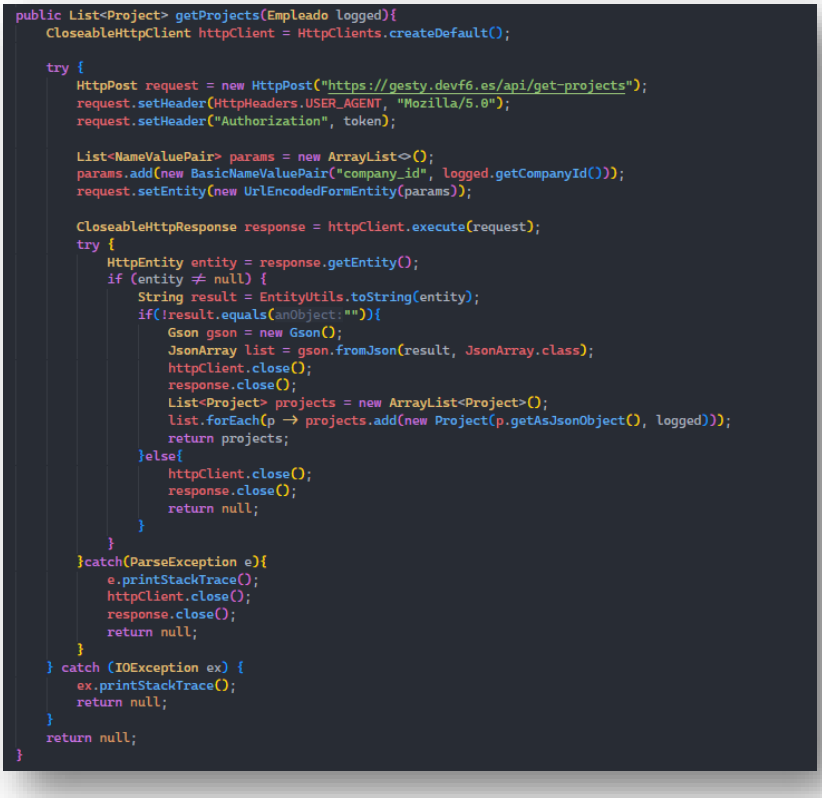


```java
public Company getCompany(Empleado logged){
    CloseableHttpClient httpClient = HttpClients.createDefault();

    try {
        HttpPost request = new HttpPost("https://gesty.devf6.es/api/get-company");
        request.setHeader(HttpHeaders.USER_AGENT, "Mozilla/5.0");
        request.setHeader("Authorization", token);

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("company_id", logged.getCompanyId()));
        request.setEntity(new UrlEncodedFormEntity(params));

        CloseableHttpResponse response = httpClient.execute(request);
        try {
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                String result = EntityUtils.toString(entity);
                if(!result.equals(anObject:"")){
                    Gson gson = new Gson();
                    JsonObject list = gson.fromJson(result, JsonObject.class);
                    httpClient.close();
                    response.close();
                    return new Company(list);
                }else{
                    httpClient.close();
                    response.close();
                    return null;
                }
            }
        }catch(ParseException e){
            e.printStackTrace();
            httpClient.close();
            response.close();
            return null;
        }
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
    return null;
}
```

Esta función es usada para obtener la empresa asignada a un usuario en específico. Para ello, realiza una petición por POST a https://gesty.devf6.es/api/get-company, adjuntando el token de autentificación en la cabecera de la petición.

En el cuerpo de la petición, se añade el correo del empleado del cual queremos obtener la empresa asignada.

Cuando la petición es correcta, devuelve una nueva empresa, en caso contrario, devuelve una empresa nula.

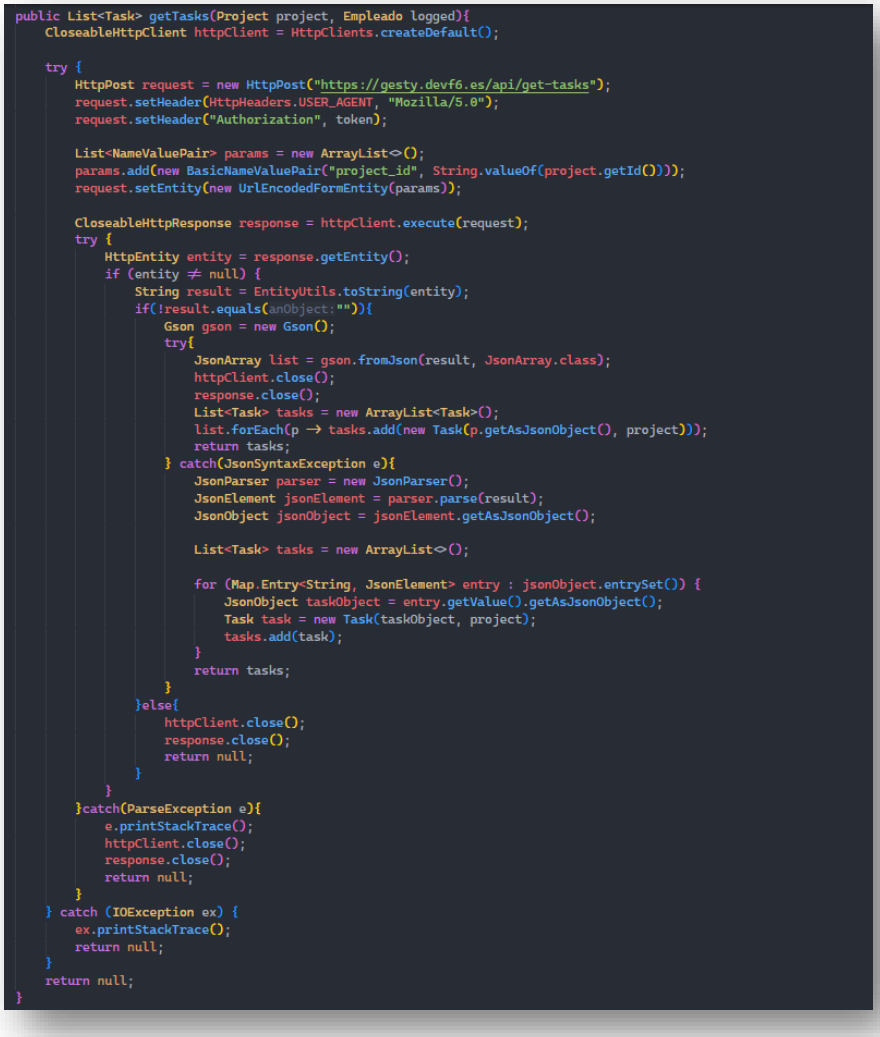


```java
public List<Project> getProjects(Empleado logged){
    CloseableHttpClient httpClient = HttpClients.createDefault();

    try {
        HttpPost request = new HttpPost("https://gesty.devf6.es/api/get-projects");
        request.setHeader(HttpHeaders.USER_AGENT, "Mozilla/5.0");
        request.setHeader("Authorization", token);

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("company_id", logged.getCompanyId()));
        request.setEntity(new UrlEncodedFormEntity(params));

        CloseableHttpResponse response = httpClient.execute(request);
        try {
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                String result = EntityUtils.toString(entity);
                if(!result.equals(anObject:"")){
                    Gson gson = new Gson();
                    JsonArray list = gson.fromJson(result, JsonArray.class);
                    httpClient.close();
                    response.close();
                    List<Project> projects = new ArrayList<Project>();
                    list.forEach(p -> projects.add(new Project(p.getAsJsonObject(), logged)));
                    return projects;
                }else{
                    httpClient.close();
                    response.close();
                    return null;
                }
            }
        }catch(ParseException e){
            e.printStackTrace();
            httpClient.close();
            response.close();
            return null;
        }
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
    return null;
}
```

Esta función nos otorga todos los proyectos de la empresa asignada para un usuario. Para ello, realiza una petición por POST a https://gesty.devf6.es/api/get-projects, adjuntando el token de autentificación en la cabecera de la petición.

En el cuerpo de la petición, se añade el identificador de empresa configurado en el empleado.

En el caso que la petición sea válida, devolverá una lista de Proyectos, en caso contrario, devolverá una lista nula.



```java
public List<Task> getTasks(Project project, Empleado logged){
    CloseableHttpClient httpClient = HttpClients.createDefault();

    try {
        HttpPost request = new HttpPost("https://gesty.devf6.es/api/get-tasks");
        request.setHeader(HttpHeaders.USER_AGENT, "Mozilla/5.0");
        request.setHeader("Authorization", token);

        List<NameValuePair> params = new ArrayList<>();
        params.add(new BasicNameValuePair("project_id", String.valueOf(project.getId())));
        request.setEntity(new UrlEncodedFormEntity(params));

        CloseableHttpResponse response = httpClient.execute(request);
        try {
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                String result = EntityUtils.toString(entity);
                if(!result.equals(anObject:"")){
                    Gson gson = new Gson();
                    try{
                        JsonArray list = gson.fromJson(result, JsonArray.class);
                        httpClient.close();
                        response.close();
                        List<Task> tasks = new ArrayList<Task>();
                        list.forEach(p -> tasks.add(new Task(p.getAsJsonObject(), project)));
                        return tasks;
                    } catch(JsonSyntaxException e){
                        JsonParser parser = new JsonParser();
                        JsonElement jsonElement = parser.parse(result);
                        JsonObject jsonObject = jsonElement.getAsJsonObject();

                        List<Task> tasks = new ArrayList<>();

                        for (Map.Entry<String, JsonElement> entry : jsonObject.entrySet()) {
                            JsonObject taskObject = entry.getValue().getAsJsonObject();
                            Task task = new Task(taskObject, project);
                            tasks.add(task);
                        }
                        return tasks;
                    }
                }else{
                    httpClient.close();
                    response.close();
                    return null;
                }
            }
        }catch(ParseException e){
            e.printStackTrace();
            httpClient.close();
            response.close();
            return null;
        }
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
    return null;
}
```

La función nos proporciona todas las tareas de un proyecto a un empleado en específico. Para ello, realiza una petición por POST a https://gesty.devf6.es/api/get-tasks, adjuntando el token de autentificación en la cabecera de la petición.

En el cuerpo de la petición, se añade el correo del empleado a configurar, asi como la referencia asignada

La aplicación gira en torno a los 5 objetos básicos para el correcto funcionamiento de la misma:

- Company:

```java
public Company(int id, String name, String companyRef){
    gc = new GestyConnector();
    this.id = id;
    this.name = name;
    this.companyRef = companyRef;
    projects = new ArrayList<>();
}
```

```java
public Company(JsonObject json){
    gc = new GestyConnector();
    id = json.get("id").getAsInt();
    name = json.get("company_name").getAsString();
    companyRef = json.get("company_ref").getAsString();
    projects = new ArrayList<>();
}
```

Para la creación de nuevas compañías existe dos posibilidades.

- A través de valores por separado
- A través de un objeto JSON

El más usado en este proyecto es la creación a partir del objeto JSON, ya que, las respuestas del servidor a las peticiones realizadas por la clase *GestyConnector* son en formato JSON.

- Project:

```java
public Project(String name, int id, Company company){
    this.name = name;
    this.id = id;
    this.company = company;
    this.gc = new GestyConnector();
    tasks = new ArrayList<>();
}

public Project(JsonObject json, Empleado logged){
    this.gc = new GestyConnector();
    this.name = json.get("project_name").getAsString();
    this.id = json.get("project_id").getAsInt();
    this.company = logged.getCompany();
    tasks = new ArrayList<>();

}
```

Al igual que con las empresas, a la hora de crear un nuevo proyecto, disponemos de dos opciones: Usando valores por separado ó usando un objeto JSON

```java
/**
 * Get a list of Tasks assigned to this project
 * @param logged logged
 * @return tasks
 */
public List<Task> getTasks(Empleado logged){
    this.tasks = gc.getTasks(this, logged);
    return this.tasks;
}
```

Para la obtención de las tareas de un proyecto, se dispone de una función que, pasando un empleado, ejecuta la función de obtención de tareas de la clase *GestyConnector* y, posteriormente, se devuelve esa lista.

- Task:

```java
public Task(String name, String desc, int id){
    this.name = name;
    this.description = desc;
    this.id = id;
}

public Task(JsonObject json, Project project){
    this.name = json.get("task_name").getAsString();
    this.description = json.get("task_desc").getAsString();
    this.id = json.get("task_id").getAsInt();
    this.project = project;
}
```

Para la creación de una nueva tarea, tenemos 2 formas de crearlas, al igual que las anteriores clases, usando valores por separado o a través de un objeto JSON

- User:

```java
private String email, sessionId;
  You, hace 2 meses • New laravel features and java
public User(String email, String sessionId){
    this.email = email;
    this.sessionId = sessionId;
}

//Getters and Setters
public String getEmail(){
    return this.email;
}
public void setEmail(String email){
    this.email = email;
}
public String getSessionId(){
    return this.sessionId;
}
public void setSessionId(String sessionId){
    this.sessionId = sessionId;
}
```

La clase User va a ser una clase básica, la cual va a ser la clase padre de la siguiente clase.

- Empleado:

  La clase Empleado es la clase principal para los usuarios de la aplicación, desde la cual vamos a poder administrar al usuario.

  En esta clase podremos administrar tanto la referencia de compañía asignada al usuario, como los proyectos y listas de tareas asignadas al mismo.

```java
public void setCompanyRef(String companyId){
    String company = gc.checkRef(companyId, super.getSessionId());
    Gson gson = new Gson();
    JsonObject obj = gson.fromJson(company, JsonObject.class);
    String companyName = obj.get("company_name").getAsString();
}
```

```java
public void setCompany(){
    this.company = gc.getCompany(this);
}

public void setProjects(){
    List<Project> projects = gc.getProjects(this);
    this.company.setProjects(projects);

}
```

En estas funciones, se realizan una serie de peticiones a través de una instancia de la clase *GestyConnector*, a través de las cuales se obtiene los datos necesarios para el correcto funcionamiento del empleado en la aplicación.

La clase Empleado cuenta con dos funciones para obtener, de forma sencilla, tanto proyectos como tareas en específico, las cuales son las siguientes:

```java
public Project getProjectFromName(String param){
    Project[] valid = new Project[1];
    List<Project> projects = company.getProjects();
    projects.forEach(p -> {
        if(p.getName().equals(param)){ valid[0] = p; }
    });
    return valid[0];
}
```

```java
public Task getTaskFromName(String param, String projectName){
    Task task;
    Project p = getProjectFromName(projectName);
    List<Task> tasks = p.getTasks();
    task = tasks.stream().filter(t -> t.getName() == param).findAny().get();
    return task;
}
```

En estas funciones se realiza una obtención de los datos y, posteriormente, se procede a un filtrado a través de una expresión lambda, la cual se ha desarrollado de dos formas diferentes en búsqueda de la mas optima, siendo las detalladas en el código.

Estas clases descritas permiten el funcionamiento correcto de la aplicación, obteniendo y almacenando los datos de forma correcta. Para la parte visual de la aplicación vamos a crear 3 JFrames, los cuales van a formar el inicio de sesión, la parte principal de la aplicación, donde encontraremos los proyectos y las tareas de la aplicación; y un pequeño chat bidireccional, el cual cuenta con un pequeño inicio de sesión junto a él.

- Main:

  Esta es la primera página de la aplicación cliente, en la cual encontramos un inicio de sesión, asi como una redirección a la pagina de registro para nuevos usuarios.


```
try {
    UIManager.setLookAndFeel(new MaterialLookAndFeel(new MaterialLiteTheme()));
} catch (javax.swing.UnsupportedLookAndFeelException ex) {
    java.util.logging.Logger.getLogger(App.class.getName()).log(java.util.logging.Level.SEVERE, msg:null, ex);
}
//</editor-fold>
```

Cargamos el nuevo *Look&Feel* de la aplicación en la función *main* de la clase.

```
private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
    btnLogin.setEnabled(b:false);
    try{
        String email = txtEmail.getText();
        String pass = txtPass.getText();

        this.logged = gc.login(email, pass);
        if(logged != null){
            app = new App(logged, gc);
            this.setVisible(b:false);
            app.setVisible(b:true);
        }else{
            txtError.setText(text:"Las credenciales no son correctas.");
            txtError.setVisible(aFlag:true);
        }
    }catch(Exception e){
        e.printStackTrace();
        txtError.setText(text:"Ha habido un problema con la aplicación.");
    }finally{
        btnLogin.setEnabled(b:true);
    }
}//GEN-LAST:event_btnLoginActionPerformed
```

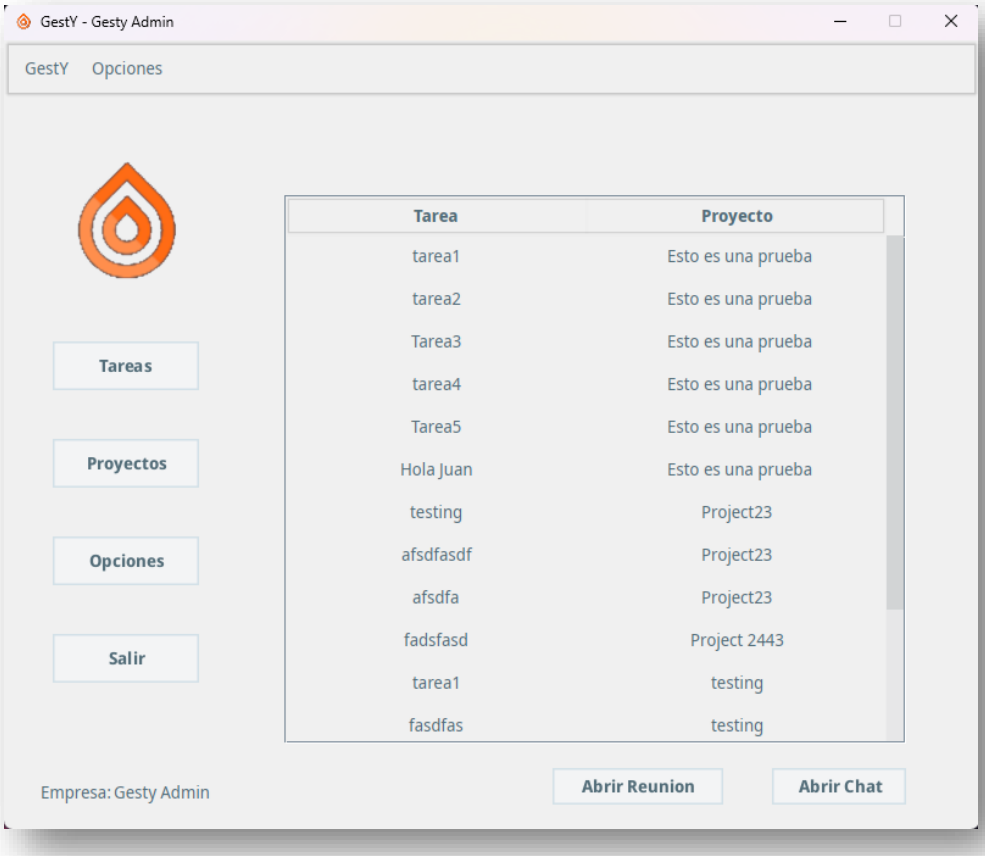
Cuando se hace click en el botón de 'Iniciar sesión' se desactiva el botón para evitar dobles clicks y se verifica si es válido o no.

En caso que sea correcto, lanza la pestaña *App*, en caso contrario, muestra por pantalla que las credenciales no son correctas.

```java
try {
    // Verificar si el Desktop es compatible con la acción de abrir el navegador
    if (Desktop.isDesktopSupported() && Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)) {
        Desktop.getDesktop().browse(new URI(str:"https://gesty.devf6.es/register"));
    } else {
        // Si no es compatible, puedes manejarlo de alguna otra manera (por ejemplo, abrir el enlac
        System.out.println(x:"El navegador web no es compatible en este sistema.");
    }
} catch (Exception e) {
    // Manejo de excepciones si ocurre algún error al abrir el navegador
    e.printStackTrace();
}
```

A la hora de hacer click en el texto de 'Regístrate aqui', abre el navegador por defecto del navegador hacia la página de registro

- App:



La clase *App* es la clase principal de la parte visual de *GestY*, ya que en esta se reúne la visualización de los datos, asi como la posibilidad de abrir un chat de texto como una nueva reunión.

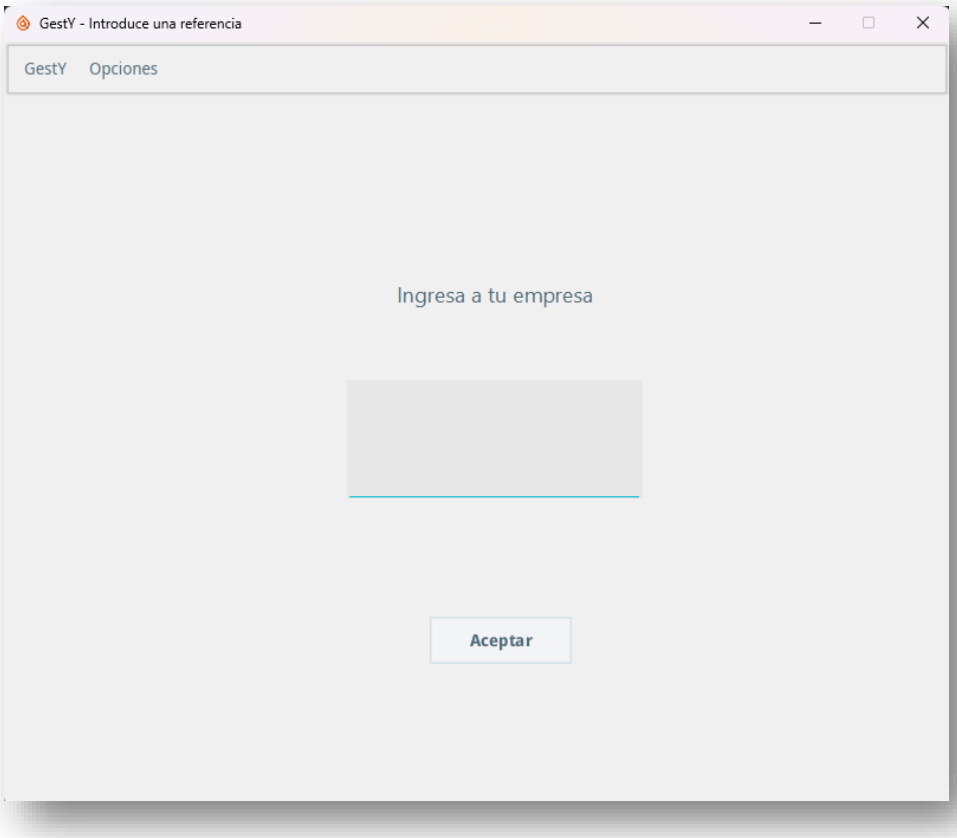
Para ello, se ha desarrollado las siguientes funciones:

```java
public App(Empleado logged, GestyConnector gc) {
    app = this;
    this.logged = logged;
    this.gc = gc;
    initComponents();
    setLocationRelativeTo(c:null);
    company = gc.getCompany(logged);
    app.checkRef();
    getData();
    this.setTitle("GestY - " + logged.getCompany().getName());
    this.setIconImage(new ImageIcon(filename:"logo.png").getImage());
    ico.setIcon(new ImageIcon(filename:"logo.png"));
    lblCompany.setText(logged.getCompany().getName());
    jtProjects.addMouseListener(ml);
    jtTasks.addMouseListener(mlt);
    printTasksTable();
    reloadData();
}
```

El constructor inicializa todo lo esencial de la aplicación, asi como ejecuta las funciones de obtención de datos, pintado de tablas y *Listeners* de las propias tablas. También se configura el título de la aplicación según la empresa a la que pertenezcas.

```java
public void checkRef(){
    if(gc.hasRef(this.logged)){
        app.jpApp.setVisible(aFlag:true);
        app.jpRef.setVisible(aFlag:false);
    }else{
        app.jpApp.setVisible(aFlag:false);
        app.jpRef.setVisible(aFlag:true);
    }
}
```

Al iniciar sesión, se verifica si el usuario introducido tiene configurado una referencia de empresa. En caso que no la tenga, se abrirá una pantalla para la configuración de la referencia.



Cuando el empleado no tiene referencia, llega a esta pestaña, la cual facilita la inserción de la referencia de la empresa.

```
private void btnRefAcceptActionPerformed(java.awt.event.ActionEvent evt) {
    if(gc.setUserRef(logged, ref:txtRefUser.getText())){
        logged.setCompany();
        reloadApp();
        loadApp();
    }
}
```

Cuando se hace click en el botón de 'Aceptar', se hace una petición al servidor para configurar la referencia. Posteriormente se recarga la aplicación con los datos pertinente y se lanza.

```java
public void reloadApp(){
    Thread wait = new Thread( () -> {
        try{
            getData();
            this.setTitle("GestY - " + logged.getCompany().getName());
            this.setIconImage(new ImageIcon(filename:"logo.png").getImage());
            ico.setIcon(new ImageIcon(filename:"logo.png"));
            lblCompany.setText(logged.getCompany().getName());
            jtProjects.addMouseListener(ml);
            jtTasks.addMouseListener(mlt);
            printTasksTable();
            reloadData();
            Thread.sleep(millis:1000);
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    });
    wait.run();
}
```

Esta función realiza una función parecida a la función constructor de la clase, teniendo el usuario con la referencia configurada.

```java
public void loadApp(){

    jpRef.setVisible(aFlag:false);
    jpApp.setVisible(aFlag:true);
}
```

Esta función abre el panel principal de la aplicación y oculta el panel de la referencia.

```java
MouseListener ml = new MouseListener() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 2) {
            int row = jtProjects.getSelectedRow();
            List<Project> projects = logged.getCompany().getProjects();
            JLabel label = (JLabel) jtProjects.getValueAt(row, column:0);
            Project p = logged.getProjectFromName(label.getText());
            if (p != null) {
                jpProjects.setVisible(aFlag:false);
                printTasksTable(p);
                jpTasks.setVisible(aFlag:true);
            }
        }
    }

    @Override
    public void mousePressed(MouseEvent e) { /**/ }

    @Override
    public void mouseReleased(MouseEvent e) { /**/ }

    @Override
    public void mouseEntered(MouseEvent e) { /**/ }

    @Override
    public void mouseExited(MouseEvent e) { /**/ }
};
```

Una de las características de la aplicación es la posibilidad de eventos de ratón en las tablas, las cuales son dinámicas gracias a la obtención e inserción de datos en las demás funciones de la aplicación.

```
MouseListener mlt = new MouseListener() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 2) {
            int row = jtTasks.getSelectedRow();

            String taskName = String.valueOf(jtTasks.getValueAt(row, column:0));
            String projectName = String.valueOf(jtTasks.getValueAt(row, column:1));
            Task t = logged.getTaskFromName(taskName, projectName);

            jdTask.setTitle(t.getName());
            jdTask.setIconImage(new ImageIcon(filename:"logo.png").getImage());

            txtTaskName.setText(t.getName());
            txtTaskDesc.setText(t.getDesc());

            jdTask.setLocationRelativeTo(c:null);
            jdTask.setVisible(b:true);
        }
    }

    @Override
    public void mousePressed(MouseEvent e) { /**/ }

    @Override
    public void mouseReleased(MouseEvent e) { /**/ }

    @Override
    public void mouseEntered(MouseEvent e) { /**/ }

    @Override
    public void mouseExited(MouseEvent e) { /**/ }
};
```

Al tener dos tablas con datos dinámicos, requiere de otra función *Listener*, ya que no es válida para las dos.

```java
public void getData(){
    boolean loading = false;
    Empleado emp = this.logged;
    if(emp.getCompany() == null){
        emp.setCompany();
    }
    if(emp.getCompany().getProjects().isEmpty() && !loading){
        loading = true;
        emp.setProjects();
        emp.getProjects().forEach(p -> {
            p.setTasks(p.getTasks(logged));
        });
        loading = false;
    }

    if(!emp.getCompany().getProjects().isEmpty() && !loading){
        if(!emp.getCompany().getProjects().isEmpty()){
            loading = true;
            emp.getCompany().getProjects().forEach(p -> p.getTasks(logged));
            loading = false;
        }
    }
}
```

Esta función es la encargada de cargar todos los datos desde el servidor y almacenarlos para un posterior uso en la aplicación.

```java
public void printTasksTable(){
    List<Project> projects = logged.getProjects();
    DefaultTableModel model = (DefaultTableModel) jtTasks.getModel();
    for( int i = model.getRowCount() - 1; i ≥ 0; i-- ) {
        model.removeRow(i);
    }

    if(projects ≠ null){
        if(!projects.isEmpty()){
            projects.forEach(p -> {
                System.out.println(p.getName());
                if(!p.getTasks().isEmpty()){
                    List<Task> tasks = p.getTasks();
                    tasks.forEach(t -> {
                        model.addRow(new Object[]{t.getName(), p.getName()});
                    });
                }
            });
        }
    }
    jpProjects.setVisible(aFlag:false);
    jpTasks.setVisible(aFlag:true);
}
```

Para el pintado de todas tareas en sus tablas, se dispone de esta función, la cual filtra las tareas disponibles en todos los proyectos del usuario y las incorpora a la tabla. Posteriormente, oculta la tabla de proyectos, en el caso que estuviera abierta, y carga la tabla de tareas.

```java
public void printTasksTable(Project project){
    DefaultTableModel model = (DefaultTableModel) jtTasks.getModel();
    for( int i = model.getRowCount() - 1; i ≥ 0; i-- ) {
        model.removeRow(i);
    }

    List<Task> tasks = project.getTasks();
    if(tasks ≠ null){
        tasks.forEach(t → {
            model.addRow(new Object[]{t.getName(), project.getName()});
        });
    }

}
```

Si se busca el pintado de las tareas de un proyecto en específico, se dispone de esta función, la cual, mediante un proyecto pasado como parámetro, obtiene todas las tareas del mismo.

```java
public void updateProjectsTable(){
    List<Project> projects = gc.getProjects(logged);
    DefaultTableModel model = (DefaultTableModel) jtProjects.getModel();
    for( int i = model.getRowCount() - 1; i ≥ 0; i-- ) {
        model.removeRow(i);
    }


    for (Project p : projects) {
        JLabel label = new JLabel(p.getName());
        TableCellRenderer renderer = new LabelTableCellRenderer();
        jtProjects.getColumnModel().getColumn(columnIndex:0).setCellRenderer(renderer);
        model.addRow(new JLabel[] { label });
    }
}
```

Para actualizar y pintar la tabla de proyecto, se dispone de esta función, la cual itera por los proyectos disponibles para el usuario

```java
private void btnMeetActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_btnMeetActionPerformed
    try {
        // Verificar si el Desktop es compatible con la acción de abrir el navegador
        if (Desktop.isDesktopSupported() && Desktop.getDesktop().isSupported(Desktop.Action.BROWSE)) {

            Desktop.getDesktop().browse(new URI("https://meet.jit.si/" + "gesty-" + logged.getCompany().getName().toLowerCase().replace(oldChar:' ', newChar:'-')));
    } else {
        // Si no es compatible, puedes manejarlo de alguna otra manera (por ejemplo, abrir el enlace en un enlace etiquetado en tu aplicación)
        System.out.println(x:"El navegador web no es compatible en este sistema.");
        }
    } catch (Exception e) {
        // Manejo de excepciones si ocurre algún error al abrir el navegador
        e.printStackTrace();
    }
}//GEN-LAST:event_btnMeetActionPerformed
```

Para abrir la reunión, al hacer click en el botón ‘Abrir Reunión’, abre el navegador por defecto del sistema, dirigiendo a una página de https://meet.jit.si/ la cual será exclusiva para cada empresa.

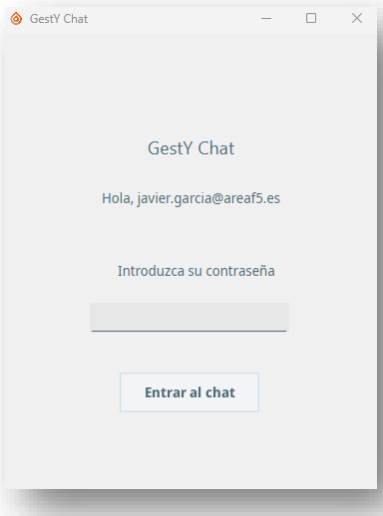
```
private void btnChatActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:eve
    String empresa = logged.getCompany().getName();
    topic = "gesty/" + empresa.toLowerCase().replace(oldChar:' ', newChar:'-');
    openChat(topic);
}//GEN-LAST:event_btnChatActionPerformed
```

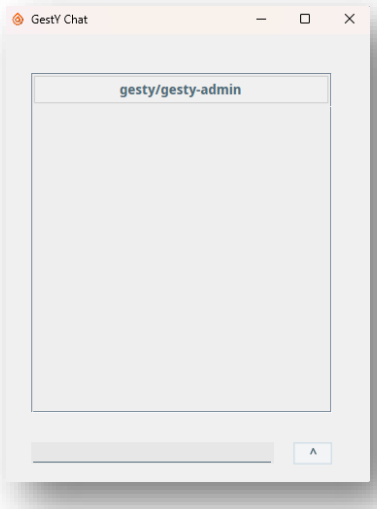
Para abrir el chat, se utilizan dos funciones. La primera consta de la acción de pulsado del botón, la cual obtiene el nombre de la compañía para crear el *topic* necesario para el chat, lanzando la siguiente función:

```
public void openChat(String topic){
    Chat chat = new Chat(logged, topic);
    chat.setLocationRelativeTo(c:null);
    chat.setVisible(b:true);
}
```

Esta función crea una nueva instancia del chat, pasándole como parámetro el *topic* y el empleado que ha iniciado sesión.

- Chat:





Para el inicio del chat, la clase carga todos lo necesario en el constructor:

```java
public Chat(Empleado logged, String topic) {
    this.logged = logged;
    this.topic = topic;
    initComponents();
    txtEmail.setText("Hola, " + logged.getEmail());
    this.setTitle(title:"GestY Chat");
    this.setIconImage(new ImageIcon(filename:"logo.png").getImage());
    jpLogin.setVisible(aFlag:true);
    jpChat.setVisible(aFlag:false);
}
```

En este, a parte de la inicialización de todo lo requerido por la clase, se configura tanto el titulo como el icono.

Cuando iniciemos sesión, se comprobará si las credenciales son válidas gracias a la siguiente función:

```java
private void btnSubmitActionPerformed(java.awt.event.ActionEvent evt) {
    String pass = txtPass.getText();
    ChatClient cc = openChat(logged, pass);
    cc.receiveMessages();
    if(cc.connected()){
        jpLogin.setVisible(aFlag:false);
        jpChat.setVisible(aFlag:true);
        printChat();
    }
}//GEN-LAST:event_btnSubmitActionPerformed
```

Se crea un nuevo objeto *ChatClient*, el cual se encargará de manejar el cliente de *MQTT*, usado para la conexión con el servicio. También, gracias a la función *receiveMessages*, quedará a la espera de nuevos mensajes entrantes.

```java
public ChatClient openChat(Empleado logged, String password){
    ChatClient cc = new ChatClient(logged, password,this.topic, this);
    this.cc = cc;
    mqttClient = cc.getMqttConnector();
    mqttClient.setChatPanel(this);
    return cc;
}
```

Esta función devuelve el cliente de chat completamente configurado, el cual es creado pasando como parámetro el usuario, contraseña, *topic* y el propio objeto Chat, ya que, el mismo cliente, va a ocuparse de varios factores de pintado. Tambien configura el Chat al cliente MQTT para posibles necesidades del mismo.

```
public void printOwnMessageOnChat(String message){
    mqttClient.publishMessage(topic, message);
    DefaultTableModel model = (DefaultTableModel) jtChat.getModel();
    model.addRow(new Object[]{"Tú → " + message});
}
```

```
public static void printRemoteMessageOnChat(MqttMessage message, Chat chat){
    List<UserProperty> props = message.getProperties().getUserProperties();

    DefaultTableModel model = (DefaultTableModel) chat.jtChat.getModel();
    model.addRow(new Object[]{props.get(0).getValue().substring(0, props.get(0).getValue().indexOf("@")) + " → " + message});
}
```

Para el pintado de los mensajes usaremos estas funciones, las cuales nos permitirán añadir al chat los mensajes, identificando si el mensaje es propio o es de una persona externa, mostrando el nombre de la persona en el chat. Esta última función es estática ya que podrá ser llamada por otra clase sin necesidad de una instancia de Chat.

Para el funcionamiento del chat, necesitamos administrar tanto el envío como la recepción de nuevos mensajes. Para ello, disponemos de dos clases para esta finalidad:

- MQTTConnector:

  El conector a MQTT es la parte fundamental del chat, ya que es el encargado de la gestión del mismo. Para ello, dispone de un complejo constructor con mucha configuración:

```
public MQTTConnector(Empleado logged, String password) {
        this.logged = logged;
        this.password = password;
        try {
            mqttClient = new MqttClient("ssl://e7fa393ea4af4647a2482dffccd1d654.s2.eu.hivemq.cloud:8883",
logged.getSessionId(), mp);
            MqttConnectionOptions options = new MqttConnectionOptions();
            options.setCleanStart(true);
            options.setUserName(logged.getEmail());
            options.setPassword(password.getBytes());

            mqttClient.setCallback(new MqttCallback() {
```

```
        public void connectionLost(Throwable cause) {
            System.out.println("Lost connection");
            cause.printStackTrace();
        }

        @Override
        public void messageArrived(String topic, MqttMessage message) throws Exception {
            System.out.println("Ha llegado un mensaje: " + message.toString());
            List<UserProperty> userProps = message.getProperties().getUserProperties();
            UserProperty up = userProps.get(0);
            if(!up.getValue().equals(logged.getEmail())){
                Chat.printRemoteMessageOnChat(message, chat);
            }
        }

        public void deliveryComplete(IMqttDeliveryToken token) {
            System.out.println("Delivery Done!  ");
        }

        @Override
        public void disconnected(MqttDisconnectResponse mdr) {
            System.out.println(mdr.getReasonString());
            disconnect();
        }

        @Override
        public void mqttErrorOccurred(MqttException me) {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public void deliveryComplete(IMqttToken imt) {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public void connectComplete(boolean bln, String string) {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public void authPacketArrived(int i, MqttProperties mp) {
            throw new UnsupportedOperationException("Not supported yet.");
        }
    });
    IMqttToken token = mqttClient.connectWithResult(options);

    token.waitForCompletion();
```

```
            this.connected = mqttClient.isConnected();



        } catch (MqttException e) {
            System.out.println("error");
            e.printStackTrace();
        }
    }
```

En el se configura la dirección del servicio, el usuario y contraseña empleado para el inicio de sesión, asi como las funciones encargadas de envió y recibo de mensajes

```
public boolean subscribeToTopic(String topic, IMqttMessageListener listener) {
    try {
        mqttClient.subscribe(topic, 1);
        return true;
    } catch (MqttException e) {
        System.err.println(x:"No posible");
        return false;
    }
}
```

Para la suscripción a un *topic*, se dispone de una función la cual nos permite hacerlo de forma sencilla, pasando como parámetro el *topic*, junto a un *Listener* para la entrada de nuevos mensajes.

```
public void publishMessage(String topic, String message) {
    try {
        MqttMessage mqttMessage = new MqttMessage(message.getBytes());
        MqttProperties props = new MqttProperties();
        List<UserProperty> userprop = new ArrayList<>();
        userprop.add(new UserProperty("user", logged.getEmail()));
        props.setUserProperties(userprop);
        mqttMessage.setProperties(props);
        System.out.println(mqttMessage.getProperties().toString());
        mqttClient.publish(topic, mqttMessage);
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Esta función nos permite enviar mensajes al *topic* suscrito, la cual recibe por parámetro el propio *topic* junto al mensaje a enviar. Para el correcto funcionamiento del chat, al mensaje se le adjunta una propiedad de usuario, la cual incorpora el email del propio usuario.

```
public void disconnect() {
    try {
        mqttClient.disconnect();
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Esta función es necesaria para el funcionamiento de chat, ya que es necesario desconectar del chat antes de cerrarlo, para evitar posibles problemas.

- ChatClient:

  Esta clase es la encargada de hacer de intermediaria entre Chat y MQTTConnector.

```
public ChatClient(Empleado logged, String password, String topic, Chat chat)
    this.topic = topic;
    this.chat = chat;
    mqttClient = new MQTTConnector(logged, password);
    mqttClient.subscribeToTopic(topic, (s, mqttMessage) -> {
        String message = new String(mqttMessage.getPayload());
        System.out.println("Received message: " + message);
    });

    if(mqttClient.connected()){
        this.connected = true;
    }
}
```

Para ello, requiere de un constructor completo, el cual necesita el empleado, su contraseña, el *topic* a suscribir y una instancia de la clase Chat.

```java
public void receiveMessages() {
    Thread t = new Thread( () -> {
        boolean connected = true;
        while(connected){
            try{
                connected = mqttClient.subscribeToTopic(this.topic, (s, mqttMessage) -> {
                    String message = new String(mqttMessage.getPayload());
                    String props = new String(mqttMessage.getProperties().getAuthenticationData().toString());
                });
            }catch(Exception e){
                System.err.println(x:"Not authorized");
                break;
            }

            if(!connected){
                chat.txtFailed.setText(text:"Las credenciales no son válidas");
            }

            try {
                Thread.sleep(millis:1000); // Espera de 1 segundo antes de volver a verificar nuevos mensajes.
            } catch (InterruptedException e) {
                System.err.println(e.getMessage());
            }
        }
    });
    t.start();
}
```

Esta es la función encargada de escuchar si llegan nuevos mensajes.

Por último, vamos a crear un pequeño renderizador que nos permita pintar, dentro de una tabla, un componente JLabel de forma que sea visible correctamente.

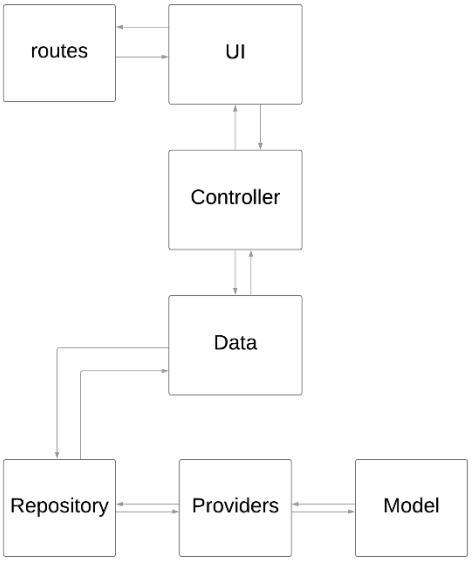
```java
public class LabelTableCellRenderer extends DefaultTableCellRenderer {
    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
        if (value instanceof JLabel) {
            JLabel label = (JLabel) value;
            setText(label.getText()); // Mostrar solo el texto del JLabel
        } else {
            setText(String.valueOf(value)); // Mostrar cualquier otro valor como texto
        }

        this.setHorizontalAlignment(SwingConstants.CENTER);
        return this;
    }
}
```

- 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*** a través del cual obtendremos un listado de

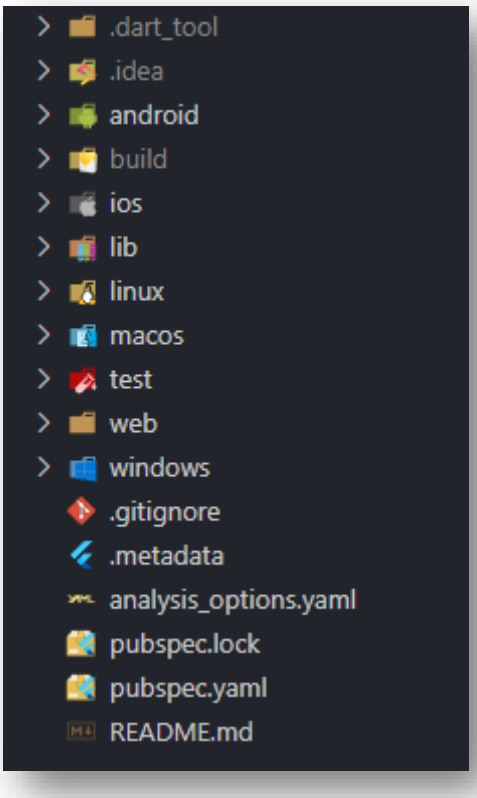
información en formato JSON, el cual formatearemos dentro de la aplicación usando el flujo de trabajo de Flutter.



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

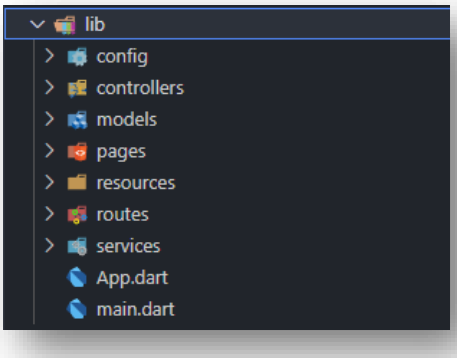
Flutter autogenera una serie de carpetas necesarias para el correcto funcionamiento del propio framework:



Dentro de todas estas carpetas, vamos a centrarnos en la carpeta *lib*, la cual va a contener los ficheros de nuestra aplicación.

En Flutter no es necesario separar las clases, ya que permite juntar todo el codigo en un mismo fichero y asi depender de menos ficheros. Aun sabiendo esto, se ha decidido separar en ficheros y carpetas diferentes para una mejor organización

Este es el árbol de archivos:



La aplicación en Flutter consiste en tres pantallas con la misma funcionalidad, cual carga un listado con la información, ya sea de compañías, proyectos o tareas.

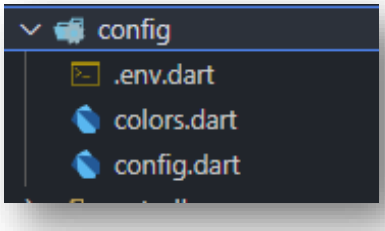
Para evitar caer en la redundancia, se explicará el funcionamiento general de la aplicación, junto al funcionamiento de las empresas, ya que, como comentaba, el funcionamiento es idéntico.

La aplicación está creada a partir del plugin *GetX*, es un potente y versátil plugin para el desarrollo de aplicaciones Flutter. Proporciona una arquitectura sólida y una amplia gama de funcionalidades para simplificar el desarrollo y mejorar la productividad de los desarrolladores.

En términos técnicos, *GetX* se compone de varios módulos que trabajan en conjunto para ofrecer una experiencia de desarrollo completa.

El proyecto de compone de las siguientes carpetas:

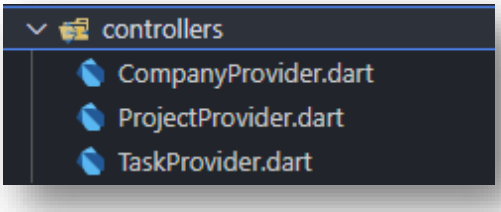
- Config:



Esta carpeta contiene los ficheros de configuración del proyecto, incluyendo en ellos las claves de conexión, colores y las cabeceras de las peticiones realizadas.

- Controllers:

Los controladores o *Providers* son aquellos encargados de hacer de intermediarios entre los consumidores y los servicios, que son los que realizan las peticiones.



El controlador de empresas contiene el numero de columnas en las que se van a pintar en pantalla, asi como una serie de funciones necesarias para el funcionamiento del mismo:

```dart
class CompanyProvider extends GetxController {
  //Es el punto medio entre el consumer y el servicio, el cual re

  bool _loading = false;
  static int columns = 2;
  bool get loading => _loading;
  final RxList<Company> _companies = <Company>[].obs;
  List<Company> get companies => _companies;

  @override
  void onInit() {
    getCompanies();
    super.onInit();
  }

  void changeLoading(bool newState){
    _loading = newState;
    update();
  }

  void setCompanies(List<Company> newList){
    _companies.clear();
    _companies.addAll(newList);
    update();
  }

  Future<void> getCompanies() async {
    changeLoading(true);
    List<Company> aux = await CompanyApiService.getCompanies();
    setCompanies(aux);
    changeLoading(false);
    update();
  }
}
```

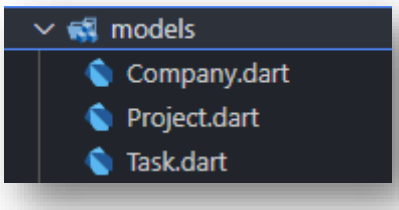
1. onInit: es la función que se ejecuta cuando se inicia el controlador. Ejecuta la función de obtención de empresas y, a su vez, la función constructora de la clase padre.

2. changeLoading: Altera la propiedad *loading* por el valor pasado como parámetro.

3. setCompanies: Limpia las compañías y añade una nueva lista pasada como parametro.

4. getCompanies: mientras esta función está en desarrollo, la propiedad *loading* pasa a ser verdadera. La función hace uso del servicio, el cual realiza una petición al servidor, obteniendo el listado de empresas.

- Models:



Los modelos son los objetos básicos que vamos a crear gracias a las peticiones realizadas por los servicios.
Contienen en si las propiedades y funciones necesarias para cada uno de ellos.

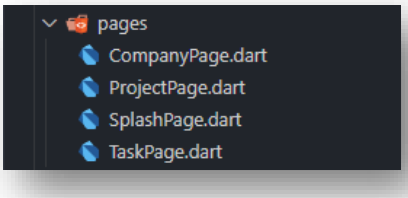
Estos modelos son creados a partir de la respuesta en formato JSON de la petición al servidor.

```dart
class Company {
  int? _id;
  String? _companyId;
  String? _companyRef;
  String? _companyName;
  String? _createdAt;
  String? _updatedAt;

  Company(
      {int? id,
        String? companyId,
        String? companyRef,
        String? companyName,
        String? createdAt,
        String? updatedAt}) {
    if (id != null) {
      this._id = id;
    }
    if (companyId != null) {
      this._companyId = companyId;
```

```
    }
    if (companyRef != null) {
      this._companyRef = companyRef;
    }
    if (companyName != null) {
      this._companyName = companyName;
    }
    if (createdAt != null) {
      this._createdAt = createdAt;
    }
    if (updatedAt != null) {
      this._updatedAt = updatedAt;
    }
  }

  int? get id => _id;
  set id(int? id) => _id = id;
  String? get companyId => _companyId;
  set companyId(String? companyId) => _companyId = companyId;
  String? get companyRef => _companyRef;
  set companyRef(String? companyRef) => _companyRef = companyRef;
  String? get companyName => _companyName;
  set companyName(String? companyName) => _companyName = companyName;
  String? get createdAt => _createdAt;
  set createdAt(String? createdAt) => _createdAt = createdAt;
  String? get updatedAt => _updatedAt;
  set updatedAt(String? updatedAt) => _updatedAt = updatedAt;

  Company.fromJson(Map<String, dynamic> json) {
    _id = json['id'];
    _companyId = json['company_id'];
    _companyRef = json['company_ref'];
    _companyName = json['company_name'];
    _createdAt = json['created_at'];
    _updatedAt = json['updated_at'];
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = new Map<String, dynamic>();
    data['id'] = this._id;
    data['company_id'] = this._companyId;
    data['company_ref'] = this._companyRef;
    data['company_name'] = this._companyName;
    data['created_at'] = this._createdAt;
    data['updated_at'] = this._updatedAt;
    return data;
  }
}
```

Esto es un modelo completo en Flutter.

- Pages:



Las páginas son aquellas clases que van a ser mostradas a la hora de cargar una ruta. También contiene los denominados *Consumers*, encargados del uso de los datos para pintado en pantalla.

Todas las páginas en flutter cuentan con 2 clases en su estructura, la clase principal, la cual usa el mismo nombre del fichero; y una clase con el mismo nombre, incorporando en el nombre el sufijo *State*.

Estas clases *state* son las encargadas de la declaración y manejo del estado mutable de la página, siendo en esta clase donde incorporemos el núcleo de las páginas.

Ya que estamos usando el plugin *GetX*, vamos a requerir de una clase externa, la cual vamos a llamar *SplashPage*, que será la encargada de la carga del controlador principal y del enrutamiento básico.

Esta pagina contará con lo siguiente:

```
void init() {
  Get.put(CompanyProvider()); // ⟹ Inicializa el
  goToHome();
}

/// Espera 2 segundos y redirige a la pantalla prin
void goToHome() {
  Future.delayed(const Duration(seconds: 2))
      .then((value) ⇒ Get.toNamed('/home'));
}

@override
void initState() {
  init();
  super.initState();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(child: customLoadingPage()),
  ); // Scaffold
}
```

- Init: Esta función inicializa y pone al alcance de la aplicación el controlador de compañías y, posteriormente lanza la función *goToHome*

- goToHome: Realiza un cambio de ruta durante una duración de 2 segundos hacia la ruta

- initState: Es la función que se ejecuta al iniciar una instancia de esta clase

- build: Devuelve el objeto base *Scaffold*, el cual contiene una instancia hija centrada en pantalla de la clase *CustomLoadingPage*

Para las empresas, se ha desarrollado la siguiente página:

```dart
class CompanyPage extends StatefulWidget {
  const CompanyPage({Key? key}) : super(key: key);
  @override
  State<CompanyPage> createState() => _CompanyPageState();
}
class _CompanyPageState extends State<CompanyPage> {
  late Size screenSize;
  CompanyProvider controller = Get.find<CompanyProvider>();

  @override
  Widget build(BuildContext context) {
    screenSize = MediaQuery.of(context).size;

    return Scaffold(
      appBar: AppBar(title: const Text('Empresas')),
      body: buildCompanyPage(),
    );
  }

  Widget buildCompanyPage() {
    return GetBuilder<CompanyProvider>(
      init: CompanyProvider(),
      builder: (controller) {
        if (controller.loading) {
          return SizedBox(
            height: screenSize.height,
            width: screenSize.width,
            child: Center(
              child: customLoadingPage(),
            ),
          );
        } else {
          return buildCompanyList();
        }
      },
    );
  }

  Widget buildCompanyList() {
    return SizedBox(
      height: screenSize.height,
      width: screenSize.width,
      child: GridView.builder(
        scrollDirection: Axis.vertical,
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: CompanyProvider
              .columns, //<- esto es para cambiar el numero de columnas, pero si no le pones 2 y ya
```

```
          mainAxisExtent: 310,
          crossAxisSpacing: 10,
        ),
        itemCount: Get.find<CompanyProvider>().companies.length,
        itemBuilder: (context, index) => buildCompanyCard(
            clickable: true,
            company: Get.find<CompanyProvider>().companies[index]),
      ),
    );
  }
  Widget buildCompanyCard({bool? clickable, required Company company}) {
    return GestureDetector(
      child: Text(company.companyName!),
      onTap: () {
        if (clickable ?? false) {
          Get.toNamed('/projects', arguments: {'company': company});
        }
      },
    );
  }
}
```

Esta pagina cuenta con las siguientes funciones:

- build:
  Obtiene la resolución de la pantalla y la almacena en la variable *screenSize*, la cual va a ser usada mas adelante para el desarrollo de las cartas.
  Ésta devuelve un *Scaffold* el cual incluye un widget *AppBar*, siendo este la barra superior de la aplicación que contiene un título; asi como una instancia hija de la siguiente función.

- buildCompanyPage:
  Es la función principal de la página, la cual se encarga de inicializar el controlador, mientras este controlador tenga la variable interna *loading* a verdadero, se mostrará por pantalla un recurso llamado *customLoadingPage*. Cuando termine de cargar, devolverá la siguiente función.

- buildCompanyList:
  Esta función devuelve una lista de las empresas que hay en la aplicación. Para ello, se utiliza un objeto *SizedBox*, requerido al usar un ancho y un alto para la definición de los bloques/*cards*. Este *SizedBox*, tendrá una instancia hija de la clase *GridView*, la cual es la encargada de crear una lista de columnas. Cada objeto a pintar, su diseño será definido por la siguiente función.

- buildCompanyCard:

Esta función crea unos bloques clickables, los cuales cuentan con el nombre del proyecto. Al hacer click sobre ellos, estos redirigirán a la ruta '*/projects*', incorporando como parametro la empresa seleccionada.

- Resources:

  En esta carpeta podremos almacenar recursos que vayamos a necesitar en más de una ocasión a lo largo de la aplicación. En este caso, se almacenará una página de carga.

```dart
Widget customLoadingPage({double? size, Color? color}){
  //Se encajona dentro de un SizedBox para poder darle u
  return SizedBox(
    height: size ?? 20,
    width: size ?? 20,
    child: CircularProgressIndicator(
      color: color?? CustomColors.main,
    ), // CircularProgressIndicator
  ); // SizedBox
}
```

  Cargará una pantalla en blanco con un indicador de carga circular

- Routes:
  En esta carpeta encontraremos lo necesario para el sistema de enrutamiento de la aplicación.

  Encontraremos dos archivos:
  - Routes:

```dart
appRoutes() => [
    //-------------------Home-------------------
    GetPage(
      name: '/',
      page: () => const SplashPage(),
      transition: Transition.leftToRightWithFade,
      transitionDuration: const Duration(milliseconds: 500),
    ), // GetPage
    //-------------------Home-------------------
    GetPage(
      name: '/home',
      page: () => const CompanyPage(),
      transition: Transition.leftToRightWithFade,
      transitionDuration: const Duration(milliseconds: 500),
    ), // GetPage
    //-------------------projects-------------------
    GetPage(
      name: '/projects',
      page: () => const ProjectPage(),
      transition: Transition.leftToRightWithFade,
      transitionDuration: const Duration(milliseconds: 500),
    ), // GetPage
    //-------------------Tasks-------------------
    GetPage(
      name: '/tasks',
      page: () => const TaskPage(),
      transition: Transition.leftToRightWithFade,
      transitionDuration: const Duration(milliseconds: 500),
    ), // GetPage
  ];
```

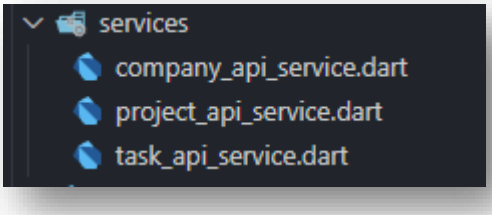
Aquí se crean las diferentes paginas que va a tener la aplicación, asi como qué archivo han de cargar o la transición de carga que tendrá.

- Routes_export:

```
export 'package:gesty/pages/SplashPage.dart';
export 'package:gesty/pages/CompanyPage.dart';
export 'package:gesty/pages/ProjectPage.dart';
export 'package:gesty/pages/TaskPage.dart';
```

En este fichero únicamente vamos a exportar las paginas para que sean accesibles desde toda la aplicación.

- Services:



Los servicios son los encargados de realizar las peticiones HTTP al servidor, pasando como parámetros todo lo necesario para el correcto funcionamiento de la aplicación.

Para la obtención de datos acerca de las compañías, se ha desarrollado el siguiente servicio:

```
class CompanyApiService {
  static Future<List<Company>> getCompanies() async {
    final response =
        await http.post(Uri.parse('$base_url/companies'), headers: headers);
    if (response.statusCode == 200) {
      List parsedList = json.decode(response.body);
      return parsedList.map((e) => Company.fromJson(e)).toList();
    } else {
      throw Exception(response.body);
    }
  }
}
```

Se realiza una petición POST a la URL base, almacenada como variable de entorno dentro del fichero de configuración del proyecto, a la cual se le adjunta las cabeceras necesarias, las cuales incluyen el token de autentificación requerido para las peticiones.
Devuelve un lista de empresas en el caso que la petición este correcta, en caso contrario, no devuelve nada.

3. FASE DE PRUEBAS

Acción	***Desencadenante***
Se lanza la aplicación Java.	Se abre el formulario de inicio de sesión.
Se hace click en el texto 'Regístrese aqui'.	Se abre el navegador por defecto en dirección a https://gesty.devf6.es/register.
Se crea una cuenta de usuario con correo *user@user.com* y contraseña *Usuario1.*	Se registra el nuevo usuario y se redirige a la página https://gesty.devf6.es/dashboard.
Inicia sesión en la aplicación Java con user@user.com y contraseña usuario1. (Contraseña no correcta).	Aparece un mensaje con texto 'Las credenciales introducidas no son válidas'.
Inicia sesión en la aplicación Java con user@user.com y contraseña Usuario1.	Inicia la aplicación, llegando a la pantalla de configuración de referencia de empresa.
Introduce la referencia 'cvs34s', siendo esta la referencia de la empresa 'Gesty Admin'.	Se cierra la pantalla de configuración de referencia, abriendose la pantalla principal con el listado de tareas en pantalla.
Hace click en el botón 'Proyectos'.	Se oculta la tabla de tareas y aparece la tabla con todos los proyectos disponibles para ese usuario.
Hace doble click en un proyecto.	Se oculta la tabla de proyectos y aparece la tabla con todas las tareas disponibles de esa empresa.
Hace doble click en una tarea.	Se abre una ventana con el nombre y la descripción de la tarea.
Cierra la ventana de la tarea.	Se cierra la ventana.
Hace click en el botón 'Opciones'.	Se abre la ventana de opciones vacía.
Hace click en el botón de 'Abrir Reunion'	Se abre el navegador por defecto en dirección a la página https://meet.jit.si/gesty-gesty-admin.
Hace click en el botón de 'Abrir Chat'	Se abre la ventana de Chat, en la cual se lee 'Hola, user@user.com'
Introduce la contraseña mal	Aparece un mensaje con texto 'Las credenciales introducidas no son válidas'.
Introduce la contraseña bien	Se cierra la ventana de inicio de sesión y se abre la ventana con el chat. En este se lee *gesty/* seguido del nombre de la empresa a la que pertenece.
Envía un mensaje con texto 'Buenos dias'	Se pinta en la tabla el mensaje, acompañado de un sufijo 'Tu -> '
Recibe un mensaje proveniente de su compañero javier.garcia con el texto 'Buenos dias, user'	Se pinta en pantalla el mensaje, acompañado de un sufijo 'javier.garcia -> '
Hace click al botón 'Salir'	Se cierra la aplicación sin dejar ningún proceso abierto.

4. DOCUMENTACION DE LA APLICACIÓN

4.1 INTRODUCCIÓN A LA APLICACIÓN

GestY es una infraestructura creada para la correcta gestión de proyectos y tareas

4.2 MANUAL DE INSTALACIÓN

Para la instalación de la aplicación, únicamente ha de ejecutar el fichero ***gesty-0.1-windows-installer.exe*** e instalar como una aplicación cualquiera.

Para la instalación de la aplicación en Android, solo tendrá que descargar el fichero ***gesty.apk*** e instalar como cualquier aplicación.

5. CONCLUSIONES FINALES

El desarrollo de la aplicación ha sido muy laborioso, ya que se ha querido integrar muchas tecnologías diferentes dentro de la misma aplicación, asi como nuevas técnicas y buenas prácticas código que ralentizan el desarrollo del proyecto.

El proyecto, por desgracia, no queda lo completo que me gustaria. Se echa en falta una parte visual mucho mas lograda en el cliente para movil, asi como una depuración de las aplicaciones mas a fondo, en búsqueda de todos los errores posibles.

Teniendo en cuenta esto, estoy muy orgulloso de las aplicaciones desarrolladas, asi como de la infraestructura creada para el buen desarrollo del proyecto.

Gracias a este proyecto, he podido aprender lenguajes nuevos, asi como técnicas y nuevos intereses como es el desarrollo de aplicaciones móviles usando Flutter, o el desarrollo de herramientas web usando Laravel.

6. 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/