

“CODIGO CONTRACT”

Preparado para:
NOMBRE CONTRACT

Manual Técnico de Plataforma Interactiva CODIGO

Centro Avanzado de Ingeniería Eléctrica y Electrónica
AC3E

Universidad Técnica Federico Santa María

Abril 2022

Rev. 1.0



Control de Cambios

1.0	XX-04-2022	Emitido a Cliente.

Índice

1. Contenido

1	Introducción	5
2	Descripción general del sistema	6
2.1	Front end	7
2.2	Back end.....	7
2.2.1	Manejo en base de datos.....	7
2.2.2	Servicio RESTful y autenticación.....	7
2.3	Back office.....	7
2.3.1	Manejo en base de datos.....	8
2.3.2	Servicio RESTful y autenticación.....	8
2.3.3	Comunicación con Back end	8
2.4	Add-in	8
3	Requisitos del sistema	9
3.1	Front end	9
3.2	Back end.....	9
3.3	Back office.....	9
4	Compilación del código fuente.....	10
4.1	Front end	10
4.1.1	Configuración de URLs e IDs de Facebook y Google	10
4.1.2	Compilación de código.....	11
4.2	Back end.....	11
4.3	Back office.....	11
5	Instalación del sistema.....	12
5.1	Back end.....	12
5.2	Back office.....	14
5.2.1	Configuración de Docker	14
5.2.1.1	Servicio PHP.....	14
5.2.1.2	Base de datos.....	15
5.2.1.3	Servidor web NGINX	15
5.2.2	Configuración de parámetros PHP	16
5.2.3	Construcción de contenedores docker	16
5.2.4	Iniciación de aplicación de Laravel	16
5.2.5	Seeding de la base de datos	17

5.3	Front end	17
6	Módulos del backend y bases de datos	18

1 Introducción

En el presente documento se entrega información técnica del sistema, explicando cómo está conformado el sistema en general y cada uno de sus subsistemas. Se ejemplifica también la arquitectura de la Base de Datos mostrando y detallando cada una de sus tablas.

Se explica también cómo instalar el sistema desde cero detallando cada uno de sus módulos.

2 Descripción general del sistema

El sistema WYS se compone de tres grandes módulos: Front end, Back end y Back office, y un módulo adicional de Add-in desarrollado para Autodesk Revit. Un diagrama general de la arquitectura del sistema se muestra en la Ilustración 1.

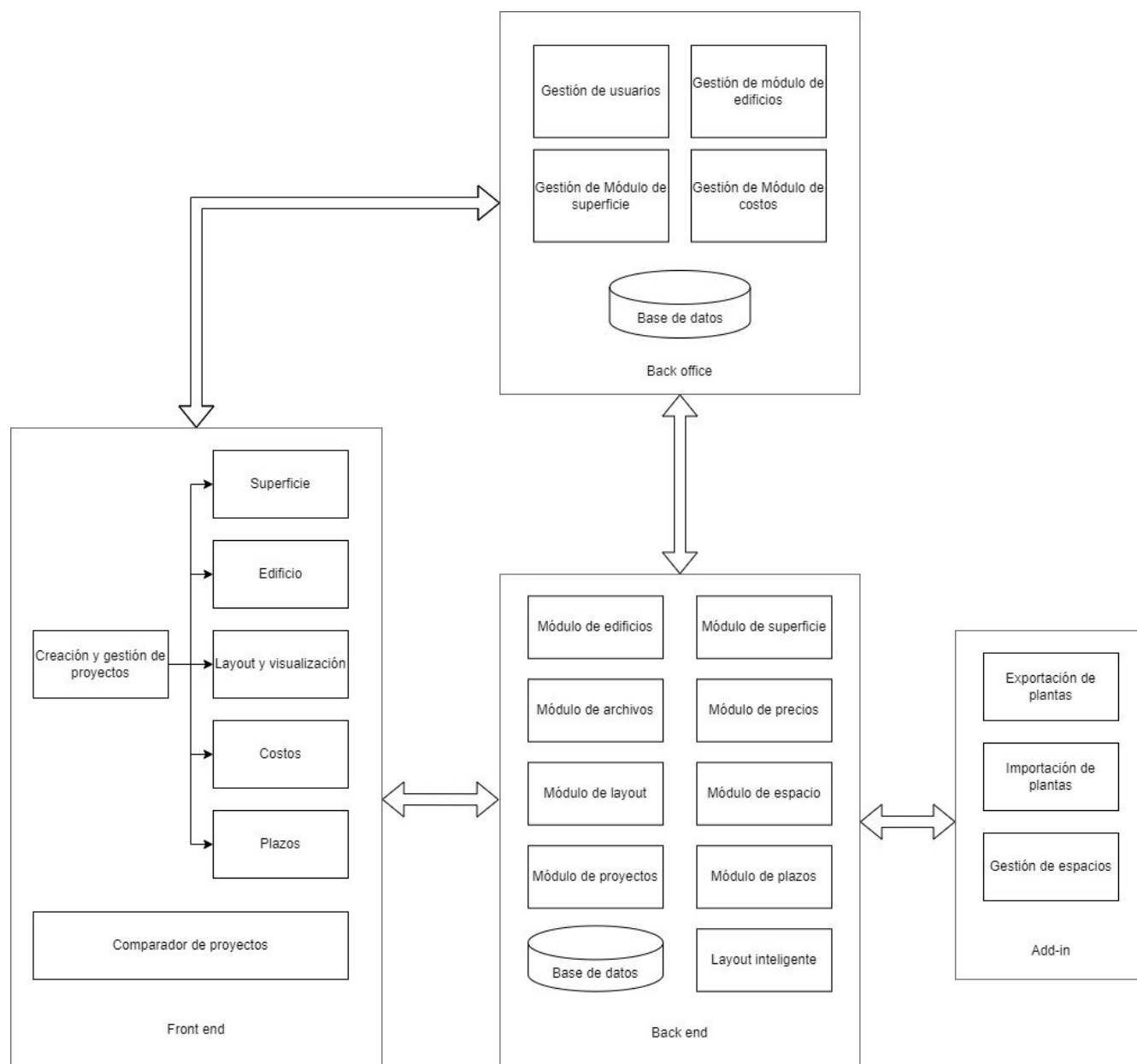


Ilustración 1 – Diagrama de conexión del sistema

2.1 Front end

Es la aplicación web visible para el usuario. Este elemento del sistema se encarga de la gestión de los proyectos y al llamado del módulo de Layout Inteligente. El frontend se encarga, específicamente, de configurar la superficie, el edificio, la estimación de costos y la estimación de costos de los proyectos, además de poder visualizar y editar el layout del mismo. El front end, a su vez, es responsable de invocar al módulo de Layout Inteligente en el Back end.

El front end fue desarrollado en Javascript, utilizando el framework de React.JS como base para su desarrollo. La aplicación es compatible con el estilo de arquitectura de software API REST para la interacción con el Back end y el Back office.

2.2 Back end

El back end se encarga de controlar el sistema de forma centralizada. Principalmente el Back end administra todas las operaciones necesarias en los proyectos para su creación y gestión. Está desarrollada bajo una arquitectura de microservicios independientes en forma de módulos, los cuales están comunicados entre sí.

Se desarrolló en Python usando el framework Flask como base de su desarrollo, el cual a su vez incorpora el framework de SQLAlchemy para el manejo de la base de datos.

2.2.1 Manejo en base de datos

Se utiliza una base de datos relacional, específicamente MySQL. Para la programación de esta en el Back end, se utiliza la técnica de programación ORM (Object-Relational Mapping) implementado en el framework SQLAlchemy. Esto, a su vez, permite un manejo de la base de datos sin la necesidad de usar SQL puro para una comunicación con la base de datos, y permite integrar la interacción con esta de forma nativa al lenguaje de programación de Python.

Además, el sistema al iniciar comprueba automáticamente la existencia de las tablas requeridas, creándolas (junto a sus columnas) en caso de ser necesario.

2.2.2 Servicio RESTful y autenticación

Flask permite manejar el estilo de arquitectura de software API REST de forma segura a través de tokens JWT. Esto para garantizar que sólo usuarios autorizados puedan acceder a los servicios de la plataforma.

Para ello, cada solicitud HTTP debe ir junto a un token JWT en el Header (específicamente, en la llave "Authorization"). Este token se genera por el Back office al iniciar sesión.

2.3 Back office

Este elemento del sistema se encarga principalmente de la gestión de usuarios, y la edición de parámetros de algunos de los módulos presentes en el Back end, específicamente los módulos de superficie, edificios y de costos. Esta plataforma solamente es accesible por usuarios administradores.

Desarrollado en PHP usando el framework de Laravel, permite además el manejo de bases de datos y comunicación con los microservicios del Back end.

2.3.1 Manejo en base de datos

El Back office maneja su propia base de datos MySQL. Esta funciona de forma sincronizada con la base de datos del Back end, actuando como información de respaldo.

Además, el Back office incorpora una funcionalidad de sincronizar la información de la base de datos del Back end, ya que este último se considera la fuente de información principal en todo el sistema. Esto permite sincronizar información de edificios y superficies con la información almacenada en el Back end.

2.3.2 Servicio RESTful y autenticación

Se incorpora, igualmente como en el Back office, un servicio RESTful por medio de API REST. Es posible así acceder a Endpoints que retornen información contenida en la base de datos local del Back office. Sin embargo, principalmente se utiliza el servicio RESTful del Back office para el inicio de sesión y obtención del token JWT utilizado para acceder al servicio RESTful del Back end.

2.3.3 Comunicación con Back end

La comunicación con el Back end es bidireccional, es decir, puede tanto leer como escribir información entre ambos elementos del sistema. Principalmente la comunicación con el Back end se realiza al configurar los módulos de superficie, edificios y costos. Estos se almacenan tanto en la base de datos MySQL local del Back office, y además se trasladan y almacenan en la base de datos principal del Back end.

2.4 Add-in

El complemento, desarrollado en C# junto con la biblioteca API de Revit, se encarga de la gestión de plantas en el sistema, y a poder importar el proyecto trabajado en la plataforma web al software Autodesk Revit.

Las directrices técnicas del Add-in, así también como sus instrucciones de compilación y uso, se encuentran en el documento “Manual técnico y de usuario Add-in contract”.

3 Requisitos del sistema

3.1 Front end

Para compilar la aplicación del cliente (Front end), la máquina debe contar con los siguientes requisitos de software:

1. Sistema operativo compatible con NodeJS (Windows, MacOS, Linux)
2. NodeJS versión 12.18.4 LTS (versiones superiores pueden ser incompatibles)

Para ejecutar la aplicación del cliente (Front end), la máquina que almacene la aplicación debe contar con los siguientes requisitos mínimos:

1. Sistema operativo compatible con Docker y Docker-compose
2. Procesador con al menos 2.4 GHz y 4 cores
3. 10 GB de almacenamiento
4. 8 GB de memoria RAM
5. Conexión a internet

3.2 Back end

Para la ejecución de los servicios de la plataforma (Back end), junto a las bases de datos necesarias, la máquina virtual o el servidor dedicado debe contar con los siguientes requisitos mínimos:

1. Sistema operativo compatible con Docker, Docker-compose y base de datos MySQL
2. Procesador con al menos 2.4 GHz y 4 cores
3. 50 GB de almacenamiento
4. 16 GB de RAM
5. Conexión a internet

3.3 Back office

Para la ejecución de los servicios de gestión de usuarios y administración de plantas (Back office), junto a las bases de datos necesarias, la máquina virtual o el servidor dedicado debe contar con los siguientes requisitos mínimos:

3. Sistema operativo compatible con Docker, Docker-compose y base de datos MySQL
4. Procesador con al menos 2.4 GHz y 4 cores
5. 50 GB de almacenamiento
6. 8 GB de RAM
7. Conexión a internet

4 Compilación del código fuente

4.1 Front end

Para la compilación del Front end, se requiere previamente tener instalado NodeJS en su versión 12.18.4 LTS.

Una vez cumplido el requisito, abrir una ventana de comandos en la carpeta raíz del código, e inicializar el proyecto descargando las dependencias usando el comando:

```
npm install
```

Esto descargará cada una de las bibliotecas utilizadas en el Front end. Una vez terminados, minimizar la ventana de comandos.

4.1.1 Configuración de URLs e IDs de Facebook y Google

En el proyecto, estos parámetros están definidos en el archivo `config.js` ubicado en `./src/config.js`. A modo de ejemplo, una muestra del contenido de dicho archivo se muestra a continuación:

```
var prod_urls = {
  auth_backend: "https://backcontract.want.cl/api",
  backend: "https://wysapi.ac3eplatforms.com/api"
}
var dev_urls = {
  auth_backend: "https://backcontractdev.want.cl/api",
  backend: "https://wysdev.ac3eplatforms.com/api"
}

export var urls = dev_urls;
export var params = {
  facebook_app_id: 1585807258292720,
  google_app_id: "704981360106-791lp7aucs2fdd7ijp5gbea1gqoj78p2.apps.googleusercontent.com"
};
```

Sobre la estructura del archivo y sus variables relevantes, se pueden notar las siguientes consideraciones:

- La llave `auth_backend` es la dirección web del Back office, utilizado por el Front end para el inicio de sesión, registro y olvido de contraseña.
- La llave `backend` es la dirección web del Back end, utilizado por los diversos módulos en la gestión de proyectos.
- La llave `facebook_app_id` es el identificador entregado por Facebook Developers para el inicio de sesión por medio de Facebook. Este ID debe ser generado cada vez que se sube el sitio a una nueva dirección.
- La llave `google_app_id` es el identificador entregado por Google Cloud para el inicio de sesión por medio de una cuenta Google. Este ID debe ser generado cada vez que se sube el sitio a una nueva dirección.
- Las URL de Back office y Backend a utilizar se establecen en la variable `urls`. Esto da posibilidad que se tengan diferentes URL para desarrollo o producción.

Luego de realizar dichos ajustes, se procede a cambiar la API Key para utilizar los servicios de Google Maps, utilizados en la gestión de proyectos. Esta llave se obtiene de la plataforma de Google Cloud Console¹, a lo cual se genera de forma única para cada dirección web nueva donde la plataforma se aloje.

Para editar el archivo, se debe abrir el código en “./src/components/CompareProjects.js”. Luego, se debe buscar el siguiente fragmento de código:

```
{this.state.buildingDetails ? (  
  <LoadScript  
    googleMapsApiKey="AIzaSyD0p2UYSQf5z10PtOD63wVs8gLbEhXP8FM"  
  >  
    <PopupBuildingDetail  
      building={this.state.buildingDetails}  
      onDismiss={() =>  
        this.setState({ buildingDetails: null })  
      }  
    />  
    </LoadScript>  
  ) : null}
```

A lo cual la variable `googleMapsApiKey` contiene la llave generada en Google Cloud Console.

4.1.2 Compilación de código

Una vez configurados los parámetros anteriores, en la ventana de comandos previamente abierta (para descargar las dependencias del proyecto), introducir el comando

```
npm build
```

A lo cual generará los archivos HTML y JS compilados en la carpeta “./build”

4.2 Back end

El Back end, al ser desarrollado en Python, no necesita de compilación, pudiendo directamente instalarse en el servidor.

4.3 Back office

El Back office, al ser desarrollado en PHP usando Laravel, no necesita de compilación, pudiendo directamente instalarse en el servidor

¹ Véase <https://developers.google.com/maps/documentation/javascript/get-api-key>

5 Instalación del sistema

Una vez compilados los elementos del sistema, es posible instalar el sistema. Como se indicó en los puntos anteriores, el sistema corre sobre contenedores Docker en su totalidad.

Es importante seguir los siguientes pasos en el orden propuesto, para así garantizar un correcto funcionamiento y puesta en marcha del sistema

5.1 Back end

Como se indicó anteriormente, el Back end no es compilado al estar desarrollado en Python, por lo que su instalación es directamente en el servidor. Cabe señalar que el servidor debe ya tener instalado Docker y Docker compose antes de proceder a la instalación del Back end.

En primer lugar, se copian todos los módulos y archivos al servidor (por ejemplo, por medio de scp). Los módulos necesarios para el funcionamiento del sistema se muestran a continuación

```
sw-backend-deploy/  
sw-backend-module-building/  
sw-backend-module-filestorage/  
sw-backend-module-layout/  
sw-backend-module-m2/  
sw-backend-module-prices/  
sw-backend-module-project/  
sw-backend-module-space/  
sw-backend-module-times/
```

Ilustración 2 – módulos del Back end

Además, entre los archivos del Back end se encuentra el manifiesto “docker-compose.yml” en la raíz del proyecto. Este es el encargado de indicar a Docker el cómo crear los contenedores, cuáles serán las variables de entorno de cada uno de ellos, y cuáles serán sus parámetros de conexión entre ellos y al exterior.

Antes de construir los contenedores, se deben configurar las variables de entorno de cada módulo. Básicamente, estas variables de entorno indicarán la dirección de la base de datos, la dirección de los otros módulos, y el puerto a utilizar de forma interna y externa. A modo de ejemplo, se muestra un ejemplo del contenido de “docker-compose.yml”, el cual muestra el manifiesto del módulo de superficie:

```
version: "3"  
services:  
  module-m2:  
    build: ./sw-backend-module-m2  
    restart: always  
    image: ac3e/wys-module-m2:1  
    ports:  
      - "8082:8082"  
    depends_on:  
      - "db"  
    environment:  
      DB_USER: "wys"  
      DB_PASSWORD: "rac3e/07"  
      DB_IP_ADDRESS: "db"  
      DB_PORT: "3306"  
      DB_SCHEMA: "wys"  
  
      SPACES_MODULE_IP: "module-space"  
      SPACES_MODULE_PORT: "8083"
```

```
SPACES_MODULE_API_CREATE: "/api/spaces/create"

PROJECTS_MODULE_HOST: "module-project"
PROJECTS_MODULE_PORT: "8081"
PROJECTS_MODULE_API: "/api/projects/"

PRICES_MODULE_HOST: "module-price"
PRICES_MODULE_PORT: "8088"
PRICES_MODULE_API: "/api/prices/"
PRICES_MODULE_API_EXISTS: "/api/prices/exists"
```

Sobre los parámetros relevantes, se pueden definir:

- ports: son los puertos con el formato "exteriorX:interiorY". Es decir, que el puerto exteriorX se mapeará con el puerto interiorY dentro del contenedor.
- DB_USER: Usuario de la base de datos
- DB_PASSWORD: Contraseña de conexión a la base de datos
- DB_IP_ADDRESS: Dirección web de la base de datos. En el caso del ejemplo, la dirección es "db" ya que se deja a docker que automáticamente busque la dirección web del servicio "db"
- DB_PORT: Puerto de conexión a la base de datos
- DB_SCHEMA: Esquema a trabajar en la base de datos
- SPACES_MODULE_IP: Dirección del módulo de espacios. En el caso del ejemplo, la dirección es "module-space", ya que dicho módulo está definido como "module-space" en el manifiesto, caso idéntico a la variable DB_IP_ADDRESS.
- SPACES_MODULE_PORT: Puerto de conexión al módulo de espacios
- SPACES_MODULE_API_CREATE: Dirección relativa a un endpoint del módulo de espacios
- PROJECTS_MODULE_HOST: Dirección del módulo de proyectos
- PROJECTS_MODULE_PORT: Puerto del módulo de proyectos
- PROJECTS_MODULE_API: Raíz del API del módulo de proyectos
- PRICES_MODULE_HOST: Dirección del módulo de precios
- PRICES_MODULE_PORT: Puerto del módulo de precios
- PRICES_MODULE_API: Raíz del API del módulo de precios
- PRICES_MODULE_API_EXISTS: Dirección relativa a un endpoint del módulo de precios

A lo largo del manifiesto, están definidos parámetros similares para cada módulo, dependiendo de las conexiones y dependencias entre módulos. Es importante recalcar que en las variables de Host o IP están los nombres de los módulos. Esto se realiza para dedicar la tarea a Docker de interconectar las IPs de los diferentes contenedores. Sin embargo, si se desea externalizar completamente los microservicios en servidores diferentes, se debe introducir la dirección IP del servidor correspondiente, en vez de la etiqueta docker del servicio (por ejemplo, <http://8.8.8.8> en vez de "module-price").

Una vez configuradas todas las variables de entorno, abrir una ventana de comandos y conectarse por SSH al servidor donde se alojó el código fuente, y ejecutar el siguiente comando en la raíz del proyecto para construir los contenedores:

```
docker-compose build
```

Finalmente, en la misma ventana de comandos ejecutar el siguiente comando para levantar todos los contenedores recién creados:

```
docker-compose up -d
```

En consecuencia, el Back end estará listo para recibir peticiones.

Si se desea actualizar un módulo en específico, se deben ejecutar los siguientes comandos, luego de subir los archivos actualizados al servidor:

```
docker-compose build module-X
docker-compose up -d module-X
```

Donde X es el nombre del módulo a actualizar, según su etiqueta del servicio en el manifiesto “docker-compose.yml”

5.2 Back office

Este elemento del sistema requiere principalmente dos pasos para poder inicializarse: la construcción y despliegue del contenedor Docker, y la inicialización de los servicios PHP dentro de este.

En primer lugar, se deben copiar todos los archivos al servidor (por ejemplo, por medio de SCP). Un ejemplo de listado de archivos en el directorio raíz se observa en la Ilustración 3.

app	composer.lock	docker-compose.yml	phpunit.xml	resources	vendor
artisan	config	Dockerfile	Procfile	routes	webpack.mix.js
bootstrap	database	docker-volumes-data	public	server.php	
composer.json	docker-compose	package.json	README.md	storage	

Ilustración 3 – Ejemplo de archivos en Back office

5.2.1 Configuración de Docker

Dentro de los archivos del código fuente, se entrega el manifiesto “docker-compose.yml”. Al igual que el Back end, este permite la fácil configuración de las variables de entorno y puertos de conexión. En concreto, el Back office se compone de tres módulos principales: el servicio PHP, la base de datos, y el servidor web NGINX. Se detalla a continuación la configuración de los parámetros de cada módulo en el manifiesto:

5.2.1.1 Servicio PHP

El servicio PHP se encuentra en el manifiesto con la etiqueta de servicio “app”. A continuación, se muestra un ejemplo de configuración en el manifiesto.

```
app:
  build:
    args:
      user: sammy
      uid: 1000
    context: ./
    dockerfile: Dockerfile
  image: back-office
  container_name: back-office-app
  restart: unless-stopped
  working_dir: /var/www/
  volumes:
    - ./var/www
  networks:
    - back-office
```

Básicamente, no hay variables de entorno configurables, ya que sólo se detalla un usuario para el contenedor Docker con nombre “sammy”.

5.2.1.2 Base de datos

La base de datos, a diferencia del servicio PHP, sí tiene variables de entorno a configurar. A continuación, se muestra un ejemplo del servicio de base de datos en el manifiesto:

```
db:
  image: mysql:5.7
  container_name: back-office-db
  restart: unless-stopped
  ports:
    - "3388:3306"
  volumes:
    - ./docker-volumes-data/db:/var/lib/database
  environment:
    MYSQL_DATABASE: back_office
    MYSQL_ROOT_PASSWORD: root
    MYSQL_PASSWORD: root
    MYSQL_USER: user
    SERVICE_TAGS: dev
    SERVICE_NAME: mysql
  networks:
    - back-office
```

Sobre las variables de entorno, estas establecen los parámetros de conexión para los demás módulos a la base de datos. Es decir, establecen el nombre del esquema, usuario y contraseña de la base de datos. En concreto, las variables de entorno se explican como:

- MYSQL_DATABASE: Nombre de la base de datos (esquema)
- MYSQL_ROOT_PASSWORD: contraseña del usuario root de la base de datos
- MYSQL_PASSWORD: contraseña de usuario por defecto en la base de datos
- MYSQL_USER: nombre de usuario por defecto en la base de datos
- SERVICE_TAGS: tags que se utilizan para la base de datos.
- SERVICE_NAME: nombre del servicio de la base de datos.

5.2.1.3 Servidor web NGINX

El Back office utiliza NGINX como servidor web. Este gestiona las páginas web y permite su visualización, tomando el servicio PHP como complemento.

```
nginx:
  image: nginx:1.17-alpine
  container_name: back-office-nginx
  restart: unless-stopped
  ports:
    - 8000:80
  volumes:
    - ./:/var/www
    - ./docker-compose/nginx:/etc/nginx/conf.d
  networks:
    - back-office
```

Básicamente no hay variables de entorno configurables. Se recomienda solamente cambiar el puerto, en caso de ser necesario.

5.2.2 Configuración de parámetros PHP

Antes de la construcción de los contenedores docker y a la inicialización de los servicios PHP, se deben configurar las variables de entorno, ubicadas en el archivo “.env.example”. El significado de cada variable de entorno se define en el documento “Documentación BackOffice Contract”, redactado por Want Ltda, adjunto en el presente manual. Es importante verificar que las direcciones URL de la base de datos y el Back end, así también como los parámetros de conexión hacia la base de datos (definidas en el manifiesto de docker compose) estén correctas.

Una vez configuradas las variables de entorno según las directrices del documento, en una ventana de comandos en la raíz del proyecto ejecutar:

```
cp .env.example .env
```

Lo que, el archivo recién editado pasará a ser un archivo de variables de entorno para el servicio PHP.

5.2.3 Construcción de contenedores docker

La construcción de los contenedores se realiza bajo docker compose. Abrir una ventana de comandos, y conectarse por SSH al servidor donde se aloja el Back office. Luego, ejecutar el siguiente comando:

```
docker-compose build
```

Esto construirá los contenedores de los tres módulos del Back office. Luego, levantar los contenedores creados usando el siguiente comando:

```
docker-compose up -d
```

Así, los tres contenedores estarán operativos.

En caso de que se quiera actualizar y/o modificar algún módulo, primero se debe ejecutar el siguiente comando luego de subir los cambios, para reconstruir un módulo en específico:

```
docker-compose build X
```

Donde X puede ser cualquiera de los tres módulos del Back office, identificándose con la etiqueta del servicio en el manifiesto “docker-compose.yml”. Luego, levantar el contenedor utilizando el comando:

```
docker-compose up -d X
```

5.2.4 Iniciación de aplicación de Laravel

Una vez contruidos y levantados los contenedores, en la misma sesión SSH, en la misma ubicación que el archivo “docker-compose.yml”, ejecutar el siguiente comando:

```
docker-compose exec app bash
```

Esto permite ingresar al contenedor del servicio PHP mediante bash. Si bien el contenedor del servicio PHP está levantado, es un contenedor sin las dependencias PHP del proyecto, ni con servicios ejecutándose correctamente. En primer lugar, ejecutar el siguiente comando para descargar las dependencias del proyecto:

```
composer install
```


Luego, en la misma sesión ejecutar el siguiente comando que permite inicializar los servicios de autenticación JWT del servicio PHP:

```
php artisan passport:install
```

Luego, ejecutar el siguiente comando que permite hacer un enlace simbólico con el almacenamiento en el contenedor docker y el servicio PHP:

```
php artisan storage:link
```

Finalmente, se inicializa la base de datos utilizando las tablas necesarias por el proyecto. Para este punto es importante haber configurado correctamente las variables de entorno “.env.example” que enlacen a la base de datos del Back office. Una vez comprobado, ejecutar el siguiente comando:

```
php artisan migrate:refresh
```

A lo cual procederá a verificar si las tablas necesarias por el proyecto están creadas, y en caso contrario se crean.

5.2.5 Seeding de la base de datos

Una vez inicializado el servicio PHP, se debe poblar la base de datos antes de acceder al Back office. En primer lugar, en la misma sesión dentro del contenedor del servicio PHP, ejecutar el comando:

```
php artisan db:seed --class="CountriesTableSeeder"
```

Lo que poblará la base de datos con una lista de países. Luego ejecutar el siguiente comando que contiene información sobre ciudades y regiones:

```
php artisan db:seed --class="RegionTableSeeder"
```

Finalmente, se debe crear un usuario administrador. Para ello, modificar y ejecutar el siguiente comando:

```
php artisan orchid:admin admin admin@admin.com password
```

Note que admin@admin.com es un correo de ejemplo, el cual es arbitrario y puede modificarse con cualquier correo. Igualmente, con “password” como contraseña, pudiendo ser arbitraria.

Con estos pasos, ya el Back office se encuentra operativo y listo para ser usado.

5.3 Front end

El Front end, luego de ser compilado, se debe subir dentro de la subcarpeta “nginx” y no en el directorio raíz. Un ejemplo de cómo se debiese ver el directorio raíz del servidor se muestra en la Ilustración 4.

```
[ec2-user@ip-172-31-18-27 ~]$ ls
docker-compose.yml  nginx
```

Ilustración 4 – Archivos dentro del directorio raíz en el Front end

Sin embargo, la carpeta “nginx” es configurable arbitrariamente modificando el manifiesto “docker-compose.yml”, cambiando el directorio en “build”

```
version: "3"
services:
  wys-frontend:
    build: ./nginx
    restart: always
```

```
ports:
  - "80:80"
  - "443:443"
```

Dentro del directorio “nginx”, se tiene la carpeta build proveniente del directorio de compilación del proyecto. El archivo “nginx.conf” es el manifiesto de configuración del servidor nginx, por lo que debe ser configurado previo a construir el contenedor del Front end. Un ejemplo con los archivos y carpetas dentro del directorio “nginx” se observa en la Ilustración 5.

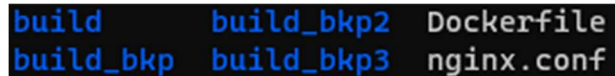


Ilustración 5 – Ejemplo de distribución en carpeta nginx

El sistema busca la carpeta “build”, y el deploy se realiza referenciando a dicho directorio. Sin embargo, esto es arbitrario y configurable en el archivo “nginx.conf”, donde dicha configuración se muestra en la etiqueta de “location”, como se muestra a continuación

```
location / {
    root    /app/build;
    index   index.html index.htm;
    try_files $uri /index.html;
```

Para cambiar el directorio “build”, solamente cambiar el directorio de la etiqueta “root”.

Finalmente, en la consola, en el directorio donde se ubica el manifiesto “docker-compose.yml”, introducir el comando:

```
docker-compose build
```

Para construir el contenedor del Front end, y luego levantar el contenedor con:

```
docker-compose up -d
```

6 Módulos del backend y bases de datos

La arquitectura contempla 2 bases de datos alojadas en distintos servidores, una forma parte del backend y la otra del backoffice.

El sistema de backend, correspondiente al conjunto de microservicios o API que utiliza el sistema, es aquel encargado de hacer la comunicación entre la interfaz del sistema WYS (frontend y backoffice) y addin, con la base de datos, además de proveer de otros servicios que se utilizan en distintos módulos de backend. En la siguiente sección se explicará cómo funciona y la estructura de la base de datos.

6.1 Base de datos del backend

La base de datos del backend contiene toda la información utilizada en WYS, a excepción de la gestión de usuarios, la cual es manejada por la base de datos del backoffice. Está conformada por 35 tablas a través de un modelo semirrelacional; lo llamamos de esta manera porque algunas relaciones entre tablas fueron manejadas a nivel lógico del código del backend, y no fue especificada en el mismo modelo mediante claves primarias y foráneas. Esto se decidió realizar debido a que el backoffice también replicó parte de la estructura de algunas tablas del backend, como respaldo para no tener saturación de consultas.

En la documentación del código del backend, a través de las “**Classes**”, se puede relacionar cuáles tablas están vinculadas a los módulos de backend a través de su nombre. Por ejemplo, en el módulo de backend Layout, se puede encontrar la clase “**Class LayoutGenerated**”, vinculada a la tabla del “**layout_generated**”.

En las siguientes tres ilustraciones se pueden apreciar las tablas del modelo y sus respectivos campos:

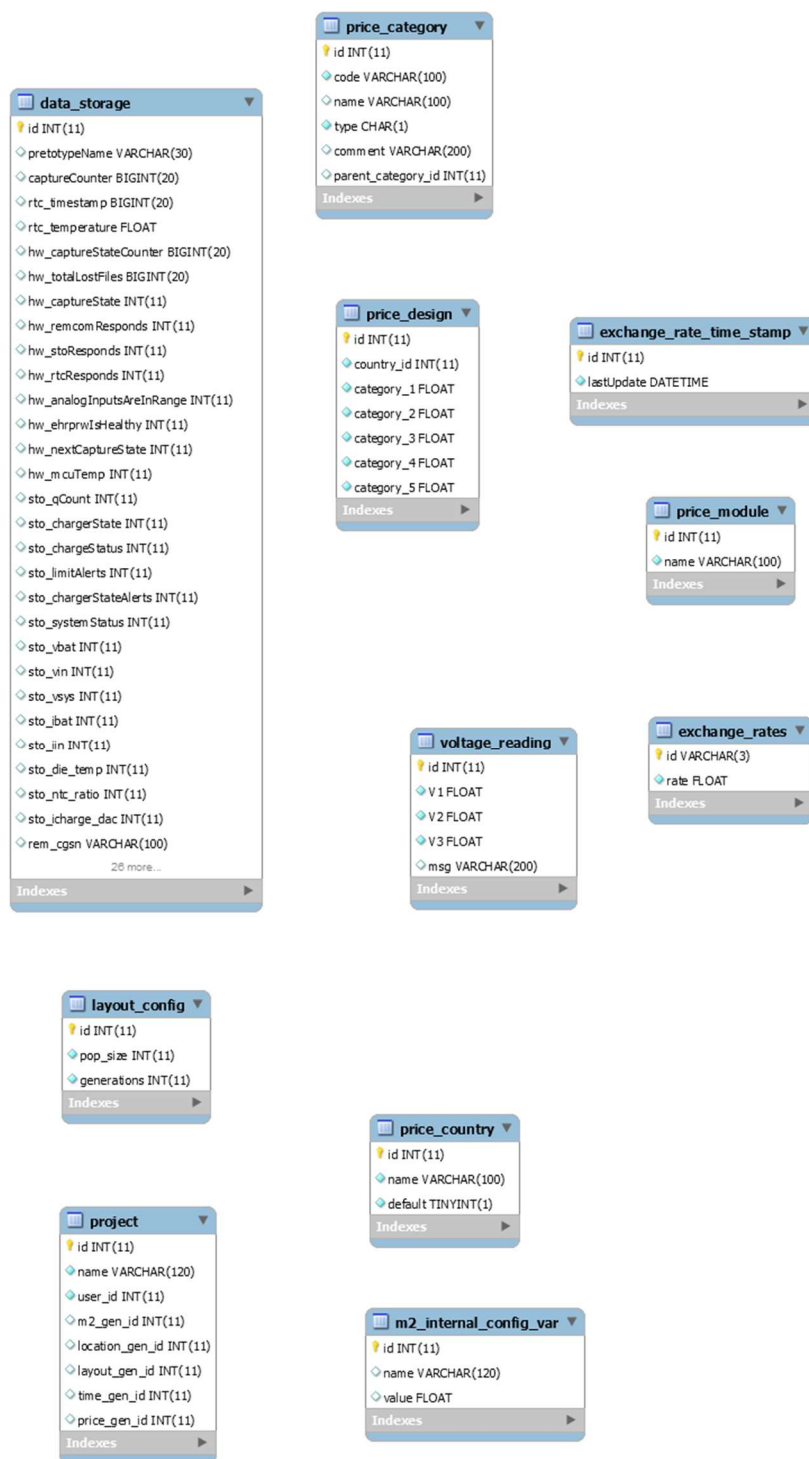


Ilustración 5 – Modelo de datos backend, 1/3

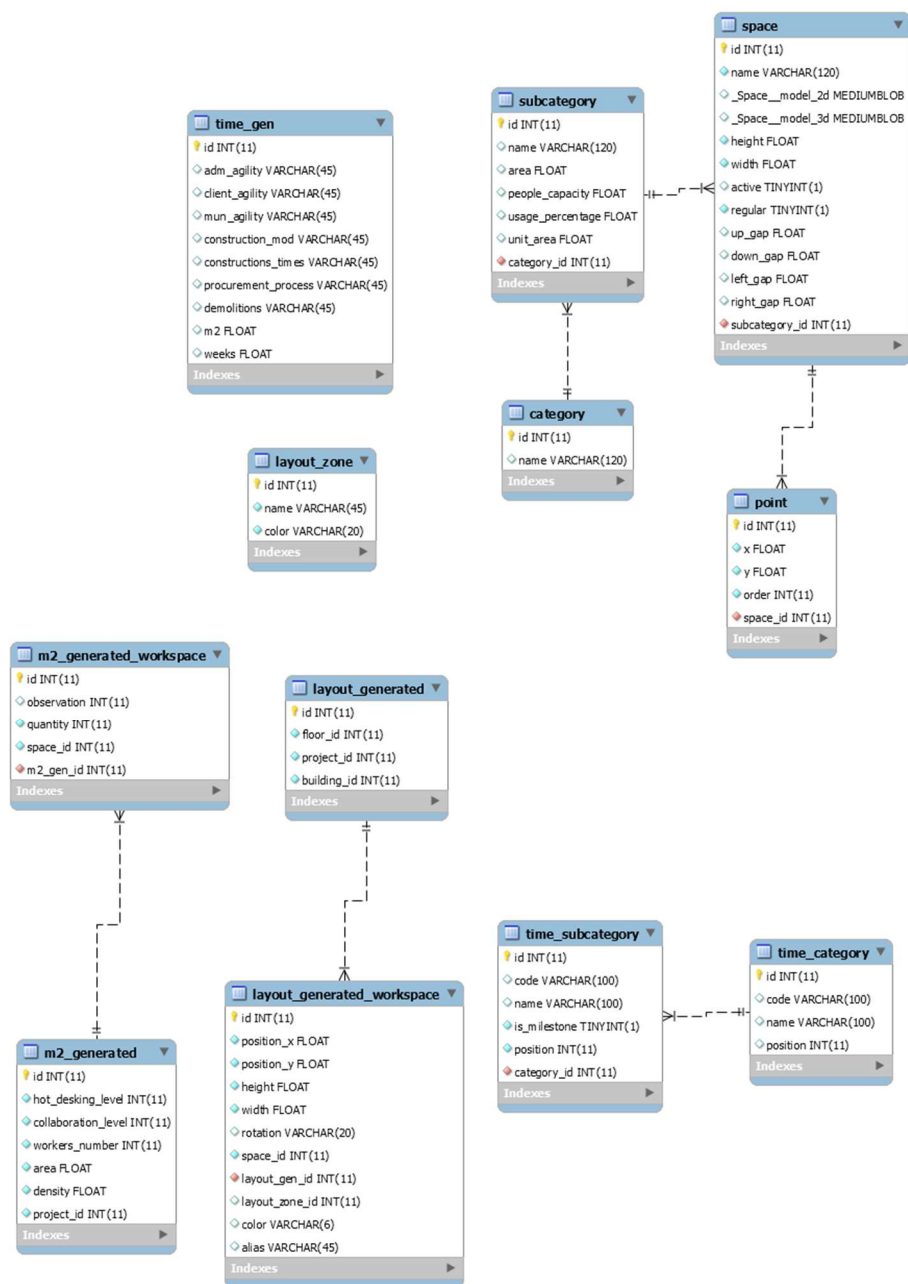


Ilustración 6 – Modelo de datos backend, 2/3

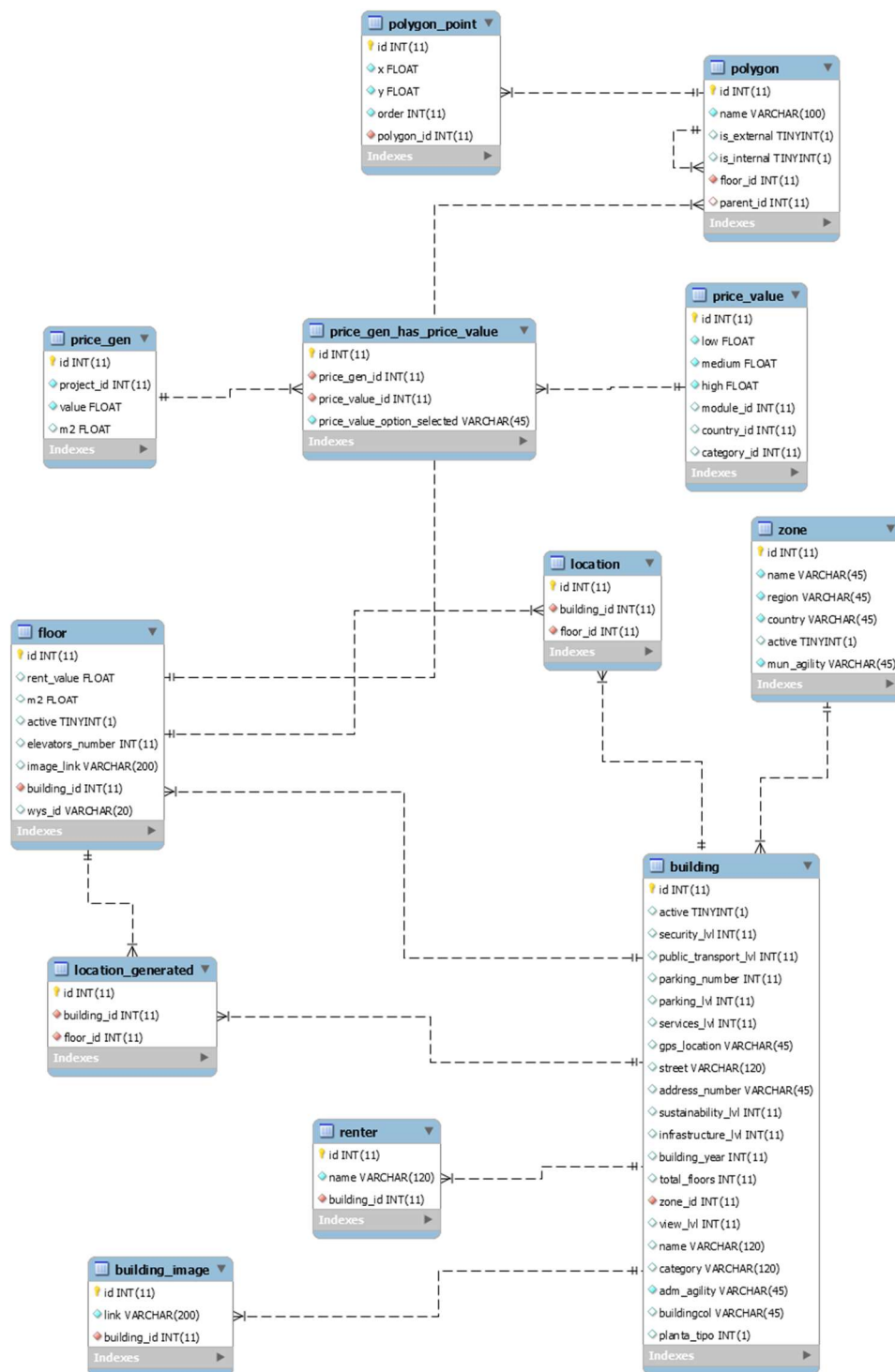


Ilustración 7 – Modelo de datos backend, 3/3

6.2 Base de datos del backoffice

Base de datos cuya finalidad es prestar soporte a su símil en el backend y de realizar toda la gestión de usuarios y autorizaciones de sesiones. Además, presta servicio para gestionar información útil en el sistema, como lo son los países y sus ciudades o zonas.

Está conformada también por un modelo semi-relacional, esto se puede apreciar en las tablas donde da soporte al backend. Contiene 27 tablas. A continuación, se muestran las ilustraciones relativas a este modelo:

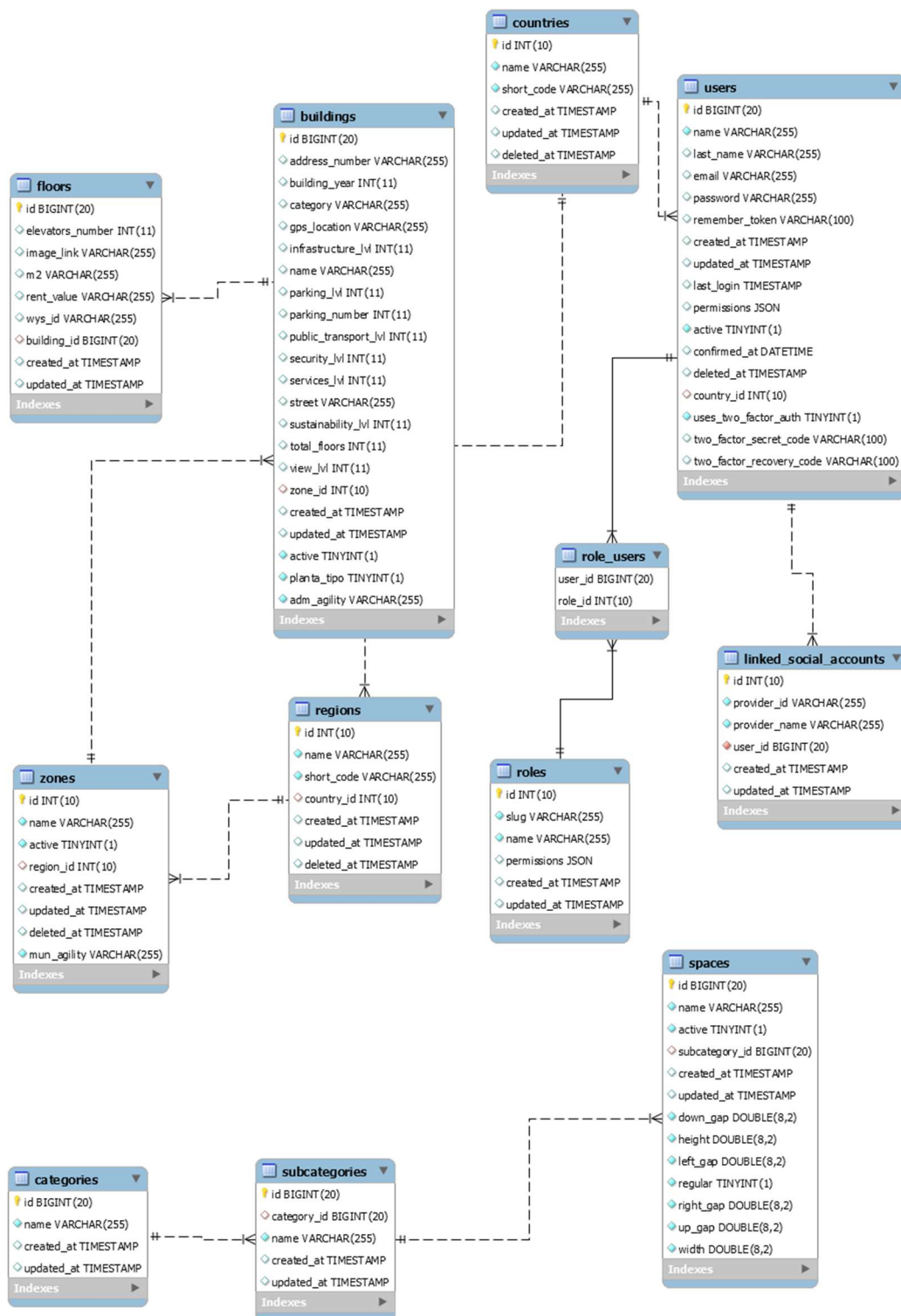


Ilustración 8 – Modelo de datos backoffice, 1/2

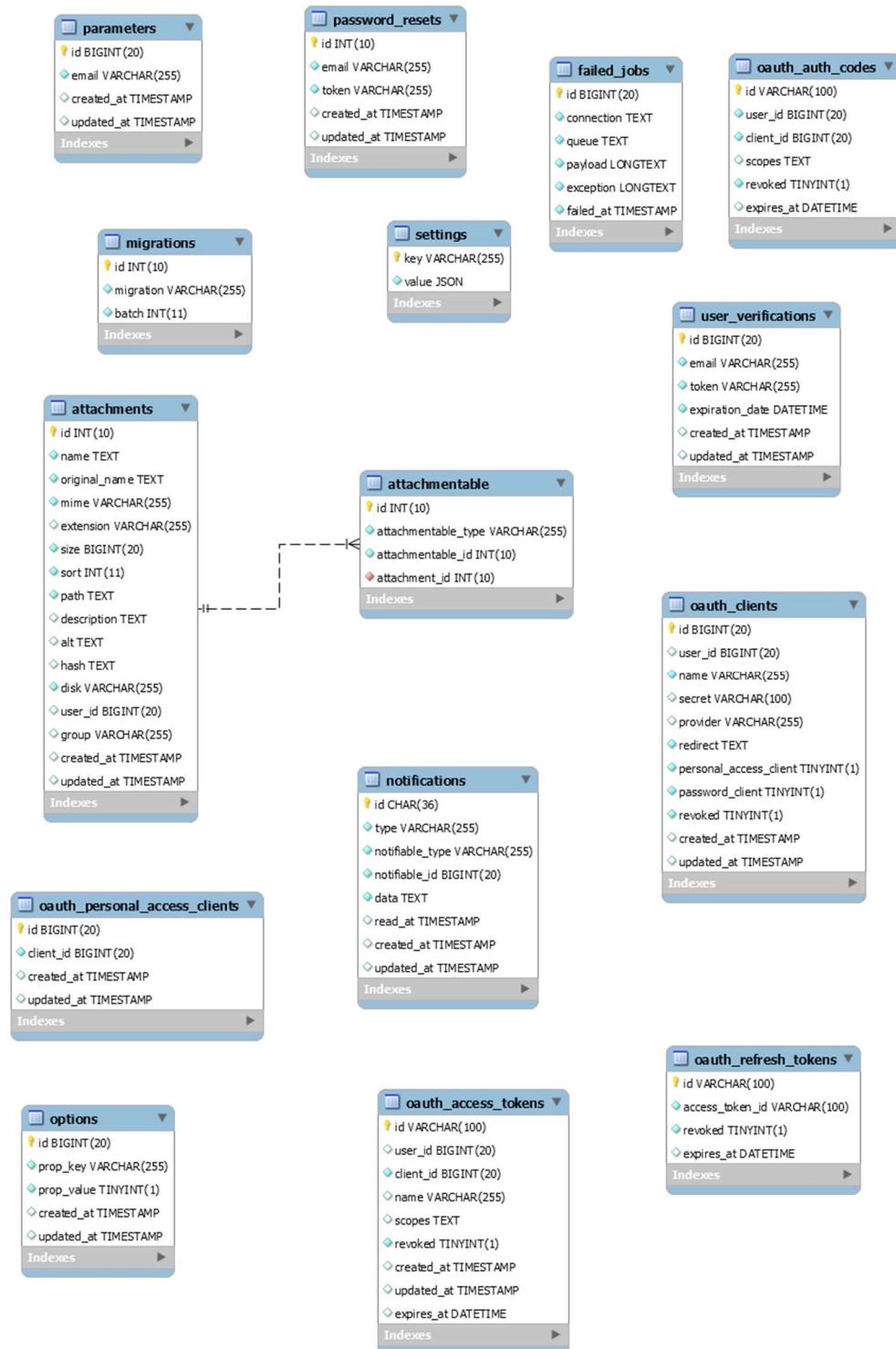


Ilustración 9– Modelo de datos backoffice, 2/2

6.3 Módulos de backend

El backend está compuesto de 8 módulos, cada uno provee un servicio distinto relacionado a un tema particular del sistema. Son los siguientes:

6.3.1 Building

Contiene la API que maneja todo lo relacionado con el módulo de edificios en WYS frontend, WYS backoffice y addin. Realiza comunicación con el módulo de proyectos (project), a través de la función **update_project_by_id** (ver documentación de código).

6.3.2 Filestorage

Contiene la API que maneja todo lo relacionado con la gestión de archivos en WYS backoffice. Se utiliza en aquellos módulos que requieren guardar archivos en el sistema, como sucede al crear un edificio y asociar imágenes; también ocurre al trabajar con las plantas (floor). No se conecta con ningún otro módulo del backend.

6.3.3 Layout

Contiene la API que maneja todo lo relacionado con el módulo de layout en WYS frontend y addin. Realiza comunicación con los módulos de edificios (building), espacios o módulos (spaces), y proyectos (projects). Un ejemplo de comunicación layout y edificios sería la función **get_floor_by_ids**; para layout y espacios **get_space_by_id**; y para layout con proyectos, **get_project_by_id** (ver documentación del código).

También es el responsable de la comunicación con el servicio “job”, el cual actúa en forma asíncrona consultando el porcentaje de avance de la construcción de un layout generado en tiempo real. Otra característica de este módulo es que se comunica con una librería independiente diseñada para hacer el “layout inteligente”, llamado **SmartLayout**.

6.3.4 M2

Contiene la API que maneja todo lo relacionado con el módulo de M2 en WYS frontend. Realiza comunicación con los módulos de espacios (spaces), costos (prices), y proyectos (projects). Un ejemplo de comunicación M2 y espacios sería el *microservicio* **generate_workspaces**; para M2 y costos, la función **exists_price_project_by_id**; y para M2 con proyectos, **update_project_by_id** (ver documentación del código).

6.3.5 Prices

Contiene la API que maneja todo lo relacionado con el módulo de costos en WYS frontend. Realiza comunicación con los módulos de proyectos (projects), espacios (spaces), tiempo o plazos (times), y M2. Un ejemplo de comunicación entre costos y proyectos sería la función **update_project_by_id**; entre costos y espacios, el *microservicio* **get_estimated_price**; costos y tiempo, la función **get_project_weeks**; y, entre costos y M2, **get_workspace_by_project_id** (ver documentación del código).

6.3.6 Project

Contiene la API que maneja todo lo relacionado con los proyectos en el sistema WYS y addin. Realiza comunicación con los módulos de edificios (buildings), M2, costos (prices), tiempo (times), y layout. Un ejemplo de comunicación entre proyectos y edificios sería la función **get_location_by_id**; entre proyecto y M2, **get_m2**; proyecto y costos, **get_price**; y, entre proyecto y tiempo, **get_time** (ver documentación del código).

6.3.7 Space

Contiene la API que maneja todo lo relacionado con los espacios o módulos de un layout en el sistema WYS y addin. Realiza comunicación con el módulo de edificios (buildings) a través de la función ***get_floor_polygons_by_ids*** (ver documentación del código).

6.3.8 Times

Contiene la API que maneja todo lo relacionado con el módulo de tiempos o plazos en WYS frontend. Realiza comunicación con los módulos de proyectos (projects) y edificios (buildings). Un ejemplo de comunicación entre tiempos y proyectos sería la función ***update_project_by_id***; y, entre tiempos y edificios, el ***microservicio get_save_times*** (ver documentación del código).