

Instituto Superior del Profesorado N° 62 “Ángela Cullen”

APLICACIÓN ANDROID



“AREQUITO SHOP”

TÉCNICO SUPERIOR EN ANÁLISIS FUNCIONAL DE SISTEMAS
INFORMÁTICOS

Prácticas Profesionalizantes II

Docente: Piccini Verónica

Alumnos: Scozziero Stefano
DNI: 39.854.971
Scozziero Adriano
DNI: 37.713.134

Febrero 2022



ÍNDICE

RESUMEN	2
INTRODUCCIÓN.....	3
DESCRIPCION DE LA APLICACIÓN	4
CONCEPTOS UTILIZADOS:.....	5
Metodología SCRUM para aplicaciones móviles:.....	7
PRIMEROS PASOS	9
Creación de tablas	11
Diagrama de Entidad-Relación:.....	15
Conexión:.....	16
Funcionamiento de la metodología MVC aplicada al proyecto.	18
Paleta de colores	24
Tipografía:.....	24
Diseño de la interfaz:.....	25
PLAN DE MANTENIMIENTO	29
EL PROYECTO A FUTURO	32
CONCLUSIÓN:	33
AUTORES	34
ANEXO A	35
Referencias:	38

RESUMEN

El presente trabajo de titulación trata sobre la creación de una Aplicación Móvil Android utilizando conceptos y competencias adquiridas durante el cursado de la carrera. También, para su desarrollo, ha sido necesario abordar conceptos relacionados a aplicaciones móviles, sistemas operativos móviles, la conexión a datos desde un dispositivo móvil, servicios de Google y metodologías de aplicaciones móviles.

Este trabajo presenta una síntesis de todos los procedimientos utilizados para el desarrollo de la app, resaltando los resultados obtenidos, figuras, tablas y diagramas.

A continuación se detallarán todos los procedimientos realizados para la creación de este proyecto:

- Introducción, detallaremos la problemática a resolver.
- Descripción del proyecto y sus objetivos.
- Definición de los conceptos adquiridos y utilizados para el desarrollo de la aplicación.
- Desarrollo, especificaremos los procedimientos más influyentes para la creación del proyecto en general.
- Presentación del Manual de Usuario.
- Plan de mantenimiento y optimización.
- El proyecto a futuro.
- Conclusión.

INTRODUCCIÓN

Actualmente, vivimos en mundo apresurado donde el tiempo que disponemos es poco, estamos cada vez más agobiados y saturados ya sea por los horarios de una jornada laboral extensa o simplemente porque los horarios de trabajo o de cursado no coinciden con el de la mayoría de personas.

Por ello, se ha propuesto la realización de este proyecto cuyo objetivo principal es desarrollar una aplicación móvil Android que integre todos los locales, comercios y farmacias que deseen participar, pudiendo brindar al cliente todos los datos necesarios sobre los productos que ofrezcan. De esta manera el cliente puede elegir un producto comparando su precio entre los distintos tipos de negocios, todo desde la comodidad de su hogar.

ArequitoShop es una aplicación móvil Android que permite comerciar de forma directa cualquier tipo de artículo que se desee vender. Por el momento está dirigida hacia el pueblo de Arequito y sus alrededores. Su finalidad es dar solución a la problemática de no poder salir de tu casa o estar con el tiempo limitado para salir a comprar lo que necesitas. Cuenta con un sistema de gestión y logística optimizado para lograr la máxima eficiencia y eficacia. Incluye servicio de delivery.

Por otra parte, para una persona que quiera asociarse como comerciante, se propuso el objetivo de crear y listar cada producto por categoría, poder editar cada artículo de la manera más conveniente para que cada local registrado pueda adaptarse.

Asimismo, esta aplicación ofrece empleos con horarios flexibles a personas que dispongan de un medio de transporte.

DESCRIPCION DE LA APLICACIÓN

ArequitoShop es un E-Commerce cuyo objetivo es agilizar y simplificar las compras y ventas creadas en el pueblo y sus alrededores, puede utilizarse de tres maneras diferentes:

Cliente/Comprador: Es un rol asignado por defecto al registrarse en la aplicación. Desde esta perspectiva, el cliente puede navegar entre los distintos comercios adheridos y elegir sus productos deseados que se verán listados por categorías. Cada producto que desee comprar se agregará a un carrito de compras, de esta manera se creará un conjunto de artículos y puede realizar el pago por todos ellos al mismo tiempo. Podrá ver su orden de compra creada y darle seguimiento en tiempo real, desde el despacho hasta que llegue a destino.

Comerciante/Vendedor: Cada comerciante deberá llenar un formulario para poder adherirse y comenzar a vender sus productos, podrá crear su perfil de vendedor, crear sus categorías y comenzar a ingresar sus productos. No se preocupe, si usted no puede o no dispone tiempo para crear su perfil, nosotros lo haremos por usted. En este rol, el comerciante esperará una notificación que se crea al momento de que un cliente efectúa el pago, dicha notificación traerá la orden de los productos creados para que comience a preparar el pedido. Una vez creado el pedido, deberá asignar un repartidor para que comience la entrega.

Repartidor/Delivery: El repartidor estará atento a que el vendedor le asigne un pedido, le llegará una notificación y deberá dirigirse al sitio de origen donde se encuentra la orden despachada, una vez que tenga el pedido, tendrá que iniciar la entrega en la aplicación, de esta manera podrá ser localizado vía GPS en todo momento y obtendrá un mapa con la ruta más cercana marcada.

CONCEPTOS UTILIZADOS:

Base de Datos: Una base de datos o banco de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

SQL: (por sus siglas en inglés Structured Query Language; en español lenguaje de consulta estructurada) es un lenguaje de dominio específico, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.

Java Script: (JS) es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase.

Kotlin: es un lenguaje de programación de código abierto que admite la programación funcional y orientada a objetos. Proporciona una sintaxis y conceptos similares a los de otros lenguajes, como C#, Java y Scala, entre muchos otros.

Un entorno de desarrollo integrado (IDE) es un sistema de software para el diseño de aplicaciones que combina herramientas comunes para desarrolladores en una sola interfaz de usuario gráfica (GUI).

MVC (Modelo-Vista-Controlador) es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización.

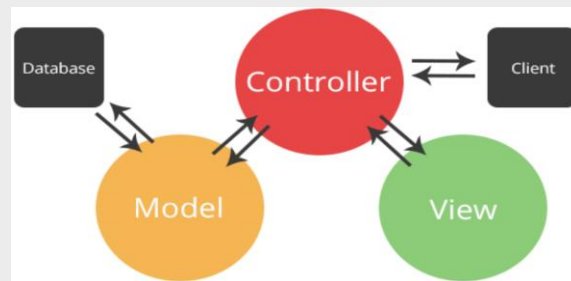
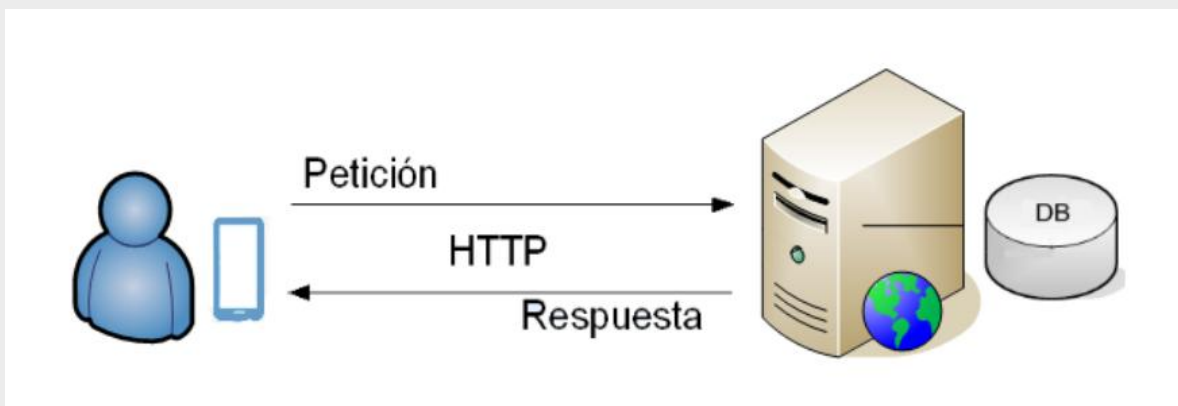


Imagen ilustrativa del modelo MVC

La programación asíncrona nos da la capacidad de “diferir” la ejecución de una función a la espera de que se complete una operación, normalmente de I/O (red, disco duro, etc.), y así evitar bloquear la ejecución hasta que se haya completado la tarea en cuestión.

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful.



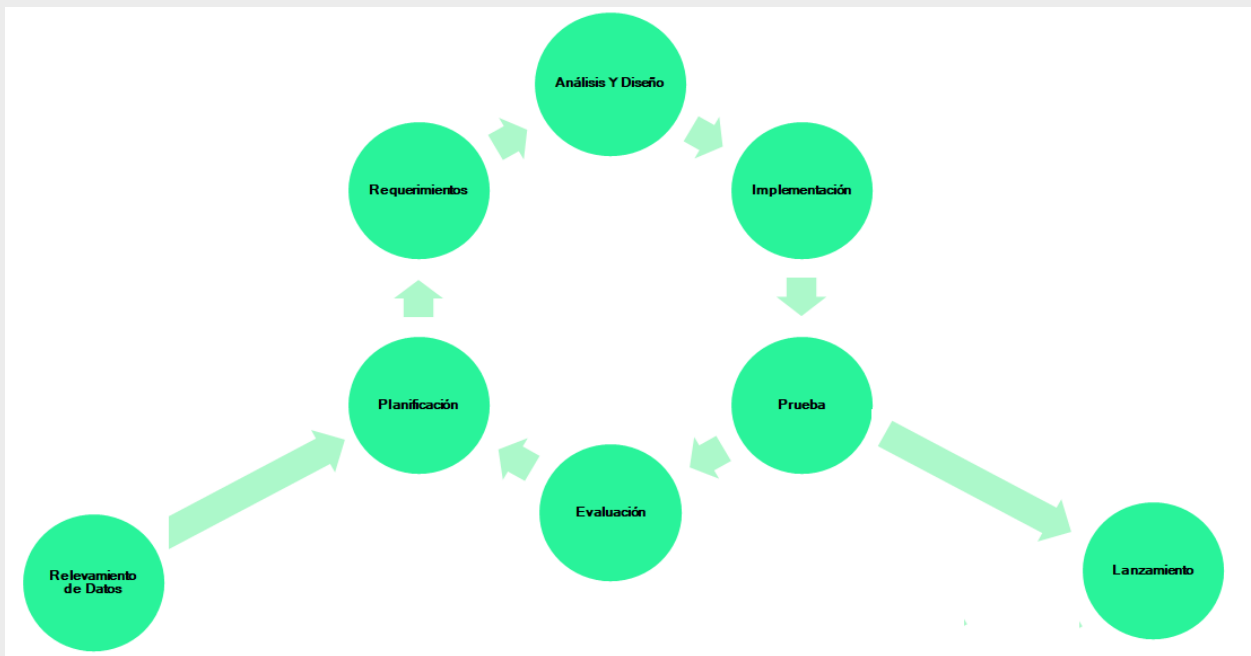
Aplicación Móvil: Las aplicaciones móviles son programas software que pueden ser descargadas y a las que se puede acceder directamente desde un teléfono o desde algún otro dispositivo móvil, como por ejemplo una tablet o un reproductor de música.

Android es un sistema operativo móvil basado en Linux enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, tabletas, Google TV y otros dispositivos. Es desarrollado por la Open Handset Alliance, liderada por Google.

Google Maps Api Google Maps API se trata de una tecnología que permite la visualización de Google Maps en páginas web con Java Script. El API proporciona unas determinadas herramientas para interaccionar con los mapas y añadir contenido a los mismos a través de una serie de servicios, permitiendo llegar a crear aplicaciones con mapas de gran complejidad y robustez.

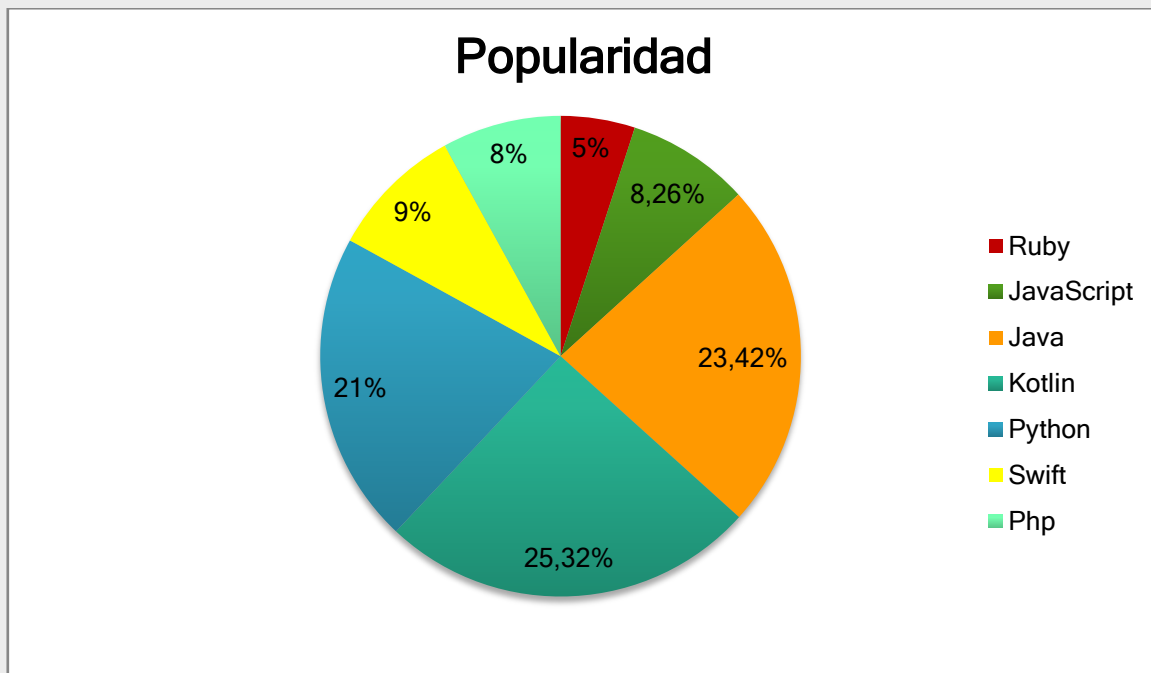
Metodología ágil SCRUM para aplicaciones móviles:

FASES	Se define por Sprints de dos semanas aproximadamente y de un producto entregable.
COMUNICACIÓN CON EL CLIENTE	El Product Owner maneja la comunicación con el cliente
PROGRAMACIÓN	El tiempo de programación se determina de acuerdo a la puntuación dada a cada tarea.
DOCUMENTACIÓN	Lo principal es utilizar GitHub y documentación nativa aportada por cada implementación
PRUEBAS	Evaluación de pruebas y análisis de resultados



PRIMEROS PASOS

Antes de comenzar a programar, se realizó un análisis total con el objetivo de seleccionar el lenguaje de codificación más óptimo que se adapte a la tecnología utilizada en la actualidad. Por ello realizamos un gráfico comparativo entre los lenguajes más utilizados con el fin de seleccionar el de conveniencia.



También se tuvo en cuenta cuál de éstos era el más adecuado para trabajar con SQL, lenguaje de Base de Datos aprendido en el transcurso de la carrera.

Luego se observó que los lenguajes principales y recomendados por Google para desarrollar aplicaciones móviles son JAVA y KOTLIN, entonces se procedió a diferenciar ventajas y desventajas entre estos al momento de programar.

FUNCIONES	JAVA	KOTLIN
Fully OOP	No es pura	Completa
Null Safety	No	Si
Checked Exception	Si	No, utiliza corrutinas
Invariant Array	No	Si
Smart Casts	No	Si
Lambda Expression	No	Si
Singletons Object	Si, con dificultad	Facilidad al crear objetos
Funcion Reactive Programming	No	Si

El lenguaje Kotlin, tiene como cualidad ser conciso, simplifica el trabajo como desarrollador y mitiga el riesgo de error. La interoperabilidad es el propósito de Kotlin, simplemente podemos escribir módulos en Kotlin que funcionen con Java, esto no podemos hacerlo en Java. Posee seguridad nula incorporada, evitando que los desarrolladores escriban código adicional para solucionar los NullPointerException.

Por estas razones, el lenguaje seleccionado para comenzar a trabajar fue Kotlin.

También se ha utilizado Java Script como intermediario, con este lenguaje se creó un API REST, cuyo objetivo es conectar todas las funciones y peticiones desde la aplicación hacia la base de datos mediante un servidor local.

Para lograr esto de una forma directa y simplificada, se aplicó la metodología MVC, en la cual se crea un modelo de Base de Datos que contiene todas las QUERYS utilizadas, un controlador, que contiene todas las funciones y una vista en la que se solicita información al cliente a través de una ruta que es el medio de comunicación. De esta manera se crea el Backend de nuestra Aplicación. Utilizamos PostgreSQL como gestor de la base de datos.

Como primera instancia al momento de crear el backend de la aplicación, se realiza un relevamiento de datos a fondo y trabajamos en él, logrando un nivel de normalización de 4ta forma.

Creación de tablas:

Tabla users:

id [PK] bigint	email character varying (80)	name character varying (80)	lastname character varying (80)	phone character varying (30)	image character varying (255)
password character varying (80)	is_available boolean	session_token character varying (255)	notification_token character varying (255)	created_at timestamp without time zone	updated_at timestamp without time zone

En esta tabla se detallan todos los datos del usuario, al momento de registrarse los datos que deberá ingresar son: Nombre, Apellido, Email, Teléfono, Imagen y Contraseña. Una vez registrado, se le asignarán por defecto una Id, un token de sesión y un token de notificación, más adelante explicaremos la función de éstas columnas. Las columnas created_at y updated_at, se actualizarán respectivamente con la fecha y hora exacta en el momento de creación de la cuenta o en el momento de actualizar algún dato.

Como se menciona anteriormente, cada usuario puede tener un rol en específico, por esta razón se crea la tabla roles:

id [PK] bigint	name character varying (40)	image character varying (255)	route character varying (255)	created_at timestamp without time zone	updated_at timestamp without time zone
1	1 CLIENTE	https://firebasestorage.googleapis.com/v0/b/kotindelivery-19d1d.appspot.com/o/Sin%20NC3%ADtulo4.png?alt=media&token=cf3890f204e4f5e97bb9b121bab65cb	cliente/home	2021-12-29 00:00:00	2021-12-29 00:00:00
2	2 COMERCIO	https://firebasestorage.googleapis.com/v0/b/kotindelivery-19d1d.appspot.com/o/Sin%20NC3%ADtulo2.png?alt=media&token=37401ba3-9a51-4082-9e49-ab65e09038cf	administrador/home	2021-12-29 00:00:00	2021-12-29 00:00:00
3	3 REPARTIDOR	https://firebasestorage.googleapis.com/v0/b/kotindelivery-19d1d.appspot.com/o/Sin%20NC3%ADtulo2.png?alt=media&token=11c97cbe-6285-4749-9e48-d5556db6d54c	repartidor/home	2021-12-29 00:00:00	2021-12-29 00:00:00

Esta tabla contiene los nombres de los roles que serán utilizados, contiene una imagen para cada rol y una ruta con la cual podrá utilizar en la aplicación para poder navegar entre actividades.

Estas tablas creadas no tienen ningún tipo de conexión entre sí, por eso, para poder establecer un vínculo, debe crearse la tabla `user_has_roles`:

	id_user [PK] bigint	id_rol [PK] bigint	created_at timestamp without time zone	updated_at timestamp without time zone

La función de esta tabla es otorgar un `id_rol` de la tabla `roles` a un `id_user` de la tabla `users`. Por eso cada tabla posee Primary Keys y Foreign Keys estableciendo un vínculo entre los Id de cada una de ellas. Todas las Keys y las sentencias para crear cada tabla serán detalladas más adelante.

Continuando con la creación de tablas, se procedió con la tabla `categories`:

	id [PK] bigint	name character varying (100)	image character varying (255)	created_at timestamp without time zone	updated_at timestamp without time zone	id_user bigint

En ella podrá observar las columnas, Nombre e Imagen. Estas columnas son necesarias para poder crear las categorías de cada producto y comenzar a filtrarlos. Nuevamente vemos las columnas `created_at` y `updated_at` que cumplen la misma función que en todas las tablas. Por último la columna `id_user`, esta sirve para relacionar dicha categoría con la id de usuario con el motivo de agruparlas por id del creador.

Se procede con la creación de la tabla `products`:

	id [PK] bigint	name character varying (150)	description character varying (255)	stock integer	price numeric	image1 character varying (255)

image2 character varying (255)	image3 character varying (255)	id_category bigint	id_user bigint	created_at timestamp without time zone	updated_at timestamp without time zone
-----------------------------------	-----------------------------------	-----------------------	-------------------	---	---

En esta tabla se agregarán los valores a cada producto, estos son, Nombre, Descripción, Cantidad en Stock, Precio. También podrá agregar tres imágenes desde distintas perspectivas para brindar mayor información. Nuevamente aparecen las columnas `created_at` y `updated_at`. Por último se requiere vincular este producto con el id de su categoría y la id del usuario que lo creó, esto se consigue utilizando los campos `id_category` y `id_user`.

Se crea una tabla llamada `order_has_product`, su función es agrupar todos los productos que el cliente seleccione:

	id_order [PK] bigint	id_product [PK] bigint	quantity bigint	created_at timestamp without time zone	updated_at timestamp without time zone
--	-------------------------	---------------------------	--------------------	---	---

Esta tabla contiene el id de la orden creada, el id de cada producto seleccionado y la cantidad de dicho producto.

Se establece una tabla `orders`, cuya finalidad es darle un seguimiento a la orden de compras:

id [PK] bigint	id_client bigint	id_delivery bigint	id_address bigint	lat numeric	lng numeric	status character varying (90)	timestamp bigint	created_at timestamp without time zone	updated_at timestamp without time zone
-------------------	---------------------	-----------------------	----------------------	----------------	----------------	----------------------------------	---------------------	---	---

Esta tabla está vinculada al Id del cliente que realizó el pedido, contiene el Id de la persona encargada a entregarlo, contiene además las columnas `lat` y `lng` (latitud y longitud) necesaria para obtener la ubicación del pedido en tiempo real. También se observa la columna “status”, en ella podrá ver el estado del pedido



(Pagado, Despachado, En camino o Entregado). La columna timestamp es utilizada para guardar el tiempo en el que la orden cambia de estado.

Se procede a crear la tabla address:

id	id_user	address	neighborhood	lat	lng	created_at	updated_at
[PK] bigint	bigint	character varying (255)	character varying (255)	numeric	numeric	timestamp without time zone	timestamp without time zone

Esta tabla contiene todos los datos pertenecientes a la ubicación del cliente que realiza la compra. Dirección (Calle y Altura), Barrio. Puede observar nuevamente las columnas lat, lng, created_at, updated_at.

De esta manera se da por finalizado de forma parcial la creación de las tablas. En el [Anexo A](#) podrá ver las sentencias utilizadas para establecer todas las Keys necesarias.

Token de Sesión: Se utiliza una función nativa del modelo JWT en VSC, con esto cada usuario obtiene una frase encriptada, esta frase llamada header otorga mayor seguridad al momento de realizar una petición. En simples palabras, si el usuario no posee este token, no podrá hacer ningún tipo de consulta. Este token se crea automáticamente al momento de registrarse.

De esta manera se evita que usuarios ajenos a la aplicación pueda obtener acceso.

Token de Notificación: Se utiliza un servicio de Google llamado Push Notification Token, importando las librerías correspondientes se crea una función en la cual se asigna un token de notificación al momento de realizar navegar en la pantalla principal, de este modo podemos enviar notificaciones entre distintos dispositivos. Como punto negativo de esta función, sólo funciona en dispositivos que posean mínimamente la versión de Android 8.0.

En principio se debe conectar el gestor PostgreSQL a VSC de la siguiente manera:

Conexión:

```
1  const promise = require ('bluebird');
2  const options = {
3    |   promiselib: promise,
4    |   query: (e) => {}
5  }
6
7  const pgp = require('pg-promise')(options);
8  const types = pgp.pg.types;
9
10 types.setTypeParser(1114, function(stringValue){
11 |   return stringValue;
12 | });
13
14
15 const databaseConfig = {
16 |   'host' : '127.0.0.1',
17 |   'port' : 5432,
18 |   'database' : 'delivery_bd',
19 |   'user' : 'postgres',
20 |   'password' : 'Jakinoto10'
21 |
22 };
23
24 const db = pgp(databaseConfig);
25
26 module.exports = db;
27
28
```

Básicamente se configura una constante db de acuerdo a la documentación de VCS y PostgreSQL.

La sentencia “module.exports = db” es fundamental, con ella se establece la conexión a cada archivo creado. Pero esto no es suficiente para terminar el vínculo, debe agregarse un archivo “server.js”, el cual funciona como servidor local y nos brinda una respuesta HTTP, debe estar configurado de la siguiente manera:

```
server.listen(3000, '192.168.0.100' || 'localhost', function(){
  console.log('Aplicacion de NodeJS ' + port + ' Iniciada...')
});

// ERROR HANDLER

app.use ((err, req, res, next) => {
  console.log(err);
  res.status(err.status || 500).send (err.stack);
});

module.exports = {
  app: app,
  server: server
}

// 201 respuesta exitosa
// 404 significa que hubo error porque no existe la url
// 400 error interno del servidor
// 402 restricción
// 501 no hay respuesta del servidor
```

De esta manera se crea un servidor local, en el cual puede obtener respuestas al momento de realizar una consulta, de este modo podemos encontrar errores de forma directa.

Funcionamiento de la metodología MVC aplicada al proyecto.

Se crea un modelo SQL, dentro de una función JS.

```
User.create = async (user) => {  
  const hash = await bcrypt.hash(user.password, 10);  
  
  const sql = `  
    INSERT INTO  
      users(  
        email,  
        name,  
        lastname,  
        phone,  
        image,  
        password,  
        created_at,  
        updated_at  
      )  
    VALUES($1, $2, $3, $4, $5, $6, $7, $8) RETURNING id  
  `;  
  
  return db.oneOrNone(sql, [  
    user.email,  
    user.name,  
    user.lastname,  
    user.phone,  
    user.image,  
    hash,  
    new Date(),  
    new Date()  
  ]);  
}
```

Mediante la constante sql, se implementa una QUERY de tipo CREATE, especificando en que tabla se insertarán los datos, y los valores de cada uno. Se precisa que esta función retorne una Id en caso de que la consulta sea exitosa, junto con esta id se obtienen los datos que el usuario ingresa en la pantalla de registro.

Para darle valores a esos parámetros, se utiliza el controlador, este es el que recibe la información y actualiza los campos en la base de datos mediante una función asíncrona.

```
async register(req, res, next) {
  try {
    const user = req.body;
    const data = await User.create(user);

    await Rol.create(data.id, 1);

    const token = jwt.sign({ id: data.id, email: user.email }, keys.secretOrKey, {
      // expiresIN:
    })

    const myData = {
      id: data.id,
      name: user.name,
      lastname: user.lastname,
      email: user.email,
      phone: user.phone,
      image: user.image,
      session_token: `JWT ${token}`
    };

    return res.status(201).json({
      success: true,
      message: 'El registro se realizo correctamente',
      data: myData
    });

  } catch (error) {
    console.log(`Error: ${error}`);
    return res.status(501).json({
      success: false,
      message: 'Hubo un error con el registro de usuario',
      error: error
    });
  }
},
```

Esta función requiere de un body o cuerpo, básicamente todos los datos que usuario ingrese serán parte de el body que estamos pidiendo. Al mismo tiempo se asigna un rol por defecto con el valor “1”, si observa la tabla roles verá que el tipo de rol es “CLIENTE”. El objeto myData es utilizado para asignar todos los valores que el cliente ingresó. Esta función retorna un status 201 en caso de ser exitoso o un 501 en caso de ocurrir algún error.

Para que el cliente pueda enviar la información y que ésta sea procesada en la función del controlador, se necesita una ruta. Por eso en esta metodología también es necesario crear un archivo que contenga todas las rutas necesarias.

En este caso es utilizada una ruta de tipo POST, y debe vincularse a la función register del controlador UsersController de la siguiente manera.

```
module.exports = (app, upload) => {

  //TRAER DATOS
  app.get('/api/users/getAll', UsersController.getAll);
  app.get('/api/users/getClientCommerce', UsersController.getClientCommerce);
  app.get('/api/users/findDeliveryMan', UsersController.findDeliveryMan);

  // GUARDAR DATOS
  app.post('/api/users/create', UsersController.register);
  app.post('/api/users/login', UsersController.login);

  // ACTUALIZAR DATOS
  //401 no esta autorizada
  app.put('/api/users/update', passport.authenticate('jwt',{session: false}), upload.array('image', 1), UsersController.update);
  app.put('/api/users/updateWithoutImage', passport.authenticate('jwt',{session: false}), UsersController.updateWithoutImage);
  app.put('/api/users/updateNotificationToken', passport.authenticate('jwt',{session: false}), UsersController.updateNotificationToken);

  /*
  * PUT ROUTES
  */
}
```

Observe que la ruta utilizada en este caso es la siguiente:

```
app.post('/api/users/create', UsersController.register);
```

Una vez creado el modelo, el controlador y la ruta, se necesita crear el vínculo entre esta ruta y la aplicación, se logra exactamente de la misma manera. Pero con documentación Kotlin.

```
@POST ( value: "users/create")
fun register (@Body user: User ) : Call<ResponseHttp>
```

Ahora debe crearse el backend de la aplicación, pero esta vez desde el IDE, en este caso utilizamos Android Studio.

Puede observar que estamos llamando a la función register y que se requiere de un Body “user”, este debe ser de tipo User (modelo).

En este modelo User, se especifican todos los campos creados de la tabla users, al mismo tiempo se asigna a cada uno, una variable para poder trabajar con mayor comodidad.

```
class User(  
    @SerializedName(value: "id") val id: String? = null,  
    @SerializedName(value: "name") var name: String,  
    @SerializedName(value: "lastname") var lastname: String,  
    @SerializedName(value: "email") val email: String,  
    @SerializedName(value: "phone") var phone: String,  
    @SerializedName(value: "password") val password: String,  
    @SerializedName(value: "image") val image: String? = null,  
    @SerializedName(value: "session_token") val sessionToken: String? = null,  
    @SerializedName(value: "notification_token") var notificationToken: String? = null,  
    @SerializedName(value: "is_available") val isAvailable: Boolean? = null,  
    @SerializedName(value: "roles") val roles: ArrayList<Rol>? = null,  
) {  
    override fun toString(): String {  
        return "$name $lastname"  
    }  
  
    fun toJson(): String {  
        return Gson().toJson(src: this)  
    }  
}
```

También debe crearse un controlador que contenga a la función “register” y que será utilizada a través de la ruta:

```
fun register(user: User): Call <ResponseHttp>? {  
    return usersRoutes?.register(user)  
}
```

De esta manera se construye el Body que requerimos en la función asíncrona del backend en VSC. Ahora lo que falta es pedirle al usuario que ingrese esos campos en orden.

Vista de la pantalla de registro:

The screenshot shows a mobile application interface for a registration screen. At the top, the title 'Registro' is displayed in a stylized font, followed by the instruction 'Completa estos datos'. Below this, there are six input fields stacked vertically, each with a light gray background and a thin border. The labels for these fields are 'Nombre', 'Apellido', 'Teléfono', 'Email', 'Contraseña', and 'Confirmar contraseña'. At the bottom of the form, there is a large, rounded blue button with the text 'REGISTRARSE' in white capital letters. Below the button, there is a blue left-pointing arrow icon, and underneath that, the text 'Volver al inicio' in a small, gray font. The entire form is enclosed in a white container with rounded corners, which is set against a dark gray background.

Para darle utilidad a esta pantalla, debe agregarse a la actividad correspondiente la siguiente función:

```
private fun registro(){
    val name = editTextName?.text.toString()
    val lastname = editTextApellido?.text.toString()
    val email = editTextEmail?.text.toString()
    val phone = editTextTelefono?.text.toString()
    val password = editTextContraseña?.text.toString()
    val confirmpassword = editTextConfirmarContraseña?.text.toString()

    var usersProvider = UsersProvider()

    if (isValidForm(name = name, lastname =lastname, email=email, phone =phone,password=password, confirmpassword= confirmpassword)){

        val user = User(
            name = name,
            lastname = lastname,
            email = email,
            phone = phone,
            password = password
        )

        usersProvider.register(user)?.enqueue(object: Callback <ResponseHttp>{
            override fun onResponse(call: Call<ResponseHttp>, response: Response<ResponseHttp>) {

                if (response.body()?.isSuccess == true) {
                    saveUserInSession(response.body()?.data.toString())
                    goToClientHome()
                }

                Toast.makeText( context: this@RegisterActivity, response.body()?.message,Toast.LENGTH_LONG).show()

                Log.d(TAG, msg: "Response: ${response}")
                Log.d(TAG, msg: "Body: ${response.body()}")

            }

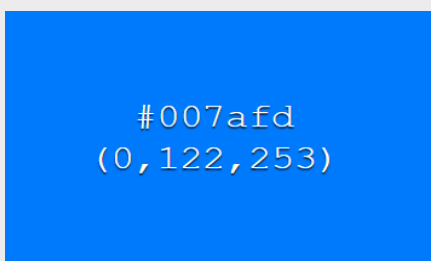
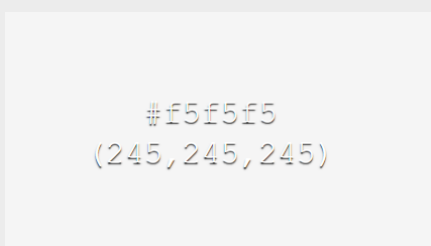
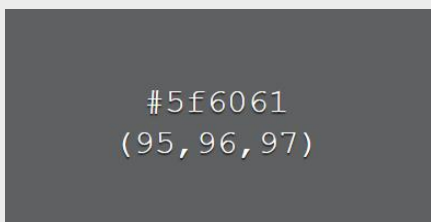
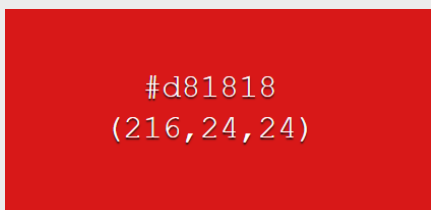
            override fun onFailure(call: Call<ResponseHttp>, t: Throwable) {
                Log.d(TAG, msg: "Se produjo un error ${t.message}")
                Toast.makeText( context: this@RegisterActivity, text: "Se produjo un error",Toast.LENGTH_LONG).show()
            }

        })
    }
}
```

Con esta función se crea un objeto user de tipo User, el mismo objeto que se requiere en la ruta que llama a la función register. De esta manera finaliza la petición, y se obtiene de ella un mensaje exitoso o un error, ya sea por una falla de conexión o porque el usuario ingresó mal un dato.

De la misma forma aplicamos esta metodología para las sentencias UPDATE, DELETE y READ.

Paleta de colores utilizados:

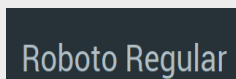


Tipografía:

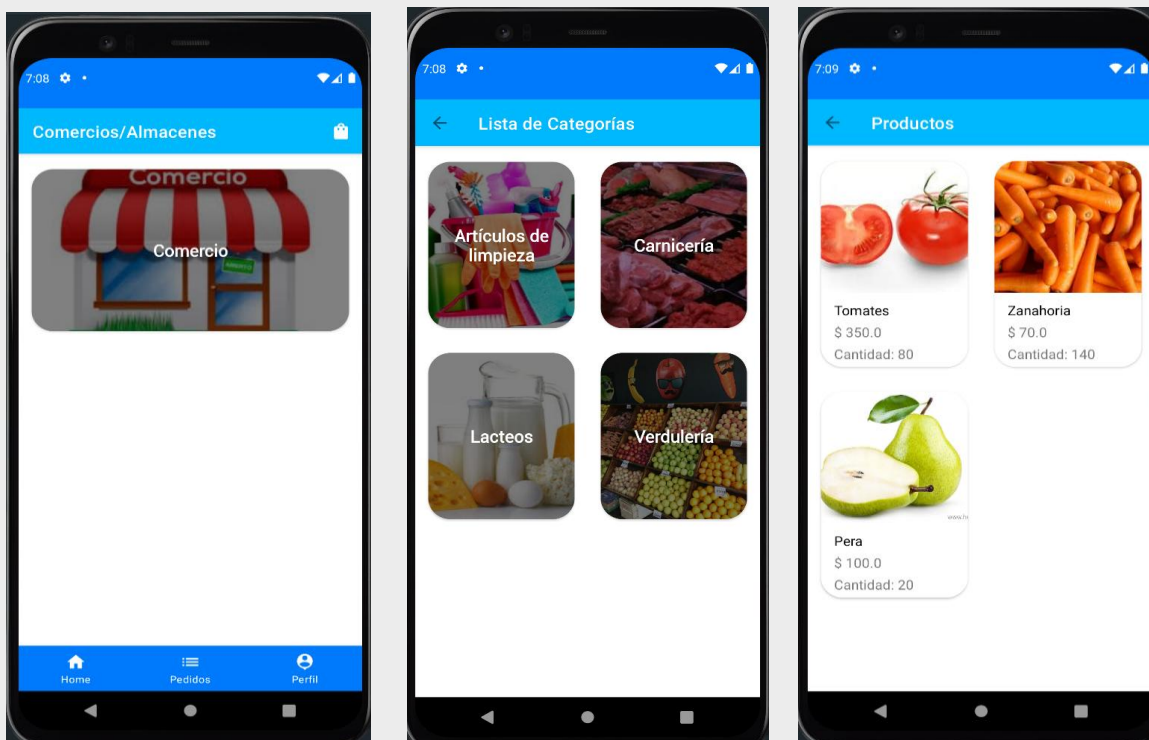
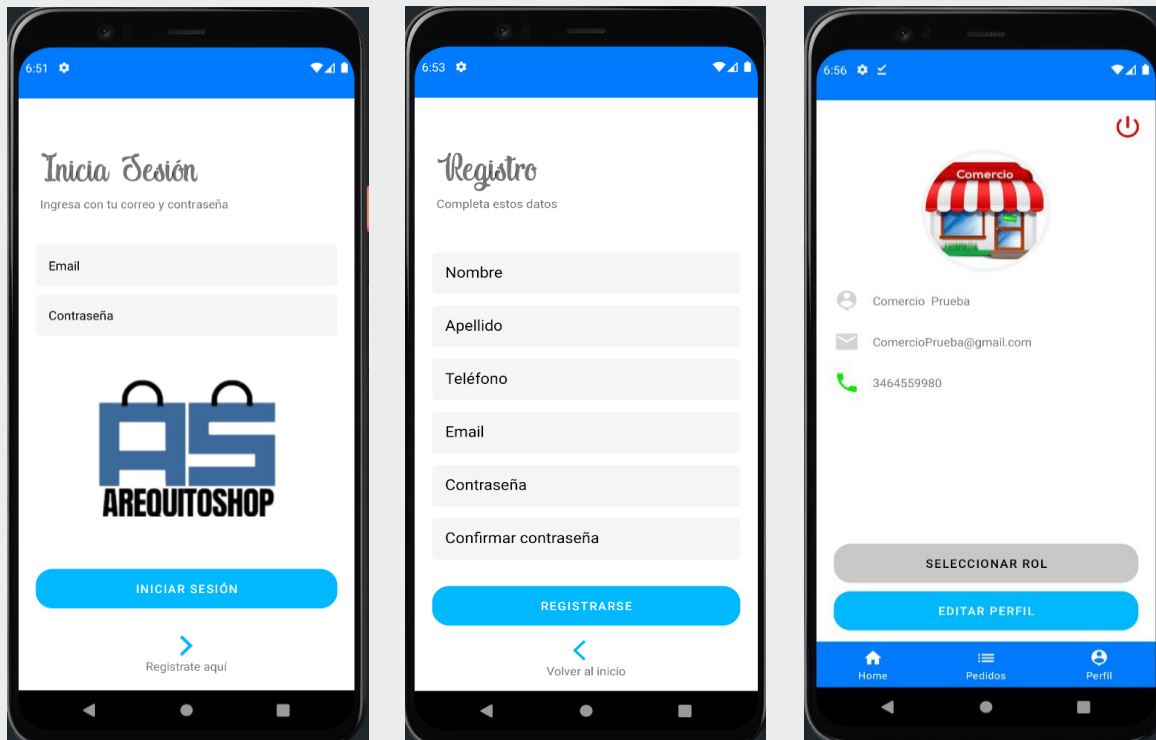
Indigo Daisy

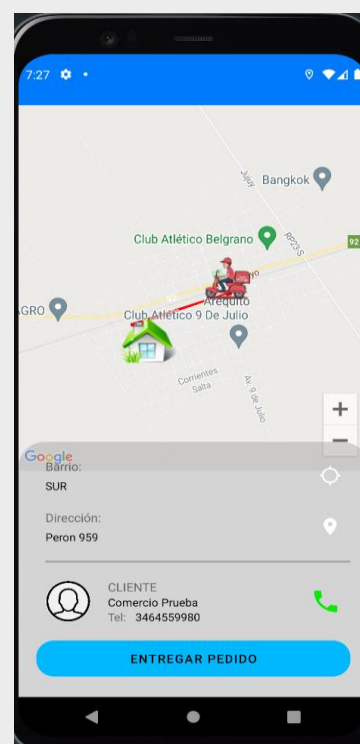
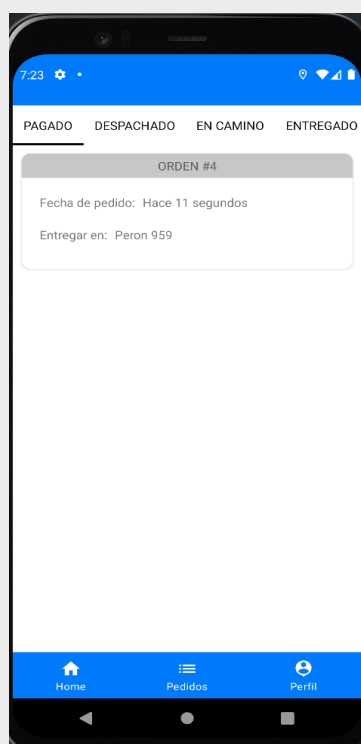
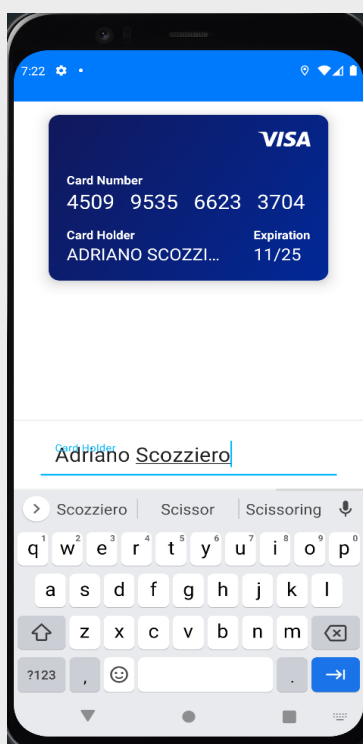
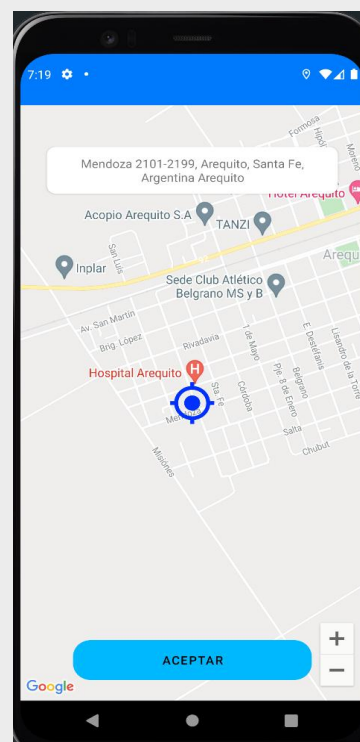
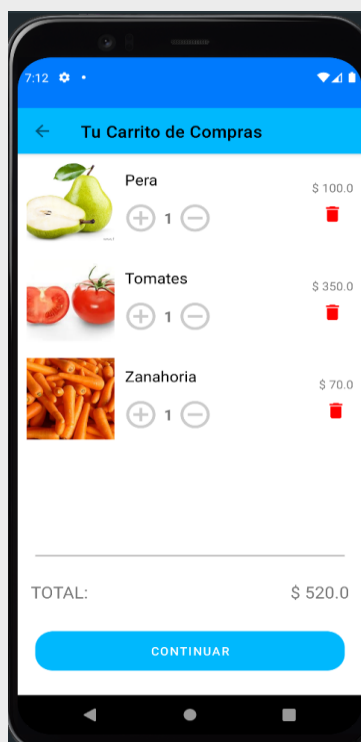


Roboto Angular

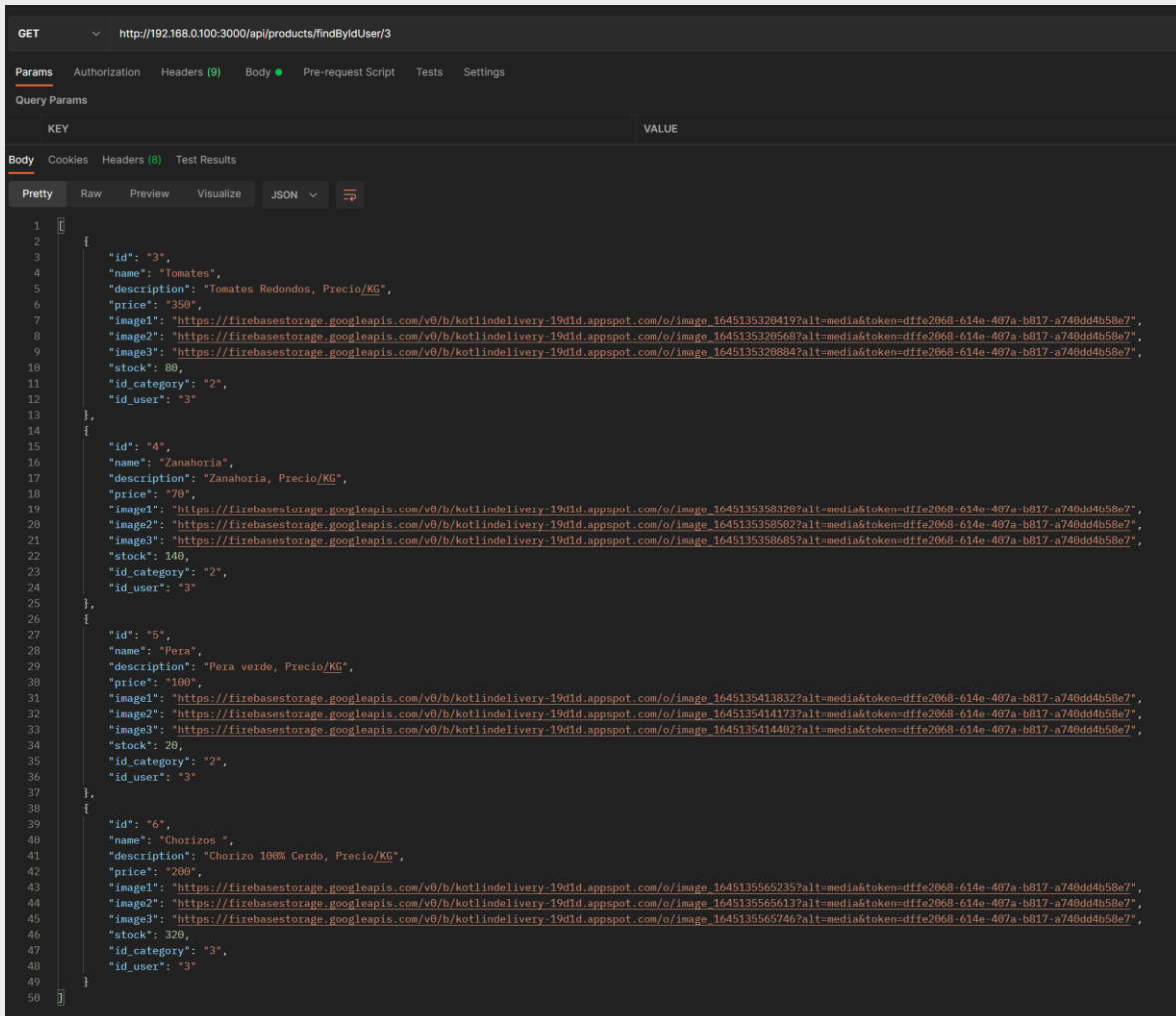


Diseño de la interfaz:





Al momento de programar tanto el backend como el frontend, se utilizó el programa Postman, con esta aplicación se enviaron peticiones mediante las rutas, de esta manera pudo detectarse si la información se transmitía correctamente o si existió un error, dónde estaba dicho error y poder solucionarlo con mayor rapidez.



```

GET http://192.168.0.100:3000/api/products/findByIdUser/3

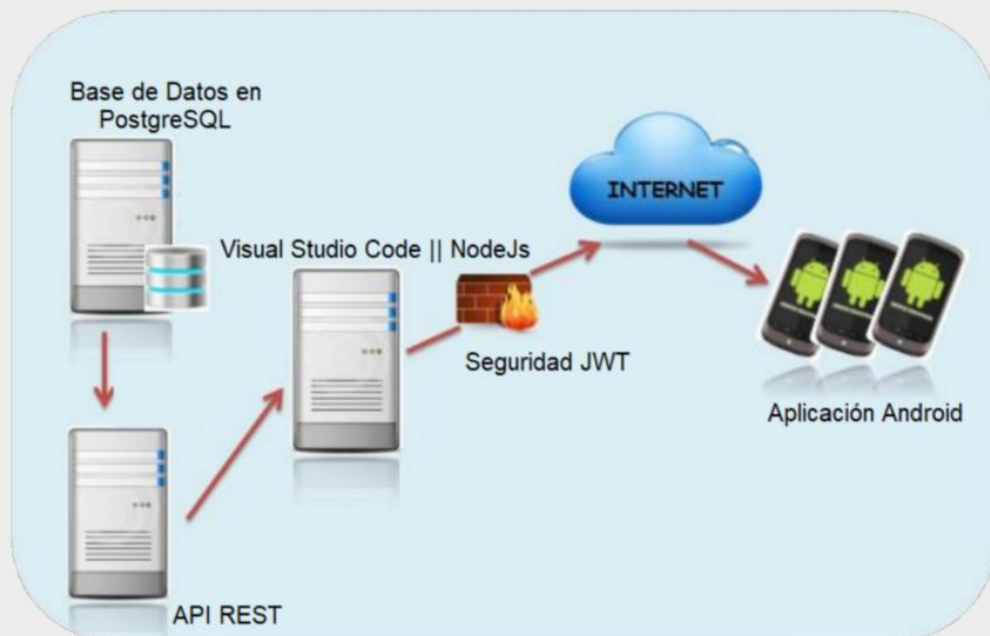
Params Authorization Headers (9) Body ● Pre-request Script Tests Settings
Query Params
KEY VALUE
Body Cookies Headers (8) Test Results
Pretty Raw Preview Visualize JSON
1
2
3 {
4   "id": "3",
5   "name": "Tomates",
6   "description": "Tomates Redondos, Precio/KG",
7   "price": "350",
8   "image1": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135320419?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
9   "image2": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135320568?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
10  "image3": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135320884?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
11  "stock": 80,
12  "id_category": "2",
13  "id_user": "3"
14 },
15 {
16   "id": "4",
17   "name": "Zanahoria",
18   "description": "Zanahoria, Precio/KG",
19   "price": "70",
20   "image1": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135358320?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
21   "image2": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135358582?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
22   "image3": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135358685?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
23   "stock": 140,
24   "id_category": "2",
25   "id_user": "3"
26 },
27 {
28   "id": "5",
29   "name": "Pera",
30   "description": "Pera verde, Precio/KG",
31   "price": "100",
32   "image1": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135413832?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
33   "image2": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135414173?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
34   "image3": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135414402?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
35   "stock": 20,
36   "id_category": "2",
37   "id_user": "3"
38 },
39 {
40   "id": "6",
41   "name": "Chorizos ",
42   "description": "Chorizo 100% Cerdo, Precio/KG",
43   "price": "200",
44   "image1": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135565235?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
45   "image2": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135565613?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
46   "image3": "https://firebasestorage.googleapis.com/v0/b/kotlindelivery-19d1d.appspot.com/o/image_1645135565746?alt=media&token=affe2068-614e-407a-b817-a740dd4b58e7",
47   "stock": 320,
48   "id_category": "3",
49   "id_user": "3"
50 }

```

Una vez finalizado el desarrollo, el proyecto entró en fase de testeo, donde se realizaron significativas y extenuantes pruebas con la finalidad de encontrar y resolver todos los problemas posibles.

Como cierre parcial de este proyecto, la BDD y la API REST fueron lanzadas a un servidor de prueba en Heroku. Esto se ha desarrollado basándonos en la documentación aportada por la página web para desarrolladores en fase de prueba. De esta manera podemos conectar nuestra aplicación de forma remota y local.

Nos enfocamos en crear una Aplicación atractiva a la vista y lo como punto central que sea lo más intuitiva posible.



Diseño del Sistema.

De manera sintética, nuestra aplicación funciona mediante peticiones que envía un usuario que tenga una conexión a internet y nuestra aplicación instalada. Con la seguridad JWT, garantizamos que toda su información está protegida ante cualquier agente ajeno al sistema. Estas peticiones ingresan a un controlador, este verificará que sean correctas y generará una vista, al mismo tiempo generará un CRUD de nuestra base de datos, siempre y cuando los datos y peticiones ingresadas sean válidos.

PLAN DE MANTENIMIENTO

Ya que existe una constante evolución de la tecnología, es indispensable construir un plan de mantenimiento eficiente, esto debe realizarse debido a posibles actualizaciones de los sistemas operativos (Android en este caso). ArequitoShop debe adaptarse a estas actualizaciones, también deberá ajustarse a la creación de dispositivos nuevos, con nuevas resoluciones de pantalla. Por este motivo, se contratará un servicio externo que nos asesore de posibles cambios y estar listos para actualizar nuestra aplicación en tiempo record. Planificamos someter a mantenimiento general a finales de cada mes. Éste constará de:

- Corrección de errores (bugs) que aparezcan después de un cambio en el Sistema Operativo.
- Adaptar la aplicación a las guías de estilos de los nuevos sistemas.
- Ajustar la aplicación a nuevos dispositivos que aparezcan en el mercado.



De esta manera garantizaremos una evolución adecuada de la aplicación para mantener la satisfacción en la experiencia del usuario y seguir aumentando el número de descargas y el uso de nuestra aplicación.

También estaremos observando las críticas expresadas por nuestros clientes y usuarios, esto también nos brindará información necesaria para seguir mejorando nuestra aplicación.

Para la parte de servidores y base de datos, estaremos observando la cantidad de usuarios que utilicen nuestra app, con esto podemos cambiar de plan adquirido en Heroku. Esto expandirá el tamaño de la base de datos, aumentará la velocidad de conexión y ampliará la cantidad de consultas realizadas al mismo tiempo. Todo automáticamente eligiendo un plan mensual adquirido basándonos en la cantidad de usuarios.

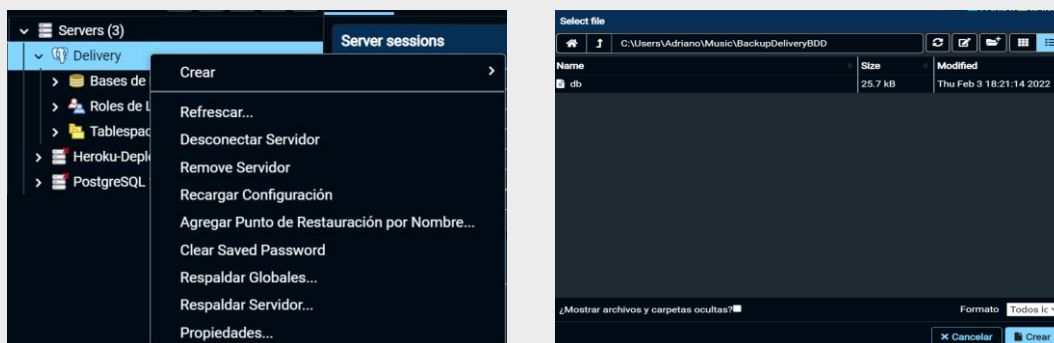
Como instancia final, se realizará un análisis de todos los registros de nuestra base de datos, eliminaremos todos los registros que no hayan sido utilizados en más de tres meses. De esta manera aumentamos la eficiencia y eficacia de nuestro servidor. También reorganizaremos y generaremos nuevas estadísticas de nuestra base de datos y eliminaremos tablas temporales. Realizamos un Backup de todo el proyecto, aprovechando las herramientas aportadas por cada programa utilizado para la construcción de la aplicación.

Visual Studio Code:

	BackendNodeJsArequitoShop	24/1/2022 18:30	Carpeta de archivos	
	BackendNodeJsArequitoShop	8/2/2022 00:08	Archivo WinRAR	15.751 KB

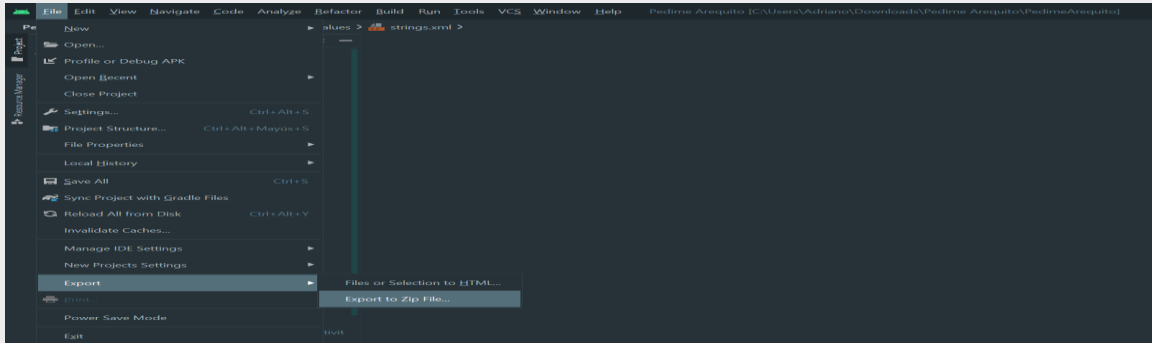
Creamos un Backup en formato WinRAR y lo subimos a la nube de Google-Storage. De esta manera garantizamos que nuestra API REST no se pierda o se dañe en caso de una falla de nuestro ordenador, igualmente se encuentra lanzada en los servidores de Heroku.

PostgreSQL:



Presionando clic derecho al servidor creado en este gestor, podemos hacer un respaldo del servidor y de la base de datos, le asignamos un nombre y seleccionamos la carpeta de destino.

Android Studio:



De la misma forma Android posee una herramienta para realizar un Backup al proyecto. De esta forma respaldamos todos los archivos en un servidor remoto.

EL PROYECTO A FUTURO

Lo primero es la mejora de la interfaz gráfica, trabajaremos constantemente para que la aplicación tenga un acabado visual más profesional.

Una vez realizados los estudios pertinentes de todos los usuarios que utilicen nuestra aplicación, pondremos en marcha la codificación para sitios web y sistema IOS. De esta manera ampliaremos aún más nuestro servicio.

Si el proyecto marcha bien, comenzaremos a expandir nuestro servicio a las demás localidades donde no llegan los servicios que ofrecen otras apps conocidas.

De acuerdo a lo recaudado, compraremos lo necesario para poder tener un servidor propio, de esta manera podremos mejorar aún más nuestro servicio. En este servidor también podremos alojar futuros proyectos propios y ajenos.

Uno de los puntos más importantes que tiene el proyecto y no fue mencionado anteriormente, es la creación de nuevos puestos de trabajo. Estamos hablando de usuarios que dispongan de un medio de transporte y quieran obtener un ingreso fijo a través de horarios flexibles. Nuestra intención es dejar a cargo del proyecto a la comuna del pueblo. De esta manera podrán asignar a un grupo de personas que sean competentes con el trabajo de repartidor.

Realizaremos un estudio exhaustivo e implementaremos nuevas metodologías con el fin de enfocar el perfil de nuestro proyecto hacia lo que es una empresa tipo B. Nuestro objetivo principal a futuro será promover un cambio positivo a nivel social, económico y medioambiental, sin dejar de generar crecimiento, las utilidades y empleo.

CONCLUSIÓN:

Se han implementado todos los procedimientos establecidos en la metodología ágil SCRUM.

Dado el lenguaje SQL aprendido en las cátedras Bases de Datos, Gestión de Software I y II para la creación y administración de una base de datos, se ha creado con éxito una base de datos cuya escalabilidad y durabilidad es óptima y se adapta correctamente a la aplicación desarrollada.

Todos los problemas que han surgido mediante la etapa de desarrollo del backend se resolvieron con éxito, cabe aclarar que ocurrieron problemas donde recurrimos a la búsqueda en internet consultando librerías necesarias para su resolución.

Por otra parte, adquirimos y refrescamos conocimientos sobre SQL, Java Script y Kotlin.

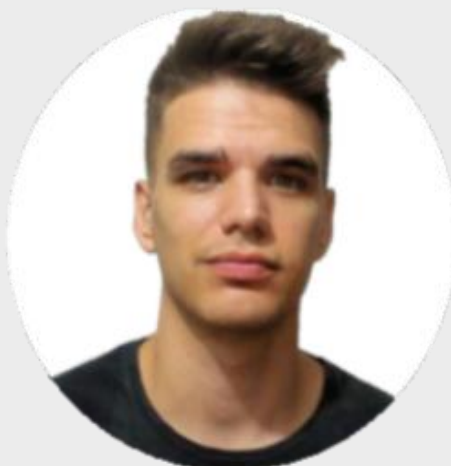
Se han cumplido todos los objetivos y requisitos mencionados. El principal de ellos fue crear una aplicación móvil de principio a fin para Android capaz de impulsar el e-commerce de nuestra localidad de manera profesional.

Se han implementado conceptos aprendidos en las cátedras Modelos de Negocios y Estrategias de Negocios para la creación de un Plan de Negocios que será presentado en busca de inversores para el lanzamiento oficial de la aplicación.

Una vez desarrollada totalmente la aplicación y comenzado la etapa de pruebas, se implementaron funciones adicionales al diseño original, ya que se plantearon posibles mejoras visuales que favorecían la usabilidad general de la aplicación.

Agradecemos a todo el personal del ISP N°62 “Ángela Cullen” por su hospitalidad, preocupación y por su labor ejercido durante nuestros años de cursado, con su apoyo logramos desarrollarnos como profesionales en el ámbito y con los conocimientos y valores adquiridos durante esta instancia, se abrieron muchas puertas en el ambiente laboral.

AUTORES



Scozziero Stefano Lucio



scozzierostefano@gmail.com



<https://www.instagram.com/stefanoscozziero/>



<https://www.facebook.com/stefano.scozziero.1/>



<https://www.linkedin.com/in/stefano-scozziero/>



<https://twitter.com/StefanoScoo>



3464 - 542887



Scozziero Adriano Alejo



scozzieroadriano@gmail.com



https://www.instagram.com/adriano_scozziero/



<https://m.facebook.com/AdrianoScozziero>



<https://www.linkedin.com/in/adriano-scozziero/>



<https://twitter.com/adrianoscozz>



3464 – 559980

ANEXO A

Querys utilizadas para la creación de tablas y establecer las llaves primarias y foráneas:

```
DROP TABLE IF EXISTS roles CASCADE;

CREATE TABLE roles (
  id BIGSERIAL PRIMARY KEY,
  name VARCHAR(40) NOT NULL UNIQUE,
  image VARCHAR(255) NULL,
  route VARCHAR(255) NULL,
  created_at TIMESTAMPTZ(0) NOT NULL,
  updated_at TIMESTAMPTZ(0) NOT NULL
);

DROP TABLE IF EXISTS users CASCADE;

CREATE TABLE users(
  id BIGSERIAL PRIMARY KEY,
  email VARCHAR(80) NOT NULL UNIQUE,
  name VARCHAR (80) NOT NULL,
  lastname VARCHAR (80) NOT NULL,
  phone VARCHAR (30) NOT NULL UNIQUE,
  image VARCHAR (255) NULL,
  password VARCHAR(80) NOT NULL,
  is_available BOOLEAN NULL,
  session_token VARCHAR(255) NULL,
  notification_token VARCHAR(255) NULL,
  created_at TIMESTAMPTZ(0) NOT NULL,
  updated_at TIMESTAMPTZ(0) NOT NULL
);

DROP TABLE IF EXISTS user_has_roles CASCADE;

CREATE TABLE user_has_roles (
  id_user BIGSERIAL NOT NULL,
  id_rol BIGSERIAL NOT NULL,
  created_at TIMESTAMPTZ(0) NOT NULL,
  updated_at TIMESTAMPTZ(0) NOT NULL,
  FOREIGN KEY(id_user) REFERENCES users(id) ON UPDATE CASCADE ON DELETE CASCADE,
  FOREIGN KEY(id_rol) REFERENCES roles(id) ON UPDATE CASCADE ON DELETE CASCADE,
  PRIMARY KEY(id_user, id_rol)
);

INSERT INTO roles(
  name,
  route,
  image,
  created_at,
  updated_at
)
VALUES(
  'CLIENTE',
  'cliente/home',
  'https://www.pclipart.com/picdir/middle/165-1653686_female-user-icon-png-download-user-colorful-icon.png',
  '2021-12-29',
  '2021-12-29'
);

INSERT INTO roles(
  name,
  route,
  image,
  created_at,
  updated_at
)
VALUES(
  'COMERCIO',
  'administrador/home',
  'https://static.vecteezy.com/system/resources/thumbnails/000/439/863/small/Basic_Ui__28186_29.jpg',
  '2021-12-29',
  '2021-12-29'
);

INSERT INTO roles(
  name,
  route,
```

```

        image,
        created_at,
        updated_at
    )
VALUES(
    'VENDEDOR',
    'vendedor/home',
    'https://i.pinimg.com/474x/d4/38/d9/d438d910c0f36a247db720a16c85873b.jpg',
    '2021-12-29',
    '2021-12-29'
);
);
DROP TABLE IF EXISTS categories CASCADE;

CREATE TABLE categories (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR (100) NOT NULL UNIQUE,
    image VARCHAR (255) NOT NULL,
    created_at TIMESTAMP (0) NOT NULL,
    updated_at TIMESTAMP (0) NOT NULL,
    id_user BIGINT,
    FOREIGN KEY(id_user) REFERENCES users (id) ON UPDATE CASCADE ON DELETE CASCADE
);
DROP TABLE IF EXISTS products CASCADE;
CREATE TABLE products(
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR (150) NOT NULL UNIQUE,
    description VARCHAR (255) NOT NULL,
    stock INT DEFAULT 0,
    price DECIMAL DEFAULT 0,
    image1 VARCHAR (255) NOT NULL,
    image2 VARCHAR (255) NULL,
    image3 VARCHAR (255) NULL,
    id_category BIGINT NOT NULL,
    id_user BIGINT,
    created_at TIMESTAMP (0) NOT NULL,
    updated_at TIMESTAMP (0) NOT NULL,
    FOREIGN KEY(id_category) REFERENCES categories (id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(id_user) REFERENCES users (id) ON UPDATE CASCADE ON DELETE CASCADE
);
);
DROP TABLE IF EXISTS address CASCADE;
CREATE TABLE address(
    id BIGSERIAL PRIMARY KEY,
    id_user BIGINT NOT NULL,
    address VARCHAR (255) NOT NULL,
    neighborhood VARCHAR (255) NOT NULL,
    lat DECIMAL DEFAULT 0,
    lng DECIMAL DEFAULT 0,
    created_at TIMESTAMP (0) NOT NULL,
    updated_at TIMESTAMP (0) NOT NULL,
    FOREIGN KEY(id_user) REFERENCES users(id) ON UPDATE CASCADE ON DELETE CASCADE
);
);
DROP TABLE IF EXISTS orders CASCADE;
CREATE TABLE orders(
    id BIGSERIAL PRIMARY KEY,
    id_client BIGINT NOT NULL,
    id_delivery BIGINT NULL,
    id_address BIGINT NOT NULL,
    lat DECIMAL DEFAULT 0,
    lng DECIMAL DEFAULT 0,
    status VARCHAR(90) NOT NULL,
    timestamp BIGINT NOT NULL,
    created_at TIMESTAMP (0) NOT NULL,
    updated_at TIMESTAMP (0) NOT NULL,
    FOREIGN KEY(id_client) REFERENCES users(id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(id_delivery) REFERENCES users(id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(id_address) REFERENCES address(id) ON UPDATE CASCADE ON DELETE CASCADE
);
);
DROP TABLE IF EXISTS order_has_products CASCADE;
CREATE TABLE order_has_products(
    id_order BIGINT NOT NULL,
    id_product BIGINT NOT NULL,
    quantity BIGINT NOT NULL,
    created_at TIMESTAMP (0) NOT NULL,
    updated_at TIMESTAMP (0) NOT NULL,
    PRIMARY KEY(id_order,id_product),
    FOREIGN KEY(id_order) REFERENCES orders(id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(id_product) REFERENCES products(id) ON UPDATE CASCADE ON DELETE CASCADE
);
);

```

La sentencia `DROP TABLE IF EXISTS` acompañado por el nombre de la tabla y la palabra `CASCADE`, nos asegura que si la tabla existe sea eliminada, y con ello elimine de manera “en cascada”, todos los campos de ésta tabla y todos los campos de otra tabla en la que estén vinculados por una llave, ya sea primaria o foránea.

Luego de establecer cada campo de una tabla, utilizamos la sentencia `FOREIGN KEY` o `PRIMARY KEY` y establecemos su referencia con la ID que corresponda, de esta manera se crea un vínculo entre dichas tablas.

Referencias:

Bibliografía:

- **“El gran libro de Android”** 8° edición – Jesús Tomás y Beatriz Tirado
- **“Aprende a programar con Kotlin”** – José Dimas Luján Castillo
- Material aportado por los docentes en el campus virtual.

Sitios Web:

- <https://creatuaplicacion.com/importante-mantenimiento-aplicacion-movil/>
- <https://www.mercadopago.com.ar/developers/es/guides>
- <https://firebase.google.com/docs>
- <https://github.com/mercadopago/px-android>
- <https://devcenter.heroku.com/articles/git>
- <https://rockcontent.com/es/blog/api-rest/>
- <https://developer.android.com/docs>

Videos:

- ¿Qué es la metodología SCRUM? - Glosario Mobile & App Marketing - YouTube
- Como Hacer Un Pitch - YouTube