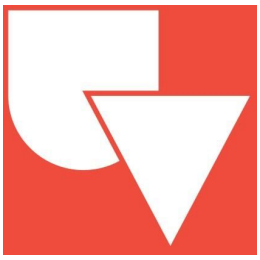

UNIVERSIDAD DEL VALLE

MANUAL TECNICO
PROYECTO FINAL

BASES DE DATOS

INTEGRANTES:

CARLOS ESTEBAN MURILLO
ANDRÉS MAURICIO MONTENEGRO
CHRISTIAN DAVID MILLAN

PRESENTADO A:
CASTILLO ROBLES ANDRÉS MAURICIO

Santiago de Cali, Marzo de 2019
Escuela de Ingeniería en Sistemas
Facultad de Ingeniería

MANUAL TECNICO
CONTENIDO

1. PASOS PARA INSTALACIÓN DE APP

Link para la descarga del docker compose
https://github.com/carban/taxiapi

Para correr las aplicaciones desde docker, usando docker compose:
descargar docker-compose.yml
ubicarse en la carpeta donde se descargo y escribir en la terminal:

docker-compose up -d

a continuación comenzará a bajar las imágenes de las aplicaciones del docker hub (esto puede tardar un poco). Al finalizar mostrará algo como esto:

```
root@andresma-HP-Laptop-14-bp0xx:/home/andresma/Escritorio/Doke# docker-compose up -d
Creating doke_api_1 ... done
Creating doke_cliente_1    ... done
Creating doke_conductor_1 ... done
```

Sin embargo la "construcción" de las imágenes aún no ha finalizado, con el comando

docker-compose logs

se puede ver lo que imprimen cada imagen mientras se monta el contenedor

```
root@andresma-HP-Laptop-14-bp0xx:/home/andresma/Escritorio/Doke# docker-compose logs
Attaching to doke_conductor_1, doke_cliente_1, doke_api_1
cliente_1    | Starting up http-server, serving dist
cliente_1    | Available on:
cliente_1    |   http://127.0.0.1:8080
cliente_1    |   http://172.19.0.3:8080
cliente_1    | Hit CTRL-C to stop the server
api_1        |  * Starting PostgreSQL 10 database server
conductor_1  | Starting up http-server, serving dist
conductor_1  | Available on:
conductor_1  |   http://127.0.0.1: 8081
conductor_1  |   http://172.19.0.4: 8081
conductor_1  | Hit CTRL-C to stop the server
```

Notará que los contenedores de las aplicaciones del cliente y conductor son los primeros en crear dado que su construcción es rápida y por eso puede verlas escribiendo en su navegador (recomendamos usar firefox) lo siguiente:

localhost:8080 para el cliente
localhost:8081 para el conductor

Pero como el contenedor de la api es un poco más lento en terminar de hacerse, tendrá que esperar a que finalice su elaboración para poder usar las aplicaciones en conjunto con la api y la base de datos.

Aquí se puede ver como el contenedor de la api terminó y está a la escucha en el puerto 8000, lo que significa que todo funcionara bien

```
cliente_1    | [Sun Apr 14 2019 22:14:42 GMT+0000 (Coordinated Universal Time)] "GET /static/js/app.d2a740801ff5d16d6e40.js.m
ap" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75 Chrome/73.0.3683.75
Safari/537.36"
cliente_1    | [Sun Apr 14 2019 22:14:42 GMT+0000 (Coordinated Universal Time)] "GET /static/js/manifest.2ae2e69a05c33dfc65f8
.js.map" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75 Chrome/73.0.368
3.75 Safari/537.36"
conductor_1  | Starting up http-server, serving dist
conductor_1  | Available on:
conductor_1  |   http://127.0.0.1: 8081
conductor_1  |   http://172.19.0.3: 8081
conductor_1  | Hit CTRL-C to stop the server
api_1        |  * Starting PostgreSQL 10 database server
api_1        |  * Removed stale pid file.
api_1        | Error: /usr/lib/postgresql/10/bin/pg_ctl /usr/lib/postgresql/10/bin/pg_ctl start -D /var/lib/postgresql/10/mai
n -l /var/log/postgresql/postgresql-10-main.log -s -o  -c config_file="/etc/postgresql/10/main/postgresql.conf"  exited with
status 1:
api_1        | 2019-04-14 17:12:52.426 -05 [25] LOG:  listening on IPv4 address "127.0.0.1", port 5432
api_1        | 2019-04-14 17:12:52.443 -05 [25] LOG:  could not bind IPv6 address "::1": Cannot assign requested address
api_1        | 2019-04-14 17:12:52.443 -05 [25] HINT:  Is another postmaster already running on port 5432? If not, wait a few
 seconds and retry.
api_1        | 2019-04-14 17:12:52.571 -05 [25] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
api_1        | 2019-04-14 17:12:52.929 -05 [26] LOG:  database system was interrupted; last known up at 2019-04-14 12:16:36 -
05
api_1        | 2019-04-14 17:13:51.822 -05 [26] LOG:  database system was not properly shut down; automatic recovery in progr
ess
api_1        | 2019-04-14 17:13:51.882 -05 [26] LOG:  redo starts at 0/2FF6BD0
api_1        | 2019-04-14 17:13:51.882 -05 [26] LOG:  invalid record length at 0/2FF6C08: wanted 24, got 0
api_1        | 2019-04-14 17:13:51.882 -05 [26] LOG:  redo done at 0/2FF6BD0
api_1        | 2019-04-14 17:13:52.142 -05 [25] LOG:  database system is ready to accept connections
api_1        | pg_ctl: server did not start in time
api_1        |    ...fail!
api_1        | npm WARN optional Skipping failed optional dependency /chokidar/fsevents:
api_1        | npm WARN notsup Not compatible with your operating system or architecture: fsevents@1.2.7
api_1        | npm WARN taxiapi@1.0.0 No description
api_1        | npm WARN taxiapi@1.0.0 No repository field.
api_1        |
api_1        | > taxiapi@1.0.0 start /app/taxiapi
api_1        | > node ./src/server.js
api_1        |
api_1        | Connected to db
api_1        | Server on port 8000
api_1        | Drivers consulted
api_1        | Drivers consulted
```

Ahora tiene los tres contenedores ejecutándose correctamente
cliente y conductor se comunican con la api y esta última contiene la bd con la que atiende las peticiones de los dos primeros

```
root@andresma-HP-Laptop-14-bp0xx:/home/andresma/Escritorio/Doke# docker ps
CONTAINER ID        IMAGE                              COMMAND                  CREATED             STATUS              PORTS
d5c8c916a4c7        andresma/cliente-taxi:version3     "http-server dist '-…"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp
_1
cd5a328da403        andresma/conductor-taxi:version3   "http-server dist '-…"   About a minute ago   Up About a minute   8080/tcp, 0.0.0.0:8081->8081/tcp
or_1
c464a379cd95        andresma/api-bd:version2           "sh inittodo.sh"         About a minute ago   Up About a minute   5432/tcp, 0.0.0.0:8000->8000/tcp
```

para detener todos los contenedores basta con escribir en la terminal que está usando

docker-compose down

2. TECNOLOGÍAS UTILIZADAS

La base de datos fue construida en Postgres junto con una librería conocida como Postgis, la cual nos permite trabajar con información espacial, la necesaria para ubicar objetos en un mapa. En la parte del Backend se hizo uso de tecnologías basadas en JavaScript, usando distintos frameworks como Express para una construcción rápida del servidor. En el Frontend se hizo uso de una tecnología conocida como VueJs la cual facilita la construcción de aplicaciones para el cliente, sobre ella se utilizaron librerías como bootstrap, Leaflet, Vuex, entre otras para otorgar a la aplicación el poder necesario para su correcto funcionamiento.

NPM https://www.npmjs.com/
VUEJS https://vuejs.org/
NODE JS https://nodejs.org/es/
Postgres https://www.postgresql.org/
Postgis https://postgis.net/
Leaflet https://leafletjs.com/
Open street Maps https://www.openstreetmap.org/#map=5/4.632/-74.299

3. API/REST

La api rest de la aplicación Univalle Taxi, está sobre nodejs y npm, y se usan módulos express, cors, body-parser, jsonwebtoken, router de express

Express: módulo para crear el servidor.
Cors: Para el manejo de coordenadas dentro nodejs.
Body-parser: Sirve para recoger información de una solicitud a la base de datos con req.body.
Jsonwebtoken: Tiene la función de guardar un fichero de cada usuario que se conecta a la página, a si la distinción entre cada cliente es única y de forma rápida.
Router: Sirve para redireccionamiento con un método de HTTP post
La api tiene los siguientes directorios src que contiene config y routes además de un archivo server.js que contiene la creación del servidor desde express

El directorio routes, tiene un archivo driver.js, este archivo contiene muchas funciones, y todas y cada una de estas funciones hace consultas a la base de datos, inserta, actualiza, elimina, y trae registros de las diferentes dependencias que tiene la base de datos y envía, trae información con el módulo router y el método post de http, usando promesas y el módulo axios que se usa desde la parte front-end de la aplicación, hay una transferencias de registros entre el front-end y back-end desde las funciones de este directorio

El archivo index.js contiene 2 métodos, el correspondiente al signup y al login que están en el front-end, en cada uno de estos métodos se hace consultas a la base de datos, en sign up inserta información a la base de datos que se trae de la parte front-end del usuario al registrarse y el login trae el phone y el password y hace una consulta para verificar si el usuario existe en la base de datos para dar permisos de acceso.

El archivo map.js tiene una función que trabaja con las coordenadas del conductor, cuando está disponible y se actualizan los registros varianteconductor por la posición del conductor

Profile.js y service.js tiene funciones que se enfocan en hacer consultas a la base de datos igual que en los archivos anteriores, inserta, elimina, actualiza y trae información de la base de datos y la envía a la parte front-end

4. POSTGRES SQL Y SU FUNCIONAMIENTO EN LA API Y APPs

En la base de datos se usó POSTGRESQL y POSTGIS, postgres para el control en LDM y LDD de la base de datos, postgis para el uso de geo localización.
Como se mencionó en el punto anterior, la api trabaja en conjunto con la apps cliente y conductor para compartir información y mostrarla en el navegador como estado final.
Diferentes métodos e instancias se han creado para hacer uso de esta funcionalidad de comunicación entre partes front-end y back-end. Se implementaron funciones almacenadas y triggers en la base de datos con el lenguaje plpgsql, para manipulación de datos.

5. DOCUMENTACIÓN FUNCIONES Y MÉTODOS.

En este capítulo se dará una descripción general de la funcionalidad de cada función que se ejecuta en la base de datos y en la api, los siguientes archivos todos usan la función async y await para realizar las consultas de forma asíncrona.
Todas las funciones descritas aquí, usan promesas y llamados a la base de datos con consultas, y llamados desde la parte front-end de la aplicación, estas llamadas llevan datos desde el front a la api.

Archivo driver.js
Metodo router.post(‘api/driver/signup')
Este método recibe información del front-end, todos los campos que un usuario tiene para poder registrarse, con el teléfono hace una consulta y mira en la base de datos si ya existe, si no existe hace otra consulta e inserta los campos traídos del front-end en la base de datos.

Router.post(‘api/driver/login’)
Comprueba si el teléfono y el password que se ingresa en los campos del front-end y traídos a la api, existen en la base de datos con una consulta select, si existen se da acceso al usuario , si no existen se manda un mensaje al front-end que no existe

Router.post(‘api/driver/profile’)
Este método con el teléfono del conductor hace una consulta select a la base de datos y me trae de la base de datos, toda la información de un usuario y la manda al front-end de la aplicación

Router.post(‘api/driver/update-profile’)
En este método se hace una consulta update sobre la base de datos y con el teléfono que se recibió de la parte front-end de la app para actualizar un determinado registro.

Router.post(api/driver/changed-password)
Actualiza el campo password de un cliente x según su telefono, se hace una consulta con un update para hacer esta acción de actualizar.

Router.post(‘api/driver/cars’)

Este método me retorna todos los automóviles que están asignados a un teléfono que llega desde el front-end, hace una consulta select en la BD para treat registros.

Router.post(api/driver/new-car)
Ingresa un auto nuevo a los registros de la BD, estos se hace con un insert into y los campos del taxi que llegan al método desde el el front-end.

Router.post('api/driver/delete-car')
Consulta la base de datos con una consulta delete y elimina el auto según una placa y un teléfono de conductor.

Router.post('api/driver/my-services')
Este método trae de la base de datos todos los campos de la entidad servicios, si encuentra algún registro en servicios, manda al front-end la información del servicio y la información del conductor que realizó el servicio con consultas select.

ARCHIVO INDEX.JS

2 métodos contiene api/signup/ y api/login descrito anteriormente.

ARCHIVO MAP.JS

api/map/info
Este método me retorna las coordenadas del conductor si la fecha es actual y si el taxista está disponible con una consulta select y varias subconsultas unidas a joins

ARCHIVO PROFILE.JS

api/profile
Este método hace una consulta a la base de datos bajo un select para mandar al front-end datos de un usuario

api/update-profile
Actualiza los registros de un usuario con una consulta update a la base de datos

api/profile/favorites
Hace una consulta select a la base de datos para traer y enviar al front-end el id_favoritos título, y coordenada en formato json.

api/profile/new-favorite
Con una consulta insert sobre la tabla favorites ingresa un registro, con sus respectivos campos

api/profile/delete-favorites

Elimina un registro de la tabla favorites, según un id_favoritos que llega del front-end, con una consulta delete y un where

api/profile/update-favorite
Actualiza un registro usando el atributo id_favorito que es la llave primaria de la tabla favorites

api/change-password
Una consulta update sobre la tabla cliente y el atributo password, para cambiar su contraseña, dependiendo del teléfono que llegue del front-end

/api/change-pic
Cambiar imagen de cliente, con una update sobre la tabla cliente y el atributo imagencliente donde el teléfono sea el que se recibe del front-end

api/near-taxi
Haciendo consultas select a la base de datos, se busca ep conductor que esté más cerca a un cliente

api/service-calification
Usa un update sobre la tabla servicio para calificar a un conductor con id_service, telefonocliente y telefonoconductor

FUNCIONES EN POSTGRES PLPGSQL:

InsertarTaxiConductor
Método que inserta un taxi y atributos del taxi taxi y taxicondcutor, si el id_carro,marca y modelo NO existen en la tabla infocarro, que es donde están la marca e info de los carros registrados en la BD, si existen entonces solo los ingresa en las tablas taxi y taxiconductor

Borrartaxi
Elimina el registro de un taxi con la placa del mismo y el teléfono del conductor, usando un delete sobre la tabla taxiconductor

Mitarifa
Retorna una id_tarifa dependiendo de la jornada y el tiempo, guarda la consulta en una variable y la retorna segun la condicion anterior.

Consultarviajesyditancia
Hace una consulta sobre la tabla servicio, para retornar una tabla con kilómetros recorridos y el número de viajes.

Consultarviajescond

Retorna una tabla con kms double precision, viajes integer, dinero integer, promedio double precision, al hacer una consulta select y varias subconsultas

Links a los repositorios

Api:
https://github.com/carban/taxiapi
Cliente:
https://github.com/carban/taxiclient
Conductor:
https://github.com/carban/taxidriver