# *Cheat-Sheet*

## *Microservices 7th - 8th Week*

## *Docker & CRUD Order´s Microservice*

Microservices Architecture (Introduction)

Teacher: Mr. Zheng Li

Student: Juan Albornoz

## 7th week´s practice:

1. Installed tools and platform
2. The docker container
3. Managing volumes for data persistency
4. Linking database with another docker container
5. Building and running with Docker-compose
6. Linking test.py and MySQL
7. Executing

## 8th week´s practice:

1. MySQL tables modeling
2. CRUD operations and definitions

    2.1. READ operation

    2.2. CREATE operation

    2.3. UPDATE operation

    2.4. DELETE operation

# 1. Installed tools and platform

DockerToolbox-19.03.1 I couldn't install the main docker resources because of a windows version problem, so I used Docker Toolbox as an alternative. (Windows 10 Home Single)

Django 3.0.7 already installed and working.

Also tried with Flask 1.1.2, although I started with Django and I'm not into it.

# 2. The docker container

While I was trying to starting up the server for Django and Flask I had the opportunity to put my hands on docker commands a lot, some of those are the next:

**$ docker run hello-world**

Since the image was not able in my local, docker searched it for me in his sources.

**$ docker run –rm hello-world**

Command for running an app and also remove it after.

**$ docker image ls (list images)**

```
uanox@LAPTOP-VEAHK10V MINGW64 /c/Program Files/Docker Toolbox
docker image ls
EPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
jangodockertest        latest              f363cfaad43c        21 minutes ago      968MB
ockertest_app          latest              2c6379aac655        7 hours ago         970MB
jango-docker           0.0.1               6daecccee073        7 hours ago         961MB
none>                  <none>              28c37fa54d90        7 hours ago         961MB
none>                  <none>              1a477aa8cfaf        7 hours ago         961MB
none>                  <none>              bb21d9d602ea        7 hours ago         961MB
ockertest              latest              1d58025715f8        7 hours ago         91.6MB
none>                  <none>              f6182ce7833a        7 hours ago         91.6MB
none>                  <none>              fc82842ab49a        7 hours ago         91.6MB
none>                  <none>              1d5062c52327        7 hours ago         91.6MB
none>                  <none>              8e2797f8612e        7 hours ago         91.6MB
none>                  <none>              e75c580aa2df        7 hours ago         91.6MB
lask-docker-demo-app   latest              bfc7e56a0180        8 hours ago         91.6MB
ysql                   latest              30f937e841c8        13 days ago         541MB
ython                  3.7.7               5e996c9d7c99        2 weeks ago         919MB
ython                  3                   659f826fabf4        2 weeks ago         934MB
ython                  alpine3.7           00be2573e9f7        16 months ago       81.3MB
```

**$ docker ps -all** (list containers created and the run states of them)

```
Juanox@LAPTOP-VEAHK10V MINGW64 /c/Program Files/Docker Toolbox
$ docker ps -all
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
6db3f049c045        dockertest_app      "python3 manage.py r…"   24 minutes ago      Up 24 minutes       0.0.0.0:8000->8000/tcp   dockertest_app_1
```

## 3. MySQL database in docker container

**$ docker run -d -p 33060:3306 --name mysql-db -e MYSQL_ROOT_PASSWORD=test mysql**

Command for running up a container with MySQL

**$ docker exec -it mysql-db mysql -p**

**$ create database testdb;**

**$ show databases;**

I can access the MySQL through the first command and create the database, show it, and all the database stuff.

TO KILL AND DELETE :

**$ docker kill mysql-db**

**$ docker rmi mysql-db**

**DELETE ALL IMAGES:**

**$docker image prune**

## 4. Managing volumes for data persistency

Using volumes for data persistency in the DB:

**$ docker rm -f mysql-db**

I removed the container and killed it

**$ docker volume prune**

Delete all the existing volumes

**$ docker volume create mysql-db-data**

Create a volume for my database

**$ docker run -d -p 33060:3306 --name mysql-db -e MYSQL_ROOT_PASSWORD=secret –mount src=mysql-db-data,dst=/var/lib/mysql mysql**

Command for running up the MySQL database, but mounting it on my recent created volume for persistency.

# 5. Building and running with Docker-compose

## 5.1. Python flask web app + MySQL + docker [FILES AND DIRS]

**$ pip install docker-compose**

Compose is a tool for defining and running multi-container Docker applications. It uses an YML file to configure your application's services.

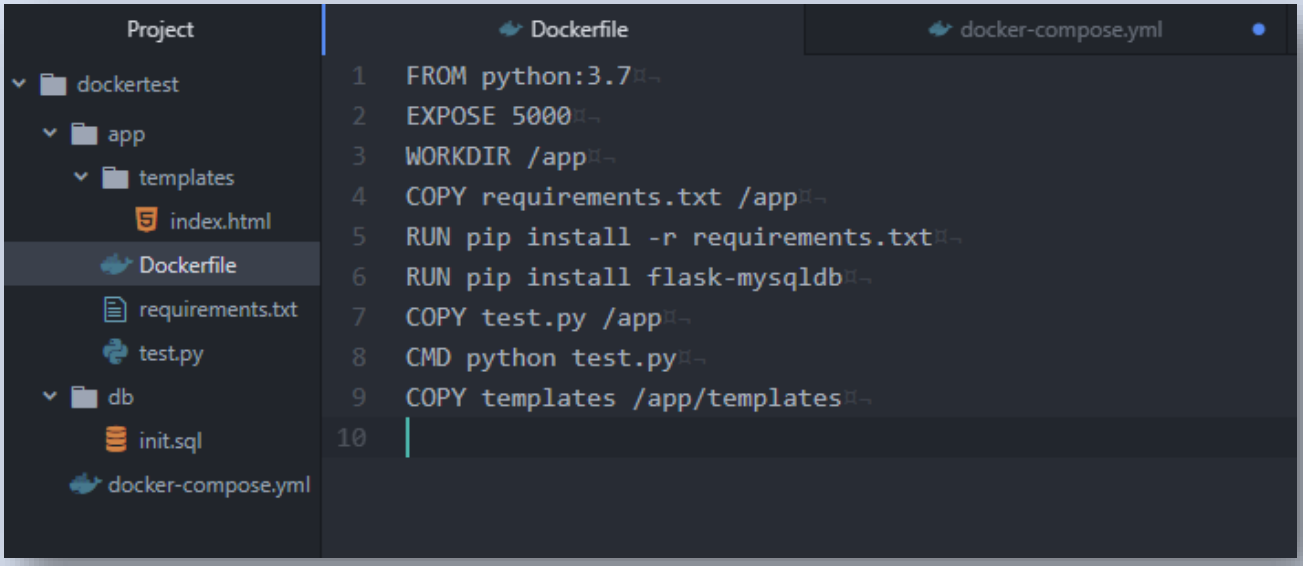
For building the image and running the container it requires three files:

docker-compose.yml for the app services configurations and running the containers

Dockerfile for the image building

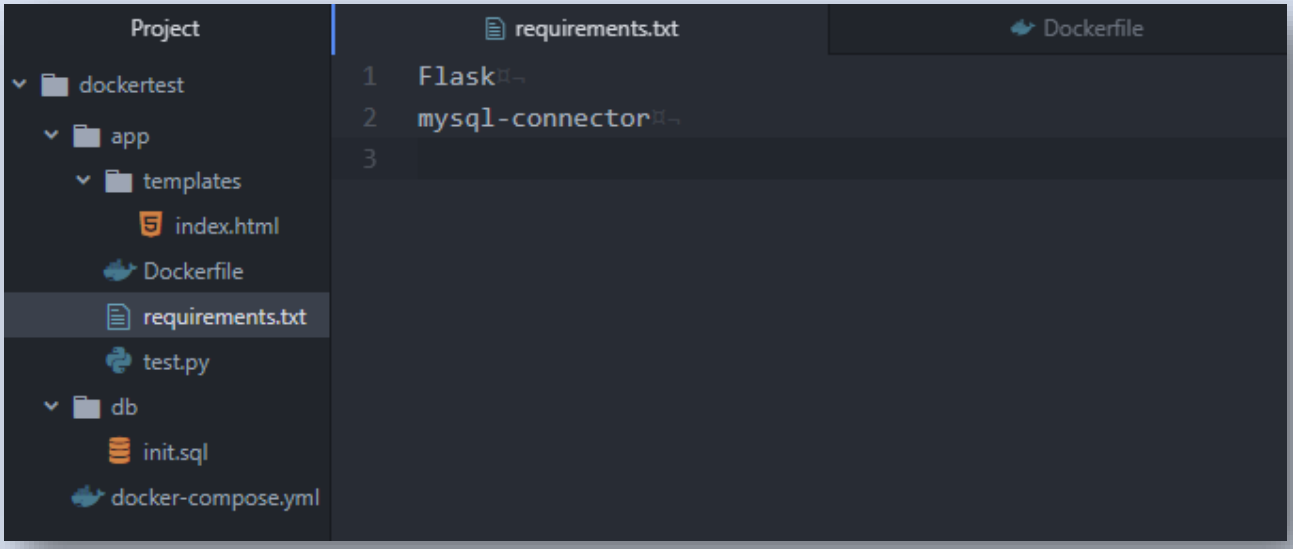
requirements.txt for the dependencies used by the image

The Dockerfile has the function of giving all the commands for building our image. This file also include the requirements.txt file for installing all the dependencies needed in the image. It's saved inside the app.py directory.

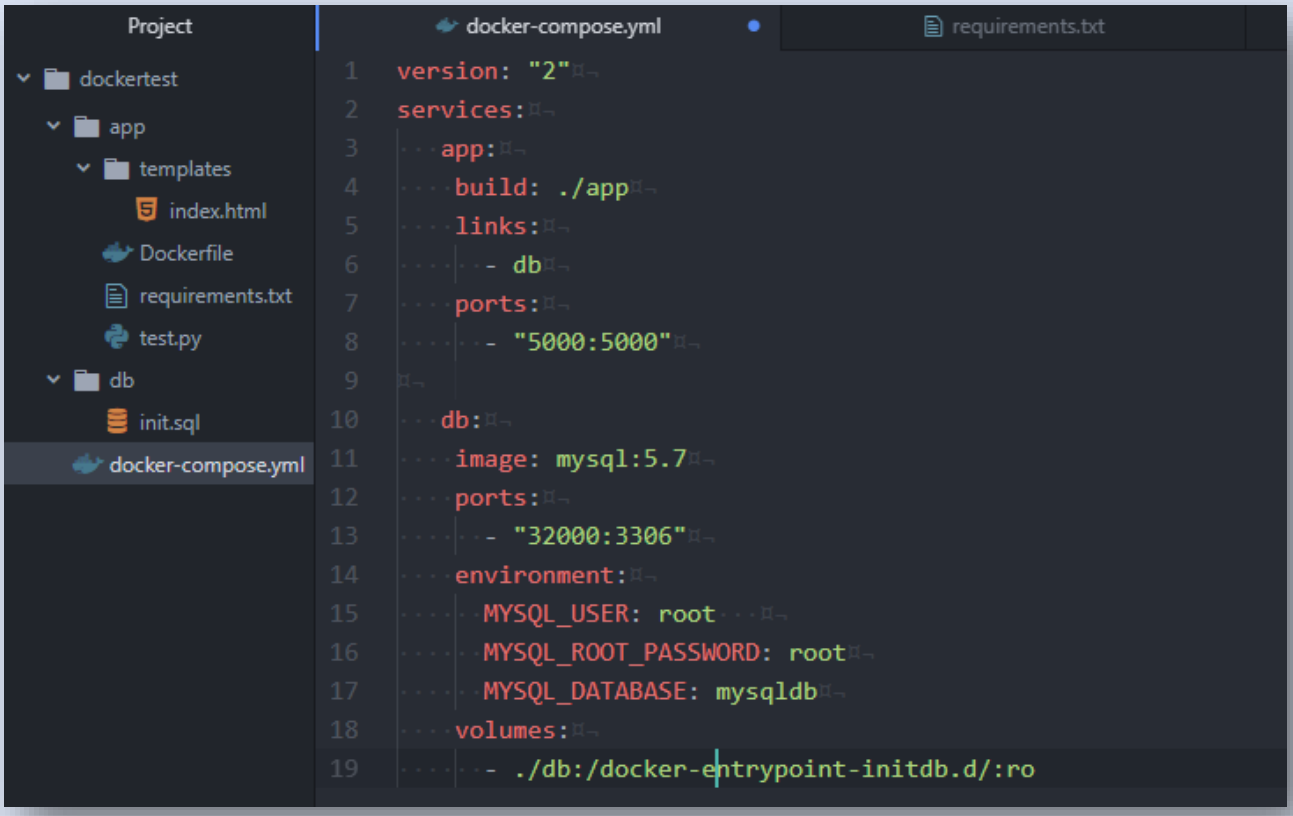



FROM, EXPOSE, WORKDIR, COPY, RUN and CMD are commands used by Docker.

The requirements.txt file is for dependencies and is also saved in the app.py directory:




The docker-compose.yml has the main services for the containers. Those are for the app itself and also for the MySQL database as it follows:




(*) Notice that the docker-compose.yml file belongs to the Flask's root directory

## 5.2. Python flask web app + MySQL + docker [SHELL COMMANDS]

**$ docker-compose build**

It takes the docker-compose.yml file and builds two images; the dockertest_app image with and ID and a size and also the mysql 5.7 image as you can see:

$ **docker image ls**

```
PS C:\Users\Juanox\github\dockertest> docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
dockertest_app      latest              9f35cbf6b951        46 minutes ago      944MB
python              3.7                 e4e55e98f1e0        15 hours ago        919MB
dockerflasktest     latest              1a1b82ff3d4a        16 hours ago        91.6MB
mysql               5.7                 9cfcce23593a        25 hours ago        448MB
mysql               latest              30f937e841c8        2 weeks ago         541MB
python              3                   659f826fabf4        3 weeks ago         934MB
postgres            latest              adf2b126dda8        3 weeks ago         313MB
```

**$ docker-compose up**

The command to create the containers and put them in running mode

```
PS C:\Users\Juanox\github\dockertest> docker-compose up
dockertest_db_1 is up-to-date
dockertest_app_1 is up-to-date
Attaching to dockertest_db_1, dockertest_app_1
app_1  |  * Serving Flask app "test" (lazy loading)
app_1  |  * Environment: production
app_1  |    WARNING: This is a development server. Do not use it in a production deployment.
db_1   | 2020-06-10 07:44:26+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.30-1debian10 started.
db_1   | 2020-06-10 07:44:26+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
app_1  |    Use a production WSGI server instead.
app_1  |  * Debug mode: on
db_1   | 2020-06-10 07:44:26+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.30-1debian10 started.
db_1   | 2020-06-10 07:44:27+00:00 [Note] [Entrypoint]: Initializing database files
db_1   | 2020-06-10T07:44:27.086376Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp s
db_1   | 2020-06-10T07:44:27.332043Z 0 [Warning] InnoDB: New log files created, LSN=45790
db_1   | 2020-06-10T07:44:27.373543Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
```

**$docker-compose ps**

```
PS C:\Users\Juanox\github\dockertest> docker-compose ps
      Name                    Command             State                 Ports
-----------------------------------------------------------------------------------------
dockertest_app_1   /bin/sh -c python test.py     Up      0.0.0.0:5000->5000/tcp
dockertest_db_1    docker-entrypoint.sh mysqld   Up      0.0.0.0:32000->3306/tcp, 33060/tcp
```

**TO KILL THE CONTAINERS:**

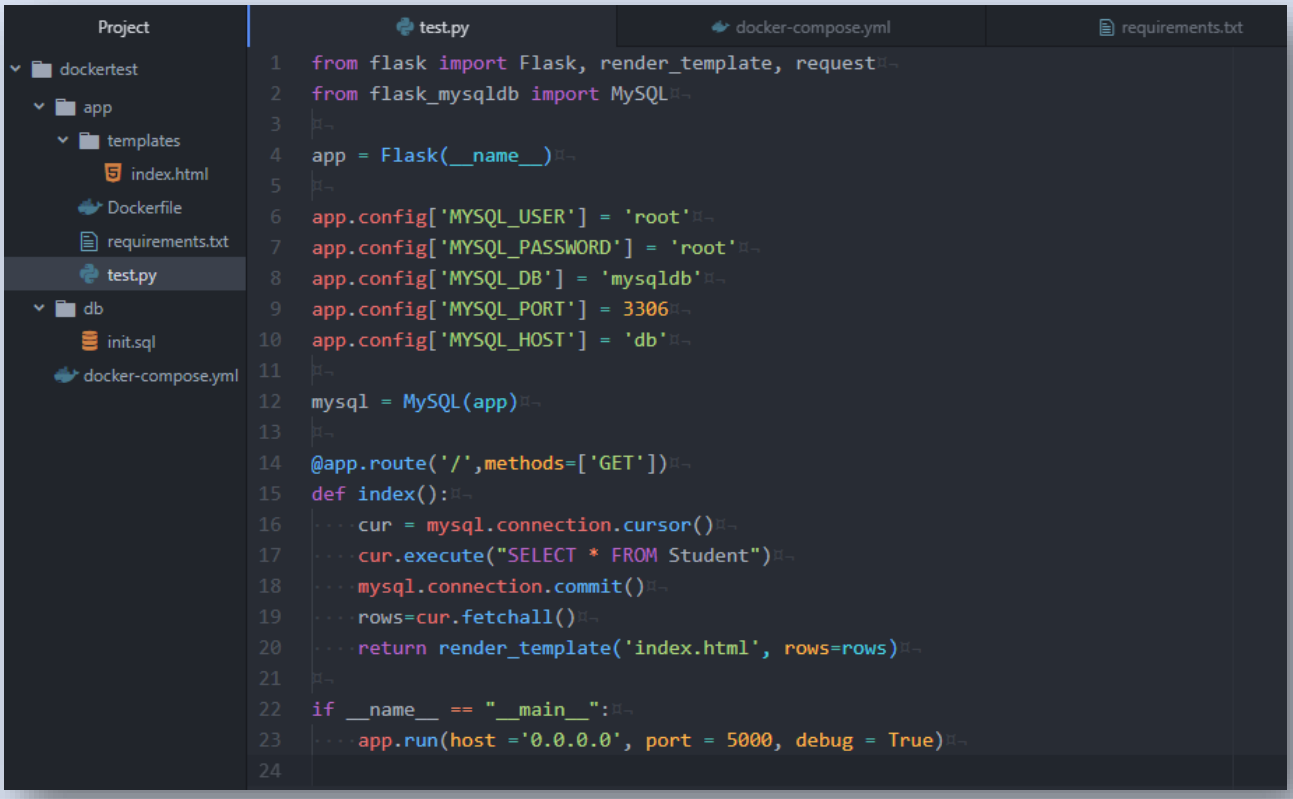
**$docker-compose down**

**TO REMOVE THE IMAGES:**

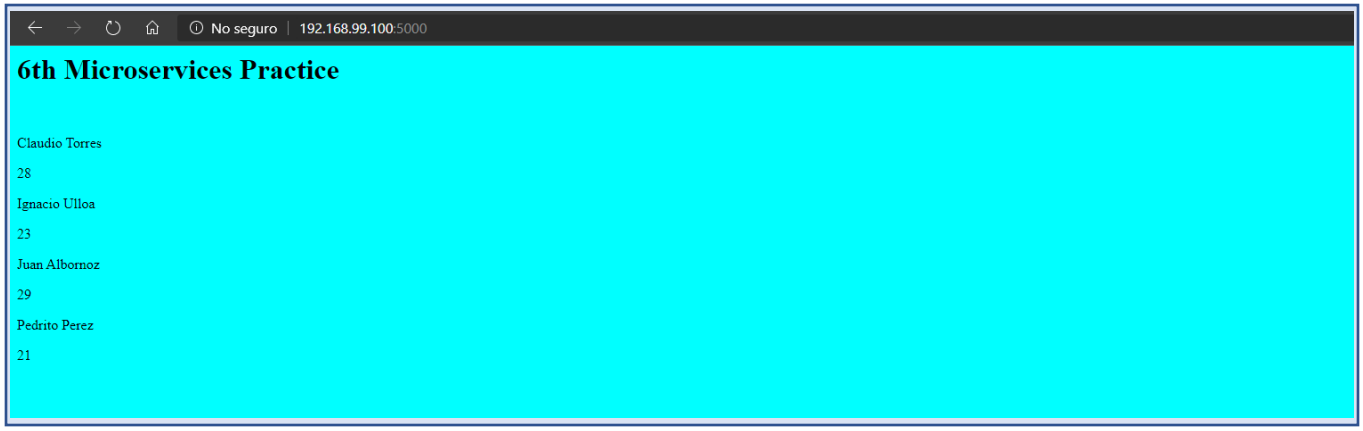
**$docker rmi dockertest_app & $docker rmi mysql:version**

## 6. Linking test.py and MySQL

- I used the MySQL module from flask_mysqldb.
- mysql.connection.cursor() opens the connection with the database.
- execute() to giving instructions in SQL language.
- commit() to send it.
- fetchall() to fetch some few rows in my database.




## 7. Executing [192.68.99.100:5000]



# 8th week´s practice

## 1. MySQL tables modeling

Pet & costumer tables: These tables are just for a proper implementation of CRUD operations on the Order table.

Customer: Primary Key: *idCustomer*

| idCustomer | nameCustomer | emailContact | phoneContact |
|---|---|---|---|
| 16.744.216-5 | Ignacio Calbucoy | ficaigna@gmail.com | +569892837489 |
| 16.897.422-1 | Iván Albornoz | ivanalbornozaraya@gmail.com | +56992784742 |
| 17.614.945-0 | Juan Albornoz | jalbornoza@udec.cl | +56936556155 |
| 9.428.544-5 | Maria Araya | mas212711@gmail.com | +56998273645 |
| NULL | NULL | NULL | NULL |

Pet: Primary Key: *idPet*

| idPet | specie | color | months_age | price |
|---|---|---|---|---|
| 347 | Chihuahua | Brown | 5 | 200000 |
| 348 | Chihuahua | White | 6 | 250000 |
| 469 | Hamster | Bright Brown | 2 | 30000 |
| 473 | Hamster | White | 3 | 45000 |
| 689 | Clown-fish | Red/white | 2 | 50000 |
| 704 | Persian-cat | Gold | 4 | 120000 |
| NULL | NULL | NULL | NULL | NULL |

Order: Primary Key: *idOrder* and Foreign Key: *idCostumer*, *idPet*

(*) *taxes_amount, total_amount and date_time* are autogenerated fields

| idOrder | idCustomer | idPet | pet_price | taxes_amount | total_amount | date_time |
|---|---|---|---|---|---|---|
| 2 | 17.614.945-0 | 347 | 200000 | 50000 | 250000 | 2020-06-15 06:52:04 |
| 3 | 9.428.544-5 | 704 | 120000 | 30000 | 150000 | 2020-06-15 06:54:22 |
| 4 | 16.897.422-1 | 473 | 45000 | 11250 | 56250 | 2020-06-15 07:23:54 |
| 6 | 16.897.422-1 | 469 | 30000 | 7500 | 37500 | 2020-06-16 00:58:32 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

## 2. CRUD operations and definitions

## 2.1. READ operation

Read operations are implemented in the index() controller function:

```
cur.execute("SELECT * FROM mysqldb.Order")
mysql.connection.commit()
rows2 = cur.fetchall()
```

It executes the SQL instruction and fetches all the rows to render them in the template´s table as it follows:

```
return render_template('index.html', rows=rows, rows2=rows2, rows3=rows3)
```

(*) I used Bootstrap tables to render the data in the html file.

192.168.99.100:5000

### Orders

| Date/Time | ID Order | ID Customer | ID Pet | Price | Taxes | Total | Options |
|---|---|---|---|---|---|---|---|
| 2020-06-15 06:52:04 | 2 | 17.614.945-0 | 347 | $200000 | $50000 | $250000 | Modify Delete |
| 2020-06-15 06:54:22 | 3 | 9.428.544-5 | 704 | $120000 | $30000 | $150000 | Modify Delete |
| 2020-06-15 07:23:54 | 4 | 16.897.422-1 | 473 | $45000 | $11250 | $56250 | Modify Delete |
| 2020-06-16 00:58:32 | 6 | 16.897.422-1 | 469 | $30000 | $7500 | $37500 | Modify Delete |

Add Order

### Pets

| ID Pet | Specie | Color | Age(months) | Price |
|---|---|---|---|---|
| 347 | Chihuahua | Brown | 5 | $200000 |
| 348 | Chihuahua | White | 6 | $250000 |
| 469 | Hamster | Bright Brown | 2 | $30000 |
| 473 | Hamster | White | 3 | $45000 |
| 689 | Clown-fish | Red/white | 2 | $50000 |
| 704 | Persian-cat | Gold | 4 | $120000 |

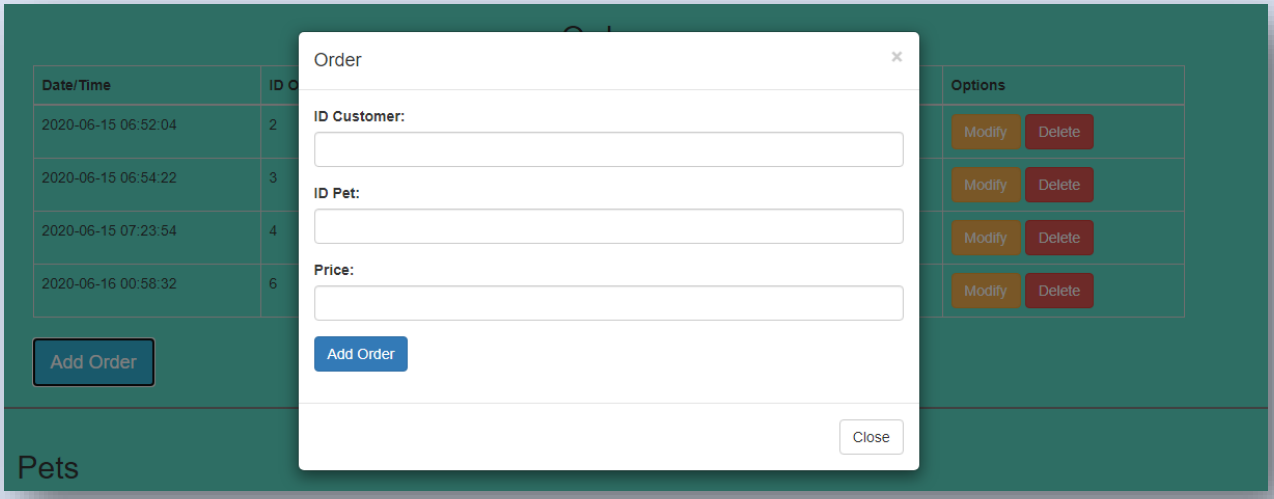
### Customers

| ID Customer | Name Customer | Email Contact | Phone Contact |
|---|---|---|---|
| 16.744.216-5 | Ignacio Calbucoy | ficaigna@gmail.com | +569892837489 |
| 16.897.422-1 | Iván Albornoz | ivanalbornozaraya@gmail.com | +56992784742 |
| 17.614.945-0 | Juan Albornoz | jalbornoza@udec.cl | +56936556155 |
| 9.428.544-5 | Maria Araya | mas212711@gmail.com | +56998273645 |

## 2.2. CREATE operation

By using forms implemented in the flask environment I can catch the data entered in the template and send it back to the controller via "POST" method and a defined route to the def add() function. The form to add the order is the next:

(*) I used *modal* resource from Bootstrap for showing the form in a kind of emergent window.




The data from the formulary is saves in variables and then passed to the SQL instruction. I'm using mysqldb from flask and it uses a format way to manage variables, in my case the variables are idCostumer, idPet and price required for creating a row in the Order table.

```
@app.route('/add', methods=['POST'])
def add():
    idCostumer = request.form['idCostumer']
    idPet = request.form['idPet']
    price = request.form['price']
    cur = mysql.connection.cursor()
    cur.execute("INSERT INTO mysqldb.Order (idCustomer, idPet, pet_price) VALUES (%s, %s, %s)",
                (idCostumer, idPet, price))
    mysql.connection.commit()
    return redirect(url_for('index'))
```

## 2.3. UPDATE operation

In a similar way to the CREATE operation, to update an Order def modify() uses "POST" method to receive the updated entries in the form (the same modal form used by CREATE). The main difference is that the form must be companied with the primary key as it follows (also catched by "POST"):

```
46                    <form action="{{url_for('modify')}}" method="POST">
47                        <div class="form-group">
48                            <label for="idCustomer"> ID Customer: </label>
49                            <input type="hidden" name="id" value={{row.0}}>
50                            <input type="text" class="form-control" name="idCustomer" value={{row.
51                        </div>
52                        <div class="form-group">
53                            <label for="idPet"> ID Pet: </label>
54                            <input type="hidden" name="id" value={{row.0}}>
55                            <input type="text" class="form-control" name="idPet" value={{row.2}}>
56                        </div>
57                        <div class="form-group">
58                            <label for="price"> Price: </label>
59                            <input type="hidden" name="id" value={{row.0}}>
60                            <input type="text" class="form-control" name="price" value={{row.3}}>
61                        </div>
62                        <div class="form-group">
63                            <button type="submit"class="btn btn-warning">Update</button>
64                        </div>
65                    </form>
```

The controller with the def modify() function defined and the route statement:

```
43  @app.route('/modify', methods=['POST'])
44  def modify():
45      idOrder = request.form['id']
46      idCustomer = request.form['idCustomer']
47      idPet = request.form['idPet']
48      price = request.form['price']
49      cur = mysql.connection.cursor()
50      cur.execute("UPDATE mysqldb.Order SET idCustomer=%s, idPet=%s, pet_price=%s WHERE idOrder=%s",
51                  (idCustomer, idPet, price, idOrder))
52      mysql.connection.commit()
53      return redirect(url_for('index'))
```