

Online Shop Template

Oriol Filter Anson
filter.oriol@gmail.com

May 28, 2021



Contents

1	Introduction	1
1.1	Description	2
1.2	Motivation	2
1.3	Keywords	2
1.4	Main Objective	3
1.5	Secondary Objectives	4
1.6	Reasons	5
1.6.1	PHP	5
1.6.2	Docker	6
1.6.3	Docker-compose	6
1.6.4	Portainer	7
1.6.5	GitHub	7
1.6.6	PostgreSQL	8
1.6.7	Nginx	9
2	Demo	10
2.1	Main Page	10
2.2	Registration Success Testing	11
2.3	Registration Fail Testing	12
2.4	Login Unactivated Account Fail Testing	13
2.5	Received email Testing	13
2.6	Activate Account Succeed Testing	14
2.7	Login Success Testing	14
2.8	Add Shipping Address Fail Testing	15
2.9	Add Shipping Address Success Testing	15
2.10	Remove Shipping Address Success Testing	16
2.11	Cookies Testing	16
3	Docker Configuration	17
3.1	Docker Network Scheme	17
3.2	Docker Distribution Explanation	18
3.2.1	Nginx	18
3.2.2	PHP	18
3.2.3	Postgresql	18
3.2.4	Adminer	19
3.2.5	Portainer	19
3.2.6	Backup_dealer	19
3.2.7	SFTP	19

3.3	Main Server Docker Configuration	20
3.3.1	Main Server Docker-Compose	20
3.3.2	PHP Dockerfile	24
3.3.3	Postgresql Dockerfile	25
3.3.4	Backup Dealer Docker-Compose	26
3.3.5	Backup Dealer Dockerfile	27
4	Services Configuration	29
4.1	Portainer Configuration	29
4.2	Adminer Configuration	29
4.3	PHP Configuration	29
4.4	Nginx Configuration	29
4.5	PostgreSQL Configuration	30
4.5.1	Contact database structure	31
4.5.2	Shop database structure	32
4.5.3	Plugins enabled	32
4.5.4	Data structure	33
4.5.5	Main functions	35
4.6	Data Validation	36
4.6.1	Regex	36
4.6.2	Javascript	37
4.6.3	PostgreSQL	37
4.6.4	PHP	37
4.7	Error Thread	37
4.7.1	Notes	37
4.7.2	Error Codes	37
5	Web Page Documentation	42
5.1	File organization	42
5.2	Public Directory Content	43
5.3	Private Directory Content	44
5.4	Code Structure	44
5.4.1	Library/package code structure	45
5.4.2	Public page code structure	45
5.4.3	API code structure	46
5.4.4	Main Class listing	48
6	Services Deployment	51
6.1	System requirements	51
6.1.1	Git installation	51

6.1.2	Docker installation	51
6.1.3	Docker configuration - allow user to use docker	52
6.1.4	Docker configuration - enable docker on boot	52
6.1.5	Docker-Compose installation	52
6.1.6	Repository installation	53
6.2	Repository deployment minimal customization	53
6.2.1	Main server deployment minimal customization	53
6.2.2	Backups client deployment minimal customization	54
6.2.3	Cron periodical backups minimal customization	54
6.2.4	Backups Server deployment minimal customization	55
6.3	Repository deployment booting services	55
6.3.1	Main Server Service Booting	55
6.3.2	Backup (Remote) Server Service Booting	55
6.3.3	Backup (Local) Server Service Booting	55
6.4	Repository deployment further customization	56
6.4.1	Custom SSL Keys	56
6.4.2	Custom Keychains	56
6.4.3	Adding more folders or modifying the current docker_backup user	56
6.4.4	Deploying Custom Databases	57
6.4.5	Change the current email sender for your own one	59

1 Introduction

This section will make a resume about what's this project about, commend which where the motivation that bring me to carry on this project, how to implement it and use it, alongside with how to modify its behaviour. With the intention to make reader able to implement and configure it at its desire following simple and concise steps while providing a deep understanding of the followed actions.

1.1 Description

This project is intended to facilitate implementing an online shop with a base infrastructure. Offering the capability of:

- Infrastructure based on dockers.
- Default web with simple configuration.
- Defined database structure.
- Backup automation to a local drive or remote server.
- Automatic error management among the web-database relation limiting the occurrence of errors.

1.2 Motivation

The motivation behind this project, was mainly finding an excuse in order to use different technologies and try to combine them, doing something that gives me a deeper understanding of the technologies and tools used.

1.3 Keywords

PHP, PDO, Docker, Docker-compose, Dockerfile, Network, Api, Security, Infrastructure, Web, Nginx, Server, Portainer, Deploy, Swarm, LaTeX, Yaml, JSON, SSL, Certificates, SSH, Key, Users, Passwords, PostgreSQL, Stack, Groups, Shop, Online, Environment, Backups, Motorisation, Script, Bash, Shell, Git, Markdown, HTML, JavaScript, AJAX, Regex, Automation, Postgres, Database, HTTP, HTTPS, Client, Server, Crontab.

1.4 Main Objective

The main objective that bring me to build this project, was acquiring a deeper understanding about database management while providing a secure interface for its users in order to interact with its accounts without affecting at the response time from the client petitions, focused on the information minimization required for the user, and its security while using our services. Another topic that woke curiosity on me, was about regarding how API worked since neither knew how to implement nor interact with them, and meanwhile.

1.5 Secondary Objectives

Meanwhile wasn't something that had on mind during the start of the project, it's something that built during the production of it, which was the desire of improve past knowledge and learn about new tools, familiarizing myself with its different applications and its possibilities.

Some of them are:

- Dockerfiles and the production of new images.
- Github, and how to keep track of a project and its updates.
- PDO and how facilitates updating our database system without the need of updating the code of our existing pages.

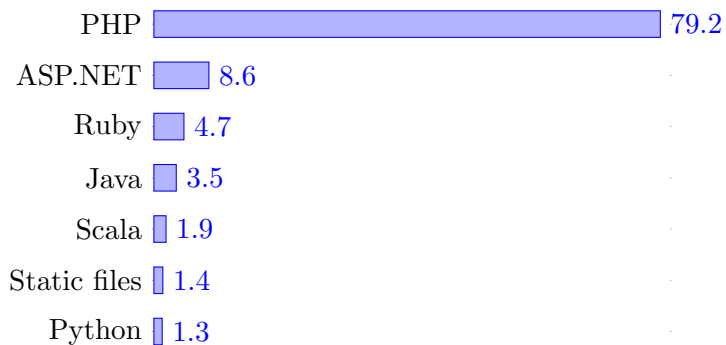
1.6 Reasons

1.6.1 PHP



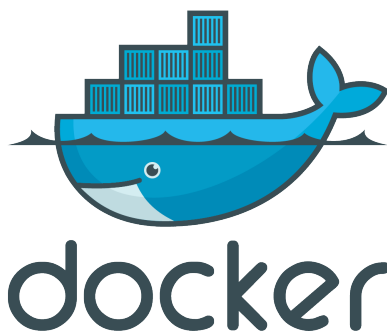
The main reason I decided to use PHP over any other technology, was its actual usage among the world, which, taking a look at this graph, we can observe that its usage it's almost an 80%, which confirms that even if there are upcoming technologies, PHP will keep there for a long time, so was a good idea familiarize myself with that language.

Web language usage among the world (26-May-2021)



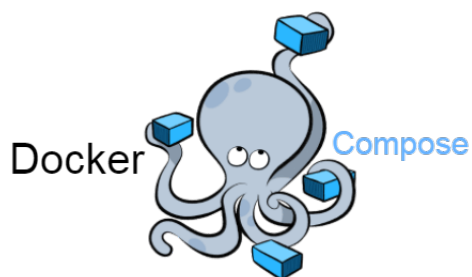
https://w3techs.com/technologies/overview/programming_language

1.6.2 Docker



Regarding the docker decision, there wasn't much to think about, since docker is a modern technology which I already knew the bases, making it easier to pick up and start working earlier.

1.6.3 DockerCompose



This tool facilitates deploying services on a server, while also being able to scalate the services or deploy them in a swarm, so there was no excuse to avoid its usage.

1.6.4 Portainer



Taking a look at different monitoring tools, decided to use docker portainer mainly by it's fast set-up. While still being up to the tasks demanded, which consist in monitoring and managing.

1.6.5 GitHub



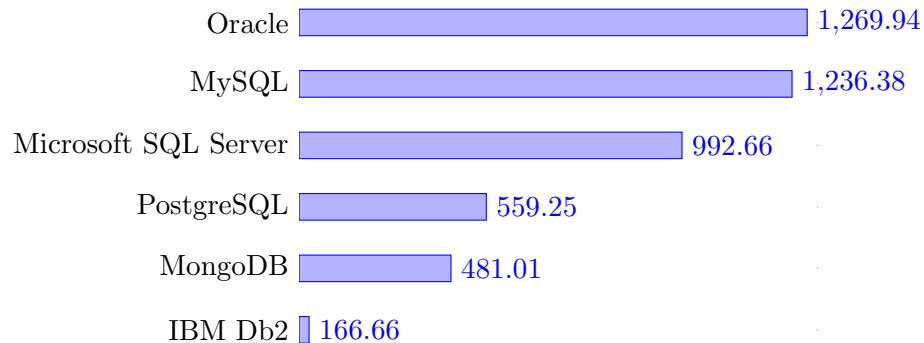
On the other hand, I decided to use GitHub to store the project mainly due having a basic understanding of how the tools work, since there are multiple tools that suits the same function, felt like that was the right decision.

1.6.6 PostgreSQL



Meanwhile the web language was picked based on its global usage, I personally already have experience with Oracle, MySQL, MariaDB and MongoDB, so in order to try a different technology, I decided to use PostgreSQL, since it suited my needs while also learning a new database, yet, its position among the ranking, made the decision easier to take.

Database global ranking (May 2021)



<https://db-engines.com/en/ranking>

1.6.7 Nginx




So far, being the main two options Apache and Nginx, I was quite limited when it came to variety, and yet, both suit its job very well, yet, due its minor differences, decided to use Nginx instead of Apache, since its less resource hungry while being more configurable-wise.

WebServers global comparasion

Nginx  34

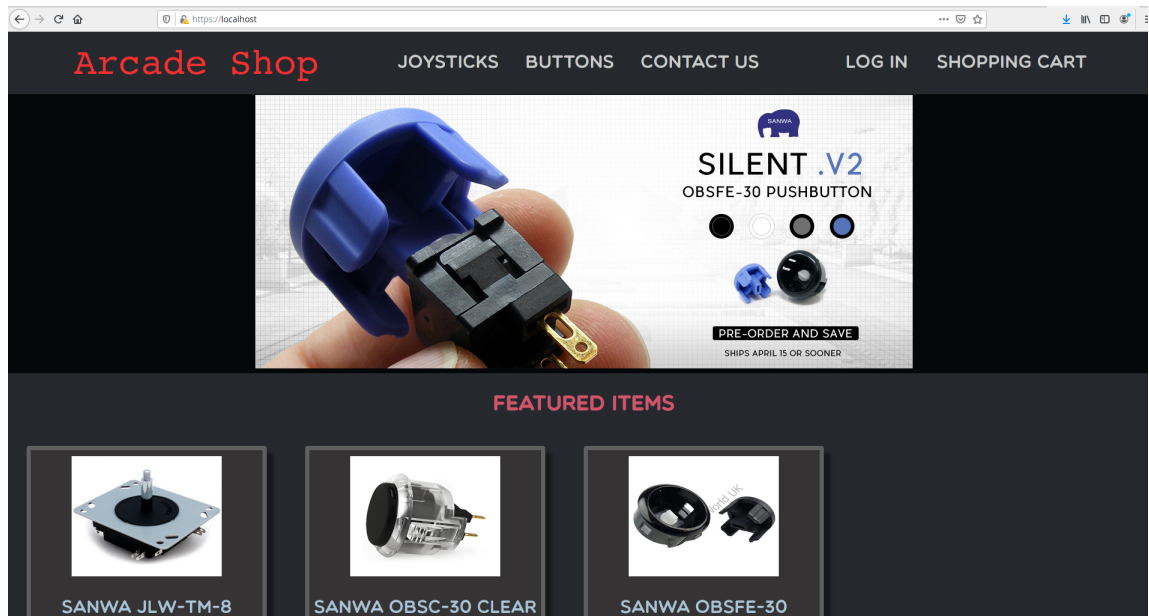
Apache  33.4

Microsoft-IIS  6.8

<https://w3techs.com/technologies/comparison/ws-apache,ws-microsoftiis,ws-nginx>

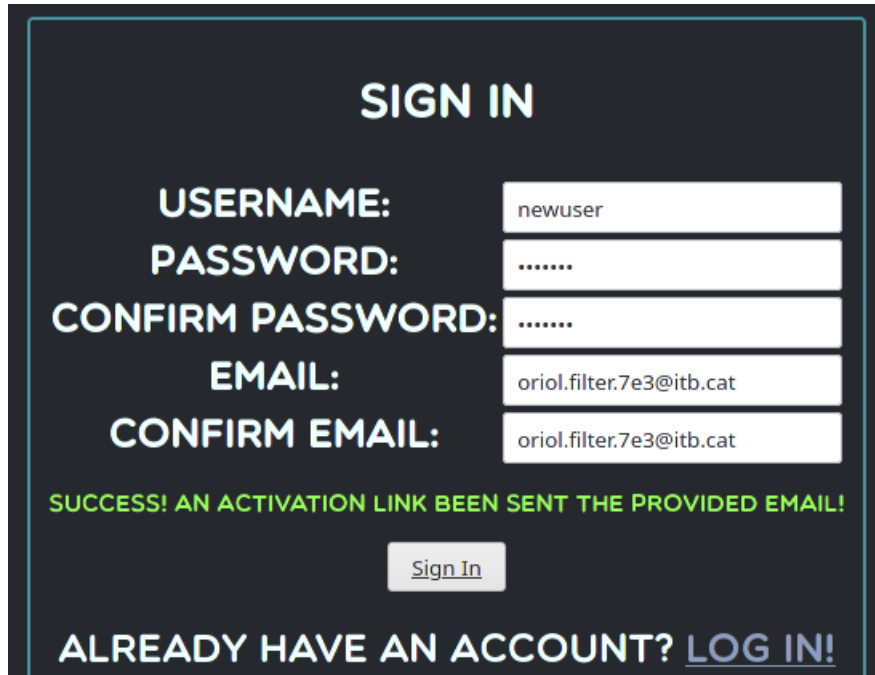
2 Demo

2.1 Main Page



A simple view of the menu with different menus available, and a product showcase.

2.2 Registration Success Testing



SIGN IN

USERNAME:

PASSWORD:

CONFIRM PASSWORD:

EMAIL:

CONFIRM EMAIL:

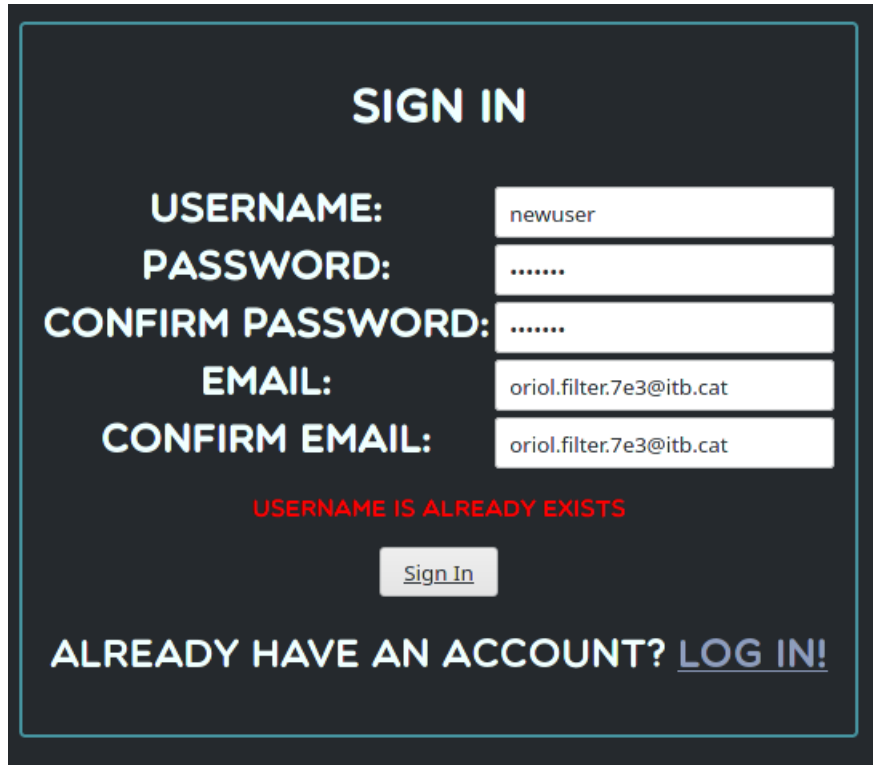
SUCCESS! AN ACTIVATION LINK BEEN SENT THE PROVIDED EMAIL!

[Sign In](#)

ALREADY HAVE AN ACCOUNT? [LOG IN!](#)

We are able to receive responses from the server without the need of updating the page, in this case we were able to succeed in our registration.

2.3 Registration Fail Testing



The image shows a dark-themed 'SIGN IN' form. It contains five input fields: USERNAME (filled with 'newuser'), PASSWORD (filled with '.....'), CONFIRM PASSWORD (filled with '.....'), EMAIL (filled with 'oriol.filter.7e3@itb.cat'), and CONFIRM EMAIL (filled with 'oriol.filter.7e3@itb.cat'). Below the fields is a red error message 'USERNAME IS ALREADY EXISTS'. A 'Sign In' button is located below the error message. At the bottom, there is a link 'ALREADY HAVE AN ACCOUNT? LOG IN!'.

SIGN IN

USERNAME: newuser

PASSWORD:

CONFIRM PASSWORD:

EMAIL: oriol.filter.7e3@itb.cat

CONFIRM EMAIL: oriol.filter.7e3@itb.cat

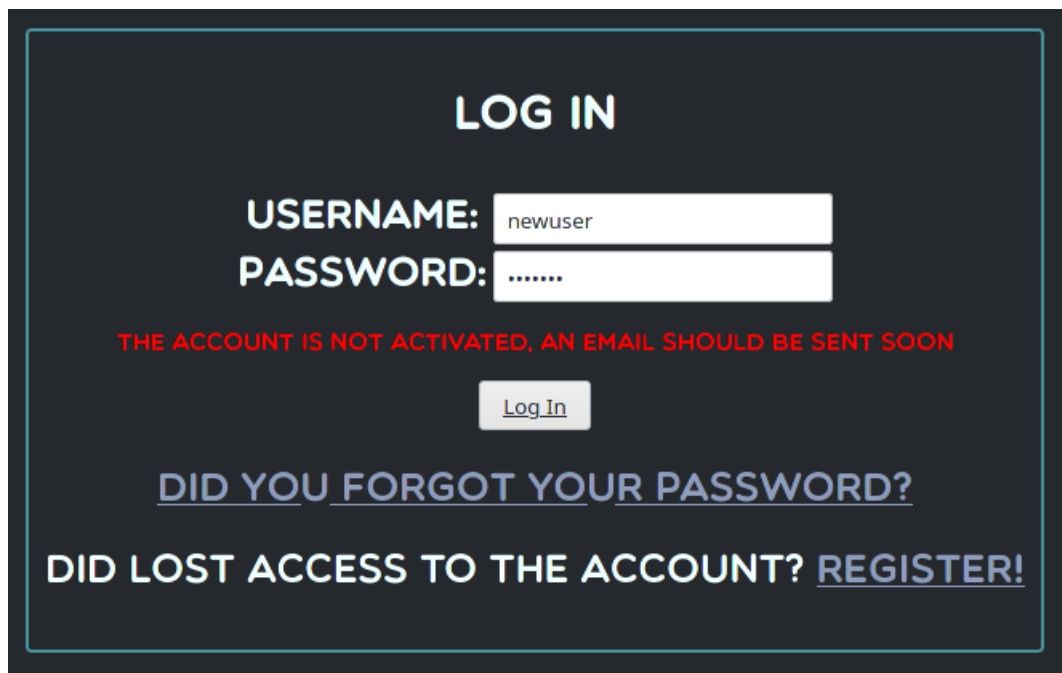
USERNAME IS ALREADY EXISTS

[Sign In](#)

ALREADY HAVE AN ACCOUNT? [LOG IN!](#)

Since our user was already registered we receive a error response.

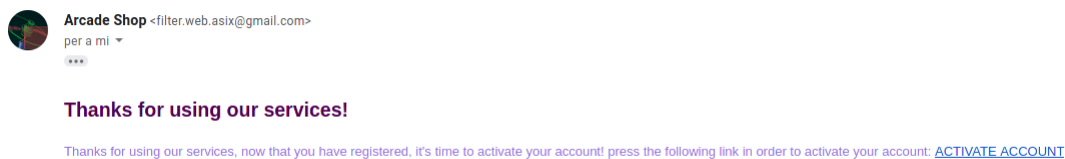
2.4 Login Unactivated Account Fail Testing



The screenshot shows a dark-themed login interface. At the top, the text "LOG IN" is displayed in white. Below it, the "USERNAME:" field contains "newuser" and the "PASSWORD:" field contains ".....". A red error message states: "THE ACCOUNT IS NOT ACTIVATED, AN EMAIL SHOULD BE SENT SOON". Below the error message is a "Log In" button. At the bottom, there are two links: "DID YOU FORGOT YOUR PASSWORD?" and "DID LOST ACCESS TO THE ACCOUNT? REGISTER!".

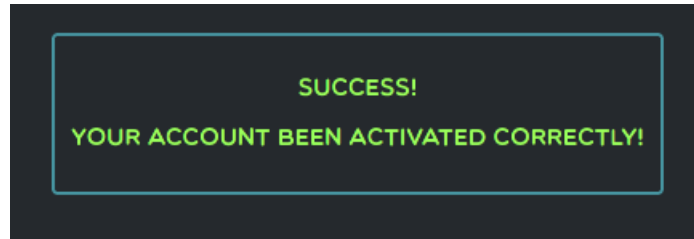
In order to activate our account, we need to activate our account via the received email.

2.5 Received email Testing



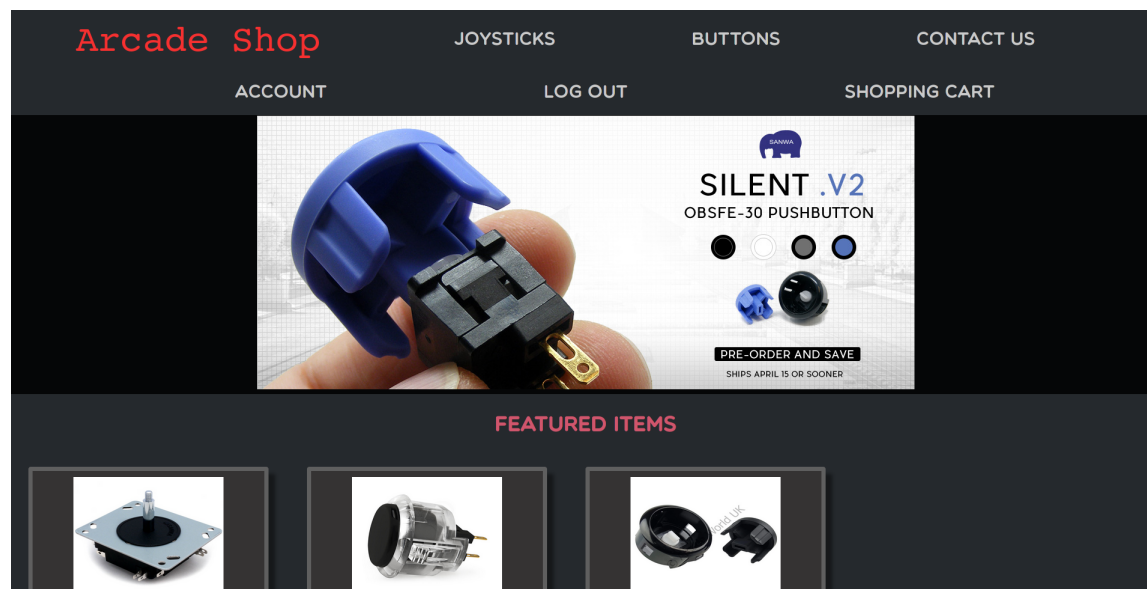
As expected, we are able to receive a mail to our given email.

2.6 Activate Account Succeed Testing



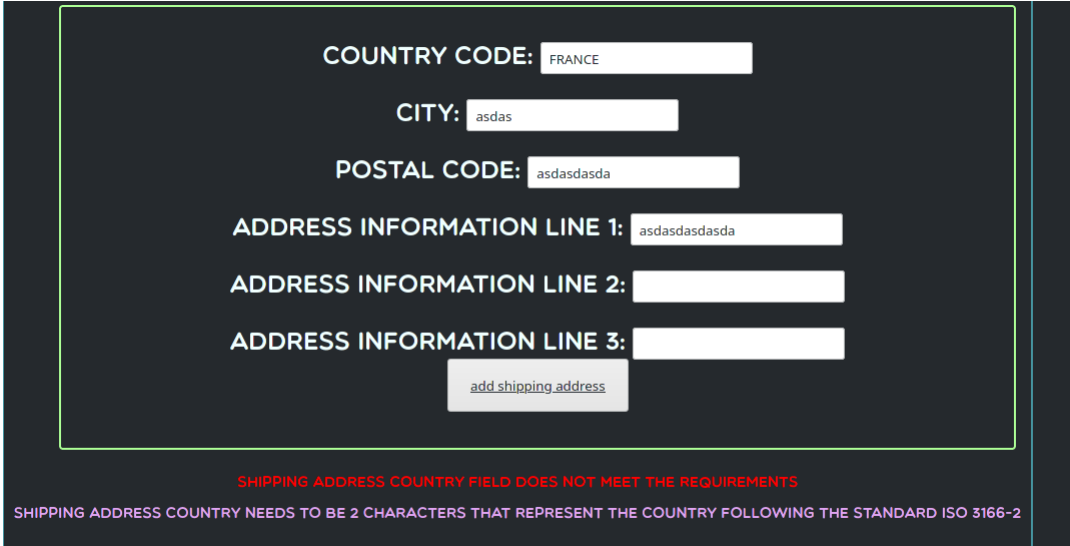
Once we open the link given by the server through the mail, we are able to activate our account.

2.7 Login Success Testing



Once we have our account activated, we are able to login correctly, and afterwards we are redirected to the main menu, as a proof we can see how the menu it's quite different compared with when we didn't log in.

2.8 Add Shipping Address Fail Testing



A screenshot of a shipping address form on a dark background. The form contains several input fields with labels: 'COUNTRY CODE:' with 'FRANCE' entered, 'CITY:' with 'asdas' entered, 'POSTAL CODE:' with 'asdasdasda' entered, 'ADDRESS INFORMATION LINE 1:' with 'asdasdasdasda' entered, 'ADDRESS INFORMATION LINE 2:' (empty), and 'ADDRESS INFORMATION LINE 3:' (empty). Below the fields is a button labeled 'add shipping address'. At the bottom of the form, there is a red error message: 'SHIPPING ADDRESS COUNTRY FIELD DOES NOT MEET THE REQUIREMENTS' and a purple message: 'SHIPPING ADDRESS COUNTRY NEEDS TO BE 2 CHARACTERS THAT REPRESENT THE COUNTRY FOLLOWING THE STANDARD ISO 3166-2'.

COUNTRY CODE: FRANCE

CITY: asdas

POSTAL CODE: asdasdasda

ADDRESS INFORMATION LINE 1: asdasdasdasda

ADDRESS INFORMATION LINE 2:

ADDRESS INFORMATION LINE 3:

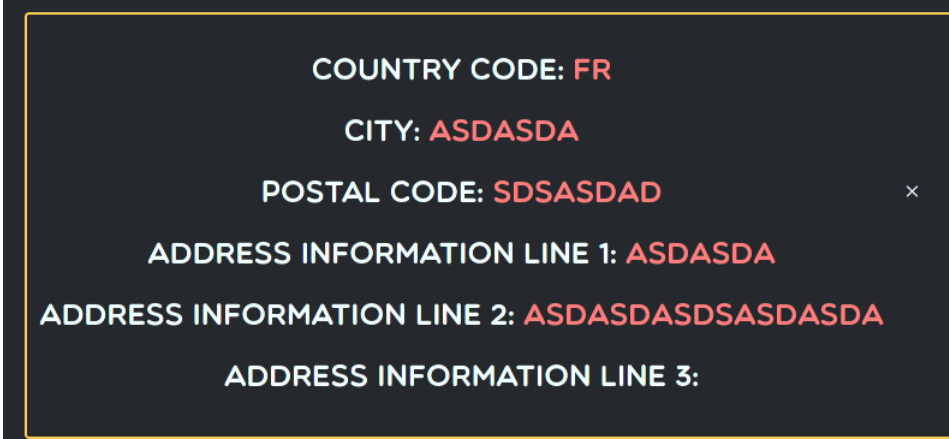
add shipping address

SHIPPING ADDRESS COUNTRY FIELD DOES NOT MEET THE REQUIREMENTS

SHIPPING ADDRESS COUNTRY NEEDS TO BE 2 CHARACTERS THAT REPRESENT THE COUNTRY FOLLOWING THE STANDARD ISO 3166-2

Since our country code doesn't consist of 2 characters, it returns error.

2.9 Add Shipping Address Success Testing



A screenshot of a shipping address form on a dark background, showing successful validation. The form contains several input fields with labels: 'COUNTRY CODE:' with 'FR' entered, 'CITY:' with 'ASDASDA' entered, 'POSTAL CODE:' with 'SDSASDAD' entered, 'ADDRESS INFORMATION LINE 1:' with 'ASDASDA' entered, 'ADDRESS INFORMATION LINE 2:' with 'ASDASDASDSASDASDA' entered, and 'ADDRESS INFORMATION LINE 3:' (empty). A close button (X) is visible on the right side of the form.

COUNTRY CODE: FR

CITY: ASDASDA

POSTAL CODE: SDSASDAD

ADDRESS INFORMATION LINE 1: ASDASDA

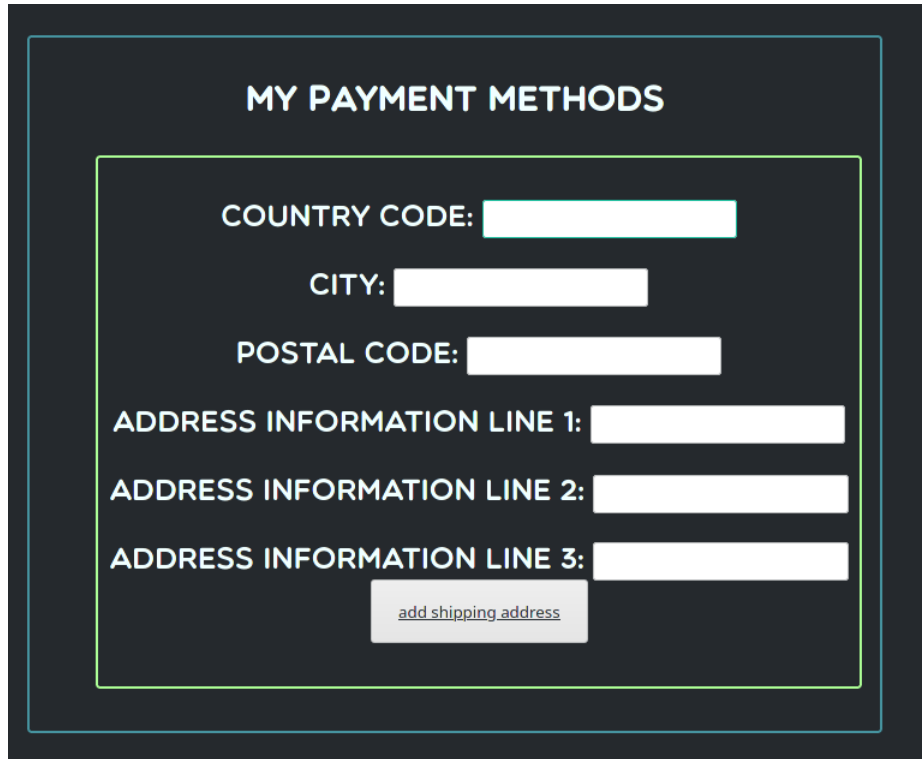
ADDRESS INFORMATION LINE 2: ASDASDASDSASDASDA

ADDRESS INFORMATION LINE 3:

X

Once we respect the criteria from the fields, we are able to upload our payment method.

2.10 Remove Shipping Address Success Testing



The screenshot shows a dark-themed web form titled "MY PAYMENT METHODS". Inside a light blue border, there are several input fields for shipping information: "COUNTRY CODE:", "CITY:", "POSTAL CODE:", "ADDRESS INFORMATION LINE 1:", "ADDRESS INFORMATION LINE 2:", and "ADDRESS INFORMATION LINE 3:". Each field is followed by a white rectangular input box. Below these fields is a button with the text "add shipping address" in a light blue font.

Deleted the created entries without issues.

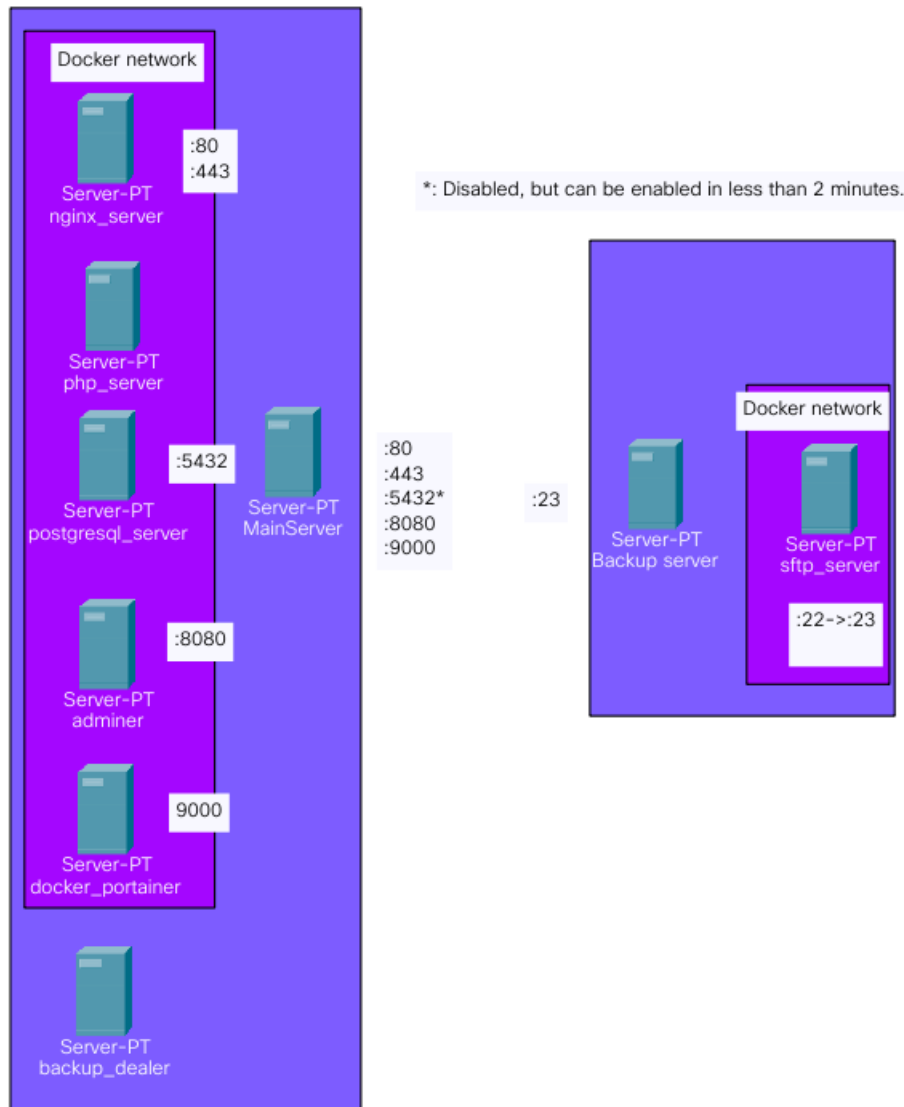
2.11 Cookies Testing

Name	Value	Domain	Path	Expires / Max-Age
session_token	cXqwhU3JtREpzhIYEWiZNarS2HY6voJJaeLF5aO8Y8J3i3qAu66d9KBb3lC	localhost	/	Fri, 28 May 2021 06:58:32 GMT

Checking the cookies, we can see our session token.

3 Docker Configuration

3.1 Docker Network Scheme



3.2 Docker Distribution Explanation

As mentioned previously, the services are distributed in two servers, yet, the second one (the backup server), could be removed and store the backups locally, moving the **sftp** service to the main server, or, through **docker** using a volume or directory binding, but even if it's possible, it's recommended to do the backups in a different server as a security measure in case the main server get compromised.

Taking a look in the main server, we can see that the docker machines are mainly distributed among a common network, while there still an **docker** machine hanging by himself, that's because this **docker** machine is used to generate backups periodically, so doesn't need to have access to another docker networks (unless we wanted to specifically upload the backups inside a docker-machine that resides inside a docker network, but we didn't have the need to publish the ports in the main server, which is not the actual scenario, nor likely to happen).

3.2.1 Nginx

Having the ports 80 and 443 exposed in the server, its configuration forces the use of the port 443, redirecting the connections fo that port, that way in case of having an HTTP petition the client is forced to use a "secure" connection, or at least, a connection that uses SSL encrypting.

Also does use of a docker volume to store the logs.

3.2.2 SFTP

While this container doesn't do nothing by himself, either has ports exposed, it's linked to the nginx server, providing php support to that container, where all will store the backups.

3.2.3 Postgresql

PostgreSQL uses by default the port 5432 to connect to the databases, and while there could be some reasons to have it exposed, but since in this scenario we aren't making use of it, it's better to keep it closed, that way the connections are limited to the docker machines themselves.

The container does use of a volume so store its data.

3.2.4 Adminer

Like previously mentioned, the ports from the database are not exposed, but, in case of needing a simple GUI access to them, this can be arranged by using the adminer docker, which provides a web-database management, and since it's in the same docker network than the postgresql database, we can have access to it without having to expose the ports. The active configuration allows you to access the service using the port 8080.

3.2.5 Portainer

To provide easy access to monitor and control the docker volumes, have decided to implement **Portainer** to the docker network, allowing the users to access to it using the port 9000.

3.2.6 BackupDealer

Like mentioned previously, this machine is used to periodically do backups of the desired volumes or directories, to a local volume or directory, or to a remote server.

3.2.7 SFTP

Observing the schema, we can see that the port 22 is being forwarded to the port 23, and publishing it, that's to avoid a port overlap with the default SSH port configuration. This container stores the backups in a volume named "backups_volume".

3.3 MainServerDockerConfiguration

3.3.1 Main Server DockerCompose

```
1 #docker-compose.yml
2 version: '3.8'
3 services:
4   nginx:
5     image: nginx
6     ports:
7       - "80:80"
8       - "443:443"
9     volumes:
10      - "nginx_logs:/var/log/nginx:rw"
11      - "./config/nginx/web.conf:/etc/nginx/conf.d/web
12        .conf:ro"
13      - "./config/nginx/default.conf:/etc/nginx/conf.d
14        /default.conf:ro"
15      - "${LOCAL_PUBLIC_WEB_PATH}:/var/www:ro"
16      - "./config/cert_ssl:/shared/ssl"
17     deploy:
18       restart_policy:
19         condition: on-failure
20       restart: on-failure
21     depends_on:
22       - php
23     networks:
24       - shop_net
25
26 php:
27   build:
28     context: Dockerfiles
29     dockerfile: php
30   image: filtershop/php
31   deploy:
32     restart_policy:
33       condition: on-failure
34   configs:
35     - uid="${UID:-1000}"
36     - gid="${GID:-1000}"
```



```
35 volumes:
36   - "${LOCAL_PUBLIC_WEB_PATH}:/var/www:rw"
37 environment:
38   DB_LOCATION: "${DB_LOCATION:-postgresql}"
39   HOSTNAME: "${HOSTNAME:-localhost}"
40 links:
41   - "${DB_LOCATION:-postgresql}"
42 networks:
43   - shop_net
44
45 postgresql:
46   build:
47     context: Dockerfiles/postgresql
48   image: filtershop/postgresql
49   deploy:
50     restart_policy:
51       condition: on-failure
52   restart: on-failure
53   environment:
54     POSTGRES_USER: "${DATABASE_USER}"
55     POSTGRES_PASSWORD: "${DATABASE_PASSWORD}"
56     POSTGRES_INITDB_ARGS: "--auth-host=scram-sha-256
57     --auth-local=scram-sha-256"
57     BUILD_DATABASE_LIST: "${BUILD_DATABASE_LIST}"
58   volumes:
59     - type: volume
60       source: "postgresql_volume"
61       target: "/var/lib/postgresql/data"
62       volume:
63         nocopy: true
64   networks:
65     - shop_net
66
67 portainer:
68   image: portainer/portainer-ce
69   ports:
70     - "9000:9000"
71   volumes:
72     - "portainer_data:/data"
73     - "/var/run/docker.sock:/var/run/docker.sock"
```

```
74   deploy:
75     restart_policy:
76       condition: on-failure
77     mode: replicated
78     placement:
79       constraints: [node.role == manager]
80   restart: on-failure
81   networks:
82     - agent_network
83
84   adminer:
85     image: adminer
86     deploy:
87       restart_policy:
88         condition: on-failure
89     restart: on-failure
90     ports:
91       - "8080:8080"
92     links:
93       - "${postgresql_LOCATION:-postgresql}"
94     environment:
95       ADMINER_DEFAULT_SERVER: "${DB_LOCATION:-
96       postgresql}"
97     networks:
98       - shop_net
99
100  networks:
101    shop_net:
102      driver: bridge
103    agent_network:
104      driver: overlay
105      attachable: true
106  volumes:
107    postgresql_volume:
108      external: false
109    portainer_data:
110    nginx_logs:
```

docker-compose.yml that contains the services for the main server.

The main things worth to mention, besides the open ports which are already commented (yet again will be listed here), it's the restart policies and pointing out which services need to be build instead of using a raw image.

1. Networks:

- shop_net:
 - nginx
 - php
 - adminer
 - postgresql
- agent_network:
 - portainer

2. Volumes

- postgresql_volume:
 - postgresql
- portainer_data:
 - portainer
- nginx_logs
 - nginx

The volume used to store the databases, it's mounted with "nocopy", to prevent data being copied data from that volume.

All the dockers have been configured to restart in case of failure, which means that unless they are stopped manually, they will always restart.

3.3.2 PHP Dockerfile

```
1 FROM php:8.0-fpm
2
3 LABEL version='1'
4 LABEL description="Customized Dockerfile for PHP-
   Debian Buster"
5 LABEL authors='filter.oriol@gmail.com'
6 LABEL packages_installed="libpq-dev pdo pgsql
   pdo_pgsql"
7 LABEL packages_enabled="pdo pgsql pdo_pgsql"
8
9 RUN apt-get update && apt-get install --no-install-
   recommends -y libpq-dev \
10   && docker-php-ext-configure pgsql -with-pgsql=/
   usr/local/pgsql \
11   && docker-php-ext-install pdo pgsql pdo_pgsql \
12   && docker-php-ext-enable pdo pgsql pdo_pgsql
```

As listed in the Dockerfile, it's main (and only) function, is to install and enable PDO extension for PHP.

3.3.3 Postgresql Dockerfile

```
1 FROM postgres:13-alpine
2
3 LABEL version='1'
4 LABEL description="Customized Dockerfile for
   Postgresql-Alpine 13 (it just adds a script)"
5 LABEL org.opencontainers.image.authors='filter.
   oriol@gmail.com'
6
7 ADD sources /sources
8 WORKDIR /sources
9 RUN ln -s $(pwd)/build.sh /docker-entrypoint-initdb.
   d/init_01.sh
```

This Dockerfile, instead of installing new packages to the image, it adds a script that will be called in case of not having data in the postgres folder, and also adds an SQL files that once executed, will be deleted from the machine in order to avoid security flaws.

3.3.4 Backup Dealer Docker-Compose

```
1 version: '3.8'
2 services:
3   backup_dealer:
4     build:
5       context: Dockerfiles/backup_client
6     image: tools/backup_client
7     environment:
8       SFTPPORT: ${SFTPPORT}
9       SFTPUSER: ${SFTPUSER}
10      SFTPHOST: ${SFTPHOST}
11      SFTPDIR: ${SFTPDIR}
12      PREFIX: ${PREFIX}
13      VERBOSE: '' # Empty means false
14     volumes:
15       - "./backup_server/user_ssh_keys/
docker_backups.key:/root/.ssh/id_rsa:ro"
16       - "./backup_server/user_ssh_keys/
docker_backups.key.pub:/root/.ssh/id_rsa.pub:ro"
17       - "backup_volume:/master/:ro"
18
19 volumes:
20   backup_volume:
21     name: ${VOLUME_TO_BK}
22     external: true
```

The main and only thing to point out, besides the fact that uses a Dockerfile, is that it's full of variables, this topic will be covered in the deployment section.

3.3.5 Backup Dealer Dockerfile

```
1 FROM postgres:13-alpine
2 LABEL version='1'
3 LABEL description="Customized Dockerfile for backup-
  ftpcli-Alpine 13"
4 LABEL org.opencontainers.image.authors='filter.
  oriol@gmail.com'
5 LABEL packages_installed="rsync tar openssh keychain
  "
6
7 ARG A_SFTPPORT=22
8 ARG A_ORIGINFOLDER=/master
9 ARG A_HOLDERFOLDER=/holder
10 ARG A_DESTINATIONFOLDER=/slave
11 ARG A_PREFIX=backup_
12
13 ENV SFTPUSER ''
14 ENV SFTPHOST ''
15 ENV SFTPDIR ''
16 ENV SFTPDIR ''
17
18 ENV SFTPPORT $A_SFTPPORT
19 ENV ORIGINFOLDER $A_ORIGINFOLDER
20 ENV HOLDERFOLDER $A_HOLDERFOLDER
21 ENV DESTINATIONFOLDER $A_DESTINATIONFOLDER
22 ENV PREFIX $A_PREFIX
23
24 RUN apk add --no-cache openssh rsync keychain tar
25 RUN mkdir $ORIGINFOLDER $HOLDERFOLDER
  $DESTINATIONFOLDER
26
27 RUN printf "/usr/bin/keychain --clear $HOME/.ssh/
  id_rsa \n  source $HOME/.keychain/$HOSTNAME-sh \
  n" > $HOME/.bashrc
28 # Adds .bashrc to our user, which enables us to
  connect using generated keys to avoid passwords
29
30 ADD scripts /scripts
31 WORKDIR /scripts
```

```
32  
33 ENTRYPOINT bash /scripts/main.sh
```

In comparison with the other 2 Dockerfiles which just fulfill one function, this one is more complex.

At the first Part it adds **Build** arguments and afterwards assign those arguments to environment variables to store its value in order to preserve it in future instances.

Also initializes some variables with an empty value, which will require the user to specify them when using this docker.

Once the variables are set up, the Docker will install the packages required in order to accomplish its functions, which in this case the packages required are:

- : **Rsync** to generate backups from directories while keeping the permissions from the files.
- : **Tar** to compress the backups generated in order to reduce the size usage on the servers.
- : **Openssh** in order to be able to connect at the desired sftp server, so the backups can be item stored in a remote server.
- : **Keychain** so the backups can be fully automatized using certificates.

Afterwards creates the default folders (based in the arguments given by the user, or using its default values), once it's done proceeds to add the file ".bashrc" with the content that will allow us to use a keychain avoiding the requirement of a password.

Finally, adds the "scripts" folder to the server.

4 Services Configuration

4.1 Portainer Configuration

As mentioned in the Docker documentation, this container will make use of a volume named "portainer_data", which will store the portainer data.

Also, to provide this container access to the docker management, we need to bind the file "/var/run/docker.sock" to the docker container.

4.2 Adminer Configuration

There is no configuration done to this package.

4.3 PHP Configuration

As mentioned in the docker documentation, this container has been configured to have the PDO extension enabled.

4.4 Nginx Configuration

To improve the security and having a better image through the client, this page uses SSL connections.

Since we have SSL connections, instead of restrict the access to the port 80, we made a simple forward from port 80 to port 443, that way we force the client to use the SSL protocol

```
1 server {  
2     listen 80 default_server;  
3  
4     server_name _;  
5  
6     return 301 https://$host$request_uri;  
7 }
```

To improve the performance, we have enabled HTTP2. As mentioned before, we are using an SSL connection, so we needed to specify the certificate path. Also, to as a security measure, the logs been enabled, which we will store in a docker volume. Finally, since we are using a container without PHP itself, we need to enable a connection, so the Nginx server sends the connections to the PHP container in order to be executed.

```
1 server {
2
3     listen    443 ssl http2;
4
5     ssl_certificate      /shared/ssl/servidor.crt;
6     ssl_certificate_key  /shared/ssl/servidor.key;
7
8     server_name _;
9     root        /var/www/public;
10    index       index.php index.html;
11    access_log   /var/log/nginx/nginx.vhost.access.log;
12    error_log    /var/log/nginx/nginx.vhost.error.log;
13
14    location ~ /\.php$ {
15        try_files $uri =404;
16        fastcgi_split_path_info ^(.+\.php)(/.+)$;
17        fastcgi_pass php:9000;
18        fastcgi_index index.php;
19        include fastcgi_params;
20        fastcgi_param SCRIPT_FILENAME
21        $document_root$fastcgi_script_name;
22        fastcgi_param PATH_INFO $fastcgi_path_info;
23    }
24
25
26 # https://docs.nginx.com/nginx/admin-guide/security-
27   controls/configuring-http-basic-authentication/
```

4.5 PostgreSQL Configuration

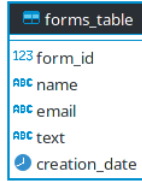
Currently, the database contains 2 databases.

- contact_form
- shop

As a security feature, the login either by local or remote host, have enabled "SCRAM-SHA-256".

4.5.1 Contact database structure

As the name hints, this database is used to store the contact forms received using the contact page from the website. Yet, the database consists of one table.



Column Name	#	Data type	Length	Precision	Scale	Identity	Collation	Not Null	Default
form_id	1	serial						[V]	nextval('forms_table_form_id_seq'::regclass)
name	2	varchar	40	40			default	[V]	
email	3	varchar	254	254			default	[V]	
text	4	text					default	[V]	
creation_date	5	timestamp		29	6			[V]	now()

Column Name	#	Data type	Length	Precision	Scale	Identity	Collation	Not Null	Default	
form_id	1	serial						[V]	nextval('forms_table_form_id_seq'::regclass)	
name	2	varchar	40	40			default	[V]		
email	3	varchar	254	254			default	[V]		
text	4	text					default	[V]		
creation_date	5	timestamp		29	6			[V]	now()	

As the name hints, this database is used to store the contact forms received using the contact page from the website. Yet, the database consists of one table.

It also contains a procedure called "insert_form", which given the next information "(p_name varchar, p_email varchar, p_text text)", checks if the values are valid and in case of not being valid, will rise an error (more information in the Regex and Error List threads).

As a security measure, there been a user implemented using the next query:

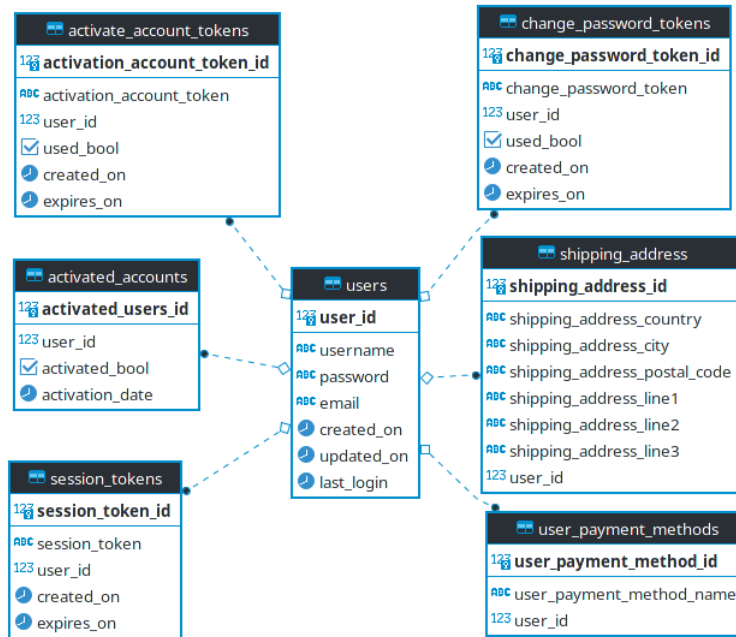
```

1 REVOKE ALL ON DATABASE contact_form FROM PUBLIC;
2 CREATE USER form_user with password 'form_pass';
3 GRANT CONNECT ON DATABASE contact_form to form_user;
4 GRANT EXECUTE on procedure insert_form to form_user;
5 GRANT USAGE on SEQUENCE contact_form.public.
  forms_table_form_id_seq to form_user;
6 GRANT INSERT on TABLE forms_table to form_user;

```

Taking a look at the query, we can observe that the first step is to revoke all the permissions in the database, that way we can ensure that all the users are limited to the options that we specifically gave.

4.5.2 Shop database structure



Taking a look at the graph we can see how all the tables has a relation with the users table, bind by its user id. If we took a look at the queries used to create the tables, we could see how all would get deleted on cascade in case that the user was deleted.

https://github.com/OriolFilter/filterweb/blob/master/Dockerfiles/postgresql/sources/shop_skel.sql

4.5.3 Plugins enabled

```
1 CREATE EXTENSION if not exists pgcrypto;
```

Pgcrypto was enabled in order to hash and salt the passwords, so the use critical information can be stored "securely".

4.5.4 Data structure

- Serial data (`_id` values)
 - Used to store data avoiding collisions while providing an identification point for the entry (used as a Primary Key).
- Username
 - Varchar of 20 length, still, it could be any length, even text, since how postgresql reserves memory, but in order to avoid issues during interactions with other applications, or, to facilitate upgrading to another Database System in the future (if it's needed), it's better to keep it sort of short.

Username are checked by the next function:
"function_check_user_exists" returning a boolean, or procedure
"proc_check_user_exists" raising an error in case the username already exists.
- Password
 - Varchar of 60 length since it's the length of a hashed value.

Passwords are hashed and use salt 8, while not providing any word to salt the value, being this one "randomized".
- Email
 - Varchar of 255 length since according to the next post, emails are actually unable to overcome that length. If in the future that changed, it could be easily modified.

<https://stackoverflow.com/questions/386294/what-is-the-maximum-length-of-a-valid-email-address>
- `created_on` - `updated_on` - `last_login`
 - Stores a time stamp.
- `expires_on`
 - Also stores a time stamp, but this one will also be used to check if the token for something (its function might depend on of which table is stored). By default it's +30 minutes from its creation.

Used to check:

- * Login session is valid
 - * Activate account token is valid
 - * Update/Change password token is valid
- _token values / varchar (200)
 - Used to store randomized tokens. Like in the username or email fields, the value could be easily modified if it was needed in the future.
 - shipping / payment methods data
 - Mainly stored as a varchar, the length might vary depending on of the needs.

4.5.5 Main functions

Like posteriorly will be shown with the error codes, postgresql it's intended to "break" (generate an error), in case something goes wrong. So it's mainly formed by procedures instead of functions, unless it's specifically required.

- function `return_crypted_pass(v_txt varchar)`
 - Returns the hashed + salted (8) password using the function "crypt" (module "pgcrypto").
- procedure `register_user`
 - Registers the user with the values given, before inserting the values, checks that the values are valid.
- proc `login_session_token`
 - Checks the given session token is valid, in case of being valid will enlarge its session 30 minutes from the moment, its used when the user loads a page.

4.6 Data Validation

4.6.1 Regex

```

1 PHP & JS:
2   Registration:
3     username: "[a-zA-Z0-9_+]{6,20}$"
4     password: "[a-zA-Z0-9$%.,?!@+_-]{6,20}"
5     email: "[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z10-9-]+\.[a-zA-Z0-9-]+$"
6
7   Contact form:
8     name: "[\w0-9]{4,40}$"
9     email: "[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z10-9-]+\.[a-zA-Z0-9-]+$"
10    text: "[\w\W]{20,255}$"
11
12  Change password form:
13    token: "[a-zA-Z0-9]{60}$"
14
15  Payment methods:
16    name: "[a-zA-Z0-9]{6,20}$"
17    id: "[0-9]+$"
18
19  Shipping address:
20    Country code: "[a-zA-Z]{2}$/"
21    id: "[0-9]+$/"
22    postal code & city: "[\w\W]+$/" #
23
24  Checks not empty
25    address line 1: "[\w\W]{5,200}$/"
26    address line 2 & 3 : "[\w\W]*$/" # A
27    lie , doesn't checks nothing
28
29  POSTGRESQL:
30    Contact form:
31      name: "[\w0-9]{4,40}$"
32      email: "[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z10-9-]+\.[a-zA-Z0-9-]+$"

```


4.6.2 Javascript

Uses Regex and default java functions in order to check if the fields are valid, yet it doesn't really look at specific stuff.

Uses Ajax when receiving a response from the server, so can format a response for the user.

4.6.3 PostgreSQL

Uses functions and procedures in order to detect errors, alongside with Regex.

4.6.4 PHP

Uses functions and procedures in order to detect errors, alongside with Regex. PHP uses the errors captured to form a response for the client.

4.7 Error Thread

4.7.1 Notes

The errors are supposed to be handled by something, but not by postgresql (unless it's something specific), since there is no need of rollback/commit, due postgresql nature, that controls this automatically.

PHP and JavaScript uses these errors to format responses for the client based in the code received.

4.7.2 Error Codes

```
1 php & js :  
2     '0': 'Unknown error',  
3  
4     '1': 'Success',  
5  
6     '2': 'Missing field(s)',  
7     '2.1': 'Username field is missing',  
8     '2.2': 'Password field is missing',  
9     '2.3': 'Email field is missing',  
10    '2.4': 'Repeat password field is missing',  
11    '2.5': 'Repeat email field is missing',
```

```
12      '2.6': 'Name field is missing',
13      '2.7': 'Text field is missing',
14      '2.8': 'Payment method name field is missing',
15      '2.9': 'Payment method info field is missing',
16      '2.10': 'Payment method id field is missing',
17      '2.11': 'Shipping address fields topic',
18      '2.11.1': 'Shipping address country field is missing',
19      '2.11.2': 'Shipping address city field is missing',
20      '2.11.3': 'Shipping address postal code field is missing',
21      '2.11.4': 'Shipping address line 1 field is missing',
22      '2.11.5': 'Shipping address id field is missing',
23
24      '3': 'Requirements not achieved',
25      '3.1': 'Username does not meet the requirements',
26      '3.2': 'Password does not meet the requirements',
27      '3.3': 'Email does not meet the requirements',
28      '3.4': 'Name does not meet the requirements',
29      '3.5': 'Text does not meet the requirements',
30      '3.6': 'Payment method name does not meet the requirements',
31      '3.7': 'Payment method info does not meet the requirements',
32      '3.8': 'Payment method id does not meet the requirements', # It's a numeric value only
33      '3.9': 'Shipping address fields topic',
34      '3.9.1': 'Shipping address country field does not meet the requirements',
```

```
35     '3.9.2': 'Shipping address city field does
not meet the requirements',
36     '3.9.3': 'Shipping address postal code field
does not meet the requirements',
37     '3.9.4': 'Shipping address line 1 field does
not meet the requirements',
38     '3.9.5': 'Shipping address line 2 field does
not meet the requirements',
39     '3.9.6': 'Shipping address line 3 field does
not meet the requirements',
40     '3.9.7': 'Shipping address id does not meet
the requirements', # It's a numeric value
only
41
42     '4': 'Field matching',
43     '4.1': 'Passwords don\'t match',
44     '4.2': 'Emails don\'t match',
45
46     '5': 'Client-Server errors',
47     '5.1': 'There was a unknown error sending
the data, please, try again bit later, if
this error is consistent please contact an
administrator.',
48     '5.2': 'Server under maintenance, please,
try again bit later.'
49
50     '6': 'Database side error'
51     '6.1': 'Data Insert errors',
52     '6.1.1': 'Username is already exists',
53     '6.1.2': 'Email is already exists',
54
55     '6.2': 'Data Select errors',
56     '6.2.1': 'Username not found',
57     '6.2.2': 'User_id not found',
58     '6.2.3': 'Email not found',
59     '6.2.4': 'Token not found',
60     '6.2.5': 'Payment method not found',
61     '6.2.6': 'Shipping address not found',
62
63     '6.3': 'Tokens',
```

```
64      '6.3.1': 'Token not valid',
65      '6.3.2': 'Token already used',
66      '6.3.3': 'Token expired',
67      '6.3.4': 'Token is null or empty',
68
69      '6.4': 'Database connection error',
70      '6.4.1': 'Error communicating to database',
71      '6.4.2': 'Wrong credentials connecting to
database',
72      '6.4.3': 'The user don\'t has permission for
the requested action(s)',
73
74      '6.5': 'Functions error',
75      '6.5.1': 'Error generating token',
76
77
78      '7': 'Account related issues',
79      '7.1': 'The account is not activated',
80      '7.2': 'The account is already activated',
81      '7.3': 'The account been banned',
82
83      '8': 'PHP mailer issues',
84      '8.1': 'Email couldn\'t be send',
85      '8.2': 'Email address is missing',
86      '8.3': 'Body is missing',
87      '8.4': 'Subject is missing',
88
89      '9': 'Invalid Credentials',
90
91      '10': 'Product stuff',
92
93      '11': 'User conf stuff',
94
95      '11.2': 'Not valid payment method data'
96
97      '12': 'Order Stuff'
98 postgresql:
99 - 'Due postgresql not being able to use the
same syntax as php and since the error codes
seems easy to read using the syntax already
```

```
done, it's been decided to leave the php and
js codes as they, while using a similar (but
valid) syntax for postgresql.'
```

100 - 'P0000'

101 - 'P + first number + second number + last
number'

102 examples:

103 - '7 :P7000'

104 - '8.3 :P8300'

105 - '6.4.4 :P6404'

106 - '2.1 :P2100'

107 - '2.10 :P2010'

5 Web Page Documentation

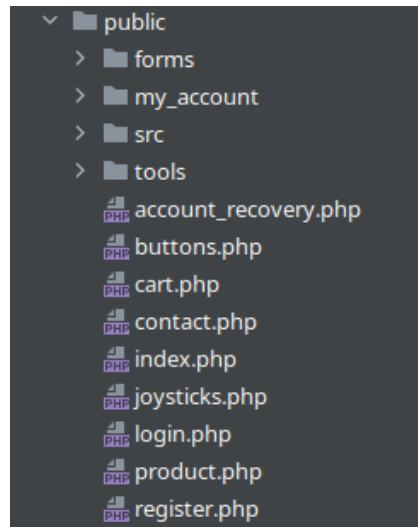
5.1 File organization

In the main folder we can find the next directories:

- public
 - In this directory is stored the files available to the user.
- private
 - In this directory will be stored the files that the user shouldn't have access.

5.2 Public Directory Content

Point out that the content of the folder is named `.index`, to make it more visually appealing, yet this could be easily avoided by configuring Nginx, all the php files could follow this structure, it's just a matter of investing 5 minutes (as much) in doing it.

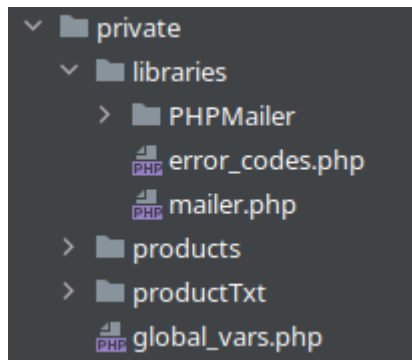


In the public folder we can find the next directories:

- forms
 - Forms used to store the **API** entry points, used by the client.
- my_account
 - Public pages that the user can access in order to manage its account.
- tools
 - Pages that fulfill a specific function and are often used.
- src
 - Used to store files that the user might require, mainly used to store:
 - * Images
 - * JavaScript
 - * Favicon

5.3 Private Directory Content

The file **global_vars.php should be renamed "manager.php" or something like it in the future and be moved to the "libraries" folder.*



In the private folder we can find the next directories:

- Used to store files that the user might require, mainly used to store:
 - libraries directory
 - * Contains the libraries of:
 - PHPMailer
 - Error_Codes
 - Products folders
 - * Used to store information about products in order to showcase it in the demo, this could be easily replaced by importing the values to the database.

5.4 Code Structure

Depending of the function of the file, the structure might vary, so far we can find 3 general structures:

- Library/package
- Public page
- API

5.4.1 Library/package code structure

Meanwhile there is no code structure, the files are built trying to keep together the related.

5.4.2 Public page code structure

In this case we have a structure, yet, if it's necessary, the structure might vary according to the needs.

```
1 <?php
2     require_once(global_vars.php);
3     $global_vars = new page_vars();
4     $global_vars->return_header();
5     /* Print/do stuff */
6     $global_vars->return_footer();
7 ?>
```

5.4.3 API code structure

The idea behind its structure, is attempt to do something, and in case of going wrong raises an Exception, then format a class that will be used as json output.

```
1 <?php
2 #/public/forms/account_management/
   add_shipping_address/index.php
3 /* Global try catch */
4 try {
5     require_once '/var/www/private/global_vars.php';
6     /* vars */
7     $page_vars= new page_vars;
8     $page_vars->import_errors();
9
10    /* DB*/
11    $db_manager = new shop_db_manager();
12    /* Json */
13    $hotashi = new hotashi();
14    $json_obj = new json_response();
15
16    /* Main */
17
18    /* Get Vars */
19    $hotashi->get_add_shipping_address_vars();
20    /* Database connection*/
21    $db_manager->add_shipping_address($hotashi);
22    /* success */
23    $json_obj->status='success';
24    $json_obj->status_code=1;
25 }
26 catch (DefinedErrors $e ) {
27     $e->formatJson($json_obj);
28 }
29 catch (Exception $e) {
30     $json_obj->status = 'failed';
31     $json_obj->error['code'] = 0;
32     $json_obj->error['message'] = 'Unknown error';
33     $json_obj->status_code = 0;
```

```
34 }  
35 finally {  
36     echo json_encode($json_obj);  
37 }  
38 ?>
```

5.4.4 Main Class listing

The current main classes created are:

- `json_response`
 - Used to store the status from the API call, and return its values to the client.
- `page_vars`
 - Mainly used to store variables to be used among all the pages, and to return a footer and header that its contents might vary depending on if the user is logged or not.
 - Also used to import the libraries if the corresponding function is used, that way the code is more consistent.
- `Hotashi`
 - Responsible of getting the user cookies, posts, gets and validate its values, capable of rising errors if needed.
 - Has some functions that make use of the class **`db_manager`**, for example has a function that checks if the user session token is valid.
- `db_user`
 - Each entry stores information about the connection to the database, which includes the username, password, host among other data.
 - Extends to **`db_user_contact_manager`** and **`db_user_shop_manager`**
- `db_manager`
 - Stores functions that contain calls to the database, also, **`__on_construct`** makes use of the **`db_user`** class to generate a PDO connection.
 - Extends to **`contact_db_manager`** and **`shop_db_manager`**
 - An example of function that requires a PDO usage might be:

```
1 public function add_shipping_address(hotashi
    $hotashi){
2 try {
3     $stmt = $this->dbconn->prepare(query: '
        call
        proc_add_shipping_address_from_stoken
        (?, ?, ?, ?, ?, ?, ?); ');
4     $stmt->execute(array($hotashi->stoken,
        $hotashi->sa_country, $hotashi->sa_city,
        $hotashi->sa_pcode, $hotashi->sa_add1,
        $hotashi->sa_add2, $hotashi->sa_add3));
5 }
6 catch (PDOException $e){
7     $this->error_manager->pg_error_handler(
        $e->getCode());
8     }
9 }
10
```

- error_manager
 - Used to store a function called "**db_error_handler**", which depending on of the database code given, will raise an error, which afterwards will be caught and mostly format the json_object to return a message to the client.
- CustomError
 - Used as a template to extend other errors, providing a function called "**format_json**", that will format the json_object with its inner values, consistently generating error messages for the client.
- Interface DefinedErrors
 - Used to group all the created errors, in order to catch them together and improve consistency and reliability, making simple generating error messages to the user among working around errors.
- mailer

- Responsible of sending emails through PHPMailer libraries.
- `mailer_info`
 - This class is the responsible for formatting the class mailer, containing functions that will generate
 - a different message depending on the desired message.

6 Services Deployment

6.1 System requirements

To install and configure the packages, a user with root access might be required, in case your system doesn't have the package "**sudo**", and your user be configured with it, consider swapping to the Root user.

- Access to internet
- Git
- Docker
- Docker-compose

6.1.1 Git installation

apt

```
1 sudo apt-get update && sudo apt-get install git
```

pacman

```
1 sudo pacman -Syu && sudo pacman -S git
```

apk

```
1 sudo apk update && sudo apk add --no-cache git
```

6.1.2 Docker installation

apt

```
1 sudo apt-get update && sudo apt-get install docker-  
ce docker-ce-cli containerd.io
```

pacman

```
1 sudo pacman -Ss && sudo pacman -S docker
```

apk

```
1 sudo apk update && sudo apk add --no-cache docker
```

6.1.3 Docker configuration - allow user to use docker

```
1 sudo usermod -a -G docker your_user
```

6.1.4 Docker configuration - enable docker on boot service

```
1 sudo service enable docker
```

```
1 sudo systemctl start
```

rc-update

```
1 sudo run rc-update add docker boot
```

6.1.5 Docker-Compose installation

apt

```
1 sudo apt-get update && sudo apt-get install python3
2 sudo curl -L "https://github.com/docker/compose/
  releases/download/1.29.2/docker-compose-$(uname -
  s)-$(uname -m)" -o /usr/local/bin/docker-compose
3 sudo chmod +x /usr/local/bin/docker-compose
```

pacman

```
1 sudo pacman -Ss && sudo pacman -S python3
2 sudo curl -L "https://github.com/docker/compose/
  releases/download/1.29.2/docker-compose-$(uname -
  s)-$(uname -m)" -o /usr/local/bin/docker-compose
3 sudo chmod +x /usr/local/bin/docker-compose
```


apk

```
1 sudo apk update && sudo apk add --no-cache py-pip
  python3-dev libffi-dev openssl-dev gcc libc-dev
  rust cargo make
2 sudo curl -L "https://github.com/docker/compose/
  releases/download/1.29.2/docker-compose-$(uname -
  s)-$(uname -m)" -o /usr/local/bin/docker-compose
3 sudo chmod +x /usr/local/bin/docker-compose
```

6.1.6 Repository installation

The only step is to clone the repository.

```
git clone https://github.com/OriolFilter/filterweb/
```

6.2 Repository deployment minimal customization

6.2.1 Main server deployment minimal customization

```
1 # .env
2 # PHP
3 UID=1000
4 GID=1000
5 HOSTNAME="localhost"
6
7 # Nginx
8 LOCAL_PUBLIC_WEB_PATH="./config/filterweb/"
9
10 # PostgresDB
11 BUILD_DATABASE_LIST="shop contact_form"
12 DATABASE_PASSWORD="test"
13 DATABASE_USER="test"
```

The only to edit it's the **hostname**, since we need to replace it for our public/local **IP** or our **Domain Name**, in case we just wanted to test it, we could use "**localhost**".

6.2.2 Backups client deployment minimal customization

```

1 #./bkcli_env_folder/env_bk_pg
2 SFTPPORT="23"
3 SFTPUSER="docker_backups"
4 SFTPHOST="backup_server.net"
5 SFTPPDIR="/backups/postgres"
6 PREFIX="postgres_db_"
7 VOLUME_TO_BK="filterweb_postgresql_volume"

```

```

1 #./bkcli_env_folder/env_bk_nx_logs
2 SFTPPORT="23"
3 SFTPUSER="docker_backups"
4 SFTPHOST="backup_server.net"
5 SFTPPDIR="/backups/logs"
6 PREFIX="nginx_logs_"
7 VOLUME_TO_BK="filterweb_postgresql_volume"

```

First we need to specify the **SFTHOST** for the **SFTP** server address, either the **IP** or the **Domain Name**, in this case, we can't use "localhost", in case we hosted the **SFTP** server in the same machine than the **backup_dealer**, we need to specify our local **IP**.

6.2.3 Cron periodical backups minimal customization

Once we have modified both **.env_bk_pg** and **.env_bk_nx_logs**, we need to execute the next command in order to do a backup every day.

```

1 echo "* * */1 * * $USER docker-compose -f $(pwd)/
  backup_dealer-compose.yml --env-file $(pwd)/
  bkcli_env_folder/.env_bk_pg up" | sudo tee -a /
  etc/cron.d/docker_backups
2 echo "* * */1 * * $USER docker-compose -f $(pwd)/
  backup_dealer-compose.yml --env-file $(pwd)/
  bkcli_env_folder/.env_bk_ng_logs up" | sudo tee
  -a /etc/cron.d/docker_backups
3

```

6.2.4 Backups Server deployment minimal customization

In case that we desire to host the server **SFTP** during the testing, we can skip this step.

The next step is copy the folder "backup_server" to the device that we want to use as a backup server, this step isn't necessary if we already have a **SFTP** server, or we desire to do local backups(which isn't recommendable).

To copy the folder to a remote server we can use the command:

```
1 scp -r ./backup_server <user>@<host>:~
```

6.3 Repository deployment booting services

6.3.1 Main Server Service Booting

This can simply be done by executing the next command:

```
1 docker-compose build && docker-compose up
```

6.3.2 Backup (Remote) Server Service Booting

Reminder that the server needs to fulfill the requirements. This can simply be done by executing the next command:

```
1 ssh <user>@<host> docker-compose -f backup_server/  
  docker-compose.yml up -d
```

6.3.3 Backup (Local) Server Service Booting

This can simply be done by executing the next command:

```
1 docker-compose -f backup_server/docker-compose.yml  
  up -d
```

6.4 Repository deployment further customization

6.4.1 Custom SSL Keys

In order to change the current certificates, you need to replace them for yours, the location of each certificate it's in the **docker-compose** document

This applies for the **NGINX** and **SFTP** containers.

6.4.2 Custom Keychains

Requirements: Keychain package installed.

Once the requirements are fulfilled. We can proceed to generate the keychains.

```
1 ssh-keygen -t rsa -b 2048 -f ./id_rsa
```

```
2
```

Once we have created the keys, it's time to replace them, remember that the files need to be replaced in the local system AND the server in case the SFTP server is remote.

In case of modifying the path to the keys in the docker-compose file, remember that the **backup dealer** also makes use of these keys in order to login without need of password, so in case of changing its path, you need to do it in both files.

6.4.3 Adding more folders or modifying the current docker_backup user

In order to do this, we need to edit the file "*users.conf*", located in the same folder as the **SFTP** server.

```
1 docker_backups:::backups,backups/postgres,backups/
  logs
```

The file structure is:

```
1 USER:hashed_password:e:UID:GUID:folder_list
```

In our case, since we don't need the user to have a password, since we are using keychains, we can just skip that field.

```
1 USER::UID:GUID:folder_list
```

```
2
```

6.4.4 Deploying Custom Databases

In order to use/create our custom databases, we have 2 options, using a volume or directory that already has them created, or using the script that triggers if the directory `"/data"` is empty, happen on mounting an empty volume, or the first time the container starts.

Script/Building explanation

This lines determine which files will search for:

```
1 declare -a FORMAT_ARR=( '_skel' '_users' '_val' )
2 declare -a DATABASE_ARR=( $( awk '{ gsub(","," ") ;
   gsub("  "," "); gsub(" ", "\n"); print}' <<< "
   $BUILD_DATABASE_LIST" ) );
3
```

First we have an array declared with 3 values, this are the suffix that will be looked for when searching the SQL files. In the second line, we receive the variable **BUILD_DATABASE_LIST**, which will be normalized, first, comas transform in spaces, afterwards, double spaces transformed in single spaces, and finally the spaces left will be transformed in newlines, which will be used to create new entries in the array.

Once the arrays are generated, the script will attempt to generate a database for every entry in the **DATABASE_ARR**, keep in mind that it's the values given by the user from the docker-compose configuration.

```
1 for key in "${!FORMAT_ARR[@]}"
2 do
3     sqlfile="${WORKDIR}/${database}${FORMAT_ARR[$key]}.sql"
4
```

Basically, will try to find an element that match the name **GIVEN_DATABASE_NAME+SUFFIX** from the array, which means that for every entry in the **DATABASE_ARR**, will search a **"DATABASE_ARR_skel"**, **"DATABASE_ARR_users"**, **"DATABASE_ARR_val"**, executed in this order, that's very important to keep in mind in case we make use of multiple SQL files, since the order of their execute might alter the result.

Configuration

The first step is to change the ".env" file with the desired database names, keep mind that the elements can be separated by spaces or commas.

```
1 #.env
2 # PHP
3 UID=1000
4 GID=1000
5 HOSTNAME="localhost"
6
7 # Nginx
8 LOCAL_PUBLIC_WEB_PATH="./config/filterweb/"
9
10 # PostgresDB
11 BUILD_DATABASE_LIST="shop contact_form"
12 DATABASE_PASSWORD="test"
13 DATABASE_USER="test"
```

Once the database names have been renamed, it's time to replace the current **SQL** files in the folder *Dockerfiles/postgresql/sources*.

For example, if we want to create the database "new_database", the files must be named "new_database_skel.sql", "new_database_users.sql", "new_database_val.sql".

6.4.5 Change the current email sender for your own one

To change the current email sender, we must go to the file "**mailer.php**", situated at "`./config/filterweb/private/libraries`". Once we are in the file, we just need to change the next values to the desired ones: *If you are using a Gmail account, access from untrusted applications needs to be enabled*

```
1      $mail_server='filter.web.asix@gmail.com';  
2      $mail_server_pass='ITB2019015';  
3
```

In case you don't use gmail, you might need to modify:

```
1      $mail->Host          = 'smtp.gmail.com';  
2      $mail->Port          = 587;  
3
```

And finally, to not keep being sending messages as "Arcade Shop", replace this with ur desired name:

```
1      $mail->setFrom($mail_server, 'Arcade Shop');  
2
```