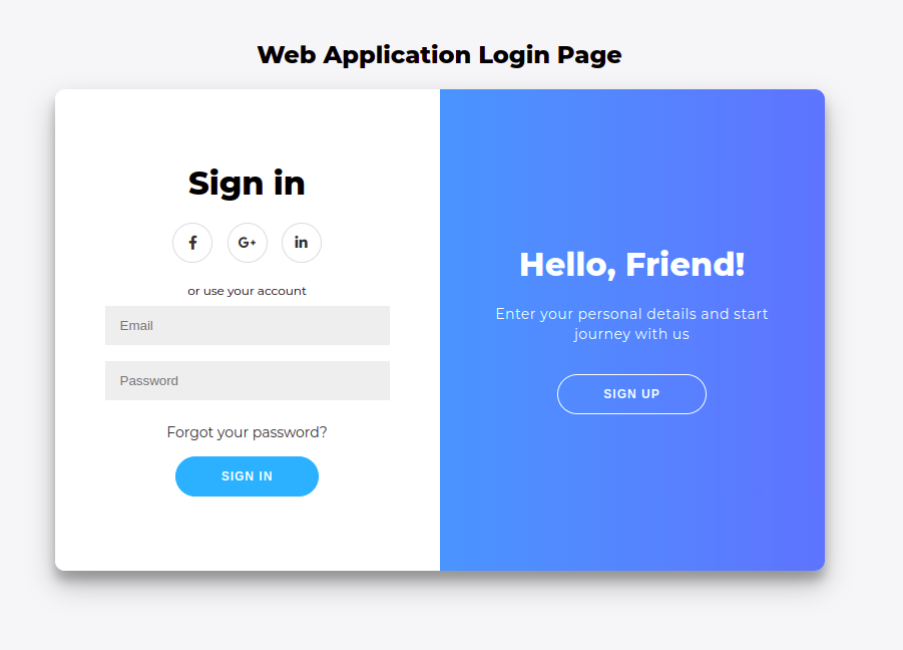
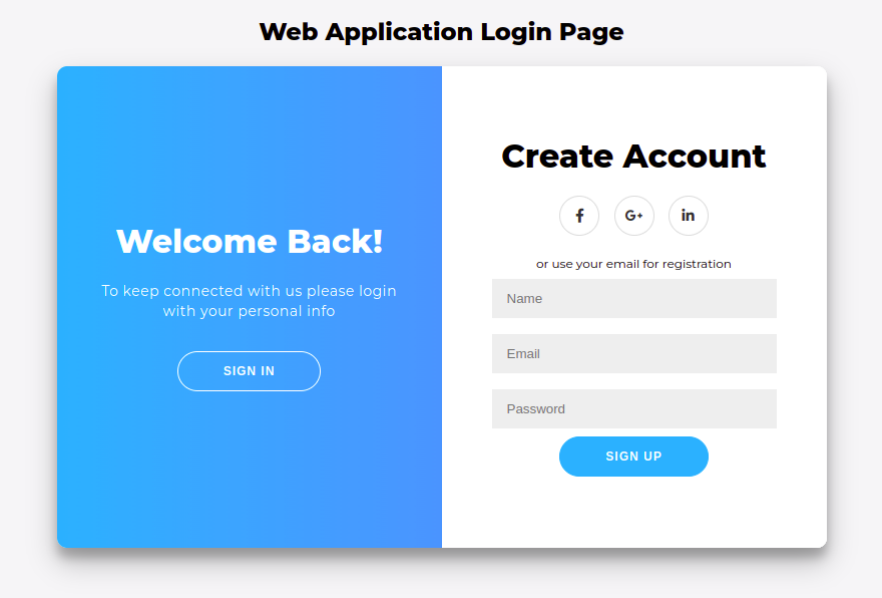
Lab 2: Web Applications Security

> Igor Mpore
>
> BS19-CS01
>
> i.mpore@innopolis.university

Part 1: Setting up a Web Application with SSL certificate and a database.

I created a Login and Sign up screens in HTML,CSS and NodeJS to use as my web application to use throughout this lab called WebToy. The project is on my github and here are the two screenshots of the pages.





After this, I created root certificate for my webapp (WebToy) using the following commands.

Creating SSL Cerficate

1. Creating a root certificate:

```
openssl genrsa -des3 -out rootCA.key 4096
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.crt
```

```
migor@migorHP:~$ openssl genrsa -des3 -out rootCA.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................................++++
............++++
e is 65537 (0x010001)
Enter pass phrase for rootCA.key:
Verifying - Enter pass phrase for rootCA.key:
```

```
migor@migorHP:~$ openssl req -x509 -new -nodes -key rootCA.key -sha256
 -days 1024 -out rootCA.crt
Enter pass phrase for rootCA.key:
You are about to be asked to enter information that will be incorporat
ed
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:Russia
string is too long, it needs to be no more than 2 bytes long
Country Name (2 letter code) [AU]:RU
State or Province Name (full name) [Some-State]:Kazan
Locality Name (eg, city) []:Innopolis
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Big Company
Organizational Unit Name (eg, section) []:Security Department
Common Name (e.g. server FQDN or YOUR name) []:mikes
Email Address []:sirmi71@gmail.com
```

2. Generating my root certificate

```
openssl genrsa -out webtoy.com.key 2048
```

```
migor@migorHP:~$ openssl genrsa -out webtoy.com.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................+++++
.........................+++++
e is 65537 (0x010001)
```

3. Creating a Certificate Signing Requests:

```
openssl req -key webtoy.com.key -new -out webtoy.com.csr
```

```
migor@migorHP:~$ openssl req -key webtoy.com.key -new -out webtoy.com.
csr
You are about to be asked to enter information that will be incorporat
ed
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:RU
State or Province Name (full name) [Some-State]:Kazan
Locality Name (eg, city) []:Innopolis
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Big Company
Organizational Unit Name (eg, section) []:Security Department
Common Name (e.g. server FQDN or YOUR name) []:Mikes
Email Address []:sirmi71@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:admin
An optional company name []:Bigger Company
```

4. Creating self signed certificate and lastly I converted the certificate to DER:

```
openssl x509 -req -in webtoy.com.csr -CA rootCA.crt -CAkey rootCA.key -
CAcreateserial -out webtoy.com.crt -days 365 -sha256

openssl x509 -in webtoy.com.crt -outform der -out webtoy.com.der
```

```
migor@migorHP:~$ openssl x509 -req -in webtoy.com.csr -CA rootCA.crt -
CAkey rootCA.key -CAcreateserial -out webtoy.com.crt -days 365 -sha256
Signature ok
subject=C = RU, ST = Kazan, L = Innopolis, O = Big Company, OU = Secur
ity Department, CN = Mikes, emailAddress = sirmi71@gmail.com
Getting CA Private Key
Enter pass phrase for rootCA.key:
```

Setting up the database and it's basic authentication.

We'll be using PostgreSQL and PGAdmin for our database. Link

```
#installing PostgreSQL
sudo apt install postgresql postgresql-contrib

#Swithing user, connecting to PostgreSQL and checking user name
sudo -u postgres psql
```

```
migor@migorHP:~$ sudo -u postgres psql
psql (12.9 (Ubuntu 12.9-0ubuntu0.20.04.1))
Type "help" for help.

postgres=# \conninfo
You are connected to database "postgres" as user "postgres" via socket
 in "/var/run/postgresql" at port "5432".
```

Now, let's add **a password to "postgres" to secure the database.**

```
# Adding password (mypass)
sudo passwd postgres
```

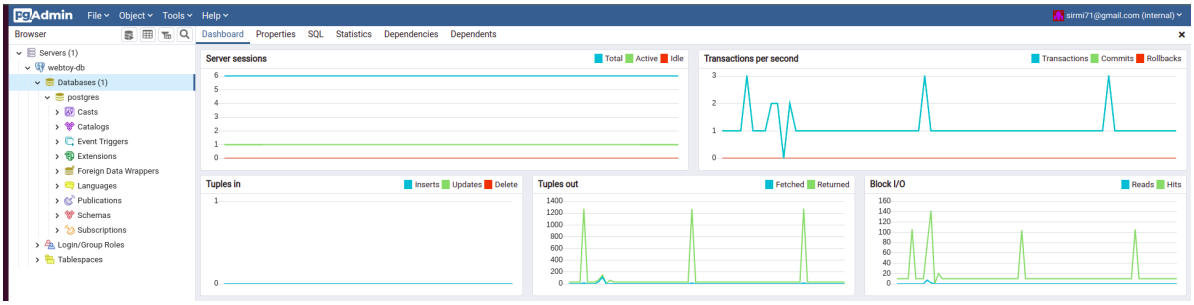
Then, let's switch to "postgres" account and also add a password to PostgreSQL. After, we'll restart the service.

```
# Switching to postgress
su - postgres
# Changing password
psql -c "ALTER USER postgres WITH PASSWORD 'mypass';"
#Exiting
exit

#Restarting the service
sudo systemctl restart postgresql
```

After installing PGAdmin too, I also added a security layer to the database that requires email and password to login. Refence Tutorial from (link):

```
sudo /usr/pgadmin4/bin/setup-web.sh
```



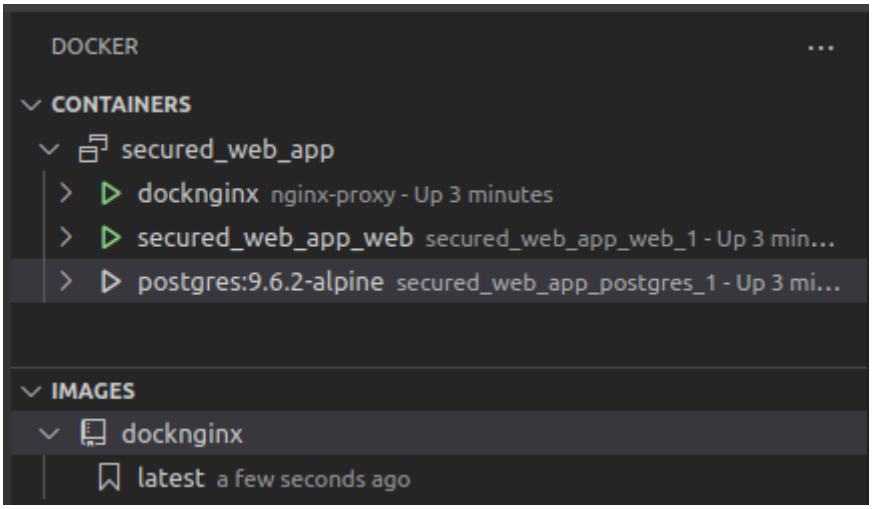
After establishing the database connection to the app, I tested it by inserting a row.

```
migor@migorHP:~/Documents/Secured_web_app$ node server.js
Connected successfuly
null Result {
  command: 'INSERT',
  rowCount: 1,
  oid: 0,
  rows: [],
  fields: [],
```

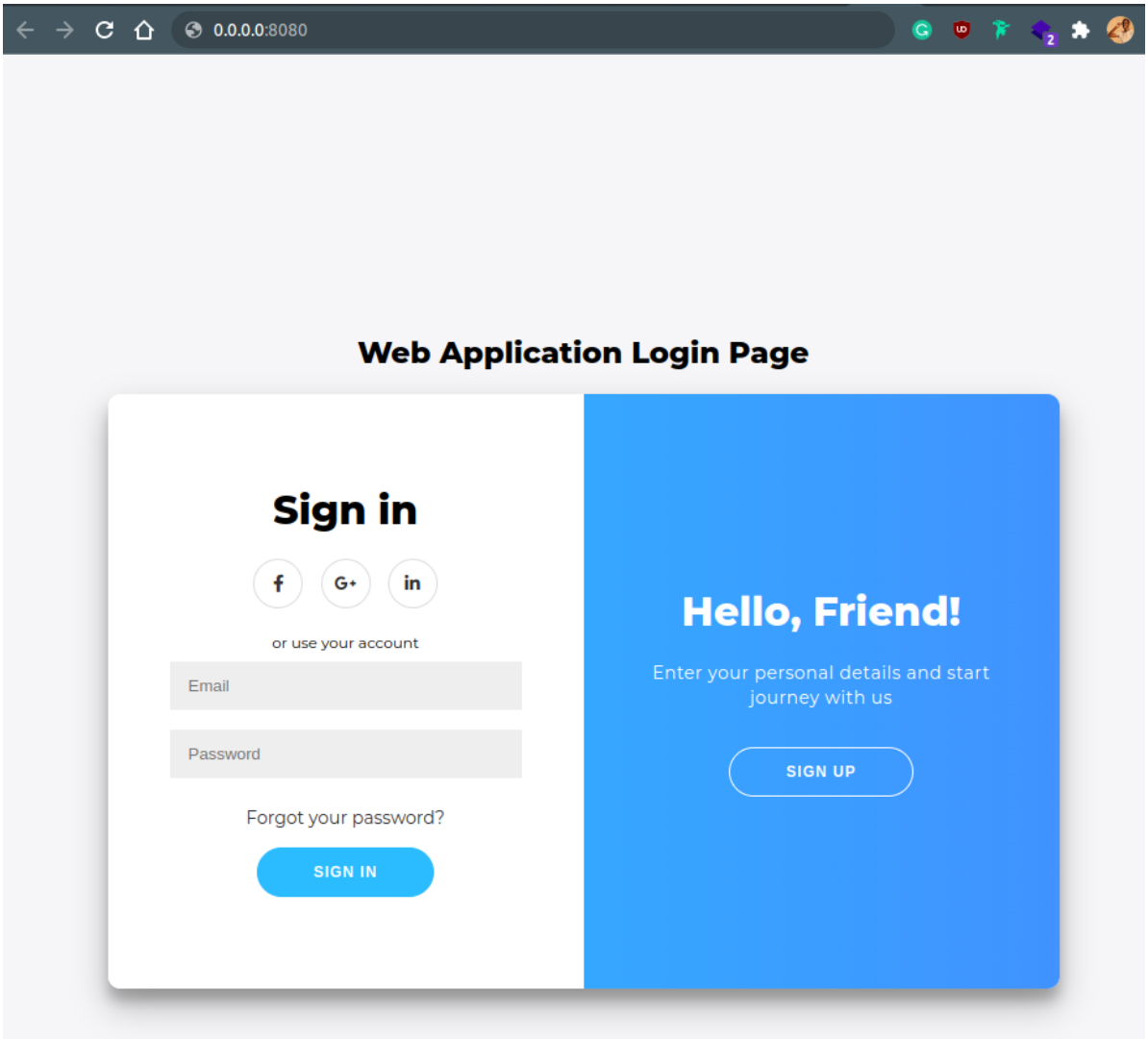
Data Output | Explain | Messages | Notifications

	user_id [PK] integer	email text	name text	password text
1	0	hi@gmail.com	Tom Hikes	stronpass

From this, I can confirm that the connection is correct and everything is fine. After this, we created a docker image for this web app. The first container is for **the database** and the other one is **for the application.** We also have another running container of nginx server proxy. The tutorial used can be found here



Now it's running on port 80 of nginx specified in the docker_compose.yml.




```
nginx:
    restart: always
    build:
        context: .
    image: dockngnix
    container_name: nginx-proxy
    hostname: nginx
    ports:
        - "80:80"
    depends_on:
        - web
    networks:
        - secured_web_app_default
    tty: true
```

Part 2: Preventing from brute-force attack

1. Screenshot with ab requests before enabling limit_req on nginx

```
Concurrency Level:      100
Time taken for tests:   2.898 seconds
Complete requests:      1000
Failed requests:        0
Non-2xx responses:      1000
Total transferred:      703000 bytes
HTML transferred:       220000 bytes
Requests per second:    345.09 [#/sec] (mean)
Time per request:       289.777 [ms] (mean)
Time per request:       2.898 [ms] (mean, across all concurrent requests)
Transfer rate:          236.91 [Kbytes/sec] received
```

2. Screenshot with ab after requests

After enabling the command to limit the number of departures, here's what happens: The failed requests became 920

```
limit_req_zone $binary_remote_addr zone=limit:10m rate=50r/s
```

```
Concurrency Level:      100
Time taken for tests:   9.575 seconds
Complete requests:      940
Failed requests:        920
   (Connect: 0, Receive: 0, Length: 920, Exceptions: 0)
Non-2xx responses:      940
Total transferred:      211380 bytes
HTML transferred:       111200 bytes
Requests per second:    98.17 [#/sec] (mean)
Time per request:       1018.611 [ms] (mean)
Time per request:       10.186 [ms] (mean, across all concurrent requests)
Transfer rate:          21.56 [Kbytes/sec] received
```

OWASP Top 10 in your own words.

This is a well-know list of vulnerabilities which are collect together each 4 years to help web developers and security experts to make sure they create secured web application. The list is organised by a company called OWASP