

Ejercicio final de Docker

- Actividad 3 -

Nombre: Yumara Vallejo Vallejo

Curso: 2º Desarrollo de Aplicaciones Web

Año: 2024 / 2025

Índice

Introducción	3
Descripción de Componentes y Servicios	3
Archivos de Configuración y Dockerfile	4
Pruebas de Funcionamiento y Rendimiento	11

Introducción

Se va a desarrollar una aplicación web que comunique con microservicios usando Docker, se basará en React desplegada en Apache con dos microservicios PHP y una base de datos MySQL. Todo esto deberá de hacerse usando Docker. Para este ejercicio necesitaremos tener ya activados los módulos ssl, proxy y proxy_http

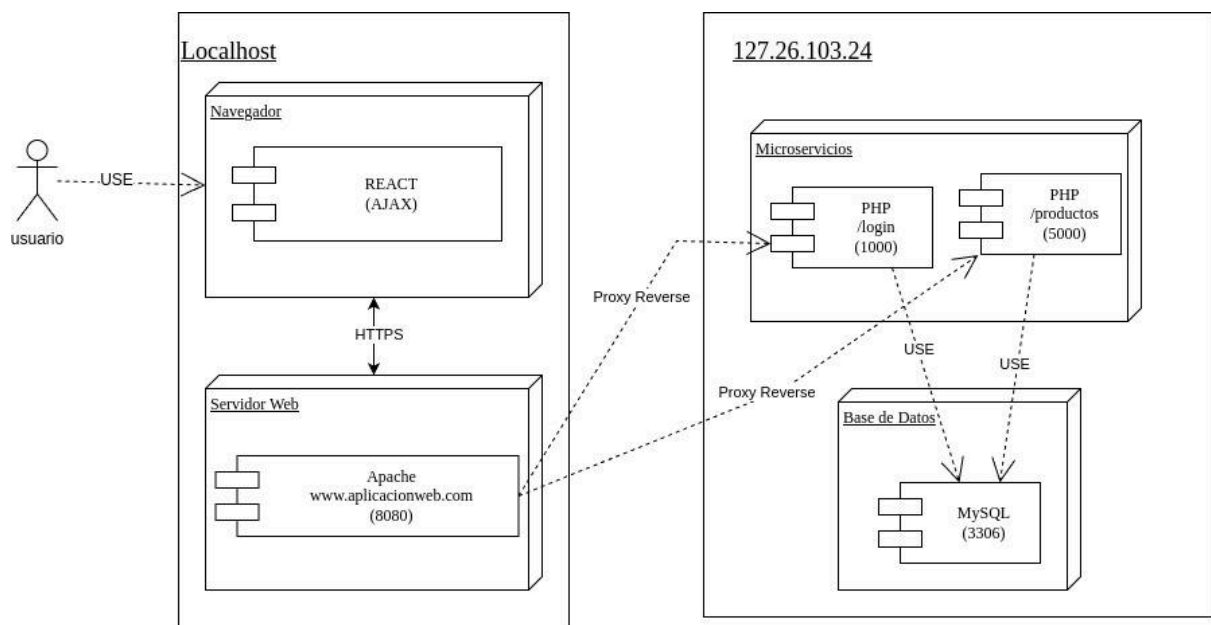
Descripción de Componentes y Servicios

React: Interfaz de usuario que se comunica con los microservicios a través de los endpoints /login y /productos.

Microservicio de Login: Se encarga de la autenticación de usuarios, gestionando las credenciales y acceso a los datos de usuarios en MySQL.

Microservicio de Productos: Gestiona la información de productos, incluyendo operaciones de lectura y escritura sobre los datos de productos almacenados en MySQL.

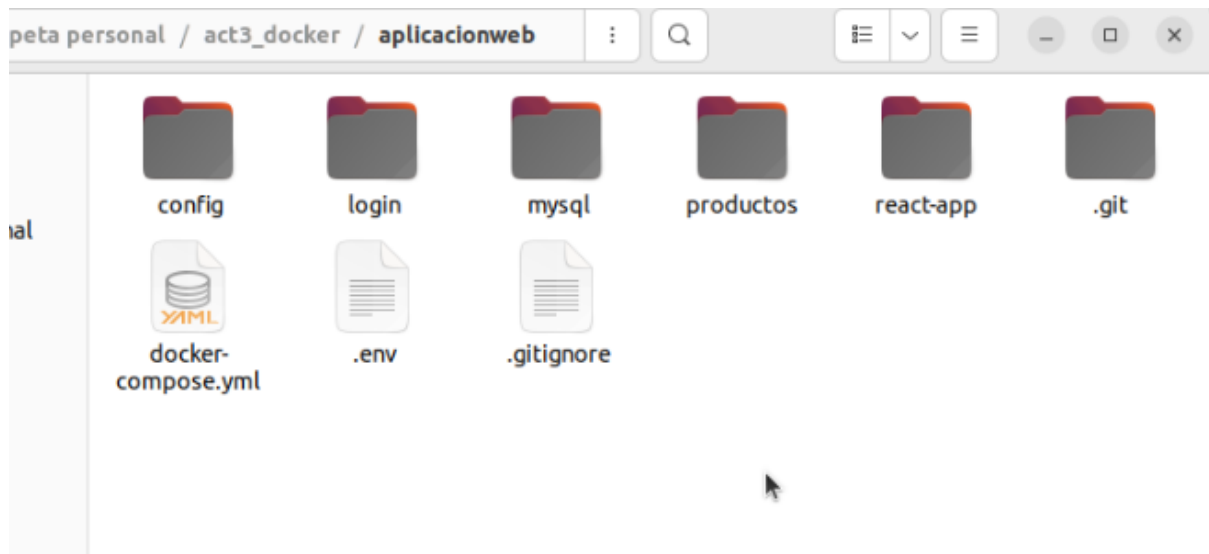
MySQL: Base de datos que almacena la información de los usuarios y los productos.



En nuestra máquina Local tendremos nuestra app react que a través de llamadas https se conectará con el servidor dentro de un contenedor apache. Este será el encargado de llamar a los microservicios de PHP con proxy reverse y estos microservicios tendrán conexión con una base de datos en otro contenedor.

Archivos de Configuración y Dockerfile

Antes de empezar para poder tener una aplicación de react hace falta instalar el npm haciendo **apt install npm**, instalar create-react-app haciendo **apt install create-react-app**, y por último **npm install web-vitals** y después crearemos la app con **npx create-react-app aplicacion-web**, si no tienes instalado create-react-app el mismo comando te preguntará si quieres instalarlo, después de esto ya tendrás tu app que podrás editar desde su App.js. Yo tengo una de ejemplo, con un login y productos de php básicos junto con un script de mysql básico.



Dentro de la carpeta config guardaré todos los archivos externos como certificados, el .conf del site, etc.

Antes de nada crearemos un certificado ssl para poder tener en nuestra web un poco más de seguridad y poder poner en la ruta de búsqueda https en lugar de http. Lo crearemos con su comando **sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/aplicacionwebcom-selfsigned.key -out etc/ssl/certs/aplicacionwebcom-selfsigned.crt**

[illegible]

Para el archivo de configuración de apache he creado www.aplicacionweb.com.conf donde pondremos todo lo necesario, como por ejemplo la especificación para el login y productos que se conecten a contenedores a través de proxy reverse, que redirige nuestras peticiones 443 del servidor apache (login o productos en este caso) a su respectivo contenedor docker proporcionando una respuesta y pasándola hacia el servidor de la misma forma al contrario, es decir actúa como un puente entre nuestro servidor y los contenedores con la información o peticiones que necesita. Activamos también ssl y pasamos su certificado y clave, además de agregar el proxy reverse para los microservicios.

Al acabar habremos de activarlo con **a2ensite** www.aplicacionweb.com.conf y editar /etc/hosts para que la ruta de nuestra local nos envíe a este. Ojo con especificar bien la ruta de tu aplicación.

```

root@desktop: /etc/apache2/sites-available
GNU nano 6.2 www.aplicacionweb.com.conf
<VirtualHost *:443>
    ServerName www.aplicacionweb.com
    DocumentRoot /var/www/build

    ProxyPass /login http://localhost:10000
    ProxyPassReverse /login http://localhost:10000

    ProxyPass /productos http://localhost:5000
    ProxyPassReverse /productos http://localhost:5000

    SSLEngine On
    SSLCertificateFile /etc/ssl/certs/aplicacionwebcom-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/aplicacionwebcom-selfsigned.key
    #logs opciones aqui abajo
</VirtualHost>

```

Para usar llamadas asíncronas en react usaremos axios por lo que habrá que instalarlo con **npm install axios** e importarlo en el proyecto, este hará las llamadas a su correspondiente url:

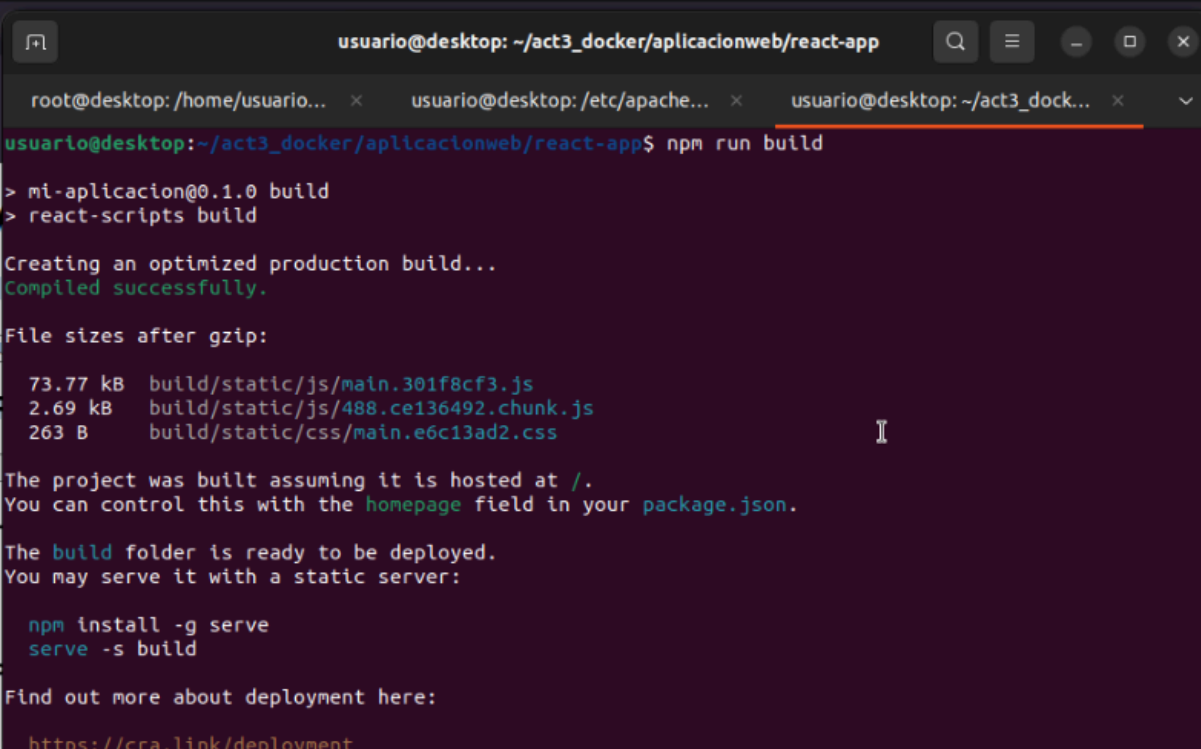
login (POST) → <http://www.aplicacionweb.com/login>

productos (GET) → <http://www.aplicacionweb.com/productos>

producto específico (GET) → http://www.aplicacionweb.com/productos?id_producto=1, para este utilizaremos un parámetro id_producto que será el encargado de darnos el id para usar en su SELECT del php.

Estas llamadas son redirigidas automáticamente por el proxy reverse, aunque ponga http estas llamadas se hacen desde la ruta principal https.

Para poder obtener una página que apache pueda producir se necesitará hacer en la app de react **npm run build** que creará una carpeta build que será la que copiaremos en /var/www



```
usuario@desktop: ~/act3_docker/aplicacionweb/react-app
root@desktop: /home/usuario... x usuario@desktop: /etc/apache... x usuario@desktop: ~/act3_dock... x
usuario@desktop:~/act3_docker/aplicacionweb/react-app$ npm run build

> mi-aplicacion@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

 73.77 kB  build/static/js/main.301f8cf3.js
 2.69 kB   build/static/js/488.ce136492.chunk.js
 263 B     build/static/css/main.e6c13ad2.css

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

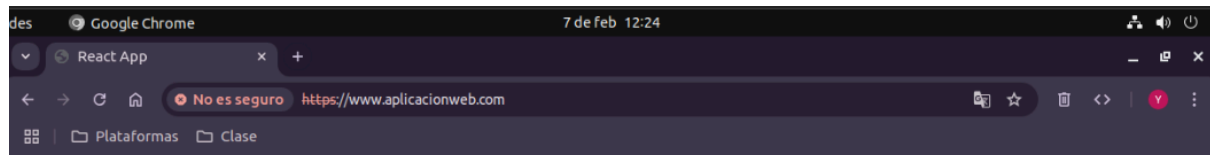
The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:

  https://cra.link/deployment
```

Una vez movida a esa carpeta /var/www ya podremos acceder a la ruta sin recibir un error 404.



Aplicación Web con React y Microservicios

Iniciar sesión

Usuario Clave

Productos

ID del Producto

Lista de Productos:

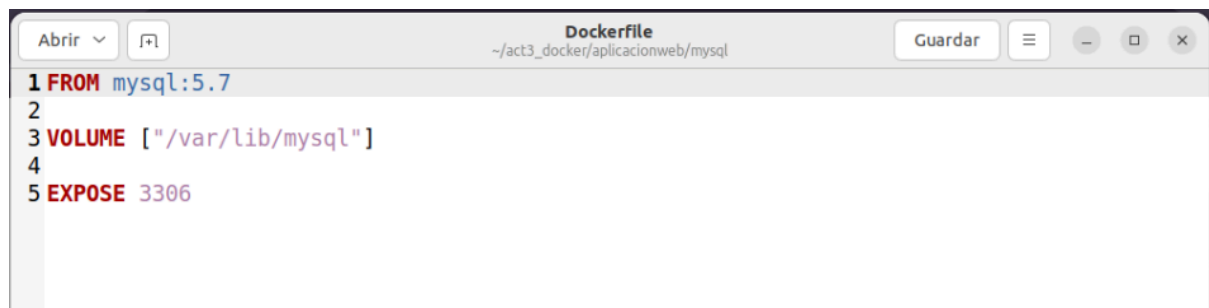
Para este despliegue necesitamos 3 contenedores docker (3 dockerfile), aunque lo desplegaré con docker compose para que se haga más sencillo todo en uno. También necesitare

Dockerfile del microservicio php de login:



Dockerfile del microservicio php de productos:

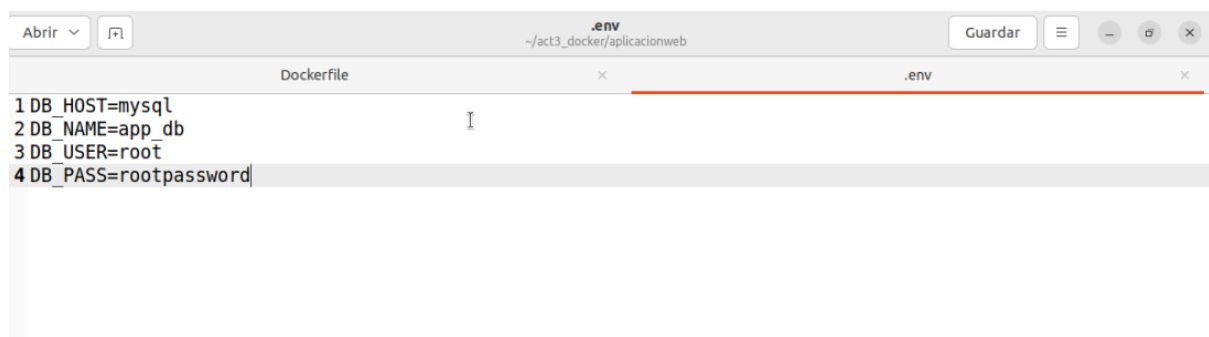


Dockerfile para nuestra base de datos:

```
1 FROM mysql:5.7
2
3 VOLUME ["/var/lib/mysql"]
4
5 EXPOSE 3306
```

En este caso, tendremos que crear un volumen para guardar todos los datos y que no se pierdan cuando cerramos y abrimos el contenedor, además añadiremos un script de creación de la base de datos (script muy básico) que solo se ejecutará la primera vez que se inicie el contenedor para así crear la base de datos pero no duplicar sus datos.

Para que se me haga más sencillo el despliegue y poder especificar aquí las variables de entorno, los link, el volumen etc... He hecho un docker compose para poder tener todo el despliegue de las imágenes y los contenedores de una vez. Es decir sería como hacer los docker build y docker run todos a la vez pero con un orden lógico seleccionado por mí, ya que en este caso los contenedores de login y productos dependen del de mysql por lo que el orden de creación será importante.

Archivo .env

```
1 DB_HOST=mysql
2 DB_NAME=app_db
3 DB_USER=root
4 DB_PASS=rootpassword
```


Archivo docker-compose.yml

```

1 version: '3.8'
2
3 services:
4   mysql:
5     build: ./mysql
6     container_name: mysql
7     restart: always
8     environment:
9       MYSQL_ROOT_PASSWORD: ${DB_PASS}
10      MYSQL_DATABASE: ${DB_NAME}
11     ports:
12       - "3306:3306"
13     volumes:
14       - mysql_data:/var/lib/mysql
15       - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
16
17   login:
18     build: ./login
19     container_name: login
20     restart: always
21     depends_on:
22       - mysql
23     env_file:
24       - .env
25     ports:
26       - "10000:80"
27
28   productos:
29     build: ./productos
30     container_name: productos
31     restart: always
32     depends_on:
33       - mysql
34     env_file:
35       - .env
36     ports:
37       - "5000:80"
38
39 volumes:
40   mysql_data:

```

Si quisieramos hacerlo sin docker compose deberíamos crear cada imagen con **docker build -t nombre_cont ./carpeta_contenedora_dockerfile**.

A la hora de hacer los docker run deberemos especificar algunos parámetros más:

MYSQL → **docker run -d --name mysql --restart always -e MYSQL_ROOT_PASSWORD=\${DB_PASS} -e MYSQL_DATABASE=\${DB_NAME} -p 3306:3306 -v mysql_data:/var/lib/mysql -v \$(pwd)/mysql/init.sql:/docker-entrypoint-initdb.d/init.sql mysql**

LOGIN → **docker run -d --name login --restart always --link mysql:mysql --env-file .env -p 10000:80 login**

PRODUCTOS → **docker run -d --name productos --restart always --link mysql:mysql --env-file .env -p 5000:80 productos**

Comprobamos con docker ps que estén activos

```

root@desktop: /home/usuario/act3_docker/aplicacionweb/react-app# docker ps
CONTAINER ID   IMAGE                                COMMAND                                  CREATED        STATUS        PORTS
c68785852bd3   aplicacionweb_productos             "docker-php-entrypoint..."           9 minutes ago  Up 9 minutes  5000/tcp, 0.0.0.0:5000->80/tcp, :::5000->80/tcp
48e2d3a6bd72   aplicacionweb_login                 "docker-php-entrypoint..."           9 minutes ago  Up 9 minutes  1000/tcp, 0.0.0.0:10000->80/tcp, :::10000->80/tcp
d01d02a7ac6f   aplicacionweb_mysql                 "docker-entrypoint.sh..."           9 minutes ago  Up 9 minutes  0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp
root@desktop: /home/usuario/act3_docker/aplicacionweb/react-app#

```

Comprobamos que la base de datos se ha creado bien entrando dentro del contenedor con `docker exec -it mysql mysql -u root -p` (-u habrá que poner el usuario que haya en .env al igual que cuando pida la contraseña), dentro del contenedor hacemos `show databases;` use `app_db;` y `show tables;` veremos que se ha creado todo bien para ver si las tablas estan llenas hacemos dos select a las tablas

```

root@desktop: /home/usuario/act3_docker/aplicacionweb
+-----+
| information_schema |
| app_db              |
| mysql               |
| performance_schema |
| sys                 |
+-----+
5 rows in set (0.01 sec)

mysql> use app_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_app_db |
+-----+
| productos         |
| usuarios          |
+-----+
2 rows in set (0.01 sec)

mysql>

```

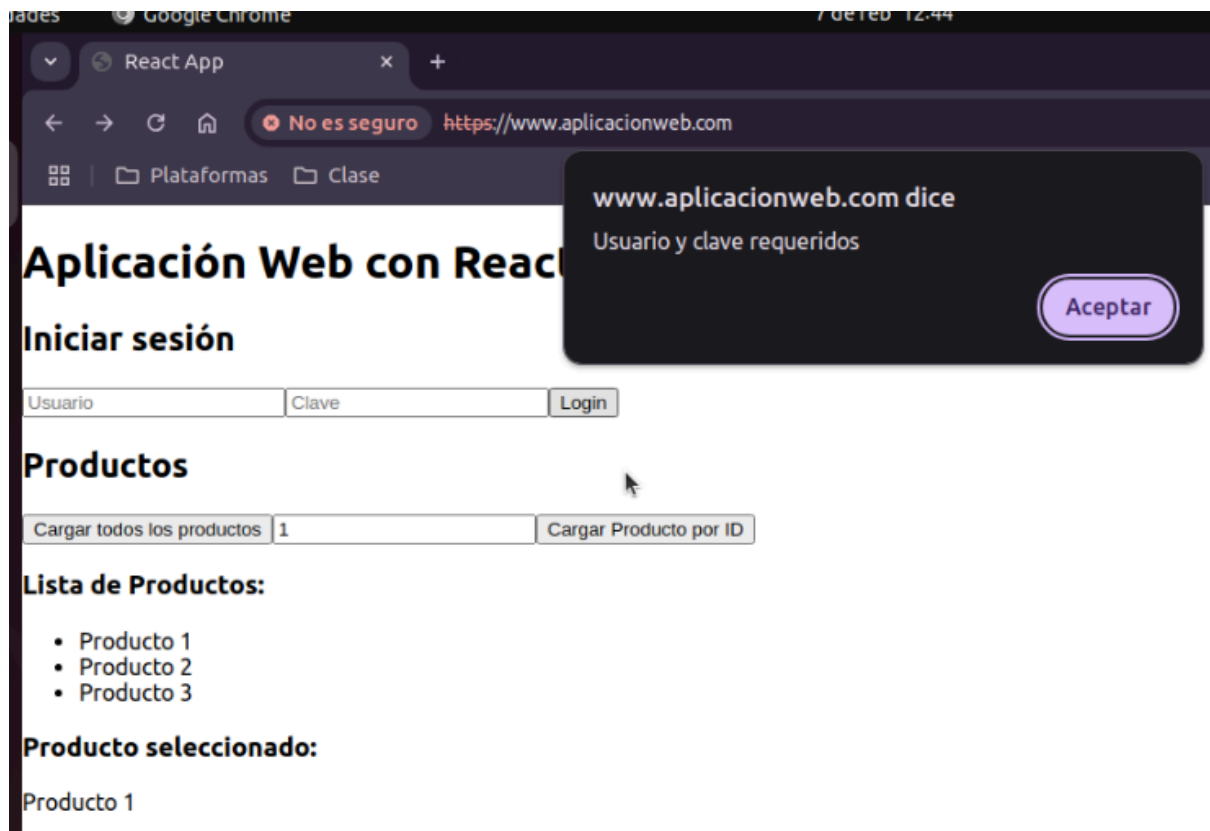
```
root@desktop: /home/usuario/act3_docker/aplicacionweb
mysql> select * from usuarios;
+-----+-----+-----+
| id | usuario | clave |
+-----+-----+-----+
| 1 | Yumara | 123456 |
| 2 | Javito | 123456 |
+-----+-----+-----+
2 rows in set (0.00 sec)

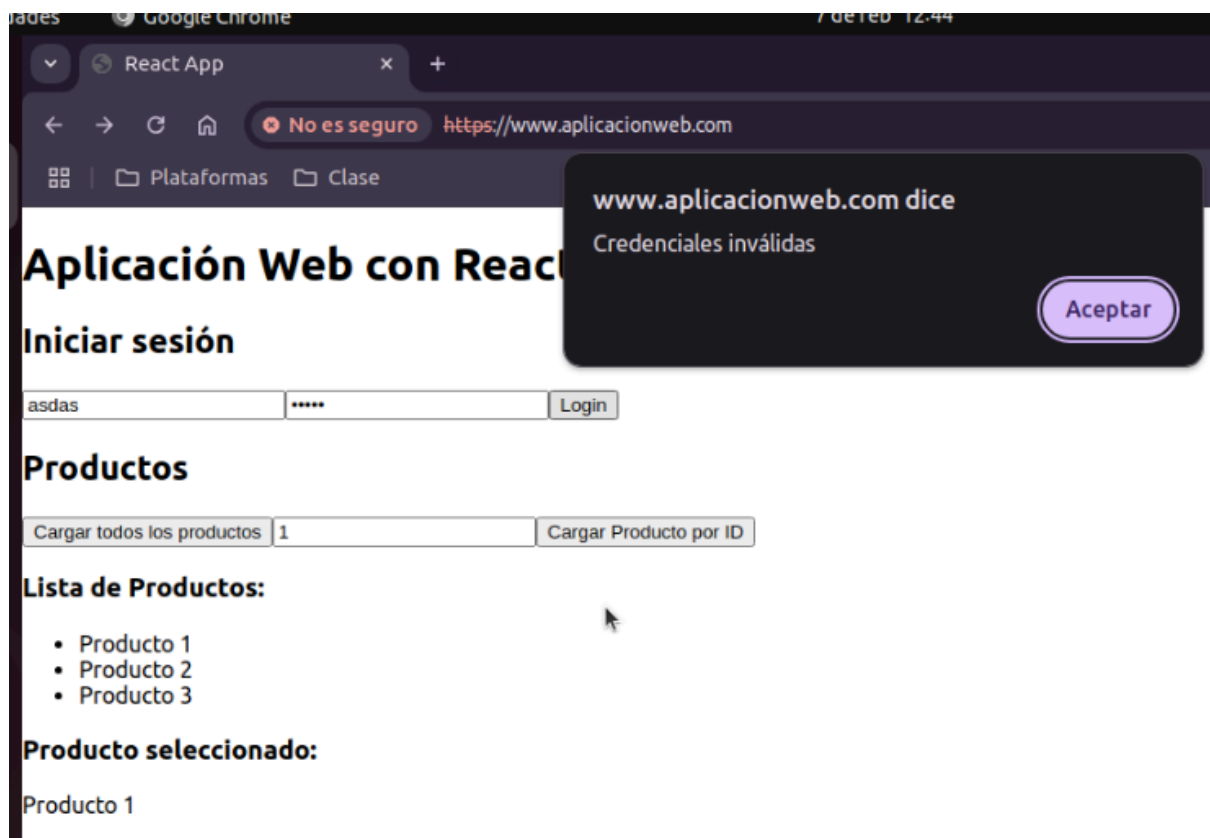
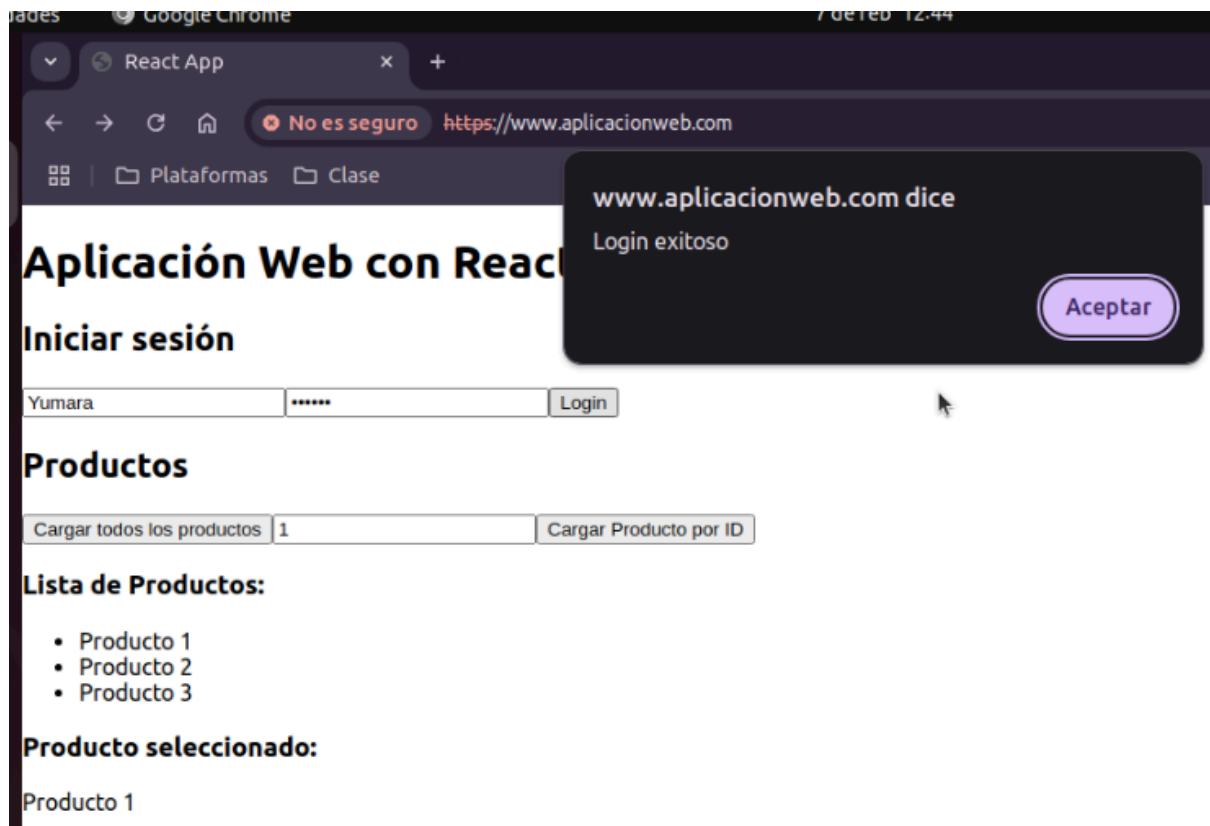
mysql> select * from usuarios and productos;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near 'and p
productos' at line 1
mysql> select * from productos;
+-----+-----+
| id | nombre |
+-----+-----+
| 1 | Producto 1 |
| 2 | Producto 2 |
| 3 | Producto 3 |
+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Tras esto ya deberíamos poder usar nuestra página y recibir los mensajes o productos

Pruebas de Funcionamiento y Rendimiento

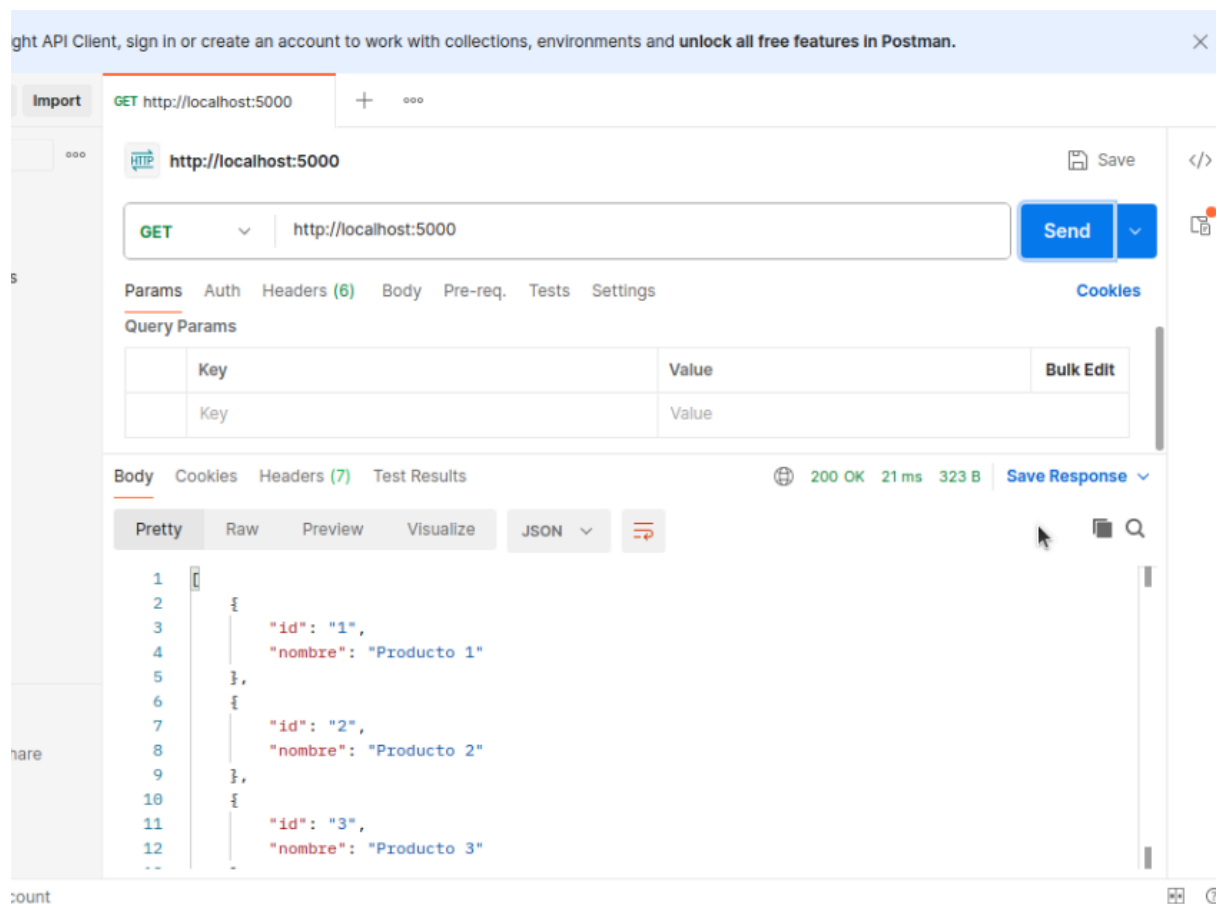




Pruebas con curl

```
usuario@desktop: ~  
usuario@desktop:~$ curl http://localhost:5000  
[{"id": "1", "nombre": "Producto 1"}, {"id": "2", "nombre": "Producto 2"}, {"id": "3", "nombre": "Producto 3"}]  
usuario@desktop:~$ ^C  
usuario@desktop:~$ curl http://localhost:10000  
{"status": "error", "message": "Usuario y clave requeridos"}  
usuario@desktop:~$ ^C  
usuario@desktop:~$
```

Pruebas con postman



the Lightweight API Client, sign in or create an account to work with collections, environments and **unlock all free features in Postman**.

New Import

POST http://localhost:10000

Save

POST http://localhost:10000 Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1 {
2   "usuario": "Yumara",
3   "clave": "123456"
4 }
```

Body Cookies Headers (7) Test Results 200 OK 13 ms 269 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "correcto",
3   "message": "Login exitoso"
4 }
```

Search Postman Sign In Create Account

client, sign in or create an account to work with collections, environments and **unlock all free features in Postman**.

POST http://localhost:5000/?id_producto=1

Save

POST http://localhost:5000/?id_producto=1 Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	id_producto	1	
	Key	Value	

Body Cookies Headers (7) Test Results 200 OK 22 ms 254 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "1",
3   "nombre": "Producto 1"
4 }
```