Universidad Rafael Landívar
Análisis y Diseño de Sistemas
Ing. José Andrés De la Cruz Franco
Teoría

Guía Técnica Paso a Paso:
Login con Flask, React, MySQL y Bootstrap usando Docker

Objetivo

Construir un sistema de login con una interfaz hecha en **React + Bootstrap**, backend en **Flask**, y base de datos **MySQL**, todo ejecutado mediante **Docker**

Tabla de Contenido

Estructura del proyecto

```
login-app/
│
├── backend/
│   ├── app.py
│   ├── requirements.txt
│   └── Dockerfile
│
├── frontend/
│   ├── public/
│   │   └── index.html
│   ├── src/
│   │   ├── App.js
│   │   └── index.js
│   ├── package.json
│   └── Dockerfile
│
├── mysql-init/
│   └── init.sql
│
└── docker-compose.yml
```

Paso 1: Backend con Flask

Archivo: backend/app.py

```python
# Importa Flask para crear la aplicación web, request para leer datos entrantes, y jsonify para responder en JSON
from flask import Flask, request, jsonify

# Importa el conector de MySQL para Python
import mysql.connector

# Importa CORS para permitir peticiones desde otros orígenes (como el frontend en React)
from flask_cors import CORS

# Crea la aplicación Flask
app = Flask(__name__)

# Habilita CORS para toda la app, permitiendo que el frontend consuma este backend
CORS(app)

# Define una ruta (endpoint) POST en /login
@app.route('/login', methods=['POST'])
def login():
    # Obtiene el JSON del cuerpo de la solicitud (usuario y contraseña)
    data = request.get_json()
    username = data.get("username")
    password = data.get("password")
```

```
    # Se conecta a la base de datos MySQL ejecutándose en el contenedor llamado 'db'
    db = mysql.connector.connect(
        host="db",
        user="root",
        password="12345",
        database="users_db"
    )

    # Crea un cursor que devuelve resultados como diccionarios
    cursor = db.cursor(dictionary=True)

    # Ejecuta una consulta SQL para buscar un usuario con ese username y password
    cursor.execute("SELECT * FROM users WHERE username=%s AND password=%s", (username, password))

    # Obtiene el primer resultado (si existe)
    user = cursor.fetchone()

    # Si encontró un usuario, devuelve mensaje exitoso
    if user:
        return jsonify({"message": "Login exitoso"})
    else:
        # Si no lo encontró, devuelve error 401 (no autorizado)
        return jsonify({"message": "Credenciales incorrectas"}), 401

# Hace que la app se ejecute cuando este archivo se ejecuta directamente
if __name__ == '__main__':
    # Ejecuta el servidor Flask accesible desde cualquier IP (útil en Docker)
    app.run(host='0.0.0.0', port=5000)
```

Archivo: backend/requirements.txt

```
flask
mysql-connector-python
flask-cors
```

Archivo: backend/Dockerfile

```
# Utiliza como base la imagen oficial de Python versión 3.10

FROM python:3.10 # Establece el directorio de trabajo dentro del contenedor en /app

WORKDIR /app # Copia todos los archivos del proyecto (desde el host) al directorio /app del contenedor

COPY . /app # Instala las dependencias definidas en requirements.txt sin usar la caché (más limpio)

RUN pip install --no-cache-dir -r requirements.txt # Define el comando por defecto que se ejecutará al iniciar el
contenedor: ejecutar app.py con Python

CMD ["python", "app.py"]
```

Paso 2: Base de Datos MySQL

Archivo: mysql-init/init.sql

```
-- Selecciona la base de datos llamada 'users_db' donde se crearán las tablas
USE users_db;

-- Crea una tabla llamada 'users' si no existe aún
CREATE TABLE IF NOT EXISTS users (

  -- Columna 'id' como clave primaria autoincremental
  id INT AUTO_INCREMENT PRIMARY KEY,
  -- Columna para el nombre de usuario con un máximo de 50 caracteres
  username VARCHAR(50),
  -- Columna para la contraseña con un máximo de 50 caracteres
  password VARCHAR(50)

);

-- Inserta un usuario inicial de prueba con username 'admin' y contraseña '1234'
```

```sql
INSERT INTO users (username, password) VALUES ('admin', '1234');
```

Paso 3: ontend con React + Bootstrap

Archivo: frontend/package.json

```json
{
  "name": "frontend",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "bootstrap": "^5.3.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build"
  }
}
```

Archivo: frontend/src/index.js

```jsx
import React from 'react'; // Importa React para poder usar JSX y componentes

import ReactDOM from 'react-dom/client'; // Importa el módulo ReactDOM para renderizar la app en el DOM (versión moderna con createRoot)

import App from './App'; // Importa el componente principal App

import 'bootstrap/dist/css/bootstrap.min.css'; // Importa los estilos de Bootstrap desde su paquete npm

// Crea el punto de entrada (root) vinculado al <div id="root"> en index.html
const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);// Renderiza el componente App dentro del div con id="root"
```

Archivo: frontend/src/App.js

```jsx
// Importa React y el hook useState para manejar estados dentro del componente
import React, { useState } from 'react';

// Define el componente funcional App
function App() {
```

```
  // Declara estados para el usuario, la contraseña y el mensaje de respuesta
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [message, setMessage] = useState('');

  // Función que maneja el envío del formulario
  const handleLogin = async (e) => {
    e.preventDefault(); // Evita que la página se recargue
    // Envia una solicitud POST al backend con los datos de login
    const response = await fetch('http://localhost:5000/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ username, password }) // Convierte el objeto JS a JSON
    });
    // Convierte la respuesta a JSON
    const data = await response.json();
    // Muestra el mensaje que retorna el backend (exitoso o error)
    setMessage(data.message);
  };
  // Estructura visual del formulario usando clases de Bootstrap
  return (
    <div className="container mt-5">
      <div className="row justify-content-center">
        <div className="col-md-4">
          <div className="card shadow">
            <div className="card-body">
              <h3 className="text-center mb-4">Iniciar Sesión</h3>
              {/* Formulario controlado */}
              <form onSubmit={handleLogin}>
                {/* Campo de usuario */}
                <div className="mb-3">
                  <label className="form-label">Usuario</label>
                  <input
                    type="text"
                    className="form-control"
                    value={username}
                    onChange={e => setUsername(e.target.value)}
                    placeholder="Ingresa tu usuario"
                    required
                  />
                </div>
                {/* Campo de contraseña */}
                <div className="mb-3">
                  <label className="form-label">Contraseña</label>
                  <input
                    type="password"
                    className="form-control"
                    value={password}
                    onChange={e => setPassword(e.target.value)}
                    placeholder="Ingresa tu contraseña"
                    required
                  />
                </div>
                {/* Botón de enviar */}
                <button type="submit" className="btn btn-primary w-100">Ingresar</button>
              </form>
              {/* Mensaje de resultado (éxito o error) */}
              {message && (
                <div className="alert alert-info mt-3 text-center">{message}</div>
              )}
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
// Exporta el componente para que pueda ser usado en index.js
export default App;
```

Archivo: frontend/public/index.html

```
<!-- Indica que el documento está escrito en HTML5 -->
<!DOCTYPE html>
<!-- Elemento raíz del documento HTML, con idioma español -->
```

```html
<html lang="es">
  <head>
    <!-- Define la codificación de caracteres como UTF-8 (soporta caracteres latinos) -->
    <meta charset="UTF-8" />
    <!-- Hace que el diseño sea responsive (adaptable a móviles) -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!-- Título que se muestra en la pestaña del navegador -->
    <title>Login App</title>
  </head>
  <body>
    <!-- Mensaje que se muestra si el usuario tiene JavaScript deshabilitado -->
    <noscript>Necesitás habilitar JavaScript para usar esta app.</noscript>
    <!-- Punto de montaje donde React insertará la interfaz de usuario -->
    <div id="root"></div>
  </body>
</html>
```

Archivo: frontend/Dockerfile

```dockerfile
# Usa una imagen oficial de Node.js versión 18 como base para el contenedor
FROM node:18

# Establece el directorio de trabajo dentro del contenedor en /app
WORKDIR /app

# Copia todos los archivos del proyecto frontend (código, config, etc.) al directorio de trabajo
COPY . .

# Instala las dependencias listadas en package.json dentro del contenedor
RUN npm install

# Construye la versión optimizada del frontend (React produce los archivos en la carpeta 'build')
RUN npm run build

# Instala globalmente el servidor estático 'serve' para servir la app construida
RUN npm install -g serve

# Define el comando por defecto para ejecutar el contenedor:
# servir la carpeta 'build' en el puerto 3000
CMD ["serve", "-s", "build", "-l", "3000"]
```

Archivo: docker-compose.yml

```yaml
# Versión de Docker Compose utilizada
version: '3.8'

# Definición de los servicios (contenedores)
services:
  # Servicio: backend (Flask)
  backend:
    # Construye la imagen Docker desde la carpeta ./backend (donde está el Dockerfile)
    build: ./backend
    # Expone el puerto 5000 del contenedor en el puerto 5000 del host
    ports:
      - "5000:5000"
    # Espera a que el contenedor de la base de datos esté listo antes de arrancar
    depends_on:
      - db
    # Variables de entorno usadas por Flask para conectarse a MySQL
    environment:
      - MYSQL_HOST=db               # nombre del contenedor de la BD
      - MYSQL_USER=root
      - MYSQL_PASSWORD=12345
      - MYSQL_DATABASE=users_db
  # Servicio: base de datos MySQL
  db:
    # Utiliza la imagen oficial de MySQL 8.0
    image: mysql:8.0

    # Siempre intenta reiniciarse si se detiene
    restart: always
    # Variables de entorno para crear la base de datos al iniciar
    environment:
```

```
    - MYSQL_ROOT_PASSWORD=12345  # contraseña del root
    - MYSQL_DATABASE=users_db    # nombre de la BD que se creará
  # Expone el puerto 3306 de MySQL para conexión externa (opcional en este caso)
  ports:
    - "3306:3306"
  # Monta un volumen para ejecutar automáticamente scripts de inicialización
  volumes:
    - ./mysql-init:/docker-entrypoint-initdb.d
# Servicio: frontend (React)
frontend:
  # Construye la imagen Docker desde la carpeta ./frontend
  build: ./frontend
  # Expone el puerto 3000 del contenedor en el puerto 3000 del host
  ports:
    - "3000:3000"
  # Se asegura de que el backend esté corriendo antes de iniciar este servicio
  depends_on:
    - backend
```

Glosario

TÉRMINO	DEFINICIÓN	ENLACE
DOCKER	Plataforma para automatizar la creación, despliegue y ejecución de aplicaciones dentro de contenedores.	https://www.docker.com/
CONTENEDOR	Unidad ligera que incluye todo lo necesario para ejecutar una app: código, librerías y configuración.	https://docs.docker.com/get-started/overview/
DOCKERFILE	Archivo con instrucciones para construir imágenes de Docker personalizadas.	https://docs.docker.com/engine/reference/builder/
DOCKER-COMPOSE	Herramienta para definir y ejecutar múltiples contenedores desde un archivo YAML.	https://docs.docker.com/compose/
REACT	Librería de JavaScript para construir interfaces de usuario reactivas.	https://reactjs.org/

BOOTSTRAP	Framework CSS para construir interfaces modernas y responsivas.	https://getbootstrap.com/
FLASK	Microframework de Python para construir APIs de forma rápida.	https://flask.palletsprojects.com/
MYSQL	Sistema gestor de bases de datos relacional.	https://dev.mysql.com/doc/
API REST	Estilo de arquitectura que permite comunicación entre cliente y servidor mediante HTTP.	https://restfulapi.net/
FETCH API	Interfaz JavaScript para hacer peticiones HTTP asíncronas.	https://developer.mozilla.org/es/docs/Web/API/Fetch_API
PACKAGE.JSON	Archivo de configuración de proyectos Node.js con scripts y dependencias.	https://docs.npmjs.com/cli/v8/configuring-npm/package-json
NPM	Gestor de paquetes de JavaScript para instalar librerías como React.	https://www.npmjs.com/
CMD	Instrucción en Dockerfile que define el comando por defecto al iniciar el contenedor.	https://docs.docker.com/engine/reference/builder/#cmd
VOLUME	Mecanismo para almacenar datos persistentes en contenedores Docker.	https://docs.docker.com/storage/volumes/
YAML	Formato legible para humanos usado en archivos de configuración como docker-compose.yml.	https://yaml.org/
CORS	Política de seguridad que controla el acceso entre orígenes distintos.	https://developer.mozilla.org/es/docs/Web/HTTP/CORS