



INFORME DE GUÍA PRÁCTICA

I. PORTADA

Tema:	Servidores (Procesos y Comunicación)
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	Nivel - Paralelo
Alumnos participantes:	Albán Chávez Melanie Elizabeth Álvarez Coello Freddy Guillermo Salas Acosta Alison Micaela Soriano Proaño Rafael Andrés
Asignatura:	Aplicaciones Distribuidas
Docente:	Ing. José Caiza, Mg.

II. INFORME DE GUÍA PRÁCTICA

2.1 Objetivos

General:

Conocer los modelos arquitectónicos de los sistemas distribuidos para una implementación exitosa manteniendo la integridad, la escalabilidad y la confiabilidad en un sistema distribuido.

Específicos:

- Diseñar e implementar una arquitectura distribuida que integre servicios de red como proxy inverso, servidor de API y base de datos utilizando contenedores Docker.
- Configurar un servidor proxy Squid para el control de acceso a sitios web mediante reglas dinámicas gestionadas desde una API.
- Automatizar la comunicación entre microservicios utilizando protocolos modernos como HTTP o gRPC para garantizar la interoperabilidad del sistema.

2.2 Modalidad

Presencial

2.3 Tiempo de duración

Presenciales: 4

No presenciales: 0

2.4 Instrucciones

El trabajo se desarrollará en parejas.

Lea las indicaciones del archivo adjunto y desarrolle las actividades solicitadas.

La práctica se revisará en clase y el informe se debe subir al aula virtual de la materia en formato PDF.

2.5 Listado de equipos, materiales y recursos

Listado de equipos y materiales generales empleados en la guía práctica:

- Inteligencia artificial
- Internet
- Bases de datos disponibles en la biblioteca virtual de la Universidad.
- Bibliografía de la asignatura.
- Material disponible en el aula virtual de la asignatura.
- Virtual Box
- Linux



TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:

- ☒ Plataformas educativas
- ☒ Simuladores y laboratorios virtuales
- ☐ Aplicaciones educativas
- ☐ Recursos audiovisuales
- ☐ Gamificación
- ☒ Inteligencia Artificial
- Otros (Especifique): Docker

2.6 Actividades por desarrollar

- Bloqueo de URLs con patrones o palabras clave con Squid (con ACLs y expresiones regulares).
- API para controlar reglas dinámicamente con ASP.NET Core API.
- Servidor web separado con Ubuntu + Kestrel (o Nginx como proxy inverso).
- Servidor de base de datos separado con Ubuntu + SQL Server o MongoDB.
- Comunicación entre microservicios con gRPC o HTTP (según preferencia)

2.7 Resultados obtenidos

La API fue desarrollada en Visual Studio utilizando el framework .NET 9 y el lenguaje C#, con el objetivo de gestionar reglas de bloqueo de URLs. Se implementó un controlador llamado `UrlRulesController`, que permite realizar operaciones CRUD sobre una base de datos MongoDB, almacenando patrones que serán utilizados por un servidor proxy Squid. Cada vez que se agrega o elimina una regla, el sistema actualiza automáticamente un archivo llamado `blocked_patterns.txt`, el cual es monitoreado por un script que recarga la configuración de Squid sin necesidad de reiniciar el servicio. Para facilitar el despliegue, la API fue contenedorizada mediante Docker utilizando un Dockerfile basado en la imagen oficial de ASP.NET 9.0 (mcr.microsoft.com/dotnet/aspnet:9.0). En este archivo se define el directorio de trabajo (`/app`), se copian los archivos publicados desde `bin/Release/net9.0/publish/`, se expone el puerto 80 y se establece como punto de entrada la ejecución del archivo `FuelApi.dll`, lo que permite que la aplicación inicie automáticamente al levantar el contenedor.

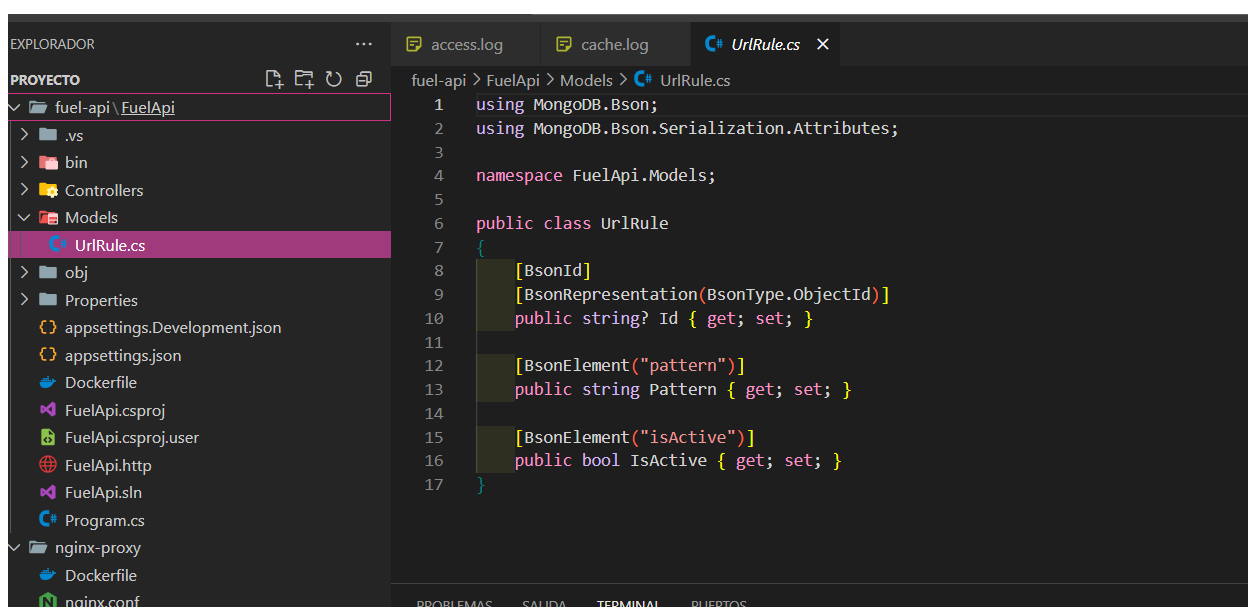


Ilustración 1. Modelo para creación de la API



```
access.log  cache.log  C# UrlRulesController.cs X
fuel-api > FuelApi > Controllers > C# UrlRulesController.cs
1  using Microsoft.AspNetCore.Mvc;
2  using MongoDB.Driver;
3  using FuelApi.Models;
4  using System.Threading.Tasks;
5  using System.Collections.Generic;
6
7  namespace FuelApi.Controllers;
8
9  [Route("api/[controller]")]
10 [ApiController]
11 public class UrlRulesController : ControllerBase
12 {
13     private readonly IMongoCollection<UrlRule> _rulesCollection;
14
15     public UrlRulesController(IMongoClient mongoClient)
16     {
17         var database = mongoClient.GetDatabase("fuel_rules");
18         _rulesCollection = database.GetCollection<UrlRule>("rules");
19     }
20
21     [HttpGet]
22     public async Task<ActionResult<List<UrlRule>>> GetRules()
23     {
24         var rules = await _rulesCollection.Find(r => true).ToListAsync();
25         return Ok(rules);
26     }
27
28     [HttpPost]
29     public async Task<ActionResult> AddRule([FromBody] UrlRule rule)
30     {
31         rule.IsActive = true;
32         await _rulesCollection.InsertOneAsync(rule);
33
34         // Solo actualiza el archivo, el script se encarga de recargar Squid
35         await UpdatesquidConfig();
36         return Ok("Regla agregada");
37     }
38 }
```

Ilustración 2. Métodos CRUD en el Controller

```
}
}

[HttpDelete("{id}")]
public async Task<ActionResult> DeleteRule(string id)
{
    await _rulesCollection.DeleteOneAsync(r => r.Id == id);

    // Solo actualiza el archivo, el script se encarga de recargar Squid
    await UpdatesquidConfig();
    return Ok("Regla eliminada");
}

private async Task UpdatesquidConfig()
{
    // Obtener reglas activas
    var rules = await _rulesCollection.Find(r => r.IsActive).ToListAsync();

    // Escribir en blocked_patterns.txt
    var patterns = string.Join("\n", rules.Select(r => r.Pattern));
    await System.IO.File.WriteAllTextAsync("/squid/blocked_patterns.txt", patterns);
}
}
```

Ilustración 3. Métodos CRUD en el Controller



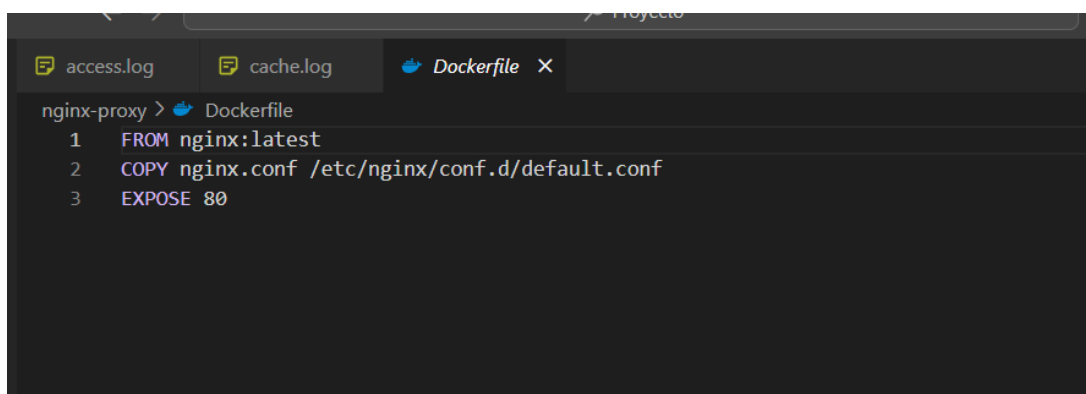
```
fuel-api > FuelApi > Program.cs
1  using MongoDB.Driver;
2  using FuelApi.Models;
3  using Microsoft.AspNetCore.Hosting;
4  using Microsoft.Extensions.Hosting;
5
6  var builder = WebApplication.CreateBuilder(args);
7
8  // Agregar servicios
9  builder.Services.AddControllers();
10 builder.Services.AddSingleton<IMongoClient>(new MongoClient("mongodb://admin:password@mongodb:27017"));
11
12 // Configurar Swagger
13 builder.Services.AddSwaggerGen(c =>
14 {
15     c.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo
16     {
17         Title = "Fuel Rules API",
18         Version = "v1"
19     });
20 });
21
22 var app = builder.Build();
23
24 // Configurar Swagger UI
25 app.UseSwagger();
26 app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "Fuel Rules API v1"));
27
28 // Configurar middleware
29 app.UseHttpsRedirection();
30 app.UseAuthorization();
31 app.MapControllers();
32
33 // Escuchar en todas las interfaces en el puerto 8080
34 app.Run("http://0.0.0.0:8080");
35
```

Ilustración 4. Configuración de Archivo Program.cs

```
... < > Proyecto
access.log watch-squid.sh Dockerfile X cache.log docker-compose.yml
fuel-api > FuelApi > Dockerfile
1  FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS runtime
2  WORKDIR /app
3  COPY ./bin/Release/net9.0/publish/ .
4  EXPOSE 80
5  ENTRYPOINT ["dotnet", "FuelApi.dll"]
```

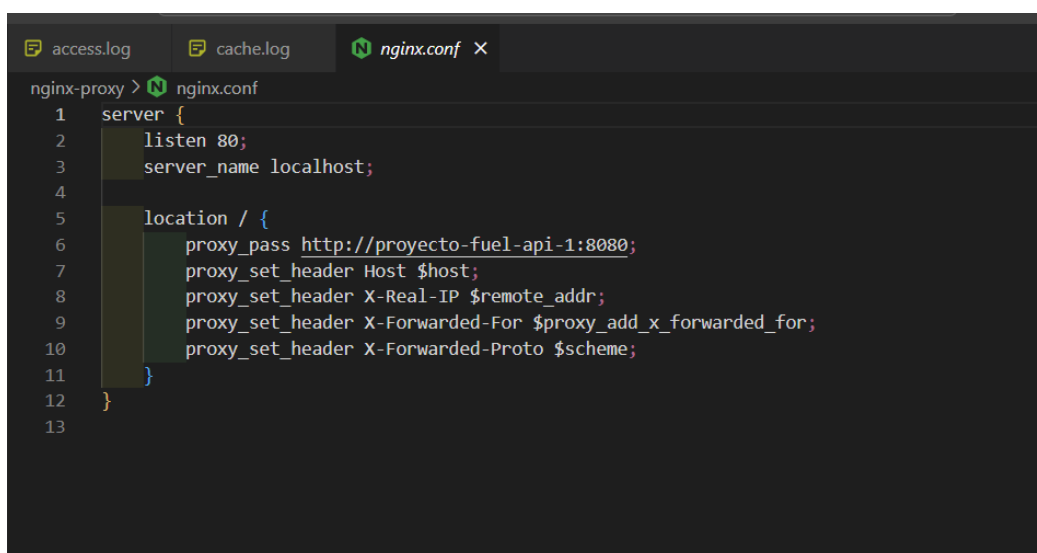
Ilustración 5. Configuración Dockerfile de la API

Para permitir el acceso externo a la API y centralizar el tráfico HTTP, se configuró un contenedor Nginx actuando como proxy inverso. El contenedor se construyó a partir de la imagen oficial `nginx:latest`, y se utilizó un Dockerfile sencillo que copia el archivo de configuración `nginx.conf` al directorio predeterminado de Nginx (`/etc/nginx/conf.d/default.conf`) y expone el puerto 80. En el archivo `nginx.conf`, se define un bloque `server` que escucha en el puerto 80 y redirige todas las solicitudes entrantes hacia el contenedor de la API (`proyecto-fuel-api-1`) utilizando la directiva `proxy_pass` apuntando a `http://proyecto-fuel-api-1:8080`. Además, se configuran encabezados HTTP como `Host`, `X-Real-IP`, `X-Forwarded-For` y `X-Forwarded-Proto` para asegurar que la API reciba la información correcta sobre las solicitudes originales, lo que facilita la integración segura y eficiente entre el cliente y el backend.



```
nginx-proxy > Dockerfile
1 FROM nginx:latest
2 COPY nginx.conf /etc/nginx/conf.d/default.conf
3 EXPOSE 80
```

Ilustración 6. Contenedor nginx-proxy



```
nginx-proxy > nginx.conf
1 server {
2     listen 80;
3     server_name localhost;
4
5     location / {
6         proxy_pass http://proyecto-fuel-api-1:8080;
7         proxy_set_header Host $host;
8         proxy_set_header X-Real-IP $remote_addr;
9         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
10        proxy_set_header X-Forwarded-Proto $scheme;
11    }
12 }
13
```

Ilustración 7. Configuración de nginx

Para controlar el acceso a sitios web y aplicar filtros de navegación, se implementó un servidor Squid Proxy dentro de un contenedor Docker utilizando una imagen basada en ubuntu:22.04. En su Dockerfile, se instalan Squid, inotify-tools (para monitoreo de archivos) y curl, además de copiarse un archivo personalizado de configuración (squid.conf) y el archivo blocked_patterns.txt, donde se almacenan los patrones de URLs que deben ser bloqueadas. El contenedor expone el puerto 3128, correspondiente al puerto por defecto de Squid. La configuración de Squid utiliza una ACL (blocked_urls) que carga dinámicamente los patrones desde el archivo blocked_patterns.txt, negando el acceso a las URLs que coincidan con dichos patrones, mientras que permite el resto del tráfico. También se incluye un script (watch-squid.sh) encargado de vigilar cambios en el archivo de patrones bloqueados y recargar automáticamente la configuración de Squid. Esta integración asegura que cualquier actualización en las reglas de bloqueo definidas por la API sea aplicada de forma inmediata en el proxy sin intervención manual.



```
access.log  cache.log  blocked_patterns.txt X
squid-proxy > blocked_patterns.txt
1  string
2  youtube
3  facebook
```

Ilustración 8. Archivo para bloqueo de rutas

```
← → Proyecto
access.log  cache.log  Dockerfile X
squid-proxy > Dockerfile
1  FROM ubuntu:22.04
2
3  # Actualizar e instalar Squid y las herramientas necesarias
4  RUN apt update && apt install -y squid inotify-tools
5  RUN apt update && apt install -y curl
6
7
8  # Copiar el archivo de configuración de Squid
9  COPY squid.conf /etc/squid/squid.conf
10
11 # Copiar el archivo blocked_patterns.txt al contenedor
12 COPY blocked_patterns.txt /etc/squid/blocked_patterns.txt
13
14 # Exponer el puerto 3128 para el proxy Squid
15 EXPOSE 3128
16
17 # Copiar el script para monitorizar los cambios
18 COPY watch-squid.sh /usr/local/bin/watch-squid.sh
19
20 # Asegurarse de que el script sea ejecutable
21 RUN chmod +x /usr/local/bin/watch-squid.sh
22
23 # Comando de inicio: ejecutar el script watch-squid.sh
24 CMD ["/usr/local/bin/watch-squid.sh"]
25
```

Ilustración 9. Configuración de contenedor Docker para squid-proxy



```
access.log x cache.log squid.conf x
squid-proxy > squid.conf
1 http_port 0.0.0.0:3128
2
3 # ACLs dinámicas desde archivo
4 acl blocked_urls url_regex -i "/etc/squid/blocked_patterns.txt"
5 acl all src all
6
7 # Reglas de acceso
8 http_access deny blocked_urls
9 http_access allow all
10
11 # Logs
12 access_log /var/log/squid/access.log squid
13 logformat squid %ts.%03tu %6tr %>a %Ss/%03>Hs %<st %rm %ru %|un %Sh/%<a %mt
14
```

Ilustración 10. Configuración Squid

```
access.log cache.log watch-squid.sh x
squid-proxy > watch-squid.sh
1 #!/bin/bash
2
3 # Ruta del archivo a monitorear
4 PATTERNS_FILE="/etc/squid/blocked_patterns.txt"
5
6 # Comando para recargar Squid
7 RELOAD_CMD="squid -k reconfigure"
8
9
10 # Verificar que el archivo existe
11 if [ ! -f "$PATTERNS_FILE" ]; then
12     echo "Error: $PATTERNS_FILE no existe."
13     exit 1
14 fi
15
16 # Iniciar Squid en foreground para mantener el contenedor vivo
17 echo "Iniciando Squid..."
18 squid -N
19
20 # Esperar a que Squid esté listo
21 sleep 2
22
23 # Bucle para monitorear el archivo
24 echo "Monitoreando $PATTERNS_FILE para cambios..."
25 inotifywait -m -e close_write --format "%w%f" "$PATTERNS_FILE" | while read -r FILE; do
26     echo "El archivo $FILE ha cambiado. Recargando Squid..."
27     $RELOAD_CMD || echo "Error al recargar Squid"
28 done
```

Ilustración 11. Script de automatización de cambios

El archivo docker-compose.yml define los servicios que conforman la infraestructura de la aplicación, incluyendo un Squid Proxy, una API de Fuel, un Nginx Proxy y una base de datos MongoDB. En el servicio squid-proxy, se configura el contenedor para construir la imagen desde el directorio ./squid-proxy, exponiendo el puerto 3128 y montando volúmenes para almacenar los logs de Squid y el archivo de patrones bloqueados blocked_patterns.txt. El servicio fuel-api construye la API desde el directorio ./fuel-api/FuelApi y comparte el archivo de patrones bloqueados con Squid. El servicio nginx-proxy se encarga de redirigir las solicitudes HTTP al servicio de API en el puerto 8080, mientras que mongodb utiliza la imagen oficial de MongoDB para proporcionar la base de datos, exponiendo el puerto 27017. Todos los servicios se conectan a una red interna llamada fuel-network para facilitar la comunicación entre ellos, y se define un volumen persistente mongo-data para almacenar los datos de MongoDB.



Esta configuración permite la integración entre los distintos componentes de la infraestructura de manera eficiente y aislada.

```
access.log  cache.log  docker-compose.yml X
docker-compose.yml
1
2 services:
3   squid-proxy:
4     build: ./squid-proxy
5     ports:
6       - "3128:3128"
7     volumes:
8       - ./squid-logs:/var/log/squid
9       - ./squid-proxy/blocked_patterns.txt:/etc/squid/blocked_patterns.txt
10    networks:
11      - fuel-network
12   fuel-api:
13     build: ./fuel-api/FuelApi
14     volumes:
15       - ./squid-proxy/blocked_patterns.txt:/squid/blocked_patterns.txt
16
17    networks:
18      - fuel-network
19   nginx-proxy:
20     build: ./nginx-proxy
21     ports:
22       - "80:80"
23     depends_on:
24       - fuel-api
25     networks:
26       - fuel-network
27   mongodb:
28     image: mongo:latest
29     environment:
30       MONGO_INITDB_ROOT_USERNAME: admin
31       MONGO_INITDB_ROOT_PASSWORD: password
32     volumes:
33       - mongo-data:/data/db
34     ports:
35       - "27017:27017"
36     networks:
37       - fuel-network
38   networks:
39     fuel-network:
40       driver: bridge
41   volumes:
42     mongo-data:
```

Ilustración 12. Configuración de docker-compose.yml

Se utilizó el comando `docker compose up -d` para iniciar los contenedores en segundo plano, creando y montando la infraestructura definida en el archivo `docker-compose.yml`. Al ejecutar este comando, los servicios como Squid Proxy, Fuel API, Nginx Proxy y MongoDB se desplegaron en Docker. Posteriormente, se pudo verificar la ejecución de los contenedores abriendo Docker que muestra los contenedores activos y sus respectivos puertos expuestos.



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: MARZO – JULIO 2025



```
✓ Network proyecto_fuel-network      Removed
PS C:\Users\Melan\Downloads\Proyecto\Proyecto> docker compose up -d
[+] Running 5/5
✓ Network proyecto_fuel-network      Created
✓ Container proyecto-mongodb-1       Started
✓ Container proyecto-fuel-api-1       Started
✓ Container proyecto-squid-proxy-1    Started
✓ Container proyecto-nginx-proxy-1    Started
```

Ilustración 13. Inicialización de Docker

Containers

[Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage 2.83% / 1200% (12 CPUs available)

Container memory usage 408.17MB / 7.43GB

Show charts

Search

Only show running containers

Delete

▶ || ◻

✓	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
✓	proyecto	-	-	-	2.83%	3 minutes ago	⌵ ⋮ 🗑
✓	squid-proxy-1	e1abbd187ba4	proyecto-squid-proxy	3128:3128	0.02%	3 minutes ago	⌵ ⋮ 🗑
✓	fuel-api-1	ccb3c43d3ea9	proyecto-fuel-api		0.08%	3 minutes ago	⌵ ⋮ 🗑
✓	mongodb-1	3467fbd4bc31	mongo:latest	27017:27017	2.73%	3 minutes ago	⌵ ⋮ 🗑
✓	nginx-proxy-1	ad5ccfc63d35	proyecto-nginx-proxy	8080:80	0%	3 minutes ago	⌵ ⋮ 🗑

Ilustración 14. Contenedores en Docker

Después de levantar los contenedores utilizando `docker compose up -d`, pudimos acceder a la vista de Swagger para la API de Fuel en `localhost:8080/swagger`. Esta interfaz nos permitió interactuar de manera visual con los endpoints de la API, facilitando las pruebas y la verificación del correcto funcionamiento de los servicios expuestos.

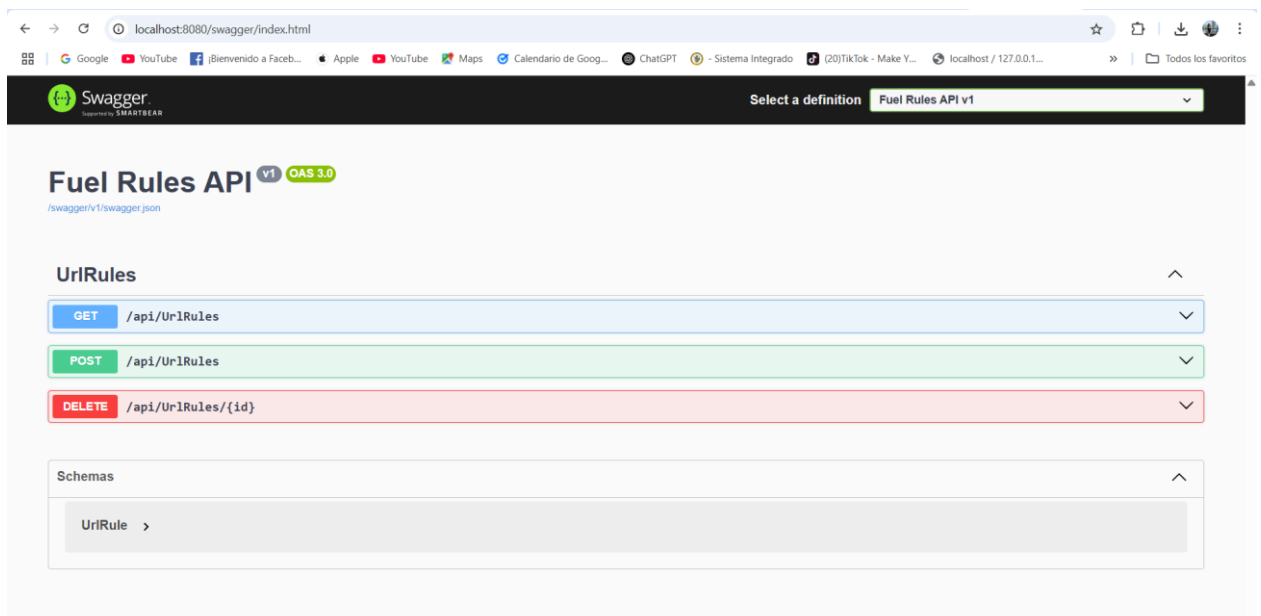


Ilustración 15. Interfaz de swagger para consumir la API



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: MARZO – JULIO 2025



Una vez configurada la API, se ingresaron datos específicos a través de la API, como "facebook" en el patrón de bloqueo para que Squid los gestionara. Al realizar una solicitud GET a la API, se pudo visualizar correctamente la entrada con "facebook" como un patrón bloqueado, lo que permitió confirmar que el valor se había ingresado correctamente y que las reglas estaban funcionando según lo esperado para bloquear el acceso a URLs relacionadas con Facebook.

The screenshot shows a REST client interface with a POST request to `/api/UrlRules`. The request body is a JSON object: `{ "pattern": "facebook", "isActive": true }`. The media type is set to `application/json`. There are "Cancel" and "Reset" buttons at the top right, and an "Execute" button at the bottom.

Ilustración 16. Creación de regla

The screenshot shows the response of the POST request. The status code is 200. The response body is `Regla agregada`. The response headers include `connection: keep-alive`, `content-type: text/plain; charset=utf-8`, `date: Sat, 12 Apr 2025 22:02:31 GMT`, `server: nginx/1.27.4`, and `transfer-encoding: chunked`. There is a "Download" button for the response body.

Ilustración 17. Conexión correcta a la base de datos

The screenshot shows the response of a GET request to `/api/UrlRules`. The status code is 200. The response body is a JSON array of three objects, each representing a blocked URL rule. The first object has `"pattern": "string"`, the second has `"pattern": "youtube"`, and the third has `"pattern": "facebook"`. All have `"isActive": true`. There is a "Download" button for the response body.

Ilustración 18. Reglas de bloqueo de rutas ingresadas



UNIVERSIDAD TÉCNICA DE AMBATO
FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL
CARRERA DE SOFTWARE
CICLO ACADÉMICO: MARZO – JULIO 2025



Se verificó que el servidor proxy Squid estaba en funcionamiento accediendo a localhost:3128 desde el navegador. La respuesta del servicio confirmó su actividad.

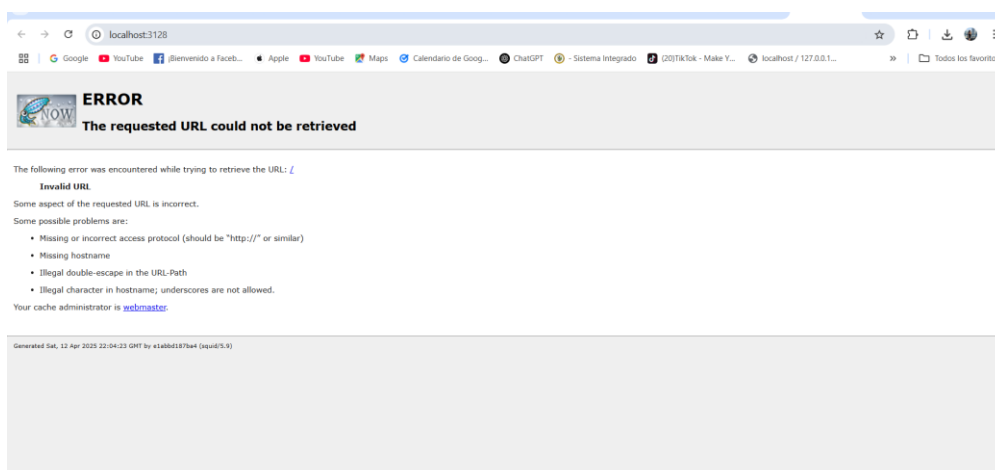


Ilustración 19. Servidor proxy funcionando

Se configuró el proxy manualmente en la máquina cliente el proxy para redirigir el tráfico web a través de localhost:3128. Esto permitió analizar las solicitudes HTTP/HTTPS registradas en los logs de Squid.

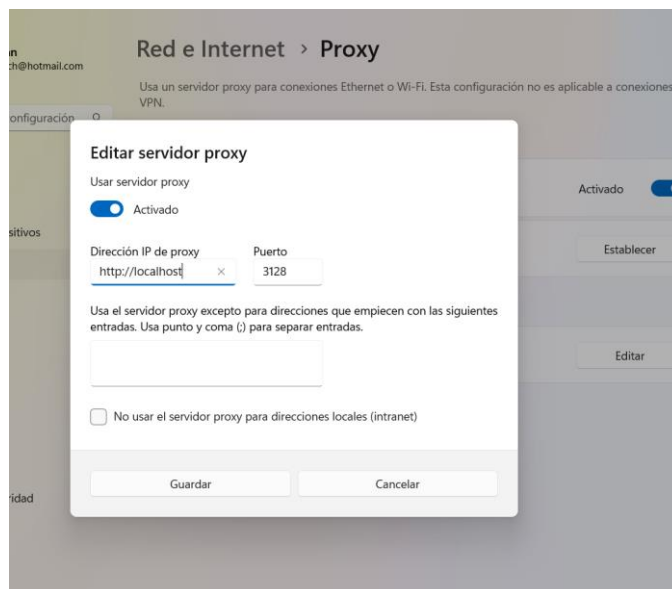


Ilustración 20. Configuración del Proxy en la maquina local

Los registros mostraron que algunas solicitudes fueron permitidas, como las de example.com (con respuestas TCP_NISS/200 y TCP_MEM_HIT/200) y las de td.doubleclick.net y cdn.segment.com (con conexiones TCP_TUNNEL/200, típicas de tráfico HTTPS); sin embargo, otras fueron bloqueadas, como las de www.youtube.com y www.facebook.com, las cuales devolvieron un código TCP_DENIED/403, lo que indica una denegación activa por parte del proxy conforme a sus políticas de restricción configuradas.



```
watch-squid.sh access.log X
squid-logs > access.log
1014 1744492358.780 29939 172.19.0.1 TCP_TUNNEL/200 39 CONNECT td.doubleclick.net:443 - HIER_DIRECT/172.217.15.194 -
1015 1744492358.781 29931 172.19.0.1 TCP_TUNNEL/200 39 CONNECT td.doubleclick.net:443 - HIER_DIRECT/172.217.15.194 -
1016 1744492358.782 29922 172.19.0.1 TCP_TUNNEL/200 39 CONNECT td.doubleclick.net:443 - HIER_DIRECT/172.217.15.194 -
1017 1744492358.783 29914 172.19.0.1 TCP_TUNNEL/200 39 CONNECT td.doubleclick.net:443 - HIER_DIRECT/172.217.15.194 -
1018 1744492358.785 29909 172.19.0.1 TCP_TUNNEL/200 39 CONNECT td.doubleclick.net:443 - HIER_DIRECT/172.217.15.194 -
1019 1744492360.633 34084 172.19.0.1 TCP_TUNNEL/200 39 CONNECT cdn.segment.com:443 - HIER_DIRECT/13.227.19.219 -
1020 1744492360.633 33777 172.19.0.1 TCP_TUNNEL/200 39 CONNECT cdn.segment.com:443 - HIER_DIRECT/13.227.19.219 -
1021 1744499450.726 340 172.19.0.1 TCP_MISS/200 1631 GET http://example.com/ - HIER_DIRECT/23.192.228.84 text/html
1022 1744499470.183 0 172.19.0.1 TCP_DENIED/403 3898 GET http://www.youtube.com/ - HIER_NONE/- text/html
1023 1744499485.288 0 172.19.0.1 TCP_MEM_HIT/200 1638 GET http://example.com/ - HIER_NONE/- text/html
1024 1744499498.516 0 172.19.0.1 TCP_DENIED/403 3901 GET http://www.facebook.com/ - HIER_NONE/- text/html
1025
```

Ilustración 21. Registro de logs exitosos para bloqueos

El mensaje de error obtenido al intentar acceder a Facebook mediante curl confirma la restricción, mostrando un mensaje claro de "Access Denied" generado por Squid. Estos hallazgos evidencian una configuración intencional del proxy para bloquear el acceso a Facebook, posiblemente como parte de políticas de control de tráfico o seguridad.

```
C:\Users\fredd>
C:\Users\fredd>curl -x http://localhost:3128 http://www.facebook.com
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html><head>
<meta type="copyright" content="Copyright (C) 1996-2020 The Squid Software Foundation and contributors">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>ERROR: The requested URL could not be retrieved</title>
<style type="text/css"><!--
/*
 * Copyright (C) 1996-2023 The Squid Software Foundation and contributors
 *
 * Squid software is distributed under GPLv2+ license and includes
 * contributions from numerous individuals and organizations.
 * Please see the COPYING and CONTRIBUTORS files for details.
 */

```

Ilustración 22. Proxy funcionando para el bloqueo de Facebook

2.8 Habilidades blandas empleadas en la práctica

- ☐ Liderazgo
- ☒ Trabajo en equipo
- ☒ Comunicación asertiva
- ☒ La empatía
- ☒ Pensamiento crítico
- ☒ Flexibilidad
- ☒ La resolución de conflictos
- ☐ Adaptabilidad
- ☒ Responsabilidad

2.9 Conclusiones

La práctica permitió comprender e implementar los fundamentos de los sistemas distribuidos mediante la integración de herramientas como Docker, Squid Proxy, ASP.NET Core y MongoDB, logrando desarrollar un entorno funcional donde la API controla de manera dinámica las reglas de bloqueo de URLs. A través de esta arquitectura distribuida, se evidenció la efectividad del uso de microservicios, contenedores y redes internas para mantener una comunicación eficiente entre los componentes del sistema, cumpliendo con los objetivos planteados y demostrando una aplicación práctica de los conceptos teóricos de la asignatura.



2.10 Recomendaciones

Se recomienda continuar profundizando en la gestión de microservicios y contenedores, especialmente en el uso avanzado de Docker, explorando aspectos como la orquestación con Docker Compose, la optimización de imágenes y el manejo de volúmenes y redes personalizadas. Esto permitirá fortalecer la infraestructura desarrollada y mejorar la escalabilidad y mantenibilidad de los servicios. Además, aplicar este tipo de prácticas en proyectos reales o académicos más exigentes contribuirá significativamente al desarrollo de competencias técnicas y a la comprensión integral de los sistemas distribuidos.

2.11 Referencias bibliográficas

<https://github.com/melanieAlban/aplicaciones-distribuidas.git>
<https://github.com/alisonMSalas/AplicacionesDistribuidas.git>
<https://github.com/rafaelsoriano04/aplicaciones-distribuidas>
<https://github.com/FreddyA12/aplicaciones-distribuidas>