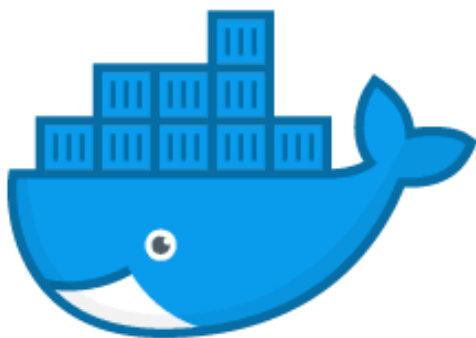

Microservicios:
Spring Boot, Java, JPA, Hibernate, Gradle,
Docker, Docker Hub.



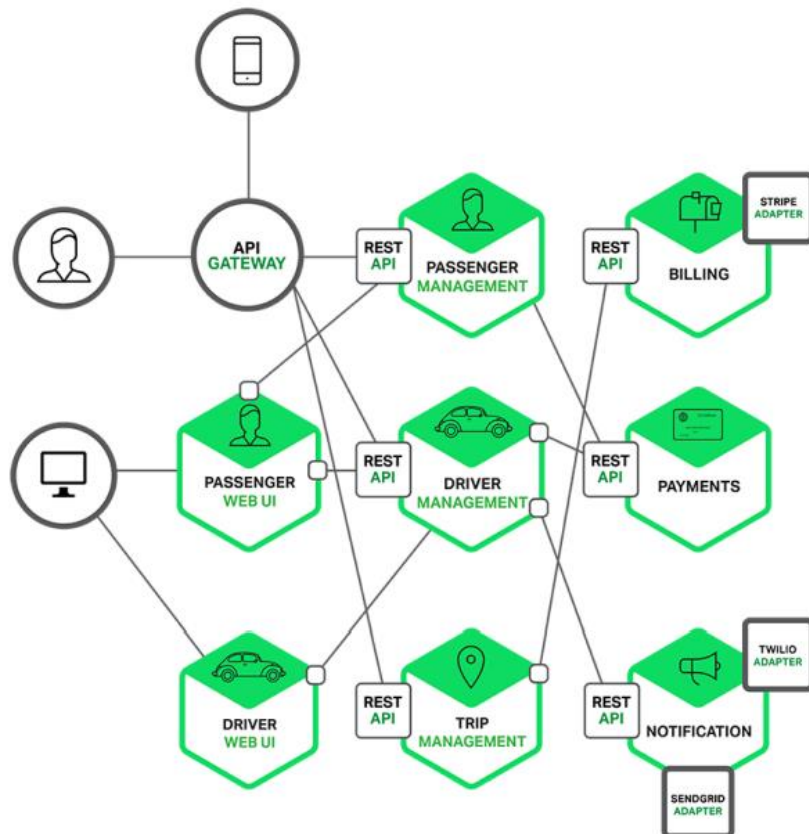
docker



FICHA TECNICA:

1. ESTRUCTURA DE APLICACIÓN

La arquitectura basada en microservicios puede tener tantas capas, servicios o componentes, como se requieran. Generalmente, las capas básicas de esta arquitectura son: API Gateway, Service Discovery, Servicios y sus correspondientes bases de datos. Ejemplo



2. MODELO

- a. **Personas:** Son las personas dispuestas a solicitar un préstamo
- Entidad

```
package persona.entity;

import lombok.Data;

import javax.persistence.*;
import java.util.Date;

@Entity
@Table(name = "persona")
@Data
public class Persona {
    @Id
    @Column(unique = true, name = "dni")
    private String dni;
    private String nombres;
    private String correo;
    @Temporal(TemporalType.TIMESTAMP)
    private Date fecha;
```

```

    @PrePersist
    public void perPersist(){
        this.fecha = new Date();
    }
}

```

- Controlador

```

package persona.controller;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import persona.entity.Persona;
import persona.service.PersonaService;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

@RestController
@RequestMapping(value = "persona")
public class PersonaController {
    @Autowired
    private PersonaService personaService;

    @PostMapping(value = "crear")
    public ResponseEntity<Persona> crearPersona(@RequestBody Persona persona){
        System.out.println(persona);
        Persona objPersona = personaService.createPersona(persona);

        if(Objects.isNull(objPersona)){
            return ResponseEntity.noContent().build();
        }

        return ResponseEntity.status(HttpStatus.CREATED).body(objPersona);
    }

    @GetMapping(value = "listar")
    public ResponseEntity<List<Persona>> listarPersonas(){
        List<Persona> objPersona = new ArrayList<>();
        objPersona = personaService.listar();
        return ResponseEntity.ok(objPersona);
    }

    @GetMapping(value =("/{dni}")
    public ResponseEntity<Persona> getPersona(@PathVariable("dni") String dni){
        Persona objPersona;
        objPersona = personaService.getPersona(dni);
        if(Objects.isNull(objPersona)){
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.ok(objPersona);
    }

    @DeleteMapping(value = "eliminar")
    public ResponseEntity<Object> deleteSolicitudes(@RequestParam(value = "dni") String dni){
        String msj = "";
        if(null == dni){
            return ResponseEntity.noContent().build();
        }else {
            msj = personaService.deletePersona(dni);
            if(msj == null){

```

```

        return ResponseEntity.noContent().build();
    }
}
ObjectMapper mapper = new ObjectMapper();
ObjectNode objMsg = mapper.createObjectNode();
objMsg.put("mensaje", "La persona con el DNI: "+dni+" ha sido eliminada.");

return ResponseEntity.ok(objMsg);
}
}

```

- b. Agente Crédito:** Es el encargado de registrar, actualizar, aprobar, rechazar y eliminar una solicitud de préstamo, así mismo con los datos de las personas.
- c. Solicitudes:** Representan los datos que intercambian los agentes de crédito con las personas, contiene los registros de solicitudes de prestamos
 - **Entidad**

```

package solicitud.entity;

import lombok.Data;
import solicitud.model.Persona;

import javax.persistence.*;
import java.util.Date;

@Entity
@Table(name = "solicitud")
@Data
public class Solicitud {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    private Double monto;
    private Integer cuotas;
    private String estado;
    @Temporal(TemporalType.TIMESTAMP)
    private Date fecha;
    @Column(name = "persona_id")
    private String personaId;

    @Transient
    private Persona persona;

    @PrePersist
    public void prePersist(){
        this.fecha = new Date();
    }
}

```

- Controlador

```

package solicitud.controller;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

```

```

import org.springframework.web.bind.annotation.*;
import solicitud.entity.Solicitud;
import solicitud.service.SolicitudService;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

@RestController
@RequestMapping(value = "solicitud")
public class SolicitudController {
    @Autowired
    private SolicitudService solicitudService;

    @PostMapping(value = "crear")
    public ResponseEntity<Solicitud> saveSolicitudes(@RequestBody Solicitud solicitud){
        Solicitud solicitudes = solicitudService.createSolicitud(solicitud);
        if(Objects.isNull(solicitudes)){
            return ResponseEntity.noContent().build();
        }

        return ResponseEntity.status(HttpStatus.CREATED).body(solicitud);
    }

    @GetMapping(value = "listar")
    public ResponseEntity<List<Solicitud>> listSolicitudes(){
        List<Solicitud> solicitudes = new ArrayList<>();
        solicitudes = solicitudService.listar();
        return ResponseEntity.ok(solicitudes);
    }

    @GetMapping(value = "consultar")
    public ResponseEntity<List<Solicitud>> listSolicitudes(@RequestParam(value = "dni") String dni){
        List<Solicitud> solicitudes = new ArrayList<>();
        solicitudes = solicitudService.listarAll(dni);
        return ResponseEntity.ok(solicitudes);
    }

    @PutMapping(value = "actualizar")
    public ResponseEntity<Solicitud> updateSolicitud(@RequestParam(value = "id") Integer id,
    @RequestParam(value = "estado") String estado){
        Solicitud sol = solicitudService.updateSolicitud(id, estado);
        if(Objects.isNull(sol)){
            return ResponseEntity.noContent().build();
        }

        return ResponseEntity.ok(sol);
    }

    @DeleteMapping(value = "eliminar")
    public ResponseEntity<Object> deleteSolicitudes(@RequestParam(value = "dni") Integer dni){
        String msj = "";
        if(null == dni){
            return ResponseEntity.noContent().build();
        }else {
            msj = solicitudService.deleteSolicitud(dni);
            if(msj == null){
                return ResponseEntity.noContent().build();
            }
        }

        ObjectMapper mapper = new ObjectMapper();
        ObjectNode objMsg = mapper.createObjectNode();
        objMsg.put("dni", dni);
        return ResponseEntity.ok(objMsg);
    }

    @DeleteMapping(value = "eliminar-todo")

```

```

    public ResponseEntity<Object> deleteSolicitudes(){
        String msj = "Se ha eliminado todo.";
        solicitudService.deleteSolicitudes();
        ObjectMapper mapper = new ObjectMapper();
        ObjectNode objMsg = mapper.createObjectNode();
        objMsg.put("msj", msj);
        return ResponseEntity.ok(objMsg);
    }
}

```

3. BASE DE DATOS

- a) **SGBD:** PostgreSQL version 14
- b) **Funcionalidad:** Almacenar y administrar los datos de la App.
- c) **PORT:** 5432
- d) **Cliente:** PgAdmin
- e) **Nombre Base de Datos:** DB_CrediFast
- f) **Usuario:** -----
- g) **Contraseña:** -----
- i) **Host:** -----

SGBD	PostgreSQL
CLIENTE DB	PgAdmin
USER	-----
CONTRASEÑA	-----
PUERTO	5432
HOST	-----
BASE DE DATOS	DB_CrediFast

4. SERVICIOS

a. CONFIG-SERVICE

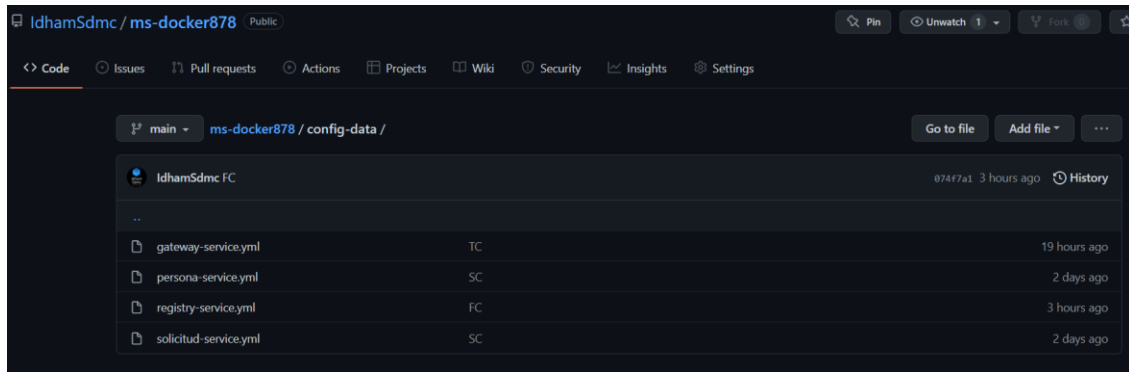
- ✓ **Descripción:** Este servicio está a la espera de recibir peticiones por parte del resto de servicios, para que se les proporcione la configuración necesaria para su ejecución en el entorno definido. Cuando este servicio recibe estas peticiones, recoge los archivos de configuración para cada servicio (de extensión ".yaml" o ".properties") de un repositorio definido para luego proporcionárselo al servicio que haya realizado la petición.
- ✓ **Tecnología:** Spring-Boot y Spring-Cloud: Spring Cloud Config Server sobre Java 11
- ✓ **Funcionalidad:** Busca sobre un repositorio los archivos de configuración de cada servicio que los solicite.
- ✓ **Dependencias**

```

dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.cloud:spring-cloud-config-server'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
}

```

✓ Repositorio de Configuración



✓ Archivo bootstrap.yml

```
server:
  port: 8081
spring:
  application:
    name: config-service
  cloud:
    config:
      server:
        git:
          uri: https://github.com/IdhamSdmc/ms-docker878.git
          searchPaths: config-data
          username: ${GIT_USER}
          password: ${GIT_PASSWORD}
          force-pull: true
          clone-on-start: true
          default-label: "main"
        repos:
          develop:
            uri: https://github.com/IdhamSdmc/ms-docker878.git
            force-pull: true
            clone-on-start: true
            searchPaths: config-data
            default-label: main
  security:
    user:
      name: root
      password: s3cr3t
```

b. DISCOVERY-SERVICE

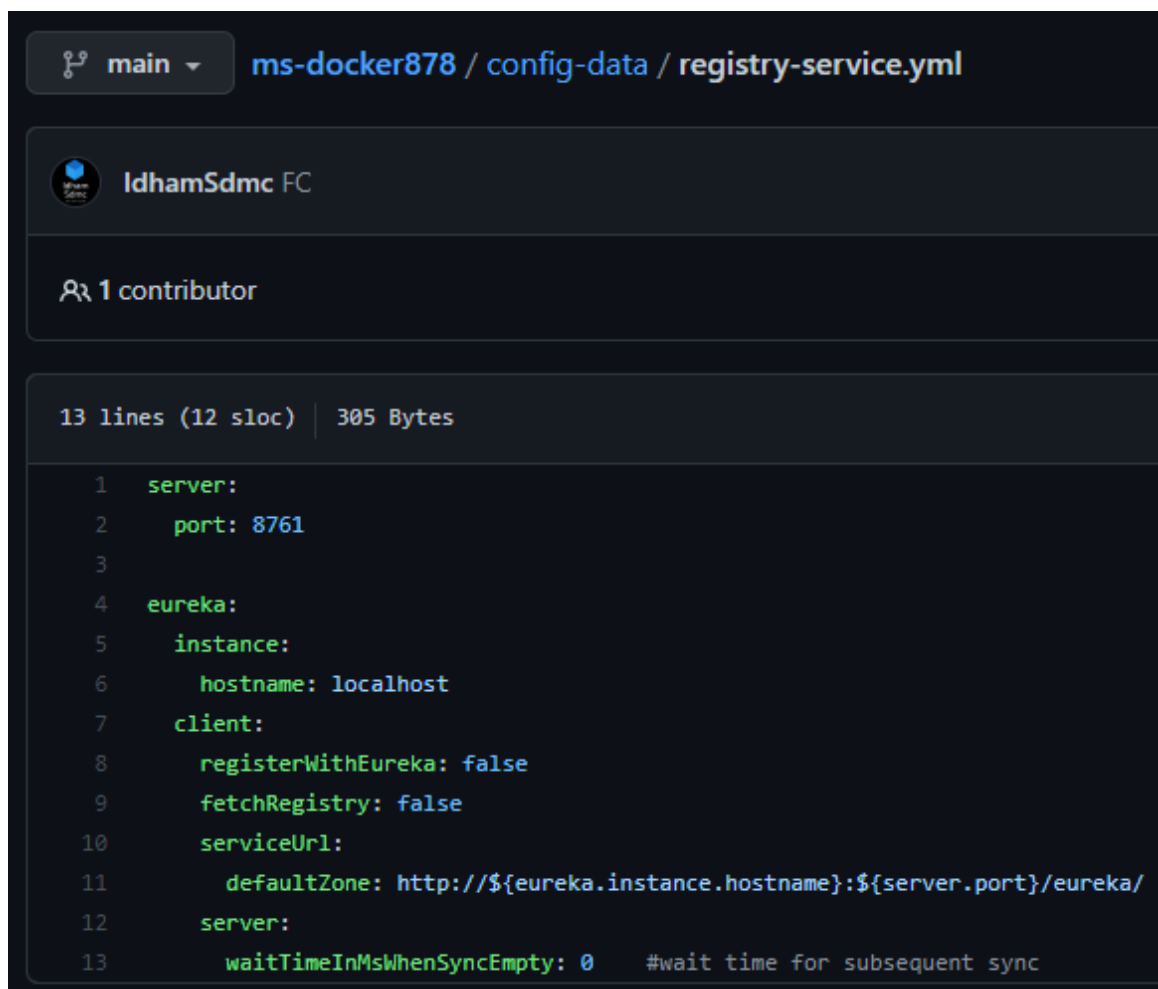
- ✓ **Descripción:** Servicio encargado de registrar las direcciones a los microservicios que componen la aplicación y redireccionar las peticiones hacia estos.
- ✓ **Tecnología:** Spring-Boot, Spring-Cloud y Netflix Eureka sobre Java 11
- ✓ **Funcionalidad:** Encargado de redirigir las llamadas realizadas a cada servicio a través de una URI genérica a la dirección del servidor en el que se encuentra dicho servicio.
- ✓ **Dependencias**

```
dependencies {
  implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'
  implementation 'org.springframework.cloud:spring-cloud-starter-config'
  implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'
  testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

✓ *Archivo bootstrap.yml*

```
server:
  port: 8761 # Indicate the default PORT where this service will be started
spring:
  application:
    name: registry-service
  cloud:
    config:
      name: registry-service
      uri: http://config-service:8081
      username: root
      password: s3cr3t
      retry:
        max-attempts: 10
        initial-interval: 5000
      fail-fast: true
```

✓ *Repositorio Archivo .yml*



The screenshot shows a GitHub repository interface. At the top, the repository path is 'ms-docker878 / config-data / registry-service.yml'. Below this, the repository owner 'IdhamSdmc FC' is shown with a profile picture. Underneath, it says '1 contributor'. The file statistics show '13 lines (12 sloc)' and '305 Bytes'. The code content is displayed in a dark-themed editor with line numbers 1 through 13. The code is a YAML configuration for a Spring Cloud application.

```
1  server:
2    port: 8761
3
4  eureka:
5    instance:
6      hostname: localhost
7    client:
8      registerWithEureka: false
9      fetchRegistry: false
10   serviceUrl:
11     defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
12   server:
13     waitTimeInMsWhenSyncEmpty: 0    #wait time for subsequent sync
```


C. GATEWAY-SERVICE

- ✓ **Descripción** Servicio encargado de centralizar las llamadas a los demás servicios a través de una URI que hace de entrada de peticiones.
- ✓ **Tecnología:** Api Rest, Spring-Boot, Spring-Cloud Gateway sobre Java 11
- ✓ **Funcionalidad:** Se encarga de centralizar las llamadas a la aplicación en una URI principal que redirige las llamadas a los servicios configurados internamente.
- ✓ **Dependencias**

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'  
    implementation 'org.springframework.cloud:spring-cloud-starter-config'  
    implementation 'org.springframework.cloud:spring-cloud-starter-gateway'  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

✓ Archivo bootstrap.yml

```
spring:  
  application:  
    name: gateway-service  
  cloud:  
    config:  
      name: gateway-service  
      uri: http://config-service:8081  
      username: root  
      password: s3cr3t  
      retry:  
        max-attempts: 10  
        initial-interval: 5000  
      fail-fast: true
```

✓ Repositorio Archivo .yml

```
1  server:  
2    port: 8099  
3  
4  eureka:  
5    client:  
6      serviceUrl:  
7        defaultZone: http://localhost:8761/eureka/  
8  
9  spring:  
10    cloud:  
11      gateway:  
12        default-filters:  
13          - DedupeResponseHeader=Access-Control-Allow-Credentials Access-Control-Allow-Origin  
14        globalcors:  
15          corsConfigurations:  
16            '[/**]':  
17              allowedOrigins: "*"   
18              allowedMethods: "*"   
19              allowedHeaders: "*"   
20        discovery:  
21          locator:  
22            enabled: true  
23        routes:  
24          - id: persona-service  
25            uri: http://localhost:8091/  
26            predicates:  
27              - Path=/persona/**  
28          - id: solicitud-service  
29            uri: https://localhost:8092/  
30            predicates:  
31              - Path=/solicitud/**
```

d. MS-PERSONA

- ✓ **Descripción** Este servicio soporta las peticiones referidas al registro de una persona.
- ✓ **Tecnología:** Api Rest, Spring-Boot, Spring-MVC Lombok, JPA, Hibernate sobre Java 11
- ✓ **Funcionalidad:** Creación, modificación, borrado y lectura de datos.
- ✓ **Dependencias**

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.cloud:spring-cloud-starter-config'  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'  
    implementation 'io.github.kilmajster:ngrok-spring-boot-starter:0.5.0'  
    compileOnly 'org.projectlombok:lombok'  
    runtimeOnly 'org.postgresql:postgresql'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

✓ Archivo bootstrap.yml

```
spring:  
  application:  
    name: persona-service  
  cloud:  
    config:  
      #Ruta donde corre el config-service  
      uri: http://config-service:8081  
      username: root  
      password: s3cr3t  
      retry:  
        max-attempts: 10  
        initial-interval: 5000  
        fail-fast: true  
  ngrok:  
    enabled: true  
  eureka:  
    hostname: persona-service  
    instance:  
      status-page-url: http://persona-service:8091/persona/listar
```

✓ *Repositorio Archivo .yml*

```
1  # Spring Boot configuration
2  spring:
3    profiles:
4      active: development
5    # Security configuration
6    security:
7      user:
8        name: user
9        password: user
10   # Database
11   datasource:
12     driver-class-name: org.postgresql.Driver
13     url: jdbc:postgresql://209.145.60.40/DB_CrediFast?stringtype=unspecified
14     username: UserProjects8
15     password: SisProjects8Pass
16   # JPA properties
17   jpa:
18     hibernate:
19       ddl-auto: update # When you launch the application for the first time - switch "none" at "create"
20     show-sql: true
21     database: postgresql
22     database-platform: org.hibernate.dialect.PostgreSQLDialect
23     open-in-view: false
24     generate-ddl: true
25   # Logger configuration
26   logging:
27     pattern:
28       console: "%d %-5level %logger : %msg%n"
29     level:
30       org.springframework: info
31       org.hibernate: debug
32   # Server configuration
33   server:
34     port: 8091
35     error:
36       include-message: always
37       include-binding-errors: always
38   eureka:
39     client:
40       serviceUrl:
41         defaultZone: http://localhost:8761/eureka/
```

e. MS-SOLICITUD

- ✓ **Descripción** Este servicio soporta las peticiones referidas al registro de una solicitud.
- ✓ **Tecnología:** Api Rest, Spring-Boot, Spring-MVC Lombok, JPA, Hibernate sobre Java 11
- ✓ **Funcionalidad:** Creación, modificación, borrado y lectura de datos.
- ✓ **Dependencias**

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-actuator'  
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.cloud:spring-cloud-starter-config'  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'  
    implementation 'io.github.kilmajster:ngrok-spring-boot-starter:0.5.0'  
    implementation 'org.springframework.cloud:spring-cloud-starter-openfeign'  
    implementation 'org.springframework.cloud:spring-cloud-starter-circuitbreaker-resilience4j'  
    implementation 'org.springframework.boot:spring-boot-starter-aop:2.6.1'  
    compileOnly 'org.projectlombok:lombok'  
    runtimeOnly 'org.postgresql:postgresql'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

✓ Archivo bootstrap.yml

```
spring:  
  application:  
    name: solicitud-service  
  cloud:  
    config:  
      uri: http://config-service:8081  
      username: root  
      password: s3cr3t  
      retry:  
        max-attempts: 10  
        initial-interval: 5000  
      fail-fast: true  
  
ngrok:  
  enabled: true  
  
eureka:  
  hostname: solicitud-service  
  instance:  
    status-page-url: http://solicitud-service:8092/solicitud/listar
```

✓ Repositorio Archivo .yml

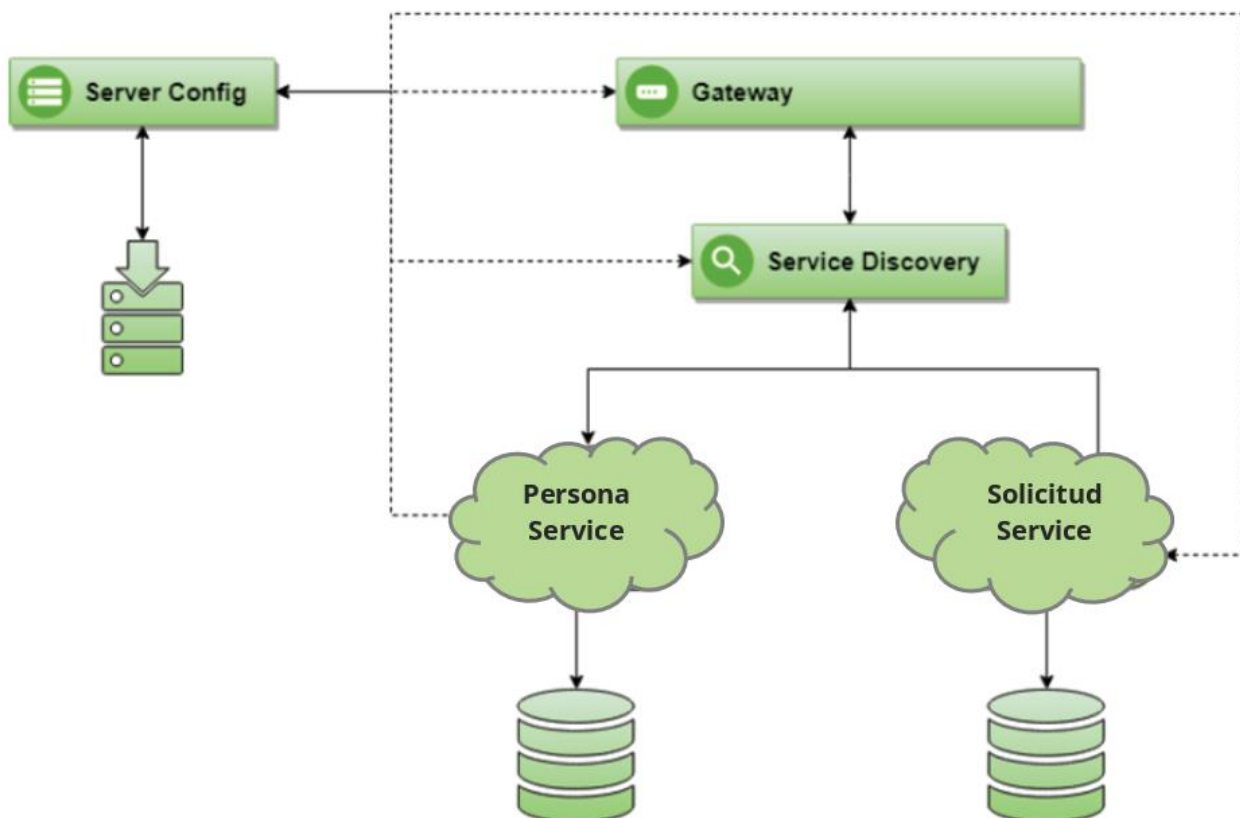
```
1 # Spring Boot configuration
2 spring:
3   profiles:
4     active: development
5   # Security configuration
6   security:
7     user:
8       name: user
9       password: user
10  # Database
11  datasource:
12    driver-class-name: org.postgresql.Driver
13    url: jdbc:postgresql://209.145.60.40/DB_CrediFast?stringtype=unspecified
14    username: UserProjects8
15    password: SisProjects8Pass
16  # JPA properties
17  jpa:
18    hibernate:
19      ddl-auto: update # When you launch the application for the first time - switch "none" at "create"
20      show-sql: true
21      database: postgresql
22      database-platform: org.hibernate.dialect.PostgreSQLDialect
23      open-in-view: false
24      generate-ddl: true
25  # Logger configuration
26  logging:
27    pattern:
28      console: "%d %-5level %logger : %msg%n"
29    level:
30      org.springframework: info
31      org.hibernate: debug
32  # Server configuration
33  server:
34    port: 8092
35    error:
36      include-message: always
37      include-binding-errors: always
38  eureka:
39    client:
40      serviceUrl:
41        defaultZone: http://localhost:8761/eureka/
```

```

42 resilience4j:
43   circuitbreaker:
44     instances:
45       personaCB:
46         registerHealthIndicator: true
47         slidingWindowSize: 10
48         permittedNumberOfCallsInHalfOpenState: 3
49         slidingWindowType: TIME_BASED
50         minimumNumberOfCalls: 4
51         waitDurationInOpenState: 50s
52         failureRateThreshold: 50
53         eventConsumerBufferSize: 10
54 management:
55   health:
56     circuitbreakers:
57       enabled: true
58   endpoints:
59     web:
60       exposure:
61         include: health
62   endpoint:
63     health:
64       show-details: always

```

5. ARQUITECTURA DE APLICACIÓN



6. DESPLIEGUE – DOCKER Y CONTENEDORES

Previamente es necesario instalar la aplicación **Docker desktop**, o lo que es lo mismo, la aplicación de escritorio de Docker. Se puede acceder a su descarga a través del siguiente enlace <https://www.docker.com/get-started>, y aunque su uso es gratuito, será necesario que nos registremos para poder descargarlo.

Generalmente, para la creación de las imágenes, existe la opción de incluir, manualmente los archivos o archivo **DockerFile**, para lo cual se le deben indicar la/las rutas.

A. Comandos Importantes:

docker images → Muestra todas las imágenes conocidas

docker ps → Muestra todos los contenedores en ejecución

docker run -p PORT:PORT image_name → Ejecuta una imagen Docker dentro de un contenedor creado según las especificaciones de la imagen en sí misma.

docker stop <CONTAINER_ID> → Para la ejecución del contenedor con la ID introducida

B. Creación de Imágenes mediante Dockerfile

a. im-config-service

```
FROM openjdk:11
COPY ./build/libs/config-service-0.0.1-SNAPSHOT.jar "config-
service.jar"
EXPOSE 8081
ENTRYPOINT ["java", "-jar", "/config-service.jar"]
docker build -t im-config-service .
```

b. im-registry-service

```
FROM openjdk:11
COPY ./build/libs/registry-service-0.0.1-SNAPSHOT.jar "
registry-service.jar"
EXPOSE 8761
ENTRYPOINT ["java", "-jar", "/registry-service.jar"]
docker build -t im-registry-service .
```

c. im-gateway-service

```
FROM openjdk:11
COPY ./build/libs/gateway-service-0.0.1-SNAPSHOT.jar "gateway-
service.jar"
EXPOSE 8099
ENTRYPOINT ["java", "-jar", "/gateway-service.jar"]
docker build -t im-gateway-service .
```

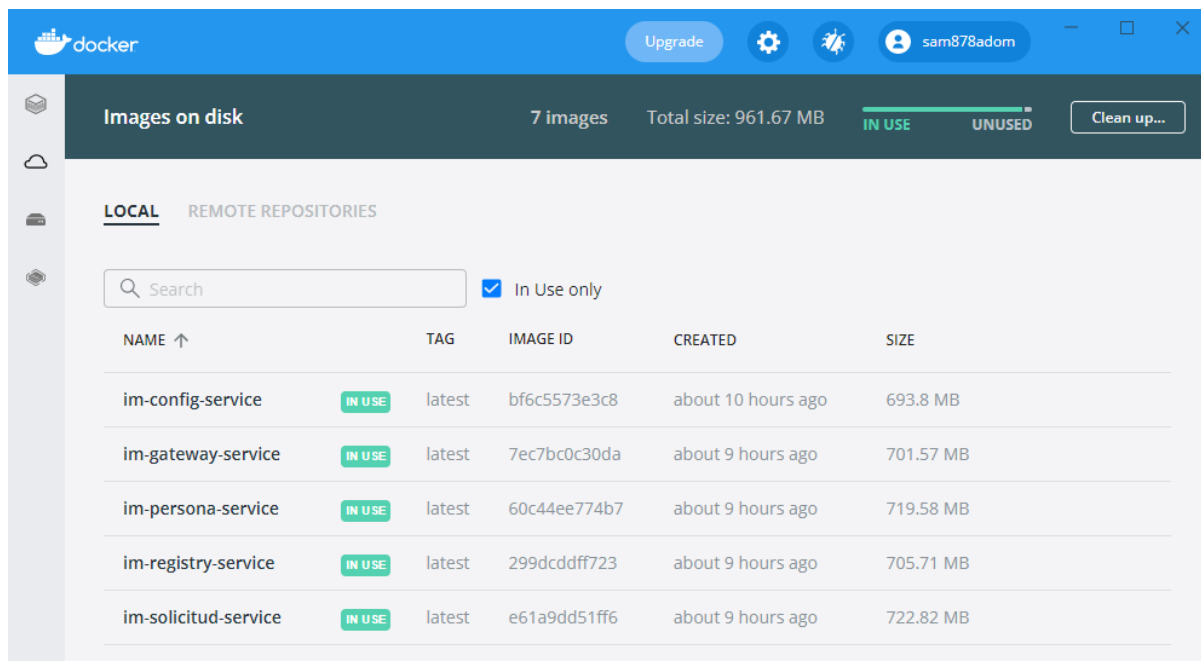
d. im-persona-service

```
FROM openjdk:11
ADD ./build/libs/service-persona-0.0.1-SNAPSHOT.jar "ms-
persona.jar"
EXPOSE 8091
ENTRYPOINT ["java", "-jar", "/ms-persona.jar"]
docker build -t im-persona-service .
```

e. im-solicitud-service

```
FROM openjdk:11
ADD ./build/libs/service-solicitud-0.0.1-SNAPSHOT.jar "ms-
solicitud.jar"
EXPOSE 8092
ENTRYPOINT ["java", "-jar", "/ms-solicitud.jar"]
docker build -t im-solicitud-service .
```

Después de haber creado las imágenes correspondientes de cada microservicio podemos visualizar las imágenes creadas en la herramienta de Docker-Desktop.



La segunda opción, es algo más avanzada, y requiere del uso de una herramienta más compleja llamada **Docker-compose**. En nuestro caso, esta será la mejor opción, y el archivo **Docker-compose** de nuestra aplicación tendrá el siguiente aspecto:

```
version: '3.3'
services:
#CONFIG - SERVER
  config-service:
    image: im-config-service:latest
    container_name: config-service
    ports:
      - "8081:8081"
    expose:
      - 8081
#EUREKA SERVER - DISCOVERY SERVER
  registry-service:
    image: im-registry-service:latest
    container_name: registry-service
    ports:
      - "8761:8761"
    expose:
      - 8761
    depends_on:
      - config-service
    links:
      - config-service
    environment:
      SPRING_CLOUD_CONFIG_URI: http://config-service:8081
      EUREKA_URI: http://registry-service:8761/eureka
#MICROSERVICIO GATEWAY
```



```

gateway-service:
  image: im-gateway-service:latest
  container_name: gateway-service
  ports:
    - "8099:8099"
  expose:
    - 8099
  depends_on:
    - config-service
    - registry-service
  links:
    - config-service
    - registry-service
  environment:
    SPRING_CLOUD_CONFIG_URI: http://config-service:8081
    EUREKA_URI: http://registry-service:8761/eureka

#MICROSERVICIO PERSONA
persona-service:
  image: im-persona-service:latest
  container_name: persona-service
  ports:
    - "8091:8091"
  expose:
    - 8091
  depends_on:
    - config-service
    - registry-service
  links:
    - config-service
    - registry-service
  environment:
    SPRING_CLOUD_CONFIG_URI: http://config-service:8081
    EUREKA_URI: http://registry-service:8761/eureka

#MICROSERVICIO SOLICITUD
solicitud-service:
  image: im-solicitud-service:latest
  container_name: solicitud-service
  ports:
    - "8092:8092"
  expose:
    - 8092
  depends_on:
    - config-service
    - registry-service
  links:
    - config-service
    - registry-service
  environment:
    SPRING_CLOUD_CONFIG_URI: http://config-service:8081
    EUREKA_URI: http://registry-service:8761/eureka

```

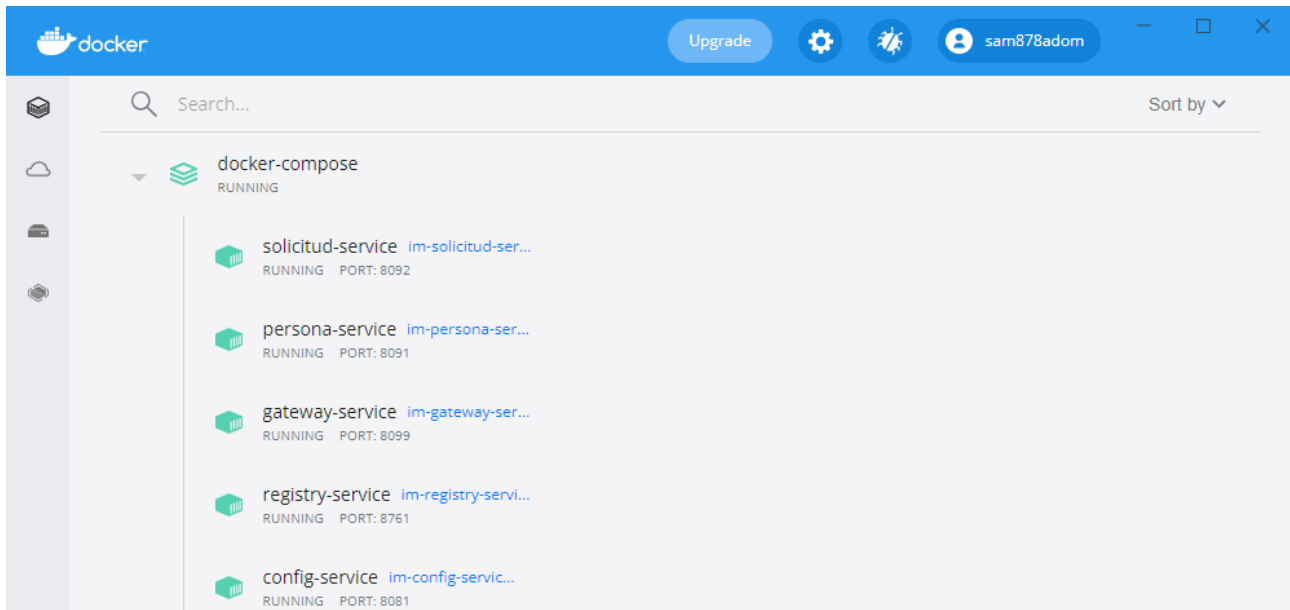
Para la ejecución de la aplicación a través de este método, será necesario navegar, a través de la consola de comandos, hasta la carpeta o directorio donde se encuentre dicho archivo y, una vez allí, deberemos ejecutar el comando **docker-compose up**.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

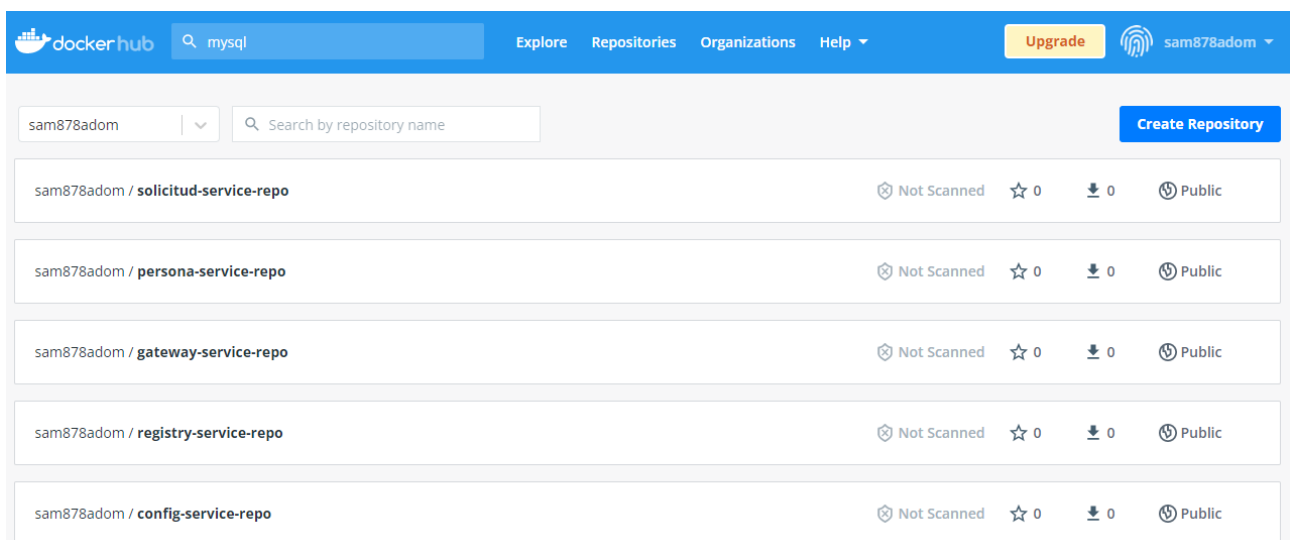
PS D:\VIII CICLO\Arquitectura de Micro - Servicios\Docke-Compose> docker-compose up
```

Este comando ejecutará cada imagen en un contenedor de acuerdo con el archivo **Docker-compose.yml**.



C. USO DE DOCKER-HUB PARA SUBIR IMÁGENES

DockerHub es el repositorio del cual Docker toma las imágenes mediante las cual provisiona nuestros contenedores en donde nuestra aplicación reside. Toma en cuenta que para poder explorar el DockerHub es necesario crear una cuenta dentro del sitio. Para subir las imágenes que hemos creado, primero en nuestra cuenta crear un repositorio para cada microservicio, luego debemos definir un **tag** o **etiqueta** a cada imagen de la siguiente manera.



- Añadiendo tag a las imágenes

im-config-service:

```
docker tag image sam878adom/registry--server-repo:liters-v1
```

im-registry-service:

```
docker tag image sam878adom/gateway-server-repo:litegs-v1
```

im-gateway-service:

```
docker tag image sam878adom/registry--server-repo:liters-v1
```

im-persona-service:

```
docker tag image sam878adom/persona-server-repo:liteps-v1
```

im-solicitud-service:

```
docker tag image sam878adom/solicitud-server-repo:litess-v1
```

- Haciendo Push de las imágenes hacia el repositorio de Docker-Hub

im-config-service:

```
docker push sam878adom/config-server-repo:litecs-v1
```

im-registry-service:

```
docker push sam878adom/registry--server-repo:liters-v1
```

im-gateway-service:

```
docker push sam878adom/gateway-server-repo:litegs-v1
```

im-persona-service:

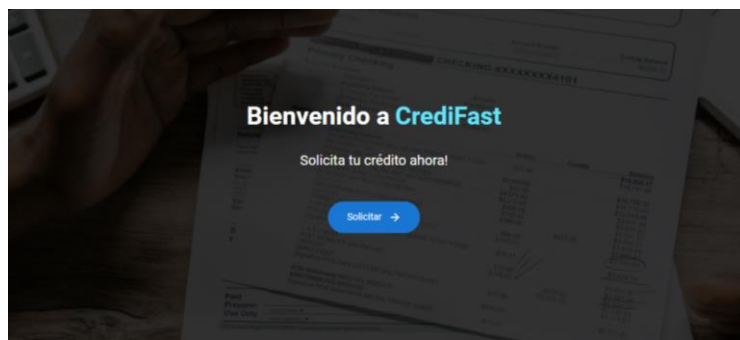
```
docker push sam878adom/persona-server-repo:liteps-v1
```

im-solicitud-service:

```
docker push sam878adom/solicitud-server-repo:litess-v1
```

7. SPA

- j) a) IDE: visual Studio code v1.61.2
- k) b) Tecnología: React v.17.0.2
- l) c) Archivos de lectura y escritura: JSON
- m) d) Funcionalidad: Registrar cliente y su solicitud.
- n) e) URL: <http://a1ad-181-64-57-212.ngrok.io>



Inicio Crédito

<div> <div>Registrar Persona</div> <div> <input type="text"/> Consultar Solicitud <div>Registrar Solicitud</div> </div> </div>										
Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar

Inicio Crédito

Registrar Persona Consultar Solicitud Registrar Solicitud

Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar

Registrar Persona

DNI * 87654321 Nombres * Luna Estrella Venus Marte

Correo Electrónico * luna@estrella.com

CERRAR REGISTRAR

Inicio Crédito

Registrar Persona Consultar Solicitud Registrar Solicitud

Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar

Registrar Solicitud

DNI * 87654321 Monto Solicitado * 50000 Cuotas * 25

CERRAR REGISTRAR

Inicio Crédito

Registrar Persona Consultar Solicitud Registrar Solicitud

Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
28	87654321	Luna Estrella Venus Marte	luna@estrella.com	50000	25	PENDIENTE	3/2/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar

Inicio Crédito

Registrar Persona Consultar Solicitud Registrar Solicitud

Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
28	87654321	Luna Estrella Venus Marte	luna@estrella.com	50000	25	PENDIENTE	3/2/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar

Consultar Solicitud

Ingrese un DNI 87654321

CONSULTAR

CERRAR

Inicio Crédito

Registrar Persona Consultar Solicitud Registrar Solicitud

Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
28	87654321	Luna Estrella Venus Marte	luna@estrella.com	50000	25	PENDIENTE	3/2/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar

Consultar Solicitud

Ingrese un DNI
87654321

CONSULTAR

Nombres: Luna Estrella Venus Marte
DNI: 87654321
Monto: 50000 Cuotas: 25
Fecha Solicitud: 3/2/2022


PENDIENTE

CERRAR

Inicio Crédito

Registrar Persona Consultar Solicitud Registrar Solicitud

Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar
28	87654321	Luna Estrella Venus Marte	luna@estrella.com	50000	25	APROBADA	3/2/2022	<input checked="" type="checkbox"/> Aprobar	<input type="checkbox"/> Rechazar	<input type="checkbox"/> Eliminar



Actualizado!

La solicitud con el N° Operación: 28 ha sido aprobada.

OK

Inicio Crédito

Registrar Persona

Consultar Solicitud

Registrar Solicitud

Nro.Op.	DNI	Nombres	Correo	Monto(S/)	Cuotas	Estado	Fecha	Aprobar	Rechazar	Eliminar
24	70610058	Waldir Sanchez	wsanchezp@unitru.edu.pe	4500	7	APROBADA	19/1/2022	<div><div></div>Aprobar</div>	<div><div></div>Rechazar</div>	<div><div></div>Eliminar</div>
26	12345678	Cesar Perez	cperez@gmail.com	4000	5	RECHAZADA	27/1/2022	<div><div></div>Aprobar</div>	<div><div></div>Rechazar</div>	<div><div></div>Eliminar</div>
28	87654321	Luna Estrella Venus Marte	luna@estrella.com	50000	25	APROBADA	3/2/2022	<div><div></div>Aprobar</div>	<div><div></div>Rechazar</div>	<div><div></div>Eliminar</div>

REFERENCIAS BIBLIOGRÁFICAS

Microservicios

Tutorial sobre Microservicios: <https://www.guru99.com/microservices-tutorial.html>

Artículo sobre la arquitectura de Microservicios:

<https://microservices.io/patterns/microservices.html>

Guía sencilla para crear microservicios – Dzone: <https://dzone.com/articles/quick-guide-to-microservices-with-spring-boot-20-e>
<https://dzone.com/articles/quick-guide-to-microservices-with-spring-boot-2-e>

Spring Cloud

- **Gateway**

Mini-guía de Spring Cloud Gateway - Bi Geek: <https://blog.bi-geek.com/arquitecturas-microserviciosspring-cloud-gateway/>

- **Service Discovery**

Documentación Spring Service Discovery Netflix - web oficial: https://cloud.spring.io/spring-cloudnetflix/multi/multi__service_discovery_eureka_clients.html

- **Config Server**

Spring cloud config client retry:

https://cloud.spring.io/spring-cloudconfig/1.4.x/multi/multi__spring_cloud_config_client.html

Spring Boot

Variables de entorno Spring – StackOverflow:

<https://stackoverflow.com/questions/47580247/optionalenvironment-variables-in-spring-app>

Construir una API REST con Spring y Java: <https://www.baeldung.com/building-a-restful-web-service-withspring-and-java-based-configuration>

Tutoriales RESTFul con Spring: <https://www.baeldung.com/rest-with-spring-series>

Spring Data

- **PostgreSQL**

Spring data y PostgreSQL: <https://dzone.com/articles/spring-boot-and-postgresql>

Persistencia con Spring Data y PostgreSQL: <http://codedpoetry.com/persistence-with-spring-datapostgresql/>

Docker

Crear un DockerFile: <https://docs.docker.com/get-started/part2/>

Crear un DockerFile para Spring Boot: <https://spring.io/guides/gs/spring-boot-docker/>
<https://spring.io/guides/topicals/spring-boot-docker>

Comandos para Docker: <https://medium.com/the-code-review/top-10-docker-commands-you-cant-livewithout-54fb6377f481> - <https://coderwall.com/p/ewk0mq/stop-remove-all-docker-containers>

<https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/>