Elaboro	Documento	Versión	Descripción	Fecha
Luis Sánchez Martínez	Práctica integración de Circuit Breaker en microservicio.	1.0	Creación de documento	05/04/2022

Práctica integración de Circuit Breaker en microservicio

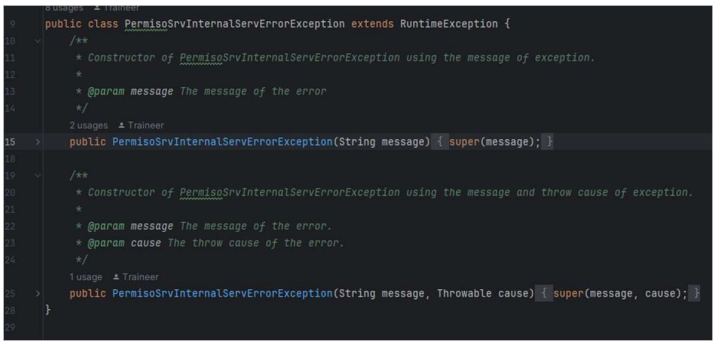
Tiempo estimado: 2 horas.

Uso de:

- JDK 17
- Maven
- IntelliJ
- Postman
- Git
- Spring Cloud
    - Resilience4J
    - Actuator

Agregar manejo de excepciones InternalServerErrorException.

1. Se crea la clase *PermisoSrvInternalServErrorException* y su manejo en la clase *ExceptionManager* creando el método *manageInternalServErrorExcpt()*.



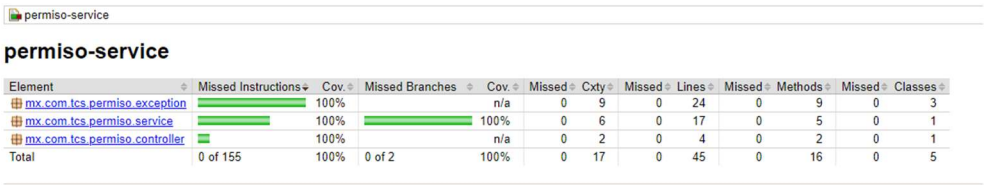
```
public class PermisoSrvInternalServErrorException extends RuntimeException {
    /**
     * Constructor of PermisoSrvInternalServErrorException using the message of exception.
     *
     * @param message The message of the error
     */
    2 usages  Trainee
    public PermisoSrvInternalServErrorException(String message) { super(message); }

    /**
     * Constructor of PermisoSrvInternalServErrorException using the message and throw cause of exception.
     *
     * @param message The message of the error.
     * @param cause The throw cause of the error.
     */
    1 usage  Trainee
    public PermisoSrvInternalServErrorException(String message, Throwable cause) { super(message, cause); }
}
```

```
/**
 * Method used to manage the exception InternalServerErrorException when the database throw a exception.
 *
 * @param ex The Exception throws in the service.
 * @return The response with ErrorDTO object filled with information about the exception.
 */
2 usages  Trainee
@ExceptionHandler
public ResponseEntity<ErrorDTO> manageInteranlServErrorExcpt(PermisoSrvInternalServErrorException ex) {
    log.error("Error [{}]: {}",INTERNALSERVERR_EXPT_CLASSNAME,ex.getMessage());
    ErrorDTO error = buildErrorMessage(
            HttpStatus.INTERNAL_SERVER_ERROR.value(),MESSAGE_INTERNAL_SERV_ERR,ex.getMessage());
    return new ResponseEntity<>(error, HttpStatus.INTERNAL_SERVER_ERROR);
}
```

2. Modificar la lógica de la clase PermisoServImpl para que en caso de que ocurran errores al consultar la BD se genere un error HTTP 500.

```java
    /**
     * Method to query from Permiso entity.
     * @return a list of Permiso object.
     */
    1 usage   Traineer
    private List<PermisoDTO> getAllPermiso() {

        try {
            List<Permiso> permisoList = (List<Permiso>) repository.findAll();

            return permisoList.
                    stream().
                    map(permiso -> modelMapper.map(permiso, PermisoDTO.class)).
                    toList();
        } catch(Exception ex) {
            log.error("Error in Service permiso API: {}",ex.getMessage());
            throw new PermisoSrvInternalServErrorException(ex.getMessage());
        }
    }
}
```

3. Modificar las pruebas unitarias para tener una cobertura total de las nuevas líneas de código agregadas.

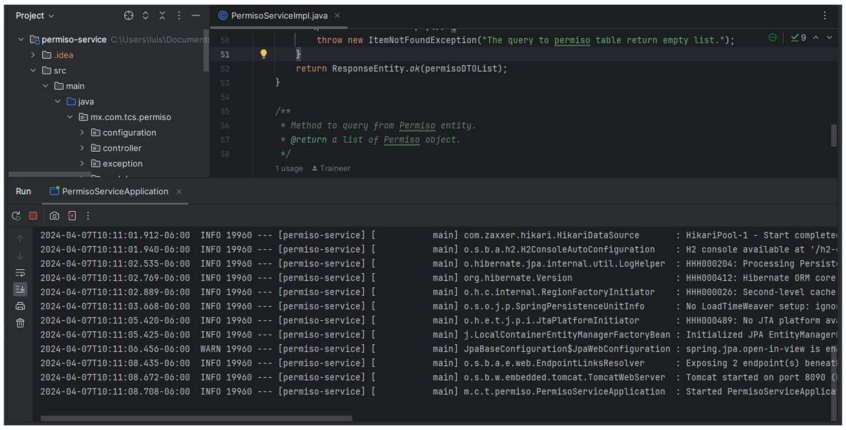


permiso-service

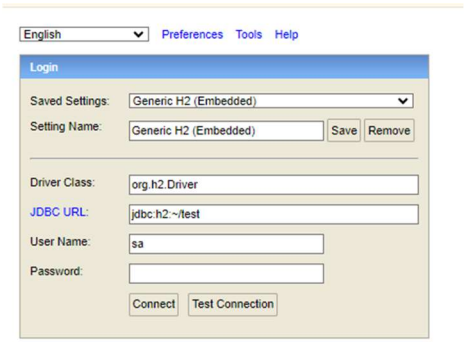
permiso-service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
mx.com.tcs.permiso.exception		100%		n/a	0	9	0	24	0	9	0	3
mx.com.tcs.permiso.service		100%		100%	0	6	0	17	0	5	0	1
mx.com.tcs.permiso.controller		100%		n/a	0	2	0	4	0	2	0	1
Total	0 of 155	100%	0 of 2	100%	0	17	0	45	0	16	0	5

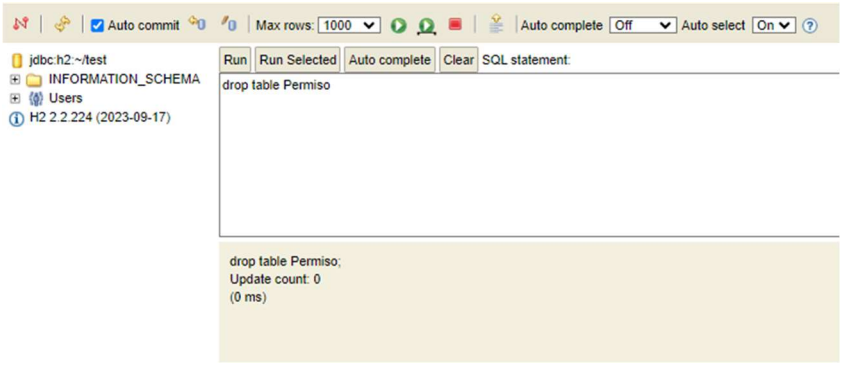
4. Ejecutar el proyecto para realizar la prueba del error HTTP 500.



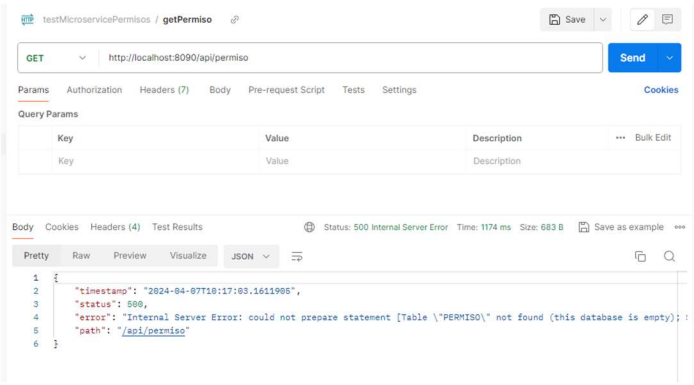
5. Abre la consola de la BD H2.



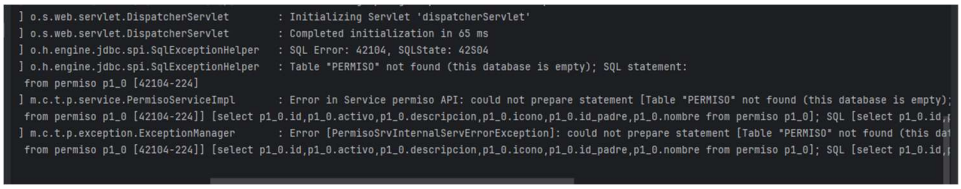
6. Elimina la tabla Permiso.



7. Ejecuta la petición al servicio */api/permiso* desde postman validando el error HTTP 500.



8. Revisar log de la consola para revisar el registro de errores.



Integrar la dependencia de actuator en el proyecto de Spring Boot.

1. Agregar la sección *<dependencyManagement>* con la dependencia *spring-cloud-dependencies*.

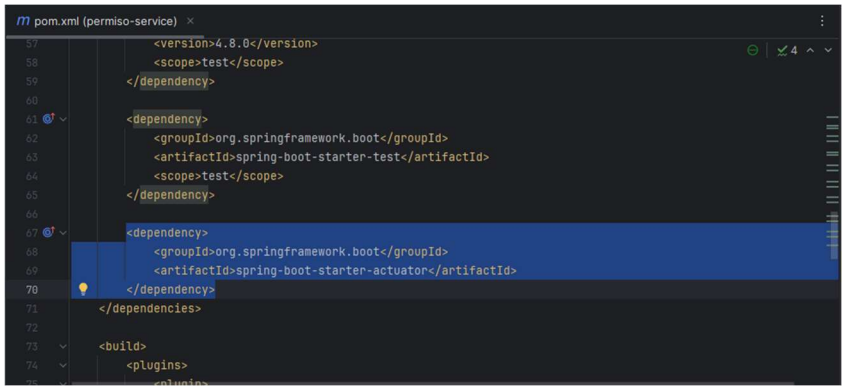
```
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
```

Nota: El uso de *<dependencyManagement>* permite controlar las versiones y evitar conflictos que podrían presentarse si cada dependencia se compila con diferentes versiones de otras librerías,

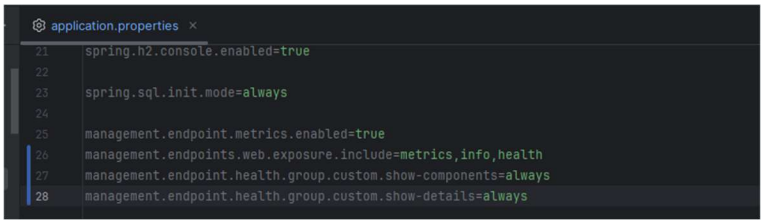
este *parent* nos permite utilizar los artefactos de Spring Cloud con facilidad minimizando los conflictos de versiones. Recordar que *Spring Cloud* está conformado por varios subproyectos, en esta práctica se utilizaran *Spring Resilience4j* y *Spring Actuator*.

2. Colocar la versión del artefacto spring-cloud-dependencies a la versión compatible 2023.0.1

3. Agregar la dependencia spring-boot-starter-actuator al archivo pom del proyecto Spring Boot.

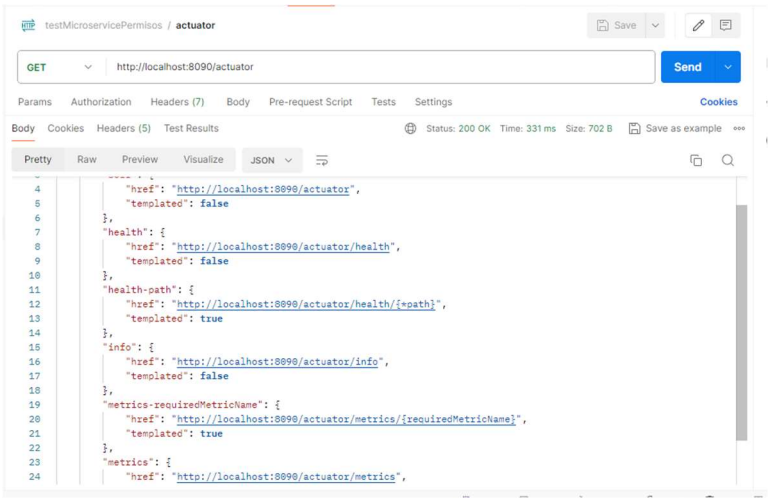


4. Configurar las propiedades para que se puedan monitorear algunos aspectos del servicio.

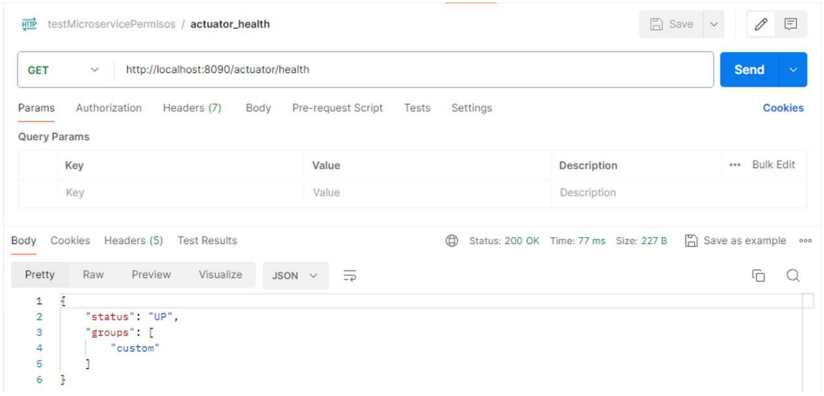


5. Iniciar el proyecto Spring Boot permiso-service.

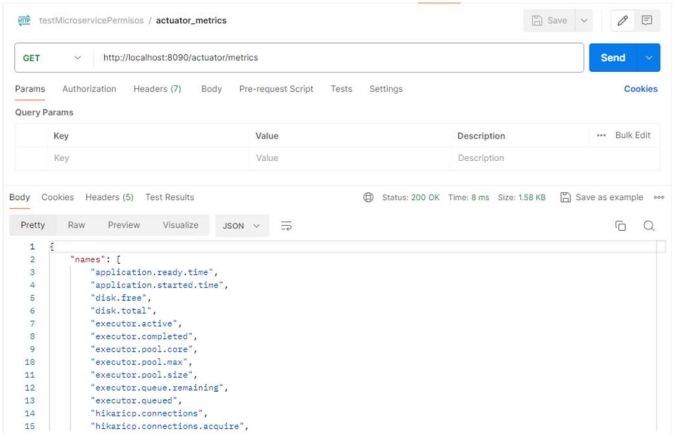
6. Ejecutar la petición en la URL http://localhost:8090/actuator y validar el monitoreo del servicio permiso.



7. Ejecutar la petición en la URL http://localhost:8090/actuator/health y validar que el servicio esté disponible.



8. Ejecutar la petición en la URL http://localhost:8090/actuator/metrics y validar las opciones que se pueden revisar.



Integrar Circuit Breaker en el proyecto de Spring Boot.

1. Agregar la dependencia resilience4j en el archivo pom del proyecto Spring Boot.

```xml
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
```

2. Agregar la configuración de la dependencia en el archivo application.properties, por default las propiedades estarán habilitadas pero si se requiere que se deshabiliten agregar la propiedad:

```
spring.cloud.circuitbreaker.resilience4j.enabled=false
```

```
spring.sql.init.mode=always

management.endpoint.metrics.enabled=true
management.endpoints.web.exposure.include=metrics,info,health
management.endpoint.health.group.custom.show-components=always
management.endpoint.health.group.custom.show-details=always

spring.cloud.circuitbreaker.resilience4j.enabled=false
```

3. Crear un objeto de Circuit Breaker usando una instancia inyectada de CircuitBreakerFactory envolviendo una funcionalidad.

```java
/**
 * Implements the method to get all records of Permiso entity.
 * @return a response entity of the list of the Permiso object.
 */
@Override
public ResponseEntity<List<PermisoDTO>> listAll() {
    CircuitBreaker circtBreakListAll = circtBreakFactory.create("circtBreakListAll");

    List<PermisoDTO> permisoDTOList = circtBreakListAll.run(
            this::getAllPermiso,throwable -> getDefaultAllPermisoList());

    if (permisoDTOList.isEmpty()) {
        throw new ItemNotFoundException("The query to permiso table return empty list.");
    }
    return ResponseEntity.ok(permisoDTOList);
}
```

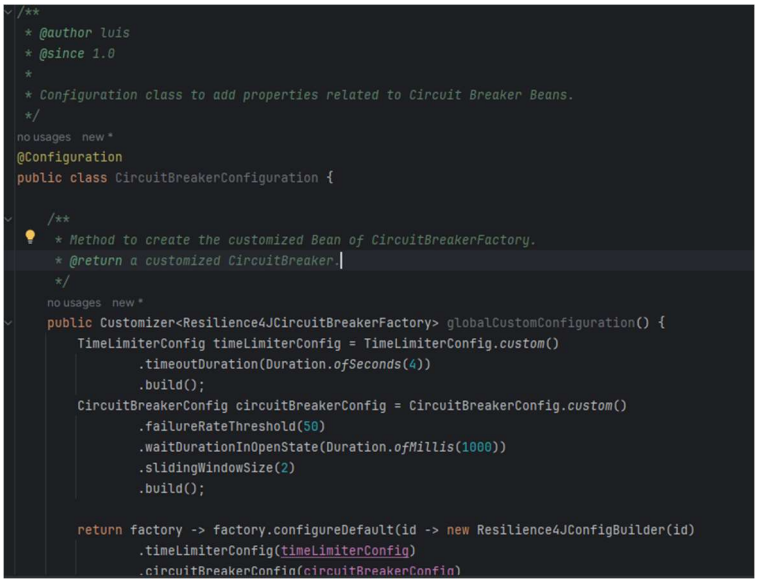
```java
    /**
     * Implements the method to get all records of Permiso entity.
     * @return a response entity of the list of the Permiso object.
     */
    5 usages  Trainee *
    @Override
    public ResponseEntity<List<PermisoDTO>> listAll() {
        CircuitBreaker circtBreakListAll = circtBreakFactory.create("circtBreakListAll");

        List<PermisoDTO> permisoDTOList = circtBreakListAll.run(
                this::getAllPermiso,throwable -> getDefaultAllPermisoList());

        if (permisoDTOList.isEmpty()) {
            throw new ItemNotFoundException("The query to permiso table return empty list.");
        }
        return ResponseEntity.ok(permisoDTOList);
    }
```

4. Agregar la respuesta fallback.

```java
/**
 * Fallback response used when throw an exception.
 * @return List of DTO permiso object.
 */
private List<PermisoDTO> getDefaultAllPermisoList() {
    List<PermisoDTO> permisoDTOList = new ArrayList<>();
    permisoDTOList.add(getDefaultPermisoDTO());
    return permisoDTOList;
}
```

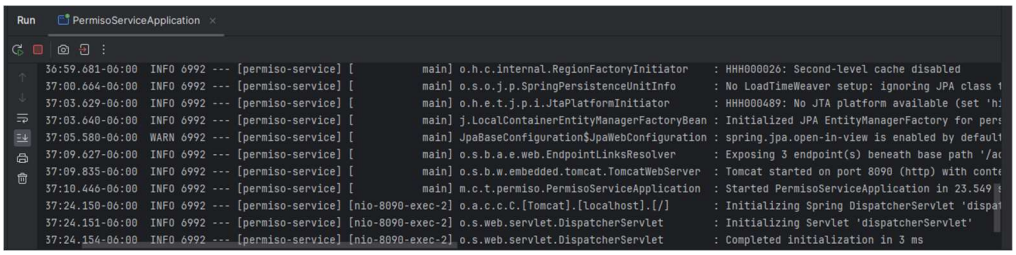
```java
/**
 * Fallback response used when throw an exception.
 * @return List of DTO permiso object.
 */
1 usage  new *
private List<PermisoDTO> getDefaultAllPermisoList() {
    List<PermisoDTO> permisoDTOList = new ArrayList<>();
    permisoDTOList.add(getDefaultPermisoDTO());
    return permisoDTOList;
}

/**
 * Create a default DTO object of Permiso entity.
 * @return DTO object from Permiso entity.
 */
1 usage  new *
private PermisoDTO getDefaultPermisoDTO() {
    PermisoDTO permisoDTO = new PermisoDTO();
    permisoDTO.setId(1);
    permisoDTO.setNombre("Default Permiso");
    permisoDTO.setDescripcion("Esta es una descripcion para la respuesta por default Permiso.");
    permisoDTO.setIdPadre(1);
    permisoDTO.setActivo(1);
    permisoDTO.setIcono("/images/icon_default_perm.gif");
    return permisoDTO;
}
```

5. Agregar la clase de configuración CircuitBreakerConfiguration que permitirá integrar las propiedades del TimeLimiter y CircuitBreaker.

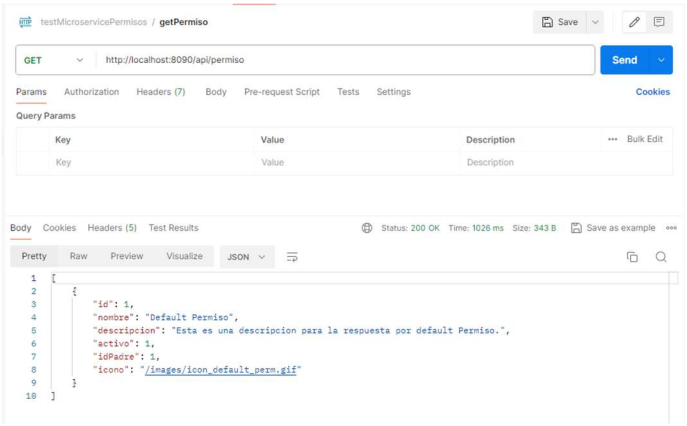


```java
/**
 * @author luis
 * @since 1.0
 *
 * Configuration class to add properties related to Circuit Breaker Beans.
 */
no usages  new *
@Configuration
public class CircuitBreakerConfiguration {

    /**
     * Method to create the customized Bean of CircuitBreakerFactory.
     * @return a customized CircuitBreaker
     */
    no usages  new *
    public Customizer<Resilience4JCircuitBreakerFactory> globalCustomConfiguration() {
        TimeLimiterConfig timeLimiterConfig = TimeLimiterConfig.custom()
                .timeoutDuration(Duration.ofSeconds(4))
                .build();
        CircuitBreakerConfig circuitBreakerConfig = CircuitBreakerConfig.custom()
                .failureRateThreshold(50)
                .waitDurationInOpenState(Duration.ofMillis(1000))
                .slidingWindowSize(2)
                .build();

        return factory -> factory.configureDefault(id -> new Resilience4JConfigBuilder(id)
                .timeLimiterConfig(timeLimiterConfig)
                .circuitBreakerConfig(circuitBreakerConfig)
```

6. Iniciar el servicio *permiso-service* para realizar pruebas a la implementación de Circuit Breaker.



7. Realizar una prueba en la cual agregamos un retraso en la respuesta de a consulta a la BD.



Referencias

https://www.baeldung.com/spring-boot-actuators

https://docs.spring.io/spring-cloud-circuitbreaker/docs/current/reference/html/

https://www.baeldung.com/spring-cloud-circuit-breaker

https://www.baeldung.com/maven-dependencymanagement-vs-dependencies-tags

https://www.javaenlasopa.com/2017/02/revisando-la-jerarquia-de-dependencias.html