

Práctica Copia.me

Tomado del contexto del final 22/12/2018

Contexto.....	2
1ra Iteración.....	3
Paso 1: Diseño.....	4
Paso 2: Creación del proyecto.....	5
Paso 3: Implementación del dominio.....	6
Paso 4: Testing.....	7
Paso 5: Actividad Test.....	8
Paso 6: Implementación de la Aplicación.....	8
Paso 7: Capa fuente de datos.....	9
Paso 8: Finalice el programa.....	10
2da Iteración.....	11
Paso 1: Ordenar Pruebas de Integración.....	11
Paso 2: Usar el servidor de CI de Github.....	13
Paso 3: Crear el bot de telegram.....	16
Paso 4: Hacer un Bot Echo.....	16
Paso 5: Procesar un batch de un adjunto.....	18
3ra Iteración.....	20
Paso 0: Entender los nuevos requerimientos.....	20
Paso 1: Diseñar la API.....	21
Paso 2: Actualizar el diseño.....	22
Escenario Bronce.....	23
Escenario Plata.....	23
Paso 3: Armado de los repositorio.....	25
Paso 4: Desarrollo de la API.....	26
Paso 5: Uso de la API.....	28
Paso 6: Cambio del telegram de Bot.....	29
Paso 7: Despliegue de la API en Render.....	31
Paso 8: Implementar revisiones.....	35
4ta Iteración.....	36
Paso 1: Modelo relacional.....	36
Paso 2: Puesta en marcha del ORM.....	37
Paso 3: Primer mapeo.....	38
Paso Opcional 1: Verificar que funcione en Postgres.....	40
Paso 4: Mapeo Completo.....	41
Paso 5: Deploy de la DB.....	44
5ta Iteración.....	48
Paso 1: Crear cuenta en un CloudAMQP.....	48
Paso 2: Crear un worker para que levante los paquetes.....	49
Paso 3: Refactor de la APP.....	53
Paso 4: Hacer que el worker procese los análisis.....	57
Paso 5: API envía un mensaje cuando recibe un nuevo lote.....	59

Contexto

Copia.me será un sistema en el que los usuarios subirán documentos de distintos tipos con el fin de analizarlos para detectar si estos son plagios.

Por ahora soportaremos documentos de texto, programas y partituras pero se espera incorporar más. También se contará con revisores freelancer que se encargarán de analizar manualmente los documentos para darles un análisis más inteligente.

Copia.me estará orientado a docentes, que cargarán lotes de documentos y recibirán un reporte que indique cuáles son copias entre sí.

Los usuarios de Copia.me podrán elegir entre tres calidades de servicio distintas; cada una realiza distintos análisis y por supuesto esto afectará en el precio de la revisión. Las calidades de servicios con los que contamos son:

- **Bronce:** sólo realiza detección automática, usando diferentes algoritmos, dependiendo del tipo de contenido:
 - Los documentos legales, literarios y académicos se comparan usando la distancia de Levenshtein.
 - Los fragmentos de código se analizan usando Detección de clones basado en árboles sintácticos.
 - Las partituras musicales se analizan usando Acoustic fingerprint.
- **Plata:** además de la detección automática, envía un cierto porcentaje (configurable por el usuario, esto influye en el costo) de los documentos a analizar contra revisores freelancers que realizan una revisión manual. Cada revisor recibe un email con cada par de documentos asignados.
- **Oro:** además de la detección automática, envía todos los documentos a los revisores, y además hace una revisión cruzada: un cierto porcentaje de los mismos (nuevamente, configurable por el usuario, esto influye en el costo) los envía a más de un revisor.

Cuando un análisis (detección automática, revisión manual o revisión cruzada) termina, genera un índice de copia para cada par de documentos, entre 0 y 1. Y cuando todos los análisis correspondientes para una calidad de servicio terminan, se marca a cada documento afectado como revisado. Se considera plagio cuando el documento recibe un promedio de puntaje mayor a 0,6. Tener en cuenta que estos análisis son asincrónicos, por lo que pueden terminar en cualquier orden.

Obviamente, contar con un gran equipo de revisores freelancers no es tarea fácil, así vamos a asumir algunas cosas:

1. De cada freelancer sabemos cuántos documentos puede revisar por máximo cada mes, y su email.
2. Siempre habrá revisores disponibles.
3. Los revisores ingresarán al sistema y cargarán los resultados de sus análisis, siempre en el plazo máximo de 1 día.

4. Para la comunicación con los revisores, ya contamos con un componente que nos permite enviar emails usando SMTP (aunque en un futuro tal vez contratemos un servicio externo).

Repo Resuelto: <https://github.com/ezequieljsosa/ddscopiame>

1ra Iteración

El objetivo de la primera iteración es familiarizarse con la tecnología Java, Maven y de testing. Por otro lado hacer una primera aplicación, siguiendo las capas aprendidas en clase.

El alcance será el siguiente:

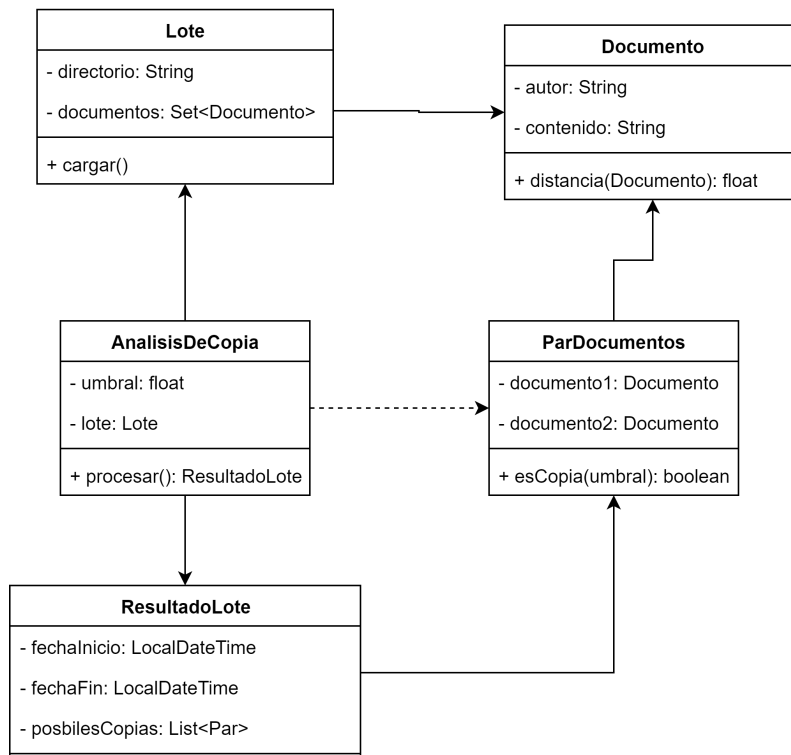
- Solo detección automática de documentos txt.
- Proceso monousuario / “monodocente”
- El lote de documentos, será procesado localmente, mediante un programa de línea de comandos.

Conceptos importantes antes de iniciar:

- Java es un lenguaje fuertemente tipado y compilado. No se darán mayores explicaciones de la sintaxis por su parecido con Wolok. Cualquier duda puede consultar los apuntes asociados
- JUnit: framework de testeo unitario para Java. Permite definir fixture de pruebas y obtener reportes sobre los resultados de las mismas
- Maven es una herramienta para la construcción y creación de proyectos, de difundido uso en Java. Tiene numerosas características, pero nosotros resaltaremos las siguientes:
 - Propone una estructura de directorios para ordenar el código
 - Nos ayuda a gestionar las dependencias
 - Ofrece un formato para configurar el proyecto en un archivo aparte

Paso 1: Diseño

Se propone el siguiente diseño para esta iteración:



1. Estamos suponiendo que los documentos no tienen un gran tamaño, por eso leemos su contenido completo
2. Comparar 2 documentos implica tener algún concepto de distancia, en el alcance de esta iteración la Levenshtein.
3. Se guardan los pares, para tener registro de cuales son copia
4. Como veremos más adelante, cuando se procesa un conjunto de datos, a parte de los resultados, es conveniente guardar el inicio y final de dicho proceso
5. El umbral del análisis determina si 2 documentos son copia por arriba de ese valor (en principio). Como todos los documentos de un lote deben ser procesados con el mismo criterio, el mismo esta en el análisis
6. `ParDocumentos#esCopia(umbral): documento1.distancia(documento2) < umbral`

Preguntas para hacerse hasta el momento

- Relacione el 1er punto con algún tema visto en la teórica
- ¿Por qué le parece que el umbral está en el AnalisisDeCopia y no en Lote?
- ¿Que opina que el umbral sea un parámetro y no un valor fijo? Por ejemplo `ParDocumentos#esCopia(umbral): documento1.distancia(documento2) < 0.6`

Paso 2: Creación del proyecto

La forma más fácil de crear un proyecto Maven es a través de la IDE¹. Por ejemplo en Eclipse la mejor forma File -> New -> Maven Project, chequear "Create a simple project", siguiente -> Group Id: ar.edu.utn.dds.copiame ; ArtifactId copiame-cli ; => finalizar.

Confiamos en la sapiencia y criterio de un alumno de 3er año para realizar los pasos anteriores en otras IDEs y cualquier cambio que haya que hacer para que funcione.

Con esto vamos a tener la siguiente estructura de proyecto

src

main

java : en este directorio van los paquetes / clases de mi programa
resources : cualquier recurso que no sea código utilizado por las clases / app
(imágenes, sonidos, texto, etc..)

test

java : Clases de para hacer pruebas: Fixtures y clases auxiliares
resources : cualquier archivo que se utilice para realizar pruebas, y/o sets de datos

pom.xml : Nombre y configuración del proyecto, independiente de la IDE o ambiente de desarrollo. Por ejemplo versión de Java y dependencias.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ar.utn.dds.copiame</groupId>
  <artifactId>copiame-cli</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <!-- Todo lo que esta de aquí en adelante hay que agregarlo -->
  <properties>
    <maven.compiler.target>17</maven.compiler.target>
    <maven.compiler.source>17</maven.compiler.source>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.9.0</version>
      <scope>test</scope>
    </dependency>
    <!-- Para el cálculo de la distancia de Levenshtein -->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-text</artifactId>
      <version>1.10.0</version>
    </dependency>
  </dependencies>
</project>
```

¹ si bien por lo general recomendamos usar más la línea de comandos, para una primera práctica usar este método está perfecto

Tips: verificar que tienen la misma versión de Java instalada (Recomendamos fuertemente usar la 17). Se elige encoding se pone el más estándar por las dudas.

En este punto es conveniente agregar al proyecto, al archivo .gitignore en la raíz del mismo (a la misma altura que el pom.xml). Para esto recomendamos usar este [sitio](#), y colocar los tags: Maven, Java, Eclipse, IntelliJ+iml. Esto nos ayudará a no hacer commit de archivos que no quiero versionar. Más adelante vamos a nombrar un par de ejemplos

Paso 3: Implementación del dominio

En este punto, implemente todas las clases, menos los métodos públicos: Lote#cargar, Documento#distancia, AnalisisDeCopia#procesar, ParDocumentos#esCopia (déjelos a todos retornando null). Todo sobre src/main/java y recuerden poner un paquete, por ejemplo “ar.utn.dds.copiame”. Los paquetes siempre van en minúscula y con el formato del ejemplo que se dio. Ejemplo para ResultadoLote:

```
package ar.utn.dds.copiame; // Las clases SIEMPRE deben estar dentro de un
paquete
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
public class ResultadoLote {
    private LocalDateTime fechaInicio;
    private LocalDateTime fechaFin;
    private List<ParDocumentos> posiblesCopias;

    public ResultadoLote() {
        super();
        this.posiblesCopias = new ArrayList<ParDocumentos>();
    }
    public LocalDateTime getFechaInicio() {
        return fechaInicio;
    }
    public void setFechaInicio(LocalDateTime fechaInicio) {
        this.fechaInicio = fechaInicio;
    }
    public LocalDateTime getFechaFin() {
        return fechaFin;
    }
    public void setFechaFin(LocalDateTime fechaFin) {
        this.fechaFin = fechaFin;
    }
    public List<ParDocumentos> getPosiblesCopias() {
        // encapsulamos la colección para que no puedan manipularla sin
        usar agregarPar
        return new ArrayList<ParDocumentos>(posiblesCopias);
    }
    public void agregarPar(ParDocumentos par) {
        this.posiblesCopias.add(par);
    }
}
```

Paso 4: Testing

Ahora vamos a implementar un par de pruebas unitarias, las cuales consisten en clases con una estructura en particular, para que puedan ser interpretadas por el framework JUnit. Dichas clases se crean sobre `src/test/java`, sobre el MISMO paquete de las clases que prueban. Por ejemplo vamos a construir el fixture de `DocumentoTest`, es decir la clase que tendrá las pruebas de la clase `Documento`.

```
package ar.utn.dds.copiame;
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

public class DocumentoTest {
    @Test
    public void testDocumentosIguales() {
        Documento doc1 = new Documento("", "hola");
        Documento doc2 = new Documento("", "hola");
        assertEquals(0, doc1.distancia(doc2),
            "los textos que son iguales tienen que dar una distancia de 0");
    }
    @Test
    public void testDocumentosDistintos() {
        Documento doc1 = new Documento("", "chau");
        Documento doc2 = new Documento("", "hola");
        assertEquals(1, doc1.distancia(doc2),
            "los textos que son totalmente distintos tienen que dar una distancia de 1");
    }
    @Test
    public void testDocumentosParecidos() {
        Documento doc1 = new Documento("", "hola");
        Documento doc2 = new Documento("", "halo");
        // el 3er parámetro es una tolerancia para comparar (decir si son iguales) los números con punto flotante
        assertEquals(0.5, doc1.distancia(doc2), 0.1,
            "los textos que son parecidos en un 50% tienen que dar ~0.5");
    }
}
```

Si ejecutamos las pruebas fallaran, todas si pusimos que “distancia” retorne null, o 2 si pusimos que retorne “0”, pero esa esta funcionando de casualidad. Ahora vamos a implementar el método de distancia:

```
public Float distancia(Documento otroDoc) {
    /* La distancia de Levenshtein mide la cantidad de operaciones
    * (inserción, eliminación o sustitución de un carácter) necesarias
    para transformar un texto en otro */
    // Mientras más distintos los textos, más alta es la distancia
    Integer rawDist = LevenshteinDistance.getDefaultInstance(
        ).apply(this.contenido, otroDoc.getContenido());
    // Normalizamos por el tamaño de la distancia por el tamaño del texto mas largo
    Integer contentSize = Math.max(this.contenido.length(),
        otroDoc.getContenido().length());
    return rawDist * 1.0f / contentSize;
}
```

Ahora las pruebas deberían funcionar normalmente.

Paso 5: Actividad Test

Realice 2 pruebas sobre la clase ParDocumentos. ¿ Qué método es el más relevante? ¿ Qué situaciones se le ocurre probar ? ¿ Hace falta más de 2 tests ?

Paso 6: Implementación de la Aplicación

Nuestra aplicación, hasta el momento, es un CLI (línea de comandos). Por ello, la interacción con el usuario se puede dar de las siguientes formas:

1. Con un argumento cuando se ejecuta el programa.
2. Pidiendo una interacción al usuario.
3. Muestra datos a través de la consola.
4. Puede escribir algo en uno o más archivos, u otros lugares.
5. Interactuar con otros sistemas.

Para que sea más fácil, nosotros utilizaremos (1) y (3). Es decir, no habrá ningún pedido de datos al usuario una vez que se arrancó el programa, este simplemente indicará qué información quiere procesar y obtendrá la respuesta por consola.

Antes que nada, tenemos que establecer pre-condiciones, sobre las entradas y salidas del sistema:

- Entrada: Directorio con archivos txt, donde el nombre de cada archivo es el autor del mismo.
- Salida: personas que probablemente se hayan copiado entre ellas

```
public class CopiaMeApp {
    public static void main(String[] args) {
        // Valido argumentos del usuario --> Capa ?
        Path pathLote = Paths.get(args[0]);
        if ( ! Files.exists( pathLote ) ) {
            System.err.println("'" + args[0] + "' no existe...");
            System.exit(1);
        }
        // -----
        //Cargo el Lote del Sistema de archivos --> Utilizo la Capa ?
        Lote lote = new Lote(args[0]);
        lote.cargar();
        // -----
        // Utilizo la capa de ? -- NO leo datos de otra fuente -- NO
        pido ni muestro información
        float umbral = 0.5f;
        AnalsisDeCopia analisis = new AnalsisDeCopia(umbral, lote);
        ResultadoLote resultado = analisis.procesar();
        //-----
        // Muestro la informacion por pantalla --> Capa de ?
        for (ParDocumentos par : resultado.getPosiblesCopias()) {
            System.out.println(par.getDocumento1().getAutor() +
                " " + par.getDocumento2().getAutor() );
        }
        //-----
    }
}
```


Método procesar de AnalsisDeCopia

Primero agreguemos una libreria que nos permitira hacer las combinaciones de los documentos:

```
<!-- https://mvnrepository.com/artifact/com.github.dpaukov/combinatoricslib3 -->
<dependency>
    <groupId>com.github.dpaukov</groupId>
    <artifactId>combinatoricslib3</artifactId>
    <version>3.3.3</version>
</dependency>
```

```
public ResultadoLote procesar() {
    // Genero todos los pares de documentos Posibles
    List<ParDocumentos> pares =
        Generator.combination(this.lote.getDocumentos())
            .simple(2)
            .stream()
            .map(t-> new ParDocumentos(t.get(0),t.get(1)) )
            .collect(Collectors.toList());

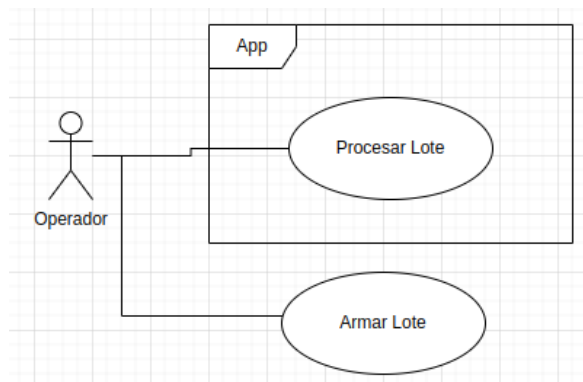
    // Armo el resultado procesando cada par
    ResultadoLote rl = new ResultadoLote();
    rl.setFechaInicio(LocalDateTime.now());
    for (ParDocumentos parDocumentos : pares) {
        if(parDocumentos.esCopia(this.umbral)) {
            rl.agregarPar(parDocumentos);
        }
    }
    rl.setFechaFin(LocalDateTime.now());
    return rl;
}
```

Actividades:

- 1) Complete las capas en los comentarios del método CopiaMeApp#main
- 2) Vea el diagrama UML, por que cree que la linea es distinta de AnalsisDeCopia a ParDocumentos respecto a ResultadoLote

Paso 7: Capa fuente de datos

Para arrancar, tenemos que poner unas precondiciones. Analicemos los CU que tenemos hasta el momento.



Tenemos 2 cosas para marcar, primero, armar el lote, si bien es necesario, no ofrecemos funcionalidad para hacerlo, eso tiene que hacerlo el actor por fuera de la aplicación. Por otro lado, debemos ser precisos a la hora de definirlo, ya que esas son las precondiciones del CU Procesar Lote:

- Toma de entrada un directorio
- Dentro del mismo hay archivos .txt, los que no lo sean, serán ignorados para la carga
- El nombre del archivo será el nombre del autor del documento (otra posibilidad sería tener el vínculo en un archivo aparte, pero vamos a hacerlo de esta forma). Para evitar algunos problemas, los espacios serán separados por “_”

Luego podemos plantear escenarios de error, tales como: el directorio no existe, el directorio tiene menos de 2 documentos, ¿qué hay archivos vacíos?, ¿qué hacer si se detecta que un archivo no es de texto a pesar de ser .txt ?. Por ahora solo trataremos la situación de que el directorio no exista.

Programa Lote#cargar, puede usar los siguientes métodos para ayudarse:

```
String directorio = "una/ruta"
Path dirpath = Paths.get(directorio);
Files.list(dirpath).collect(Collectors.toList()) ; // nos da los archivos
de un directorio (suponiendo que dirpath es un directorio)
A los objetos de la clase Path le puedo pedir .getFileName(), para que nos
dé el nombre de archivo completo
```

Paso 8: Finalice el programa

Complete lo que haga falta y pruebe el main, con un par de lotes distintos.

- ¿Cree usted que falta interacción con el usuario?
- ¿Si tuviese que dar la opción de guardar el resultado en un archivo, donde lo haría?

2da Iteración

El objetivo de la segunda iteración es mejorar un poco el testing y agregar una nueva app que utilice el dominio modelado hasta el momento.

El alcance será el siguiente:

- Vamos a separar pruebas de “integración” de las unitarias
- Automatizar un flujo de trabajo
- Crear un Bot de Telegram que haga lo mismo que hacemos con el CLI

Para arrancar, recomiendo crear un repo nuevo e importar el contenido de la rama iteración1 una de este [repo](#). Lo más fácil es bajar los fuentes directamente desde [acá](#) iteracion1.

Paso 1: Ordenar Pruebas de Integración

Primero vamos a agregar unas pruebas más, así cuando empecemos a agregar cambios, estamos más confiados de que no estamos rompiendo algo. Hagamos un test de integración, que pruebe una corrida completa.

```
public class CopiameAppIT {
    @Test
    public void testBronceTexto() throws Exception {
        // Armado del Escenario
        Lote lote = new Lote("src/test/resources/lotel");
        lote.validar();
        lote.cargar();
        float umbral = 0.5f;
        AnalisisDeCopia analisis = new AnalisisDeCopia(umbral, lote);
        // Ejecución
        ResultadoLote resultado = analisis.procesar();
        // Chequeo
        assertEquals(1, resultado.getPosiblesCopias().size(),
            "Hay al menos una copia, ya que Pepe y Raúl se copiaron ");
    }
}
```

Vamos a decir que es de integración porque: Estamos probando varias clases y su interacción y estamos usando archivos. Y podemos notar 2 cosas mas, estamos usando un path del fuente a los datos sobre test y el final de la clase de test termina en IT no en Test².

¿ En este tipo de test, conviene probar si el directorio del lote no existe?

Ahora mediante la línea de comando o haciendo click derecho sobre el proyecto y buscando la opción de correr “mvn test” puede verificar que solo se ejecuta DocumentoTest, esto es pq maven, por defecto, solo usa archivos con esa regla de nombres.

Ahora, si agregamos ese elemento al pom.xml, a la misma altura que “properties” y “dependencies”, vamos a tener una acción más llamada “verify”, que no solo correrá la acción de test, sino que luego correrá los tests de integración, es decir, los que terminan en IT. Si quiere correrla por la IDE y no por la línea de comandos, tendrá que agregar el método de ejecución³(opcional si quiere hacerlo).

² Si bien no es una práctica tan comúnmente utilizada, nos pareció interesante su implementación

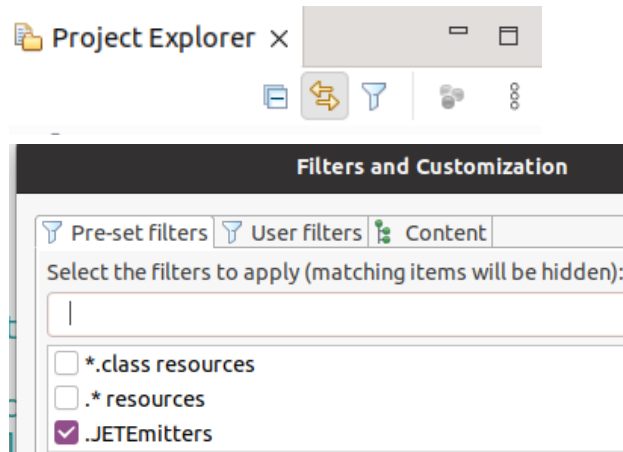
³ Por ejemplo en Eclipse: click derecho sobre el proyecto, Run As , Run Configurations, en la izquierda buscar “Maven build”, click derecho sobre ese titulo, Poner “Tests integración” en el titulo,

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>2.22.0</version>
      <executions>
        <execution>
          <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

seleccionar el proyecto como Base directory (click sobre Workspace... y elegir el proyecto) y donde dice "Goals:" poner solo "verify". Con eso debería poder ejecutarlo

Paso 2: Usar el servidor de CI de Github

Ahora que tenemos más tests, vamos a configurar la acción de ejecutar las pruebas en un servidor externo (en este caso Github) cada vez que se hace un push a una rama. Para eso, vamos a crear el archivo `ci.yml` dentro de la carpeta `.github/workflows` (también hay que crearlos) dentro del proyecto. Notar que las IDEs (y a veces el navegador de archivos) no suelen mostrar archivos que arranquen con `."`, pero puede habilitarlo desde el embudo en el "project explorer" y sacar el filtro `*.resources"`:



Sino simplemente puede usar el navegador de archivos y verificar que tenga configurada la opción de ver archivos ocultos.

Para hacer eso, vamos a utilizar Github Actions, y configurar que cada vez que se realice un "push" a cualquier rama, se corran todas las pruebas. Esto es aplicable, para un modelo de ramificación "trunk based" por ejemplo.

Dentro de `.github/workflows/ci.yml` (a la misma altura del `pom.xml` tiene que estar el directorio `.github`, también notar que tiene que crear tanto la ruta como el archivo). Los espacios en el archivo `.yml` SON importantes.

name: Mini CI

on:

[push]

jobs:

test_commit:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3
- name: Usamos la JDK 17
uses: actions/setup-java@v3
with:
java-version: '17'
distribution: 'temurin'
- name: Correr pruebas
run: mvn verify

Ahora, agreguemos todo esto al proyecto (pom, CopiameAppIT y ci.yml) y hagamos el push. Luego sobre su repositorio en Github, vaya a la solapa “Actions” y le tendría que mostrar el workflow ejecutado por la subida del último commit. Si hace click sobre la misma, podrá ver en el log la ejecución de la tarea “Mini CI”

The screenshot shows the GitHub Actions interface. At the top, there's a navigation bar with links to Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights. The 'Actions' tab is selected. On the left, there's a sidebar with 'All workflows' selected, and a list of workflows including 'Mini CI', 'Management', and 'Caches'. The main area shows 'All workflow runs' with a summary of '2 workflow runs'. One run is highlighted: 'CI + Test de integración' (status: success), triggered by 'ezequieljsosa pushed' (commit f732046) with a total duration of 22s. Below this, there's a section for 'ci.yml' on: push, showing a job 'test_commit' that succeeded in 10s.

The screenshot shows the GitHub Actions log for the 'test_commit' job. The log is displayed in a dark theme. At the top, it says 'test_commit' and 'succeeded 3 minutes ago in 10s'. There's a search bar and icons for refreshing and settings. The log shows the following steps:

- > Set up job (1s)
- > Run actions/checkout@v3 (0s)
- > Usamos la JDK 17 (0s)
- ✓ Correr pruebas (7s)

The 'Correr pruebas' step is expanded, showing the following commands and output:

```
1 ▶ Run mvn verify
7 [INFO] Scanning for projects...
8 [INFO]
9 [INFO] -----< ar.utn.dds.copiame:copiame-cli >-----
10 [INFO] Building copiame-cli 0.0.1-SNAPSHOT
11 [INFO] from pom.xml
```

```
4337 [INFO]
4338 [INFO] -----
4339 [INFO] T E S T S
4340 [INFO] -----
4341 [INFO] Running ar.utn.dds.copiams.CopiamsAppIT
4342 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.058 s - in ar.utn.dds.copiams.CopiamsAppIT
4343 [INFO]
4344 [INFO] Results:
4345 [INFO]
4346 [INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
4347 [INFO]
4348 [INFO]
4349 [INFO] --- maven-failsafe-plugin:2.22.0:verify (default) @ copiams-cli ---
4350 [INFO] -----
4351 [INFO] BUILD SUCCESS
4352 [INFO] -----
4353 [INFO] Total time: 6.504 s
4354 [INFO] Finished at: 2023-04-29T03:23:24Z
4355 [INFO] -----

> ✓ Post Usamos la JDK 17 0s
> ✓ Post Run actions/checkout@v3 0s
> ✓ Complete job 0s
```

¿Se le ocurre alguna otra prueba de integración?. Implementarla y volver a hacer push, para verificar que el workflow se vuelve a correr.

Paso 3: Crear el bot de telegram

En esta iteración vamos a agregar una nueva capa de presentación, creando un bot de Telegram para procesar los lotes de trabajo, en lugar de usar la línea de comandos. Crear un bot, implica simplemente crear las credenciales para que telegram nos envíe los mensajes de 3ros a nuestro proyecto.

- 1) Instalar Telegram, puede ser en el celular o en una aplicación en la PC. Incluso puede hacerse sin número de celular, aunque es bastante más fácil si se usa.
- 2) Agregar a los contactos al @botfather



- 3) crear un nuevo bot con el comando “/newbot”
- 4) Seguir los pasos y tomar nota del token
- 5) Agregue el bot que recién creó como contacto de telegram

Paso 4: Hacer un Bot Echo

Vamos a armar un primer boot que repite todo lo que se le envía. Para facilitar las cosas vamos a usar una librería telegramsbots. Vamos primero a gregar la dependencia al pom:

```
<dependency>
  <groupId>org.telegram</groupId>
  <artifactId>telegramsbots</artifactId>
  <version>6.5.0</version>
</dependency>
```

Ahora vamos a generar la clase CopiameBot


```

public class CopiameBot extends TelegramLongPollingBot {
    @Override
    public void onUpdateReceived(Update update) {
        // Esta función se invocará cuando nuestro bot
        // reciba un mensaje
        // Se obtiene el mensaje escrito por el usuario
        final String messageTextReceived =
            update.getMessage().getText();
        // Se obtiene el id de chat del usuario
        Long chatId = update.getMessage().getChatId();
        // Se crea un objeto mensaje
        SendMessage message = new SendMessage();
        message.setChatId(chatId.toString());
        message.setText(messageTextReceived);
        try {
            // Se envía el mensaje
            execute(message);
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }
    @Override
    public String getBotUsername() {
        // Se devuelve el nombre que dimos al bot al crearlo
        // con el BotFather
        return "nombrebot";
    }
    @Override
    public String getBotToken() {
        // Se devuelve el token que nos generó el BotFather
        // de nuestro bot
        return "token bot";
    }
    public static void main(String[] args)
        throws TelegramApiException {
        // Se crea un nuevo Bot API
        final TelegramBotsApi telegramBotsApi = new
            TelegramBotsApi(DefaultBotSession.class);
        try {
            // Se registra el bot
            telegramBotsApi.registerBot(new CopiameBot());
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }
}

```

Pruebe un poco la aplicación, trate de entender como funciona. Vea que responde lo que le envía y también vea que pasa si pone un breakpoint en el metodo `onUpdateReceived`.

Paso 5: Procesar un batch de un adjunto

Ahora vamos a procesar un zip con un conjunto de documentos. Vamos a reemplazar el contenido del método `onUpdateReceived` con lo siguiente:

```
@Override
public void onUpdateReceived(Update update) {
    Message message = update.getMessage();
    if (message.hasDocument()) {
        Document document = message.getDocument();
        if (document.getMimeType().equals("application/zip")) {
            try {
                // Obtiene el archivo
                GetFile getFile = new GetFile();
                getFile.setFileId(message.getDocument().getFileId());
                org.telegram.telegrambots.meta.api.objects.File file =
                    execute(getFile);

                java.io.File downloadedFile = downloadFile(file);
                // Descomprime los archivos en un directorio
                String destDirectory = "Poner una ruta cualquiera a un directorio
temporal/" + message.getDocument().getFileId();
                UnzipUtility.unzip(downloadedFile, destDirectory );

                // Procesa al lote
                Lote lote = new Lote(destDirectory);
                lote.validar();
                lote.cargar();
                float umbral = 0.5f;
                AnalisisDeCopia analisis = new AnalisisDeCopia(umbral, lote);
                ResultadoLote resultado = analisis.procesar();

                // Genera la salida y manda el mensaje
                String se_copiaron = "";
                for (ParDocumentos par : resultado.getPosiblesCopias()) {
                    se_copiaron += par.getDocumento1().getAutor() + " " +
                        par.getDocumento2().getAutor() + "\n";
                }
                // Envia el mensaje al usuario
                SendMessage responseMsg = new SendMessage();
                responseMsg.setChatId(message.getChatId());
                if (se_copiaron.isBlank()) {
                    responseMsg.setText("No se copió nadie");
                } else {
                    responseMsg.setText("Se copiaron: \n" + se_copiaron);
                }
                execute(responseMsg);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Y copiese tal cual la clase [UnzipUtility](#), para que le compile el proyecto

Tómese un tiempo para analizar el código, pero básicamente, primero se descarga el archivo (estamos suponiendo que el zip solo tiene archivos, no directorios), se descomprime en un directorio temporal, se procesa el lote y finalmente se informa de los resultados en un mensaje.

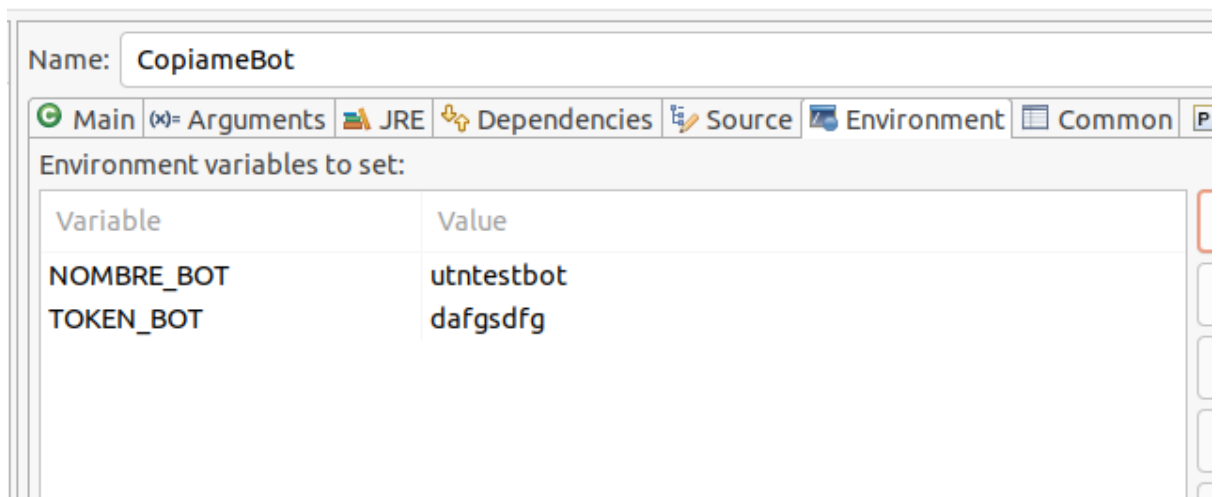
Nuevamente, trate de identificar las capas en el código, piense el paralelo con el programa de comando que hizo antes, y con el test de integración.

Ahora, antes de subir los cambios, vamos a hacer unas correcciones. La primera es de seguridad, y consiste en NO subir el token de nuestro bot (el que tiene el token tiene el bot, lo perdemos), con lo cual hay que sacarlo del código. Lo segundo, es que queremos preparar nuestro bot para que pueda usar otros bots, con lo cual, debemos parametrizar el nombre del bot y su token. Para hacer esto hay varias formas, como parámetros, archivos de configuración, variables de entorno o combinaciones de lo anterior.

Nosotros implementaremos variables de entorno, ya que se suelen usar mucho en los despliegues en la nube y son seguras de implementar.

```
@Override
public String getBotUsername() {
    return System.getenv("NOMBRE_BOT");
}
@Override
public String getBotToken() {
    return System.getenv("TOKEN_BOT");
}
```

Para agregarlo a Eclipse, desde "Run configurations", seleccionen la clase CopiameBot y seleccionen la solapa de Environment. Ahí pueden dar de alta las variables, como figura aca:



En el IntelliJ es más o menos parecido, TODAS las IDEs tienen una forma de modificar las variables de entorno.

3ra Iteración

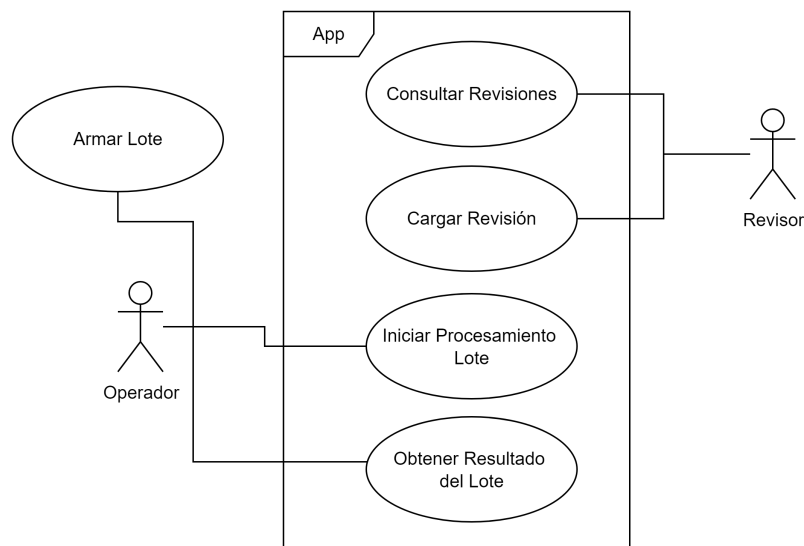
El objetivo de la 3ra iteración es implementar una API REST para los revisores, desplegarla en la nube y hacer que el BOT la pueda consumir.

El alcance será el siguiente:

- Implementar el nivel de servicio Plata, es decir, asignar pares de documentos a revisores, para que determinen si son copia o no.
- Dar acceso a lo anterior con una API REST
- Desplegar la API en el servicio PaaS Render
- Cambiar los comandos del Bot de Telegram y hacer que consuma la API REST, en lugar de procesar llamar directamente al dominio
- No se implementara seguridad de ningún tipo

Puede partir del repo que creó anteriormente o sino bajar los fuentes desde [acá](#)
Iteración v2.1.

Paso 0: Entender los nuevos requerimientos



Vemos que se agrega un nuevo actor, que tiene que poder ver qué revisiones tiene pendientes y cargar el puntaje en cada una.

Por otro lado, el procesamiento del lote ya no es inmediato (si se solicitan revisiones manuales), sino que se inicia el procesamiento, se reparten las revisiones y recién cuando todos los resultados fueron cargados, se puede obtener el resultado final.

Paso 1: Diseñar la API

Hay 2 acciones importantes que vamos a exponer para los revisores, que representan los 2 casos de usos en el diagrama anterior:

- 1) Listar las revisiones que deben realizar y los documentos de las mismas

GET /revisor/{id}/revisiones

```
[  
  {  
    "fecha": "ddmmaa",  
    "estado": "pendiente"  
  },  
  ...  
]
```

GET /revisor/{id}/revisiones/{revid}/1 -> retorna el contenido del primer archivo / doc

GET /revisor/{id}/revisiones/{revid}/2 -> retorna el contenido del segundo archivo / doc

- 2) Indicar que realizaron la revisión con el % de copia correspondiente

POST /revisor/{id}/revisiones/{revid}

```
body: {  
  "estado": "revisado",  
  "valorCopia": 0.1 // número entre cero y uno  
}
```

Ahora ponemos en la API, la acción que veníamos realizando:

- 3) Crear un Análisis de Copia subiendo un Lote

POST /analisis

→ el body debe ser un adjunto con el nombre "file" que tenga un zip con los documentos a procesar.

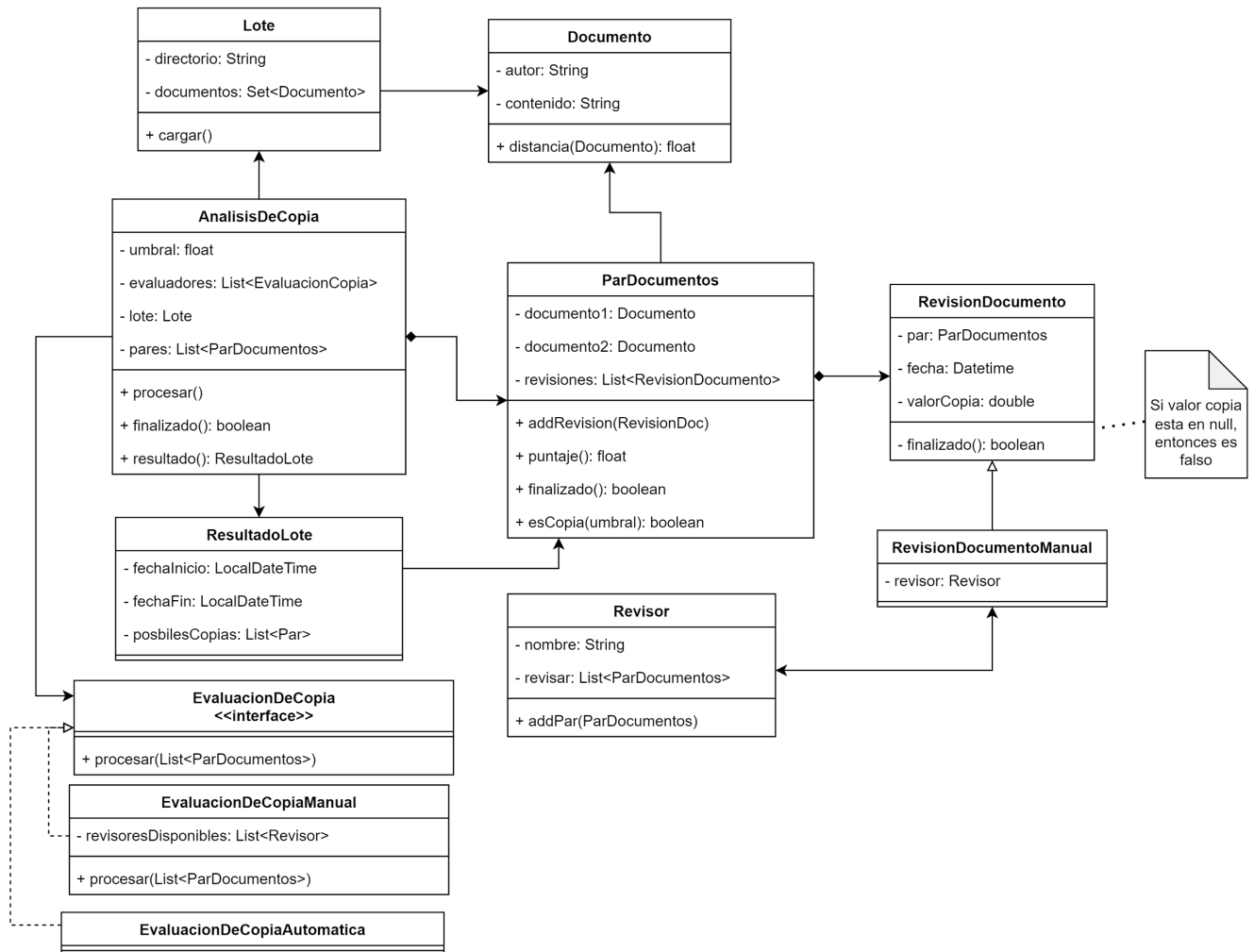
- 4) Listar Análisis de Copia

GET /analisis

```
[{analisis1},{analisis2},{analisis3},]
```

Paso 2: Actualizar el diseño

Vamos a actualizar el diseño para incorporar los requerimientos para la revisión manual. Donde introduciremos el concepto de revisor y quien asigna los revisores. Por otro lado, ya no es posible calcular de un solo paso automático, hay que esperar a que los revisores terminen.

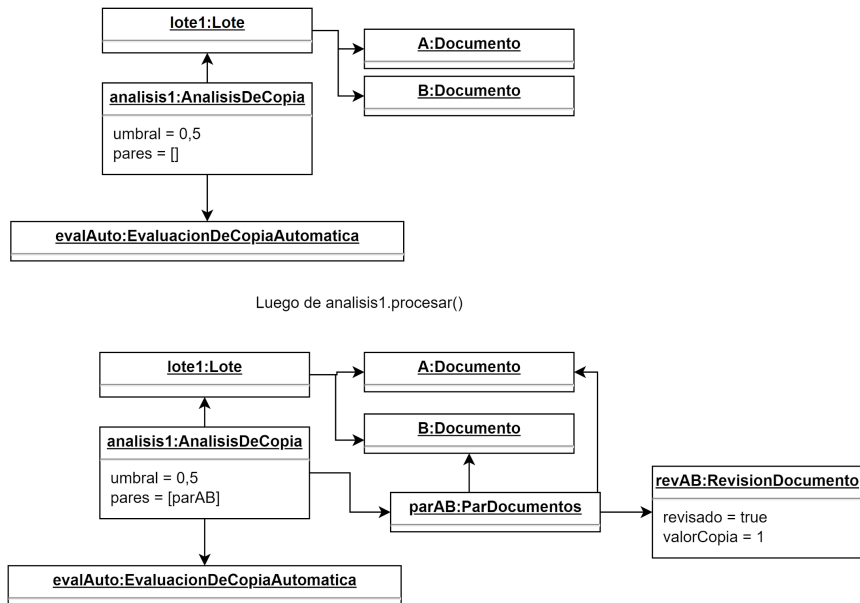


Podemos notar varias cosas:

- 1) Analisis de copia, ahora depende de evaluadores copia para hacer su trabajo
- 2) Se agrega el concepto de "finalizado", un análisis de copia sólo está finalizado cuando todos sus pares fueron revisados
- 3) Como un Par de documentos puede ser revisado automáticamente y por un revisor, vamos a tomar el promedio de ambas como puntaje final.

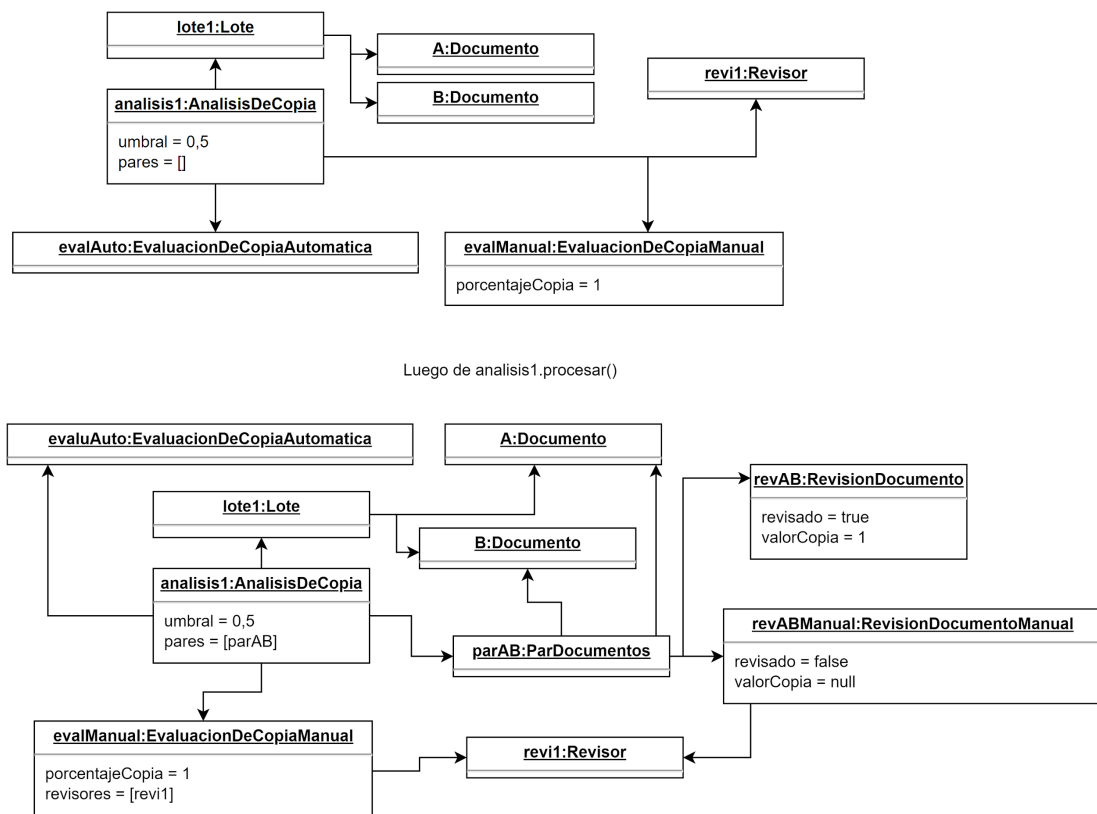
Escenario Bronce

Vamos a analizar el siguiente escenario, donde tenemos los documentos A,B. El análisis a realizar es con servicio “bronce”, es decir, sólo automático.



Escenario Plata

Vamos a analizar el siguiente escenario, donde tenemos los documentos A,B. El análisis a realizar es con servicio “plata”, es decir, automático y revisión simple.



Vemos que para que el analisis1 este terminado, el revisor rev1 debe completar la revision revABManual. Recién ahí se pueden calcular los puntos de parAB documentos. Por ejemplo, si parABManual tiene un valor copia de 0.6, el puntaje en parAB documento sería 0.6, por lo cual será mayor que el umbral y no caracterizada como una copia.

```
@Test
public void testPlataTexto() throws Exception {
    // Armado del Escenario
    Lote lote = new
    Lote("src/test/resources/lotel");
    lote.validar();
    lote.cargar();
    float umbral = 0.4f;
    AnalisisDeCopia analisis = new
    AnalisisDeCopia(umbral, lote);
    analisis.addEvaluador(new
    EvaluadorDeCopiaAutomatico());
    Revisor revisor = new Revisor();
    List<Revisor> revisores =
    Arrays.asList(revisor);
    EvaluadorDeCopiaManual eval = new
    EvaluadorDeCopiaManual(revisores, 1.0);
    analisis.addEvaluador(eval);
    // Ejecucion
    analisis.procesar();
    // Marco manualmente el valor de copia en un
    valor alto
    // 1 --> no se copiaron | 0 se copiaron y los
    docs son identicos
    for (RevisionDocumento revision :
    revisor.getRevisar()) {
        revision.setValorCopia(0.9f);
    }
    ResultadoLote resultado = analisis.resultado();
    // Chequeo
    assertEquals(0,
    resultado.getPosiblesCopias().size(),
    "La revisión manual cambio el resultado de la
    posible copia, con lo que no se tiene que
    detectar copia alguna");
}
```


Paso 3: Armado de los repositorio

Necesitamos un lugar donde ir “guardando” los análisis. En principio, el repositorio será en memoria (o sea se reinicia la App y se pierde todo) y en próximas iteraciones le agregaremos persistencia:

```
public class AnalisisRepository {
    private Map<String, AnalisisDeCopia> lista;
    public AnalisisRepository() {
        super();
        this.lista = new HashMap<String,
        AnalisisDeCopia>();
    }
    public void save(AnalisisDeCopia analisis) {
        String key =
        UUID.randomUUID().toString().substring(0,
        5);
        analisis.setId(key);
        this.lista.put(key, analisis);
    }
    public AnalisisDeCopia findById(String id) {
        return this.lista.get(id);
    }
    public Collection<AnalisisDeCopia> all() {
        return this.lista.values();
    }
}
```

Paso 4: Desarrollo de la API

Para implementar esta API utilizaremos el framework [JavalinIO](#), el cual permite la construcción de aplicaciones web. La misma nos deja levantar un servidor web y asociar pedidos HTTP que cumplen con una ruta específica a partes de nuestro código, clases que denominaremos controladores o controllers. Los mismos son capaces de procesar pedidos HTTP y retornar las respectivas respuestas.

Primero vamos a agregar las dependencias al pom:

```
<dependency>
  <groupId>io.javalin</groupId>
  <artifactId>javalin</artifactId>
  <version>5.4.2</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>2.0.6</version>
</dependency>
```

Ahora tenemos que armar la clase que inicia el servidor web y configura las rutas de los endpoints.

```
public class CopiameAPI {
  public static void main(String[] args) {

    Integer port = Integer.parseInt( System.getProperty("port", "8080"));
    Javalin app = Javalin.create().start(port);
    AnalsisRepository repo = new AnalsisRepository();

    app.get("/analisis", new AnalisisListController(repo));
    app.post("/analisis", new AnalisisAddController(repo));
    /*
    app.get("/revisor/{id}/revision",
        new RevisorRevisionesListController(repo));
    app.post("/revisor/{id}/revision",
        new RevisorAddRevisionController(repo));
    app.get("/revisor/{id}/revision/{rev}/file/{fileid}",
        new RevisorRevisionFilesController(repo));
    */
  }
}
```

Cada uno de los controller, implementan la interfaz `io.javalin.http.Handler`, que tiene el método “void handle(Context ctx) throws Exception”. En el mismo, el objeto Context, nos permite obtener datos del HTTP Request y generar el HTTP Response.

Veamos el controller más simple:

```
import io.javalin.http.Handler;

public class AnalisisListController implements Handler {
    private AnalisisRepository repo;
    public AnalisisListController(AnalisisRepository repo) {
        super();
        this.repo = repo;
    }
    @Override
    public void handle(Context ctx) throws Exception {
        ctx.json(repo.all());
    }
}
```

Ahora vamos a replicar lo que hace el bot de Telegram, es decir, cargar un lote mediante una subida de archivo.

```
public class AnalisisAddController implements Handler {
    private AnalisisRepository repo;
    public AnalisisAddController(AnalisisRepository repo) {
        super();
        this.repo = repo;
    }
    @Override
    public void handle(Context ctx) throws Exception {
        // Proceso de parámetros de entrada
        String destDirectory = "/tmp/unlugar";
        UnzipUtility.unzip(ctx.uploadedFile("file").content(),
            destDirectory);

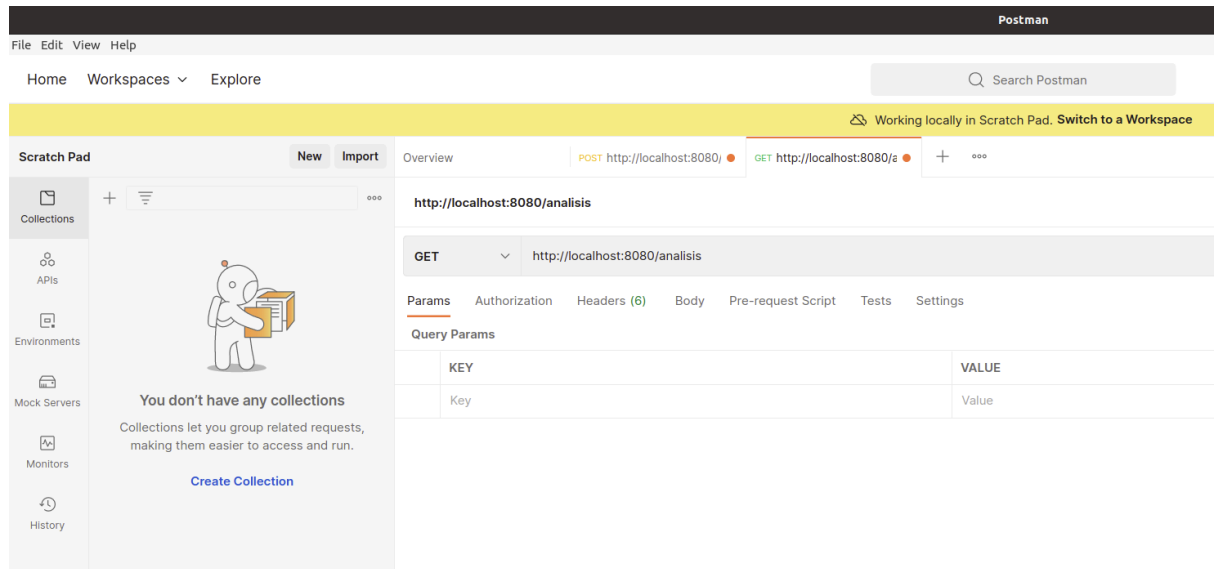
        // Armado del Análisis
        Lote lote = new Lote(destDirectory);
        lote.validar();
        lote.cargar();
        float umbral = 0.5f;
        AnalisisDeCopia analisis =
            new AnalisisDeCopia(umbral, lote);
        analisis.addEvaluador(new EvaluadorDeCopiaAutomatico());
        Revisor revisor = new Revisor();
        List<Revisor> revisores = Arrays.asList(revisor);
        EvaluadorDeCopiaManual eval =
            new EvaluadorDeCopiaManual(revisores, 1.0);
        analisis.addEvaluador(eval);

        // Proceso del lote
        analisis.procesar();
        // Guardar
        repo.save(analisis);

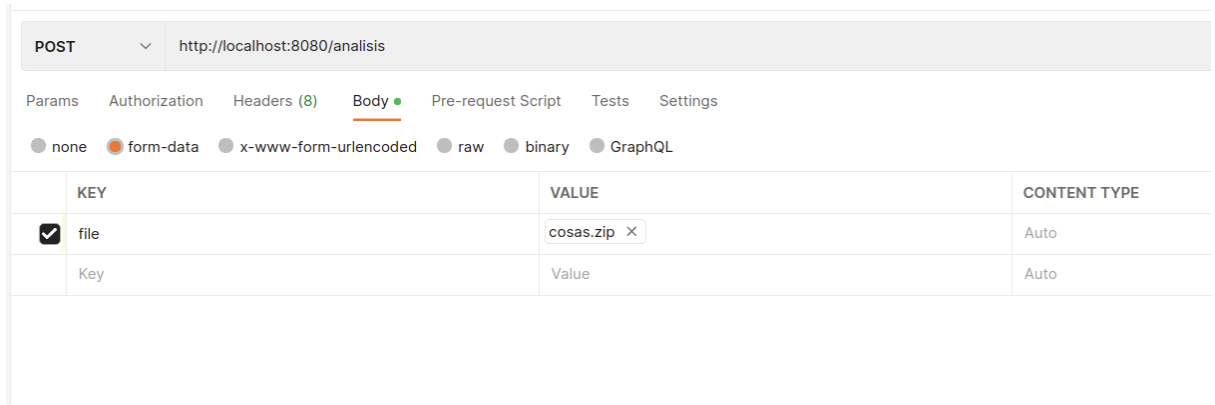
        // Armado de la respuesta
        Map<String, String> rta = new HashMap<String, String>();
        rta.put("analisis", analisis.getId());
        ctx.json(rta);
    }
}
```

Paso 5: Uso de la API

Para listar los análisis podemos usar el navegador web en la dirección <http://localhost:8080/analisis>. Pero para subir el archivo, necesitamos un cliente HTTP que nos de más funcionalidades. Para eso vamos a usar el cliente [Postman](#). Instalelo, es un poco críptico al principio, pero eventualmente tiene que ver una pantalla como esta:



Pruebe obtener un listado de los análisis, en principio le va a dar una lista vacía, pero es suficiente para ver que funciona. Ahora para probar que funciona el upload, prepare un archivo como el que uso en la iteración pasada para procesar un batch en el bot, pero lo vamos a subir desde el Postman:

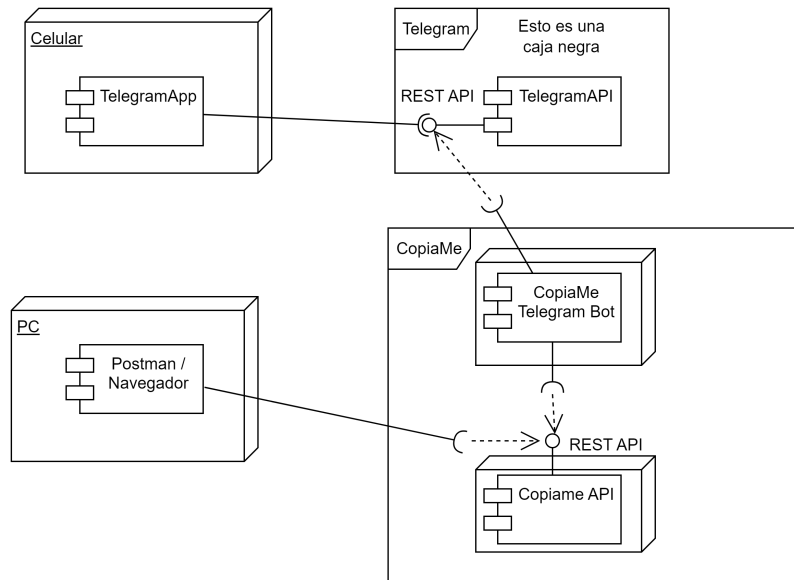


En la celda de la columna Key que dice file, del lado derecho, hay un drop oculto, que cuando le pasan el mouse por arriba, pueden elegir “File” en lugar de text. Si no lo encuentra puede ver este [video](#).

Verifique que se procesa el batch y consulte el listado de análisis a ver que le aparece.

Paso 6: Cambio del telegram de Bot

El bot que habíamos construido, ejecutaba directamente el dominio, es decir, tenía los objetos para resolver el procesamiento del lote. Ahora que tenemos una API, donde estamos desarrollando la persistencia, vamos a hacer que el bot use la misma para procesar los pedidos.



Si bien, nosotros tenemos todo en un proyecto, probablemente deberíamos separarlo, pero no lo vamos a hacer, para que el planteo siga siendo simple. Por otro lado, también estamos mostrando que la API y el Bot están en nodos distintos, ya que el despliegue de las mismas debería ser independiente. En particular, al final de esta práctica nosotros vamos a desplegar el bot en nuestras PCs y la api en un servidor PaaS llamado Render.

Ahora bien, veamos el código modificado para CopiameBot, primero vamos a agregar la siguiente dependencia al pom, que es el cliente HTTP para poder comunicarnos del Bot a la API.

```
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.13</version>
</dependency>
```

```

public void onUpdateReceived(Update update) {
    Message message = update.getMessage();
    if (message.hasDocument()) {
        Document document = message.getDocument();
        if (document.getMimeType().equals("application/zip")) {
            try {
                // Obtiene el archivo
                GetFile getFile = new GetFile();
                getFile.setFileId(message.getDocument().getFileId());
                ;
                org.telegram.telegrambots.meta.api.objects.File file
                = execute(getFile);
                java.io.File downloadedFile = downloadFile(file);
                // Envia el archivo a la API
                String rta = enviarLote(downloadedFile);
                System.out.println(rta);
                // Envia el mensaje al usuario
                SendMessage responseMsg = new SendMessage();
                responseMsg.setChatId(message.getChatId());
                responseMsg.setText(rta);
                execute(responseMsg);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

private String enviarLote(java.io.File downloadedFile) throws
IOException, ClientProtocolException {
    HttpClient httpClient = HttpClient.createDefault();
    HttpPost httpPost = new HttpPost(
        this.apiEndpoint + "/analisis");
    MultipartEntityBuilder builder =
        MultipartEntityBuilder.create();
    builder.addBinaryBody("file", downloadedFile,
        ContentType.DEFAULT_BINARY, "data.zip");
    HttpEntity multipart = builder.build();
    httpPost.setEntity(multipart);
    HttpResponse execute = httpClient.execute(httpPost);
    String rta = IOUtils.toString(
        execute.getEntity().getContent(),
        StandardCharsets.UTF_8.name());
    return rta;
}

```

Ahora implemente el código en el Bot y complete lo necesario para que al enviar un zip por Telegram, el mismo sea procesado por la API y envíe la respuesta.

Paso 7: Despliegue de la API en Render

Para desplegar la API, vamos a usar el PaaS [Render](#). Lo primero que vamos a hacer, es agregar un archivo Dockerfile, que es lo que utiliza Render para poder desplegar la aplicación. Los Dockerfile son archivos diseñados para ser procesados por el programa [Docker](#) y si bien **NO** es necesario para la continuación de esta práctica o desplegar en Render, es una herramienta super útil y puede ayudar en futuras prácticas⁴. En particular, puede hacer pruebas para ver si el archivo Dockerfile funciona localmente, antes de verificar eso haciendo deploy de la aplicación.

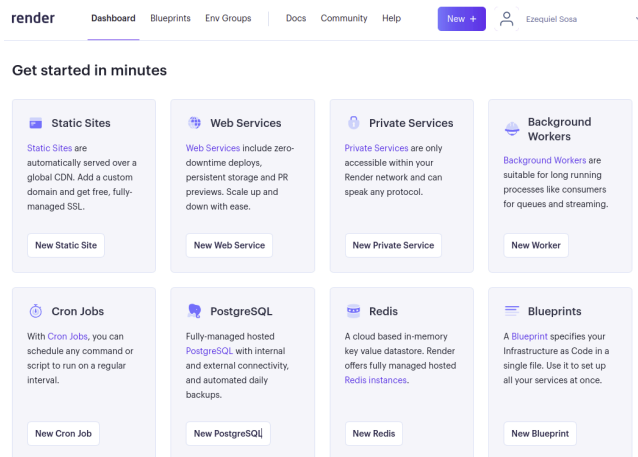
Vamos a crear el archivo Dockerfile (respetar mayúscula y minúsculas) en la raíz del proyecto (a la altura del pom.xml), no es necesario que entienda todo lo que hace, pero, lo importante es que tiene los pasos para armar la aplicación e indicar que se debe ejecutar para levantarla. Este es el texto del Dockerfile:

```
# syntax = docker/dockerfile:1.2
#
# Build stage
#
FROM maven:3.8.6-openjdk-18 AS build
COPY . .
RUN mvn clean package assembly:single -DskipTests

#
# Package stage
#
FROM openjdk:17-jdk-slim
COPY --from=build /target/copiame-cli-0.0.1-SNAPSHOT-jar-with-dependencies.jar copiame.jar
# ENV PORT=8080
EXPOSE 8080
ENTRYPOINT ["java", "-classpath", "copiame.jar", "ar.utn.dds.copiame.CopiameAPI"]
```

Ahora, entramos a la página y creamos un usuario. Desde el Dashboard vamos a crear un Web Service.

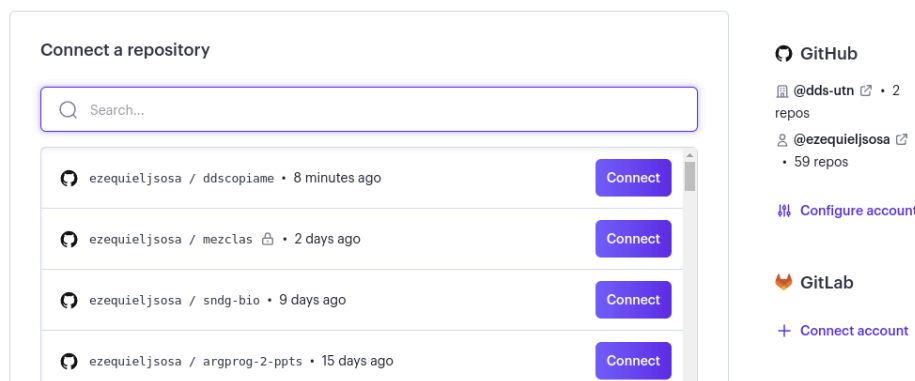
⁴ Para instalarlo en Windows, primero tiene que tener activada la virtualización en su PC (la mayoría de las pcs la tiene activada), puede ver este [link](#). Luego, debe instalar WSL, Docker Engine para Windows y finalmente una VM de Ubuntu dentro del marketplace de Windows, para hacer esto puede ver el siguiente [video](#), que si bien cambiaron algunas pantallas, si sigue la lógica y la secuencia que propone debería hacerlo funcionar sin problema. Si quiere un entorno para jugar un rato, puede usar lo siguiente: <https://labs.play-with-docker.com/>



Desde esta pantalla, vamos a elegir “+ Connect account” de Github, una vez que termine el proceso, debería ver el listado de repositorios de su usuario.

Create a new Web Service

Connect your Git repository or use an existing public repository URL.




Una vez que termine lo anterior, oprima connect sobre el repo indicado y le aparecerá una pantalla de este tipo.

WEB SERVICE

 **copiametest**

Docker

Free

 ezequieljsosa/ddscopiametest main

Connect ▾

Manual Deploy ▾

<https://copiametest.onrender.com> 


Events

Builds too slow? [Upgrade to a paid instance type](#) to go faster. Learn more about [free instance type limits](#).

Logs

Disks

April 30, 2023 at 1:53 PM

 In progress

Environment

ced8b2c Dockerfile

Shell

PRs

Search logs

Search

 Maximize

Scroll to top

Jobs

Metrics

Scaling

Settings

```
Apr 30 01:54:07 PM #18 exporting config sha256:d32922ee0354d27a51f9c362e4710f0a9c50e2efb9b8011da4a7d59273aa55d2 done
Apr 30 01:54:14 PM #18 DONE 8.1s
Apr 30 01:54:15 PM
Apr 30 01:54:15 PM #19 exporting content cache
Apr 30 01:54:15 PM #19 preparing build cache for export
Apr 30 01:54:37 PM Pushing image to registry...
Apr 30 01:54:41 PM Upload succeeded
Apr 30 01:54:41 PM DONE
Apr 30 01:54:36 PM #19 DONE 22.0s
Apr 30 01:54:48 PM SLF4J: No SLF4J providers were found.
Apr 30 01:54:48 PM SLF4J: Defaulting to no-operation (NOP) logger implementation
Apr 30 01:54:48 PM SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
```

En este punto no estamos usando muchas variables de entorno, pero mas adelante probablemente vayamos a usarlas. En principio la API tiene la variable PORT y el default es 8080, que justamente es el puerto que usa Render, pero si quisiéramos cambiarlo, este sería el lugar. En el ejemplo vemos por ejemplo, cómo sería setear las variables de entorno para el Bot (que no es lo que estamos haciendo ahora, ya que el bot, lo vamos a ejecutar localmente, es decir, en nuestra computadora).

WEB SERVICE

 **copiametest**

Docker

Free

 ezequieljsosa/ddscopiametest main

Connect

Manual Deploy

<https://copiametest.onrender.com>

Events

Logs

Disks

Environment

Shell

PRs

Jobs

Metrics

Scaling

Settings

Environment Variables


Use environment variables to store API keys and other configuration values and secrets. You can access them in your code like regular environment variables, for example with `os.getenv()` in Python or `process.env` in Node.

Key

Value


NOMBRE_BOT

.....



TOKEN_BOT

.....



Create Environment Group

Add Environment Variable

Save Changes

Secret Files

You can store secret files (like `.env` or `.npmrc` files and private keys) in Render. These files can be accessed during

Después de actualizar variables de entorno o el código de la aplicación (push al remoto), debemos ir a “Manual Deploy” → “Deploy latest commit” y aceptar los detalles de la siguiente pantalla.

You are deploying a web service for [ezequieljsosa/ddscopiametest](#).

You seem to be using **Docker**, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name

A unique name for your web service.

copiametest

Region

The [region](#) where your web service runs.

Oregon (US West)

Branch

The repository branch used for your web service.

main

Root Directory

Optional

Defaults to repository root. When you specify a [root directory](#) that is different from your repository root, Render runs all your commands in the [specified directory](#) and ignores changes outside the directory.

e.g. `src`

Runtime

The runtime for your web service.

Docker

Please [enter your payment information](#) to select an instance type with higher limits.

Instance Type	RAM	CPU	Price
<input checked="" type="radio"/> Free	512 MB	0.1 CPU	\$0 / month

Una vez desplegada la API:

- 1) Vuelva a probar su funcionamiento con el navegador y el Postman, agregue un par de lotes y vea si se actualiza la lista de análisis
- 2) Levante el bot de telegram desde su computadora, pero en lugar de apuntar la url a localhost, hágalo contra la url que le provee Render

Paso 8: Implementar revisiones

Finalmente, implemente los controllers relacionados con los revisores: listar sus revisiones, poner puntuación y descargar los textos. Nuevamente, en esta etapa va a estar todo en memoria, pero igualmente cree 2 repositorios: RepoRevisores y RepoRevisiones, RepoRevisores puede inicializarlo con datos fijos cuando se levanta la app y RepoRevisiones debería actualizarlo luego de que procesa el análisis de copia.

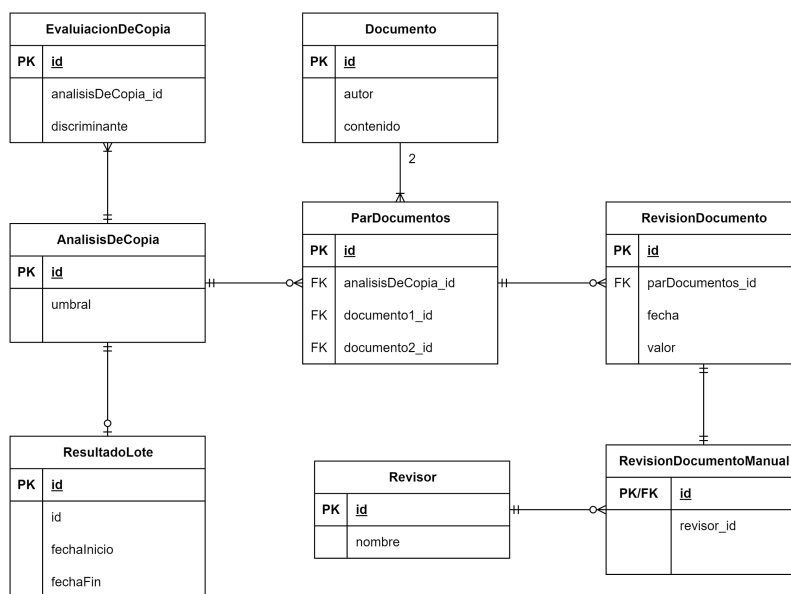
4ta Iteración

El objetivo de esta iteración es implementar la persistencia en el proyecto. Para ello utilizaremos al ORM Hibernate y la base de datos Postgres provista por Render.

Para arrancar, recomiendo crear un repo nuevo e importar el contenido de la rama iteración3 una de este [repo](#). Lo más fácil es bajar los fuentes directamente desde [acá](#), release “Iteración 3”

Paso 1: Modelo relacional

A continuación tenemos el siguiente diagrama de datos, a nivel físico, de las clases implementadas hasta el momento.



Decisiones tomadas:

- Lote no se persiste: en definitiva, solo se usa para crear el análisis de copia, luego este último posee todos los datos necesarios.
- Para el impedance mismatch de identidad, se declararon ids numéricos sintéticos autoincrementales en todas las clases.
- Se cambió EvaluacionDeCopia de interfaz a clase abstracta, con el objetivo de mapear la jerarquía de manera más simple y hacer un uso polimórfico más sencillo, desde las consultas. Como estrategia de herencia, se eligió SingleTable. Por otro lado, NO vamos a persistir el listado de revisores en EvaluacionDeCopiaManual, ya que vamos a cambiar cómo se obtienen los revisores.
- La herencia de RevisionDocumento, se mapeara como Joined.

Paso 2: Puesta en marcha del ORM

Vamos a usar la implementación Hibernate de JPA para realizar la persistencia de nuestra aplicación. Primero, vamos a agregar las dependencias al pom.xml, primero la base de datos [H2](#), que utilizaremos para hacer pruebas simples y luego la base de datos Postgres, que es la que usaremos para producción y finalmente Hibernate (la versión 6 cambia bastante la API, así que si decide usarla, tendrá que adaptar bastante código)

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>testing</scope>
  <version>2.1.214</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.6.0</version>
</dependency>

<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>5.6.15.Final</version>
</dependency>
```

Una vez configurado esto, vamos a completar el archivo de configuración que usa JPA, el persistence.xml. El mismo debe estar dentro del directorio META-INF, por lo que para pruebas, vamos a crear uno en src/test/resources/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd" version="2.2">
  <persistence-unit name="copiamedb" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <properties>
      <!-- Configuración de la fuente de datos -->
      <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:test" />
      <!-- con esto decimos que use una DB en memoria -->

      <property name="javax.persistence.jdbc.user" value="sa" />
      <property name="javax.persistence.jdbc.password" value="" />

      <!-- Creación de tablas -->
      <property name="hibernate.flushMode" value="FLUSH_AUTO" />
      <property name="hibernate.hbm2ddl.auto" value="create" />

      <!-- Mostrar las sentencias de SQL por consola -->
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.use_sql_comments" value="true" />

      <property name="hibernate.connection.pool_size" value="1" />
    </properties>
  </persistence-unit>
</persistence>
```

Notar la línea `pool_size=1`, esto nos sirve para **no** abrir múltiples conexiones a la DB, que es correcto para un ambiente de testing⁵. Ahora armamos un test de integración para probar la persistencia, notar que cuando se crea el `EntityManagerFactory`, coincide el nombre `copiamedb` que está en el XML: “... `<persistence-unit name="copiamedb" ...`”

```
public class PersistenceIT {

    static EntityManagerFactory entityManagerFactory ;
    EntityManager entityManager ;

    @BeforeAll
    public static void setUpClass() throws Exception {
        entityManagerFactory =
            Persistence.createEntityManagerFactory("copiamedb");
    }
    @BeforeEach
    public void setup() throws Exception {
        entityManager = entityManagerFactory.createEntityManager();
    }
    @Test
    public void testConectar() {
        // vacío, para ver que levante el ORM
    }
}
```

Notar que el `EntityManagerFactory` debe crearse al inicio. Acá estamos probando la configuración del ORM, y si bien H2 es una base que soporta la mayoría de las sentencias SQL, es un entorno que dista un poco del real, en principio, la comunicación entre nuestra app y ella no es a través de puertos.

Paso 3: Primer mapeo

Ahora que sabemos que la conexión con la DB a través del ORM está ok, vamos a probar “persistir” nuestra primer clase. Para eso vamos a mapear la clase `Documento`, ya que no está acoplada a ninguna otra.

```
@Entity
public class Documento {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String autor;
    private String contenido;

    protected Documento() {
        super();
    }
}
```

5

La anotación Entity avisa que es una clase que voy a querer persistir, por otro lado, me veo obligado a agregar un campo id al modelo, el cual indico que debe generarse automáticamente. Por cuestiones de compatibilidad, conviene elegir como estrategia IDENTITY. Por otro lado, notar que se agregó un constructor vacío, ahora que usamos un ORM, todas las clases necesitarán un constructor vacío y todos los getters y setters. Si se desea manejar el scope, se pueden dejar como protección, pero de ahora en adelante, será obligatorio tener estos métodos. Esto último se denomina contrato de “bean” y es muy común que los frameworks lo asuman, sino, hibernate en particular nos avisara: “no tenes el constructor vacío”.

En el persistence.xml, tenemos que agregar este elemento, a la misma altura que “provider”

```
<class>ar.utn.dds.copiame.Documento</class>
```

Ahora vamos a agregar al test, una funcionalidad para guardar y recuperar el objeto:

```
@Test
public void testGuardarYRecuperarDoc() throws Exception {
    Documento doc1 = new Documento("pepe", "cosas");
    entityManager.getTransaction().begin();
    entityManager.persist(doc1);
    entityManager.getTransaction().commit();
    entityManager.close();

    entityManager = entityManagerFactory.createEntityManager();
    Documento doc2 = entityManager.find(Documento.class, 1L);

    assertEquals(doc1.getContenido(), doc2.getContenido()); // también
    puede redefinir el equals
}
```

Estrictamente, lo anterior es un test trivial, pq se supone que hibernate va a persistir bien las cosas, siempre y cuando los mapeos estén bien hechos. Pero a nosotros nos sirve para ir validando que vamos bien.

Si salio todo bien, los tests deberían estar verdes y visualizar una salida así:

```
Hibernate:
drop table if exists Documento CASCADE
Jun 03, 2023 4:53:22 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess
[org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@6d33a66e] for (non-JTA) DDL execution
was not in auto-commit mode; the Connection 'local transaction' will be committed and the Connection will be set into auto-commit mode.
Hibernate:
create table Documento (
  id bigint generated by default as identity,
  autor varchar(255),
  contenido varchar(255),
  primary key (id)
)
Jun 03, 2023 4:53:22 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess
[org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@68699afc] for (non-JTA) DDL execution
was not in auto-commit mode; the Connection 'local transaction' will be committed and the Connection will be set into auto-commit mode.
Jun 03, 2023 4:53:22 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate:
/* insert ar.utn.dds.copiame.Documento
*/ insert
into
Documento
(id, autor, contenido)
values
(default, ?, ?)
Hibernate:
select
documento0_.id as id1_0_0_,
documento0_.autor as autor2_0_0_,
documento0_.contenido as contenid3_0_0_
from
Documento documento0_
where
documento0_.id=?
```

Como vemos creó la DB, hizo el select y salió todo bien, pero como se hizo todo en memoria, los más desconfiados quizás quieran hacer la siguiente actividad opcional.

Paso Opcional 1: Verificar que funcione en Postgres

En este punto, teóricamente, por más que cambiemos de DB relacional, entre las capas de abstracción de JDBC y JPA, dicho cambio debería ser transparente, pero a veces no lo es... Y como en la práctica vamos a usar una DB postgres, vamos a verificar que la misma funciona. Primero que nada, recomendamos descargar el programa [DBeaver](#) (o el cliente que quiera) e instalar el motor de base de datos [Postgresql](#)⁶ (RECUERDE/ANOTE QUE CONTRASEÑA PUSO, SINO DESPUÉS RECUPERARLA ES UN PARTO). Una vez instalado, verifique que se esté ejecutando correctamente, conéctese con el cliente que haya decidido descargarse (el usuario por default es "postgres") y cree la base de datos "copiamedb". Puede notar que algunas cosas no son muy intuitivas, pero con un poco de paciencia, sentido común y pensamiento lateral, salen.

Una vez que la DB esta creada dentro del postgres, modifique el persistence.xml (será solo para esta prueba) las lineas de conexion a la DB:

```
<property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
  <property name="javax.persistence.jdbc.url"
value="jdbc:postgresql://localhost:5432/copiamedb"/>
  <property name="javax.persistence.jdbc.user" value="postgres"/>
  <property name="javax.persistence.jdbc.password" value="ACA VA LA
CONTRASEÑA QUE PUSO CUANDO INSTALO" />
```

Para la url "jdbc:postgresql://localhost:5432/copiamedb" , como la DB está en su misma computadora, la dirección es localhost , si no toco la configuración del puerto, el por defecto de postgres es el 5432 y copiamedb lo creo en el paso anterior.

Ahora vuelva a correr el test y vea usando el DBeaver que el documento esta creado correctamente.

⁶ si tiene instalado docker, puede correr la siguiente línea:

```
docker run --name postgres-container -e POSTGRES_PASSWORD=123 -v
$PWD/db:/var/lib/postgresql/data -p 5432:5432 -it --rm postgres:15
```


Paso 4: Mapeo Completo

Ahora vamos a completar los mapeos del resto de las clases. Lo vamos a hacer de una, pero como recomendación, sobre todo si está arrancando, vaya mapeando las clases de a poco, puede usar la anotación `@Transient` cuando hay dependencias para ignorarlas y no tener que mapear todo de golpe.

Rápidamente, tenemos que agregar al `persistence.xml` todas las clases, como hicimos con documento, anotar las que queremos persistir con `@Entity`, agregarles un ID, constructor vacío y getters y setters (con scope `protected` si quiere)

Ahora vamos a hacer hincapié en mapear las relaciones y las estrategias de herencia. Arrancamos con `AnalisisDeCopia`:

```
@Entity
public class AnalisisDeCopia {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long pk;
    private String id;
    private float umbral;
    @Transient
    private Lote lote;
    @OneToMany
    @JoinColumn(name = "analisis_id")
    private List<ParDocumentos> pares;
    @OneToMany
    @JoinColumn(name = "analisis_id")
    private List<EvaluadorDeCopia> evaluadores;
    @OneToOne
    private ResultadoLote rl;
```

Como ya teníamos un id, a la PK le pusimos “pk”. Por otro lado, como no mapeamos el lote, lo anotamos como `Transient`, para que el ORM no lo tenga en cuenta. Después tanto los pares de documentos como los evaluadores SON del análisis, con lo cual lo puedo mapear con una anotación `OneToMany` (un análisis a muchos pares / evaluadores), y que los pares y evaluadores tengan el atributo `analisis_id`. Por otro lado, como solo tenemos un resultado lote (o sea, con esto NO puedo persistir múltiples), lo persisto con `one to one`.

Lo que hay que hacer es bastante largo, así que para verificar que funciona bien, podemos implementar el test “`testGuardarYRecuperarAnalisis`” que se muestra más abajo. En el tag “iteración 4” el mapeo está completo, yo les recomiendo intentar que funcione y usar el resuelto para revisar nomas.

Para que sea más o menos fácil, puede poner varias cosas en cascada, cosa de que cuando guarde se propague a todos los objetos, por otro lado, mas abajo tienen el código de los repositorios `RevisorRepository` y `AnalisisDeCopiaRepository`. Notar que este último tiene 2 implementaciones, una en memoria (la que se usó la iteración pasada) y la JPA, que usamos acá. Se hizo una interfaz común para poder usarlos.

```

@Test
public void testGuardarYRecuperarAnalisis() throws Exception {
    // Pre condiciones: se supone que el revisor esta dado de alta ANTES de cargar el lote
    Revisor revisor = new Revisor();
    // Guardamos los documentos
    entityManager.getTransaction().begin();
    revisorRepo.save(revisor);
    entityManager.getTransaction().commit();
    entityManager.close();
    // Notar que volvemos a inicializar la persistencia
    entityManager = entityManagerFactory.createEntityManager();
    docRepo = new DocumentoRepository(entityManager);
    revisorRepo = new RevisorRepository(entityManager);
    List<Revisor> revisores = revisorRepo.all();
    Lote lote = new Lote("src/test/resources/lote1");
    lote.validar();
    lote.cargar();
    float umbral = 0.4f;
    AnalisisDeCopia analisis = new AnalisisDeCopia(umbral, lote);
    analisis.setId(UUID.randomUUID().toString());
    analisis.addEvaluador(new EvaluadorDeCopiaAutomatico());
    EvaluadorDeCopiaManual eval = new EvaluadorDeCopiaManual(revisores, 1.0);
    analisis.addEvaluador(eval);
    // Ejecución
    analisis.procesar();
    entityManager.getTransaction().begin();
    entityManager.persist(analisis);
    entityManager.getTransaction().commit();
    entityManager.close();
    // Otra interaccion
    entityManager = entityManagerFactory.createEntityManager();
    revisorRepo = new RevisorRepository(entityManager);
    entityManager.getTransaction().begin();
    Revisor revisorX = revisorRepo.findById(revisor.getId());
    // Marco manualmente el valor de copia en un valor alto
    // 1 --> no se copiaron | 0 se copiaron y los docs son identicos
    for (RevisionDocumento revision : revisorX.getRevisar()) {
        revision.setValorCopia(0.9f);
    }
    entityManager.getTransaction().commit();
    entityManager.close();

    entityManager = entityManagerFactory.createEntityManager();
    AnalisisJPAREpository analisisRepo = new AnalisisJPAREpository(entityManager);
    entityManager.getTransaction().begin();
    // Notar que VOLVEMOS a traer el análisis
    analisis = analisisRepo.findById(analisis.getId());
    ResultadoLote resultado = analisis.resultado();
    entityManager.getTransaction().commit();
    // Chequeo
    assertTrue(analisis.finalizado());
    assertEquals(0, resultado.getPosiblesCopias().size(),
        "La revisión manual cambio el resultado de la posible copia, con lo que no se tiene que detectar copia alguna");
}

```

```

public class RevisorRepository {
    private EntityManager entityManager ;

    public RevisorRepository(EntityManager entityManager) {
        super();
        this.entityManager = entityManager;
    }

    public void save(Revisor revisor) {
        this.entityManager.persist(revisor);
    }

    public Revisor findById(Long id) {
        return this.entityManager.find(Revisor.class, id);
    }

    public List<Revisor> all() {
        CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
        CriteriaQuery<Revisor> criteriaQuery =
            criteriaBuilder.createQuery(Revisor.class);
        Root<Revisor> root = criteriaQuery.from(Revisor.class);
        criteriaQuery.select(root);
        return entityManager.createQuery(criteriaQuery).getResultList();
    }
}

public class AnalisisJPAREpository implements AnalisisRepository {
    private EntityManager entityManager ;

    public AnalisisJPAREpository(EntityManager entityManager) {
        super();
        this.entityManager = entityManager;
    }

    @Override
    public void save(AnalisisDeCopia analisis) {
        this.entityManager.persist(analisis);
    }

    @Override
    public AnalisisDeCopia findById(String id) {
        // Notar que esto no es la PK, queda poco consistente respecto al
        resto de las clases
        CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
        CriteriaQuery<AnalisisDeCopia> criteriaQuery =
            criteriaBuilder.createQuery(AnalisisDeCopia.class);
        Root<AnalisisDeCopia> root = criteriaQuery.from(AnalisisDeCopia.class);
        criteriaQuery.select(root).where(criteriaBuilder.equal(
            root.get("id"), id));
        return entityManager.createQuery(criteriaQuery).getSingleResult();
    }

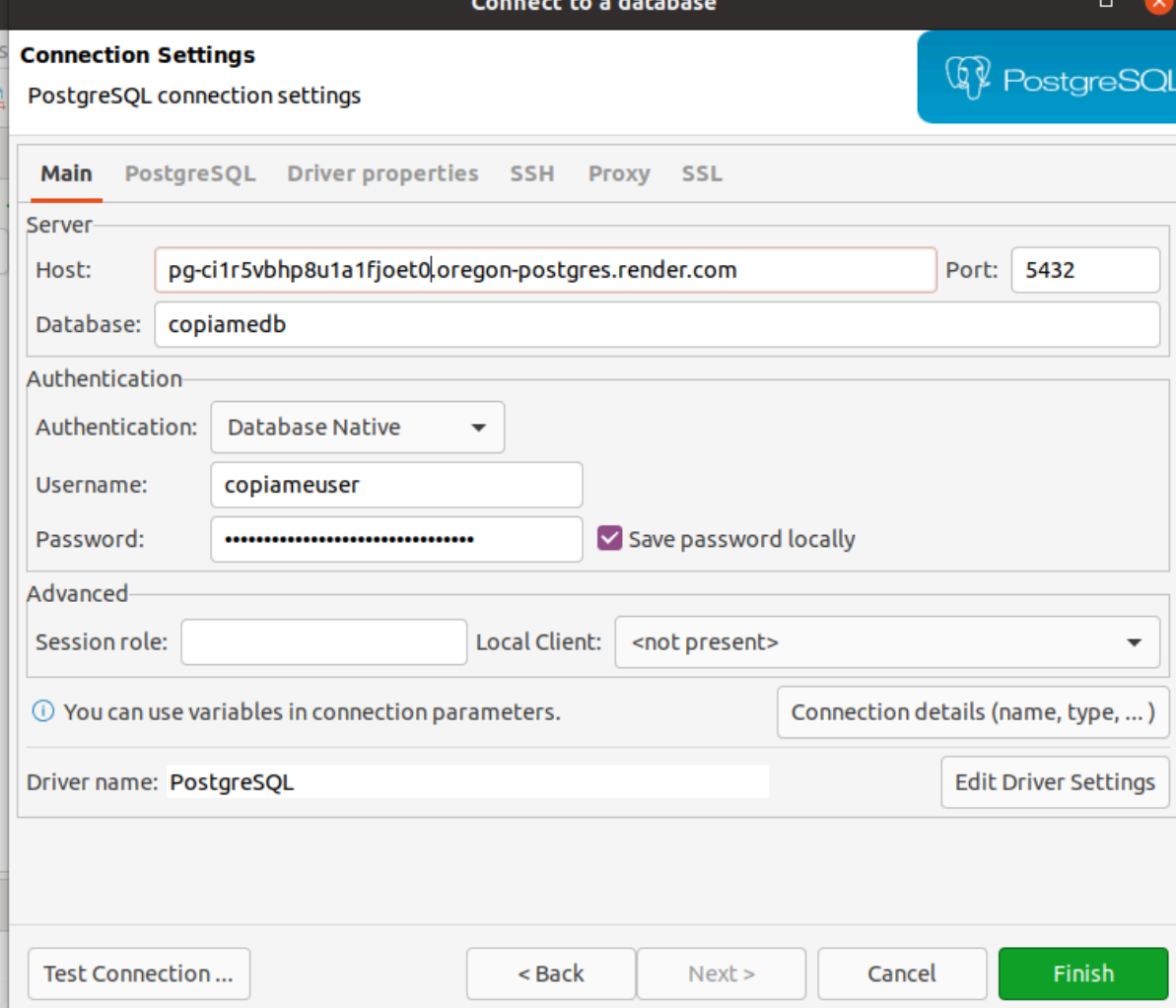
    @Override
    public Collection<AnalisisDeCopia> all() {
        return null;
    }
}

```

Paso 5: Deploy de la DB

Como el paso anterior, puede tomar cierto tiempo, si quiere puede partir del branch “iteracion4.5”, que ya tiene los mapeos hechos.

Primero vamos a crear una instancia de DB PostgreSQL en Render, yendo al botón “New +” en la parte superior derecha de la pantalla y eligiendo PostgreSQL. Luego llenamos los datos (tome nota de los mismos!) y dejamos todo por defecto, asegúrese de elegir la instancia gratuita. Cuando la base esté inicializada, puede ir a la propiedad: “External Database URL” y verificar usando DBeaver que puede conectarse a la base de datos.



The screenshot shows the 'Connect to a database' dialog box in DBeaver, specifically the 'PostgreSQL connection settings' tab. The 'Main' sub-tab is active, showing fields for 'Host', 'Port', 'Database', 'Authentication', 'Username', 'Password', 'Session role', and 'Local Client'. The 'Host' field contains 'pg-ci1r5vbhp8u1a1fjoet0.oregon-postgres.render.com', 'Port' is '5432', and 'Database' is 'copiamedb'. The 'Authentication' dropdown is set to 'Database Native', 'Username' is 'copiameuser', and 'Password' is masked with dots. The 'Save password locally' checkbox is checked. The 'Session role' is empty and 'Local Client' is '<not present>'. At the bottom, there are buttons for 'Test Connection ...', '< Back', 'Next >', 'Cancel', and a green 'Finish' button. A note at the bottom left states: 'You can use variables in connection parameters.' and a button 'Connection details (name, type, ...)' is next to it. The 'Driver name' is set to 'PostgreSQL' and there is an 'Edit Driver Settings' button.

Ahora bien, para configurar en su persistence.xml el usuario y la contraseña, NO, repito NO debería subir contraseñas, direcciones ni usuarios al repo. Entonces, vamos a generar una configuración que “complete” el persistence.xml usando variables de entorno, vamos a ver como queda la clase CopiameAPI. Solo vamos a implementar el alta y obtención del Revisor.

```

public class CopiameAPI {
    public static EntityManagerFactory entityManagerFactory;
    public static void main(String[] args) {
        startEntityManagerFactory();
        Integer port = Integer.parseInt(System.getProperty("PORT",
            "8080"));
        Javalin app = Javalin.create().start(port);
        AnalisisRepository repo = new AnalisisInMemoryRepository();
        app.get("/analisis", new AnalisisListController(repo));
        app.post("/analisis", new AnalisisAddController(repo));

        app.get("/revisor/{id}",
            new RevisorGetController(entityManagerFactory));
        app.post("/revisor/",
            new RevisorPostController(entityManagerFactory));
    }

    public static void startEntityManagerFactory() {
        //
        https://stackoverflow.com/questions/8836834/read-environment-variables-in-persistence-xml-file
        Map<String, String> env = System.getenv();
        Map<String, Object> configOverrides = new HashMap<String,
            Object>();
        String[] keys = new String[] { "javax.persistence.jdbc.url",
            "javax.persistence.jdbc.user",
            "javax.persistence.jdbc.password",
            "javax.persistence.jdbc.driver", "hibernate.hbm2ddl.auto",
            "hibernate.connection.pool_size", "hibernate.show_sql" };
        for (String key : keys) {
            if (env.containsKey(key)) {
                String value = env.get(key);
                configOverrides.put(key, value);
            }
        }
        entityManagerFactory =
            Persistence.createEntityManagerFactory("db", configOverrides);
    }
}

```

Debemos dejar un src/main/resources/persistence.xml “pelado” (notar que el otro que usamos para test, está en src/test !, lo que estamos haciendo ahora es otro nuevo)

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
    version="2.0">

    <persistence-unit name="db" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

        <properties>
            <property name="hibernate.default_schema" value="public"/>

```

```

        <property name="hibernate.connection.pool_size" value="10" />
        <property name="hibernate.archive.autodetection" value="class" />
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.format_sql" value="true" />
        <property name="use_sql_comments" value="true" />
    </properties>

</persistence-unit>

</persistence>

```

Finalmente programamos los controller:

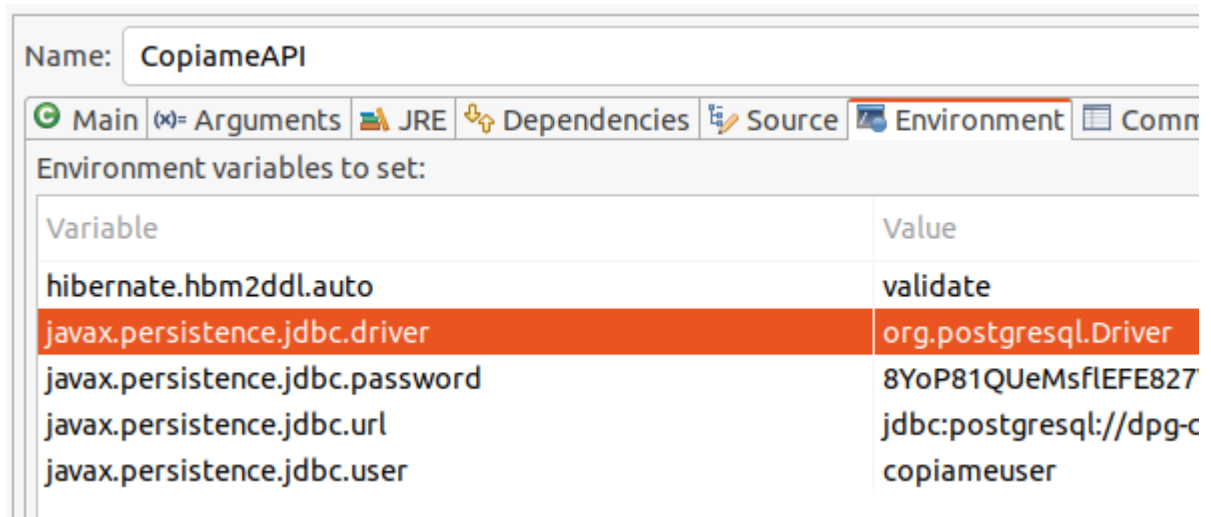
```

public class RevisorGetController implements Handler {
    private EntityManagerFactory entityManagerFactory;
    public RevisorGetController(EntityManagerFactory entityManagerFactory)
    {
        this.entityManagerFactory = entityManagerFactory;
    }
    @Override
    public void handle(Context ctx) throws Exception {
        RevisorRepository repo = new
        RevisorRepository(entityManagerFactory.createEntityManager());
        long revisorId = Long.parseLong(ctx.pathParam("id"));
        Revisor revisor = repo.findById(revisorId);
        ctx.json(revisor);
    }
}

public class RevisorPostController implements Handler {
    private EntityManagerFactory entityManagerFactory;
    public RevisorPostController(EntityManagerFactory
    entityManagerFactory) {
        this.entityManagerFactory = entityManagerFactory;
    }
    @Override
    public void handle(Context ctx) throws Exception {
        EntityManager em = entityManagerFactory.createEntityManager();
        RevisorRepository repo = new RevisorRepository(em);
        Revisor revisor = ctx.bodyAsClass(Revisor.class);
        em.getTransaction().begin();
        repo.save(revisor);
        em.getTransaction().commit();
        em.close();
        ctx.json(revisor);
    }
}

```

Ahora, pruebe configurar las variables de entorno apuntando a su base de produccion de Render.com, pero ejecutar local, por ejemplo:



Notar que el jdbc tiene esta forma:

jdbc:postgresql://xxxx-postgres.render.com/copiamedb

a diferencia de lo que dice en la web de Render, que arranca con postgres:// ...

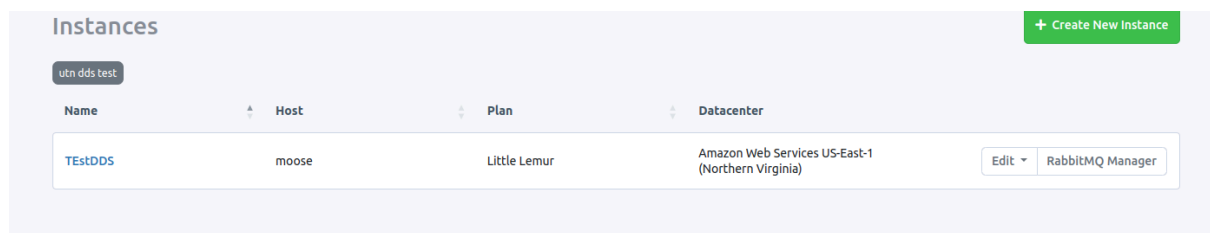
5ta Iteración

El objetivo de esta iteración es implementar un worker para el procesamiento de la detección automática. El alcance será el siguiente:

- Crear una worker para que procese los análisis automáticos
- Comunicar la app web y el worker

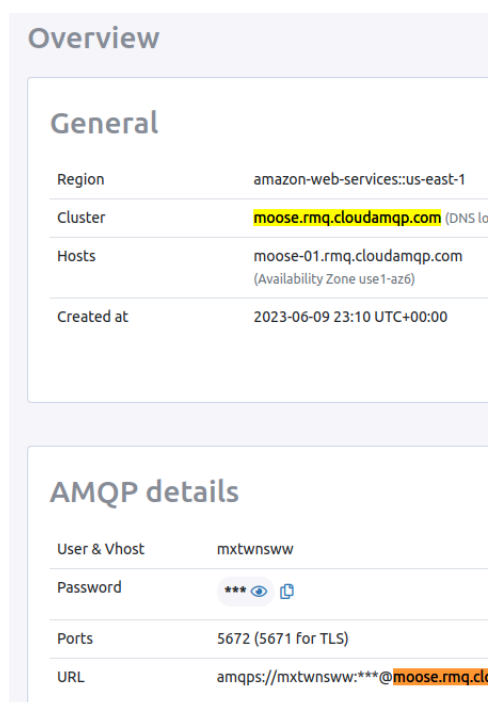
Paso 1: Crear cuenta en un CloudAMQP

Ir al sitio <https://www.cloudamqp.com/> y crearse una cuenta. Luego dar de alta una instancia gratuita (Little Lemur), elija cualquier región / datacenter, al final les tiene que aparecer algo así:



Instances				+ Create New Instance
Name	Host	Plan	Datacenter	
TestDDS	moose	Little Lemur	Amazon Web Services US-East-1 (Northern Virginia)	Edit RabbitMQ Manager

Haciendo click sobre la instancia tendrá los datos necesarios para configurar la mensajería. En particular da 2 opciones: AMQP y MQTT, vamos a usar el primero, el segundo implementa un protocolo más simple, en general utilizado para la publicación de información de sensores. Entonces vamos a ver algo así:



Overview	
General	
Region	amazon-web-services::us-east-1
Cluster	moose.rmq.cloudamqp.com (DNS lo
Hosts	moose-01.rmq.cloudamqp.com (Availability Zone use1-az6)
Created at	2023-06-09 23:10 UTC+00:00

AMQP details	
User & Vhost	mxtwnsww
Password	*** Show Copy
Ports	5672 (5671 for TLS)
URL	amqps://mxtwnsww:***@moose.rmq.cl

Lo que dice “cluster” va a ser el host, el user y vhost, son el mismo dato y el password está oculto en un campo, con estos 4 datos configuraremos las variables de entorno necesarias para conectarnos a la mensajería.

Paso 2: Crear un worker para que levante los paquetes

Ahora vamos a crear una nueva app (nuevamente siempre en el mismo repo), llamada CopiaMeWorker. El mismo se conectara a la mensajería y obtendrá mensajes de una cola determinada. Primero agregamos la dependencia de Maven.

```
<!-- https://mvnrepository.com/artifact/com.rabbitmq/amqp-client -->
<dependency>
<groupId>com.rabbitmq</groupId>
<artifactId>amqp-client</artifactId>
<version>5.17.0</version>
</dependency>

<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.datatype/jackson-dat
atype-jsr310 -->
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>2.15.2</version>
</dependency>
```

Luego creamos la clase Worker con su correspondiente main:

```
public class CopiaMeWorker extends DefaultConsumer {

    private String queueName;
    private EntityManagerFactory entityManagerFactory;

    protected CopiaMeWorker(Channel channel, String queueName) {
        super(channel);
        this.queueName = queueName;
        this.entityManagerFactory = entityManagerFactory;
    }

    private void init() throws IOException {
        // Declarar la cola desde la cual consumir mensajes
        this.getChannel().queueDeclare(this.queueName, false, false, false, null);
        // Consumir mensajes de la cola
        this.getChannel().basicConsume(this.queueName, false, this);
    }

    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        // Confirmar la recepción del mensaje a la mensajería
        this.getChannel().basicAck(envelope.getDeliveryTag(), false);
        String analisisId = new String(body, "UTF-8");
        System.out.println("se recibio el siguiente payload:");
        System.out.println( analisisId);
    }

    public static void main(String[] args) throws Exception {
        // Establecer la conexión con CloudAMQP
        Map<String, String> env = System.getenv();
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(env.get("QUEUE_HOST"));
        factory.setUsername(env.get("QUEUE_USERNAME"));
        factory.setPassword(env.get("QUEUE_PASSWORD"));
        // En el plan más barato, el VHOST == USER
    }
}
```

```

factory.setVirtualHost(env.get("QUEUE_USERNAME"));
String queueName = env.get("QUEUE_NAME");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

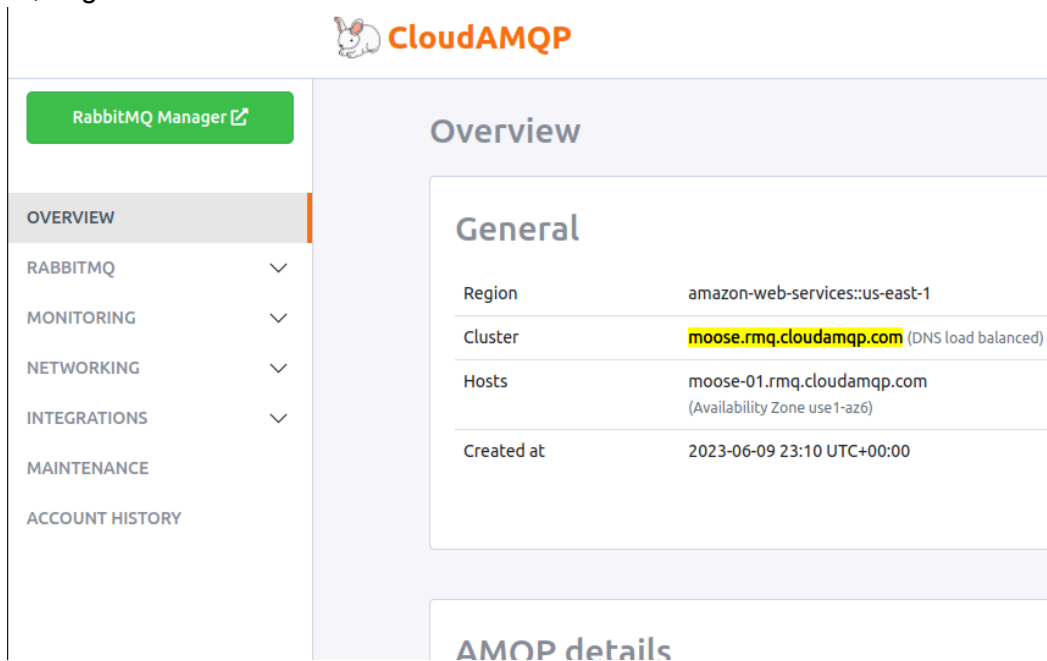
CopiaMeWorker worker = new CopiaMeWorker(channel, queueName);
worker.init();
}

}

```

Sobre el main están las variables de entorno que tienen que completar con la información del paso anterior. QUEUE name puede ser cualquier cosa. Una vez que terminen, tendrán un proceso listo, que ejecutará “handleDelivery” cada vez que se publique un mensaje en dicha cola. Asegurense de que el proceso esté corriendo sin errores.

Para hacer esto, volvamos a la pantalla de Cloud AMQP, entremos a nuestra instancia y veamos que en la parte superior izquierda hay un botón de “RabbitMQ Manager”, hagan click sobre el mismo.



The screenshot shows the CloudAMQP RabbitMQ Manager interface. At the top, there's a green button labeled "RabbitMQ Manager" with an external link icon. Below it is a sidebar menu with options: OVERVIEW (selected), RABBITMQ, MONITORING, NETWORKING, INTEGRATIONS, MAINTENANCE, and ACCOUNT HISTORY. The main content area is titled "Overview" and contains a "General" section with the following details:

Region	amazon-web-services::us-east-1
Cluster	moose.rmq.cloudamqp.com (DNS load balanced)
Hosts	moose-01.rmq.cloudamqp.com (Availability Zone use1-az6)
Created at	2023-06-09 23:10 UTC+00:00

Below the General section, there's a section titled "AMQP details" which is currently empty.

Se abrirá una nueva pagina, vaya a la solapa Queues

Queues

▼ All queues (1)

Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
unaCola2	classic	HA mxtwnswv-max-length	idle	0	0	0	0.00/s	0.00/s	0.00/s	

► Add a new queue

En la tabla verán una cola con el QUEUE_NAME que colocaron en las variables de entorno . Hagan click sobre el nombre de la misma en la 1er columna y busquen sobre la pantalla siguiente el título “Publish message”, desplieguelo, escriba algo donde dice “Payload” y apriete “Publish message”. Después de esto, deberá visualizar lo que escribió en la consola de CopiaMeWorker.

Effective policy definition	expires: 2419200000 ha-mode: all ha-sync-mode: automatic max-length: 10000 queue-mode: lazy	Process memory
-----------------------------	---	----------------

► Consumers (0)

► Bindings (1)

▼ Publish message

Message will be published to the default exchange with routing key **unaCola2**, routing it to this queue.

Delivery mode: 1 - Non-persistent

Headers: ? = String

Properties: ? =

Payload: algo aca

Payload encoding: String (default)

Publish message

► Get messages

Ahora haga las siguientes pruebas:

- 1) apague CopiaMeWorker, mande un par de mensajes por la página como recién y vuelva a prender a worker, ¿qué ocurrió?
- 2) Agregue una demora de 10 segundos justo después de leer el texto. Con eso envíe 3 mensajes seguidos y vuelva rápidamente a la consola. ¿Cómo atiende los pedidos? ¿Cómo me puede servir esto?

```
public void handleDelivery(String consumerTag, Envelope envelope,
    AMQP.BasicProperties properties, byte[] body) throws IOException {
    // Confirmar la recepción del mensaje a la mensajería
    this.getChannel().basicAck(envelope.getDeliveryTag(), false);
    String analisisId = new String(body, "UTF-8");
    System.out.println("Demora..");
    try {
        Thread.sleep(10000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("se recibió el siguiente payload:");
    System.out.println( analisisId);
}
```

Paso 3: Refactor de la APP

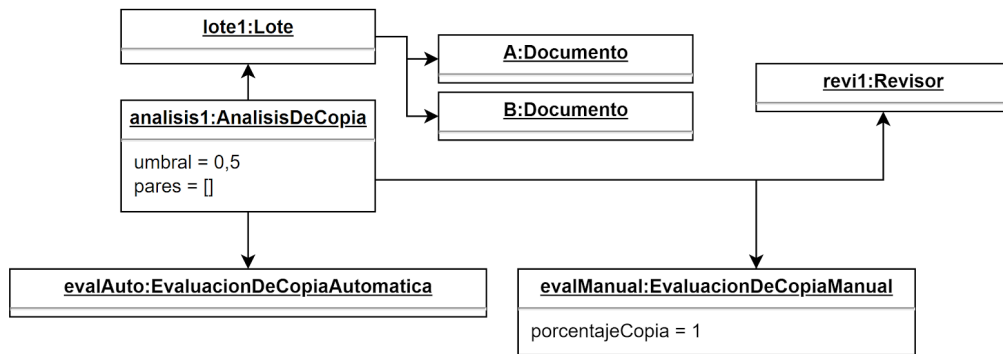
Para todo lo que siga, recomendamos usar una base persistente, no el H2 que configuramos para los tests. Puede usar la que tiene en render o mejor aún, levantar una local. Si usa una que no sea Postgres, recuerde incluir la dependencia en el pom y poner el driver adecuado en las variables de entorno.

Ahora ataquemos la suposición que disparó todo este cambio de arquitectura: “Calcular la distancia entre los documentos es un proceso largo”. Entonces, primero vamos a separar el proceso de crear los pares de el de evaluar las copias, que se encuentra en la clase AnalisisDeCopia

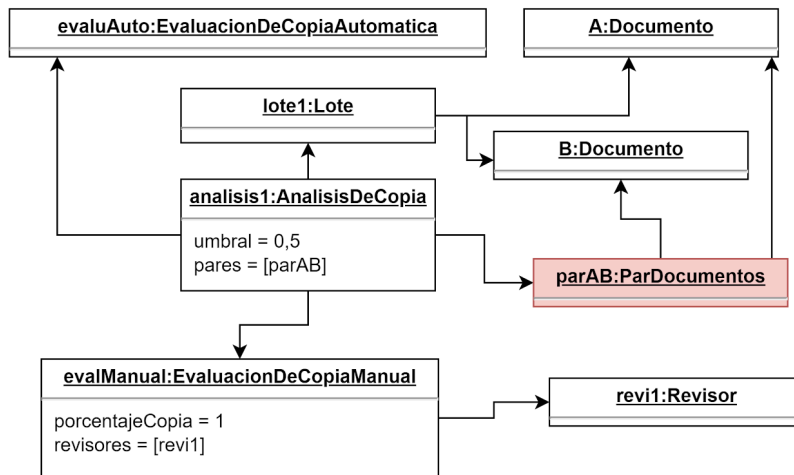
<pre>public void procesar() { // Genero todos los pares de documentos Posibles this.pares = Generator.combination(this.lote.getDocumentos() .simple(2) .stream() .map(t-> new ParDocumentos(t.get(0),t.get(1))) .collect(Collectors.toList()); // Armo el resultado procesando cada par this.rl = new ResultadoLote(); rl.setFechaInicio(LocalDateTime.now()); for (EvaluadorDeCopia evaluador : this.evaluadores) { evaluador.procesar(pares); } }</pre>	<pre>public void crearPares() { // Genero todos los pares de documentos Posibles this.pares = Generator.combination(this.lote.getDocumentos() .simple(2) .stream() .map(t-> new ParDocumentos(t.get(0),t.get(1))) .collect(Collectors.toList()); } //----- public void procesar() { // Armo el resultado procesando cada par this.rl = new ResultadoLote(); rl.setFechaInicio(LocalDateTime.now()); for (EvaluadorDeCopia evaluador : this.evaluadores) { evaluador.procesar(pares); } }</pre>
Antes	Después

Entonces, partiendo este método en 2, lo que vamos a hacer es ejecutar el primero en la API, cosa de que se guarde el análisis con todos los pares, y el worker justamente haga la parte de la evaluación.

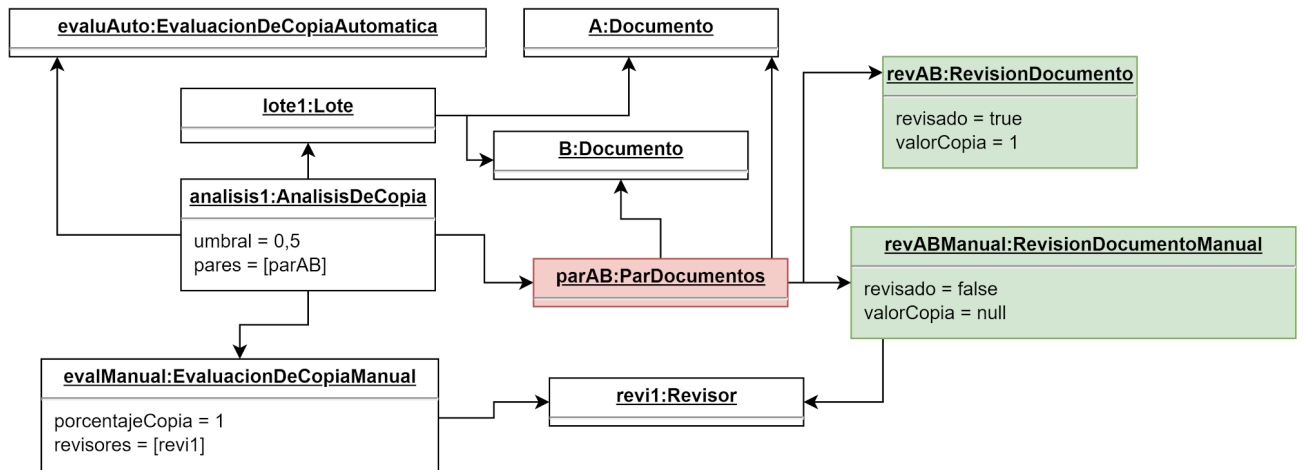
Escenario: Análisis de Plata de 2 documentos



Luego de analisis1.crearPares() --> ESTO SE HACE EN LA API



Luego de analisis1.procesar() --> ESTO SE HACE EN EL WORKER



Si recordamos, CopiaMeAPI tiene un repo en memoria para los análisis, que ya quedó obsoleto, vamos a cambiar de esto:

```
public class CopiameAPI {
    public static EntityManagerFactory entityManagerFactory;
    public static void main(String[] args) {
        Integer port = Integer.parseInt(System.getProperty("PORT", "8080"));
        Javalin app = Javalin.create().start(port);
        AnalisisRepository repo = new AnalisisInMemoryRepository();
        app.get("/analisis", new AnalisisListController(repo));
        app.post("/analisis", new AnalisisAddController(repo));
    }
}
```

A lo que está a continuación. Notar que los controller necesitan el EMFactory, ya que CADA vez que reciben un request, tienen que crear un entity manager nuevo. Por otro lado, para crear al entityManagerFactory, con las variables de entorno como vimos, lo delegamos en la clase PersistenceUtils, ya que no solo la API lo utilizara, también lo hará el worker.

```
public class CopiameAPI {
    public static EntityManagerFactory entityManagerFactory;

    public static void main(String[] args) {
        entityManagerFactory =
            PersistenceUtils.createEntityManagerFactory();
        Integer port = Integer.parseInt(
            System.getProperty("PORT", "8080"));
        Javalin app = Javalin.create().start(port);
        app.get("/analisis",
            new AnalisisListController(entityManagerFactory));
        app.post("/analisis",
            new AnalisisAddController(entityManagerFactory));
        app.get("/revisor/{id}",
            new RevisorGetController(entityManagerFactory));
        app.post("/revisor/",
            new RevisorPostController(entityManagerFactory));
    }
}
```

Y ahora hacemos refactor sobre los controller, AnalisisListController nos da un listado de los análisis que hay en la DB y AnalisisAddController, que es bastante más complicado, da de alta un análisis. Veamos que ya no lo procesa más, sino que “solo” genera los pares, persiste los datos y le retorna algo al usuario.

```
public class AnalisisListController implements Handler {
    private EntityManagerFactory entityManagerFactory;

    public AnalisisListController(
        EntityManagerFactory entityManagerFactory) {
        super();
        this.entityManagerFactory = entityManagerFactory;
    }

    @Override
    public void handle(Context ctx) throws Exception {
        AnalisisJPARepository repo = new AnalisisJPARepository(
            entityManagerFactory.createEntityManager());
        ctx.json(repo.all());
    }
}
```

```

public class AnalisisAddController implements Handler {

    private EntityManagerFactory entityManagerFactory;
    public AnalisisAddController(
        EntityManagerFactory entityManagerFactory) {
        super();
        this.entityManagerFactory = entityManagerFactory;
    }

    @Override
    public void handle(Context ctx) throws Exception {
        EntityManager entityManager =
            entityManagerFactory.createEntityManager();
        AnalisisJPAREpository repo = new AnalisisJPAREpository(
            entityManager);

        // Proceso de parámetros de entrada
        String destDirectory = "/tmp/unlugar";

        UnzipUtility.unzip(ctx.uploadedFile("file").content()
            , destDirectory);

        // Armado del Análisis
        Lote lote = new Lote(destDirectory);
        lote.validar();
        lote.cargar();
        float umbral = 0.5f;
        AnalisisDeCopia analisis = new AnalisisDeCopia(umbral, lote);
        analisis.addEvaluador(new EvaluadorDeCopiaAutomatico());

        // genero los pares de documentos del lote
        // YA NO LOS PROCESO
        analisis.generarPares();

        // Guardado
        entityManager.getTransaction().begin();
        repo.save(analisis);
        entityManager.getTransaction().commit();
        entityManager.close();
        // Armado de la respuesta
        Map<String, String> rta = new HashMap<String, String>();
        rta.put("analisis", analisis.getId());
        rta.put("terminado", analisis.finalizado().toString());
        ctx.json(rta);
    }
}

```

```

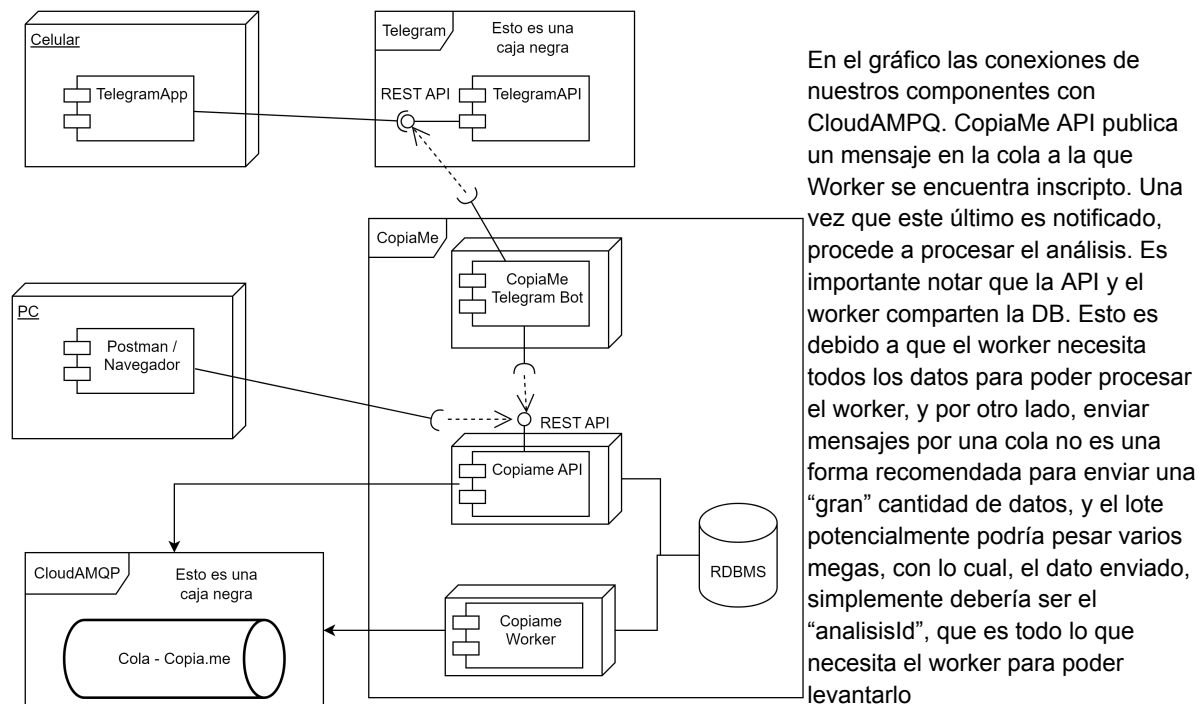
=====
Debe arreglar lo siguiente en la clase AnalisisJPAREpository
public void save(AnalisisDeCopia analisis) {
    String key = UUID.randomUUID().toString().substring(0, 5);
    analisis.setId(key);
    this.entityManager.persist(analisis);
}
=====

```

Con todo esto, genere un par de análisis usando la API o telegram como lo vimos en la 3ra iteración, paso 5. Verifique en la base de datos, que no se creó ningún objeto de ResultadoEvaluacion.

Paso 4: Hacer que el worker procese los análisis.

Para integrar al worker con la API, lo haremos de 2 formas distintas, primero usamos la Mensajería, para avisar cuando hay un trabajo, y luego, para que el worker procese el análisis y deje los resultados, utilizaremos la base de datos. Esto último es cuestionable, pero por razones de simplicidad y tiempos lo dejaremos así, al final de esta iteración lo discutiremos un poco. Entonces, primero que nada, veamos la arquitectura actualizada:



Ahora vamos a proceder a darle acceso a la db relacional al worker, a través de hibernate también, para que a partir del id del análisis que viaja en la mensajería, pueda instanciar el mismo. Entonces, el método que procesa los mensajes en el Worker quedaria asi:

```
public class CopiaMeWorker extends DefaultConsumer {
    private String queueName;
    private EntityManagerFactory entityManagerFactory;
    protected CopiaMeWorker(Channel channel, String queueName,
        EntityManagerFactory entityManagerFactory) {
        super(channel);
        this.queueName = queueName;
        this.entityManagerFactory = entityManagerFactory;
    }

    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {

        // Confirmar la recepción del mensaje a la mensajería
        this.getChannel().basicAck(envelope.getDeliveryTag(), false);

        // Leer el mensaje
        String analisisId = new String(body, "UTF-8");
    }
}
```

```

// Obtengo el analisis a procesar
EntityManager entityManager =
entityManagerFactory.createEntityManager();
entityManager.getTransaction().begin();

AnalisisJpaRepository repo = new AnalisisJpaRepository(
                                                                    entityManager);

// Obtengo y proceso el analisis
AnalisisDeCopia ads = repo.findById(analysisId);
ads.procesar();
// fijense que lo anterior, es la ÚNICA línea de código de negocio, //
sin embargo delega, genera los puntajes etc. Acá lo importante es //
que me estoy trayendo de la DB el análisis que dejó la API,
// levanto todos los objetos que necesito, ejecuté la colaboración y
// vuelvo a guardar el escenario.

entityManager.getTransaction().commit();
entityManager.close();
}

```

Para verificar que esto funciona correctamente, vea en la API cuales son los IDs de análisis pendientes y publique un mensaje con los mismos (de a uno a la vez), a ver como funciona el worker. Consulte en la API, si una vez que el worker procesa cada análisis, pasa a estado completado

Así deberían lucir las variables de entorno (es un ejemplo), tanto para el worker, como para la API:

```

hibernate.hbm2ddl.auto=update
javax.persistence.jdbc.driver=org.postgresql.Driver
javax.persistence.jdbc.password=123
javax.persistence.jdbc.url=jdbc:postgresql://localhost/copiamedb
javax.persistence.jdbc.user=postgres
QUEUE_HOST=moose-01.rmq.cloudamqp.com
QUEUE_NAME=unaCola
QUEUE_PASSWORD=dj1RD64wN6456435NbNIL75Jkt1BD0ui
QUEUE_USERNAME=mxtnsxx

```

Paso 5: API envía un mensaje cuando recibe un nuevo lote

Ahora vamos a agregar que en el POST / analisis, luego de guardar el análisis, envíe un mensaje a la mensajería en la nube. Primero vamos a crear una clase de útiles para acceder a la mensajería:

```
public class MQUtils {

    private String host;
    private String username;
    private String password;
    private String virtualHost;
    private String queueName;
    private Connection connection ;
    private Channel channel;

    public MQUtils(String host, String username, String password,
        String virtualHost, String queueName) {
        super();
        this.host = host;
        this.username = username;
        this.password = password;
        this.virtualHost = virtualHost;
        this.queueName = queueName;
    }

    public void init() throws IOException, TimeoutException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(this.host);
        factory.setUsername(this.username);
        factory.setPassword(this.password);
        factory.setVirtualHost(this.virtualHost);
        this.connection = factory.newConnection();
        this.channel = connection.createChannel();
    }

    public void publish(String message) throws IOException {
        channel.basicPublish("", this.queueName, null,
            message.getBytes());
    }
}
```

=====

Debe agregar lo siguiente en la clase RevisionDocumentoManual

```
public RevisionDocumentoManual() {
    super();
}

y EvaluadorDeCopiaManual:

public class EvaluadorDeCopiaManual extends EvaluadorDeCopia {

    @ManyToMany(cascade = CascadeType.ALL)
    private List<Revisor> revisores;
    private Double porcentajeRev;

    public EvaluadorDeCopiaManual() {
        super();
        this.revisores = new ArrayList<>();
    }
}
```

=====

Y modificamos CopiaMeAPI para que la inicialice:

```
public static void main(String[] args) throws Exception {

    entityManagerFactory = PersistenceUtils.createEntityManagerFactory();
    Map<String, String> env = System.getenv();
    MQUtils mqutils = new MQUtils(
        env.get("QUEUE_HOST"),
        env.get("QUEUE_USERNAME"),
        env.get("QUEUE_PASSWORD"),
        env.get("QUEUE_USERNAME"),
        env.get("QUEUE_NAME")
    );
    mqutils.init();

    Integer port = Integer.parseInt(System.getProperty("PORT", "8080"));
    Javalin app = Javalin.create().start(port);
    app.get("/analisis", new
        AnalisisListController(entityManagerFactory));
    app.post("/analisis", new
        AnalisisAddController(entityManagerFactory, mqutils));
    app.get("/revisor/{id}", new
        RevisorGetController(entityManagerFactory));
    app.post("/revisor/", new
        RevisorPostController(entityManagerFactory));
}
```

Finalmente, modificamos AnalisisAddController (encargado de procesar los lotes), para que publique el mensaje (notar que ahora necesita al MQUtils como parámetro):

```
public class AnalisisAddController implements Handler {

    private EntityManagerFactory entityManagerFactory;
    private MQUtils mqutils;

    public AnalisisAddController(
        EntityManagerFactory entityManagerFactory,
        MQUtils mqutils) {
        super();
        this.entityManagerFactory = entityManagerFactory;
        this.mqutils = mqutils;
    }

    @Override
    public void handle(Context ctx) throws Exception {
        EntityManager entityManager =
            entityManagerFactory.createEntityManager();
        AnalisisJPAREpository repo = new
            AnalisisJPAREpository(entityManager);
        // Proceso de parámetros de entrada
        String destDirectory = "/tmp/unlugar";
        System.out.print(ctx.uploadedFiles());
        UnzipUtility.unzip(ctx.uploadedFile("file").content(),
            destDirectory);
        // Armado del Análisis
        Lote lote = new Lote(destDirectory);
        lote.validar();
        lote.cargar();
    }
}
```

```

float umbral = 0.5f;
AnalisisDeCopia analisis = new AnalisisDeCopia(umbral, lote);
analisis.addEvaluador(new EvaluadorDeCopiaAutomatico());

// genero los pares de documentos del lote
analisis.generarPares();
// Guardado
entityManager.getTransaction().begin();
repo.save(analisis);
entityManager.getTransaction().commit();
entityManager.close();

// AQUI SE ENVIA EL MENSAJE A LA COLA
mqutils.publish(analisis.getId());
// -----

// Armado de la respuesta
Map<String,String> rta = new HashMap<String, String>();
rta.put("analisis", analisis.getId());
rta.put("terminado", analisis.finalizado().toString());
ctx.json(rta);
}

```

Desde la consola web de AMPQ cloud, la parte de “RabbitMQ Manager” que vimos en el paso 2, puede verificar que se están enviando a la cola que se creó. Una vez que confirme que está mandando mensajes, pruebe el sistema completo. Debe levantar los 2 procesos, la API y el Worker. Recomendamos que lo haga local, pero puede desplegar la API en Render si lo desea. No así el worker, ese si o si tiene que ser local.

Realice un POST /analisis (si no recuerda como, vea la iteración 3 Paso 5 de este doc) y verifique que (1) la API pública el mensaje a la mensajería en la nube (2) el worker recibe el mensaje (3) el worker procesa el análisis y crea los resultados correspondientes