2º Desarrollo de Aplicaciones Multiplataforma

Objetivo del proyecto

Nos ha contratado el Ayuntamiento de Madrid para que hagamos un estudio del análisis de  
limpieza y gestión de basuras que

Reciclaje y limpieza de madrid

Hecho por: Jeremy Ramos Y Daniel Carmona Rodríguez

# Índice

[Índice 1](#_Toc117016522)

[Diseño y propuesta de solución establecidas 3](#_Toc117016523)

[Diseño 3](#_Toc117016524)

[Propuesta de solución 3](#_Toc117016525)

[GitFlow. 4](#_Toc117016526)

[Estructura 4](#_Toc117016527)

[Disposición de Clases 5](#_Toc117016528)

[Controladores 5](#_Toc117016529)

[DTO 5](#_Toc117016530)

[Exceptions 5](#_Toc117016531)

[Mappers 6](#_Toc117016532)

[Models 6](#_Toc117016533)

[Utils 7](#_Toc117016534)

[Main 7](#_Toc117016535)

[Transformación de formatos de información 8](#_Toc117016536)

[Métodos y/o sentencias que nos permiten el parseo de los datos 8](#_Toc117016537)

[Realización de las consultas (Creación del método Resumen). 10](#_Toc117016538)

[Método Resumen 11](#_Toc117016539)

[Método Resumen por Distrito 12](#_Toc117016540)

[Gráficos 12](#_Toc117016541)

[Forma Programática Realizada 13](#_Toc117016542)

[Gráfico con el total de contenedores por distrito. 13](#_Toc117016543)

[Gráfico de media de toneladas mensuales de recogida 13](#_Toc117016544)

[Gráfico con el total de toneladas por residuo en ese distrito. 14](#_Toc117016545)

[Gráfica del máximo, mínimo y media por meses en dicho distrito. 15](#_Toc117016546)

[Aplicación de otras técnicas que consideres interesantes. 16](#_Toc117016547)

[Generación de JSON Y XML 16](#_Toc117016548)

[Promesas (FUTURE) 17](#_Toc117016549)

[Logger 18](#_Toc117016550)

[Justificación tecnológica 19](#_Toc117016551)

[Lenguaje 19](#_Toc117016552)

[Realización de Consultas 19](#_Toc117016553)

[Transformación de los datos 19](#_Toc117016554)

[Partes Opcionales No Descritas 20](#_Toc117016555)

[Bitácora 20](#_Toc117016556)

[Referencias 21](#_Toc117016557)

# Diseño y propuesta de solución establecidas

## Diseño

Captura de pantalla de computadora

Descripción generada automáticamente

Nos hemos centrado en una estructura de código mediante contenedores donde almacenamos las clases de nuestro programa siguiendo una estructura esquemática para facilitar la refactorización y la monitorización de nuestro programa. Tal y como vemos en la siguiente imagen.

## Propuesta de solución

Imagen que contiene taza, tabla, café, mujer

Descripción generada automáticamente  
Nos hemos planteado una solución afable y correlacionada a nuestros conocimientos y utilizando tecnologías varias las cuales veremos posteriormente en el programa. Como base al trabajar con csv´s lo que haremos será trabajar con transformadores del contenido CSV a una opción legible para nuestro programa. Basándonos en esta base construiremos la arquitectura propuesta por la comunidad, implementando métodos para parsear los datos y trabajar con varios formatos tales como XML y Json. También como veremos posteriormente implementaremos el trabajar con los datos de forma organizada con mecanismos de consultas tales como DataFrames. Y la generación de varios mecanismos de visualización tales como HTML.

# GitFlow.

## Estructura

Icono

Descripción generada automáticamenteUtilizamos el sistema GIT con una arquitectura de GitFlow la cual nos ayuda a organizar nuestro código gracias a trabajar con una estructura ramificada dependiendo del administrador que esté trabajando sobre el programa. Y manejando las ramas dispuestas de forma que nada se filtra sobre la rama principal, sino que cada administrador va trabajando sobre su rama.

A continuación, observaremos nuestro flujo de GIT el cual está separado en las ramas especificadas arriba. Con los nombres de las ramas y la disposición de estas junto con su autor.

Texto

Descripción generada automáticamente

# Disposición de Clases

## Controladores

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteSe encargan de manejar el funcionamiento del programa de forma ordenada y encapsulada en varios mecanismos distribuidos sobre las mismas.

***ContenedorControler/ResiduosController***

Clases las cuales trabajan con el trato de los dos csv’s introducidos tales como pasarlos a los distintos formatos CSV, XML y Json. Trabajando como veremos posteriormente con el trato de DTO´s

## Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto Descripción generada automáticamenteDTO

Clase la cual nos ayuda a conseguir transformar de los archivos csv para trabajar en el propio programa con los mismos.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## Exceptions

Pequeña clase la cual hemos abstraído un modelo o tipo de excepción.



## Mappers

Texto

Descripción generada automáticamenteClase con la cual tratamos los datos para transformarlos en DTO y viceversa de forma organizada.

Imagen de la pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

## Models

Clases donde manejamos la entrada y parametrización de los datos tales como los campos del csv, y dentro del csv encontramos datos relevantes de forma monótona y para esto hemos hecho clases enums las cuales están repartidas dentro de las clases correspondientes.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

## Utils

Paquete que contiene clases de utilidad las cuales nos permiten hacer distintas cosas tales como: Generar HTML, ciertos parseadores y formateadores de formatos tales como LocalDate y Formateo de Strings etc.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Main

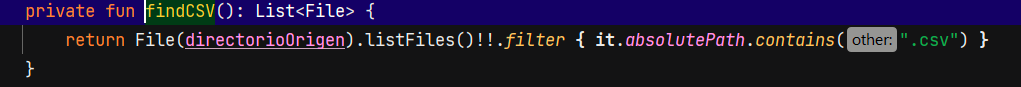
Clase principal donde nuestro programa se construye de forma progresiva y gestiona los mecanismos de entrada a nuestro programa los cuales son: parser, resumen y resumen por x Distrito.



Ahora procederemos a la trata y las transformaciones de los propios datos para operar con ellos y que sean legibles tanto para la entrada de estos como la salida de los mismos

# Transformación de formatos de información

## Métodos y/o sentencias que nos permiten el parseo de los datos

Ahora procederemos a analizar el trato de los datos a la entrada del programa. Comenzaremos con la búsqueda de ficheros csv dentro de nuestro directorio especificado para que en caso de que no esté o no encuentre ningún csv clave 

Una vez comprobado de que en nuestro fichero se encuentre nuestros csv lo que se le ha ocurrido al Administrador Daniel Carmona ha sido el identificar el tipo de csv que será comprobando el filtrado de la entrada de las cabeceras introducidas mediante el csv con una clase identifier la cual según su cabecera nos da el csv correspondiente.

Captura de pantalla de un celular

Descripción generada automáticamente

Comprobamos en el parser sí la cabecera es de residuos hacemos el load en el programa a base de los dto´s de este. Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla con la imagen de una pantalla

Descripción generada automáticamente con confianza mediaUna vez hecho esto comienza el proceso de parseo el cual ha sido implementado con un Excecutor que los definiremos más tarde dentro de la misma documentación. Y procedemos a parsear el csv a los distintos tipos de formatos establecidos JSON,XML y el propio CSV limpio de errores de entrada. Indicándole obviamente el directorio de destino el cual se va a guardar dicho archivo

***JSON***

Texto

Descripción generada automáticamente

***XML***

Texto

Descripción generada automáticamente

***CSV***

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

# Realización de las consultas (Creación del método Resumen).

Para esto se creó una función especial para resumen donde dentro contendrá todas las consultas propuestas por la comunidad de Madrid. Como primer paso lo que hemos planteado es que primero nos pase el propio csv para el trámite de las consultas.

Captura de pantalla de computadora

Descripción generada automáticamente con confianza media

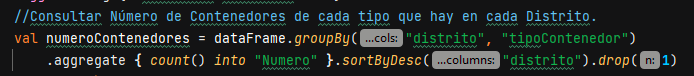
Imagen que contiene Diagrama

Descripción generada automáticamenteUna vez hecho el DataFrames aplicamos las consultas utilizando [DataFrames](https://github.com/Kotlin/dataframe). Básicamente moldeamos el documento a una hoja de datos sobre la cual podemos realizar consultas de forma progresiva sobre la hoja llamada Dataftame. Por esto tenemos el residuos.toDataFrame() -> Método el cual nos pasa la lista a dataframe para operar con las consultas.

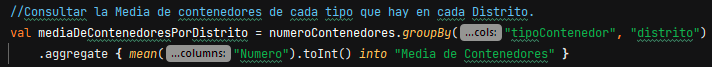
## Método Resumen

Aquí veremos la resolución de las consultas mandadas por la comunidad de Madrid.

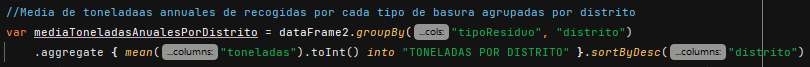
***#1***



***#2***



***#3***



***#4***

Captura de pantalla de un celular

Descripción generada automáticamente

***#5***

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

***#6***

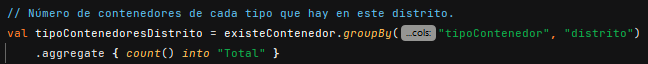


## Método Resumen por Distrito

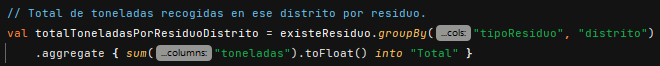
Lo primero que hacemos en nuestro programa es filtrar según el tipo de distrito que le pasamos en el paso de parámetros del inicio de nuestro programa.



***#1***



***#2***

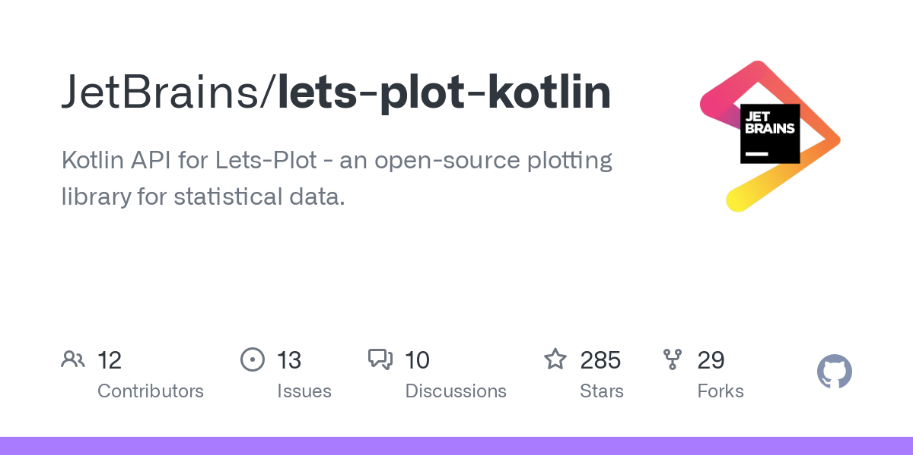


***#3***

Texto, Chat o mensaje de texto

Descripción generada automáticamente

# Gráficos

Todo esto lo realzaremos con la librería [lest-plot](https://lets-plot.org) la cual nos permite representar los datos de un DataFrame los cuales se realizan mediante estructuras preestablecidas y barras ya prediseñadas como geomBar, labs etc. También veremos como se genera el HTML correspondiente el cual dispondrá del gráfico creado a partir de las consultas hechas.

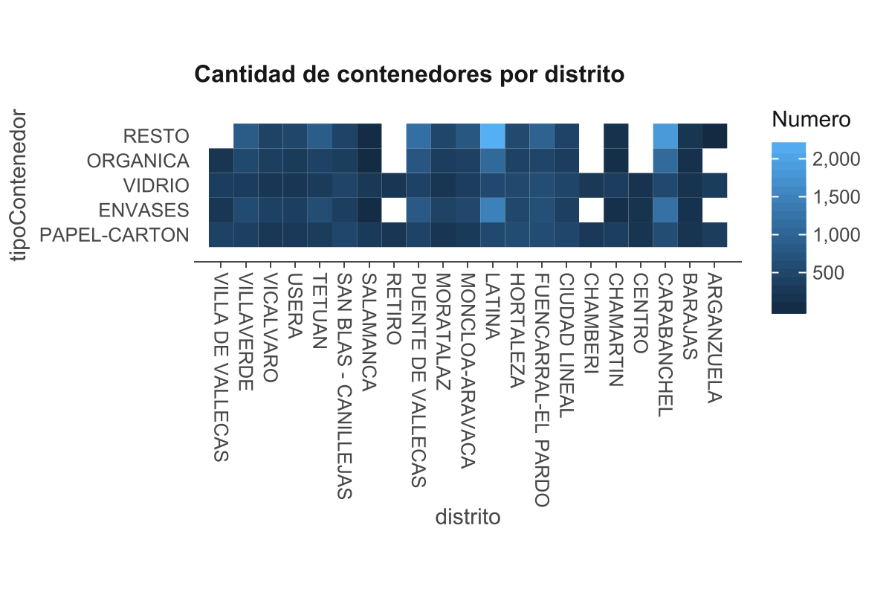
## Forma Programática Realizada

### Gráfico con el total de contenedores por distrito.

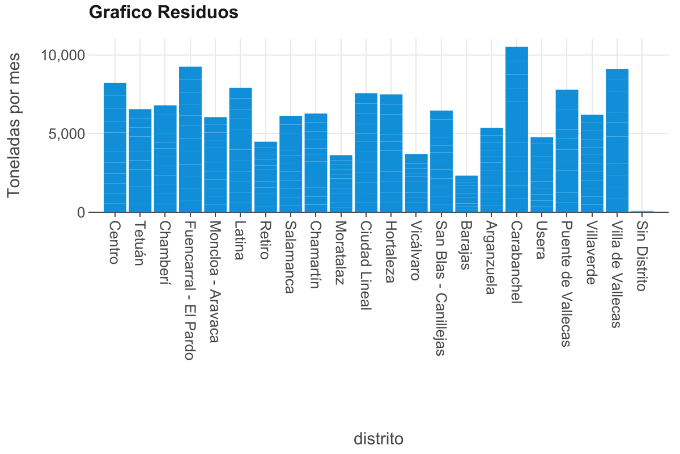
Para esto utilizaremos la variante geomTile.

Texto

Descripción generada automáticamente



### Gráfico de media de toneladas mensuales de recogida

Texto, Chat o mensaje de texto

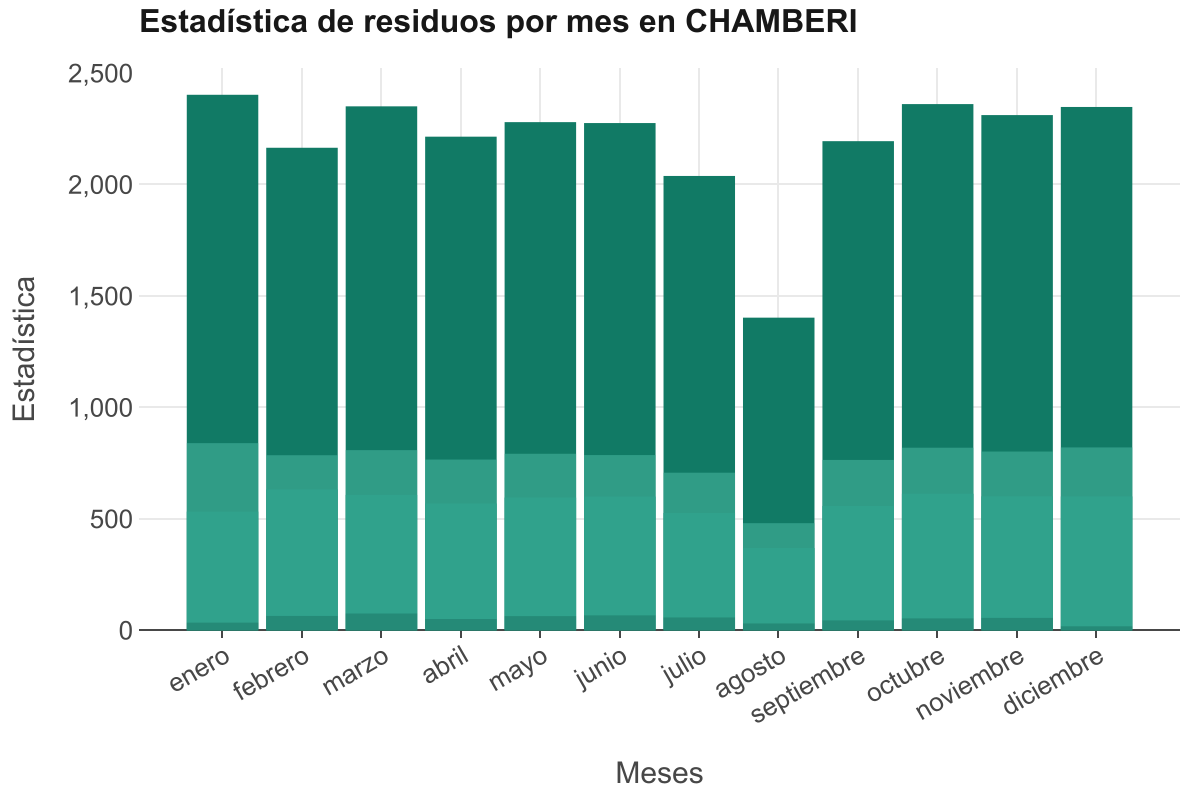
Descripción generada automáticamente

### Captura de pantalla de un celular Descripción generada automáticamenteGráfico con el total de toneladas por residuo en ese distrito.

### Gráfica del máximo, mínimo y media por meses en dicho distrito.

Texto

Descripción generada automáticamente



# Aplicación de otras técnicas que consideres interesantes.

## Texto, Aplicación, Chat o mensaje de texto Descripción generada automáticamenteGeneración de JSON Y XML

Sobre este apartado hemos tenido varias opciones pero las que nos pareció más practica según nuestro lenguaje base y las tecnologías que nos ofrece sobre otras hemos utilizado [Kotlin Serialization.](https://kotlinlang.org/docs/serialization.html) Gracias a esta librería nos permite de forma más practica indicarle al propio serialization que clases vamos a utilizar para realizar el Json gracias a la etiqueta @Serializable. Y las etiquetas @XmlElement(true) gracias a esto nos permitirá escribir también el XML.

Lo que conlleva a que el método de creación del JSON y XML quede de forma compacta y sin restricciones de por medio como veremos en las siguientes imágenes posteriores:

Captura de pantalla de computadora

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Promesas (FUTURE)

Diagrama

Descripción generada automáticamenteLas promesas es un mecanismo de manejo de hilos de forma paralela lo cual nos permite realizar tareas de procesamiento y/o entrada y salida de forma simultanea gracias al manejo de las promesas con la ayuda de un manejador el cual se denomina Excecutor, que se encargará de gestionar las mismas de forma que de forma paralela se realizaran acciones obteniendo un resultado con el método .get sobre la misma promesa y una vez realizada dicha tarea hemos de indicarle a nuestro Excecutor que finalice el proceso ya que se quedará en stanBy sí no se lo indicamos con la función .shutdown().  
Esto nos va a permitir optimizar el tiempo de ejecución de nuestro programa de forma considerable.

***Estructura y composición de la entrada de promesas dentro del proyecto.***

Texto

Descripción generada automáticamente

## Logger

Logger es un mecanismo de muestreo de resultados por consola de forma sistemática el cual nos brinda de varias opciones de muestreo de los datos dependiendo de el uso que le queremos dar a lo que enseñemos por pantalla. Tales como:

* logger.trace { "This is trace log" }
* logger.debug { "This is debug log" }
* logger.info { "This is info log" }
* logger.warn { "This is warn log" }
* logger.error { "This is error log" }

Para más información e implementación y configuración de la misma podéis visitar el link que nos redirige a la página Baeldung de Kotlin la cual nos especifica paso por paso lo que hemos de hacer. 🡪 [CLIC AQUI](https://www.baeldung.com/kotlin/kotlin-logging-library)

Ejemplos de utilización de Logger.

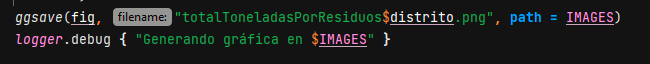
**Creación de la clase.**

Logotipo

Descripción generada automáticamente

**Implementación del Logger.debug**





**Vista preliminar del Logger.**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

# Justificación tecnológica

## Logotipo, nombre de la empresa Descripción generada automáticamenteLenguaje

Hemos debatido sobre que lenguaje hemos de utilizar o de hacer nuestro proyecto con una arquitectura en multilenguaje la cual nos tomaría tiempo, al final lo que hemos decidido es utilizar Kotlin ya que el mismo frente a otros como JAVA está más optimizado en la realización de clases, métodos, tecnologías, ciertas librerías de procesamiento de datos tales como la descrita anteriormente la cual era Kotlin Serialization que frente a JAVA no se encuentra. Y es un lenguaje el cual cuenta con características superiores a java como la opción de no admitir tipos sin formato o la seguridad a la hora de trabajar con la Nullability que es el manejo de nulos de java. Por esto nos pareció una buena opción. :D

## Realización de Consultas

Para este apartado hemos interiorizado en las librerías pertinentes dichas y expuestas en clase y llegamos a la conclusión de utilizar DataFrames antes que ApiStream ya que, aunque el uso de DataFrames es utilizado en Python para la resolución de los llamados “*pandas”* podemos implementarlo en Kotlin gracias a su librería.

Nos resulto una alternativa superior ya que el manejo de los datos y la representación y el manejo de estos realizando las consultas no resultó más cómodo.

## Transformación de los datos

Icono

Descripción generada automáticamente con confianza bajaAquí tuvimos una idea clara: *Tratar de utilizar una librería la cual nos proporcione soltura y nos permita realizar todo lo expuesto para la transformación de los datos a distintos formatos de ficheros de intercambio*. Llegamos a la conclusión como anteriormente hemos descrito de utilizar Kotlin Serialization ya que como se compagina con el lenguaje utilizado en nuestro proyecto decidimos que sería una alternativa mejor frente a Librerías tales como DOM, JAXB,SAX etc.

# Partes Opcionales No Descritas

## Bitácora

Implementamos el sistema de bitácoras dentro de nuestro programa para indicarle a los usuarios que lo utilicen un registro de como o que se ha hecho dentro de la aplicación misma la cual nos indicará sí se realizó, fecha de creación, sí tuvo éxito etc.

Aquí varios ejemplos.

***Mecanismo programático realizado sobre la bitácora.***

Texto

Descripción generada automáticamente

***Función que le indica al programa donde hemos de guardar la bitácora.***

Captura de pantalla de computadora

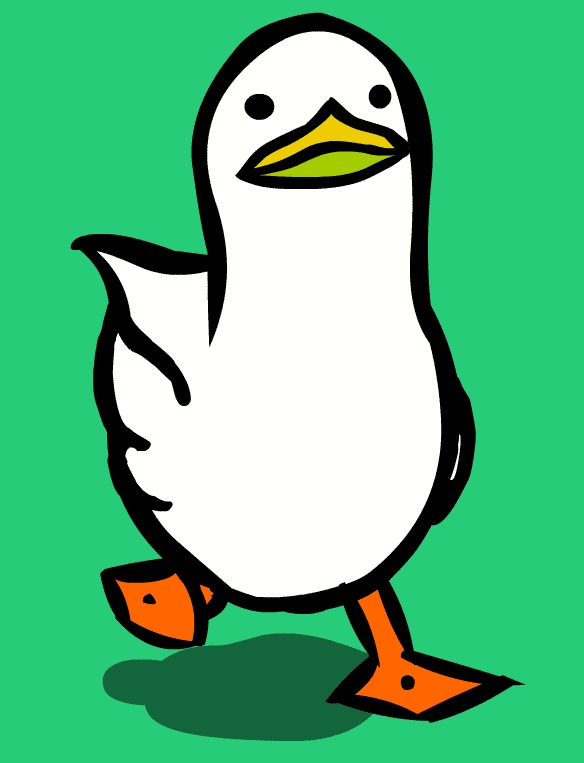
Descripción generada automáticamente

# Referencias

[*https://kotlinlang.org/docs/home.html*](https://kotlinlang.org/docs/home.html)

[*https://www.baeldung.com/kotlin/*](https://www.baeldung.com/kotlin/)

[*https://github.com/joseluisgs?tab=repositories*](https://github.com/joseluisgs?tab=repositories)



***Ces´t la vie***