# Capítulo 2: Análisis y diseño del sistema para las recomendaciones en las plataformas del proyecto z17.

**Introducción**

En este capítulo, a partir del estudio de los procesos del negocio, sus descripciones y su modelado, se describe el sistema a desarrollar. Se obtienen los artefactos relacionados a la ingeniería de software aplicada a la propuesta de solución tomando como punto de partida el problema de investigación. Además, se plasman los requisitos funcionales y no funcionales de la propuesta, así como los diferentes artefactos relacionados con la metodología de desarrollo

## 2.1 Descripción de la Propuesta de Solución.

A continuación se hará una detallada explicación de cómo está compuesto el sistema propuesto. Para ello el autor se basó en las etapas que lo forman y los modelos que lo integran.

### 2.1.1 Etapas.

El sistema está compuesto por **3 etapas** fundamentales, **Recuperación, Clasificación y Re-clasificación**. Estas etapas están conectadas entre sí de manera secuencias respectivamente donde la salida de una es la entrada de la otra siendo la última etapa la encargada de devolver los datos a recomendar y la primera la encargada de obtener el corpus de elementos candidatos inicial. A continuación cada una de las etapas:

* **Recuperación**: Esta etapa es responsable de seleccionar un conjunto inicial de cientos o miles de candidatos entre todos los posibles candidatos que pueden ser millones. El objetivo principal de esta etapa es eliminar de manera eficiente a todos los candidatos que no les interesan al usuario. Dado que se puede estar tratando con millones de candidatos, tiene que ser computacionalmente eficiente. Esta etapa puede estar compuesta por uno o varios modelos generadores de candidatos donde cada uno se basa en los datos implícitos (véase epígrafe 1.1.1.2) almacenados de los usuarios.
* **Clasificación**: Toma los resultados del modelo de recuperación y los ajusta para seleccionar el mejor puñado de recomendaciones posibles basándose en los datos explícitos (véase el epígrafe 1.1.1.2) que tenga disponible. Su tarea es reducir el conjunto de elementos que pueden interesar al usuario a una lista corta de posibles candidatos que ronde entre los cientos. La idea de esta etapa es obtener candidatos más preciso que en la etapa anterior ya que al basarse en los datos explícitos tiene información de mayor valor sobre que le interesa o no al usuario.
* **Re-clasificación:** En la etapa final el sistema puede volver a clasificar para considerar criterios o limitaciones adicionales como pueden ser la actualidad, la diversidad y la equidad. Estos son algunos de los diversos factores que pueden ayudar a mejorar considerablemente las recomendaciones de un sistema.

**La actualidad** vela por que los elementos sean recientes y no generar candidatos antiguos.

**La diversidad** valida que los candidatos sean diversos según las preferencias del usuario, todas las recomendaciones no pueden ser igual a lo que el usuario ve, ya que esto elimina la visibilidad del contenido, algo fundamental en este tipo de sistemas.

**La equidad** evita sesgos en las recomendaciones como podrían ser sebos de clics o falsas tendencias y permite que todos los usuarios sean tratados de manera justa.

Cada una de estas etapas está compuesta por 1 o varios modelos de redes neuronales profundas dependiendo de qué tan complejos sean los datos a utilizar (Covington et al., 2016; *Descripción general de los sistemas de recomendación | Machine Learning*, s. f.; *Reclasificación | Machine Learning*, s. f.; *Recomendar películas*, s. f.).

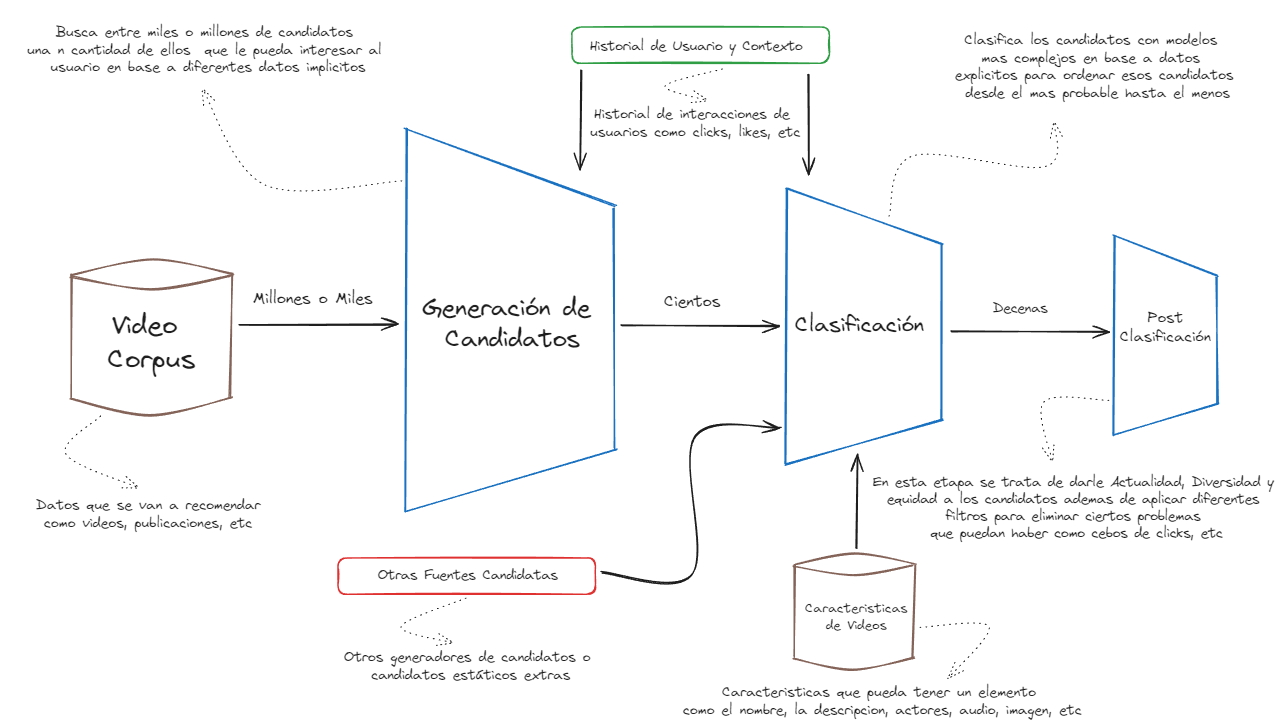


Figura 8: Arquitectura de capas del sistema.

### 2.2.1 Modelos.

Existen tres tipos de modelos utilizados en este sistema, Modelos de Dos Torres, Modelo de Recuperación y Modelo de Clasificación:

**Modelo de Dos Torres**: es una arquitectura de red neuronal diseñada específicamente para tareas de aprendizaje de representaciones y, en particular, para problemas de recomendación, búsqueda y clasificación. Su nombre proviene de la estructura visual que adopta: dos "torres" de redes neuronales que procesan independientemente dos tipos de datos, y luego combinan sus representaciones finales para obtener una puntuación o predicción.

En una arquitectura de dos torres, cada torre es una red neuronal que procesa características de entrada candidatas o de consulta para producir una representación integrada de esas características. Debido a que las representaciones de incrustación son simplemente vectores de la misma longitud, podemos calcular el producto escalar entre estos dos vectores para determinar qué tan cerca están. Esto significa que la orientación del espacio de incrustación está determinada por el producto escalar de cada par <query, candidato> en los ejemplos de entrenamiento (*Tensorflow Deep Retrieval Using Two Towers Architecture*, s. f.).

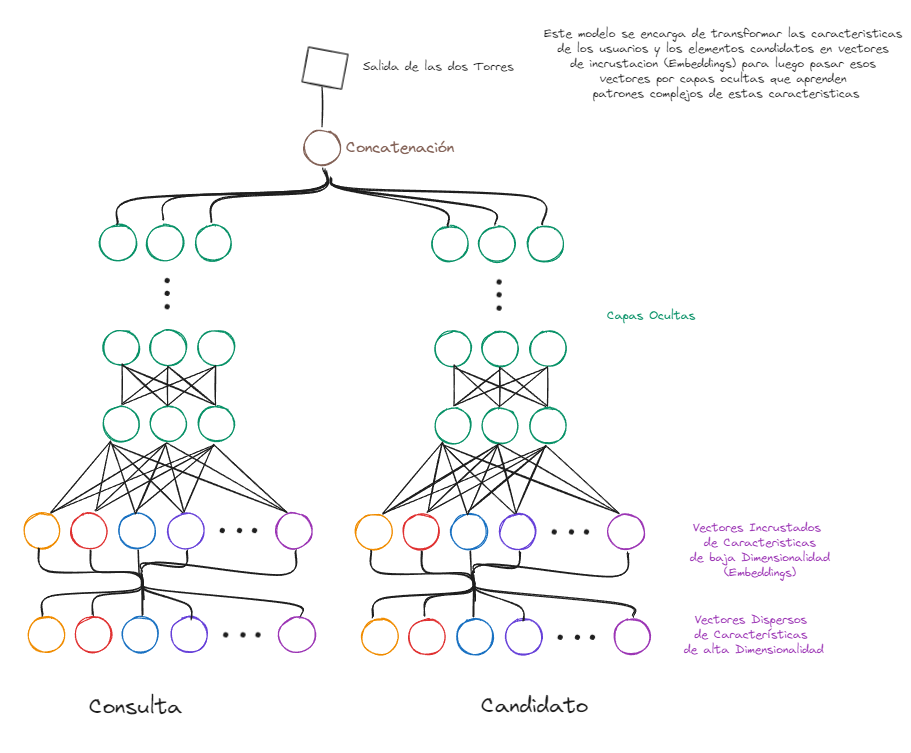


Figura 9: Arquitectura de una red neuronal de Dos Torres.

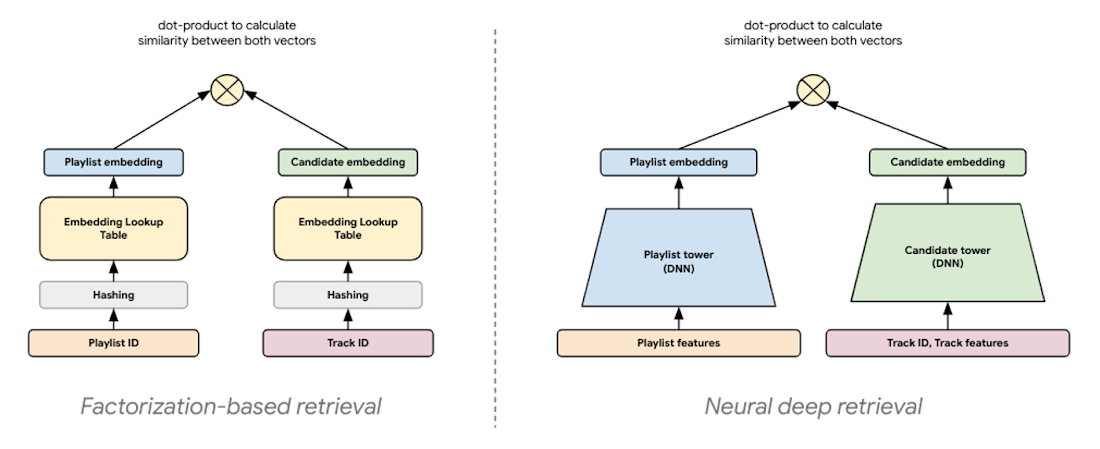


Figura 10: Las arquitecturas NDR (neural deep retrieval), como los codificadores de dos torres, son conceptualmente similares a los modelos de factorización. Ambas son técnicas de recuperación basadas en incrustaciones que calculan representaciones vectoriales de consultas y candidatos de dimensiones inferiores, donde la similitud entre estos dos vectores se determina calculando su producto escalar (*Tensorflow Deep Retrieval Using Two Towers Architecture*, s. f.).

**Modelo de Recuperación**: Es el encargado de leer el corpus de datos completo y recuperar unos miles de candidatos más probables a ser de interés para el usuario. Este tipo de modelos es entrenado con datos implícitos (véase el epígrafe 1.1.1.2) del usuario.

* **Representación**: Los modelos de recuperación a menudo se componen de dos sub modelos:
* Un modelo de consulta que calcula la representación de la consulta (normalmente un vector de incorporación de dimensionalidad fija) mediante funciones de consulta.
* Un modelo candidato que calcula la representación candidata (un vector de igual tamaño) utilizando las características candidatas.

Estos dos sub modelos son representados con el modelo de Dos Torres explicado anteriormente.

* **Función de activación de la capa de salida**: Se utiliza una capa de salida con función de activación Softmax para calcular la probabilidad de interés de un usuario hacia cada candidato. A continuación la función softmax (*Modelos de red neuronal profunda | Machine Learning | Google for Developers*, s. f.).

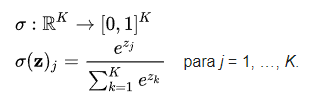


Figura #: Función de activación softmax.

* **Métrica y función de coste**: Para medir la precisión se utiliza una métrica llamada **FactorizedTopK** específica diseñada para evaluar el desempeño de modelos de recomendación factorizados. Esta métrica se enfoca en medir la capacidad del modelo para predecir correctamente los elementos (por ejemplo, productos, películas) más relevantes para un usuario dado en una lista ordenada de recomendaciones.

Recibe la lista de candidatos donde para cada consulta (usuario) se obtiene la lista de candidatos predichos por el modelo y se compara esta lista con la lista real de candidatos seleccionados por el usuario. Luego se calcula la precisión para diferentes valores de k (generalmente 1, 5, 10, 50, 100). La precisión se define como el porcentaje de candidatos predichos que coinciden con los candidatos reales (*Recomendar películas*, s. f.).

* **Servicio**: Este modelo una vez entrenado es explotado para construir un servicio eficiente mediante la construcción de un índice aproximado de vecinos más cercanos (ANN) el cual se utiliza en producción para obtener los n candidatos en la etapa de recuperación.

El sistema de recomendación puede tener varios generadores de candidatos que usen diferentes fuentes, como de la siguiente manera:

* Elementos relacionados de un modelo de factorización de matrices.
* Funciones del usuario que tienen en cuenta la personalización.
* “Local” y "distante" ítems; es decir, que toman información geográfica a tener en cuenta.
* Artículos populares o en tendencia.
* Un gráfico social; es decir, los elementos recomendados por amigos.

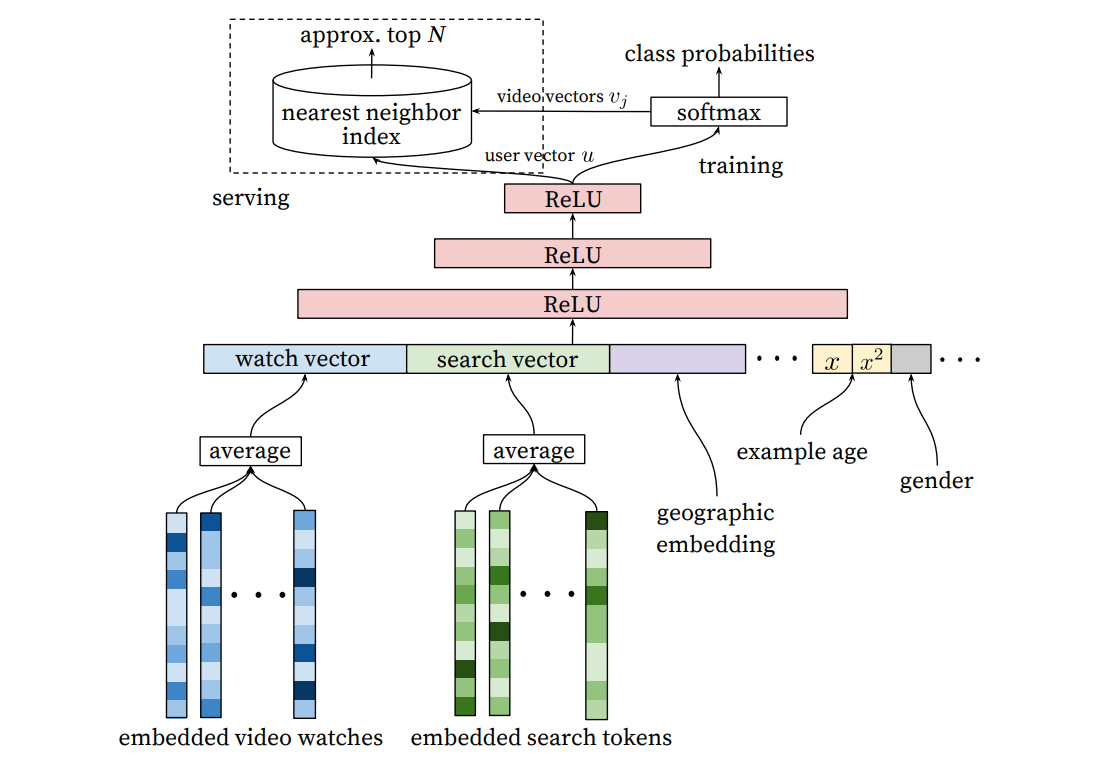


Figura 11: Arquitectura de un Modelo de Recuperación (Covington et al., 2016).

**Modelo de Clasificación:** Después de la generación del candidato, otro modelo califica y clasifica los candidatos para seleccionar el conjunto de elementos que se mostrarán.

El sistema combina estas diferentes fuentes en un grupo común de candidatos que se califican con un solo modelo y se clasifican según esa puntuación. Por ejemplo, el sistema puede entrenar un modelo para que prediga probabilidad de que un usuario mire un video en YouTube a partir de los datos explícitos del mismo como podrían ser los likes y teniendo en cuenta los siguientes factores:

Funciones de búsqueda (por ejemplo, historial de reproducciones del usuario, idioma, país y hora).

Elementos de video (por ejemplo, título, etiquetas, incorporación de video, duración).

* **Representación**: Este modelo utiliza la misma representación que el de Recuperación.
* **Función de activación de la capa de salida:** Este modelo puede variar en cuanto a la capa de salida ya que se podrían utilizar diferentes enfoques para clasificar a los candidatos dependiendo del set de datos que se pretenda utilizar.
* **Métrica y función de coste:** puede variar ya que se podrían utilizar diferentes enfoques para clasificar a los candidatos dependiendo del set de datos que se pretenda utilizar.
* **Servicio:** Se utiliza el propio modelo para hacer predicciones sobre los candidatos.



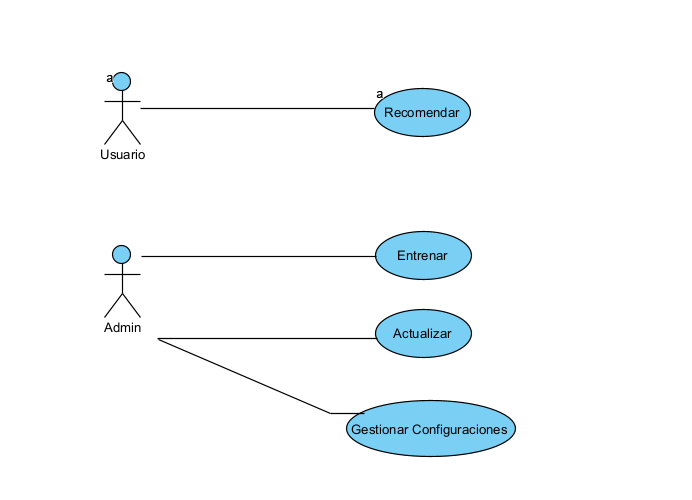
Figura 12: Arquitectura de un Modelo de Clasificación (Covington et al., 2016).

## Modelo Conceptual

Un modelo conceptual es un artefacto de la disciplina de análisis, construido con las reglas UML. Tiene como objetivo comprender y describir las clases más importantes, así como, identificar y explicar los conceptos significativos en el dominio del problema, identificando los atributos y las asociaciones existentes entre ellos (González Matos, 2021) (Sommerville, 2011).

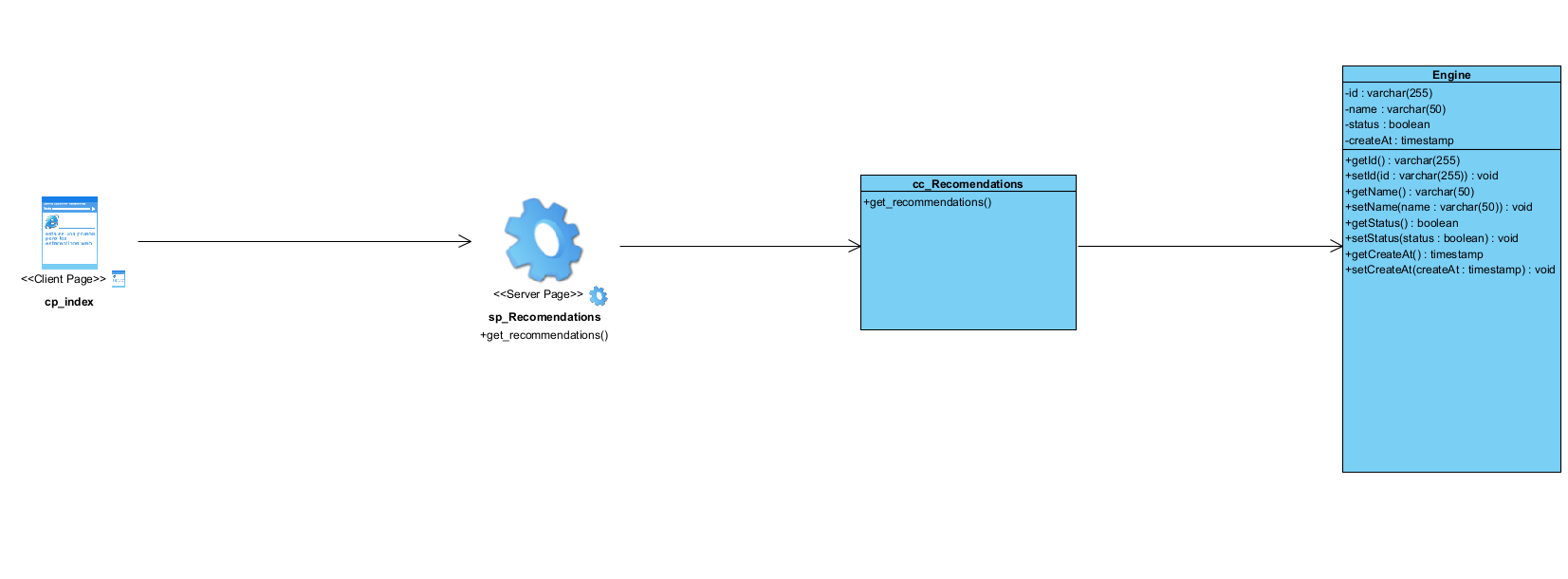
## Diagrama de caso de uso del sistema

Los diagramas de caso de uso son una técnica para capturar requisitos o información de cómo un sistema o negocio trabaja, y están compuesto por los casos de uso, los actores que se pueden definir como algo con comportamiento, como una persona (identificada por un rol), sistema informatizado u organización (Larman, 2003), y las relaciones existentes entre ambos (González Matos, 2021).



## Diagramas de clases del diseño

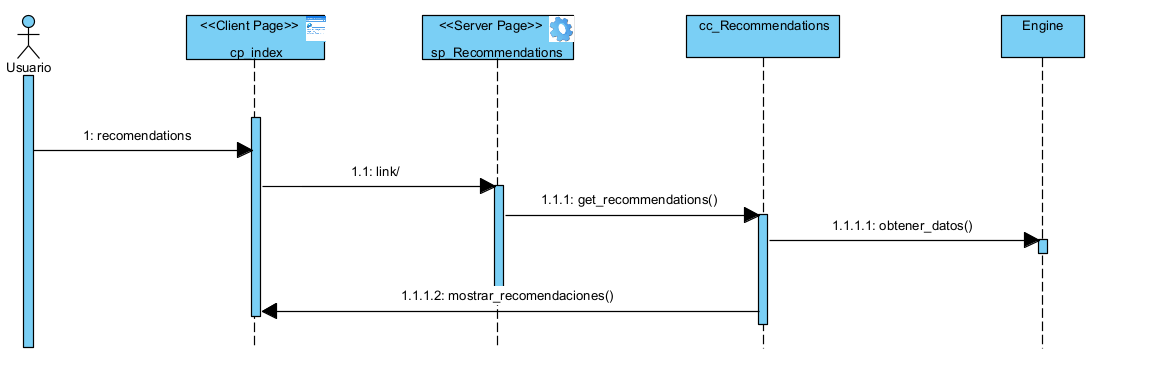
Un diagrama de clases de diseño (DCD) representa las especificaciones de las clases e interfaces software en una aplicación. A diferencia de las clases conceptuales del Modelo del Dominio, las clases de diseño de los DCD muestran las definiciones de las clases software en lugar de los conceptos del mundo real (González Matos, 2021) (Sommerville, 2011).



El DCD está compuesto por 1 server\_page (página controladora/servidora) que se corresponde con la clase principal de obtener recomendaciones, 1 client\_page (página cliente/vista) que muestran la página index correspondiente a la página principal del sistema, la clase controladora que contiene todas las funcionalidades de la server\_page (página controladora/servidora) , la clase que permite la conexión con la base de datos y la clase de la base de datos con los campos y funciones asociados al contenido Engine.

## Diagrama de secuencia

Los diagramas de secuencia (DS) en el UML se usan principalmente para modelar las interacciones entre los actores y los objetos en un sistema, así como las interacciones entre los objetos en sí (González Matos, 2021) (Sommerville, 2011).



El flujo representado en el DS comienza cuando el actor (usuario) del sistema entra a la página principal. La página cliente hace la solicitud a la server\_page (página controladora/servidora). La clase controladora obtiene los datos de la clase Engine la cual hace la petición a la base de datos y por ultimo estos datos son devueltos a la página cliente index donde son mostrados.

## 2.3 Historias de usuario.

Tabla 2 Historial de usuario del caso de uso Recomendar

|  |  |  |
| --- | --- | --- |
|  | | |
| **Número**: 1 | **Nombre del requisito**: Recomendar | |
| **Programador**: Alejandro Figueroa Rodríguez | | **Iteración Asignada**: Primera Iteración |
| **Prioridad**: Alta | | **Tiempo Estimado**: 0.10 días |
| **Riesgo en Desarrollo**: Gespro/Plan de riesgo | | **Tiempo Real**: 0.7 días |
| **Descripción**: Las platafomas integradas Picta, Todus y Apklis acceden a la API del sistema de recomendación con la dirección dirección-servicio/recommend la cual genera las recomendaciones y las devuelve a la plataforma. | | |
| **Observaciones**:   1. Las plataformas deben de estar registradas en el sistema. | | |
| **Prototipo de interfaz:** | | |

Tabla 3 Historial de usuario del caso de uso Entrenar

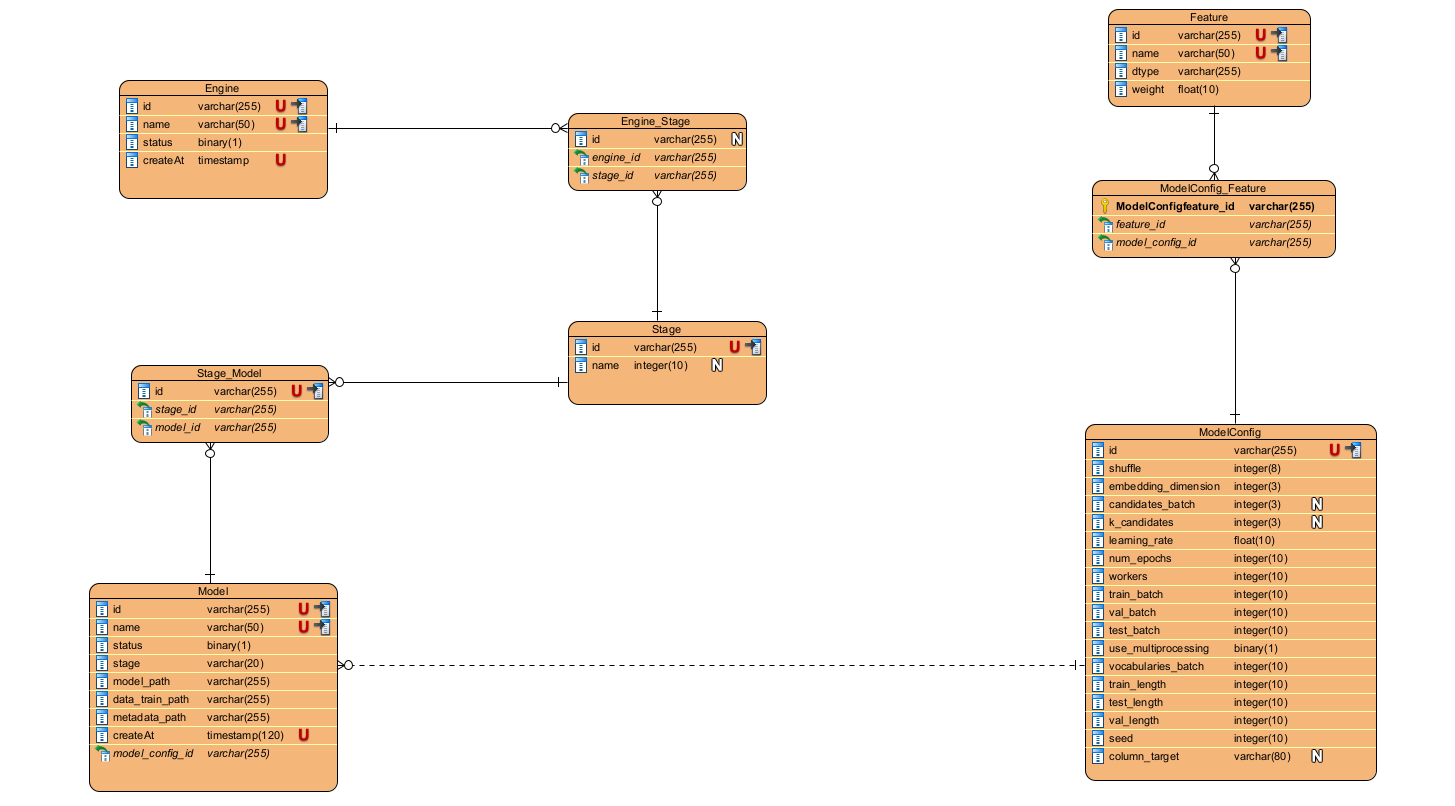
|  |  |  |
| --- | --- | --- |
|  | | |
| **Número**: 1 | **Nombre del requisito**: Entrenar | |
| **Programador**: Alejandro Figueroa Rodríguez | | **Iteración Asignada**: Primera Iteración |
| **Prioridad**: Alta | | **Tiempo Estimado**: 0.15 días |
| **Riesgo en Desarrollo**: Gespro/Plan de riesgo | | **Tiempo Real**: 0.20 días |
| **Descripción**: El administrador entrena el sistema utilizando las configuraciones pertenecientes al mismo. Para entrenar el sistema se requieren los siguientes datos:   * **Nombre:** El nombre con el que se va a identificar y guardar el sistema de recomendación a entrenar. * **Ruta del sistema:** La ruta donde se va a guardar el sistema una vez termine el proceso de entrenamiento. | | |
| **Observaciones**:   1. La ruta del sistema debe de ser una ruta válida para poder comenzar el entrenamiento. | | |
|  | | |

Tabla 4 Historial de usuario del caso de uso Actualizar

|  |  |  |
| --- | --- | --- |
|  | | |
| **Número**: 1 | **Nombre del requisito**: Actualizar | |
| **Programador**: Alejandro Figueroa Rodríguez | | **Iteración Asignada**: Primera Iteración |
| **Prioridad**: Alta | | **Tiempo Estimado**: 0.15 días |
| **Riesgo en Desarrollo**: Gespro/Plan de riesgo | | **Tiempo Real**: 0.22 días |
| **Descripción**: El administrador actualiza el sistema ya entrenado con los nuevos datos generados por las plataformas. Para la actualización el sistema automáticamente utiliza los datos del entrenamiento. Este requisito requiere los datos con los que se actualiza el sistema, estos datos provienen de la plataforma a la que se integra | | |
| **Observaciones**: | | |
|  | | |

## Modelo de datos

Un modelo de base de datos muestra la estructura lógica de la base, incluidas las relaciones y limitaciones que determinan cómo se almacenan los datos, la relación que existe entre sí, los procesos que los transforman y cómo se accede a ellos. Se basa en la identificación de los objetos primarios que va a procesar el sistema, la composición y atributos de los mismos. En algunos casos, esta base de datos es independiente del sistema software (González Matos, 2021) (Sommerville, 2011).



## Requisitos de la propuesta de solución

Los requisitos funcionales (RF) son declaraciones de las funcionalidades que debe cumplir el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares (González Matos, 2021) (Sommerville, 2011).

### Requisitos Funcionales

Los requisitos funcionales (RF) son declaraciones de las funcionalidades que debe cumplir el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares (González Matos, 2021) (Pressman, 2002).

|  |  |  |  |
| --- | --- | --- | --- |
| Requisitos Funcionales | Nombre del requisito funcional | Prioridad | Complejidad |
| RF1 | Entrenar el sistema | Alta | Alta |
| RF2 | Actualizar el sistema | Alta | Alta |
| RF3 | Utilizar el sistema | Alta | Media |
| RF4 | Mostrar sistema en uso | Alta | Baja |
| RF5 | Listar sistemas | Baja | Baja |
| RF6 | Crear Modelo |  |  |
| RF7 | Entrenar modelo | Alta | Alta |
| RF8 | Guardar modelo para producción | Alta | Alta |
| RF9 | Guardar modelo para reentreno | Alta | Alta |
| RF10 | Guardar el dataset de entrenamiento del modelo | Alta | Alta |
| RF11 | Cargar modelo | Alta | Alta |
| RF12 | Reentrenar modelo | Alta | Alta |
| RF13 | Eliminar modelo | Media | Media |
| RF14 | Listar modelos | Media | Baja |
| RF15 | Utilizar modelos | Alta | Media |
| RF16 | Crear Etapa | Alta | Baja |
| RF17 | Actualizar Etapa | Alta | Media |
| RF18 | Eliminar Etapa | Alta | Baja |
| RF19 | Mostrar Etapa | Alta | Media |
| RF20 | Listar Etapas | Media | Baja |
| RF21 | Crear Configuraciones | Alta | Media |
| RF22 | Actualizar Configuraciones | Media | Media |
| RF23 | Mostrar Configuraciones | Alta | Baja |
| RF24 | Listar Configuraciones | Baja | Baja |

### Requisitos no funcionales

Los requisitos no funcionales (RnF) hacen referencia a las propiedades emergentes del sistema: fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento, son limitaciones sobre servicios o funciones que ofrece el mismo. Los requisitos no funcionales se aplican al sistema como un todo, más que a características o a servicios individuales (González Matos, 2021) (Sommerville, 2011).

|  |  |
| --- | --- |
| N.º | Descripción |
| Usabilidad | |
| RNF1 | El sistema para la gestión y aseguramiento material para la ONBC debe ser una aplicación web. |
| RNF2 | La aplicación debe presentar una interfaz agradable e intuitiva. |
| Confiabilidad | |
| RNF3 | La información manejada por el sistema está protegida de acceso no autorizado de usuarios, definiéndose los permisos según sus roles. |
| Eficiencia | |
| RNF4 | El sistema debe permitir que los usuarios interactúen con él de manera concurrente. |
| RNF5 | El tiempo de demora de una petición al servidor debe ser menor de cinco (5) segundos aproximadamente. |
| Restricciones de Implementación y Diseño | |
| RNF6 | El sistema debe ser desarrollado en su totalidad con tecnologías de código abierto. |
| Software | |
| RNF7 | Para el uso del sistema se requiere una PC cliente con cualquier sistema operativo, que se pueda instalar navegadores web para el uso de la aplicación. |
| RNF8 | La comunicación entre la PC cliente y el servidor de aplicaciones web se realiza a través del protocolo HTTPS. |
| Hardware | |
| RNF8 | Teniendo en cuenta que este tipo de proyectos basados en redes neuronales profundas y análisis de grandes conjuntos de datos necesita de una muy buena potencia de cómputo para echar a andar de la cual no se dispone en el momento de implementación y prueba del mismo el autor ha establecido unos requisitos mínimos con los que cuenta para correr este sistema bajo ciertas condiciones.  Se requiere un mínimo de 80gb de disco duro, una tarjeta de red de 100MB, un procesador Core i3-4170 a 3.70GHz y 16GB de memoria RAM. Se requieren estos requisitos para un modelo de x parámetros, un corpus de entrenamiento de x filas y z columnas. |

## Arquitectura de software

El sistema entero está formado por dos partes: un núcleo o engine el cual tiene una estructura basada en la explicación que dio en el epígrafe 2.1 donde se desarrolla la propuesta de solución. Existen diferentes capas de abstracción como:

Las Etapas

Los Modelos

Las acciones

Donde esta última es la capa de Apis del sistema con las cuales se pueden realizar diferentes acciones como entrenar el sistema, usarlo, actualizarlo, etc.

Este núcleo o Engine está envuelto por un servidor o backend el cual se encarga de crear las Apis que permiten al sistema comunicarse con otros y viceversa para realizar las integraciones. Este envoltorio backend se llevó a cabo por el framerwork FastApi desarrollado en python el cual utiliza la arquitectura Modelo-Vista-Controlador (MVC). Es un patrón de diseño de software que utiliza y a la vez mantiene separados los componentes descritos en esta metodología. Además, es muy utilizado para la arquitectura en la mayoría de los frameworks modernos y uno de los más usados en la industria para poder crear proyectos escalables y modulares.

La idea detrás de MVC es que cada uno de los componentes en su código tenga un propósito, y que esos propósitos sean diferentes. Además de que la forma en que se relacionan estas partes ayuda con la ventaja de realizar un mejor mantenimiento en el futuro. Principalmente, tiene como objetivo dar soporte a los modelos funcionales y mapas mentales de la información relevante para los usuarios, permitiendo un modelo que facilite la consulta y manejo de los mismos. Este patrón permite una separación muy clara de los datos de la aplicación que consta de tres partes interconectadas: vista, modelo y controlador (Sánchez, 2020).

* **Vista**: este elemento hace referencia a la parte de una aplicación que considera la interfaz gráfica. Es decir, cada elemento gráfico que interactúa con el usuario forma parte de la Vista. Su función es obtener la información que requiere el usuario, lo cual se denomina como, evento. Esta capa siempre muestra la información proporcionada por el Modelo.
* **Modelo**: esta capa tiene la función de relacionar y gestionar los datos con los cuales la aplicación va a operar, como consultas, actualizaciones, creación de información o eliminación. Todo esto se le denomina como, Lógica de Negocio. En otras palabras, esta parte se refiere a la transformación de las actividades del mundo real a la forma en la que se va a modificar la información.
* **Controlador**: este componente responde ante eventos o acciones que realiza el usuario a través de la Vista para poder solicitar una operación de la información. Además, tiene la tarea de la elección de la Vista que se mostrará al usuario de acuerdo con la solicitud recibida, por lo que es el vínculo entre el Modelo y la Vista.

## 2.2 Patrones de Diseño

Los **patrones de diseño** son unas técnicas para resolver problemas comunes en el [desarrollo de software](https://es.wikipedia.org/wiki/Desarrollo_de_software) y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias («Patrón de diseño», 2024).

### 2.2.1 Patrones GRASP

En [diseño orientado a objetos](https://es.wikipedia.org/wiki/Dise%C3%B1o_orientado_a_objetos), **GRASP** son *patrones generales de software para asignación de responsabilidades*, es el [acrónimo](https://es.wikipedia.org/wiki/Acr%C3%B3nimo) de "GRASP (object-oriented design General Responsibility Assignment Software Patterns)". Aunque se considera que más que [patrones](https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o) propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software («GRASP», 2020).

#### Experto

El GRASP de experto en información es el principio básico de asignación de responsabilidades. Nos indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo («GRASP», 2020).

Este patrón ha sido fuertemente utilizado en el sistema propuesto ya que permite una mejor estructura y organización de la lógica en el sistema dando como resultado clases u objetos que encapsulan su propia información para llevar a cabo sus tareas haciendo que el código sea más fácil de entender y mantener. Un ejemplo donde se ha utilizado este patrón es en los modelos de aprendizaje, cada modelo se encapsulo como una clase diferente guardando cada uno sus propias tareas y acciones.



Figura #: Clase del modelo de recuperación.

#### Creador

El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación (o [instanciación](https://es.wikipedia.org/wiki/Instancia_(programaci%C3%B3n))) de nuevos [objetos](https://es.wikipedia.org/wiki/Objetos_(programaci%C3%B3n_orientada_a_objetos)) o [clases](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)).

La nueva instancia deberá ser creada por la clase que:

* Tiene la información necesaria para realizar la creación del objeto.
* Usa directamente las instancias creadas del objeto.
* Almacena o maneja varias instancias de la clase
* Contiene o agrega la clase.

Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización («GRASP», 2020).

Para implementar este patrón se han utilizado “acciones”, estas “acciones” no son más que funciones que permiten realizar funcionalidades del sistema como son train (permite entrenar el sistema), fine\_tunning (permite actualizar el sistema ya entrenado con nuevos datos) y use\_engine (permite generar recomendaciones). Estas acciones son las encargadas de implementar el patrón **creador** instanciando modelos y clases para el manejo de datos.

#### Alta cohesión y bajo acoplamiento

#### Alta cohesión

La información que almacena una [clase](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)) debe ser coherente y debe estar (en la medida de lo posible) relacionada con la clase («GRASP», 2020).

**Bajo acoplamiento**

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la [reutilización](https://es.wikipedia.org/wiki/Reutilizaci%C3%B3n_de_c%C3%B3digo), y disminuyendo la dependencia entre las clases («GRASP», 2020).

En el sistema propuesto el autor ha tenido muy en cuenta los siguientes puntos basándose en estos dos patrones:

1. **Cohesión Coincidente**: El módulo realiza múltiples tareas, sin ninguna relación entre ellas.
2. **Cohesión de Comunicación**: Las tareas corresponden a una secuencia de pasos propia del “producto” y todas afectan a los mismos datos.
3. **Cohesión Funcional**: Cuando el módulo ejecuta una y sólo una tarea, teniendo un único objetivo a cumplir, se dice que tiene Cohesividad Funcional.
4. **Acoplamiento de Contenido**: Cuando un módulo referencia directamente el contenido de otro módulo. (En lenguajes de alto nivel es muy raro)
5. **Acoplamiento Común**: Cuando dos módulos acceden (y afectan) a un mismo valor global.
6. **Acoplamiento de Control**: Cuando un módulo le envía a otro un elemento de control que determina la lógica de ejecución del mismo.



Figura #: La clase DataPipeline muestra una alta cohesión, ya que todas sus responsabilidades están estrechamente relacionadas con el procesamiento y manipulación de datos.



Figura #: La clase ModelConfig es utilizada para pasar configuraciones a los modelos  lo que reduce el acoplamiento entre la configuración y la implementación del modelo.

### 2.2.2 Patrones GOF

**Singleton:** Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia (*Patrones Gof - EcuRed*, s. f.).



Figura #: Creación de dos instancias globales en el sistema

#### Factory Method

Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que se crea. La clase DataPipeline actúan utiliza este patron ya que muchos de sus métodos crean objetos específicos Vease la figura (clase datapipeline arriba) (*Patrones Gof - EcuRed*, s. f.).

#### Facade

Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. La clase DataPipeline utiliza este patron ya que actúa como una fachada, proporcionando una interfaz simplificada para varias operaciones complejas de procesamiento de datos. Vease la figura (clase datapipeline arriba) (*Patrones Gof - EcuRed*, s. f.).

#### Template Method (Método plantilla)

Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.

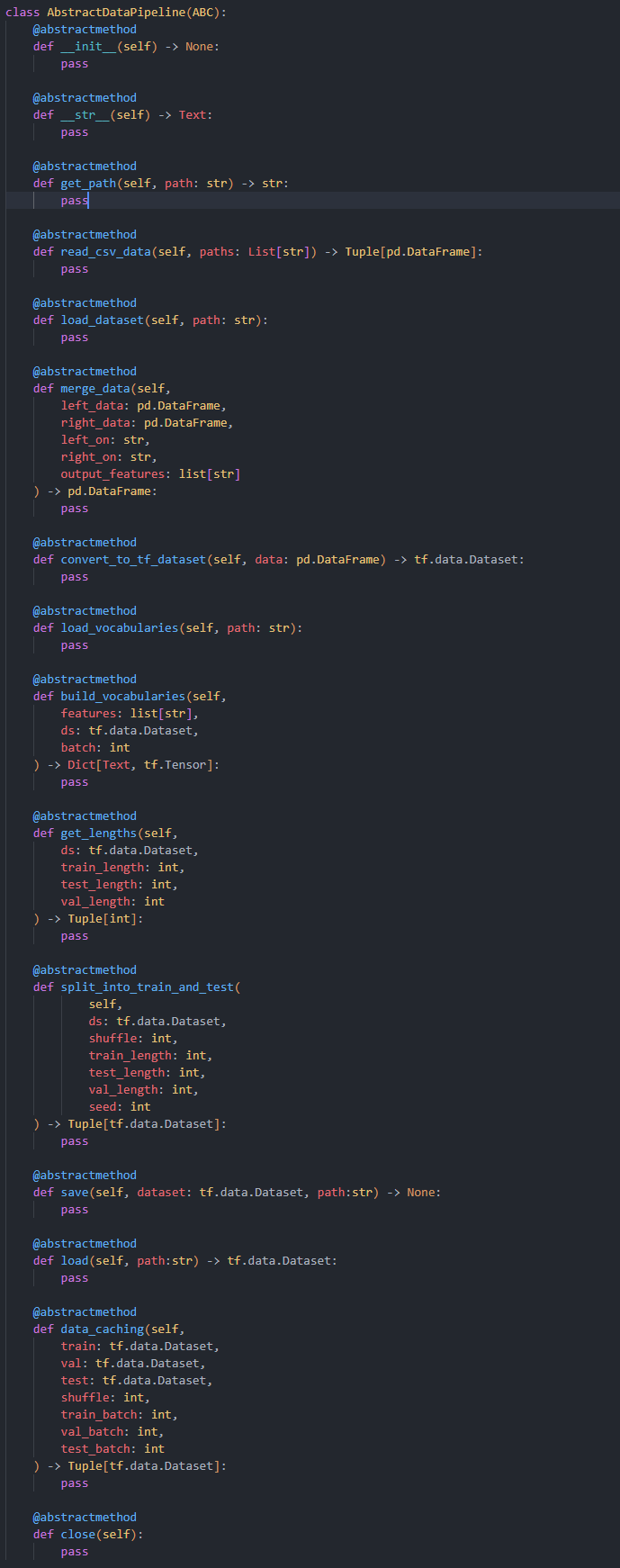


Figura #: Clase abstracta que se utiliza como platilla para la clase DataPipeline

## Conclusiones del capítulo

Como parte del desarrollo del presente capítulo se determinan las siguientes conclusiones parciales:

* El análisis de las características del sistema y la modelación del dominio permitió identificar los principales requisitos funcionales y no funcionales del sistema de recomendaciones, los cuales fueron agrupados y categorizados por casos de uso.
* El diseño de los diagramas de clases el entendimiento sobre la composición física y lógica del sistema.
* Los artefactos generados según la metodología de desarrollo utilizada y los patrones de arquitectura y diseño descritos, constituyeron una guía fundamental para la construcción de la propuesta de solución.