

Euskoflix - Sprint 2

Ingeniería del Software

Cañadillas Patricio, Daniel

González Llaguno, Urtzi

Lahuerta Vázquez, Xabier

Pérez García, Iván

11 de abril de 2019

Índice

1. Introducción	4
2. Planificación	5
2.1. Historias de usuario	5
2.2. Sprints	6
2.3. Herramientas	6
3. Diseño	8
3.1. Diagrama relacional	8
3.2. Diagrama de clases	8
3.2.1. Sprint 1	8
3.2.2. Sprint 2	12
3.3. Diagramas de secuencia	16
3.3.1. HU4 - Filtrado basado en personas	16
4. Desarrollo	18
4.1. Sprint 1	18
4.2. Sprint 2	20
5. Anexo	21
5.1. Documentación de pruebas	21
5.1.1. Sprint 1	21

5.1.2. Sprint 2	21
---------------------------	----

1. Introducción

El proyecto Euskoflix forma parte de la asignatura Ingeniería del Software. El objetivo consiste en desarrollar un sistema de recomendación entorno a películas.

El proyecto pretende ser una aplicación de los conocimientos teóricos aprendidos en clase, tales como: la aplicación de patrones de diseño de software; los principios SOLID; la metodología de trabajo ágil Scrum y la importancia de una documentación adecuada con diagramas que expliquen el funcionamiento del sistema.

El desarrollo de la aplicación sigue el marco de desarrollo ágil **Scrum**. Primero se establecen las **historias de usuario** en base a los requisitos impuestos por el cliente (en este caso el cliente es la persona que imparte la asignatura). A continuación, se realiza una planificación temporal que se representa en **sprints**, los cuales consisten en entregar un prototipo del producto para mostrar al cliente el estado presente del desarrollo.

2. Planificación

Hemos optado por seguir la planificación propuesta en clase, donde se han establecido unas fechas para los sprints y nosotros tenemos que crear las historias de usuario.

2.1. Historias de usuario

Las historias de usuario son una representación sencilla de los requisitos impuestos por el cliente. A las 3 historias de usuarios definidas para el primer sprint, hemos añadido otras 3 historias de usuario relacionadas con los filtros realizados para el segundo sprint.

1. **Iniciar el sistema:** Se crea el esqueleto vacío de las estructuras de datos.
2. **Carga de datos:** Se cargan los datos de usuarios y películas en el sistema.
3. **Obtener datos sobre una película:** Se conecta al servicio TMDb para obtener información adicional sobre una película concreta.
4. **Filtrado basado en personas:** Se identifican los productos que puedan ser de interés a una persona basándose en el criterio de otras personas que han mostrado tener preferencias similares.
5. **Filtrado basado en productos:** Los sistemas de recomendación que utilizan esta técnica se basan en la similitud de los productos entre sí para estimar las valoraciones que las personas otorgarían a los productos.
6. **Filtrado basado en el contenido:** Esta técnica determina la idoneidad de un producto para una persona de acuerdo a las características del producto y las preferencias de la persona.

2.2. Sprints

Se han establecido 3 sprints. Comenzando el Sprint 1 el 27 de febrero y terminando el Sprint 3, es decir, el proyecto, el 8 de mayo.

- **Sprint 1 - Cargar datos:** Desde el 27 febrero hasta el 20 de marzo.
- **Sprint 2 - Valoración del producto:** Desde el 20 de marzo hasta el 10 de abril.
- **Sprint 3 - Elaboración de la recomendación:** Desde el 10 de abril hasta el 8 de mayo.

2.3. Herramientas

Se han utilizado las siguientes herramientas para apoyar el desarrollo del proyecto:

- **Git (concretamente GitLab):** Para el control de versiones del código y para gestionar las tareas Scrum.
- **L^AT_EX:** Para la composición de documentos.
- **Maven:** Gestor de dependencias y librerías externas del código.
- **SQLite:** Sistema de gestión de bases de datos.
- **Visual Paradigm:** Creación de diagramas de clases, relacionales y de secuencia.

uni > Ingeniería del Software > Issues

Open 0 Closed 9 All 9

Search or filter results... Created date

Documentación actas y test #10 · opened 1 day ago by Daniel Cañadillas · Carga de datos · Mar 21, 2019	CLOSED · updated 8 hours ago
Documentación diagramas #9 · opened 1 day ago by Daniel Cañadillas · Carga de datos · Mar 21, 2019	CLOSED · updated 1 day ago
Rellenar estructuras de datos #8 · opened 1 week ago by Xabier Lahuerta Vázquez · Carga de datos · Mar 20, 2019	CLOSED · updated 1 day ago
Crear estructuras de datos. #7 · opened 1 week ago by Xabier Lahuerta Vázquez · Carga de datos · Mar 20, 2019	CLOSED · updated 1 day ago
Creación Base de Datos #5 · opened 1 week ago by Xabier Lahuerta Vázquez · Carga de datos · Mar 20, 2019	CLOSED · updated 1 day ago
Creación Interfaz Gráfica #4 · opened 1 week ago by Xabier Lahuerta Vázquez · Carga de datos · Mar 20, 2019	CLOSED · updated 1 day ago
Diagrama de Secuencia Carga de Datos #3 · opened 1 week ago by Ivan · Carga de datos · Mar 20, 2019	CLOSED · updated 1 day ago
Decisión de herramientas #2 · opened 1 week ago by Xabier Lahuerta Vázquez · Carga de datos · Mar 8, 2019 · Done	CLOSED · updated 1 week ago
Diagrama de clases #1 · opened 1 week ago by Daniel Cañadillas · Carga de datos · Mar 20, 2019	CLOSED · updated 1 day ago

Figura 1: Gestión de Scrum con GitLab.

uni > Ingeniería del Software > Milestones

Open 3 Closed 0 All 3

Filter by milestone name Due soon New milestone

Carga de datos Feb 27, 2019–Mar 20, 2019 Expired uni / Ingeniería del Software	9 Issues · 0 Merge Requests 100% complete	Close Milestone
Valoración del producto Mar 20, 2019–Apr 10, 2019 uni / Ingeniería del Software	0 Issues · 0 Merge Requests 0% complete	Close Milestone
Elaboración de la recomendación Apr 10, 2019–May 8, 2019 Upcoming uni / Ingeniería del Software	0 Issues · 0 Merge Requests 0% complete	Close Milestone

Figura 2: Gestión de sprints con GitLab.

3. Diseño

3.1. Diagrama relacional

El programa utiliza SQLite como sistema de gestión de base de datos. En la base de datos se almacena información sobre: usuarios, películas y géneros. También se almacena la información surgida de las relaciones entre las entidades, tales como: valoraciones, etiquetas y géneros de las películas. A continuación se muestra el diagrama relacional de la base de datos:

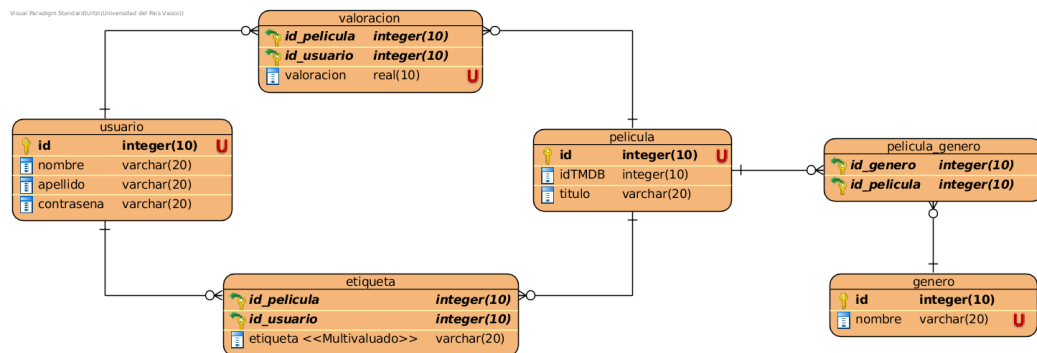


Figura 3: Diagrama relacional de la base de datos.

3.2. Diagrama de clases

3.2.1. Sprint 1

A continuación, se muestra la estructura del diagrama de clases diseñado para el Sprint 1:

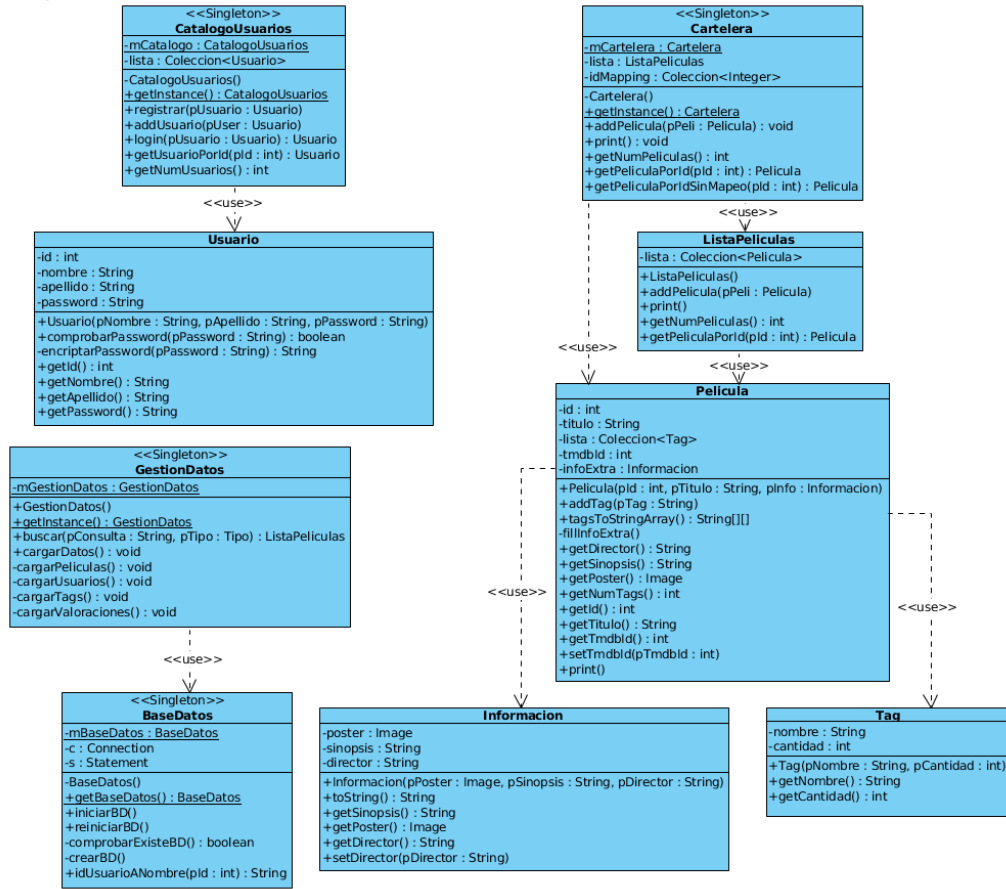


Figura 4: Diagrama de clases tras Sprint 1 (parte 1).

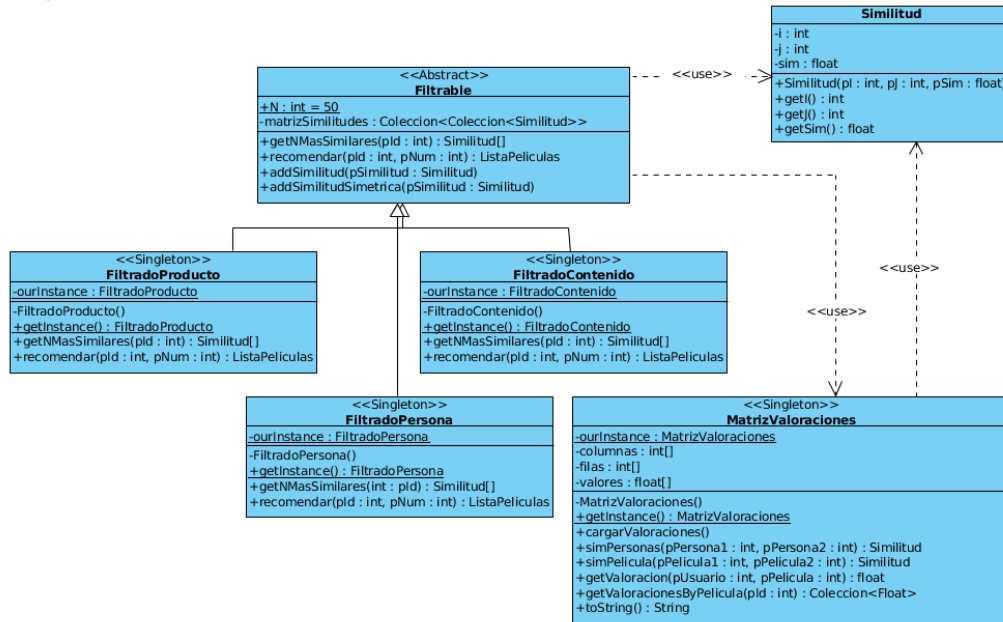


Figura 5: Diagrama de clases tras Sprint 1 (parte 2).

- **CatalogoUsuarios** : Singleton usado para almacenar y tratar la información de los usuarios.
 - **registrar(pUsuario)** : método para los nuevos usuarios que se registren.
 - **login(pUsuario)** : método para los usuarios registrados inicien sesión.
- **Usuario** : Almacena la información del usuario.
 - **comprobarPassword(pPassword)** : método para comprobar que el usuario ha escrito bien su contraseña.
 - **encriptarPassword(pPassword)** : método para encriptar nuestra contraseña y no sea fácil descubrirla.
- **Cartelera** : Singleton usado para almacenar y tratar la información de las películas.

- ListaPelículas : Clase encargada de tratar la información de las películas.
- Película : Almacena la información de la película.
 - fillInfoExtra() : método para conseguir la información extra de la película.
- Información : Almacena la información de la película obtenida de la API.
- Tag : Almacena el nombre de la etiqueta y la cantidad de veces que se ha mencionado dicha etiqueta.
- GestiónDatos : Fachada encarga de gestionar la obtención y manejo de datos desde la base de datos o la API.
 - cargarDatos() : método para cargar los datos de nuestra base datos.
 - cargarPelículas() : método para cargar las películas de nuestra base datos.
 - cargarUsuarios() : método para cargar los usuarios de nuestra base datos.
 - getValoraciones() : método par obtener todas las valoraciones.
 - getValoracionesUsuarios() : método para obtener las valoraciones de los usuarios.
 - getValoracionesPelículas() : método para obtener las valoraciones de las películas.
 - getInfoExtra() : método para los datos extra de la película a través de la API.
 - getTags(pPelícula) : método para obtener las apariciones de cada tag en la película.
- BaseDatos : Clase encargada de la creación y de las consultas a la bases de datos.
 - crearBD() : método para crear la base de datos.

- `anadirDatos()` : método para rellenar la base de datos creada con los datos de los CSV.
- `MatrizValoraciones` : Singleton encargado de gestionar la matriz de valoraciones
 - `getValoracionesByPelicula(pID)` : método para obtener las valoraciones de una película.
- `Similitud` : Clase que almacena la similitud y el índice que los componentes que son similares
- `Filtrable` : Clase abstracta que contiene el array de similitudes.
- `FiltradoProducto` : Clase encargada de filtrar en base del producto.
- `FiltradoContenido` : Clase encargada de filtrar en base del contenido.
- `FiltradoPersona` : Clase encargada de filtrar en base de los usuarios.

3.2.2. Sprint 2

A continuación, se muestra la estructura del diagrama de clases diseñado para el Sprint 2, que ha sufrido algunos cambios respecto al diagrama inicial, ya que se le han añadido nuevas clases y se han tenido que modificar clases ya existentes para ajustar nuestro programa a un programa eficaz y eficiente:



- Clases nuevas :
 - ListaTags : Clase encargada de tratar la información de los tags.
 - rellenarTf() : método encargado de que dada una película y una ListaEtiquetasFiltrado nos calcule los Tf.
 - Filtrado : Clase que nos almacenará el registro de los filtrados realizados con el usuario actual para poder mostrarle los resultados según el tipo de recomendación que el usuario desee.
 - TipoRecomendacion : Clase enumerada creada para indicar adecuadamente en que método de recomendación nos encontramos actualmente.
 - ListaEtiquetasFiltrado : Clase que almacena los ids de las pelis junto a sus tags y los tfidf de cada tag.
 - completarIDF() : calcula el IDF de los tags de las películas.
 - calcularRelevanciasSimilitudes() : método que calcula las similitudes de la lista de etiquetas de filtrado de las películas que ha visto el usuario conectado con los demás usuarios.
 - ListaPeliculasRecomendadas : Clase que almacena todas las películas que se pueden recomendar al usuario conectado según los filtrados realizados.
 - Normalizable : Clase abstracta encargada de definir métodos y atributos para que las clases usuario y película puedan trabajar con la media y cuasi desviación.
 - normalizar() : método encargado de normalizar un valor según el usuario o película.
 - desnormalizar() : método encargado de desnormalizar un valor según el usuario o película.
- Clases modificadas :
 - CatalogoUsuarios :
 - cargarMediasDesviacionesUsuarios() : método que empleamos para cargar las medias y desviación típicas de los usuarios.
 - cargarModeloPersona() : método en el que cargamos el modelo persona respecto al usuario conectado en ese momento.

- ListaPelículas :
 - cargarMediasDesviacionesPelículas() : método que empleamos para cargar las medias y desviación típicas de las películas.
 - cargarModeloProducto() : método en el que cargamos el modelo producto respecto al usuario conectado en ese momento.
 - getPelículasNoValoradas() : método que empleamos para obtener las películas no vistas por dado un usuario.
- MatrizValoraciones :
 - simPersonas() : método encargado que dado dos usuarios calcule su similitud.
 - simPelicula() : método encargado que dada dos películas calcule su similitud.
 - coseno() : método encargado de hallar el coseno entre dos elementos.
 - cargarMediasDesv() : método encargado que dado un usuario calcule su media y desviación típica.
 - tieneValoracionesUsuario() : método que nos dice si un usuario tiene valoraciones.
 - cargarMediasDesv() : método encargado que dada una película calcule su media y desviación típica.
 - tieneValoracionesPelicula() : método que nos dice si una película tiene valoraciones.
- Filtrable : Clase abstracta que define los métodos que luego usaran las clases de los filtrados según el tipo que sea.
 - getNMasSimilares() : método que emplearemos que pasándole el id de un usuario nos devuelva una lista con los usuarios más similares.
 - recomendar() : método que deberán definir cada tipo de recomendación y recomendará las películas al usuario que haya iniciado sesión.
 - calcularRecomendaciones() : método que deberán definir cada tipo de recomendación y calculará las recomendaciones para el usuario que haya iniciado sesión.

- `generarValoracionRecomendada()` : método que nos generará la valoración que el usuario actual le pondría a la película recomendada.
- `FiltradoContenido` : Clase encargada de hacer la recomendación de películas según el filtrado por contenido que hereda de la clase `Filtrable`.
 - `calcularRecomendaciones()` : método para hallar las recomendaciones según el filtrado por contenido.
- `FiltradoPersona` : Clase encargada de hacer la recomendación de películas según el filtrado por persona que hereda de la clase `Filtrable`.
 - `calcularRecomendaciones()` : método para hallar las recomendaciones según el filtrado por persona.
- `FiltradoProducto` : Clase encargada de hacer la recomendación de películas según el filtrado por producto que hereda de la clase `Filtrable`.
 - `calcularRecomendaciones()` : método para hallar las recomendaciones según el filtrado por producto.

3.3. Diagramas de secuencia

Para facilitar la comprensión del funcionamiento del sistema, a continuación se muestra la secuencia de las tareas realizadas por el programa.

3.3.1. HU4 - Filtrado basado en personas

La cuarta historia de usuario consiste en recomendar películas que puedan ser de interés para una persona en base al criterio de personas con gustos similares. Para ello, se han realizado dos diagramas de secuencia. En la figura 7 se ve el proceso de creación de la recomendación, mientras que en la figura 8 se muestra como, una vez creada la recomendación, esta se le recomienda a un usuario en concreto.

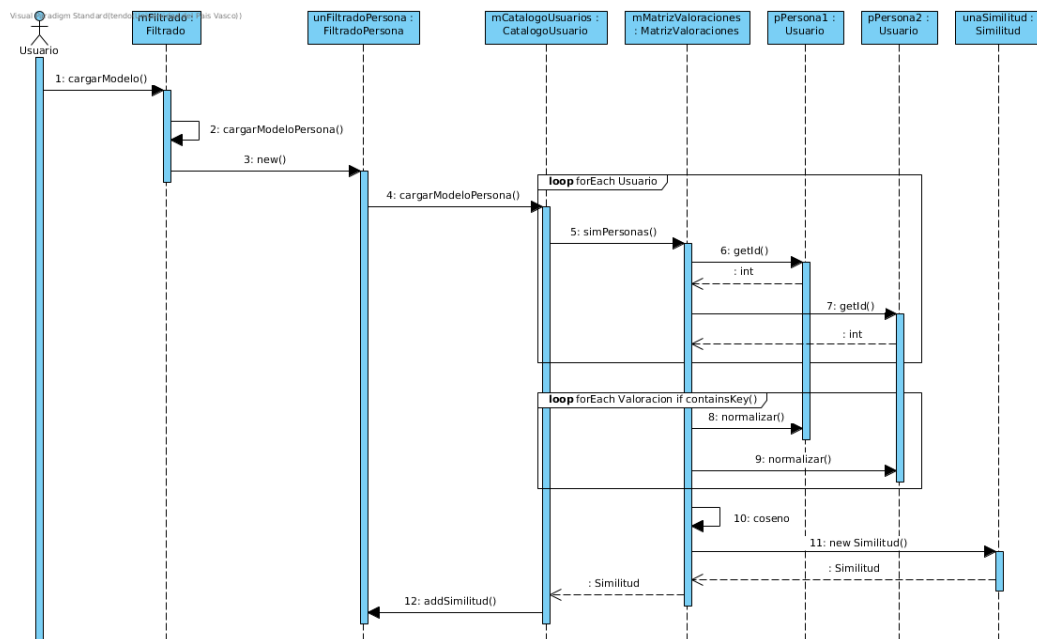


Figura 7: Diagrama de secuencia de la creación del filtrado basado en personas.

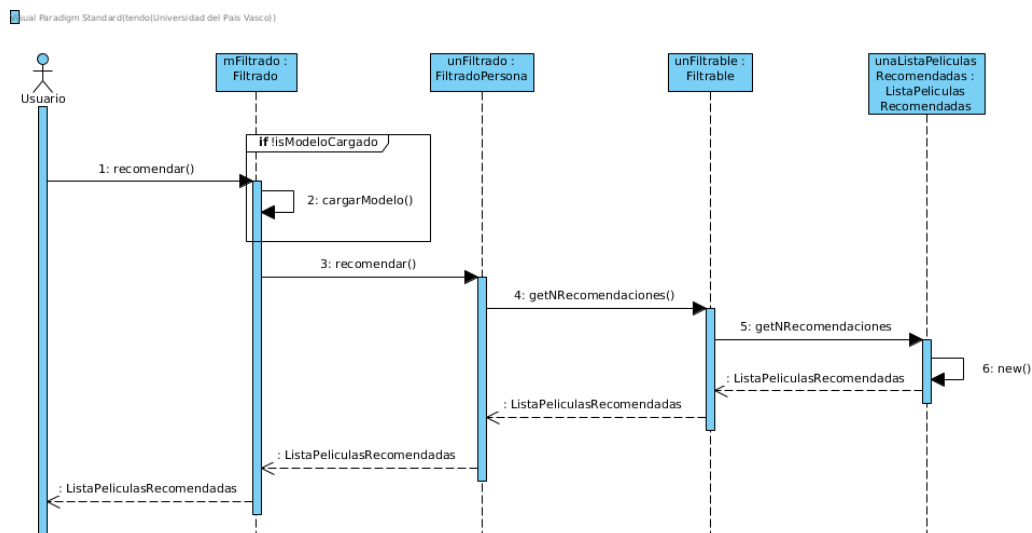


Figura 8: Diagrama de secuencia de la recomendación del filtrado basado en personas.

4. Desarrollo

4.1. Sprint 1

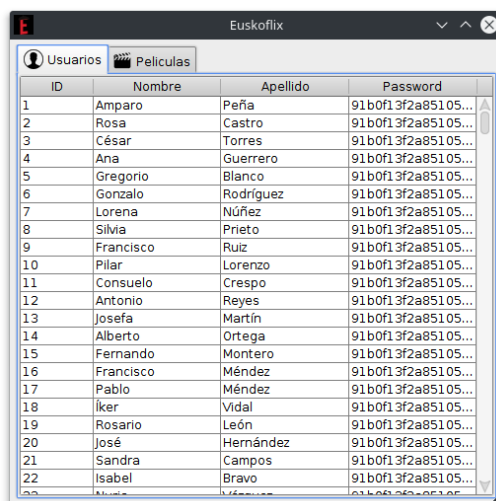
En este Sprint se ha realizado la carga de los datos necesarios para que el sistema pueda hacer recomendaciones.

Se han implementado las estructuras adecuadas para la recoger y utilizar la información sobre las películas y usuarios. También se ha desarrollado una interfaz gráfica que permite visualizar el contenido de dichas estructuras para verificar si la carga ha sido correcta.

Adicionalmente, se ha implementado la API de TMDb (The Movie Database) para obtener el póster y la sinopsis de la película.



Figura 9: Ventana de carga del sistema.

The image is a screenshot of a web application window titled 'Euskoflix'. It has two tabs: 'Usuarios' (selected) and 'Películas'. Below the tabs is a table with four columns: 'ID', 'Nombre', 'Apellido', and 'Password'. The table contains 22 rows of user data. The window has a standard OS-style title bar with minimize, maximize, and close buttons.

ID	Nombre	Apellido	Password
1	Amparo	Peña	91b0f13f2a85105...
2	Rosa	Castro	91b0f13f2a85105...
3	César	Torres	91b0f13f2a85105...
4	Ana	Guerrero	91b0f13f2a85105...
5	Gregorio	Blanco	91b0f13f2a85105...
6	Gonzalo	Rodríguez	91b0f13f2a85105...
7	Lorena	Núñez	91b0f13f2a85105...
8	Silvia	Prieto	91b0f13f2a85105...
9	Francisco	Ruiz	91b0f13f2a85105...
10	Pilar	Lorenzo	91b0f13f2a85105...
11	Consuelo	Crespo	91b0f13f2a85105...
12	Antonio	Reyes	91b0f13f2a85105...
13	Josefa	Martín	91b0f13f2a85105...
14	Alberto	Ortega	91b0f13f2a85105...
15	Fernando	Montero	91b0f13f2a85105...
16	Francisco	Méndez	91b0f13f2a85105...
17	Pablo	Méndez	91b0f13f2a85105...
18	Iker	Vidal	91b0f13f2a85105...
19	Rosario	León	91b0f13f2a85105...
20	José	Hernández	91b0f13f2a85105...
21	Sandra	Campos	91b0f13f2a85105...
22	Isabel	Bravo	91b0f13f2a85105...

Figura 10: Ventana que contiene la lista de los usuarios del sistema.

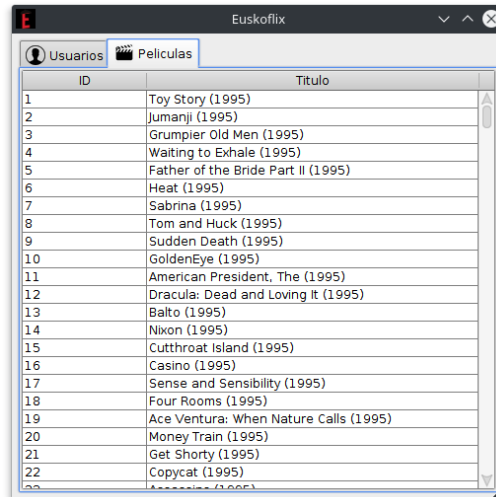


Figura 11: Ventana que contiene la lista de las películas del sistema.

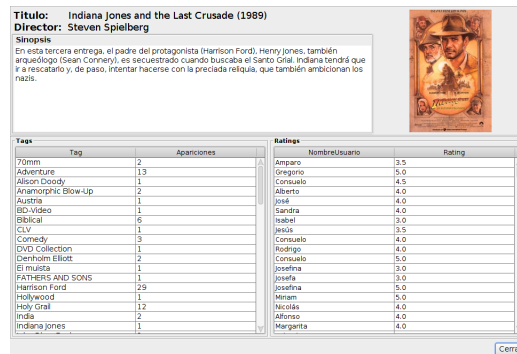


Figura 12: Ventana mostrando información sobre una película.

Conclusiones del primer sprint: Por el momento no se han presentado dificultades. El diseño inicial se ha hecho pensando de cara a futuras expansiones y la implementación ha ido acorde al diseño.

4.2. Sprint 2

En este segundo Sprint se ha realizado las operaciones necesarias para recomendar películas a los usuarios.

Se han implementado nuevas clases adecuadas para los cálculos de los métodos proporcionados de recomendación con los datos que hemos rellenado nuestras estructuras. Para este sprint no es necesario crear ninguna interfaz solo es necesario hacer las clases adecuadas para hacer bien los cálculos y hacer sus respectivos test para comprobar su pleno funcionamiento.

Nos hemos centrado en la implementación de los tres distintos tipos de modelado, todos bajo una misma fachada encargada de generar la recomendación que el usuario necesite según el modelo y la cantidad que ha seleccionado en dicha petición. Decidimos crear un jerarquía de filtros ya que los tres comparten atributos como la matriz de similitudes y la lista de recomendaciones del usuario y de esta forma el añadir filtros de las mismas características no supone ningún problema.

Tras analizar el tiempo de ejecución de los distintos métodos y teniendo en cuenta la poca paciencia que tiene el usuario promedio con la pantallas de carga , decidimos crear un sistema de login obligando a calcular lo relativo al usuario logueado y no a todos los usuarios, reduciendo los tiempos de carga al mínimo, aun así, si fuera necesario nuestro código estaría preparado para calcular todo lo necesario con los mínimos cambios requeridos.

Durante este sprint se ha requerido del uso de diversas herramientas para asegurar el buen funcionamiento y el contraste con programas y artículos referentes a los filtros que hemos tenido que implementar para asegurar la mejor implementación y ejecución posible minimizando espacio en disco y tiempo de ejecución.

5. Anexo

5.1. Documentación de pruebas

5.1.1. Sprint 1

5.1.2. Sprint 2

En este sprint nos hemos centrado mucho en las pruebas unitarias de nuestra fachada encargada de las recomendaciones para asegurarnos del correcto funcionamiento de nuestro programa. Para esto además de las pruebas unitarias propiamente dichas hemos realizado cálculos a mano ayudados por un script en python que nos calculaba la similitud mediante el coseno dados dos vectores de la misma longitud.

```
#!/usr/bin/env python
from math import*
def square_rooted(x):
    return round(sqrt(sum([a*a for a in x])),3)
def cosine_similarity(x,y):
    numerator = sum(a*b for a,b in zip(x,y))
    denominator = square_rooted(x)*square_rooted(y)
    return round(numerator/float(denominator),3)
```

Partiendo del cálculo de la similitud por parte de nuestro programa, pasamos a comprobar las funciones propiamente dichas de nuestra fachada y de todo lo que necesario para que genere la recomendación solicitada.

A continuación se detalla el procedimiento seguido para la realización de las pruebas unitarias de los tres tipos de filtrados: colaborativo basado en personas, colaborativo basado en productos y basado en contenidos.

Se han calculado tanto a mano como con un script en python los resultados que se deberían obtener mediante los tres tipos distintos de filtrado, tomando como datos las valoraciones representadas en la siguiente tabla:

Tabla 1: Matriz de valoraciones utilizada para realizar las pruebas unitarias.

	i_1	i_2	i_3	i_4	i_5	i_6
u_1	—	3,2	2,1	4	—	—
u_2	4,5	—	3,5	3	3,2	—
u_3	3,5	2	4	3,5	4	2
u_4	5	4	4,5	4,7	3,6	—
u_5	—	4,5	5	—	—	4

Filtrado colaborativo basado en personas

```
#!/usr/bin/env python
valoraciones = [
    [0,3,2,2,1,4,0,0],
    [4,5,0,3,5,3,3,2,0],
    [3,5,2,4,3,5,4,2],
    [5,4,4,5,4,7,3,6,0],
    [0,4,5,5,0,0,4]
]

no_valoradas = [0,4,5]

def filtradoPersona():
    print ("filtrado_persona")
    for j in no_valoradas:
        numerador = 0
        denominador = 0
        for i in range(1,len(valoraciones)):
            numerador+=valoraciones[i][j]*cosine_similarity(valoraciones[0],valoraciones[i])
            denominador+= cosine_similarity(valoraciones[0],valoraciones[i])

        print(f'i{j+1}:-{str(numerador/denominador)}')
```

Filtrado colaborativo basado en productos

```
#!/usr/bin/env python
valoraciones = [
    [0,3,2,2,1,4,0,0],
    [4,5,0,3,5,3,3,2,0],
    [3,5,2,4,3,5,4,2],
    [5,4,4,5,4,7,3,6,0],
    [0,4,5,5,0,0,4]
]

no_valoradas = [0,4,5]
```

```

def filtradoProducto():
    valoracionesPeliculas = [
        [0,4.5,3.5,5,0],
        [3.2,0,2,4,4.5],
        [2.1,3.5,4,4.5,5],
        [4,3,3.5,4.7,0],
        [0,3.2,4,3.6,0],
        [0,0,2,0,4]
    ]
    print("\n\nfiltrado_pelicula")

    for j in no_valoradas:
        numerador = 0
        denominador = 0

        for i in range(1,len(valoracionesPeliculas)):
            numerador+=valoraciones[0][i]*cosine_similarity(valoracionesPeliculas[j],valoracionesPeliculas[i])
            if valoraciones[0][i] != 0:
                denominador+= cosine_similarity(valoracionesPeliculas[j],valoracionesPeliculas[i])

        print(f'i{j+1}:-{str(numerador/denominador)}')

```

Filtrado basado en contenidos

```

#!/usr/bin/env python
valoraciones = [
    [0,3.2,2.1,4,0,0],
    [4.5,0,3.5,3,3.2,0],
    [3.5,2,4,3.5,4,2],
    [5,4,4.5,4.7,3.6,0],
    [0,4.5,5,0,0,4]
]

no_valoradas = [0,4,5]

def filtradoContenido():
    print("\n\nfiltrado_contenido")

    nt = {'aburrida': 1, 'alucinante': 1, 'dinamica': 2, 'divertida': 2, 'encantadora': 3, 'filosofica': 3, 'horrible': 1, 'mejorable': 2}
    tf = [{'dinamica': 1, 'encantadora': 1, 'filosofica': 1}, {'aburrida': 1, 'filosofica': 1, 'mejorable': 2}, {'dinamica': 1, 'encantadora': 1, 'filosofica': 3}, {'alucinante': 2, 'divertida': 2}, {'encantadora': 2, 'mejorable': 1}, {'divertida': 1, 'horrible': 1}]

    for dc in tf:
        for key, value in dc.items():
            dc[key] = value * log(len(valoraciones[0])/nt[key])
        x = 0
        for v in dc.values():
            x += pow(v,2)
        x = sqrt(x)
        for k in dc.keys():

```

```

dc[k] = dc[k]/x

relev = [None] * 5

for i in range(0,5):
    relev[i] = {'aburrida': 0, 'alucinante': 0, 'dinamica': 0, 'divertida': 0, 'encantadora': 0, 'filosofica': 0, 'horrible': 0, 'mejorable': 0}

for i in range(0,len(valoraciones)):
    dc = []
    for j in range(0,len(valoraciones[i])):
        if (valoraciones[i][j]>= 3.5):
            for k in tf[j].keys():
                if (k not in dc):
                    dc.append(k)
                    for x in range(0,len(tf)):
                        if k in tf[x]:
                            relev[i][k] += tf[x][k]

valoradas = [1,2,3]

for j in no_valoradas:
    numerador = 0
    denominador = 0
    for i in range(1,len(valoraciones)):
        numerador+=valoraciones[i][j]*cosine_similarity(relev[0].values(),relev[i].values())
        denominador+= cosine_similarity(relev[0].values(),relev[i].values())

    print('i{j+1}:-{str(numerador/denominador)}')

```

Resultados

Tabla 2: Tabla descriptiva de los jUnits respectivos a los tres tipos de filtrado.

Id	Objetivo	Entrada	Condiciones de Ejecución	Película Recomendada	Resultado esperado	Resultado Obtenido
1	Filtrado Personas	Fichero Test Usuario 1	N = 30	i ₁	3.3408490075004718	3.3409
2	Filtrado Productos			i ₅	2.7865983552531834	2.7866
				i ₆	1.4587493934195785	1.4587
				i ₁	3.113335797057115	3.1133
3	Filtrado Contenido			i ₅	3.1099595124884605	3.1099
				i ₆	2.8151062314139437	2.8151
		i ₁	3.270152037256428	3.2701		
		i ₅	2.9302327249783064	2.9301		
				i ₆	1.695080131359008	1.6951

Las siguientes tres gráficas representan las valoraciones calculadas para las películas que no ha valorado el usuario 2048.

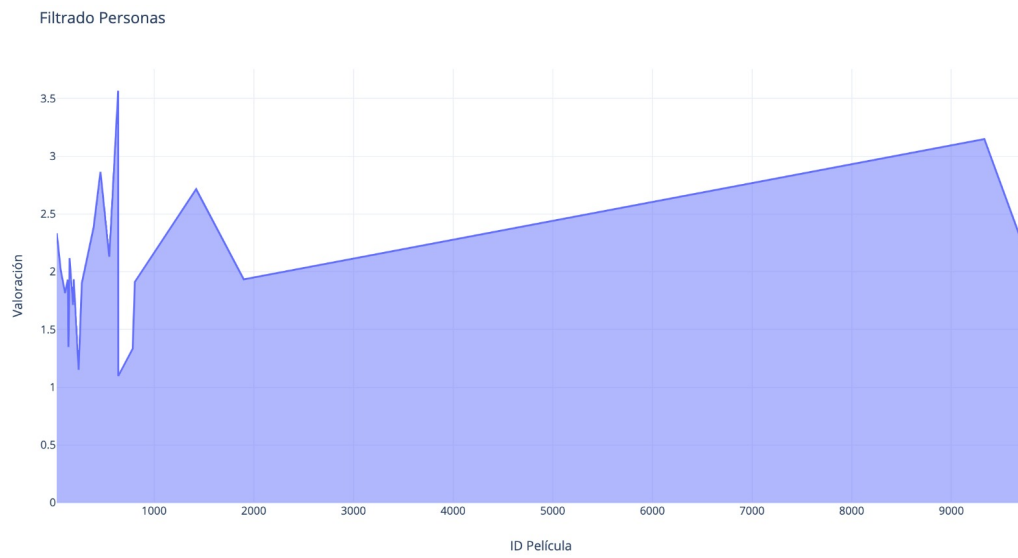


Figura 13: Filtrado basado en personas

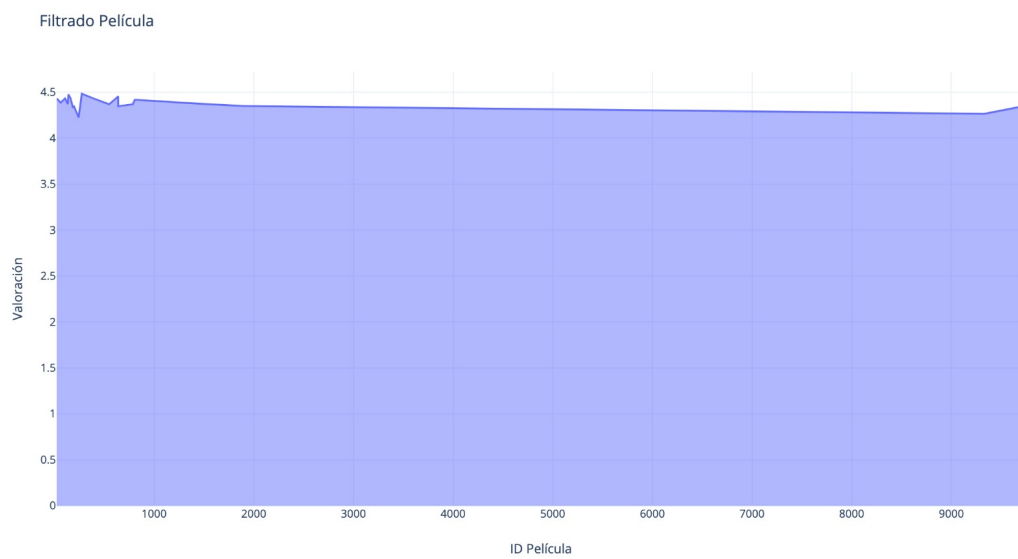


Figura 14: Filtrado basado en productos

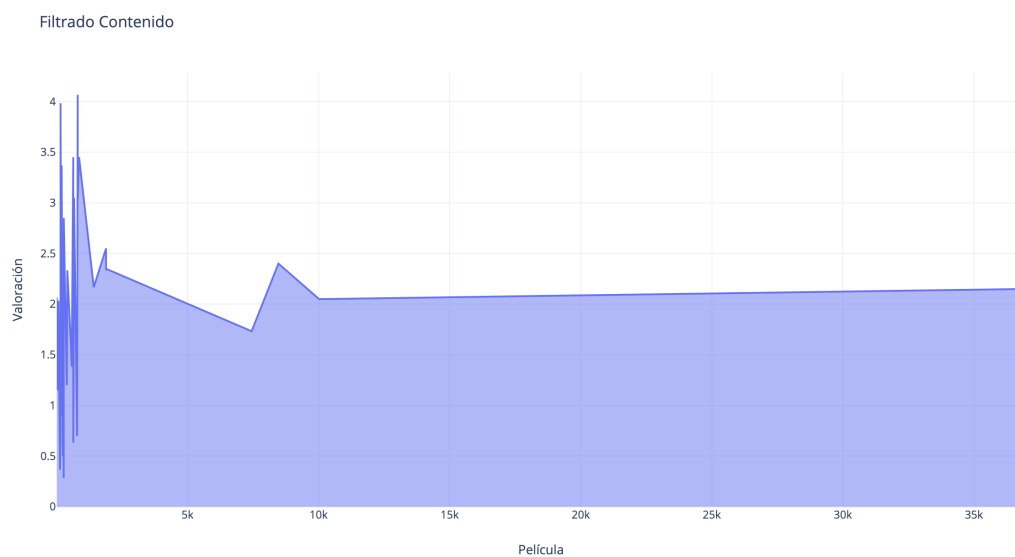


Figura 15: Filtrado basado en contenidos