Distribució de productes a un supermercat

Descripció general

Segona Entrega

Identificador de l'equip: subgrup-prop41.1
Antonia Laura Apolzan: antonia.laura.apolzan@estudiantat.upc.edu
Hajweria Hussain Shaheen: hajweria.hussain@estudiantat.upc.edu
Ariadna Mantilla Puma: ariadna.mantilla@estudiantat.upc.edu
Guillem Sturlese Ruiz: guillem.sturlese@estudiantat.upc.edu

Índex

Consideracions del sistema

- Hem decidit utilitzar un rang entre 0 i 1 per representar la similitud entre dos productes. Això garanteix que tots els valors estiguin dins d'un interval normalitzat, la qual cosa facilita la interpretació i comparació de les similituds, independentment de com es calculin. 0 indica que els elements són completament diferents (sense similitud) i 1 indica que els elements són completament similars.
    - Hem decidit que un producte amb si mateix tingui una similitud de 0, perquè un producte mai es pot col·locar al costat de si mateix.

- La disposició de la prestatgeria s'actualitza si canviem la similitud entre dos productes o afegim un producte, ja que abans hem decidit que tots els productes del conjunt de productes estan presents a la prestatgeria.

- Tenim en compte que a l'hora d'executar els nostres jocs de proves, el programa no valida les dades (i llença una excepció impedint el contínuament del programa) si l'usuari no proporciona el format correcte d'aquestes.

- L'usuari només podrà modificar la posició d'un producte si l'intercanvia amb un altre, és a dir, no podrà moure solament un producte.

- Els algoritmes necessiten d'un mínim de 2 productes per funcionar, amb 1 o 0 productes no hi han similituds, i per tant, no té sentit utilitzar-los.

Gestió de prestatgeria

Cas d’ús Crear Prestatgeria

Nom: Crear prestatgeria

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol crear una prestatgeria.
2. El sistema executa el cas d'ús 'Gestió dades prestatgeria' i crea la prestatgeria.

Errors possibles i cursos alternatius:

1. Les dades proporcionades per l'usuari son invalides.

Cas d’ús Gestió Dades Prestatgeria

Nom: Gestió dades prestatgeria

Actor: Usuari

Comportament:

1. L’usuari ha indicat que vol crear o modificar una prestatgeria.
2. El sistema li ensenya diferents documents que ha de modificar.
3. L’usuari modifica els documents indicant al sistema quan ha acabat.
    a. Si l’ha de crear, li ensenya documents en blanc que ha de modificar afegint les dades necessàries, files, columnes…
    b. Si l’ha de modificar, li ensenya la disposició que té l’usuari per modificar-la segons correspongui.
4. El sistema comprova que totes les dades siguin adequades.
5. L’usuari decideix generar la prestatgeria.
6. El sistema genera una prestatgeria en base a les dades que l’usuari ha proporcionat.

Errors possibles i cursos alternatius:

1. Les dades de la prestatgeria estan en blanc.

Cas d’ús Modificar Prestatgeria

Nom: Modificar prestatgeria

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol modificar una prestatgeria.
2. El sistema executa el cas d'ús ‘Gestió dades prestatgeria' i modifica la prestatgeria.

Errors possibles i cursos alternatius:

1. La prestatgeria que es vol modificar no existeix.

Cas d’ús Consultar Prestatgeria

Nom: Consultar prestatgeria

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol consultar les prestatgeries que té.
2. El sistema mostra les diferents disposicions dels productes de les prestatgeries.

Errors possibles i cursos alternatius:

1. No hi ha cap prestatgeria disponible per a consultar.

Cas d’ús Importar Prestatgeria

Nom: Importar prestatgeria

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol importar una prestatgeria.
2. El sistema proporciona una opció per seleccionar el fitxer que conté les dades a importar.
3. L'usuari selecciona el fitxer.
4. El sistema importa les dades i actualitza la prestatgeria.

Errors possibles i cursos alternatius:

1. El fitxer importat està buit.

2. El fitxer importat no conté dades vàlides.

Cas d’ús Esborrar Prestatgeria

Nom: Esborrar prestatgeria

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol esborrar la prestatgeria.
2. El sistema mostra un missatge de confirmació per assegurar-se de que vol esborrar el teclat.
3. L'usuari confirma l'acció.
4. El sistema esborra la prestatgeria.

Errors possibles i cursos alternatius:

1. No hi ha prestatgeria a esborrar.

Cas d’ús Gestió Productes

Nom: Gestió productes

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol gestionar els productes.
2. El sistema ofereix opcions per crear, modificar o eliminar productes.
3. L'usuari selecciona una de les opcions disponibles.
4. El sistema executa el cas d’ús corresponent.

Errors possibles i cursos alternatius:

1. L'opció seleccionada per l'usuari no és vàlida.

Cas d’ús Crear Producte

Nom: Crear producte

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol crear un nou producte.

2. L’usuari introdueix les dades del producte (id, nom, similituds).
3. El sistema guarda el nou producte.

Errors possibles i cursos alternatius:

1. Les dades proporcionades no son valides.

Cas d’ús Modificar Producte

Nom: Modificar producte

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol modificar un producte.
2. El sistema executa el cas d'ús 'Consultar producte’ i modifica el producte.

Errors possibles i cursos alternatius:

1. El producte que es vol modificar no existeix.

Cas d’ús Consultar Producte

Nom: Consultar producte

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol consultar el producte.
2. El sistema mostra la informació del producte.
3. El sistema mostra l’opció de modificar les dades del producte.
    a. Si l’usuari vol modificar les dades del producte, s’executa el cas d'ús de ‘Gestió dades producte’.
    b. Si l’usuari només vol consultar, no es fa res.

Errors possibles i cursos alternatius:

1. No existeix el producte a consultar.

Cas d’ús Esborrar Producte

Nom: Esborrar producte

Actor: Usuari

Comportament:

1. L'usuari indica al sistema que vol eliminar un producte.
2. El sistema mostra un missatge de confirmació per assegurar-se de que vol esborrar el producte.
3. L'usuari confirma l'acció.
4. El sistema esborra el producte.

Errors possibles i cursos alternatius:

1. No existeix el producte a esborrar.

Cas d'ús Gestió Dades Producte

Nom: Gestió dades producte

Actor: Usuari

Comportament:

1. L'usuari ha indicat que vol crear o modificar un producte.
2. El sistema li ensenya diferents documents que ha de modificar.
3. L'usuari modifica els documents indicant al sistema quan ha acabat.
    a. Si l'ha de crear, li ensenya documents en blanc que ha de modificar afegint les dades necessàries, id, nom, similituds…
    b. Si l'ha de modificar, li ensenya la informació que té el producte per modificar-la segons correspongui.
4. El sistema comprova que totes les dades siguin adequades.
5. L'usuari decideix generar o guardar el producte.
6. El sistema genera un producte en base a les dades que l'usuari ha proporcionat.

Errors possibles i cursos alternatius:

1. Les dades introduïdes no son valides.

Gestió d’usuaris

Cas d’ús Crear Usuari

Nom: Crear perfil

Actor: Usuari

Comportament:

1. L'usuari indica al Sistema que vol crear un perfil.
2. L'usuari indica al Sistema les seves dades (nom usuari i contrasenya).
3. El Sistema valida i enregistra les dades de l'usuari.
4. El Sistema crea un nou perfil amb les dades indicades per l'usuari.

Errors possibles i cursos alternatius:

1. L'usuari entra dades no vàlides (nom d'usuari ja existent).
2. El Sistema no valida les dades i indica a l'usuari que no són vàlides.

Cas d’ús Iniciar Sessió

Nom: Iniciar sessió

Actor: Usuari

Comportament:

1. L'usuari indica al Sistema que vol iniciar sessió.
2. L'usuari indica al Sistema les dades per a iniciar sessió.
3. El Sistema valida i enregistra les dades.
4. El Sistema obre la sessió de l'usuari amb les dades indicades.

Errors possibles i cursos alternatius:

1. No existeix un usuari amb el nom d'usuari indicat per l'usuari.
2. La contrasenya no concorda amb el nom d'usuari indicat.

Cas d’ús Tancar Sessió

Nom: Tancar sessió

Actor: Usuari

Comportament:

1. L'usuari amb sessió d'usuari oberta indica al Sistema que vol tancar sessió.

2. El Sistema enregistra l'acció i demana a l’usuari una segona confirmació.
3. L’usuari indica al Sistema la confirmació.
4. El Sistema tanca la sessió de l'usuari.

Errors possibles i cursos alternatius:

Cas d’ús Esborrar Usuari

Nom: Esborrar perfil

Actor: Usuari

Comportament:

1. L'usuari amb sessió oberta indica al Sistema que vol esborrar el seu perfil.
2. El Sistema enregistra l'acció i demana a l’usuari una segona confirmació.
3. L’usuari indica al Sistema la confirmació.
4. El Sistema esborra el perfil de la base de dades.

Errors possibles i cursos alternatius:

1. No hi ha usuari amb el que s’hagi iniciat sessió.

Descripció de classes capa de Domini

Classe Producte

Classe i atributs

Com tots sabem, l'experiència d'anar a un supermercat es basa en un element clau: comprar un producte (o molts!). Els productes són la base de tot i en el nostre supermercat, aquests son molt més que simples objectes estàtics que el client pot comprar.

És aquí on entra en joc la classe Producte, la qual permet que cada article tingui uns atributs propis, com són un codi identificador únic (Integer) i un nom (String). Cada producte pot establir relacions de similitud amb altres productes a través d'un Map<Integer, Double>, creant una xarxa de connexions que ens permet gestionar-los de manera eficient. Degut a que un producte pot situar-se a diverses prestatgeries, també s'estableix una relació per identificar la posició d'un producte en una prestatgeria determinada mitjançant un Map<Integer, Pair<Integer, Integer>>. Gràcies a aquesta classe, els dotem d'una informació rellevant de cara a establir un sistema organitzat, intel·ligent i òptim que permet afavorir les vendes dels productes.

Funcions de la classe

- private void validarId(int id)
- private void validarSimilitud(double similitud)
- Producte(int id, String nom)
- Producte(int id, String nom, Map<Integer, Double> similituds)

Quan es crea un producte, ens assegurem que tot estigui en ordre. Així, hem de validar l'id perquè sigui superior a zero, mentre que el mapa de similituds, si es proporciona, es revisa per garantir que cap valor sigui negatiu. Si no es proporciona, inicialitzem el producte sense cap similitud associada. Per a tots els productes, en un principi, no establim una posició en cap prestatgeria fins que l'usuari decideixi afegir aquest producte en alguna prestatgeria. Si tot està correcte, el producte ja es pot construir. En cas contrari, llancem una excepció amb un missatge concret que informa a l'usuari d'un error a l'hora de realitzar aquest procés.

- public int getId()
- public String getNom()
- public Map<Integer, Double> getSimilituds()
- public Map<Integer, Pair<Integer, Integer>> getPosPrestatgeries()

- public Pair<Integer, Integer> getPosPrestatgeria(int idPres)

Amb aquestes funcions podem obtenir informació dels atributs d'un producte, ja sigui el seu identificador, el nom, les similituds amb la resta de productes, la posició d'un prestatgeria en concret o les posicions que ocupa a totes les prestatgeries on es troba.

- public void setId(int id)
- public void setNom(String nom)
- public void setSimilituds(Map<Integer, Double> similituds)
- public void setPosPrestatgeries(Map<Integer, Pair<Integer, Integer>> posPres)
- public void setPosPrestatgeria(int idPres, Pair<Integer, Integer> novaPos)

Amb aquestes funcions podem actualitzar o modificar els atributs d'un producte, tenint en compte les restriccions de l'identificador i de les similituds o posicions que es poden assignar.

- public void afegirSimilitud(int id, double similitud)
- public void modificarSimilitud(int id, double novaSimilitud)
- public double getSimilitud(int id)
- public int getIdProducteMillorSimilitud()

Aquest grup de funcions ens permet gestionar les relacions de similitud d'un producte amb operacions per afegir, modificar, obtenir aquest atribut o determinar amb quin producte té la millor similitud. En el cas que no es compleixi que el valor de la similitud sigui positiu o el producte no existeix llancem una excepció o un missatge informant a l'usuari que no s'ha pogut realitzar aquest procediment.

- public void afegirPosPrestatgeria(int idPres, Pair<Integer, Integer> pos)
- public void modificarPosPrestatgeria(int idPres, Pair<Integer, Integer> novaPos)

Amb aquestes dues funcions podem realitzar la gestió de les posicions d'un producte quan s'afegeix a una prestatgeria o es realitza un canvi general de la seva posició

- public void imprimirSimilituds()
- public String toString()

Com a altres funcionalitats, podem convertir un producte en una representació textual simple, a l'hora de necessitar una descripció general d'aquest, amb l'identificador i el nom, i també podem mostrar totes les similituds d'un producte, enumerant la seva relació amb la resta de productes.

Classe CjtProductes

Classe i atributs

Ja hem establert que en el nostre supermercat, tenim diversos productes amb informació detallada dels seus atributs. Per aquest motiu, és essencial tenir un mecanisme que els organitzi, els relacioni i els gestioni de forma eficient.

Això és el que fa precisament la classe CjtProductes, actuar com un contenidor que agrupa tots els productes en un Map<Integer, Producte>, que els emmagatzema mitjançant la seva identificació (id) com a clau i l'objecte Producte com a valor (amb tota la seva informació), el qual permet un accés ràpid i ordenat als productes. Aquest conjunt de productes és associat a un usuari específic (String) a través d'un nom i funciona com el "propietari" d'aquest conjunt de productes concret. D'aquesta manera, podem garantir una gestió dinàmica i precisa de tots els productes.

Funcions de la classe

- public CjtProductes(String usuari)

Amb aquesta función creadora associem un conjunt de productes, inicialment sense cap producte, a un usuari concret.

- public Map<Integer, Producte> getProductes(String nomUsuari)
- public Producte getProducte(int idProd)
- public Pair<Integer, Integer> getPosProducte(int idProd, int idPres)
- public void setMapProductes(Map<Integer, Producte> prods)
- public Map<Integer, Pair<Integer, Integer>> getPosPrestatgeriesProducte(int idProd)
- public Producte[] getVecProductes()

Aquestes funcions ens permeten realitzar consultes de diversos aspectes: ens pot retornar el conjunt de productes associat a un usuari concret, ens pot proporcionar un producte específic segons el seu id, quina és la seva posició en una prestatgeria determinada o directament, ens retorna totes les posicions de les prestatgeries on es troba o un vector amb tots els productes del conjunt.

- public boolean existeixProducte(int idProd)

A través d'aquesta funció, podem determinar si un producte amb un identificador concret es troba dins del conjunt de productes. Una funció senzilla però fonamental per garantir el correcte funcionament de les següents operacions, com afegir, editar o eliminar productes.

- public void afegirProducte(int id, String nom, Map<Integer, Double> similituds)

Aquesta funció ens permet afegir un nou producte, sempre que no en tingui ja un amb el mateix id. Si el producte inclou les similituds amb altres productes, també s'actualitzen les relacions bidireccionals de similitud, per tal de mantenir la informació de forma correcta entre tots els productes. Si no té similituds, es genera un error que indica a l'usuari que cal associar-ne almenys una.

- public void editarProducte(Producte p)
- public void editarIdProducte(int idProd, int nou_idProd)
- public void editarNomProducte(int idProd, String nouNom)
- public void editarPosProducte(int idProd, int idPres, Pair<Integer, Integer> novaPos)

Aquest conjunt de funcions ens permet realitzar diverses operacions d'edició dels atributs d'un producte del conjunt. Tenim opcions diferents segons el que necessitem modificar. D'una banda, podem canviar l'identificador d'un producte i actualitzar totes les relacions de similitud que altres productes tinguin amb ell, per tal de no perdre la resta de la informació del producte original. També podem modificar el nom d'un producte o actualitzar la posició d'un producte d'una prestatgeria en concret.

- public void eliminarProducte(int idProd)

Mitjançant aquesta funció, podem treure un producte del conjunt. A més, elimina qualsevol relació de similitud que altres productes tinguessin amb ell, així com en la funció anterior ens assegurem que el sistema mantigui la coherència.

- public void modificarSimilitud(int idProd1, int idProd2, double novaSimilitud)

Amb aquesta funció podem actualitzar la similitud entre dos productes. Si no hi ha una relació prèvia, un dels dos o ambdós productes no existeixen, es genera un error.

- public Map<Integer, Double> getSimilituds(int idProd)
- public double[][] getMatriuSimilituds()

Aquestes dues funcions les utilitzem, la primera, per obtenir les similituds d'un producte concret amb la resta, i la segona, per generar una matriu que representa totes les similituds entre els productes del conjunt, una operació molt útil per a realitzar anàlisis posteriorment i que farem servir com a entrada en els algoritmes per tal de determinar la millor disposició dels productes en la prestatgeria.

- public Producte[] getProductesPerIds(Set<Integer> idsProds)

- public double[][] getMatriuSimilitudsPerIds(Producte[] idsProds)

Quan només volem obtenir un vector de productes o la matriu de similituds d'uns productes determinats, fem servir aquestes funcions.

- public Map<Integer, Producte> listToProductes(List<String> producteJsonList)
- public List<String> productesToList(Map<Integer, Producte> productes)
- public Map<String, Map<String, String>> llistarProductesUsuari()

Amb aquestes tres funcions podem convertir i gestionar dades relacionades amb productes de manera flexible entre representacions en Java i formats serialitzables com JSON, el que ens facilita la seva manipulació i emmagatzematge.

Classe Usuari

Classe i atributs

La classe Usuari representa els usuaris de l'aplicació, gestionant la seva informació personal i permetent accedir a les seves credencials. Un usuari està definit pels següents atributs. Un username, un atribut de tipus String que emmagatzema el nom d'usuari i una contrasenya, que es un atribut de tipus String que guarda la contrasenya de l'usuari.

Funcions de la classe

- public Usuari (String name, String contra)

Aquesta creadora inicialitza un objecte Usuari amb els atributs username i contrasenya. Rep per paràmetre el nom d'usuari (name) i la contrasenya (contra), i els assigna als atributs corresponents.

- public Usuari getUsuari()
- public String getUsername()
- public String getContrasenya()

Amb aquestes funcions podem obtenir informació dels atributs d'un usuari, ja sigui el seu username, la contrasenya, o l'objecte Usuari.

- public void canviaUsername(String username)
- public void canviaContrasenya(String contra)

La primera funció ens permet canviar el username de l'usuari i la segona funció permet canviar la contrasenya de l'usuari.

Classe Prestatgeria

Classe i atributs

La nostra classe **Prestatgeria** funciona com una estructura utilitzada per simular l'organització de productes en un supermercat, amb l'objectiu d'obtenir una distribució òptima i funcional. Cada prestatgeria disposa d'una identificació única mitjançant un atribut id i guarda informació sobre el seu nom, el nombre de files i columnes que la conformen, i el conjunt de productes que conté. A més, utilitza una matriu (layout) per representar la seva disposició, on cada compartiment emmagatzema un producte específic, i una matriu (disp), que en lloc

Funcions de la classe

- public Prestatgeria(int id, String nom, int filas, int columnas, Set<Integer> productes)

Aquest constructor permet crear una nova prestatgeria amb un identificador, un nom, dimensions definides per files i columnes, i un conjunt de productes associats. En inicialitzar-se, el layout queda buit.

- public Producte[][] getLayout()

Retorna la disposició actual dels productes dins la prestatgeria en format de matriu, on cada compartiment correspon a una posició.

- public int getId()

Retorna l'identificador únic de la prestatgeria.

- public String getNom()

Retorna el nom assignat a la prestatgeria.

- public int getNumFilas()

Retorna el nombre de files de la prestatgeria.

- public int getNumColumnas()

Retorna el nombre de columnes de la prestatgeria.

- public Set<Integer> getProductes()

Retorna el conjunt d'identificadors dels productes que formen part de la prestatgeria.

- public Producte[] getPrestatge(int indexFila)

Retorna una fila concreta del layout, representant els productes que hi ha en aquesta fila.

- public List<List<Pair<String, Integer>>> getDisp()

Retorna una representació simplificada del layout actual de la prestatgeria, emmagatzemant el nom i l'identificador de cada producte.

- public void setId(int id)

Permet modificar l'identificador de la prestatgeria, assegurant que sigui un valor vàlid.

- public void setLayout(Producte[][] disposicio)

Assigna una nova disposició a la prestatgeria, actualitzant les files i columnes en funció del nou layout. També actualitza l'atribut disp per reflectir el nou layout i la informació sobre la ubicació dels productes.

- public void intercanviarDosProductes(int filaProd1, int colProd1, int filaProd2, int colProd2)

Permet intercanviar la posició de dos productes dins del layout, assegurant que la coherència del sistema es manté intacta. També actualitza disp per reflectir el canvi.

- public void esborrarPrestatgeria()

Buida completament la prestatgeria, eliminant tots els productes que hi ha col·locats i deixant-la preparada per a futures configuracions. També es buida l'atribut disp.

Classe CtrlDomini

Classe i atributs

La classe CtrlDomini actua com un controlador centralitzat que connecta usuaris, productes i prestatgeries, i és el responsable de garantir que tot el sistema funcioni correctament i el nostre supermercat sigui el més òptim possible (així podrem vendre més productes!). Aquesta classe implementa el patró Singleton, assegurant que tan sols hi hagi un únic "director" que gestioni totes les operacions a la vegada.

Funcions de la classe

- public CtrlDomini()
- public void inicialitzarCtrlDomini()

Quan es crea un objecte de CtrlDomini, aquest crida al segon mètode per preparar els conjunts d'usuaris i altres elements essencials.

- public static CtrlDomini getInstance()

Fent ús d'aquest mètode podem garantir, com ja hem mencionat anteriorment, que hi hagi una sola instancia d'aquesta classe, de forma que evitem qualsevol tipus de desajust en la gestió del sistema.

- public void iniciarSessio(String username, String pwd)
- public void crearUsuari(String name, String pwd)
- public void canviarUsuari(String nomUsuari, String pwd)

El nostre controlador ens permet gestionar els usuaris d'una forma ben estructurada i senzilla de manejar, on podem controlar l'autenticació d'aquest o la possibilitat d'alternar entre usuaris. En el cas que un usuari no existeixi, el sistema crea un compte nou. També podem registrar nous usuaris amb un nou i una contrasenya corresponents i iniciar sessió en aquell mateix moment.

- public Producte[] llistarProductesUsuari()
- public Producte[] llistarPrestatgeriaUsuari()
- public Set<String> llistarUsuaris()

Amb aquests mètodes podem veure el conjunt de productes disponibles que té l'usuari actual, el contingut que presenta una prestatgeria o els usuaris creats fins ara pertanyents al cjtUsuaris.

- public void crearProducte(int id, String nom, Map<Integer, Double> similituds, Boolean bruteForce)
- public void crearPrestatgeria(Boolean bruteForce)
- public void modificarProducte(Integer idProdActual1, Integer nouId, String nouNom)
- public void modificarSimilituds(Integer idProdActual1, Integer idProdActual2, double novaSim, Boolean bruteForce)
- public void modificarPrestatgeria(int pos1, int pos2)
- public void esborrarProducte(int id, Boolean bruteForce)
- public void esborrarPrestatgeria()

Aquest conjunt de funcions ens permet gestionar els casos d'ús dels productes i de les prestatgeries, realitzant operacions de creació, modificació o eliminació corresponents a cada objecte. Donem la possibilitat d'insertar nous productes al sistema amb la seva informació i de generar una prestatgeria per tal de acomodar-los, així com d'editar qualsevol dels atributs d'un producte, ja sigui l'id, el nom, la posició en la que es troba o, fins i tot, podem canviar la similitud que hi ha entre dos productes (aquesta es modificarà correctament si es tracta d'un número vàlid entre 0 i 1). Així mateix, podem eliminar un producte i actualitzar la prestatgeria.

Al crear una prestatgeria, utilitzem un algorisme de generació per disposar els productes segons les seves similituds. Donem l'opció a l'usuari de poder triar entre un enfocament de força bruta amb l'algoritme de BruteForce o un algorisme d'aproximació de DosAproximacio. Addicionalment, es pot reorganitzar manualment la prestatgeria intercanviant la posició de dos productes o buida completament la prestatgeria.

- public String getUsuariActual()

Aquest mètode és un mètode directe per tal d'obtenir el nom d'usuari de l'usuari que està connectat al sistema en un moment donat.

- public void esborrarUsuari()

Si un usuari ja no vol estar en el sistema, hi ha l'opció de poder eliminar-lo completament.

- public void tancarSessio()

Amb aquesta funció podem desconnectar l'usuari actual del sistema i restablir l'objecte UsuariActual, és a dir, restablim l'estat del controlador perquè no hi hagi cap usuari actiu.

Classe GeneradorSolucio

Classe i atributs

La classe GeneradorSolucio és una interfície que defineix els acords que les classes dels algoritmes hauran de seguir. Aquesta interfície proporciona els mètodes necessaris per generar i obtenir els resultats d'un layout de productes, així com per obtenir la millor similitud calculada entre els productes.

Funcions de la classe

La primera funció crida a l'algoritme BruteForce o DosAproximacio segons la configuració desitjada, i retorna la millor disposició dels productes. Les altres dues funcions proporcionen el resultat un cop l'algoritme ha estat executat, permetent obtenir el resultat sense necessitat de cridar de nou a l'algoritme, així com la millor similitud total obtinguda per aquest resultat.

- Producte[] generarLayout();
- Producte[] getResultat();
- double getMillorSimilitud();

Classe BruteForce

Classe i atributs

La classe BruteForce implementa un algorisme de força bruta per resoldre problemes d'optimització en la disposició de productes. Aquesta classe forma part del paquet Domini i implementa la interfície GeneradorSolucio. El seu objectiu principal és determinar la millor configuració de productes que maximitzi la similitud entre productes. Per aconseguir-ho, utilitza una matriu de similituds, matSimilituds, que defineix les relacions entre els productes; un vector de ints que representen els identificadors dels productes, vecProductes, que conté els elements a ordenar; i tres atributs principals per emmagatzemar els resultats: millorSimilitud, que guarda el valor màxim de similitud trobat, vecResultat, que conté la configuració òptima de productes corresponent a aquesta similitud, i matResultat, que també conté la configuració òptima de productes corresponent a aquesta similitud però en forma de matriu. A més, per construir aquesta matriu s'han afegit dos atributs més: columnes i files, per saber les dimensions de la prestatgeria (la matResultat).

Funcions de la classe

- BigInteger factorial(int n);

Aquesta funció calcula el nombre de permutacions que s'hauran de fer. L'ús de BigInteger garanteix que es puguin tractar números grans sense desbordaments, ja que el nombre de permutacions creix ràpidament amb el nombre de productes.

- double calcularSimilitudTotal(int[] vecActualProd);

Aquesta funció calcula la similitud total d'una configuració específica de productes tenint en compte la geometria circular de la prestatgeria.

- void swap(int[] arr, int i, int j);
- void permutacions(int[] vProd, int L, int R, BigInteger[] numPermutacions);

Amb aquestes funcions calculem totes les permutacions possibles d'un subconjunt de productes i, per cada configuració, cridem a calcularSimilitudTotal i ens guardem la disposició si aquesta és la millor trobada fins al moment. S'ha inclòs un comptador (numPermutacions) per limitar el nombre màxim de permutacions.

- int [][] generarLayout();

Aquesta funció inicia el procés per trobar la millor configuració de productes. Utilitza la funció de permutacions per explorar totes les opcions i retorna la configuració que maximitza la similitud. El seu disseny garanteix que sigui fàcil d'utilitzar des de fora de la classe, encapsulant tota la complexitat interna.

- int [][] getResultat();
- double getMillorSimilitud();

Aquestes funcions permeten accedir directament al resultat òptim un cop completat el càlcul.

Classe DosAproximacio

Classe i atributs

La classe DosAproximacio és una implementació de l'algorisme d'aproximació per resoldre un problema de distribució o ordenació de productes, basant-se en les similituds entre ells. A través d'un conjunt d'operacions, la classe genera una configuració òptima de productes per maximitzar la similaritat total. Aquesta classe disposa d'un conjunt d'atributs que defineixen el seu comportament i l'estructura interna.

- **private int n:** Representa el nombre total de productes involucrats en el procés. Es calcula a partir de la longitud de la matriu de similituds,matS, i indica el nombre de nodes (productes) amb què es treballa en els càlculs.
- **private List<Aresta> llista_arestes:** És una llista que emmagatzema les arestes del graf de similituds entre els productes. Cada aresta és representada per un objecte de la classe Aresta que conté la informació sobre els dos productes connectats i la seva similitud corresponent. Aquesta llista es construeix a partir de la matriu de similituds inicial.

- **private int[] pare:** Un array utilitzat per gestionar les unions de conjunts al fer l'algoritme de Kruskal. Cada posició d'aquest array conté el pare d'un node dins d'un conjunt, facilitant les operacions de "find" i "uneix" que es realitzen a l'algorisme.

- **private int[] mida:** Un altre array utilitzat per optimitzar les operacions del Kruskal. En aquest cas, emmagatzema la mida dels conjunts, de manera que l'unió de conjunts es realitzi sempre de la manera més eficient, unint el conjunt més petit amb el més gran.

- **private Producte[] prods:** Un array que conté els productes involucrats en el càlcul. Cada producte es representa amb un objecte de la classe Producte. Aquest array s'usa per associar els productes amb les seves posicions en els càlculs de similitud.

- **private double sumaSimilitud:** Aquesta variable guarda el valor de la millor suma de similituds trobada durant l'execució de l'algorisme. Inicialment es defineix com a 0, i es va actualitzant a mesura que es calculen diferents disposicions possibles dels productes.

- **private Producte[] res_productes**: Un array que emmagatzema la millor configuració de productes trobada segons la suma de similituds. Aquest array es genera com a resultat final de l'algorisme i representa l'ordre òptim dels productes.

- **private double[][] matS:** És la matriu de similituds, on cada element [i][j] representa la similitud entre el producte [i] del vetor *prods* i el producte [j]. Aquesta matriu és essencial per calcular la qualitat de les diferents configuracions de productes.

Funcions de la classe

La funció pública principal generarLayout() busca generar un layout òptim per a una prestatgeria de productes circular, maximitzant la suma de similituds entre productes. Per fer-ho, comença construint un Maximum Spanning Tree utilitzant l'algoritme de Kruskal.

Un cop obtingut aquest MST, es converteix en un cicle eulerià que connecta totes les arestes de l'mst, mantenint una circularitat a la prestatgeria. Aquest cicle es crea recorrent les connexions del graf amb la funció findEuleria.

Finalment, per eliminar els nodes repetits de forma òptima, s'utilitza la funció calculaSuma per provar totes les configuracions possibles eliminant duplicats i calculant la suma de similituds per a cada opció. Si es troba una configuració millor, es guarda com a resultat final. Finalment, es retorna el vector de productes que maximitza la similitud acumulada.

La funció pública getMillorSimilitud retorna la similitud del vector resultat, mentres que la funció pública getResultat retorna el vector de Productes resultat.

Classe Pair

Classe i atributs

Aquesta classe és una implementació genèrica que permet emmagatzemar i gestionar una parella de valors, composta per una clau i un valor. Aquesta estructura ens resulta molt útil per representar relacions entre dues dades de diferents o iguals tipus.

Funcions de la classe

- public Pair(T1 clau, T2 valor)
- public T1 getClau()
- public void setClau(T1 clau)
- public String clauToString()
- public T2 getValor()
- public void setValor(T2 valor)
- public String valorToString()
- public String toString()

A través d'aquestes funcions, inicialitzem una instància de la classe amb els valors proporcionats amb qualsevol combinació de tipus, també tenim els getters i setters dels atributs de la classe i unes funcions que els converteixen a una cadena.

Classe CjtPrestatgeries

Classe i atributs

La classe CjtPrestatgeries representa un conjunt de prestatgeries associades a un usuari. Aquesta classe permet gestionar tota la informació rellevant sobre les prestatgeries, incloent la seva creació, modificació, consulta, i eliminació. A més, actua com a intermediària entre la capa de domini i la capa de persistencia, facilitant les conversions entre objectes Java i representacions JSON.

La classe CjtPrestatgeries té dos atributs principals: user, que identifica l'usuari associat al conjunt de prestatgeries, i map_prest, un mapa que enllaça els identificadors únics de les

prestatgeries amb les seves instàncies per a un accés ràpid i organitzat. Aquesta estructura permet gestionar múltiples prestatgeries de forma eficient.

Funcions de la classe

- getConjPrestatges(String userID): Map<Integer, Prestatgeria>

Retorna el conjunt de prestatgeries associat a un usuari específic si l'identificador coincideix amb el nom de l'usuari actual. Si no coincideix, retorna null.

- getPrestatgeria(int prestatgeID): Prestatgeria

Retorna l'objecte Prestatgeria associat a un identificador únic. Si no existeix cap prestatgeria amb aquest ID, mostra un missatge d'error i retorna null.

- getLayout(int prestatgeID): Producte[][]

Retorna la disposició (layout) dels productes dins d'una prestatgeria específica, representada com una matriu.

- getPrestatge(int prestatgeID, int indexFila): Producte[]

Retorna una fila concreta del layout d'una prestatgeria, corresponent als productes col·locats en aquesta fila.

- getProductes(Integer prestatgeID): Set<Integer>

Retorna el conjunt d'identificadors de productes que formen part d'una prestatgeria específica.

- getNumCols(Integer prestatgeID): int

Retorna el nombre de columnes d'una prestatgeria específica.

- getIdsPrestatgeriesSame(Map<Integer, Pair<Integer, Integer>> parella1, Map<Integer, Pair<Integer, Integer>> parella2): Integer[]

Identifica les prestatgeries que comparteixen productes en comú entre dos mapes. (Funció pendent d'implementació).

- crearPrestatgeria(int id, String nom, int filas, int columnas, Set<Integer> setProds)

Crea una nova prestatgeria amb un identificador únic, un nom, unes dimensions definides per files i columnes, i un conjunt inicial de productes. Si ja existeix una prestatgeria amb el mateix ID, mostra un missatge d'error.

- editarPrestatgeria(int id, String nom, int filas, int columnas, Set<Integer> setP)

Modifica una prestatgeria existent canviant el nom, les dimensions, i els productes associats. Si no existeix, mostra un missatge d'error.

- setLayout(Producte[][] mat, Integer prestatgeID)

Assigna un nou layout a una prestatgeria existent. Aquesta funció actualitza la disposició interna dels productes.

- intercanviarDosProductes(int prestatgeID, int filaProd1, int colProd1, int filaProd2, int colProd2)

Permet intercanviar la posició de dos productes dins el layout d'una prestatgeria.

- esborrarPrestatgeria(int prestatgeID)

Elimina completament el contingut d'una prestatgeria i deixa l'estructura buida.

- eliminarPrestatgeria(int id)

Elimina completament una prestatgeria del mapa si existeix.

Descripció de classes capa de Persistencia

Per gestionar les dades en el nostre projecte, hem decidit utilitzar arxius JSON, ja que ens ha semblat simple i fàcil d'utilitzar.

Classe CtrlPersistencia

Classe i atributs

La classe CtrlPersistencia és la classe encarregada de comunicar-se amb la capa de domini, evitant així que hi hagi acoblament entre capes. Fa les crides necessàries als gestors de la capa.
Aquesta classe no té no atributs, ja que no li calen.

Funcions de la classe

Per la gestió dels usuaris tenim les següents funcions:

- boolean existeixUsuari(String user)
- boolean verificarContrasenya(String user, String password)

Aquestes funcions verifiquen, respectivament, si un usuari existeix al sistema i si la contrasenya d'un usuari en concret és correcta. Es podria haver implementat una única funció per comprovar si l'usuari i la seva contrasenya són correctes simultàniament, però es va decidir separar-les per oferir més claredat a l'usuari. Aquesta separació permet informar si l'error es troba en el nom d'usuari o en la contrasenya.

- void afegirUsuari(String user, String password)
- void eliminarUsuari(String user)

Afegeixen un nou usuari al sistema amb una contrasenya i eliminen un usuari, respectivament. Com que s'han d'eliminar totes les dades associades a l'usuari, s'utilitzen els tres gestors.

- void canviarContrasenya(String user, String novaPassword)

Aquesta funció permet que l'usuari canviï la seva contrasenya. Aquesta funcionalitat es va considerar important, però vam identificar un problema si s'implementava a la fase d'inici de sessió, com sol ocórrer en moltes aplicacions, quan es pregunta si s'ha oblidat la contrasenya. Com que no disposem d'un mètode d'autenticació addicional més enllà del nom d'usuari, això podria permetre a qualsevol persona canviar la contrasenya simplement coneixent el nom d'usuari. Per evitar aquest risc, hem decidit que aquesta opció només estigui disponible un cop l'usuari ja s'ha autenticat correctament.

Per la gestió de les prestatgeries tenim les següents funcions:

- List<String> importarPrestatgeria(String usuari)
- void guardarPrestatgeries(List<String> prestatgeries, String usuari)
- List<String> importarFitxerPrestatgeria(String path)

Aquestes funcions permeten carregar les prestatgeries associades a un usuari, retornant una llista amb la seva informació; guardar les prestatgeries d'un usuari; i importar prestatgeries des d'un fitxer, respectivament.
Per la gestió dels productes tenim les següents funcions:

- List<String> importarProductes(String usuari)
- void guardarProductes(List<String> productes, String usuari)

- List<String> importarFitxerProductes(String path)

Aquestes funcions permeten carregar els productes associats a un usuari, retornant una llista amb la seva informació; guardar els productes d'un usuari; i importar productes des d'un fitxer, respectivament.

Classe GestorCjtPrestatgeries

Classe i atributs

La classe GestorCjtPrestatgeries és responsable de gestionar les operacions relacionades amb la persistència de les prestatgeries d'un usuari, incloent la importació, exportació i eliminació de les seves dades. Les prestatgeries es guarden en fitxers JSON associats a cada usuari, situats en una ruta específica del sistema. Aquesta classe també permet importar prestatgeries des de fitxers externs i treballar amb el seu contingut de manera estructurada.

Funcions de la classe

- importarPrestatgeries(String usuari)

Importa totes les prestatgeries associades a un usuari des d'un fitxer JSON. Llegeix el fitxer i converteix cada prestatgeria en un objecte JSON que s'emmagatzema com una cadena en una llista. Cada prestatgeria conté informació com l'identificador, el nom, les dimensions i el layout dels productes.

- guardarPrestatgeries(List<String> prestatgeries, String usuari)

Desa la llista de prestatgeries proporcionada al fitxer JSON associat a l'usuari. Converteix les prestatgeries en objectes JSON i les estructura correctament per escriure-les al fitxer. Si les prestatgeries són nul·les, elimina el fitxer associat a l'usuari.

- esborrarPrestatgeriesUsuari(String usuari)

Esborra el fitxer JSON associat a un usuari específic. Si el fitxer existeix, intenta eliminar-lo i retorna un valor booleà que indica si l'operació ha estat exitosa.

- getRuta(String usuari)

Genera la ruta del fitxer JSON corresponent a l'usuari, assegurant-se que el directori existeix i creant el fitxer si no és present. Aquesta funció garanteix que la infraestructura de persistència estigui preparada abans de qualsevol operació.

- importarFitxerPrestatgeria(String path)

Importa els identificadors d'una prestatgeria des d'un fitxer extern. Verifica que el fitxer conté només una línia amb IDs numèrics positius separats per espais. Retorna una llista amb els IDs vàlids si el fitxer és correcte; en cas contrari, retorna null i informa dels errors detectats

Classe GestorCjtProductes

Classe i atributs

El gestor d'aquesta classe s'encarrega de gestionar la importació, exportació i manipulació de conjunts de productes des d'arxius en format JSON. Serveix com a intermediari entre la representació de dades en el nostre sistema i el seu emmagatzematge a la base de dades. Com a tal, no en té d'atributs propis, degut a que totes les funcionalitats es gestionen a través de mètodes estàtics, la qual ens permet el seu ús sense necessitat de crear una instància de la classe.

Funcions de la classe

- public static List<String> importarProductes(String usuari)

Aquesta funció importa productes associats a un usuari des d'un arxiu JSON situat en el sistema de fitxers.

- public static void guardarProductes(List<String> productes, String usuari)

Amb aquesta funció, podem desar una llista de productes en format String associats a un usuari en un arxiu JSON. En el cas que no hi hagi productes per desar, elimina l'arxiu amb l'ajuda de la següent funció.

- public static boolean esborrarProductes(String usuari)

Quan hem d'esborrar l'arxiu JSON associat a un usuari, fem servir aquesta funció.

- private static String getRuta(String usuari)

Aquesta funció retorna la ruta on es troba l'arxiu JSON que conté les dades dels productes de l'usuari. Si la carpeta o l'arxiu no existeixen, els crea automàticament.

- public static List<String> importarFitxerProducte(String path)

En el cas de voler importar productes des d'un fitxer de text, fem servir aquesta funció.

Classe GestorUsuaris

Classe i atributs

Aquesta classe s'encarrega de gestionar els usuaris. Permet realitzar diverses operacions sobre els usuaris del sistema, incloent l'afegiment, eliminació, comprovació d'existència, verificació de contrasenyes i la modificació de la contrasenya dels usuaris.
No té atributs explícits, ja que treballa directament amb el fitxer JSON per a gestionar les dades.

Funcions de la classe

- void afegirUsuari(String user, String password)

Aquesta funció afegeix un nou usuari a l'arxiu JSON. Si l'arxiu ja conté dades, carrega els usuaris existents, afegeix un nou objecte JSON amb el nom d'usuari i la contrasenya, i desa la nova llista d'usuaris al fitxer.

- void eliminarUsuari(String user)

Aquesta funció elimina un usuari de l'arxiu JSON. Busca l'usuari pel seu nom d'usuari i el retira de la llista. Després, escriu la nova llista d'usuaris al fitxer.

- boolean existeixUsuari(String user)

Comprova si un usuari ja existeix en l'arxiu JSON. Si troba un usuari amb el nom especificat

- boolean verificarContrasenya(String user, String password)

Aquesta funció verifica si la contrasenya proporcionada coincideix amb la contrasenya de l'usuari especificat.

- void canviarContrasenya(String user, String novaPassword)

Aquesta funció permet canviar la contrasenya d'un usuari. Primer, elimina l'usuari amb la contrasenya antiga i després afegeix l'usuari amb la nova contrasenya.

- String getRuta()

Aquesta funció retorna la ruta on es troba l'arxiu JSON que conté les dades dels usuaris. Si la carpeta o l'arxiu no existeixen, els crea automàticament.

Descripció classes capa de Presentació

Classe CtrlPresentacio

Classe i atributs

Aquesta classe actua com a intermediari entre les vistes i el controlador de domini. Per aquest motiu, té com a atribut el controlador de domini (private CtrlDomini ctrlDomini), que s'utilitza per interactuar amb aquesta capa, i una referència al JFrame actual.

Funcions de la classe

- public CtrlPresentacio()

Mitjançant aquesta funció constructora, s'inicialitza el controlador de domini i es mostra el menú inicial.

- public void mostrarMenuInici()
- public void mostrarMenuUsuari()

Amb aquestes funcions es realitza la gestió de les vistes del menú inicial i del menú d'usuari.

- public Map<String, Map<String, String>> mostrarProductes()
- public Map<String, Map<String, String>> mostrarPrestatgeries()

Aquestes són funcions amb les que podem obtenir la llista de productes i de prestatgeries de l'usuari.

- private Map<Integer, Double> convertirStringAMap(String input)
- private Set<Integer> convertirStringASet(String input)

Es fan servir aquestes funcions auxiliars per tal de convertir una cadena en un mapa d'enters i doubles o en un conjunt d'enters.

- public void crearProducte(String idProd, String nomProd, String similituds)
- public void esborrarProducte(String idProd)

- public String getNomProducte(String idProd)
- public Map<Integer, Double> getSimilitudsProducte(String idProd)
- public Map<Integer, Pair<Integer, Integer>> getPosPrestatgeriesProducte(String idProd)
- public void editarIdProducte(String idActual, String nouId)
- public void editarNomProducte(String idProd, String nouNom)
- public void modificarSimilitudProductes(String idProd1, String idProd2, String novaSimilitud, String bf)
- public void modificarPosicioProductes(String idPres, String idProd1, String idProd2)
- public boolean existeixProducteId(String idProd)

Amb aquest conjunt de funcions, podem gestionar els productes, tant la creació, com l'eliminació, consultar, editar i validar informació sobre aquests.

- public void crearPrestatgeria(String idPres, String nom, String numCols, String productes, String bf)
- public void esborrarPrestatgeria(String idPres)
- public boolean existeixPrestatgeriaId(String idPres)

Amb aquest conjunt de funcions, fem el mateix però en quant a les prestatgeries.

- public boolean validarLogin(String username, String pwd)
- public void realizarLogin(String username, String pwd)

Per tal de validar les credencials d'un usuari o iniciar sessió, fem servir aquestes funcions.

- public boolean existeixUsuari(String username)
- public String nomUsuariActual()
- public void registrarUsuari(String username, String pwd)
- public void esborrarUsuari()
- public void canviarContrasenya(String username, String pwd)

Amb aquestes funcions podem registrar un nou usuari, comprovar si existeix un, obtenir el nom de l'usuari actual, eliminar l'usuari o, fins i tot, canviar la contrasenya.

- public void logout()

Finalment, quan l'usuari vulgui tancar la sessió, fem servir aquesta funció.

Classe VistaMenuInici

Classe i atributs

Funcions de la classe

Classe VistaLogIn

Classe i atributs

Aquesta classe s'encarrega de crear la finestra de la interfície gràfica de la pàgina de login del nostre programa. Permet als usuaris introduir el seu nom d'usuari i contrasenya per autenticar-se i accedir a l'aplicació. Aquesta classe implementa una sèrie de mecanismes per validar la informació de l'usuari abans de permetre el registre a l'aplicació.
Aquesta vista consta d'uns quants components visuals, com ara camps de text per a l'usuari i la contrasenya, així com un conjunt de botons i etiquetes informatives. El disseny inclou logotips i elements decoratius per millorar l'experiència de l'usuari.

Funcions de la classe

La classe inclou un conjunt de mecanismes per validar les dades introduïdes per l'usuari:

- contrasenyaMousePressed(evt)
- usuariMousePressed(evt)

S'executa quan l'usuari fa clic al camp de contrasenya o al camp del username de l'usuari. Si el text per defecte està present ("********" o "Introdueix el teu nom d'usuari") , el borra i canvia el color a negre. Si els altres camps estan buits deixa el text per defecte en color gris.

- entrarMouseClicked(evt)

S'executa quan l'usuari fa clic al botó "Entrar". Verifica si les contrasenyes coincideixen i si els camps d'usuari i contrasenya estan complets. Si no es compleixen aquestes condicions, mostra un missatge d'error.

- registreMouseClicked(evt)

Aquesta funció s'executa quan es fa clic al botó de registre. Canvia la vista de la pantalla per mostrar la pantalla de registre, canviant el panell de "LogIn" pel de "SignUp".

Classe VistaSignUp

Classe i atributs

Aquesta classe s'encarrega de crear la finestra de la interfície gràfica de la pàgina on enregistrem un nou usuari en el nostre programa. Aquesta vista recull el nom d'usuari i la contrasenya, i posteriorment valida la informació introduïda per assegurar-se que sigui correcta i compleixi amb els requisits del sistema.
Els components visuals inclouen camps de text per al nom d'usuari i les contrasenyes, així com etiquetes informatives que guien l'usuari en el procés de registre. El disseny també inclou elements visuals com logotips per millorar l'aparença de la interfície.

Funcions de la classe

La classe inclou un conjunt de mecanismes per validar les dades introduïdes per l'usuari:

- contrasenyaMousePressed(evt)
- contrasenya2MousePressed(evt)
- usuariMousePressed(evt)

S'executa quan l'usuari fa clic al camp de contrasenya, contrasenya2 (què és la confirmació de la contrasenya) o al camp del username de l'usuari. Si el text per defecte està present ("********" o "Introdueix el teu nom d'usuari") , el borra i canvia el color a negre. Si els altres camps estan buits deixa el text per defecte en color gris.

- entrarMouseClicked(evt)

S'executa quan l'usuari fa clic al botó "Entrar". Verifica si les contrasenyes coincideixen i si els camps d'usuari i contrasenya estan complets. Si no es compleixen aquestes condicions, mostra un missatge d'error.

- enrereMouseClicked(evt)

Aquesta funció s'executa quan es fa clic al botó d'enrere. Canvia la vista de la pantalla (panell "Parent") per tornar a la pantalla de login, canviant el panell de "SignUp" pel de "LogIn".

Classe VistaMenuUsuari

Classe i atributs

Funcions de la classe

Classe VistaProducte

Classe i atributs

Aquesta classe és una interfície gràfica dissenyada per gestionar productes dins d'una aplicació, proporcionant diverses funcionalitats com la consulta, edició i eliminació d'aquests productes. Aquesta vista fa ús d'un layout del tipus CardLayout, que permet alternar entre diferents panells per organitzar la informació i les opcions d'interacció amb els productes.

Pel que fa als atributs principals, la classe inclou una instància del controlador de presentació, que actua com a controlador per coordinar les accions de la vista amb la lògica de negoci del programa. També defineix un CardLayout, assignat al contenidor cardPanel, que serveix com a panell principal per gestionar els diferents subpanells de la interfície. Addicionalment, hi ha un panell específic anomenat panelGrid, destinat a organitzar elements visuals en un format de graella quan sigui necessari.

Els panells de la classe tenen funcions ben diferenciades. El ‘llistaPanel’ ofereix una vista global de tots els productes disponibles en el sistema, presentant-los en una llista interactiva. Aquesta llista permet als usuaris seleccionar productes individuals per dur a terme accions com editar-los o esborrar-los. Per millorar l'experiència d'usuari, el panell inclou una barra de desplaçament i botons específics per gestionar les accions sobre els productes seleccionats.

D'altra banda, el ‘infoPanel’ està dissenyat per mostrar informació detallada d'un producte seleccionat. Inclou etiquetes i camps de text que mostren atributs com l'ID i el nom del producte, a més d'oferir eines per consultar similituds amb altres productes mitjançant

botons específics. Aquest panell no permet edicions directes, ja que el seu objectiu principal és la consulta d'informació.

Finalment, el ‘editarPanel’ proporciona un entorn d'edició on l'usuari pot modificar els atributs del producte, com ara el seu identificador, nom o paràmetres de similitud. Aquest panell incorpora camps de text editables i un botó de desament que permet aplicar els canvis realitzats al sistema. La funcionalitat del panell està orientada a la gestió dels atributs del producte de manera segura i controlada.

Funcions de la classe

- VistaProducte()

Inicialitza la vista amb els atributs bàsics, configura el CardLayout i estableix la mida de la interfície.

- initComponents()

És responsable de crear i organitzar tots els components gràfics de la classe, configurant els panells, assignant-los al contenidor principal cardPanel.

- botoConsultarActionPerformed(evt)

Mostra el panell infoPanel on es consulta informació detallada d’un producte.

- botoEditarActionPerformed(evt)

Canvia al panell editarPanel per editar la informació d’un producte.

- botoConsultarSimilitudActionPerformed(evt)

Gestiona la consulta de similituds entre el producte seleccionat i altres productes.

- botoEsborrarMouseClicked(evt)

Permet esborrar un producte seleccionat amb una confirmació prèvia per part de l’usuari, ja que si s’elimina aquest producte, s’eliminaran totes les prestatgeries on es troba aquell producte.

- botoSortirActionPerformed(evt)

Torna a una vista anterior (la vista general amb la llista de productes de l’usuari).

- botoGuardarActionPerformed(evt)

Desa els canvis efectuats en els camps d'edició i utilitza el controlador per actualitzar la informació del producte al sistema.

Classe VistaCrearProducte

Classe i atributs

Aquesta classe s'encarrega de dissenyar la finestra de la interfície gràfica per a la creació de nous productes dins del nostre programa. Permet als usuaris introduir les dades necessàries per generar un nou producte, com ara l'identificador, el nom i les similituds amb altres productes. Aquesta classe implementa diversos mecanismes per validar la informació proporcionada abans de permetre la creació del producte.

La vista inclou diversos components visuals, com camps de text per a l'identificador i el nom del producte, una àrea de text per a les similituds, així com un conjunt de botons i etiquetes informatives. El disseny incorpora elements decoratius, com una imatge representativa, per millorar l'experiència de l'usuari.

La finestra presenta un disseny estructurat que inclou un camp per introduir l'identificador del producte, que és validat abans de continuar, un panell d'informació addicional, on es poden introduir el nom del producte i la llista de similituds amb altres productes, botons per interactuar, com validar l'ID, crear el producte o importar dades d'un fitxer de text.

Funcions de la classe

- VistaCrearProducte()

Es tracta del constructor que inicialitza els components visuals, configura l'aspecte inicial de la finestra i estableix la connexió amb el controlador principal

- initComponents()

Genera i organitza tots els elements de la interfície gràfica, com camps de text, botons i panells, configurant els camps perquè mostrin un text predeterminat.

- botoSortirMouseClicked()

Gestiona l'acció que es produeix quan l'usuari fa clic al botó "Sortir", és a dir, torna a la vista general on es mostren tots els productes de l'usuari.

- textNomMousePressed()
- textIdMousePressed()

- textAreaSimsMousePressed()

Gestionen esdeveniments quan l'usuari interactua amb els camps de text, com esborrar textos predeterminats o restaurar altres camps.

- botoValidarIdMousePressed()

Verifica que el camp no estigui buit i comprova que el valor introduït sigui numèric i positiu. Després es connecta amb el controlador per assegurar que l'ID no està duplicat al sistema. Si l'ID és vàlid, mostra un missatge visual positiu. Si no és vàlid, apareix un missatge d'error explicant la causa (duplicat, buit o invàlid).

- botoCrearMouseClicked()

Comprova que tots els camps obligatoris estan emplenats correctament i valida el format de les dades introduïdes. Si és així, mostra un missatge de confirmació indicant que el producte s'ha creat correctament. En cas d'error, proporciona un missatge explicatiu amb instruccions per corregir-lo.

- bgPanelMouseClicked()

Comprova si algun camp està buit i, si és així, torna a mostrar el text predeterminat i evita que els camps quedin amb espais en blanc o sense informació clara, assegurant que la interfície es mantingui ordenada. A més, això ajuda a evitar que l'usuari deixi informació parcialment introduïda o incompleta.

Classe VistaPrestatgeria

Classe i atributs

Aquesta classe s'encarrega de crear la finestra de la interfície gràfica que permet als usuaris consultar el contingut de les seves prestatgeries a més de donar-los-hi l'opció d'editar-los o eliminar-los.

Només es pot accedir a aquesta finestra un cop l'usuari hagi seleccionat Prestatgeries en el Menú Usuari. Aquesta finestra actua com un punt central per veure la llista de prestatgeries i permet als usuaris crear-ne de noves a través d'una altra finestra associada. A més, l'aplicació facilita la navegació entre dues pàgines principals: una per mostrar les prestatgeries disponibles i l'altra per visualitzar en detall la disposició dels productes d'una prestatgeria seleccionada.

La primera pàgina permet als usuaris veure totes les prestatgeries en una llista interactiva, on cada prestatgeria es mostra com un botó que inclou el seu identificador i nom. Aquesta llista es construeix dinàmicament amb un panell en format de graella (GridLayout) que es troba dins d'un panell desplaçable (JScrollPane). També hi ha un botó per crear una nova prestatgeria, que obre una finestra específica per a aquesta tasca (vistaCrearPrestatgeria).

La segona pàgina es mostra quan l'usuari selecciona una prestatgeria. Aquesta pàgina conté el títol de la prestatgeria seleccionada i una visualització detallada del seu layout en un panell de graella dinàmica. Cada producte i la seva posició es representen amb etiquetes (JLabel) formatejades per millorar la claredat visual. Els usuaris poden tornar a la pàgina inicial mitjançant un botó "enrere". A més, poden eliminar la prestatgeria o intercanviar les posicions de dos productes.

Funcions de la classe

- vistaPrestatgeria() (Constructor)

Aquesta funció inicialitza els components principals de la finestra (initComponents) i crida altres funcions auxiliars per configurar l'estil (initStyles), gestionar la navegació entre pàgines (cardInit) i carregar les prestatgeries en la interfície (cargarPrestatgeriesEnScrollPanel). Assegura que la finestra estigui preparada per ser visualitzada correctament.

- initStyles()

Configura els estils visuals de la interfície gràfica, com ara la mida de la finestra, la seva posició centrada i el disseny de les etiquetes (JLabel) i dels panells. També ajusta el disseny de la graella per mostrar prestatgeries amb separació adequada entre els elements.

- cardInit()

Inicialitza el CardLayout i afegeix les dues pàgines principals (page1 i page2) al panell principal (cardPanel). Estableix que la pàgina inicial visible sigui la que mostra la llista de prestatgeries (page1).

- cargarPrestatgeriesEnScrollPanel()

Carrega la llista de prestatgeries a partir de les dades generades (o provinents del controlador de presentacio que ha carregat desde persistencia) i afegeix un botó interactiu per a cadascuna d'elles al panell de graella (panelGrid). Cada botó permet accedir a la vista detallada de la prestatgeria corresponent, canviant a la segona pàgina amb el layout del producte.

- mostrarDispEnPage2(String layout)

Mostra el layout detallat d'una prestatgeria seleccionada en la segona pàgina (page2). Divideix la informació del layout en files i columnes i afegeix etiquetes (JLabel) amb les posicions i productes associats. Aquesta funció assegura que el panell de graella es regeneri dinàmicament per adaptar-se al layout seleccionat.

- btnCrearActionPerformed()

Aquesta funció s'activa quan l'usuari fa clic al botó per crear una nova prestatgeria. Obre la finestra vistaCrearPrestatgeria per gestionar el procés de creació i oculta temporalment la finestra actual.

- btnBackActionPerformed()

Torna a la pàgina inicial (page1) quan l'usuari fa clic al botó d'enrere en la segona pàgina. Utilitza el CardLayout per canviar de pàgina dins de la mateixa finestra.

Classe VistaCrearPrestatgeria

Classe i atributs

Aquesta classe s'encarrega de crear la finestra de la interfície gràfica que permet als usuaris crear una nova prestatgeria i configurar-la amb els paràmetres necessaris. Inclou funcions per introduir informació sobre la prestatgeria. L'objectiu d'aquesta interfície és facilitar la creació d'una prestatgeria personalitzada de manera intuitiva i validant totes les dades proporcionades per l'usuari.

La finestra està dividida en dues pàgines principals gestionades amb un CardLayout. La primera pàgina, page1, permet als usuaris introduir dades bàsiques com l'identificador de la prestatgeria, el nom, i el nombre de columnes. Aquesta pàgina conté diversos camps de text (JTextField), etiquetes (JLabel) per guiar l'usuari, i botons d'acció com "Següent" per avançar a la següent pàgina o per anar cap enrere. Per assegurar que les dades introduïdes siguin correctes, es realitza la validació dels camps amb missatges d'error visibles just a sota de cada camp.

La segona pàgina, page2, es centra en l'assignació de productes a la prestatgeria i la selecció d'un algorisme per a la seva configuració. Aquesta pàgina conté un panell de productes en forma de graella, on cada producte està representat com una casella seleccionable (JCheckBox). També inclou opcions per seleccionar l'algorisme, amb dos botons d'opció (JRadioButton) que representen els mètodes "Força Bruta" i "Dos Aproximació". Si l'usuari té més de 14 productes, i escolleix Força Bruta, se li avisa de que Força Bruta per productes major de 14 productes pot tardar.

Funcions de la classe

- vistaCrearPrestatgeria()

El constructor de la classe inicialitza els components de la finestra mitjançant la funció initComponents. Configura també algunes propietats visuals, com la mida i la ubicació de la finestra, i defineix el gestor de disseny Card Layout per a la navegació entre pàgines. Finalment, crida la funció cargarProductesEnScrollPanel() per emplenar la graella amb els productes disponibles de l'usuari.

- validarID()

Comprova si el camp d'identificador conté un valor vàlid. Verifica que l'entrada no estigui buida, que sigui un número enter positiu, i actualitza l'etiqueta d'error errorId amb missatges adequats en funció del resultat.

- validarNom()

Valida el camp del nom (nomtxt). Comprova que el text no estigui buit i actualitza l'etiqueta errorNom amb un missatge positiu o d'error, segons el cas.

- validarNumColumnes()

Aquesta funció valida el camp del nombre de columnes (colstxt). Verifica que l'entrada sigui un número enter major que zero i mostra el missatge corresponent a errorNcols.

- btnCrearActionPerformed()

Gestiona l'acció del botó "Següent". Si tots els camps de la primera pàgina són vàlids (validarID, validarNom, validarNumColumnes), canvia la vista a la segona pàgina mitjançant el gestor CardLayout.

- btnSortirActionPerformed()

Gestiona l'acció del botó per cancelar l'accio de crear una prestatgeria i tornar a visualitzar les prestatgeries.

- btnSortir2ActionPerformed()

Gestiona l'acció del botó per tornar enrere per canviar id, nom o numcols.

- cargarProductosEnScrollPane()

Aquesta funció s'encarrega d'emplenar el panell de productes (panelGrid) amb caselles seleccionables (JCheckBox). Per a cada producte, es crea una casella amb l'identificador del producte i s'afegeix al panell. També defineix el disseny del panell en forma de graella.

- IdtxtFocusGained()
- IdtxtFocusLost()
- colstxtFocusGained()
- colstxtFocusLost()
- nomtxtFocusGained()

- nomtxtFocusLost()

Aquestes funcions de focus, controlen l'aspecte i el comportament dels camps de text, assegurant que mostrin un text predeterminat quan estan buits i canviïn el color per millorar l'experiència d'usuari

Classe MenuUsuariPanel

Classe i atributs

La classe VistaMenuUsuari és una finestra que conté un menú d'opcions per a un usuari. Cada botó dins de la finestra té un gestor d'esdeveniment associat (per exemple, botoEstanteriaActionPerformed(), botoProductesActionPerformed(), etc.) que defineix què succeeix quan l'usuari fa clic en aquests botons. Els mètodes initComponents() i initContent() són responsables de configurar la interfície gràfica i els seus comportaments. Aquest menú permet navegar per les diverses funcionalitats de l'aplicació de manera còmode i visual.

La finestra està composta per dues parts diferenciades. En primer lloc, hi ha un menú a l'esquerra amb els següents botóns: Prestatges, Productes, Sobre Nosaltres, Usuari i Tancar Sessió. Cada una mostra un panel amb la informació corresponent. També es pot interactuar amb el títol de la nostra aplicació, Prestatgeries n' Products,el cual porta a una pantalla de benvinguda. Aquesta pantalla és també la que apareix un cop s'inicia sessió.

A l'altra banda de la finestra, hi ha un panel que mostra les diverses vistes mencionades anteriorment.

Funcions de la classe

- VistaMenuUsuari(): És el nom de la classe que representa una vista del menú d'usuari. Crea una finestra que conté elements gràfics per interactuar amb l'usuari.
- initComponents(): Aquest mètode s'utilitza per inicialitzar i configurar tots els components gràfics que es veuran a la finestra, com botons, etiquetes, panells, etc. Aquí s'estableixen propietats dels components com la mida, el text, i les accions que s'han d'executar quan s'interactua amb ells.
- initContent(): Aquest mètode és un lloc on es configuren detalls específics del contingut o la lògica de la vista, com afegir els components a un contenidor o assignar valors a camps específics. Mentre que initComponents es centra en la

creació dels components, initContent és utilitzat per configurar les dades i el comportament dinàmic de la interfície.

- botoEstanteriaActionPerformed(): Aquest és un gestor d'esdeveniment associat a un botó que té el text "Estanteria". Quan l'usuari fa clic en aquest botó, es crida el mètode botoEstanteriaActionPerformed(). Es mostra al panel la vista Prestatgeria.
- botoProductesActionPerformed(): Similar a l'anterior, aquest és un altre gestor d'esdeveniment, però relacionat amb un botó anomenat "Productes". Quan l'usuari fa clic en aquest botó, es mostra la vista relacionada amb els productes.
- botoUsuariActionPerformed(): Aquest és un altre gestor d'esdeveniment per al botó "Usuari". Quan es clica, es mostra un panel amb informació sobre el compte. Aquest panel té un botó per a canviar la contrasenya.
- botoAboutUsActionPerformed(): Rutina de gestió del botó About Us. Passa a mostrar per pantalla informació sobre l'equip, com ara les nostres motivacions o la nostre foto de grup.
- botoTancarSessioActionPerformed(): Aquesta funció gestiona la rutina de tractament del botó Tancar Sessió. Tanca la sessió de l'usuari actual i retorna a la vista d'inici de sessió.

Algorismes i estructures de dades

Estructura de dades i algorismes utilitzats

Brute Force Algorithm

L'objectiu principal de la classe BruteForce es determinar la millor disposició d'un conjunt de productes basant-se en una matriu de similituds. Per aconseguir això, es genera cada permutació possible dels productes, seleccionant la que maximitza la similitud total. Aquesta implementació es basa en un algorisme de força bruta, adequat per a conjunts petits de dades, tot i les seves limitacions de rendiment en conjunts grans.
Inicialment, utilitzàvem la classe Producte per tenir el vector de productes que volíem transformar en l'òptim, però ens vam adonar que era millor treballar amb un vector d'ints, que representen els identificadors dels productes, per evitar el acoblament.
A més, també vam haver de modificar l'algorisme per tal que pogués optimitzar la disposició en prestatgeries amb més d'una fila. Inicialment, l'algorisme estava dissenyat per a prestatgeries amb una sola fila, però per adaptar-lo a la nova estructura, vam decidir mantenir la seva implementació original i afegir una funció addicional. Aquesta nova funció transforma la disposició d'una prestatgeria amb una fila en una disposició més complexa,

amb les files i columnes que l'usuari desitja. Així, l'algoritme manté la seva estructura bàsica, però és més flexible i pot gestionar prestatgeries amb múltiples files, adaptant-se a les necessitats específiques de l'usuari.

-double[][] matSimilituds: Conté les similituds entre parelles de productes. Cada valor *matSimilituds[i][j]* representa la similitud entre el producte ***i*** i el producte ***j***.

-int[] vecProductes: Conté els identificadors dels productes que volem disposar de manera òptima a la prestatgeria. Els índexs del vector vecProductes estan directament relacionats amb els índexs de la matriu de similituds. És a dir, l'element *vecProductes[i]* correspon al producte que ocupa la fila ***i*** i la columna ***i*** a la matriu de similituds.

Aquesta relació és essencial per calcular la similitud entre productes en una permutació donada, ja que permet utilitzar els índexs d'una permutació per accedir als valors de similitud a la matriu de forma directa i eficient.

Inicialment, vam considerar treballar directament amb el vector de productes (vecProductes) durant la generació de les permutacions. La idea era generar permutacions del vector complet d'objectes Producte i, per a cada permutació, fer una cerca (find) per determinar l'índex de cada producte dins del vector original vecProductes. Però fer cerques (find) repetides per trobar l'índex de cada producte durant cada permutació és ineficient, especialment quan el nombre de productes creix. A més, aquest enfocament feia que l'algorisme fos més complicat del necessari.

Per solucionar això, vam pensar en generar les permutacions no sobre els productes directament, sinó sobre un vector d'índexs que representa la seva posició en *vecProductes*. Els índexs es feien servir per accedir directament a *matSimilituds* per calcular la similitud total. Només al final del procés, quan ja s'havia determinat quina permutació era la millor, s'utilitzava el vector original *vecProductes* per reconstruir la solució final amb els objectes Producte.
Més endavant ens vam adonar que no era una bona idea treballar directament amb els Productes, ja que podíem treballar amb els seus identificadors (ints) de manera més eficient. D'aquesta manera, evitem l'acoblament entre classes i fem l'algorisme més flexible, ja que només es fa servir el valor identificador per realitzar les operacions necessàries. Per tant, el problema mencionat anteriorment va desaparèixer.

-int[] vecResultat: Conté la disposició òptima dels productes, obtinguda després d'explorar totes les permutacions.
-int[][] matResultat: Conté la disposició òptima dels productes, obtinguda després d'explorar totes les permutacions possibles, amb les columnes i files que l'usuari ha especificat per a la prestatgeria. Aquesta disposició s'obté a partir de vecResultat.
-int columnes: les columnes que ha de tenir la prestatgeria
-int files: les files que ha de tenir la prestatgeria
-double millorSimilitud: Emmagatzema el valor de la millor similitud trobada fins al moment. Inicialment, està definit com -1.0 per identificar que encara no s'ha trobat cap resultat vàlid. Al final contindrà la similitud total de la solució òptima.

-Algorisme: El pas clau de l'algorisme és generar totes les possibles disposicions dels productes (identificadors). Això es fa mitjançant una tècnica recursiva que intercanvia elements dins del vector d'índexs. Per a cada permutació generada, es calcula la similitud total de la disposició i es guarda en *millorSimilitud* i en *vecResultat* si és la millor disposició que hem trobat fins al moment.

Sabem que l'estanteria és cíclica, és a dir, que el primer producte està connectat amb l'últim. Aquesta propietat implica que la disposició final no només depèn de l'ordre dels productes consecutius, sinó també de com tanquen el cicle entre el primer i l'últim producte. Gràcies a aquesta propietat cíclica, podem simplificar el problema sense perdre cap combinació rellevant: podem fixar sempre el primer producte en una posició específica, reduint així el nombre total de permutacions que cal generar.

Si no tinguéssim aquesta propietat cíclica, hauríem d'explorar totes les permutacions possibles dels n productes, cosa que generaria un total de n! permutacions, però ara només hem de fer (n - 1)! permutacions.

Inicialment, per calcular el factorial, es feia servir un tipus de dada int, ja que semblava suficient per a la majoria dels casos previstos. Durant les primeres proves de l'algorisme de força bruta, el rendiment semblava correcte en problemes petits. Quan vam començar a comparar el seu funcionament amb l'algorisme de 2-aproximació per a problemes lleugerament més grans, ens vam adonar que el programa deixava de funcionar o resolia el problema de manera anormalment ràpida. En revisar detalladament el codi i els resultats, vam descobrir que el problema no estava en l'algorisme en si, sinó en les limitacions del tipus int. Com que el factorial creix molt ràpidament, els valors excedien la capacitat d'un int, provocant desbordaments i fent que els càlculs fossin incorrectes.

Per solucionar-ho, vam decidir substituir l'ús d'int pel tipus BigInteger, que permet treballar amb nombres enters molt més grans. No obstant això, cal tenir en compte que l'algorisme de força bruta no està dissenyat per a problemes amb un nombre molt gran d'elements.

Dos Aproximació

L'algoritme Dos Aproximació intenta aconseguir un resultat similar a l'algoritme anterior amb un temps molt més reduït, de tal manera que la suma de similituds resultant sigui com a molt dues vegades pitjor. La idea principal serà construir un MST per tal de trobar les arestes amb més similitud, i d'allà construïr la nostra estanteria resultant.

L'algoritme comença creant una llista d'arestes, el conjunt de les quals formen un Maximum Spanning Tree. Per això, es crida a la funció MST. Aquesta funció comença ordenant el conjunt de totes les arestes amb un MergeSort en ordre descendent (ens interessen les arestes més grans, és a dir, amb més similitud). Un cop obtingudes les arestes ordenades, fem ús de l'algoritme de Kruskal per generar aquest MST. Aquest algoritme consisteix en anar penjant arbres MST fins que et queda un únic MST. Inicialment, hi ha n MSTs, on cada node del graf és el seu propi arbre. Aquest algoritme fa ús de la funció Find i la funció Uneix:

- Find: Retorna l'arrel de l'arbre al qual pertany el node.
- Uneix: Penja l'arrel de l'arbre amb més nodes a l'arrel de l'arbre menys nodes.

Mentres totes les arestes no formin part del mateix arbre, es va iterant per totes les arestes en ordre descendent. Si els dos vertex d'una aresta pertanyen a arbres diferents (es mira amb la funció Find), llavors s'uneixen amb la funció Uneix, i s'afegeix l'aresta a la llista resultant. Finalment, l'algoritme retorna la llista d'arestes que formen aquest arbre.

Tot i que ens encantaria tenir una estanteria de disseny amb calaixeres repartides arbitràriament, la nostra estanteria és una fila de calaixeres circular, i per tant, hem d'interpretar aquest arbre com a un vector, on l'últim element es comunica amb el primer. Aquí comença la segona part de l'algoritme: hem de construir un cicle eulerià (és a dir, un cicle que passi per totes les arestes). Per poder crear aquest cicle, primer hem de crear un graf dirigit de l'arbre resultant anterior. Per això, es crea una llista de llistes d'enters, on cada posició de la primera llista representa la aresta, i el contingut (la segona llista) són els vertex amb els que es comunica. Finalment, es crida a la funció findEuleria, la qual s'encarrega de trobar aquest cicle.

Aquesta funció va guardant en una pila els vèrtex del graf i, per cada vèrtex, es van afegint els seus veïns a la pila i s'eliminen les connexions corresponents del graf. Quan un vèrtex no té més connexions, s'afegeix al cicle i es retira de la pila. Aquest procés es repeteix fins que s'han recorregut totes les arestes.

Un cop obtingut el cicle eulerià, ja comptem amb una disposició adient de la prestatgeria, però hi ha un problema… Hi poden haver nodes repetits! Inicialment, no ens vam preocupar massa, i eliminavem els nodes repetits començant desde la primera posició. Més endavant, testejant jocs de prova, ens vam adonar que en alguns casos, la solució era molt ineficient:

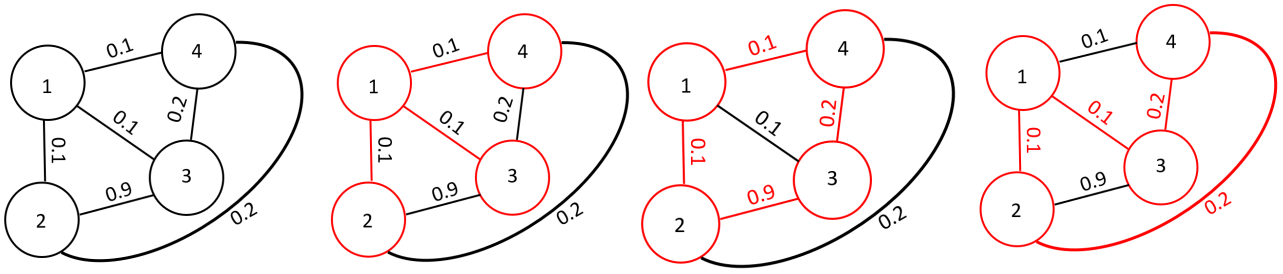

Figura 1 *Figura 2* *Figura 3* *Figura 4*

Figura1: Grafs d'exemple d'eliminació de nodes. L'ordre mencionat és d'esquerra a dreta.

En aquest exemple, on es busca la suma mínima, la 1ra figura mostra un graf de similituds, mentres que la 2na remarca en vermell l'MST d'aquest graf. Després d'executar el nostre algoritme, el cicle Eulerià ens retornaría un vector que seria: [1, 2, 1, 3, 1, 4]. Eliminant els elements repetits, el vector resultant seria [1, 2, 3, 4], tal com es mostra a la 3ra figura, amb similitud total 1.3. Però aquesta solució és molt més ineficient que la solució òptima, la de la 4ta figura, amb similitud 0.6 i vector resultant [2, 1, 3, 4].

És per això, que vam canviar aquesta última part. La idea és generar una disposició tal que maximitza la suma de similituds. Una manera de resoldre el problema seria anar calculant totes les sumes començant desde cada element del cicle Eulerià, tot i que encara es poden optimitzar més. El que fa el nostre algoritme és, per a cada element del cicle, calcular quants cops hi apareix. Després, per a cada element que hi apareix més d'una vegada, es mira el vector resultant eliminant els repetits. Això es realitza amb la funció calculaSuma, que recorre el cicle i calcula la similitud acumulada per cada possible ordre dels productes.

Si es troba una configuració millor que l'actual, aquesta es guarda com la millor solució. Finalment, retorna el millor vector trobat, que es va guardant a la variable res_productes.
La funció calculaSuma també es crida abans de començar a calcular totes les sumes, ja que pot ser que no hi hagin elements repetits, i per tant el vector que s'estaria retornant seria Null.

Comparació dels Algoritmes

Per veure les diferències entre els dos algoritmes, hem executat un programa que compara els dos algoritmes, imprimint-ne el cost i el temps que ha trigat cadascun. La següent taula mostra, per a diversos jocs de prova, els resultats. Els jocs de prova mostrats van augmentant de mida (prova2 té 2 productes, prova3 en té 3, etc).

Per fer aquesta prova i poder assegurar que la solució 2-aproximada quedava acotada per 2 cops la solució òptima (calculada amb l'algoritme de força bruta), s'han canviat els criteris dels algoritmes per buscar la solució mínima. Aquest canvi no està present als algoritmes finals, ja que la solució que busca el nostre problema és la màxima.

provaX	**OPT**	**2 A**	**2 OPT**	**temps BF (ms)**	**temps 2A (ms)**
prova2	0,72	0,72	1,44	18,92	21,91
prova3	1,63	1,63	3,26	27,19	29,87
prova4	0,79	0,79	1,58	20,02	19,15
prova5	1,79	1,79	3,58	29	14,91
prova6	0,73	0,73	1,46	28,32	30,05
prova7	1,16	1,16	2,32	30,33	28,7
prova8	1,88	2,11	3,76	36,67	45,71
prova9	1,79	2,21	3,58	19	44,69
prova10	2,66	2,66	5,32	123,57	36,64
prova11	2,01	3,1	4,02	250,28	41,58

prova12	1,78	1,84	3,56	2090,43	48,58
prova13	1,36	2,24	2,72	22125,78	67,93

Figura 2: Taula de comparació entre l'algoritme 2Aproximació i la solució òptima

Observant les 3 primeres columnes, obtenim la següent gràfica:

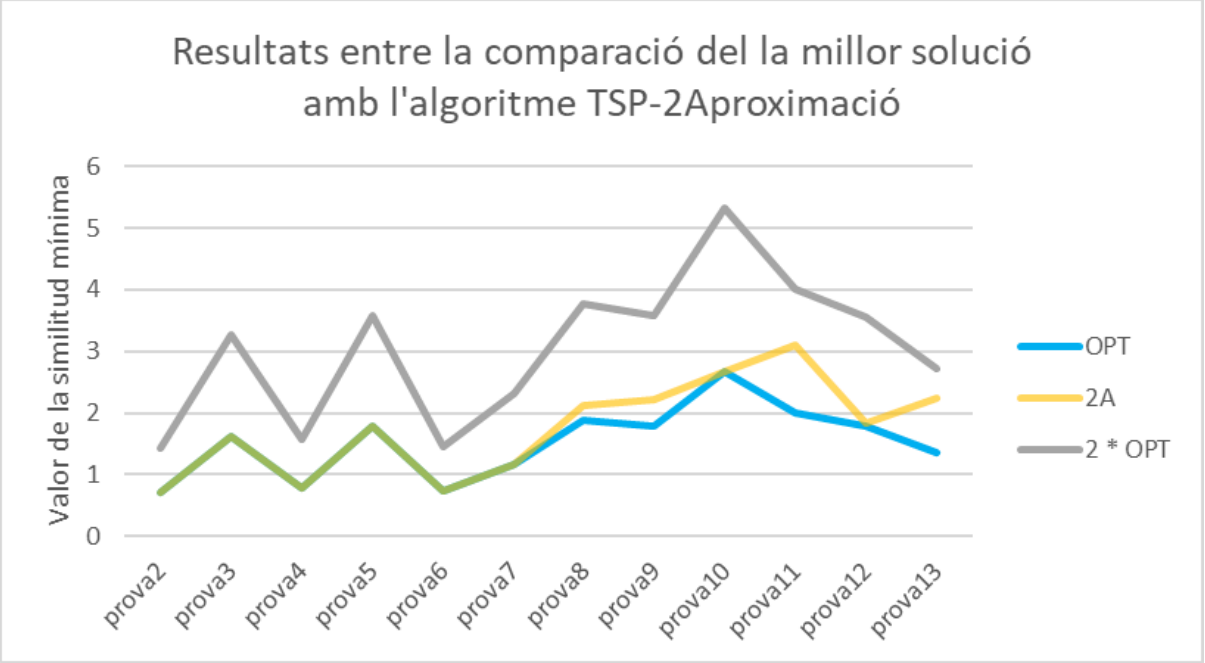

Figura 3: Gràfica comparativa de l'algoritme 2-Aproximació respecte la solució òptima.

Amb aquest experiment es veu clarament que l'algoritme BF té un temps d'execució que creix exponencialment amb el nombre de productes. Això es fa especialment evident a partir de *Prova10*, on el temps de BF arriba a més de 120 mil·lisegons i continua incrementant-se dràsticament amb la mida dels jocs de prova. De fet, ens va ser impossible provar jocs amb mida més gran que 13, ja que el cost computacional era massa elevat.

En canvi, l'algoritme 2 Aproximació (2A) manté un temps d'execució molt més estable i eficient, fins i tot per a jocs de prova més grans. Aquesta diferència de rendiment fa que 2A sigui l'opció preferida quan es treballa amb conjunts de dades de gran mida. L'algoritme BF es torna impracticable per a jocs de prova amb més de 12 productes, mentre que 2A pot continuar processant eficientment la informació, fins i tot amb proves que tenen centenars de productes.

En termes de la qualitat dels resultats, ambdós algoritmes produeixen resultats similars per a la majoria de jocs de prova. Això indica que 2A és capaç d'aproximar-se molt bé a la solució òptima trobada per BF, especialment en casos on el nombre de productes no és extremadament gran. Com es veu a la gràfica de l'experiment, entre prova 2 i prova 7 les línies que marquen la solució se sobreposen perquè donen el mateix resultat. A partir d'aquest valor, la línia groga, corresponent al resultat del 2 Aproximació, augmenta respecte a la línia blava, corresponent a la solució òptima. Tot i això, la solució del 2A sempre queda acotada per sota de 2 vegades la solució òptima (línia grisa), confirmant així la correctesa d'aquest algoritme de manera empírica.

Aquest petit experiment demostra que, tot i que l'algoritme BruteForce ofereix solucions òptimes, el seu alt cost computacional el fa inviable per a conjunts de dades grans. DosAproximacio, en canvi, és significativament més ràpid i, a la pràctica, genera resultats amb una qualitat molt propera a l'òptima. Per tant, DosAproximacio es presenta com l'algoritme més adequat per a la majoria d'aplicacions pràctiques, especialment quan es tracta de grans volums de dades.

Millora Dos Aproximació

Després de la primera entrega, es va fer un experiment a classe per provar els algoritmes de tots els grups. Tot i que la nostra solució es troba dintre del llindar d'un TSP-Dos Aproximació, es trobava més llunyana que la resta de solucions. És per això que vam decidir implementar una millora d'aquest algoritme per a la entrega final.

La nova solució canvia el mètode d'eliminació de nodes del camí Eulerià. Originalment es començava a comptar els nodes no repetits des de cada element del vector, i ens quedavem amb la solució més alta. El problema d'aquesta solució es que no quedava massa acotada dins de les solucions possibles. Per això, s'ha implementat una solució recursiva que corretgeix l'error. Pe rmostres més grans, aquesta solució és més ineficient, ja que s'itera masses vegades sobre el conjunt, per això, quan la mostra supera un cert llindar, es torna a la solució anterior.

Estructures de dades de les classes

- Classe Producte:

Aquesta s'encarrega de representar un producte dins del sistema, amb atributs de tipus simples com Integer o String que defineixen les característiques bàsiques de l'objecte. A més, presenta una estructura de dades de tipus Map<Integer, Double> on emmagatzamem les similituds que té amb els altres productes. La clau representa l'identificador de l'altre producte i el valor Double és un valor numèric entre 0 i 1 (veure restricció de valors al document de consideracions del sistema).

En lloc de crear una classe separada Similitud, hem optat per aquesta opció perquè aquesta relació quedi directament connectada a cada producte. L'ús d'un Map<Integer, Double> per a la similitud fa que l'estructura de la classe Producte sigui més intuïtiva i més flexible de cara a futures entregues. Una estructura d'aquest tipus fa més ràpida i eficient el poder accedir a la similitud entre dos productes, ja que només cal consultar el *map* del producte en qüestió, així com el poder realitzar operacions com afegir, eliminar o modificar de manera més dinàmica sense necessitat de canviar el disseny de la classe.

Aquest disseny ens permet una gestió centralitzada de la informació rellevant del producte, ja que concretem el concepte de les connexions entre productes i simplifiquem tant l'accès com la manipulació de les dades en el sistema.

- Classe CjtProductes:

El disseny de la classe permet gestionar un conjunt de productes associats a un usuari concret. Aquesta estructura permet organitzar, accedir i controlar eficientment els productes, a l'igual que la classe Producte, a través d'un mapa clau-valor.

Com a atributs principals, fem servir un String per representar el nom d'usuari al qual pertany aquest conjunt de productes i que actua com a identificador. Això ens garanteix que les operacions i dades del conjunt de productes estiguin restringides a l'usuari associat en aquell moment. D'altra banda, fem ús d'una estructura de dades de tipus Map<Integer, Producte> on la clau representa l'identificador de cada producte i el valor és l'objecte que conté tota la informació del producte (id, nom, posició i similituds).

Com hem mencionat anteriorment, aquest tipus d'estructura ens proporciona un accés ràpid a qualsevol producte a través del seu *id*, amb una complexitat de cerca constant (O(1)), és a dir, amb gran eficiència i una gestió ordenada dels productes, on es pot accedir directament amb l'*id* i podem realitzar operacions d'edició sense afectar altres elements.

Degut a que es tracta d'un conjunt dinàmic (el nombre de productes pot variar al llarg del temps segons les necessitats de l'usuari), hem decidit que un *map* és també la millor opció i el fet que associem aquest conjunt a un usuari ens assegura que estigui totalment limitat a un usuari concret, atorgant una seguretat i coherència a les dades.

- Classe Usuari:

La funció principal és la de representar a un usuari del sistema, simplificant la informació necessària per tal d'identificar-lo i autenticar-lo. La seva estructura és bastant senzilla però suficient per a garantir un principi de seguretat i personalització dins del nostre projecte.

Es basa principalment en l'ús d'un String que guarda el nom d'usuari com a identificador únic que representa l'usuari al sistema i que ens permet distinguir aquest usuari d'altres i s'utilitza com a referència en diverses operacions, i d'una contrasenya que afavoreix la seva autenticació, ja que s'utilitza per verificar que l'usuari que intenta accedir és el propietari real del compte. Això millora en termes de seguretat les dades associades. Tot i que, som conscients que una implementació real en un sistema complex hauria de protegir aquest atribut amb altres tècniques que no sigui un simple String.

De cara a futures entregues o possibles millores, aquesta estructura es pot ampliar per adaptar-se a necessitats pròximes.

- Classe Prestatgeria:

Hem millorat l'estructura de la classe **Prestatgeria** per representar millor la disposició dels productes en un supermercat, adaptant-la a les necessitats del sistema. Ara, una prestatgeria pot contenir diversos prestatges organitzats en forma de matriu, representada per un **layout** de productes i una altra matriu separada d'identificadors. Aquesta organització ens permet gestionar i accedir a la informació de manera més eficient i mantenir un disseny desacoblat entre les diferents parts del sistema.

La nova implementació utilitza una **LinkedList** per emmagatzemar les files del layout. Aquesta decisió es justifica perquè una **LinkedList** facilita operacions dinàmiques com l'addició o eliminació de files, que poden ser freqüents quan es manipula la configuració de

la prestatgeria. Això aporta més flexibilitat en comparació amb un array estàtic, evitant la necessitat de copiar elements manualment en operacions de redimensionament.

Layout de Producte[][]:
La matriu principal emmagatzema directament objectes de tipus **Producte**, representant la distribució visual i física dels productes en la prestatgeria. Cada fila correspon a un prestatge, i cada columna a una posició dins d'aquest prestatge. Això permet accedir o modificar els productes directament segons la seva posició amb un cost constant.

Layout d'identificadors (disp):
Per evitar acoblaments innecessaris entre la lògica dels algorismes i els objectes de tipus **Producte**, s'ha introduït un altre layout, **disp**, que emmagatzema únicament els identificadors dels productes en una llista de llistes. Aquesta estructura separa la informació sobre la disposició (IDs dels productes) de la lògica dels objectes, cosa que facilita l'aplicació d'algorismes que només necessiten treballar amb identificadors.

La classe Prestatgeria a més inclou diversos atributs que defineixen la seva identificació i característiques essencials. L'atribut id és un identificador únic en format enter que permet diferenciar cada prestatgeria dins del sistema, mentre que nom és una cadena que descriu la prestatgeria de manera comprensible per a l'usuari. Els atributs numFilas i numColumnas especifiquen les dimensions físiques de la prestatgeria, també en format enter, per delimitar el seu espai. A més, la prestatgeria conté productes, un conjunt (Set<Integer>) d'identificadors únics que garanteixen una gestió eficient dels productes que formen part de la prestatgeria, facilitant operacions de cerca i assegurant que no es repeteixin. Aquesta estructura permet una gestió òptima i clara de les prestatgeries dins del sistema.

- Classe CtrlDomini:

Com a controlador principal, aquest gestiona la lògica del domini en el nostre sistema i unifica les operacions que involucren la resta de les classes, com són els usuaris, productes i prestatgeries, assegurant una gestió coordinada de les dades. Per aquest motiu, fem ús de diverses estructures de dades que ens faciliten aquest treball.

D'una banda, tenim un atribut de tipus Usuari que representa l'usuari que ha iniciat sessió al sistema en aquell moment i que ens permet delimitar les operacions de les classes segons l'usuari actiu. De cara a futures entregues, desenvoluparem l'administració de sessions. De

manera que, per mantenir un ordre i seguretat de la independència de les dades, hem disposat d'un conjunt d'usuaris, que estableix tots els usuaris registrats al sistema i que ens garanteix que cadascú tingui només tingui accés al seu conjunt associat de productes i d'un conjunt de productes que engloba tots els productes associats a l'usuari actual, el qual es podrà recuperar o crear quan s'inicia sessió.

Altrament, entra en joc una prestatgeria que representa la distribució actual dels productes de l'usuari en el format d'un objecte Prestatgeria i on podrem realitzar les operacions pertinents de la classe. També hem establert una matriu de valors *double* (mateix tipus que les similituds) que guarden les similituds entre els productes de l'usuari i la fem servir ja que és una estructura clau per generar les solucions òptimes dins la prestatgeria segons quin algoritme decideix l'usuari que vol emplear i un vector de Productes amb tots els productes actuals de l'usuari i que serveix com a base per construir o modificar la prestatgeria en funció dels criteris que tingui. Ens hem decidit per aquestes estructures principalment per guanyar eficiència en les operacions d'accés i càlculs relacionats amb les classes involucrades.

- Classe CjtPrestatgeries:

Com a atributs principals, utilitzem un **String** per identificar l'usuari al qual pertany aquest conjunt de prestatgeries, assegurant que totes les operacions i dades estan restringides exclusivament a l'usuari associat. A més, fem ús d'un **Map<Integer, Prestatgeria>**, on la clau és l'identificador únic de cada prestatgeria i el valor és l'objecte que conté tota la seva informació (id, nom, layout, dimensions i productes).