

Supermaket Manager

Subgrup 11.3

Guillem Cabré Farré, @guillem.cabre
Marc Peñalver Guilera, @marc.penalver
Àlex Rodríguez Rodríguez, @alex.rodriguez.r
Marc Teixidó Sala, @marc.teixido

Versió del lliurament 2.0
PROP - Quadrimestre de tardor 2024
Data: December 16, 2024

Continguts

1	Canvis realitzats respecte al primer lliurament	3
1.1	Canvis en la Definició de Classes i Estructures de Dades	3
1.1.1	Canvis en la Classe <code>Catalog</code>	3
1.1.2	Mappers i DTOs	4
1.1.3	Controlador de Persistència	4
1.1.4	Utilitari per a la Gestió d'Imatges	5
1.2	Canvis en els Algorismes de Distribució de Productes	6
2	Definició de classes i les seves estructures de Dades	7
2.1	Classes, interfícies i enums	7
2.1.1	Capa de Presentació	8
2.1.2	Capa de Domini	11
2.1.3	Capa de Persistència	13
2.2	Relació de classes	14
2.3	Estructura JSON	16
3	Algorismes per a la distribució de productes al Supermercat	17
3.1	Introducció	17
3.2	Objectiu	17
3.3	Organització de les prestatgeries	17
3.3.1	Adreçament de les prestatgeries i les posicions dels productes	18
3.4	Algorisme de força bruta	19
3.4.1	Pseudocodi de l'algorisme	19
3.4.2	Complexitat	19
3.5	Algorisme Greedy	21
3.5.1	Pseudocodi de l'algorisme	21
3.5.2	Complexitat	21
3.6	Algorisme d'aproximació	23
3.6.1	Simulated Annealing	23
3.6.2	Pseudocodi de l'algorisme	24
3.6.3	Estratègia de generació de solucions inicials	25
3.6.4	Paràmetres de Simulated Annealing	25
3.6.5	Operadors	25
3.6.6	Complexitat	26
3.7	Conclusions	28

4	Descripció de l'Arquitectura en 3 Capes	29
4.1	Introducció	29
4.2	Capa de Presentació	29
4.3	Capa de Domini	29
4.4	Capa de Persistència	30

Chapter 1

Canvis realitzats respecte al primer lliurament

En aquest segon lliurament, hem realitzat una sèrie de canvis i millores en el codi i la documentació del projecte, en base a les noves incorporacions a l'aplicació i a les crítiques constructives que vam rebre per part del professorat. A continuació, es detallen els principals canvis realitzats respecte al primer lliurament.

1.1 Canvis en la Definició de Classes i Estructures de Dades

L'únic canvi sobre les classes de model respecte al primer lliurament ha estat la modificació d'un dels mètodes de la classe *Catalog*.

Més enllà d'aquests canvis, s'ha realitzat una extensió del sistema per facilitar la comunicació entre les diferents capes del projecte. Aquesta extensió inclou la creació de mappers i DTOs per les principals estructures de dades del model, una factoria pel controlador de persistència, interfícies per aquest mateix controlador, i una classe utilitària per a la gestió d'imatges. A continuació, es descriuen aquests elements amb més detall:

1.1.1 Canvis en la Classe Catalog

El canvi de la classe *Catalog* s'ha realitzat sobre el mètode **searchProduct** per tal de fer la cerca de productes més intuïtiva i útil. A continuació, es detalla el pseudocodi del nou mètode (s'ignoren majúscules i minúscules a la cerca amb la instrucció *toLowerCase()*):

Algorithm 1: Mètode **searchProduct**

```
1 public List < Product > searchProduct(searchInput):  
2   filteredProducts ← empty list;  
3   foreach product in Catalog do  
4     if product.name contains searchInput or  
5     any keyword in product.keywords contains searchInput then  
6       add product to filteredProducts;  
7     end  
8   end  
9   return filteredProducts;
```

Diferències amb l'anterior mètode

L'anterior mètode es basava en l'algorisme Jaccard per calcular la similitud entre els productes. Aquest, a més estava fet de tal manera que sempre apareguessin 10 productes com a resultat de cada cerca.

El problema que vam trobar va ser que per un cerca com per exemple "Apple", apareixien abans diversos productes que el propi amb el nom "Apple". A més, treballant sobre la capa de presentació, vam detectar que la cerca, en molts casos, no era intuïtiva per l'usuari. És per això que vam optar per un mecanisme més senzill, sense fer aparèixer productes amb similituds altes entre sí, ni tampoc exigir-ne 10 resultats.

El nou mètode, a més de simplificar la implementació, permet una cerca més humana i intuïtiva, fent-la més similar a buscadors d'aplicacions similars, ja que busca productes que continguin la paraula introduïda en el nom o en les paraules clau del producte, ignorant majúscules i minúscules.

Tot i que la cerca no és tan complexa com l'anterior, hem considerat que aquesta és més útil per a l'usuari final, i, per tant, hem decidit canviar-la.

1.1.2 Mappers i DTOs

S'han incorporat mappers i DTOs seguint el **Single Responsibility Principle (SRP)** i el **Dependency Inversion Principle (DIP)**, permetent transportar la informació de la capa de domini a la capa de presentació sense afegir acoblament directe entre aquestes dues capes.

Els Mappers són responsables de traduir entre les classes de model i els DTOs, i viceversa. Aquesta separació permet mantenir el codi modular i fàcil de mantenir. Les classes mapejades són les següents:

- **ShelvingUnits**
- **Products**
- **RelatedProducts**

Només s'han creat mappers per aquestes estructures de dades, ja que tota la informació necessària per a la capa de presentació es troba en aquestes tres entitats. La funcionalitat principal dels mappers és convertir les classes de model en DTOs, però també es dona suport a la conversió inversa en casos concrets, com la modificació de productes mitjançant DTOs.

1.1.3 Controlador de Persistència

S'ha afegit una factoria i una interfície per al controlador de persistència amb l'objectiu de desacoblar la lògica específica de la capa de persistència. Aquesta decisió segueix el **Dependency Inversion Principle (DIP)**, assegurant que les capes superiors (com la de domini) no depenguin de les implementacions concretes de persistència.

La factoria permet instanciar diferents implementacions del controlador de persistència segons les necessitats del projecte, com ara alternar entre persistència en fitxers o en bases de dades. Encara que aquestes implementacions alternatives no formen part de la versió actual del projecte, el disseny està pensat per garantir l'extensibilitat en el futur. L'ús de la interfície assegura que totes les implementacions compleixin els mateixos requisits, cosa que facilita els canvis futurs i millora la flexibilitat del sistema.

1.1.4 Utilitari per a la Gestió d'Imatges

Durant el disseny inicial de la capa de domini, no es va tenir en compte la gestió d'imatges per a la capa de presentació ni on s'emmagatzemarien aquestes. Per solucionar-ho, s'ha decidit que les imatges es trobaran dins la carpeta d'execució del projecte, en una subcarpeta anomenada **assets**.

Per gestionar la creació, modificació i eliminació d'imatges, s'ha creat una classe utilitària dedicada. Aquesta decisió es basa en el **Single Responsibility Principle (SRP)**, ja que afegir aquesta funcionalitat a qualsevol de les classes existents hauria trencat aquest principi. Aquesta classe utilitària proporciona:

- Traducció de paths entre la capa de persistència i domini (paths relatius al projecte) i la capa de presentació (paths absoluts accessibles pel sistema operatiu).
- Funcions per a la creació i eliminació d'imatges, assegurant una gestió eficient i segura.

D'aquesta manera, es redueix la càrrega de responsabilitats en les classes de domini i es manté la separació de preocupacions, millorant la mantenibilitat i la claredat del codi.

1.2 Canvis en els Algorismes de Distribució de Productes

Els canvis realitzats en els algorismes de distribució de productes han estat orientats a millorar l'eficiència, la qualitat de les solucions obtingudes i la llegibilitat del codi. A continuació, es detallen els canvis més rellevants realitzats en aquesta part del projecte:

- **Millores de l'algorisme de força bruta:** S'ha millorat l'eficiència de l'algorisme de força bruta mitjançant l'ús de tècniques de poda. Ara a cada crida recursiva es calcula la similaritat invertida acumulada. D'aquesta manera podem mantenir un màxim global i podar les branques que superen aquest mínim:

$$\text{Similaritat invertida acumulada} = \sum_{i=1}^{k-1} 1 - \text{similitud}(p_i, p_{i+1})$$

```
% Funció de poda per l'algorisme de força bruta
private bool shouldPruneBranch:
    Input: La similaritat invertida i la similaritat actuals
    Output: True si la branca es pot podar, False en cas contrari

return currentInvertedSimilarity >= this.bestScore &&
    currentSimilarity <= this.highestSimilarity;
```

A més de la millora en eficiència, s'ha millorat també la llegibilitat i mantenibilitat del codi, encapsulant el codi en funcions més petites i senzilles.

- **Millores de l'algorisme Greedy:** En l'algorisme Greedy s'ha implementat la mateixa estratègia de poda que en l'algorisme de força bruta.

A més, s'ha millorat la gestió de casos extrems en què no hi ha un empat entre productes per col·locar a continuació d'un altre. Ara, en aquests casos, es fa una recursió amb tots aquests per a trobar la millor solució possible.

També, pel que respecte a la llegibilitat i mantenibilitat del codi, s'han fragmentat parts del codi en funcions més petites i senzilles, per tal de facilitar-ne la comprensió i la modificació.

- **Millores de l'algorisme d'aproximació:** En l'algorisme d'aproximació s'ha millorat la qualitat de les solucions obtingudes mitjançant la creació de diverses solucions inicials aleatòries i una solució inicial generada per l'algorisme Greedy. Per a cadascuna d'aquestes solucions inicials, s'aplica l'algorisme Simulated Annealing i es manté la millor solució obtinguda, la qual s'acaba retornant.

A tots els algorismes s'han extret les funcions compartibles i s'han afegit a la classe `HelperFunctions` per tal de facilitar-ne la reutilització, manteniment i llegibilitat.

Chapter 2

Definició de classes i les seves estructures de Dades

2.1 Classes, interfícies i enums

Per al desenvolupament d'aquest projecte, hem decidit utilitzar un conjunt de classes i interfícies que ens permetin implementar de manera eficient els diferents casos d'ús que hem dissenyat. Aquestes classes no només es fonamenten en les funcionalitats essencials per al bon funcionament de l'aplicació, sinó que també segueixen els principis i patrons de disseny orientat a objectes per assegurar una bona qualitat de codi i facilitat d'ampliació.

Un dels principis fonamentals que hem adoptat en el disseny de les nostres classes és el ****principi obert-tancat**** (*Open-Closed Principle*), el qual estableix que una classe hauria d'estar oberta per a l'extensió, però tancada per a la modificació. Això vol dir que podem afegir noves funcionalitats a través de l'extensió de classes existents o la implementació de noves interfícies, sense necessitat de modificar el codi ja existent. D'aquesta manera, garantim que el sistema sigui altament mantenible i escalable, facilitant la incorporació de noves característiques sense alterar la base del sistema ja implementat.

Per a l'arquitectura del projecte, hem adoptat una estructura de tres capes: Presentació, Domini i Persistència. Aquesta divisió permet una millor organització del codi, facilita la mantenibilitat i millora la separació de responsabilitats entre les diferents parts del sistema.

- **Capa de Presentació:** Aquesta capa inclou la interfície d'usuari (UI) i s'encarrega de la interacció directa amb l'usuari. S'implementa mitjançant components de JavaFX que permeten mostrar i capturar la informació que l'usuari introdueix.
- **Capa de Domini:** Aquesta és la capa central del sistema i conté la lògica del sistema. Inclou les classes que defineixen el comportament del supermercat, la gestió dels productes, les prestatgeries i del catàleg.
- **Capa de Persistència:** Aquesta capa gestiona l'emmagatzematge de la informació. Inclou les classes responsables de llegir i escriure fitxers `.json`, gestionant la persistència de les dades. D'aquesta manera, permet desar i carregar la configuració del supermercat de forma eficient i segura.

A continuació, es detallen les principals classes i interfícies que hem dissenyat, les quals formen l'estructura fonamental del projecte. Aquestes classes estan dissenyades per ser reutilitzades i adaptades a les diferents necessitats que poden sorgir a mesura que el projecte creixi i es modifiqui.

2.1.1 Capa de Presentació

- **Classe ShelvingUnitDTO:**
 - **Descripció:** Representa una prestatgeria amb informació rellevant per a la presentació. Com la seva alçada i el conjunt de **ProductDTO** que conté.
- **Classe ProductDTO:**
 - **Descripció:** Conté la informació dels productes necessària per ser mostrada, com ara el nom, el preu, les paraules clau, la ruta absoluta a la imatge a mostrar i les relacions amb altres productes.
- **Classe RelatedProductDTO:**
 - **Descripció:** Representa la informació sobre la relació entre productes i el grau de similitud.
- **Classe PrimaryButtonController:**
 - **Descripció:** Component d'un botó amb un estil determinat per usar en funcionalitats importants.
- **Classe SecondaryButtonController:**
 - **Descripció:** Component d'un botó amb un estil determinat per usar en funcionalitats secundàries.
- **Classe SetTemperatureController:**
 - **Descripció:** Component d'un choicebox per seleccionar una temperatura de les existents.
- **Classe ShelvingUnitController:**
 - **Descripció:** Component per visualitzar una Shelving Unit del supermercat.
 - **Estructures de dades:**
 - * **supermarketPosition** (Integer): posició que ocupa la shelving unit al supermercat.
 - * **shelvingUnitDto** (ShelvingUnitDto): informació dels productes que conté la shelving unit i la seva temperatura en forma de DTO.
- **SubClasse ShelvingUnitEditionController:**
 - **Descripció:** Component per visualitzar una Shelving Unit del supermercat junt amb més elements per modificar-la.

- **Estructures de dades:**
 - * Les del pare.
- **SubClasse ShelvingUnitSwapController:**
 - **Descripció:** Component per visualitzar una Shelving Unit del supermercat junt amb més elements per poder seleccionar-la per fer swaps.
 - **Estructures de dades:**
 - * Les del pare.
 - * **onToggleButtonStateChanged** (BiConsumer(Integer, Boolean)): Atribut per veure quan ha canviat un toggleButton i a on.
 - * **pos** (Integer): Atribut per saber la posició de si algun togglebutton estava seleccionat per seguir deixant-lo seleccionat encara que s'actualitzi una vista externa que tingui el component.
 - * **height** (Integer): Atribut per saber la alçada d'un togglebutton de si estava seleccionat per seguir deixant-lo seleccionat encara que s'actualitzi una vista externa que tingui el component.
- **Classe ToastLabelController:**
 - **Descripció:** Component d'un text que pot ser d'error o de validació amb els seus dos estils definits.
- **Classe TopLabelController:**
 - **Descripció:** Component d'una barra superior on estan les icones per operar el supermercat junt amb més elements.
- **Classe LoginController:**
 - **Descripció:** Controlador de la vista per fer log in a l'aplicació vista amb dos camps de text per l'usuari i la contrasenya.
- **Classe MainScreenController:**
 - **Descripció:** Controlador de la vista principal de l'aplicació on es visualitza la distribució actual del supermercat.
 - **Estructures de dades:**
 - * **NBDISPLAYEDUNITS** (Integer): Atribut que determina quantes shelving units es poden veure alhora.
 - * **currentIndex** (Integer): Quin es l'index inicial des del que veiem les NBDISPLAYEDUNITS shelving units.
 - * **shelvingUnits** (List(Nodes)): Atribut que determina quantes shelving units es poden veure alhora.
- **Classe EditDistributionScreenController:**
 - **Descripció:** Controlador de la vista d'edició de la distribució del supermercat, semblant a la vista principal pero amb elements per modificar la distribució.

- **Estructures de dades:**
 - * **NBDISPLAYEDUNITS** (Integer): Atribut que determina quantes shelving units es poden veure alhora.
 - * **currentIndex** (Integer): Quin es l'índex inicial des del que veiem les NBDISPLAYEDUNITS shelving units.
 - * **shelvingUnits** (List(Nodes)): Atribut que determina quantes shelving units es poden veure alhora.
 - * **swapping** (Boolean): Atribut que ens indica quant estem en el moment de selecció per fer swaps o quan no ho estem.
 - * **plusIcons** (List(FontIcons)): Quins són les icones que s'han de imprimir per pantalla.
 - * **swappedProducts** (List(Pair(Integer, Integer))): Llista de les posicions que es volen fer swap de dos productes per detectar quan s'hi han dos seleccionats i fer el swap.
 - * **swappedUnits** (List(Integer)): Llista de les posicions de les shelving units per fer el swap entre les dos que apareguin.
- **Classe EditShelvingUnitController:**
 - **Descripció:** Controlador de la vista per editar una shelving unit en concret que consta de veure la shelving unit amb unes opcions tant a l'esquerra com a la dreta d'ella per modificar-la.
 - **Estructures de dades:**
 - * **shelvingUnitPosition** (Integer): Atribut que diu la posició de la shelving unit de la vista.
- **Classe CatalogController:**
 - **Descripció:** Controlador de la vista del catàleg on es pot buscar productes i fer-los les modificacions necessàries.
- **Classe PresentationController:**
 - **Descripció:** Controlador principal de vista encarregat de linkar tots els controladors mitjançant navegacions i també encarregat de comunicar-se amb la capa de domini.
- **Interfície IDomainController:**
 - **Descripció:** Defineix els mètodes necessaris per a la gestió de domini. És a dir totes les funcions necessàries per controlar el supermercat.
- **Classe PersistenceControllerFactory:**
 - **Descripció:** Proporciona una instància del controlador de domini.

2.1.2 Capa de Domini

– Classe Supermarket:

- * **Descripció:** Representa la distribució d'un supermercat, com a un conjunt de prestatgeries amb productes. Aquesta classe serà de tipus *singleton*, d'aquesta manera serà accessible en tot moment i només hi haurà una instància d'aquesta en tot el programa.
- * **Estructures de dades:**
 - **instance** (Supermarket): Instància d'ella mateixa per poder agafar-la des de qualsevol lloc del codi.
 - **registeredUsers** (ArrayList(Users)): Llista d'usuaris del supermercat.
 - **loggedUser** (User): Usuari que té la sessió iniciada.
 - **shelvingUnits** (ArrayList(ShelvingUnit)): Llista d'unitats d'emmagatzematge.
 - **shelvingUnitHeight** (int): Alçada de les prestatgeries.
 - **orderingStrategy** (OrderingStrategy): Estrategia d'ordenació.
 - **importFileStrategy** (ImportFileStrategy): Estrategia d'importació.
 - **exportFileStrategy** (ExportFileStrategy): Estrategia d'exportació.
 - **ADMIN-NAME** (String): Nom d'usuari de l'administrador.
 - **ADMIN-PASSWORD** (String): Contrasenya de l'administrador.
 - **EMPLOYEE-NAME** (String): Nom d'usuari de tots els empleats.
 - **EMPLOYEE-PASSWORD** (String): Contrasenya de tots els empleats.

– Classe ShelvingUnit:

- * **Descripció:** Representa una unitat d'emmagatzematge "prestatgeria" en un supermercat, on s'emmagatzemen un determinat tipus de productes en diferents alçades.
- * **Estructures de dades:**
 - **uid** (Enter): Identificador únic per a la prestatgeria.
 - **products** (List(Product)): Llista que conté els productes de la prestatgeria ordenats per alçades.
 - **temperature** (ProductTemperature): Temperatura que proporciona la prestatgeria per emmagatzemar els productes que necessitin aquella temperatura.

– Classe Product:

- * **Descripció:** Representa un producte dins del sistema, amb els atributs essencials.
- * **Estructures de dades:**
 - **name** (String): Nom del producte.
 - **price** (float): Preu del producte.
 - **temperature** (ProductTemperature): Temperatura necessitada per emmagatzemar el producte.
 - **keyWords** (List(String)): Paraules clau associades al producte per fer busques.

- **relatedProducts** (List(RelatedProduct)): Llista que mostra tots els productes relacionats amb ell junt amb el seu grau de relació.
- **Classe RelatedProduct:**
 - * **Descripció:** Gestiona la relació d'un producte amb un altre amb un grau de relació anomenat similitud.
 - * **Estructures de dades:**
 - **value** (float): Grau de similitud dels dos productes.
 - **product1** (Product): Primer producte de la relació. No pot ser null.
 - **product2** (Product): Segon producte de la relació. Diferent al primer i no pot ser null.
- **Enum ProductTemperature:**
 - * **Descripció:** Enum per gestionar les temperatures de emmagatzematge recomanades per a productes que necessiten condicions específiques de temperatura.
- **Classe Catalog:**
 - * **Descripció:** Gestiona una col·lecció de productes, proporcionant mètodes per afegir, eliminar i cercar a través de l' inventari disponible.
 - * **Estructures de dades:**
 - **catalog** (Catalog): Instància del catàleg per poder usar-lo en qualsevol lloc del codi.
 - **products** (List(Product)): Col·lecció de tots els productes al catàleg.
- **Classe Approximation:**
 - * **Descripció:** Classe per implementar el algorisme d'ordenació per aproximació.
- **Classe BruteForce:**
 - * **Descripció:** Classe per implementar el algorisme d'ordenació per força bruta.
- **Interfície OrderingStrategy:**
 - * **Descripció:** Interfície per a estratègies d'ordenació del supermercat per decidir quin algorisme es fa servir.
- **Classe SupermarketData:**
 - * **Descripció:** Conté informació de tot el supermercat per gestionar la comunicació entre capa de persistència i domini.
 - * **Estructures de dades:**
 - **distribution** (List(ShelvingUnit)): Llista amb totes les instàncies de ShelvingUnit d'un supermercat.
 - **products** (List(Product)): Llista de tots els productes del Catalog d'un supermercat.
 - **shelvingUnitHeight** (Enter): Alçada de les ShelvingUnits d'un supermercat.
- **Classe DomainController:**

- * **Descripció:** Controlador de domini basat en els casos d'ús. Reflexa els canvis de la capa de presentació a la resta de capes.
- * **Estructures de dades:**
 - **hasChanges** (Boolean): Indica si s'han realitzat canvis a l'aplicació que no estan guardats.
- **Interfície IPersistenceController:**
 - * **Descripció:** Defineix els mètodes necessaris per a la gestió de persistència. És a dir, exportar i importar la distribució del supermercat.
- **Classe PersistenceControllerFactory:**
 - * **Descripció:** Proporciona una instància del controlador de persistència.
- **Classe ShelvingUnitMapper:**
 - * **Descripció:** Traducció entre ShelvingUnit i ShelvingUnitDTO.
- **Classe ProductMapper:**
 - * **Descripció:** Traducció entre Product i ProductDTO.
- **Classe RelatedProductMapper:**
 - * **Descripció:** Traducció entre RelatedProduct i RelatedProductDTO.
- **Classe AssetsImageHandler:**
 - * **Descripció:** Classe utilitària encarregada de gestionar la manipulació d'imatges dins del projecte, com ara la creació, modificació i eliminació. També gestiona la traducció entre paths relatius i absoluts per garantir que la capa de presentació pugui accedir a les imatges de manera consistent.
 - * **Estructures de dades:**
 - **ASSETS-IMAGES-PATH** (String): Adreça de la carpeta de les imatges dels productes.
 - **ASSETS-TEMPERATURES-PATH** (String): Adreça de la carpeta dels icones de temperatures.
 - **ASSETS-NOT-FOUND** (String): Misatge d'error quan no s'ha trobat la carpeta assets.
 - **PATH-PREFIX** (String): Prefix per poder instanciar imatges a la capa de presentació.
- **Classe RelatedProductSerializer:**
 - * **Descripció:** Classe utilitària encarregada per serialitzar els RelatedProducts.

2.1.3 Capa de Persistència

- **Classe PersistenceController:**
 - * **Descripció:** Controlador que gestiona tota la lògica de la persistència.
 - * **Estructures de dades:**
 - **importFileStrategy** (ImportFileStrategy): Referència a la estratègia de importació de fitxers.

- **exportFileStrategy** (ExportFileStrategy): Referència a la estratègia d'exportació de fitxers.
- **Interfície ImportFileStrategy:**
 - * **Descripció:** Interfície per a estratègies d'importació de fitxers, permetent la importació de dades des de diferents formats de fitxer.
- **Interfície ExportFileStrategy:**
 - * **Descripció:** Interfície per a estratègies d'exportació de fitxers, permetent l'exportació de dades en diversos formats.
- **Classe ImportFileJSON:**
 - * **Descripció:** Classe per importar arxius JSON que contenen dades de productes i informació relacionada.
- **Classe ExportFileJSON:**
 - * **Descripció:** Classe per exportar configuracions a arxius JSON que contenen dades de productes i informació relacionada.

2.2 Relació de classes

A la següent taula es mostra com es distribuirà el treball entre els membres de l'equip. Cada membre té assignades diverses classes amb els seus tests unitaris respectius. En el cas de la classe **Supermarket**, aquesta s'ha dividit entre dos membres. A continuació s'especifica com es durà a terme aquesta divisió.

guillem.cabre	marc.penalver	alex.rodriguez.r	marc.teixido
Catalog	Supermarket	OrderingStrategy	Supermarket
ShelvingUnit	User	BruteForce	DomainController
Product	Admin	Approximation	Javadoc
RelatedProduct	ExportFileStrategy	Greedy	L ^A T _E X
ProductTemperature	ImportFileStrategy	DomainController	DomainController-Driver
ExportFileJSON	DomainController	L ^A T _E X	Tests d'integració
ImportFileJSON	DomainController-Driver	DomainController-Driver	ShelvingUnit-Controller
SupermarketData	L ^A T _E X	TopBarController	EditDistribution-ScreenController
RelatedProduct-Serializer	AssetsImageHandler	CatalogController	ShelvingUnit-SwapController
L ^A T _E X	ShelvingUnitMapper	PrimaryButton-Controller	MainScreen-Controller
ToastLabel-Controller	ProductMapper	SecondaryButton-Controller	
Presentation-Controller	RelatedProduct-Mapper	EditKeywords-Controller	
LoginController	ShelvingUnitDTO		
ShelvingUnit-Controller	ProductDTO		
SetTemperature-Controller	RelatedProduct-DTO		
ShelvingUnit-EditionController	PersistenceControllerFactory		
	PersitenceController		
	DomainControllerFactory		
	TopBarController		
	ToastLabel-Controller		

En marc.penalver ha realitzat les funcionalitats relacionades amb els usuaris, crida a les estratègies d'ordenació i gestió de fitxers. D'altra banda, en marc.teixido ha realitzat les funcionalitats relacionades amb les **ShelvingUnits**.

2.3 Estructura JSON

El fitxer JSON utilitzat en l'aplicació “Supermarket Manager” emmagatzema la informació relacionada amb els productes disponibles al supermercat, així com la distribució d'aquests en els prestatges. Aquest fitxer conté els elements següents:

- **shelvingUnitHeight**: Representa l'alçada en unitats per als prestatges del supermercat. És un enter que indica el nombre de files que pot tenir cada prestatgeria.
- **products**: Aquesta és una llista que conté informació detallada sobre cada producte disponible al supermercat. Cada producte té els atributs següents:
 - * **name**: El nom del producte.
 - * **price**: El preu del producte, expressat com un nombre decimal.
 - * **temperature**: La temperatura de conservació necessària per al producte, que pot ser *AMBIENT*, *REFRIGERATED* o *FROZEN*.
 - * **imgPath**: La ruta relativa de la imatge associada al producte.
 - * **keyWords**: Una llista de paraules clau que descriuen el producte i permeten una cerca més eficient.
 - * **relatedProducts**: Una llista de relacions amb altres productes. Cada relació conté:
 - **value**: Un valor numèric que indica el grau de relació entre dos productes (per exemple, en termes de complementarietat).
 - **product1** i **product2**: Els noms dels productes que estan relacionats.
- **distribution**: Aquesta és una llista que defineix la distribució dels productes en els diferents prestatges del supermercat. Cada entrada d'aquesta llista té els atributs següents:
 - * **uid**: Un identificador únic per al prestatge.
 - * **height**: L'alçada del prestatge, que defineix quantes files té.
 - * **temperature**: La temperatura requerida per al prestatge, que pot ser *AMBIENT*, *REFRIGERATED* o *FROZEN*.
 - * **products**: Una llista que conté els noms dels productes col·locats en les diferents posicions del prestatge. Si un espai està buit, es representa amb un valor **null**.

Aquest fitxer JSON és fonamental per mantenir la configuració del supermercat quan es tanca l'aplicació, ja que permet carregar tant el catàleg de productes com la seva distribució de forma eficaç en reobrir el programa.

Chapter 3

Algorismes per a la distribució de productes al Supermercat

3.1 Introducció

Al supermercat s'ha vist que al col·locar certs productes seguits d'altres els clients tendeixen a comprar-los junts i, per tant, augmenten les vendes del supermercat. És per aquest motiu que s'han decidit implementar tres algorismes que permetin distribuir els productes a les prestatgeries de manera que aquestes relacions es maximitzin i, per tant, que així ho facin, també, els beneficis de la botiga.

3.2 Objectiu

El problema a resoldre consisteix en:

- Col·locar el màxim de productes $P = \{p_1, p_2, \dots, p_m\}$ possibles en les prestatgeries $S = \{s_1, s_2, \dots, s_n\}$.
- Respectar les restriccions de temperatura de cada prestatgeria i producte: un producte només es pot col·locar en una prestatgeria si la seva temperatura és compatible amb la de la prestatgeria.
- Maximitzar la **puntuació total**, que es defineix com la suma de les **similituds** entre productes consecutius:

$$\text{Puntuació total} = \sum_{i=1}^{k-1} \text{similitud}(p_i, p_{i+1}),$$

on k és el nombre de productes col·locats.

3.3 Organització de les prestatgeries

El supermercat està organitzat en prestatgeries d'una alçada fixa h , idèntica per a totes les prestatgeries. Cada prestatgeria es pot considerar com una columna que

emmagatzema productes, on la **primera posició** correspon al nivell més alt i la **darrera posició** al nivell més baix.

No obstant, si considerem la visió global del supermercat, aquest es pot representar com una única llista **ciclíca** de posicions, tal i com es pot veure a la figura 3.1. Per recórrer la llista es va horitzontalment fins a arribar a la prestatgeria inicial o final i quan s'arriba a aquest punt es baixa o puja una posició, respectivament. D'aquesta manera, el **nombre total de posicions** on es poden col·locar productes ve donat per la fórmula:

$$\text{Nombre total de posicions} = n \times h$$

on n és el nombre de prestatgeries i h l'alçada de cadascuna.

3.3.1 Adreçament de les prestatgeries i les posicions dels productes

Per identificar a quina prestatgeria pertany un producte, utilitzem la seva posició i dins la llista. Això es calcula amb l'expressió:

$$\text{Prestatgeria actual} = i \bmod n$$

on i és la posició del producte en la llista.

D'altra banda, per accedir al nivell d'alçada del producte dins de la prestatgeria, fem servir la fórmula següent:

$$\text{Alçada actual} = h - 1 - \left\lfloor \frac{i}{n} \right\rfloor$$

Aquesta estructuració permet indexar els productes de manera eficient i mantenir una representació unificada del supermercat tant com una llista global com des de la perspectiva de cada prestatgeria.

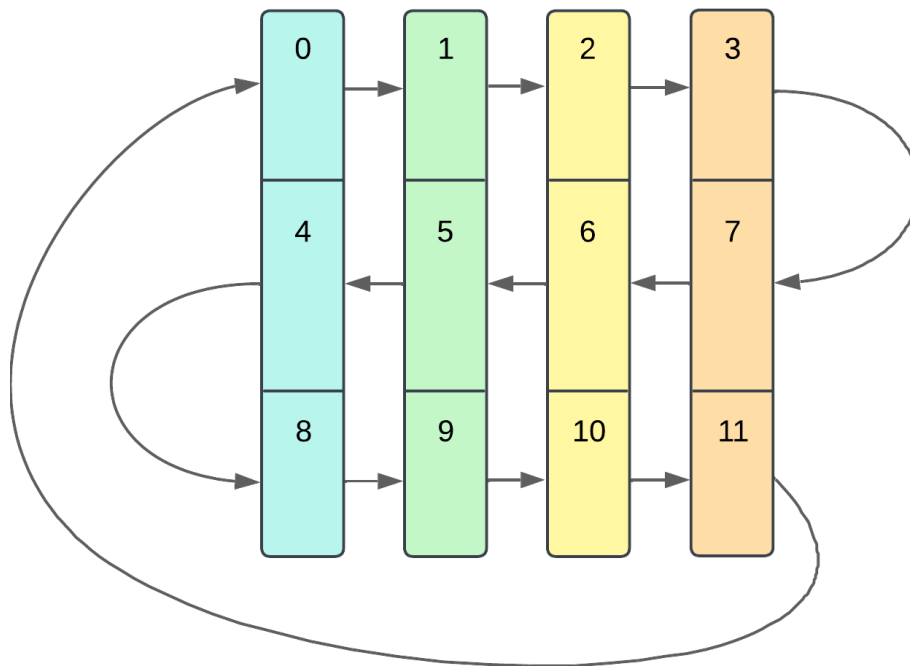


Figura 3.1: Recorregut de les prestatgeries en els algorismes

A continuació, es descriuran els tres diferents algorismes implementats per a la distribució de productes al supermercat.

3.4 Algorisme de força bruta

Aquest algorisme, definit a la classe **BruteForce**, utilitza una estratègia de **backtracking** per trobar la distribució òptima dels productes en les prestatgeries del supermercat. L'objectiu és maximitzar una funció de **puntuació total**, que considera la similitud entre productes col·locats consecutivament i l'eficiència en l'ús de l'espai.

3.4.1 Pseudocodi de l'algorisme

L'algorisme està dividit en dues parts principals: una funció principal que inicialitza els paràmetres i una funció recursiva que col·loca els productes fent servir **backtracking**.

Algorithm 2:	Col·locació òptima de productes amb força bruta
Input:	Prestatgeries S , productes P
Output:	Distribució òptima S^*
1	$bestScore \leftarrow \infty$, $highestSimilarity \leftarrow 0.0$;
2	$optimalDistribution \leftarrow S$;
3	$recursivelyPlaceProducts(0, P, S, \text{null}, 0.0, 0.0)$;
4	return $optimalDistribution$;
5	
6	$recursivelyPlaceProducts(i, P, S, prev, score, similarity)$
7	if ($score \geq bestScore$ and $similarity \leq highestSimilarity$) then
8	return ;
9	if P buida o $i \geq n \times h$ then
10	if $score > bestScore$ and $similarity > highestSimilarity$ then
11	$bestScore \leftarrow score$, $highestSimilarity \leftarrow similarity$;
12	$optimalDistribution \leftarrow S$;
13	return ;
14	$shelf \leftarrow S[i \bmod n]$, $height \leftarrow h - 1 - \lfloor \frac{i}{n} \rfloor$; for $c \in P$ do
15	if $shelf$ pot emmagatzemar c then
16	Afegir c a $shelf$;
17	Crida recursiva amb $nextIndex$;
18	Desfer col·locació de c ;

3.4.2 Complexitat

L'algorisme **BruteForce** utilitza una estratègia de **backtracking** per trobar la distribució òptima dels productes a les prestatgeries. La seva complexitat depèn del nombre de prestatgeries (n), de l'alçada de les prestatgeries (h) i del nombre de productes (m).

Funció principal (orderSupermarket)

La funció principal itera sobre totes les posicions possibles del supermercat ($n \times h$) i, per cada posició, comprova tots els productes disponibles (m). Això dona una complexitat inicial de:

$$O(n \times h \times m)$$

Recursió (recursivelyPlaceProducts)

En cada pas de la recursió:

- **Exploració de totes les combinacions:** Aquest algoritme prova totes les combinacions possibles de col·locació de productes. Això implica que la recursió pot explorar totes les permutacions dels m productes, la qual cosa té una complexitat de $O(m!)$.
- **Comprovació de compatibilitat:** Per cada producte candidat, es comprova si és compatible amb la prestatgeria actual. Aquesta operació es fa m vegades per cada pas de la recursió, afegint un factor multiplicatiu de $O(m)$ a cada pas.

La complexitat total de la recursió és, per tant:

$$O(m \times m!) = O((m + 1)!)$$

Poda

L'algorisme utilitza estratègies de poda per evitar explorar configuracions innecessàries. Les principals estratègies són:

- **Poda basada en la puntuació:** Si la suma de les similituds invertides acumulades supera la millor puntuació actual, vol dir que la configuració actual no pot millorar la solució actual i, per tant, es pot podar.
- **Compatibilitat:** Només es consideren productes que són compatibles amb la prestatgeria actual.

Tot i això, en el pitjor cas, la poda no pot evitar explorar totes les permutacions possibles.

Complexitat global

Combinant els costos de la funció principal i la recursió, la complexitat global en el pitjor cas és:

$$O(n \times h \times m \times m!)$$

Resum

- **Pitjor cas teòric:** $O(n \times h \times m \times m!)$
- **Millor cas pràctic:** En situacions on la poda és efectiva, l'espai de cerca es redueix significativament, però segueix sent exponencial en funció de m .
- **Quan fer-lo servir:** Aquest algorisme és adequat per conjunts de productes petits o amb una forta relació entre productes, ja que en aquests casos la poda pot reduir significativament l'espai de cerca.

3.5 Algorisme Greedy

L'algorisme Greedy, implementat a la classe `GreedyBacktracking`, segueix una estratègia de **selecció de productes** basada en la similitud entre productes consecutius. Aquest algorisme busca maximitzar la puntuació total de la distribució, però no garanteix una solució òptima.

L'estratègia que segueix és semblant a la de l'algorisme de força bruta, amb la diferència que en cada pas, selecciona el producte que **millor similitud** té amb el producte col·locat a la posició anterior a l'actual, en comptes de provar totes les combinacions possibles.

3.5.1 Pseudocodi de l'algorisme

La part diferencial de l'algorisme respecte al `BruteForce` es troba en la funció `findBestProductToPlace`, que selecciona el producte amb la millor similitud amb el producte anterior. Aquesta funció retorna el producte seleccionat i la similitud associada.

Algorithm 3: Col·locació òptima amb backtracking greedy

```
1 findBestProductToPlace( $i, P, S, prev$ )  $shelf \leftarrow S[i \bmod n]$ ;
2  $bestProduct \leftarrow \text{null}, bestSimilarity \leftarrow 0$ ;
3 foreach  $p \in P$  do
4   if  $shelf$  pot emmagatzemar  $p$  then
5      $similaritat \leftarrow \text{similitud}(prev, p)$ ;
6     if  $similaritat > bestSimilarity$  then
7        $bestSimilarity \leftarrow similaritat$ ;
8        $bestProduct \leftarrow p$ ;
9 return ( $bestProduct, bestSimilarity$ );
```

3.5.2 Complexitat

La funció principal itera sobre totes les posicions possibles del supermercat ($n \times h$) i, per cada posició, comprova tots els productes disponibles (m). Això dona una complexitat inicial de:

$$O(n \times h \times m)$$

Recursió (`recursivelyPlaceProducts`)

En cada pas de la recursió:

- **Selecció del millor producte:** Es fa mitjançant la funció `findBestProductToPlace`, que revisa tots els productes restants (m) per seleccionar el millor. Això té una complexitat de $O(m)$.
- **Crides recursives:** L'algorisme pot fer fins a m crides recursives, ja que cada producte es col·loca una vegada. En el pitjor cas, es podrien explorar totes les configuracions possibles de col·locació.

Per tant, el cost de la recursió en el pitjor cas és:

$$O(m!)$$

Poda

L'algorisme incorpora una estratègia de **poda greedy** per evitar explorar configuracions poc prometedores. Això redueix el nombre de configuracions explorades en la pràctica, però no elimina el factor $m!$ en el pitjor cas, ja que encara podria caldre explorar totes les opcions en escenaris adversos (per exemple, si la similitud entre productes és molt baixa i cal provar totes les opcions). A més, s'aplica també la poda basada en la puntuació, com en l'algorisme de força bruta.

Combinant els costos de la funció principal i la recursió, tenim la següent complexitat global en el pitjor cas:

$$O(n \times h \times m \times m!)$$

No obstant això, gràcies a la doble poda:

- S'eviten moltes de les ramificacions que no aporten solucions millors.
- Si la majoria de productes són incompatibles amb moltes prestatgeries, el nombre de combinacions que realment s'exploren és molt menor.
- Si aviat es troben solucions amb bona similitud i puntuació, això limita encara més l'exploració de configuracions pitjors.

En conseqüència, tot i que la complexitat teòrica segueixi sent factorial, la **complexitat efectiva es redueix significativament** gràcies a la poda. La quantitat real de configuracions explorades depèn fortament de la qualitat d'aquesta poda i de la distribució dels productes, el que fa que, en escenaris pràctics, l'algorisme sigui molt més eficient del que la seva complexitat teòrica suggereix.

Resum

Aquest algorisme és més eficient que un de força bruta pur, però encara és costós per a valors grans de m . A la pràctica, però, s'ha vist que és capaç de trobar solucions properes a la òptima en un temps significativament més raonable que el de força bruta per a la majoria de casos d'ús.

3.6 Algorisme d'aproximació

L'algorisme d'aproximació implementat en la classe `Approximation` segueix una estratègia de **cerca local** per trobar una solució aproximada al problema de la distribució de productes al supermercat. Aquest algorisme busca maximitzar la puntuació total de la distribució, però al igual que el **Greedy** no garanteix una solució òptima.

3.6.1 Simulated Annealing

Simulated Annealing és un mètode d'optimització inspirat en el procés de refredament i cristal·lització dels metalls. Aquest consisteix en una cerca aleatòria que accepta moviments no òptims amb una probabilitat que disminueix amb el temps. Això permet explorar l'espai de cerca de manera més eficient i **evitar quedar atrapat** en òptims locals.

3.6.2 Pseudocodi de l'algorisme

Algorithm 4: Simulated Annealing per l'ordenació de prestatgeries

Input: Prestatgeries inicials S , productes P
Output: Distribució òptima S^*

// Paràmetres de Simulated Annealing

```
1 steps  $\leftarrow$  100000;  
2  $k \leftarrow 5.0$ ,  $\lambda \leftarrow 0.99$ ;  
3  $T \leftarrow 1000.0$ ;  
4 currentS  $\leftarrow$  generarSolucióInicialGreedy( $S$ ,  $P$ );  
5 unplacedP  $\leftarrow$  eliminarProductesJaCol·locats(currentS,  $P$ );  
6 resultWithGreedyInitial  $\leftarrow$  simulatedAnnealing(currentS, unplacedP, steps,  
   T,  $k$ ,  $\lambda$ );  
7 bestShelves  $\leftarrow$  resultWithGreedyInitial.getKey();  
8 bestScore  $\leftarrow$  resultWithGreedyInitial.getValue();  
9 for  $i \leftarrow 0$  to 4 do  
10   currentS  $\leftarrow$  generarSolucióInicialAleatoria( $S$ ,  $P$ );  
11   unplacedP  $\leftarrow$  eliminarProductesJaCol·locats(currentS,  $P$ );  
12   resultWithRandomInitial  $\leftarrow$  simulatedAnnealing(currentS, unplacedP,  
     steps,  $T$ ,  $k$ ,  $\lambda$ );  
13   if resultWithRandomInitial.getValue() > bestScore then  
14     bestShelves  $\leftarrow$  resultWithRandomInitial.getKey();  
15     bestScore  $\leftarrow$  resultWithRandomInitial.getValue();  
16  
   // Simulated Annealing  
17   simulatedAnnealing(currentS,  $P$ , steps,  $T$ ,  $k$ ,  $\lambda$ )  
18   currentScore  $\leftarrow$  calculateTotalSimilarity(currentS);  
19   highestScore  $\leftarrow$  currentScore;  
20   optimalDistribution  $\leftarrow$  currentS;  
21   for step  $\leftarrow 0$  to steps do  
22     // Escollir operador aleatori  
23     operatorChoice  $\leftarrow$  random(0, 2);  
24     if operatorChoice = 0 then  
25       neighborS  $\leftarrow$  swapTwoProducts(currentS);  
26     else  
27       if operatorChoice = 1 then  
28         neighborS  $\leftarrow$  moveProductToEmptyPosition(currentS);  
29       else  
30         neighborS, neighborUnplacedProducts  $\leftarrow$   
31           swapWithUnplacedProduct(currentS,  $P$ );  
       // Calcular puntuació del veí  
32     neighborScore  $\leftarrow$  calculateTotalSimilarity(neighborS);  
33      $\Delta \leftarrow$  neighborScore - currentScore;
```

3.6.3 Estratègia de generació de solucions inicials

Per aproximar-se el màxim possible a l'òptim global, l'algorisme utilitza una estratègia de generació de solucions inicials que combina un enfocament **greedy** amb un enfocament **aleatori**:

L'algorisme genera una primera solució inicial fent servir l'algorisme **Greedy** anterior i, a continuació, executa l'algorisme **Simulated Annealing** amb aquesta solució com a punt de partida. Posteriorment, executa l'algorisme amb 4 altres solucions inicials, generades aleatòriament per explorar més configuracions, i finalment tria la millor distribució obtinguda d'aquestes 4 i la greedy.

D'aquesta manera augmentem la probabilitat d'arribar a una solució més òptima.

3.6.4 Paràmetres de Simulated Annealing

El Simulated Annealing depen de diversos paràmetres que cal ajustar per obtenir bons resultats. Els paràmetres són:

- **Temperatura inicial (T)**: La temperatura inicial determina la probabilitat d'acceptar solucions pitjors. Una temperatura més alta permet explorar més configuracions, però també pot fer que l'algorisme es quedi atrapat en òptims locals.
- **Nombre de passos ($steps$)**: El nombre de passos determina la durada de l'algorisme. Un nombre més gran permet explorar més configuracions, però també augmenta el temps d'execució.
- **Factor de refredament (λ)**: El factor de refredament determina com disminueix la temperatura en cada iteració. Un factor més petit permet explorar més configuracions, però també pot fer que l'algorisme es quedi atrapat en òptims locals.
- **Factor de Boltzmann (k)**: El factor de Boltzmann determina la sensibilitat de l'algorisme a les diferències de puntuació. Un valor més gran permet acceptar solucions pitjors amb més facilitat, però també pot fer que l'algorisme es quedi atrapat en òptims locals.

Els diferents paràmetres han estat triats a partir de proves empíriques per trobar una combinació que funcioni bé en la pràctica. Un estudi estadístic més profund podria permetre ajustar millor els paràmetres per a un conjunt de dades específic en un futur.

3.6.5 Operadors

Operador swapTwoProducts

- **Descripció**: Aquest operador intercanvia dos productes ja col·locats a les prestatgeries.
- **Opcions disponibles**:

- * Si hi ha p posicions ocupades, el nombre de combinacions possibles per intercanviar dos productes és:

$$\binom{p}{2} = \frac{p \cdot (p-1)}{2}$$

- **Factor de ramificació:**

$$O(p^2), \quad \text{on } p \leq n \times h \text{ és el nombre de posicions ocupades.}$$

Operador `moveProductToEmptyPosition`

- **Descripció:** Mou un producte col·locat a una posició buida compatible dins les prestatgeries.
- **Opcions disponibles:**
 - * Es poden seleccionar p productes ocupats i moure'ls a e posicions buides. En el pitjor cas, si totes les posicions són buides o ocupades:

$$p \cdot e = (n \times h) \cdot (n \times h - p)$$

- **Factor de ramificació:**

$$O(p \cdot e), \quad \text{on } e \leq n \times h - p \text{ és el nombre de posicions buides.}$$

Operador `swapWithUnplacedProduct`

- **Descripció:** Intercanvia un producte col·locat amb un producte no col·locat.
- **Opcions disponibles:**
 - * Es poden seleccionar p productes col·locats i intercanviar-los amb u productes no col·locats. El nombre de combinacions és:

$$p \cdot u$$

- **Factor de ramificació:**

$$O(p \cdot u), \quad \text{on } u \leq m - p \text{ és el nombre de productes no col·locats.}$$

3.6.6 Complexitat

L'algorisme **Simulated Annealing** busca una solució aproximada per a l'ordenació òptima dels productes a les prestatgeries. La seva complexitat depèn del nombre de prestatgeries (n), de l'alçada de les prestatgeries (h), del nombre de productes (m) i dels diferents paràmetres propis de l'algorisme.

Complexitat de cada pas del bucle

En cada iteració del bucle principal:

- **Selecció de l'operador:** Un operador es selecciona aleatòriament entre tres opcions. Aquest pas té una complexitat constant: $O(1)$.
- **Aplicació de l'operador:**
 - * **swapTwoProducts:** Es seleccionen dues posicions de productes i es comprova la compatibilitat. Això implica revisar com a màxim totes les posicions ocupades, que és $O(n \times h)$.
 - * **moveProductToEmptyPosition:** Es selecciona un producte i una posició buida. Això implica revisar totes les posicions ocupades i buides, també $O(n \times h)$.
 - * **swapWithUnplacedProduct:** Es selecciona un producte col·locat i un de no col·locat. Revisar compatibilitats amb els productes no col·locats té una complexitat $O(m)$, mentre que revisar les posicions col·locades és $O(n \times h)$.

El cost màxim per qualsevol operador és $O(n \times h + m)$.

- **Càlcul de la puntuació:** La puntuació de la distribució es calcula sumant les similituds de tots els productes col·locats, amb un cost $O(n \times h)$.
- **Acceptació del veí:** Calcular la probabilitat d'acceptació i decidir si es pren el veí té un cost constant: $O(1)$.

En total, el cost d'una iteració és:

$$O(n \times h + m)$$

Complexitat total del bucle principal

El bucle principal s'executa *steps* vegades. Per tant, la complexitat total del bucle és:

$$O(steps \times (n \times h + m))$$

Complexitat de la generació de la solució inicial

La solució inicial aleatòria es genera col·locant productes de manera aleatòria fins que s'omplen les posicions. Aquest procés té un cost:

$$O(n \times h \times m)$$

ja que cada producte pot ser comprovat fins a m vegades en el pitjor cas. Com es generen 4 solucions inicials aleatòries, la complexitat total és 4 vegades l'anterior.

Pel que fa a la solució inicial generada amb l'algorisme Greedy, la complexitat és la descrita en la secció de l'algorisme Greedy.

Tot plegat, la complexitat de la generació de la solució inicial és la suma de les dues anteriors.

Complexitat global

Combinant la generació de la solució inicial i el bucle principal, la complexitat total de l'algorisme és:

$$O(4 * n \times h \times m + \text{complexitatGreedy} + \text{steps} \times (n \times h + m))$$

Resum

- **Pitjor cas:** $O(4 * n \times h \times m + \text{complexitatGreedy} + \text{steps} \times (n \times h + m))$.
- **Complexitat pràctica:** Els valors de *steps* i λ controlen la durada del procés i poden ajustar-se per equilibrar entre precisió i temps d'execució.

Aquest algorisme és més eficient que les alternatives de força bruta, però no garanteix trobar la solució òptima, ja que depèn de la qualitat de les solucions veïnes i del procés de refredament.

3.7 Conclusions

Els tres algorismes implementats per a la distribució de productes al supermercat presenten avantatges i inconvenients segons les necessitats de l'aplicació. A continuació, es resumeixen les característiques de cada algorisme:

- **Força bruta:** Aquest algorisme és el més precís, ja que explora totes les configuracions possibles. No obstant, la seva complexitat exponencial el fa prohibitivament lent per a valors grans de m . És adequat per conjunts de productes petits o amb una forta relació entre productes.
- **Greedy:** L'algorisme Greedy és més eficient que el de força bruta, però encara té una complexitat alta en el pitjor cas. La seva estratègia de selecció de productes basada en la similitud permet trobar solucions ràpidament, però no garanteix la solució òptima.
- **Simulated Annealing:** Aquest algorisme busca una solució aproximada mitjançant una cerca local. La seva complexitat és més baixa que la de força bruta i Greedy, però la qualitat de la solució depèn dels paràmetres de l'algorisme i de la qualitat de les solucions veïnes. És adequat per trobar solucions ràpides i acceptables en conjunts de dades grans.

Donat que l'aplicació requereix una distribució no només òptima, sinó també eficient, a mesura que el nombre de productes creix, l'algorisme de força bruta esdevé inviable. Per aquest motiu, s'ha optat per implementar l'algorisme de Simulated Annealing, que permet trobar una solució aproximada en un temps raonable. Aquest algorisme és capaç de trobar solucions acceptables per a conjunts de dades grans, tot i que no garanteix la solució òptima. Igualment, aquesta solució pot ser estudiada en un futur mitjançant l'ajust dels paràmetres de l'algorisme o l'ús d'altres tècniques d'optimització.

Chapter 4

Descripció de l'Arquitectura en 3 Capes

4.1 Introducció

Aquest apartat del document té com a objectiu proporcionar una visió general del projecte, exposant les decisions d'arquitectura i la comunicació entre les capes. Es recomana seguir aquest apartat amb els diferents diagrames del model per obtenir una millor comprensió de l'escrit.

4.2 Capa de Presentació

Per a la capa de presentació, s'ha optat pel patró Model-Vista-Controlador. El **Model** és l'encarregat de gestionar la lògica de negoci i la persistència de les dades. Les **Vistes** són els diferents arxius FXML i CSS que presenten la informació a l'usuari. Finalment, els **Controladors** són responsables de gestionar la interacció entre el model i la vista.

Hi ha un controlador de presentació principal que s'ocupa de gestionar la navegabilitat entre les vistes, inicialitzar i tancar l'aplicació, així com inicialitzar els controladors específics de cada vista. Aquests controladors són els encarregats d'actualitzar les vistes amb la informació del model, gestionar les peticions dels usuaris (ja sigui per a consultes o modificacions de dades), tot mitjançant la comunicació amb el controlador de domini. També són responsables de mostrar missatges informatius per mantenir l'usuari al corrent de l'estat de l'aplicació.

4.3 Capa de Domini

La capa de domini s'encarrega de gestionar la lògica de negoci. Proporciona la informació requerida per a la capa de presentació i manté la comunicació amb la capa de persistència. Per al controlador de domini, s'ha optat pel patró cas d'ús, on cada cas d'ús especificat a les especificacions de l'aplicació té el mètode corresponent

al controlador de domini. Aquest serà responsable de cridar a les classes concretes per gestionar la lògica de cada cas d'ús.

També és rellevant destacar que la classe **Supermarket** recull tota la informació de la sessió d'un supermercat i s'encarrega de comunicar-se amb el controlador de persistència per exportar o importar les dades.

4.4 Capa de Persistència

Atès que el projecte té la restricció de no poder utilitzar una base de dades com PostgreSQL, s'ha ideat un sistema de persistència per al **SupermarketManager**. Aquest sistema utilitza un únic fitxer `.json` per emmagatzemar totes les dades de la sessió.

El sistema de fitxers es divideix de la següent manera:

- **default.json**: Aquest fitxer es troba en els recursos de la classe **Main.class** i es carrega automàticament quan es realitza un *log in* amb èxit. Emmagatzema la informació del supermercat actual, com la distribució i els productes disponibles en aquell moment.
- **Assets**: En aquesta carpeta es troben tots els fitxers relacionats amb les imatges utilitzades a la capa de presentació. S'hi inclou una imatge d'error per defecte, així com carpetes amb les icones de temperatures i totes les imatges dels productes generades per l'usuari.
- **Altres fitxers**: L'usuari també pot carregar fitxers `.json` per provar diferents configuracions amb productes diversos. No és necessari reescriure el **default.json**, ja que existeix un cas d'ús per importar un supermercat complet, que inclou tant el catàleg de productes com la seva distribució.

Un cop carregada la configuració del supermercat al sistema, l'usuari pot fer modificacions i guardar els canvis mitjançant les funcionalitats de *Save* i *Save As*. A continuació es detallen com funcionen cadascuna d'aquestes funcionalitats:

- **Save**: Exporta la configuració actual del sistema, incloent les modificacions realitzades per l'administrador, al fitxer **default.json** (que sobreescriurà).
- **Save As**: Obre un menú que permet a l'usuari seleccionar un directori i especificar el nom del fitxer amb l'extensió `.json` on es desarà la configuració del supermercat. Aquesta funcionalitat ofereix la possibilitat de guardar la configuració amb un altre nom o en una ubicació diferent.