HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

LAPTOP CONSULTATION SYSTEM INTERGRATING INFORMATION RETRIEVAL & LARGE LANGUAGE MODEL

Group: 7

Team members:

Nguyen Tuan Duong – 20235924

Nguyen Duc Anh – 20235890

Hoang Quoc Cuong –20235903

Nguyen Minh Duc –20235915

Pham Minh Duc –20235918

ONE LOVE. ONE FUTURE.

OUTLINE

I. Introduction

Project Objectives:

- Develop a system for collecting laptop data from the web
- Efficiently store and manage data using PostgreSQL
- Provide basic product filtering functionalities
- Enable intelligent product search based on user queries using embedding techniques and AI.

Significance:

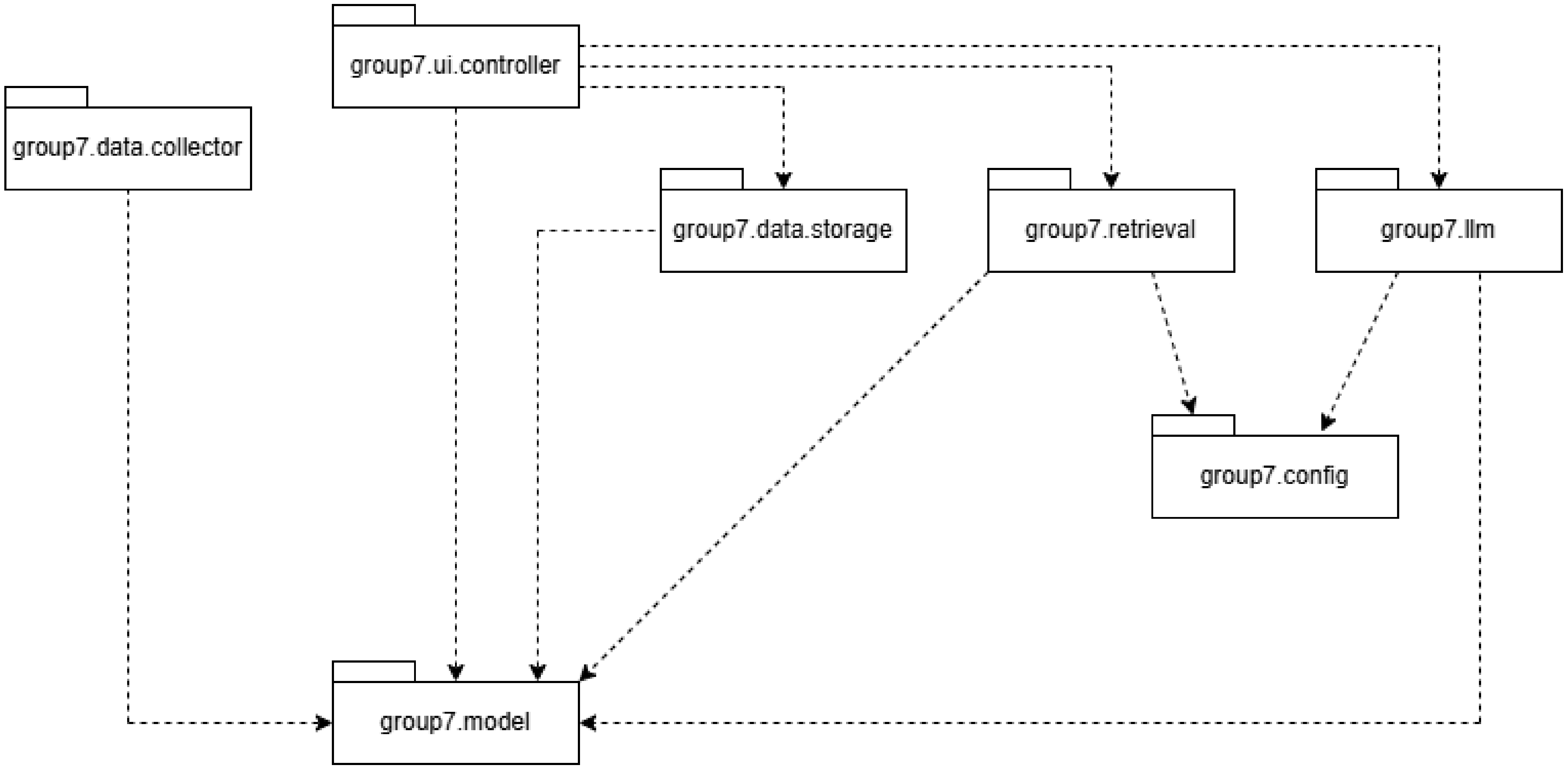
- Help users find suitable laptops based on their needs.
- Automate the process of collecting and analyzing product data.
- Apply modern AI technologies to provide smart product recommendations.

II. System architecture overview

System Architecture Description:

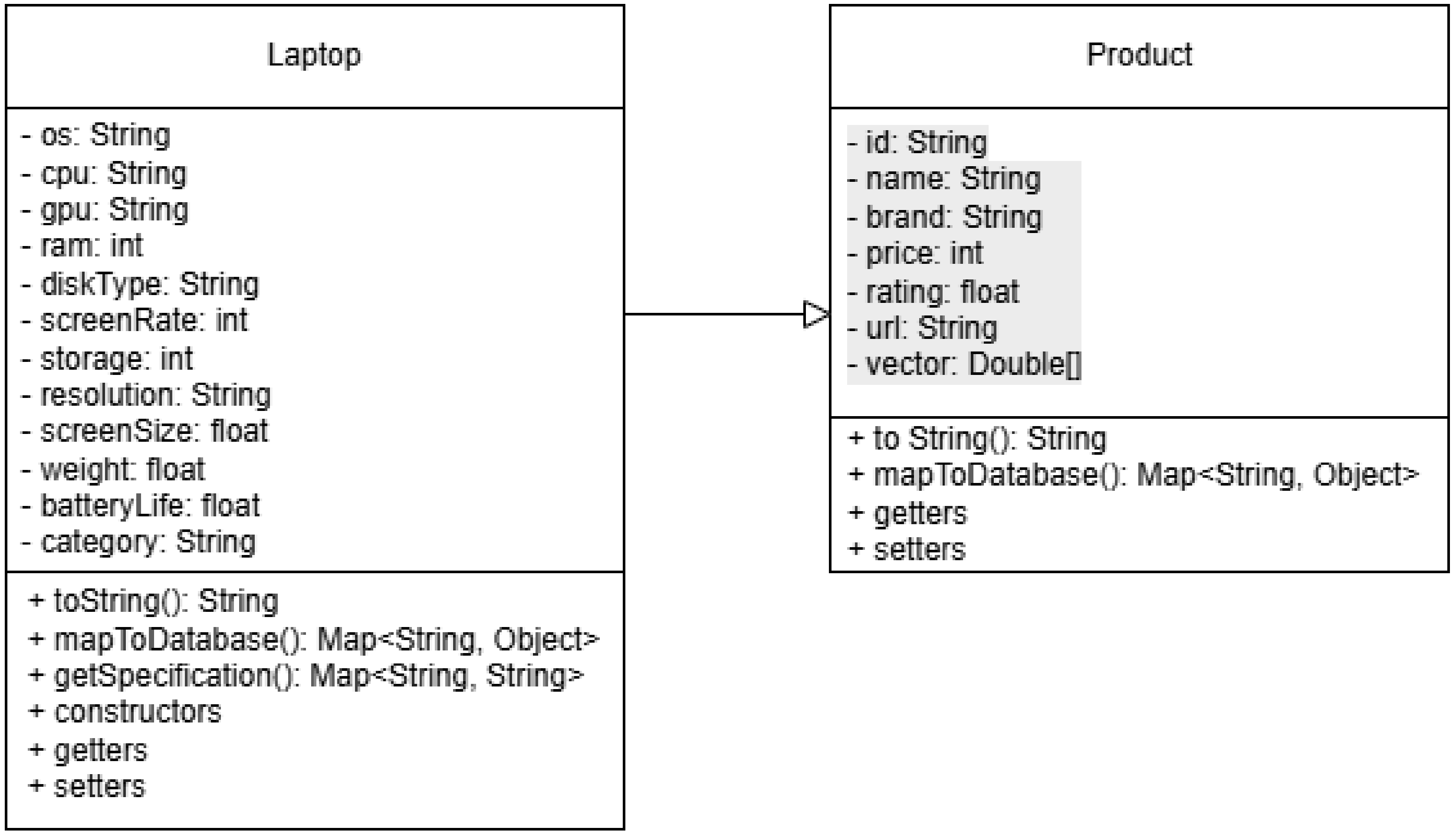
- **Data Collection:** Utilizes Selenium (`DataCollector`) to scrape data from laptop
- **Search and Recommendation:** Combines vector embeddings (`EmbeddingService`) and an AI model (`MistralClient`) to process user queries.
- **Product Modeling:** The `Laptop` class inherits from `Product`, storing detailed technical specifications.

II. System architecture overview



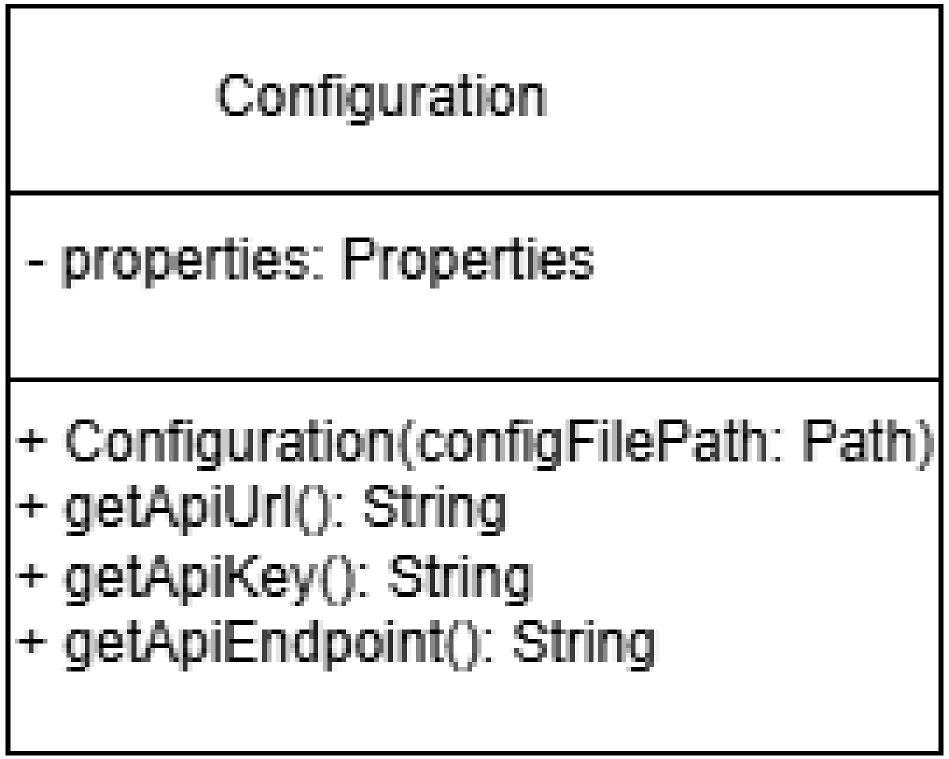
II. System architecture overview

Package: `group7.model`



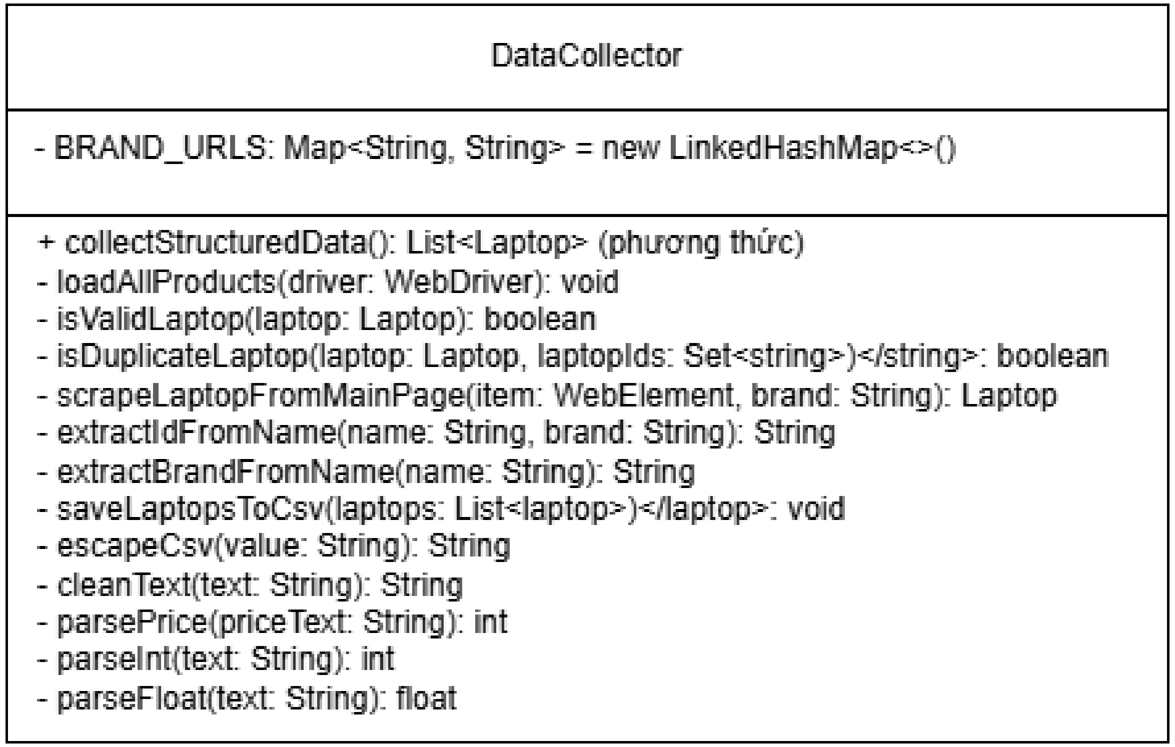
II. System architecture overview

Package: **`group7.config`**



II. System architecture overview

Package: `group7.data.collector`



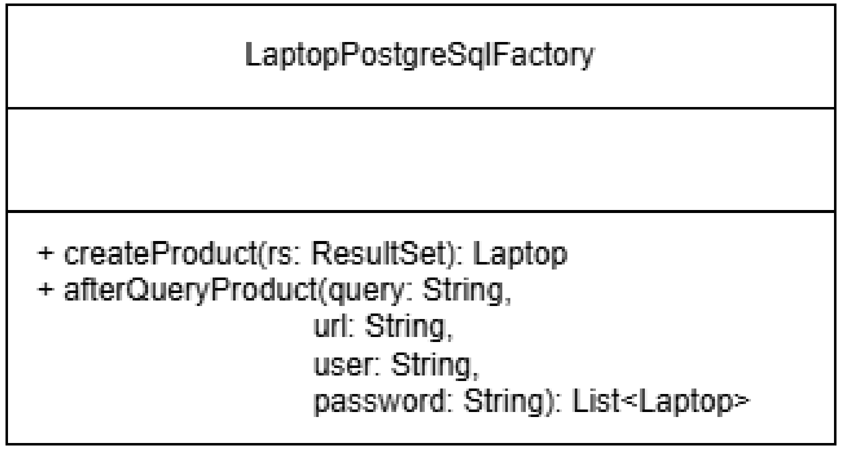
DataCollector

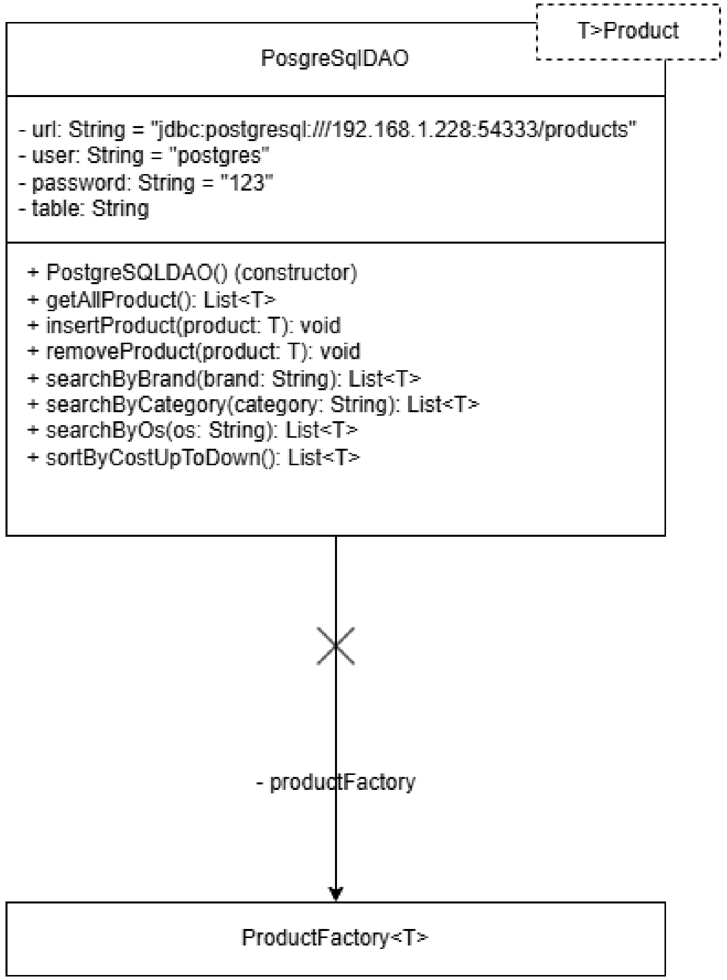
- BRAND_URLS: Map<String, String> = new LinkedHashMap<>()

+ collectStructuredData(): List<Laptop> (phương thức)
- loadAllProducts(driver: WebDriver): void
- isValidLaptop(laptop: Laptop): boolean
- isDuplicateLaptop(laptop: Laptop, laptopIds: Set<string>)</string>: boolean
- scrapeLaptopFromMainPage(item: WebElement, brand: String): Laptop
- extractIdFromName(name: String, brand: String): String
- extractBrandFromName(name: String): String
- saveLaptopsToCsv(laptops: List<laptop>)</laptop>: void
- escapeCsv(value: String): String
- cleanText(text: String): String
- parsePrice(priceText: String): int
- parseInt(text: String): int
- parseFloat(text: String): float

II. System architecture overview

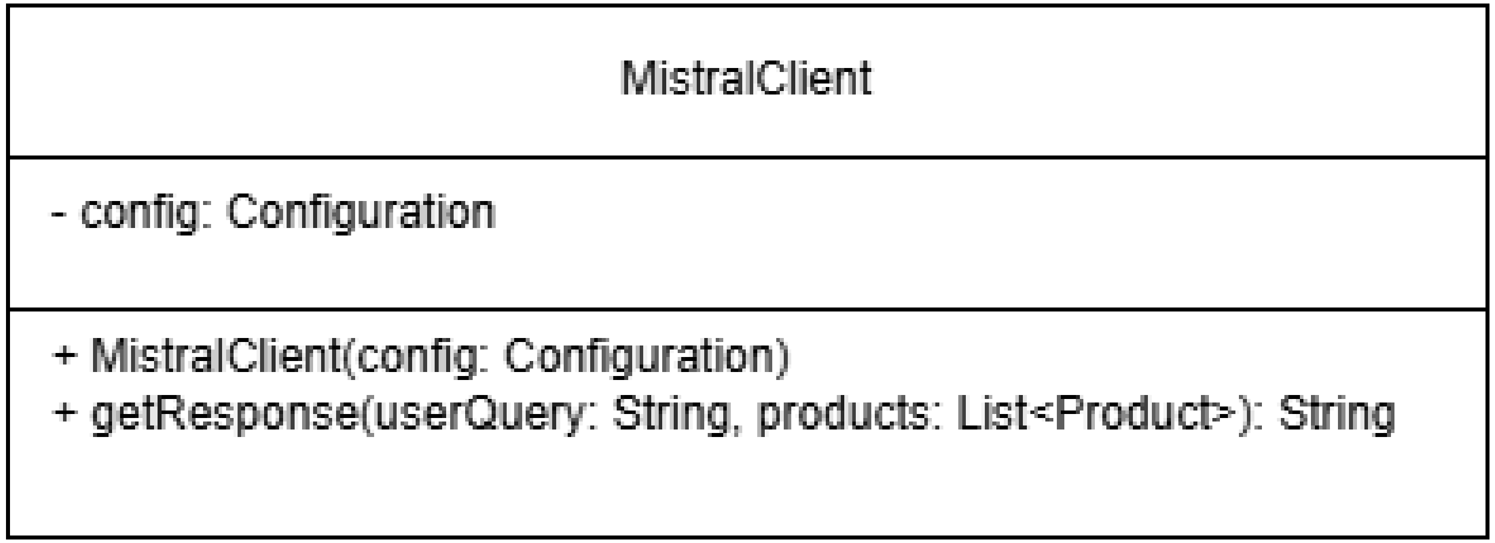
Package: `group7.data.storage`


LaptopPostgreSqlFactory
+ createProduct(rs: ResultSet): Laptop
+ afterQueryProduct(query: String,
url: String,
user: String,
password: String): List<Laptop>


T>Product
PosgreSqlDAO
- url: String = "jdbc:postgresql:///192.168.1.228:54333/products"
- user: String = "postgres"
- password: String = "123"
- table: String
+ PostgreSQLDAO() (constructor)
+ getAllProduct(): List<T>
+ insertProduct(product: T): void
+ removeProduct(product: T): void
+ searchByBrand(brand: String): List<T>
+ searchByCategory(category: String): List<T>
+ searchByOs(os: String): List<T>
+ sortByCostUpToDown(): List<T>
- productFactory
ProductFactory<T>

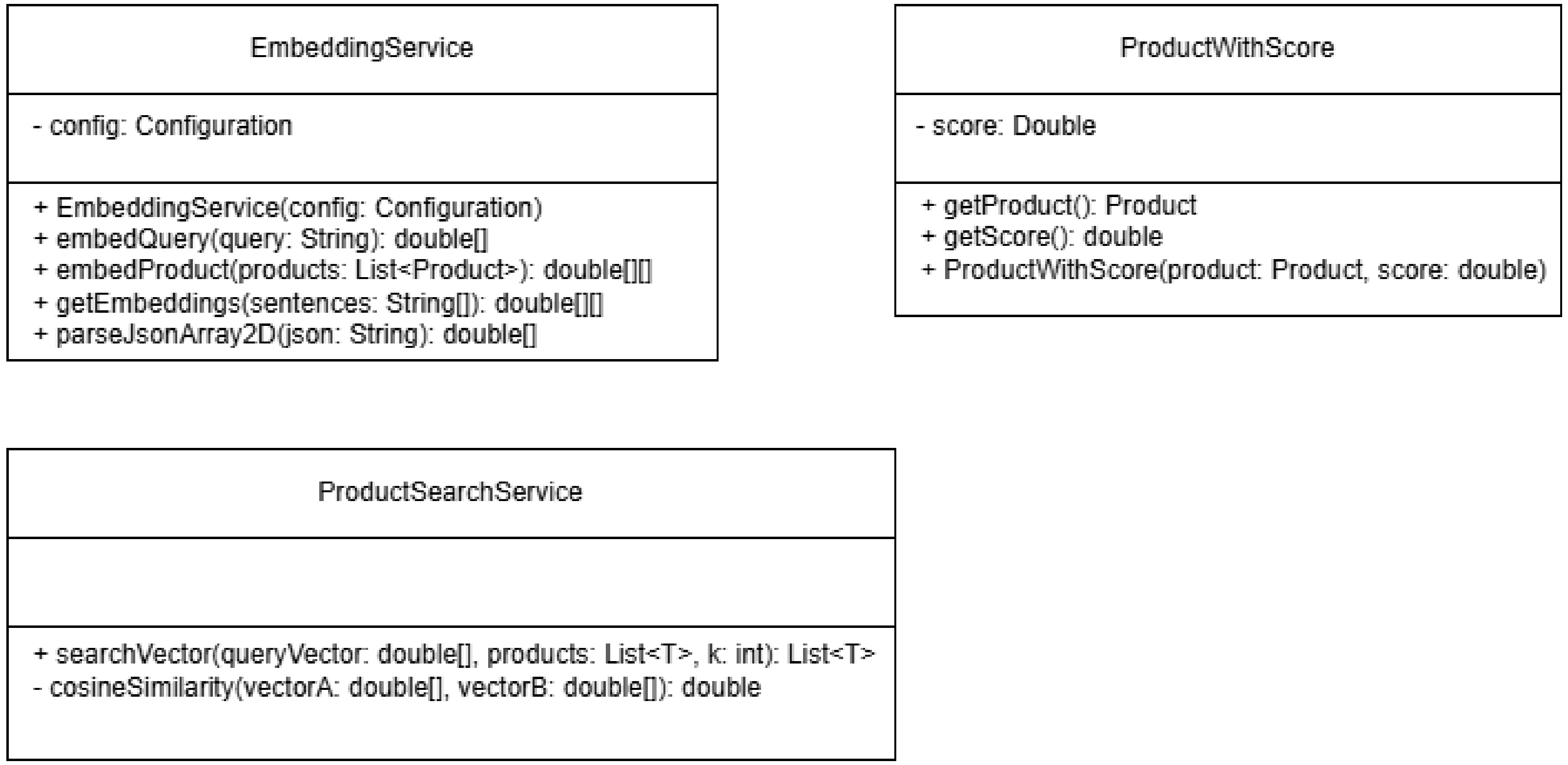
II. System architecture overview

Package: **`group7.llm`**


MistralClient
- config: Configuration
+ MistralClient(config: Configuration)
+ getResponse(userQuery: String, products: List<Product>): String

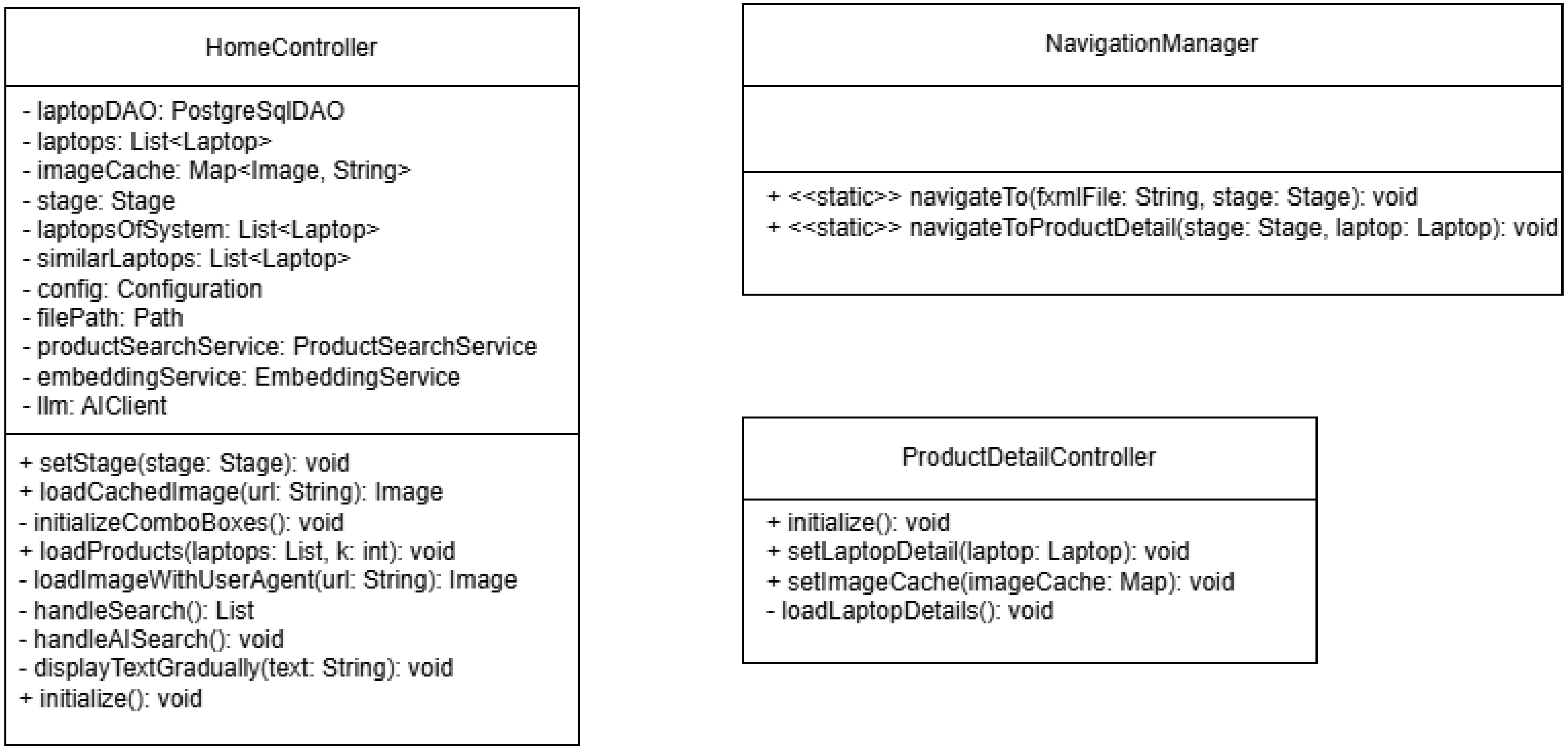
II. System architecture overview

Package: `group7.retrieval`


EmbeddingService
- config: Configuration
+ EmbeddingService(config: Configuration)
+ embedQuery(query: String): double[]
+ embedProduct(products: List<Product>): double[][]
+ getEmbeddings(sentences: String[]): double[][]
+ parseJsonArray2D(json: String): double[]
ProductWithScore
- score: Double
+ getProduct(): Product
+ getScore(): double
+ ProductWithScore(product: Product, score: double)
ProductSearchService
+ searchVector(queryVector: double[], products: List<T>, k: int): List<T>
- cosineSimilarity(vectorA: double[], vectorB: double[]): double

II. System architecture overview

Package: **`group7.ui.controller`**



HomeController

- laptopDAO: PostgreSqlDAO
- laptops: List<Laptop>
- imageCache: Map<Image, String>
- stage: Stage
- laptopsOfSystem: List<Laptop>
- similarLaptops: List<Laptop>
- config: Configuration
- filePath: Path
- productSearchService: ProductSearchService
- embeddingService: EmbeddingService
- llm: AIClient

+ setStage(stage: Stage): void
+ loadCachedImage(url: String): Image
- initializeComboBoxes(): void
+ loadProducts(laptops: List, k: int): void
- loadImageWithUserAgent(url: String): Image
- handleSearch(): List
- handleAISearch(): void
- displayTextGradually(text: String): void
+ initialize(): void

NavigationManager

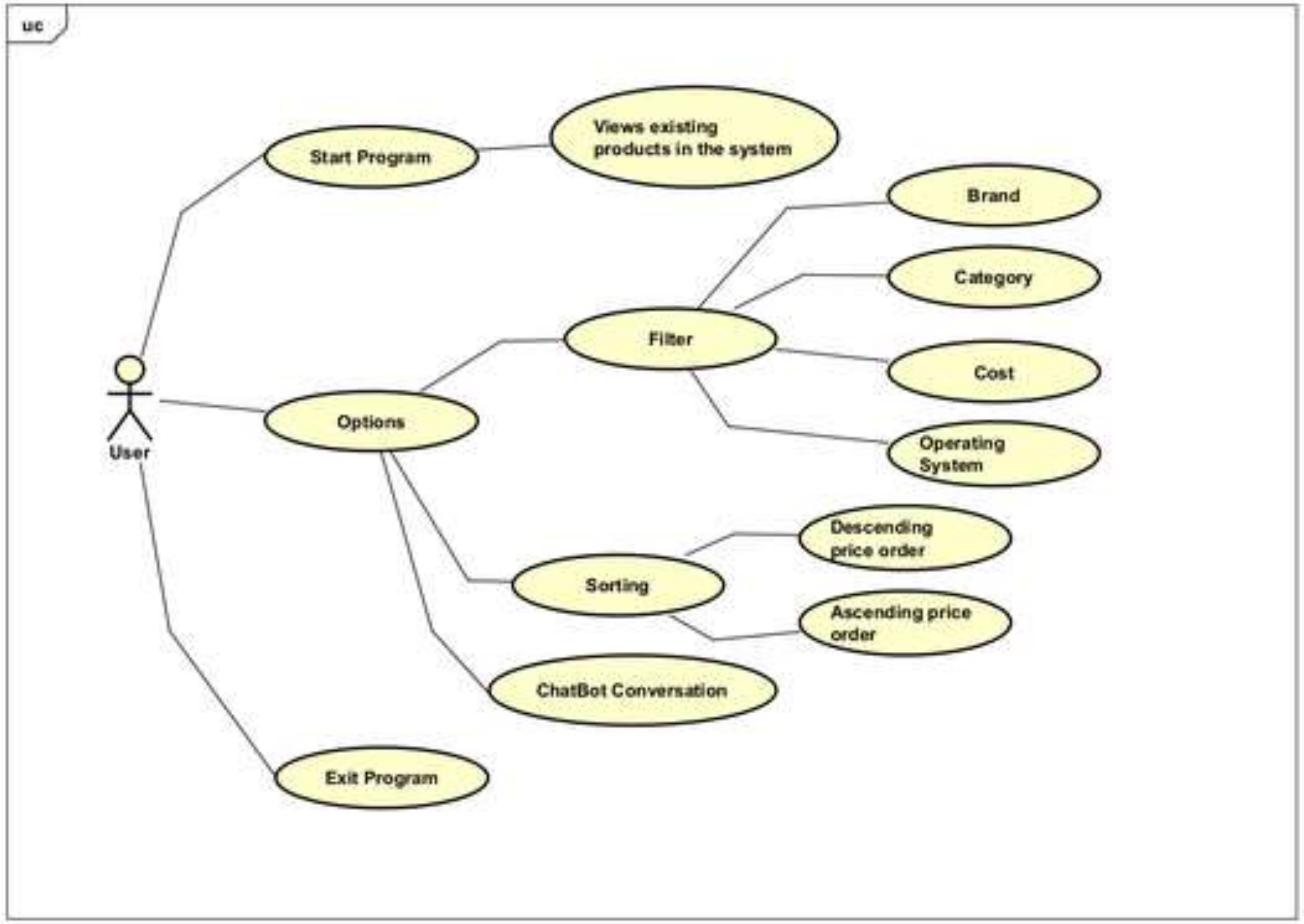
+ <<static>> navigateTo(fxmlFile: String, stage: Stage): void
+ <<static>> navigateToProductDetail(stage: Stage, laptop: Laptop): void

ProductDetailController

+ initialize(): void
+ setLaptopDetail(laptop: Laptop): void
+ setImageCache(imageCache: Map): void
- loadLaptopDetails(): void

II. System architecture overview

User case diagram:



III. Main components of the system

1. Product Class and Laptop Class

The abstract class **`Product`** serves as the foundation for all products in the system.
Basis attributes: `id, name, brand, price, rating and url.`
Abstract methods: `mapToDataBase()` and `getVector()` to support storage and search functionalities.

Characteristic:

- **Abstraction:** Allows easy extension to other product types (e.g., phones, tablets).
- **Reusability:** Common attributes are defined once, reducing code duplication.
- **Easy integration:** Supports data mapping for database storage.

III. Main components of the system

1. Product Class and Laptop Class

The **`Laptop`** class inherits from the **`Product`** class and extends it with laptop-specific attributes.

It implements the `mapToDatabase()` method to map laptop information into the database.

Additionally, it provides the `getSpecification()` method to return technical specifications in the form of a map.

III. Main components of the system

2. DataCollector Class

Uses Selenium WebDriver to scrape data from websites

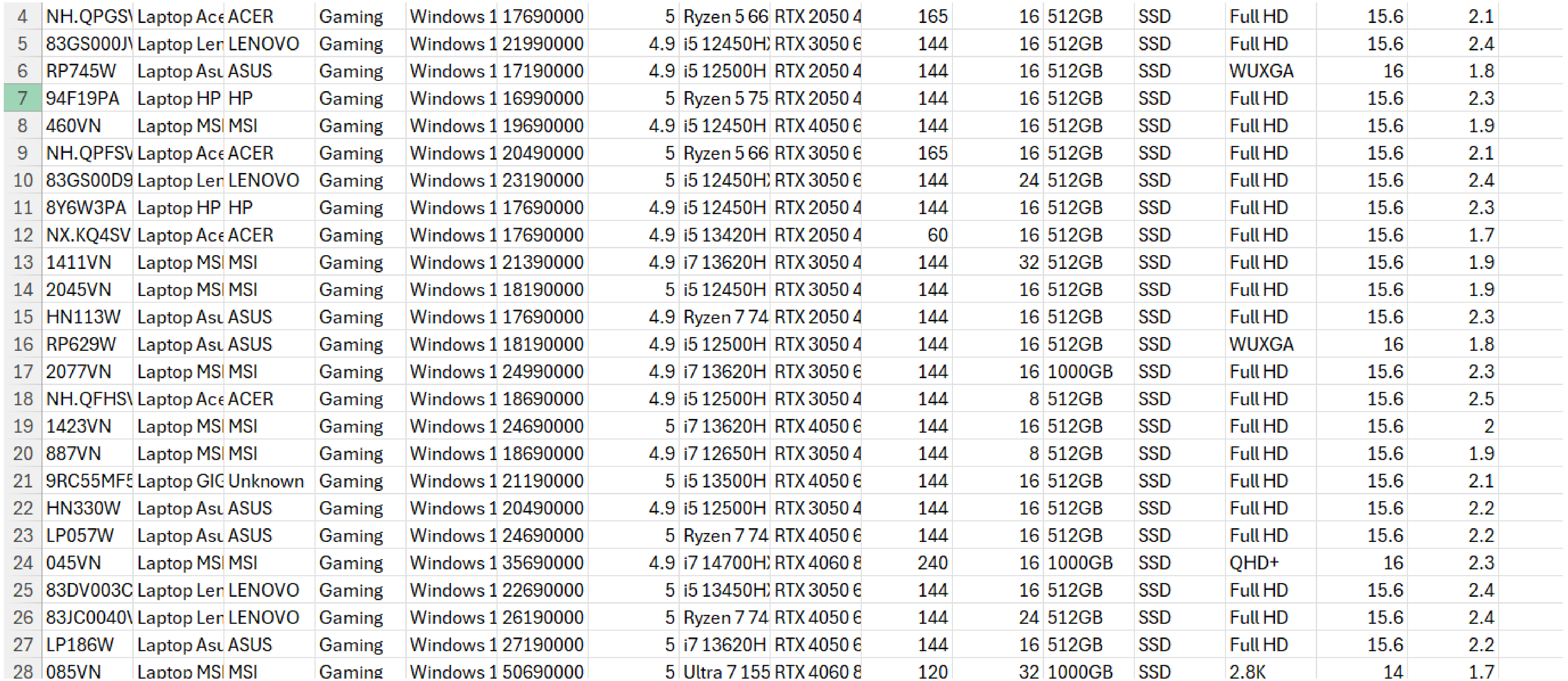
1. Access brand URLs from BRAND_URLS.
2. Auto-scroll and click "Load More" to load all products (`loadAllProducts`).
3. Extract product data from HTML (`scrapeLaptopFromMainPage`).
4. Validate and check for duplicates (`isValidLaptop, isDuplicateLaptop`).
5. Save to CSV and return Laptop objects (`saveLaptopsToCsv`).

Characteristic:

- High automation
- Robust exception handling (e.g., network errors, HTML changes)
- High performance (with processing time measurement).
- Flexible and adaptable design

III. Main components of the system

2. DataCollector Class



4	NH.QPGS\	Laptop Ace	ACER	Gaming	Windows 1	17690000	5	Ryzen 5 66	RTX 2050 4	165	16	512GB	SSD	Full HD	15.6	2.1
5	83GS000J\	Laptop Len	LENOVO	Gaming	Windows 1	21990000	4.9	i5 12450H)	RTX 3050 6	144	16	512GB	SSD	Full HD	15.6	2.4
6	RP745W	Laptop Asu	ASUS	Gaming	Windows 1	17190000	4.9	i5 12500H	RTX 2050 4	144	16	512GB	SSD	WUXGA	16	1.8
7	94F19PA	Laptop HP	HP	Gaming	Windows 1	16990000	5	Ryzen 5 75	RTX 2050 4	144	16	512GB	SSD	Full HD	15.6	2.3
8	460VN	Laptop MSI	MSI	Gaming	Windows 1	19690000	4.9	i5 12450H	RTX 4050 6	144	16	512GB	SSD	Full HD	15.6	1.9
9	NH.QPFS\	Laptop Ace	ACER	Gaming	Windows 1	20490000	5	Ryzen 5 66	RTX 3050 6	165	16	512GB	SSD	Full HD	15.6	2.1
10	83GS00D9	Laptop Len	LENOVO	Gaming	Windows 1	23190000	5	i5 12450H)	RTX 3050 6	144	24	512GB	SSD	Full HD	15.6	2.4
11	8Y6W3PA	Laptop HP	HP	Gaming	Windows 1	17690000	4.9	i5 12450H	RTX 2050 4	144	16	512GB	SSD	Full HD	15.6	2.3
12	NX.KQ4SV	Laptop Ace	ACER	Gaming	Windows 1	17690000	4.9	i5 13420H	RTX 2050 4	60	16	512GB	SSD	Full HD	15.6	1.7
13	1411VN	Laptop MSI	MSI	Gaming	Windows 1	21390000	4.9	i7 13620H	RTX 3050 4	144	32	512GB	SSD	Full HD	15.6	1.9
14	2045VN	Laptop MSI	MSI	Gaming	Windows 1	18190000	5	i5 12450H	RTX 3050 4	144	16	512GB	SSD	Full HD	15.6	1.9
15	HN113W	Laptop Asu	ASUS	Gaming	Windows 1	17690000	4.9	Ryzen 7 74	RTX 2050 4	144	16	512GB	SSD	Full HD	15.6	2.3
16	RP629W	Laptop Asu	ASUS	Gaming	Windows 1	18190000	4.9	i5 12500H	RTX 3050 4	144	16	512GB	SSD	WUXGA	16	1.8
17	2077VN	Laptop MSI	MSI	Gaming	Windows 1	24990000	4.9	i7 13620H	RTX 3050 6	144	16	1000GB	SSD	Full HD	15.6	2.3
18	NH.QFHS\	Laptop Ace	ACER	Gaming	Windows 1	18690000	4.9	i5 12500H	RTX 3050 4	144	8	512GB	SSD	Full HD	15.6	2.5
19	1423VN	Laptop MSI	MSI	Gaming	Windows 1	24690000	5	i7 13620H	RTX 4050 6	144	16	512GB	SSD	Full HD	15.6	2
20	887VN	Laptop MSI	MSI	Gaming	Windows 1	18690000	4.9	i7 12650H	RTX 3050 4	144	8	512GB	SSD	Full HD	15.6	1.9
21	9RC55MF5	Laptop GIC	Unknown	Gaming	Windows 1	21190000	5	i5 13500H	RTX 4050 6	144	16	512GB	SSD	Full HD	15.6	2.1
22	HN330W	Laptop Asu	ASUS	Gaming	Windows 1	20490000	4.9	i5 12500H	RTX 3050 4	144	16	512GB	SSD	Full HD	15.6	2.2
23	LP057W	Laptop Asu	ASUS	Gaming	Windows 1	24690000	5	Ryzen 7 74	RTX 4050 6	144	16	512GB	SSD	Full HD	15.6	2.2
24	045VN	Laptop MSI	MSI	Gaming	Windows 1	35690000	4.9	i7 14700H)	RTX 4060 8	240	16	1000GB	SSD	QHD+	16	2.3
25	83DV003C	Laptop Len	LENOVO	Gaming	Windows 1	22690000	5	i5 13450H)	RTX 3050 6	144	16	512GB	SSD	Full HD	15.6	2.4
26	83JC0040\	Laptop Len	LENOVO	Gaming	Windows 1	26190000	5	Ryzen 7 74	RTX 4050 6	144	24	512GB	SSD	Full HD	15.6	2.4
27	LP186W	Laptop Asu	ASUS	Gaming	Windows 1	27190000	5	i7 13620H	RTX 4050 6	144	16	512GB	SSD	Full HD	15.6	2.2
28	085VN	Laptop MSI	MSI	Gaming	Windows 1	50690000	5	Ultra 7 155	RTX 4060 8	120	32	1000GB	SSD	2.8K	14	1.7

III. Main components of the system

3. Data Storage & Search

Interface: `ProductDAO<T extends Product>`

- Defines essential functionalities that any database should support for handling product data.
- Utilizes **generics** for flexibility when switching product types.
- Implemented by specific DAO classes for different database, e.g., `PostgreSqlDAO` in this project.

III. Main components of the system

3. Data Storage & Search

Interface: `ProductFactory<T>`

Provides two main functionalities:

- Convert a database query result into a product of type T.
- Return a list of `T` products by executing a query with a given database URL, username, and password. Implemented by the abstract class `SqlFactory`.

III. Main components of the system

3. Data Storage & Search

Abstract Class: `SqlFactory`

- Extended by product-specific factory classes for different databases
  e.g., `LaptopPostgreSqlFactory`.
- Eliminate code duplication.
  Without it, adding a new product (e.g., Phone) would require writing a separate factory class (e.g., PhonePostgreSqlFactory) and duplicating common logic.
- The `createProduct()` method must still be implemented individually in each factory, as each product type has distinct attributes.

III. Main components of the system

3. Data Storage & Search

Class: `LaptopPostgreSqlFactory` **`implements`** `ProductFactory<Laptop>`

• Generates Laptop objects from database queries.

• To support additional product types or databases, similar factory classes should be created by extending `SqlFactory` — e.g., `PhoneMySqlFactory`.

III. Main components of the system

3. Data Storage & Search

Class: `PostgreSqlDAO<T extends Product>`

- Implements the `ProductDAO<T>` interface to handle database operations.
- Uses Java generics to easily switch between different product types in the PostgreSQL database.
- Contains a `ProductFactory<T>` attribute to separate responsibilities:
  PostgreSqlDAO handles connection and query logic.
  The factory handles converting query results into product objects.
- Enhances reusability:

```
ProductDAO<Laptop> admin
           = new PostgreSqlDAO<>("laptop", new LaptopPostgreSqlFactory());
ProductDAO<Phone> admin
           = new PostgreSqlDAO<>("phone", new PhonePostgreSqlFactory());
```

III. Main components of the system

4. Query & Response Generation

4.1. Package retrieval

Function: Handle information retrieval tasks - product search based on vector embeddings and similarity computation.

III. Main components of the system

4. Query & Response Generation

4.1. Package retrieval

Class: `ProductWithScore`

- Attributes:

```
private final Product product;
private double score;
```

- Encapsulate a Product object along with its associated relevance score.
- The **`Product`** attribute is marked as `final`, ensuring that the associated **`Product`** object cannot be changed after initialization.
- **`ProductWithScore`** can hold any object of type **`Product`** or its subclasses, allowing the class to be reused in the future when the system expands to include other product types.

III. Main components of the system

4. Query & Response Generation

4.1. Package retrieval

Class: **`ProductSearchService`**

Method:

```
    public <T extends Product> List<T>
        searchVector(double[] queryVector, List<T> products, int k)
```

returns the k products with the highest similarity scores to the query vector.

This method uses a generic type parameter `<T extends Product>`, meaning T must be a subclass of `Product` ⇒ Flexibility & Code reuse

Method:

```
    private double
        cosineSimilarity(double[] vectorA, double[] vectorB)
```

calculate the cosine similarity between two vectors using the **cosineSimilarity**.

III. Main components of the system

4. Query & Response Generation

4.1. Package retrieval

Class: **`EmbeddingService (1)`**

Function: Converts user queries and product descriptions (via toString()) into numerical vector embeddings using an external API.

Key Components:

- `private final Configuration config`: Stores the API URL (via `config.getApiUrl()),` ensures immutability and safety.
- `public double[] embedQuery(String query):` Returns the embedding vector of a text query.
- `public double[][] embedProducts(List<? extends Product> products)`: Returns embedding vectors for a list of products.

III. Main components of the system

4. Query & Response Generation

4.1. Package `retrieval`

Class: **`EmbeddingService (2)`**

This class is highly generalized, offering flexibility and abstraction because:

- Supports Multiple Product Types

  The embedProducts method uses `List<? extends Product>,` allowing it to handle any subclass of `Product` (e.g., Laptop, Smartphone).

- Decoupled Configuration

  Uses the Configuration class to fetch API URLs (via `getApiUrl()),` making it easy to change endpoints or settings without modifying the code.

- Batch Processing Support

  The getEmbeddings method accepts a String[] array, enabling batch requests for multiple queries/products in a single API call to optimize performance.

III. Main components of the system

4. Query & Response Generation

4.2. Package `llm`

Function: Integrate Large Language Models (LLMs) to generate intelligent, context-aware responses for product consultation tasks.

Interface: `AIClient`

Defines a contract for AI integration, requiring implementing classes to provide a `getResponse(String userQuery, List<? extends Product> products)` method that returns a string-based response based on a user query and a list of products.

Class: `MistralClient implements AIClient`

+ Implements the AIClient interface and interacts with the Mistral AI API.
+ Constructor accepts a Configuration object to retrieve API URL and API key, ensuring flexibility and avoiding hard-coded values.
+ Encapsulates API logic, making it easy to switch to or support other AI providers in the future.

IV. Demo

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !