**NỘI DUNG THUYẾT TRÌNH OOP**

**🎤 Slide 1, 2:**

**Speaker:**  
Hello everyone, we are Group 7.  
Today, we will present our project:  
**“Laptop Consultation System Integrating Information Retrieval and Large Language Models.”**

**📋 Slide 3 – Outline:**  
We will go through four parts showed in the screen.

**🧭 Slide 4 – Introduction:**

In our project, we developed a system for collecting laptop data, stored and managed data, provided products using embedding techniques and AI.

Our system will help users find suitable laptops, collect and analyze data automatically, and apply modern AI technologies.

**🎯 Slide 5 – System overview:**

**Our system includes:**

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

**🏗️ Slide 6 – System Architecture:**

The system follows a modular architecture with clear package dependencies. The central package is **group7.model**, which is highly dependent upon other components, making it the core of the system's logic. **group7.ui.controller** has multiple outward dependencies, acts as a coordinator between the user interface and backend processes. **group7.retrieval** and **group7.llm** both depend on **group7.config**, showing that configuration settings are shared between them. **group7.data.collector** and **group7.data.storage** are low in dependency, suggesting their functionality is more isolated or service-oriented.

**📦 Slide 7 – Package: group7.model:**

The group7.model package defines our main data models — like the Product and Laptop classes.

**⚙️ Slide 8 – Package: group7.config:**

group7.config stores configuration settings like API keys and database URLs.  
It ensures reusability, flexibility, safety and encapsulation.

**🕷️ Slide 9 – Package: group7.data.collector:**

group7.data.collector with the main method is the function collectStructuredData which stores and retrieves laptop data from the database.

The others functions support the main one.

**💽 Slide 10 – Package: group7.data.storage:**

The group7.data.storage handles all database interactions using Java generics and PostgreSQL.

There are 3 classes:

**🤖 Slide 11 – Package: group7.llm:**

group7.llm handles the connection to Large Language Models like **Mistral**.  
It takes user queries and generates smart responses based on AI.

**🔍 Slide 12 – Package: group7.retrieval:**

The group7.retrieval package performs vector-based search. It finds the most relevant laptops based on the user’s intent and similarity score. The class “llm” (Large Language Models) take this vector as its input.

**🧑‍💻 Slide 13 – Package: group7.ui.controller :**

The group7.ui.controller handles user interactions, connects GUI with the logic.  
Its 3 classes will manage switching between different scenes or views; manage the search bar, query submission, and result display**;** manage the product detail screen that appears after a user clicks.

**👤 Slide 14 – Use Case Diagram:**

The user can start the program, find laptops by using shown filters, sort choices in specific order, ask ChatBot for more advanced search, and exit the program.

**🧱 Slide 15 – Product & Laptop Classes:**

We have an abstract class Product with general attributes: id, name, brand, price, rating and url. We use method mapToDataBase() and getVector() to support storage and search functionalities. This class helps maintain abstraction, reusability and easy integration.

**💻 Slide 16 – Laptop Class Details:**

The Laptop class **inherits** from the Product class.

It implements the mapToDatabase() method to map laptop information into the database and the getSpecification() method to return technical specifications in the form of a map.

**🕸️ Slide 17 – DataCollector Process:**

The DataCollector uses Selenium to scrape the data. This process starts with (steps):

* Access brand URLs
* Load all products (loadAllProducts).
* Extract data from HTML (scrapeLaptopFromMainPage).
* Validate and check for duplicates (isValidLaptop, isDuplicateLaptop).
* Save to CSV and return objects.

It is highly automated and handles errors gracefully.

**📈 Slide 18 – Example image of Data collectors:**

**📈 Slide 19 – ProductDAO Interface:**

ProductDAO<T> will:

* Defines essential functionalities.
* Utilizes **generics** for flexibility.
* Implemented by specific DAO classes for different databases.

**🏭 Slide 20 – ProductFactory Interface:**

The ProductFactory<T> interface helps:

* Convert a database query into a product of type T.
* Return to a list of T products. Implemented by the abstract class SqlFactory.

**🏗️ Slide 21 – SqlFactory Class:**

The SqlFactory is an abstract class that:

* Extended by product-specific factory classes for different databases
* Eliminate code duplication.
* The createProduct() method must still be implemented individually in each factory, as each product type has distinct attributes.

**🔄 Slide 22 – LaptopPostgreSqlFactory:**

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

* Generates Laptop objects from database queries.
* To support additional product types or databases, similar factory classes should be created.

**🗃️ Slide 23 – PostgreSqlDAO:**

Class: PostgreSqlDAO<T extends Product> will:

* Implement the ProductDAO<T> interface to handle database operations.
* Use Java generics to switch between different product types in the PostgreSQL database.
* Contain a ProductFactory<T> attribute to separate responsibilities:

PostgreSqlDAO handles connection and query logic.

The factory handles converting query results into product objects.

* Enhance reusability.

**🔍 Slide 24 – Retrieval Package Overview:**

Now we will move to **search and response generation**.

Package retrieval handles information retrieval tasks - product search based on vector embeddings and similarity computation.

**🧮 Slide 25 – ProductWithScore:**

The ProductWithScore class encapsulates a Product object along with its associated relevance score.

**ProductWithScore** can hold any object of type **Product** or its subclasses.

**🧠 Slide 26 – ProductSearchService:**

The ProductSearchService has 2 methods:

* **searchVector:** retrieves the top k products most like a query vector from a given list. This method uses a generic type parameter <T extends Product>, meaning T must be a subclass of Product. It provides flexibility and code reusability.
* **cosineSimilarity**: calculates the cosine similarity between two vectors.

**🧬 Slide 27 – EmbeddingService (Part 1)**

**EmbeddingService** converts user queries and product descriptions (via toString()) into numerical vector embeddings using an external API.

It contains:

* **Configuration**: stores the API URL (via config.getApiUrl()), ensures immutability and safety.
* **embedQuery():** Returns the embedding vector of a text query.
* **embedProducts():** Returns embedding vectors for a list of products.

**🧬 Slide 28 – EmbeddingService (Part 2)**

**EmbeddingService** class is highly generalized, offering flexibility and abstraction because:

* Supports Multiple Product Types
* Decoupled Configuration
* Batch Processing Support

**🧠 Slide 29 – Package llm**

**Package llm** has 2 classes:

**Interface AIClient** defines a contract for AI integration, requiring implementing classes to provide a getResponse() method that returns a string-based response based on a user query and a list of products.

**Class MistralClient:**

* Implements the AIClient interface and interacts with the Mistral AI API.
* Constructor accepts a Configuration object to retrieve API URL and API key.
* Encapsulates API logic.

**🧠 Slide 30, 31 – Demo & “thank you”**

That’s the end of our presentation.  
Thank you for listening!  
We’re happy to take your questions now.