Medtech Documentation

Nicolas Pinto - Student ID: 807348
https://github.com/npinto97/medtech

A.Y. 2023-2024

1 Logical Design

The logical design of the database system for *MedTech Logistics* is based on the interpretation of the functional requirements and frequency of operations described in the exam prompt. I chose to implement an *object-relational model* using Oracle's support for *object types*, *inheritance*, *nested tables*, and *REFs*. This approach enables a modular, extensible structure that mirrors the real-world complexity of medical logistics.

1.1 Real-World Modeling with Objects

Entities such as products, batches, customers, and logistics teams are modeled as *user-defined object types*, each encapsulating not only attributes but also behaviors through *member functions*. For example, a `Product` can determine if it's expired, a `Batch` can check if its quantity is valid, and a `Customer` can verify the correctness of all its orders and complaints. This encapsulation improves cohesion and ensures that business rules are consistently enforced throughout the application logic.

1.2 Strategic Use of REF vs Nested Table

One of the key design decisions involved choosing *when to use REF* and *when to use nested tables* to model relationships.

I used REFs to represent *associations between entities that are independently stored in dedicated tables*, especially when those entities are shared across multiple relationships or have a long lifecycle. For example:

- Batch refers to Product and DistributionCenter via REFs.
- Order refers to Batch objects through a nested table of REF Batch_TY.
- Order is also linked to the responsible LogisticsTeam using a REF.

- Batch refers to Product and DistributionCenter via REFs.
- Order refers to Batch objects through a nested table of REF Batch_TY.
- Order is also linked to the responsible LogisticsTeam using a REF.

This allows for better decoupling and enables updates to referenced entities without needing to modify the referencing objects. It also supports complex querying, such as identifying all orders managed by a specific logistics team or tracking expired products across batches.

In contrast, nested tables were used to represent exclusive, contained relationships, where the lifecycle of the nested elements is strongly tied to the parent object. For example:

- A Customer contains a nested table of Department objects, since departments are conceptually and functionally tied to their customer.
- Customer also includes nested tables of Order and Complaint objects, which only make sense in the context of a specific customer and are not shared across other entities.

- Each LogisticsTeam has a nested table of TeamMember objects, again tightly coupled to the parent team.

This choice simplifies data management, makes insertion and deletion more efficient, and reflects the natural ownership of these elements.

1.3 Inheritance and Role Differentiation

To handle employee specialization, I introduced *type inheritance* with an abstract base type `LogisticsEmployee_TY`, from which I derived the `ChiefLogisticsOfficer` and `TeamMember` types. This allows us to represent shared attributes (such as name, tax code, and employment date) only once, while also enabling specific roles to have additional fields—e.g., `yearsOfExperience` for chiefs.

This use of inheritance improves code reuse and ensures consistency across similar object types, while still allowing for flexible specialization where needed.

1.4 Alignment with Operational Needs

The object-relational model is directly aligned with the five main operations described in the prompt:

1. *Registering batches*: done frequently, so it's optimized with a simple procedure using REFs to existing products and centers.

2. *Placing orders*: supports multibatch selections through a string-based interface, internally converted into REF collections.

3. *Assigning deliveries*: updates the logistics team reference directly via REF, allowing efficient reassignment without restructuring the object.

4. *Viewing deliveries by chief*: enabled through queries that traverse REF relationships and nested orders.

5. *Listing expired batches*: made efficient through encapsulated logic within the Product and Batch types to detect expiration.

2 Indexes

To ensure optimal performance of the Health Supply Management System, several indexes have been defined to support frequent and critical query patterns. This section describes the purpose and expected utility of each index.

- `idx_batch_arrival` on `Batch_TABLE(arrivalDate)`

  This index supports operations that involve filtering or ordering batches based on their arrival date. For instance, when querying recently received batches (e.g., for restocking or inspection), this index allows Oracle to perform a range scan rather than a full table scan, significantly improving performance. Example usage:

  ```
      SELECT * FROM Batch\_TABLE
      WHERE arrivalDate > SYSDATE - 30;

  ```

- `idx_batch_quantity` on `Batch_TABLE(quantity)`

  This index facilitates quick retrieval of batches with critically low quantity, which is relevant for monitoring stock levels. It can be particularly helpful in conjunction with business logic (e.g., within functions or triggers) that checks whether a batch needs restocking. Example usage:

```
SELECT * FROM Batch\_TABLE
WHERE quantity < 10;
```

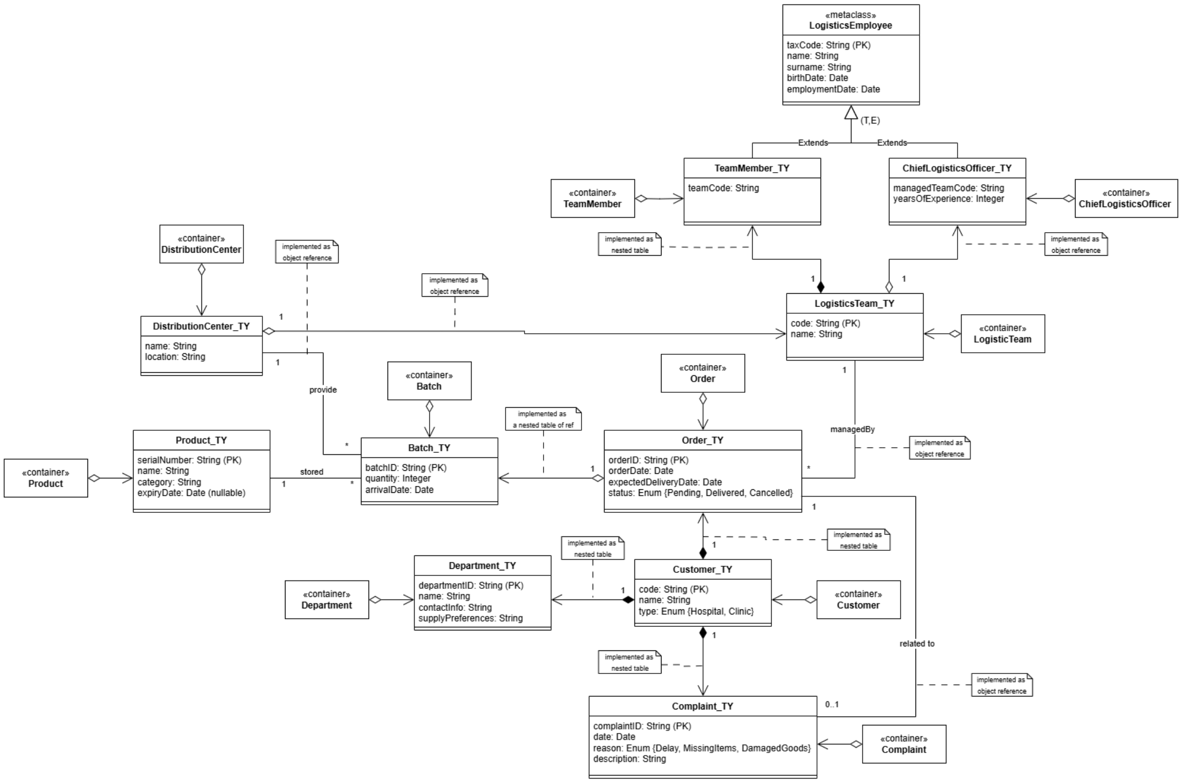


Figure 1: UML schema for Medtech's object-oriented database.

- `idx_product_expiry` on `Product(expiryDate)`

  This index is essential for operations that detect expired medical products. It directly supports queries or procedures that call the is_expired member function of the Product_TY object type, or that explicitly filter by the expiryDate field.

  Example usage:

```sql
SELECT serialNumber, productName
FROM Product
WHERE expiryDate < SYSDATE;
```

- `idx_customer_type` on `Customer(customerType)`

  This index supports filtered searches based on customer category (e.g., "Hospital", "Clinic", etc.). It is especially useful when generating reports or targeting specific customer types in distribution or analytics operations.

  Example usage:

```sql
SELECT customerID, customerName
FROM Customer
WHERE customerType = 'Hospital';
```

These indexes are designed to enhance query efficiency by reducing I/O cost and improving response times. Their effectiveness depends on query selectivity and data distribution, and should be monitored periodically using Oracle's execution plans (EXPLAIN PLAN) and performance tools (DBMS_STATS, AWR reports) to ensure they continue to provide value.

3 Procedures and Functions

This system defines a set of stored procedures and functions that implement the five main operations required by the logistics and distribution use case. These PL/SQL modules encapsulate the logic for batch management, order placement, delivery coordination, and reporting, ensuring modularity, reusability, and data integrity.

- Operation 1: Register a New Product Batch - Procedure: `register_new_batch`

  This procedure registers a new batch by accepting its identifier, quantity, arrival date, and references to the related product and distribution center. It internally resolves the corresponding object references and inserts a new Batch_TY object into the `Batch_TABLE`. It's expected to be called frequently, as batch arrivals occur approximately 100 times per day.

- Operation 2: Place a New Order - Procedure: `place_order`

  Designed to handle about 70 new orders daily, this procedure receives a customer's code and the order's metadata, including a comma-separated string of batch IDs. It uses a local function (split_string) to parse the batch IDs and convert them into a list of object references. It then builds an `Order_TY` object and appends it to the customer's nested collection of orders.

- Operation 3: Assign a Delivery to a Logistics Team - Procedure: `assign_delivery`

  This procedure associates a specific customer order with a logistics team. Given the order ID, customer code, and logistics team code, it retrieves the team reference and assigns it to the appropriate order via an update on the nested table of orders.

- Operation 4: View Deliveries Coordinated by a Specific Chief Officer - Function: `get_team_deliveries`

  This query function returns a SYS_REFCURSOR pointing to all deliveries assigned to logistics teams coordinated by a specific chief officer (identified by tax code). It joins customer orders and logistics team information to retrieve relevant delivery details.

- Operation 5: List All Batches of Expired Products - Function: `get_expired_batches`

  This reporting function returns a cursor containing all batches that reference expired products. It performs a join between `Batch_TABLE` and Product using object references and filters out entries where the product's expiryDate has already passed. The function is useful for quality assurance and regulatory compliance and is executed roughly 20 times per day.

4 Triggers

Triggers play a crucial role in preserving data integrity and enforcing business rules automatically during data manipulation operations. Below is an overview of the main triggers defined in the system, organized by entity and function.

- `trg_validate_batch` – Validates Batch Quantity

  This trigger ensures that a new or updated batch has a valid quantity before being inserted into the database. By instantiating a `Batch_TY` object, it calls the `is_valid` member function. If the quantity is zero or negative, the operation is rejected with a custom error.

- `trg_prevent_expired_batch` – Prevents Association with Expired Products

  Before inserting or updating a batch, this trigger checks whether the associated product has already expired. It uses the `is_product_expired` member function of the `Batch_TY` object. If the product is expired, insertion is blocked, ensuring that only viable products are included in new batches.

- `trg_check_customer_orders` – Validates Customer Orders

  Triggered before changes to a Customer record, this trigger validates that all the customer's orders are considered valid according to the `all_orders_valid` function in `Customer_TY`. It helps avoid inconsistencies due to incomplete or invalid orders associated with a customer.

- `trg_check_customer_complaints` – Validates Customer Complaints

  This trigger checks whether all complaints linked to a customer are valid through the `all_complaints_valid` function. It ensures data correctness and prevents malformed or incomplete complaints from being inserted or persisted.

- `trg_block_expiry_update` – Restricts Expiry Date Modification

  This critical trigger blocks updates to the `expiryDate` of a product if there are already existing, delivered batches referencing it, and the new expiry date is in the past. It ensures historical accuracy and avoids introducing inconsistencies that could affect batch traceability.

- `trg_validate_chief` – Ensures Chief Reference Validity

  Before inserting or updating a `LogisticsTeam`, this trigger verifies that the assigned chief is a valid, existing `ChiefLogisticsOfficer`. If the reference is broken or invalid, the operation is aborted, ensuring referential integrity between teams and their leaders.

- `trg_protect_chief_deletion` – Prevents Deletion of Assigned Chiefs

  To preserve the organizational structure, this trigger prevents deletion of a `ChiefLogisticsOfficer` if they are currently referenced as a chief by any `LogisticsTeam`. It enforces a rule of safe removal, requiring all dependent teams to be reassigned before deletion.

5 Views

To enhance usability, performance, and readability of queries on complex nested structures, we designed several *SQL views*. These views serve as abstraction layers that simplify access to frequently used or deeply nested data, helping developers and analysts retrieve information without having to navigate the full object model manually.

- `Expired_Batches_VW`

  This view provides an immediate and readable list of all expired product batches. While the `Batch_TY` type includes a method to check expiration via the associated product, this view makes it possible to query expired items directly using SQL, without invoking PL/SQL methods or dereferencing manually.

```sql
SELECT b.batchID, b.quantity, b.arrivalDate, p.productName, p.expiryDate
FROM Batch_TABLE b
JOIN Product p ON b.productRef = REF(p)
WHERE p.expiryDate IS NOT NULL AND p.expiryDate < SYSDATE;
```

  This is particularly useful for Operation 5 (list all batches of expired products), allowing the operation to be executed efficiently many times per day.

- `Customer_Orders_VW`

  Accessing nested tables in Oracle requires the use of the TABLE() keyword, which can be verbose and repetitive when working with multiple customers and their respective orders. This view flattens the relationship between customers and their orders, exposing each order as a standalone row with its associated customer code.

```sql
SELECT c.code AS customerCode, o.orderID, o.status, o.orderDate, o.expectedDeliveryDate
FROM Customer c, TABLE(c.orders) o;
```

  This view is especially useful for reporting, filtering orders by customer, or analyzing delivery status over time.

- `Team_Deliveries_VW`

  The assignment of deliveries to logistics teams is an important operation in the system. This view exposes all customer orders along with the responsible logistics team and its chief officer. By pre-joining this data and resolving the REF relationships, the view enables easy queries to support Operation 4 (view all deliveries assigned to the team coordinated by a specific chief).

```sql
SELECT c.code AS customerCode, o.orderID, o.status, o.expectedDeliveryDate,
       lt.code AS teamCode, lt.chief.taxCode AS chiefTaxCode
FROM Customer c, TABLE(c.orders) o, LogisticsTeam lt
WHERE o.managedBy = REF(lt);
```

This view also supports auditing, performance monitoring, and team accountability checks.

- `All_Complaints_VW`

  To streamline the handling of customer complaints, this view extracts all complaint data from the nested collections and presents it in a flat format. This avoids the need for complex joins and allows for fast inspection or filtering based on complaint reason or date.

```
SELECT c.code AS customerCode, comp.complaintID, comp.compliantDate,
       comp.reason, comp.complaintDescription
FROM Customer c, TABLE(c.complaints) comp;
```

This view is ideal for dashboards, service analytics, and alerting systems that need to track delivery issues in real time.

6 Data Population

To ensure meaningful testing of the system and simulate real-world usage scenarios, the database has been populated with both *manually inserted sample data* and *automatically generated records* using PL/SQL blocks. The goal was twofold: to allow quick initial demonstrations and to stress-test the system with a high volume of interrelated objects.

6.1 Manual Data Insertion

Initially, I inserted a small but representative set of entities manually. These include products, distribution centers, logistics officers, teams, batches, and customers with associated orders and complaints. These records are designed to test all key functionalities of the system, such as expiration checks, complaint handling, and delivery assignment.

In the manual phase, I deliberately mixed products with and without an expiration date, batches with varying arrival dates, and complaints covering different types of issues (e.g., delivery delays, product problems). I also showed how to insert teams with nested members, and how to properly use ‘REF‘ values to link objects like batches, products, teams, and customers.

This step was also useful to verify the *correct functioning of constructors*, nested tables, and type references.

6.2 Use of REF vs Nested Tables in Insertion

During data insertion, I chose to use ‘REF‘ types when modeling *relationships between major entities* (such as batches pointing to a product, or orders pointing to a team), because this better reflects the nature of their association and allows for independent manipulation of the objects involved. For example, products can be inserted and queried independently from batches, while the ‘REF‘ maintains the logical link.

On the other hand, nested tables were used to model contained sub-entities, like the departments inside a customer or the team members inside a logistics team. These elements don’t need to exist on their own in the system and are logically ”owned” by their parent.

6.3 Automatic Population via PL/SQL

To simulate a realistic workload and enable stress testing, I added a PL/SQL procedure that populates the database with dozens of additional entities. This includes:

- 20 products with varying categories and expiry dates.
- 10 distribution centers in different cities.
- 5 chief logistics officers.
- 5 logistics teams, each with 4 unique members and associated to a specific center and chief.
- 50 batches distributed across the centers and linked to the generated products.

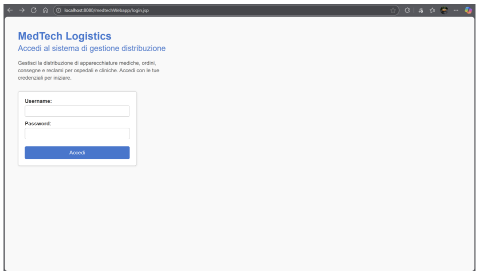
- 10 customers, each with:
  - 2 internal departments,
  - 2 orders (each pointing to different batches and teams),
  - 1 complaint of varying reason.

The generated data preserves internal consistency by correctly resolving 'REF''s using 'SELECT REF(...)' statements. For example, before inserting an order, the script dynamically retrieves the reference to the proper batch and logistics team.

7 MedtechWebapp

MedtechWebapp is a web application developed using Java EE and deployed on an Apache Tomcat server. It is designed to support and streamline internal processes in the healthcare domain, with a focus on tracking customer orders, complaints, and product batch logistics. The application interacts with an Oracle database, retrieving information using SQL queries, views, PL/SQL functions, and structured objects.

The application logic is organized across several servlets, each dedicated to a specific task. For instance, to manage authentication, the application includes a `LoginServlet`, which verifies the username and password against the USERS table. If authentication is successful, it stores the username and role in the session and redirects the user to the main page. If not, it forwards them back to the login page with an error message. There's also a `LogoutServlet`, which simply invalidates the session and redirects the user to the login form, ensuring session cleanup and security.



localhost:8080/medtechWebapp/login.jsp

MedTech Logistics

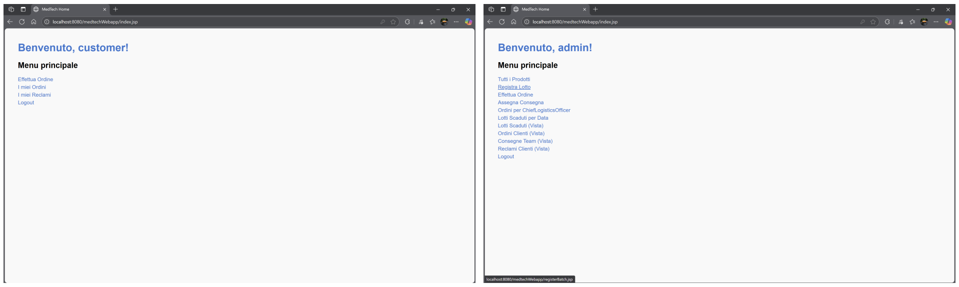
Accedi al sistema di gestione distribuzione

Gestisci la distribuzione di apparecchiature mediche, ordini, consegne e reclami per ospedali e cliniche. Accedi con le tue credenziali per iniziare.

Username:

Password:

Figure 2: Medtech Webapp home page.



MedTech Home

localhost:8080/medtechWebapp/index.jsp

Benvenuto, customer!

Menu principale

MedTech Home

localhost:8080/medtechWebapp/index.jsp

Benvenuto, admin!

Menu principale

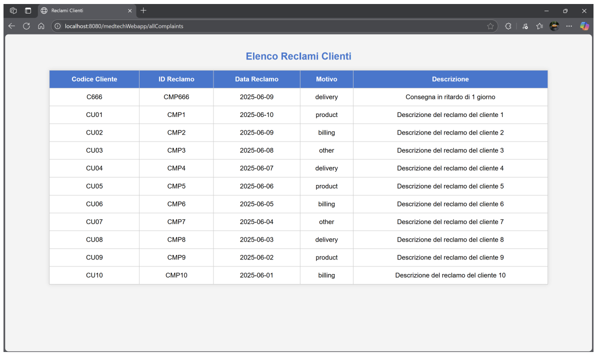
localhost:8080/medtechWebapp/registerBatch.jsp

Figure 3: Customer home and Admin home.

The `AllComplaintsServlet` handles the retrieval and display of all complaints by querying the `All_Complaints_VW` view and collecting the results into a list. Each complaint includes details such as the customer code, complaint ID, date, reason, and a description.

Another important component is `AssignDelivery`, which retrieves the list of deliveries assigned to a specific team. This servlet calls a PL/SQL function (`get_team_deliveries`) that returns a `SYS_REFCURSOR`, and displays the data in an HTML table generated in response to the user's request.

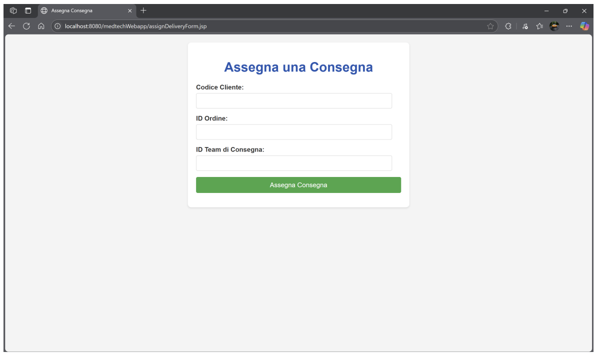
The `CustomerOrdersServlet` focuses on displaying all customer orders. It queries a dedicated view and builds an HTML table with key information such as order status, order date, and expected delivery date. Null dates are handled carefully to avoid display issues.



Elenco Reclami Clienti

Codice Cliente	ID Reclamo	Data Reclamo	Motivo	Descrizione
C666	CMP666	2025-06-09	delivery	Consegna in ritardo di 1 giorno
CU01	CMP1	2025-06-10	product	Descrizione del reclamo del cliente 1
CU02	CMP2	2025-06-09	billing	Descrizione del reclamo del cliente 2
CU03	CMP3	2025-06-08	other	Descrizione del reclamo del cliente 3
CU04	CMP4	2025-06-07	delivery	Descrizione del reclamo del cliente 4
CU05	CMP5	2025-06-06	product	Descrizione del reclamo del cliente 5
CU06	CMP6	2025-06-05	billing	Descrizione del reclamo del cliente 6
CU07	CMP7	2025-06-04	other	Descrizione del reclamo del cliente 7
CU08	CMP8	2025-06-03	delivery	Descrizione del reclamo del cliente 8
CU09	CMP9	2025-06-02	product	Descrizione del reclamo del cliente 9
CU10	CMP10	2025-06-01	billing	Descrizione del reclamo del cliente 10

Figure 4: All complaints.



Assegna una Consegna

Codice Cliente:

ID Ordine:

ID Team di Consegna:

Figure 5: Assign delivery.

For batch management, two servlets are responsible for identifying expired product lots. `ExpiredBatches` displays all batches whose associated product has an expiry date earlier than the current date (`SYSDATE`). On the other hand, `ExpiredBatchesByData` allows users to filter results based on a custom expiry date provided through the web interface. Both servlets construct an HTML table showing details like batch ID, quantity, arrival date, product name, and expiration date.

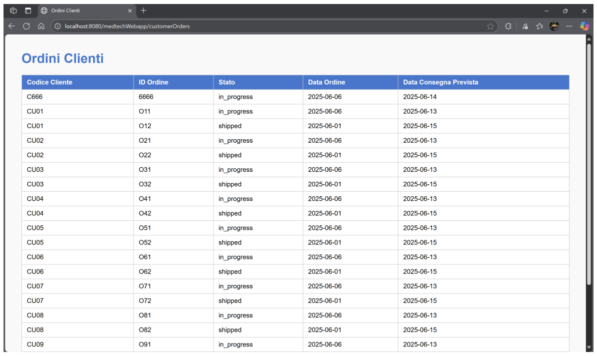
The `ListAllProducts` servlet retrieves all products stored in the Oracle database, which are represented as user-defined objects. It uses the SQL query `SELECT VALUE(p) FROM Product p` to return each `Product` object and then extracts its attributes using Oracle's STRUCT type. The output is displayed in a clean HTML table showing the serial number, product name, category, and expiration date. If no products are found, it simply shows a message stating that the database is empty.

The `ListChiefOrders` servlet focuses on logistics team management. It allows a Chief Logistics Officer to see all orders handled by their team. The servlet takes the chief's tax code as input and uses it in a query that navigates nested and referenced objects to find the relevant orders. If valid, the results are shown in a table including customer codes, order IDs, dates, and status.

For order insertion, the `PlaceOrder` servlet allows users to submit a new order via a form. It collects parameters such as customer code, order ID, dates, order status, and the IDs of the product batches involved. Then it calls a PL/SQL stored procedure (place_order) to persist the order in the database. Success or error messages are passed back to the view for user feedback.

Similarly, the `RegisterBatch` servlet registers new product batches in the system. It receives details like the batch ID, quantity, arrival date, product serial number, and destination logistics center. These values are sent to the database through a stored procedure called register_new_batch, and again, the user is informed whether the operation was successful or not.

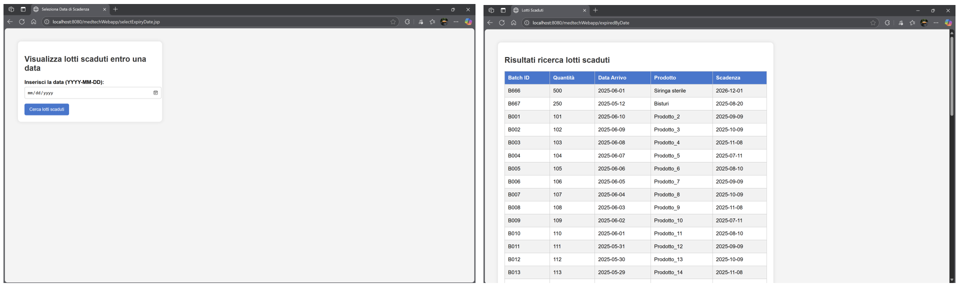
Lastly, the `TeamDeliveriesServlet` retrieves all delivery orders handled by logistics teams. It queries a database view (`Team_Deliveries_VW`) and displays each delivery in a table with information about the customer, order, status, expected delivery date, team code, and the chief's tax code. This view allows team members and managers to monitor delivery responsibilities in real-time.



Ordini Clienti

Codice Cliente	ID Ordine	Stato	Data Ordine	Data Consegna Prevista
C666	6666	in_progress	2025-06-06	2025-06-14
CU01	O11	in_progress	2025-06-06	2025-06-13
CU01	O12	shipped	2025-06-01	2025-06-15
CU02	O21	in_progress	2025-06-06	2025-06-13
CU02	O22	shipped	2025-06-01	2025-06-15
CU03	O31	in_progress	2025-06-06	2025-06-13
CU03	O32	shipped	2025-06-01	2025-06-15
CU04	O41	in_progress	2025-06-06	2025-06-13
CU04	O42	shipped	2025-06-01	2025-06-15
CU05	O51	in_progress	2025-06-06	2025-06-13
CU05	O52	shipped	2025-06-01	2025-06-15
CU06	O61	in_progress	2025-06-06	2025-06-13
CU06	O62	shipped	2025-06-01	2025-06-15
CU07	O71	in_progress	2025-06-06	2025-06-13
CU07	O72	shipped	2025-06-01	2025-06-15
CU08	O81	in_progress	2025-06-06	2025-06-13
CU08	O82	shipped	2025-06-01	2025-06-15
CU09	O91	in_progress	2025-06-06	2025-06-13

Figure 6: Customers orders.


Seleziona Data di Scadenza
localhost:8080/medtechWebapp/selectExpiryDate.jsp
Cerca lotti scaduti
Lotti Scaduti
localhost:8080/medtechWebapp/expiredByDate

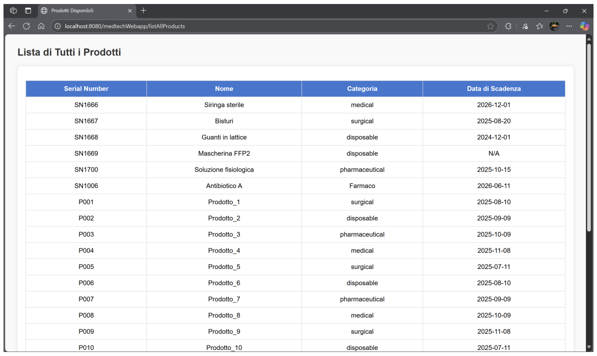
Visualizza lotti scaduti entro una data

Inserisci la data (YYYY-MM-DD): mm/dd/yyyy

Risultati ricerca lotti scaduti

Batch ID	Quantità	Data Arrivo	Prodotto	Scadenza
B666	500	2025-06-01	Siringa sterile	2026-12-01
B667	250	2025-05-12	Bisturi	2025-08-20
B001	101	2025-06-10	Prodotto_2	2025-09-09
B002	102	2025-06-09	Prodotto_3	2025-10-09
B003	103	2025-06-08	Prodotto_4	2025-11-08
B004	104	2025-06-07	Prodotto_5	2025-07-11
B005	105	2025-06-06	Prodotto_6	2025-08-10
B006	106	2025-06-05	Prodotto_7	2025-09-09
B007	107	2025-06-04	Prodotto_8	2025-10-09
B008	108	2025-06-03	Prodotto_9	2025-11-08
B009	109	2025-06-02	Prodotto_10	2025-07-11
B010	110	2025-06-01	Prodotto_11	2025-08-10
B011	111	2025-05-31	Prodotto_12	2025-09-09
B012	112	2025-05-30	Prodotto_13	2025-10-09
B013	113	2025-05-29	Prodotto_14	2025-11-08

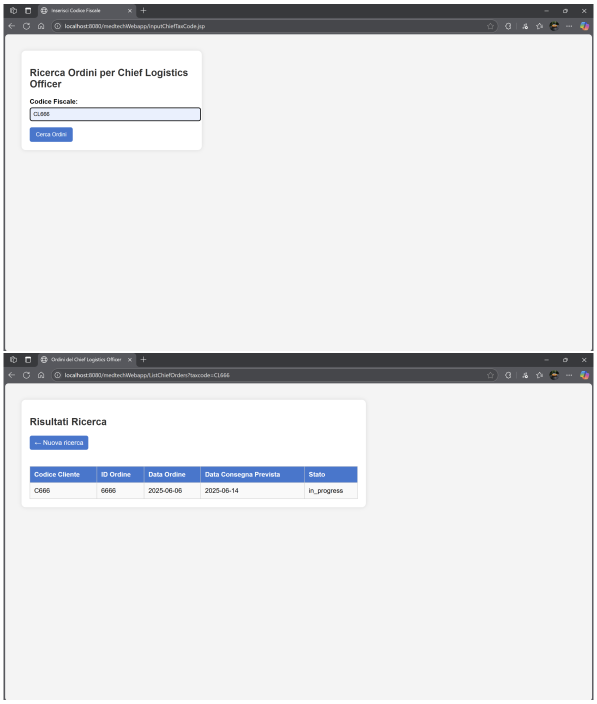
Figure 7: Expired batches by a given data.



Lista di Tutti i Prodotti

Serial Number	Nome	Categoria	Data di Scadenza
SN1666	Siringa sterile	medical	2026-12-01
SN1667	Bisturi	surgical	2025-08-20
SN1668	Guanti in lattice	disposable	2024-12-01
SN1669	Mascherina FFP2	disposable	N/A
SN1700	Soluzione fisiologica	pharmaceutical	2025-10-15
SN1006	Antibiotico A	Farmaco	2026-06-11
P001	Prodotto_1	surgical	2025-08-10
P002	Prodotto_2	disposable	2025-09-09
P003	Prodotto_3	pharmaceutical	2025-10-09
P004	Prodotto_4	medical	2025-11-08
P005	Prodotto_5	surgical	2025-07-11
P006	Prodotto_6	disposable	2025-08-10
P007	Prodotto_7	pharmaceutical	2025-09-09
P008	Prodotto_8	medical	2025-10-09
P009	Prodotto_9	surgical	2025-11-08
P010	Prodotto_10	disposable	2025-07-11

Figure 8: List all products


Inserisci Codice Fiscale
localhost:8080/medtechWebapp/inputChiefTaxCode.jsp

Ricerca Ordini per Chief Logistics Officer

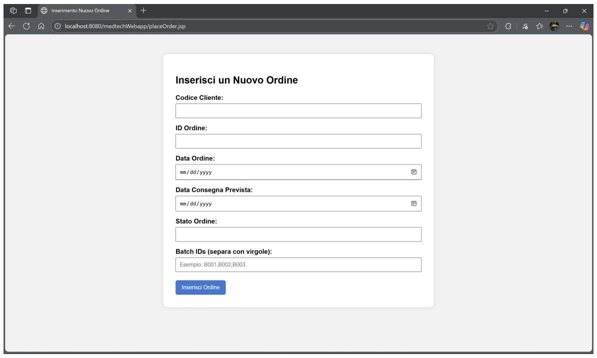
Codice Fiscale: CL666

Ordini del Chief Logistics Officer
localhost:8080/medtechWebapp/ListChiefOrders?taxcode=CL666

Risultati Ricerca

Codice Cliente	ID Ordine	Data Ordine	Data Consegna Prevista	Stato
C666	6666	2025-06-06	2025-06-14	in_progress

Figure 9: Enter Caption



Inserisci un Nuovo Ordine

Codice Cliente:

ID Ordine:

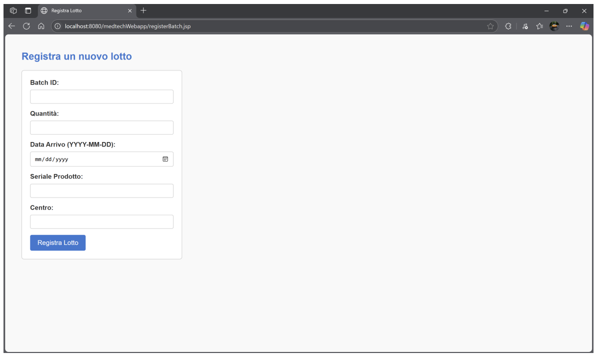
Data Ordine: mm/dd/yyyy

Data Consegna Prevista: mm/dd/yyyy

Stato Ordine:

Batch IDs (separa con virgole): Esempio: B001,B002,B003

Figure 10: Place an order.



Registra un nuovo lotto

Batch ID:

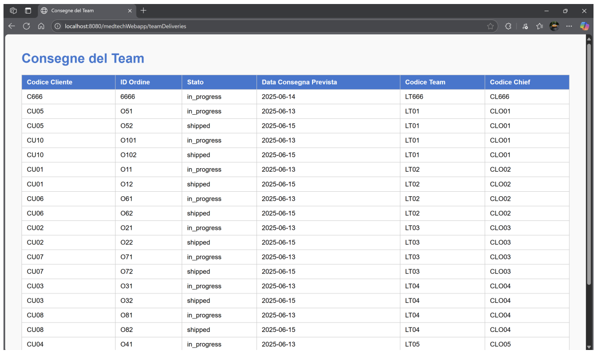
Quantità:

Data Arrivo (YYYY-MM-DD): mm/dd/yyyy

Seriale Prodotto:

Centro:

Figure 11: Register a batch.



Consegne del Team

Codice Cliente	ID Ordine	Stato	Data Consegna Prevista	Codice Team	Codice Chief
C666	6666	in_progress	2025-06-14	LT666	CL666
CU05	O51	in_progress	2025-06-13	LT01	CLO01
CU05	O52	shipped	2025-06-15	LT01	CLO01
CU10	O101	in_progress	2025-06-13	LT01	CLO01
CU10	O102	shipped	2025-06-15	LT01	CLO01
CU01	O11	in_progress	2025-06-13	LT02	CLO02
CU01	O12	shipped	2025-06-15	LT02	CLO02
CU06	O61	in_progress	2025-06-13	LT02	CLO02
CU06	O62	shipped	2025-06-15	LT02	CLO02
CU02	O21	in_progress	2025-06-13	LT03	CLO03
CU02	O22	shipped	2025-06-15	LT03	CLO03
CU07	O71	in_progress	2025-06-13	LT03	CLO03
CU07	O72	shipped	2025-06-15	LT03	CLO03
CU03	O31	in_progress	2025-06-13	LT04	CLO04
CU03	O32	shipped	2025-06-15	LT04	CLO04
CU08	O81	in_progress	2025-06-13	LT04	CLO04
CU08	O82	shipped	2025-06-15	LT04	CLO04
CU04	O41	in_progress	2025-06-13	LT05	CLO05

Figure 12: Teams deliveries.