JAVA INSTITUTE for Advanced Technology

INNOVATIONS FOR THE KNOWLEDGE SOCIETY

ISO 9001:2008 Certified Institute

Java Institute for Advanced Technology

BUSINESS COMPONENT DEVELOPMENT II
JIAT/BCD II
JIAT/BCD II/EX/01

OSHADHA KAVINTHA
SCN NO
200027703100

Real-Time Cargo Tracking Services

Role and Impact on Logistics

Real-time cargo tracking is a critical component of modern logistics management systems, and its importance cannot be overstated. In the context of a transportation logistics company, real-time tracking enables continuous monitoring of cargo as it moves through the supply chain. This capability enhances supply chain visibility, customer satisfaction, and operational efficiency, making it a cornerstone of effective logistics management.

Supply Chain Visibility

Real-time tracking offers unparalleled visibility into the supply chain. By providing up-to-the-minute information about the status of cargo, users can make informed decisions promptly. This visibility helps in identifying and addressing potential issues before they escalate, such as delays, route deviations, or unforeseen disruptions. Enhanced visibility leads to better planning and coordination among various components of the supply chain, resulting in a more resilient and responsive system.

The integration of real-time tracking would involve the following components:

Entities like Route, Vehicle, and OrderToManage would be designed to store and manage tracking data. The Route entity could be extended to include real-time tracking fields such as status updates, and timestamps.

Customer Satisfaction

Customer satisfaction is significantly enhanced by the transparency and reliability provided by real-time tracking.

Servlets and RESTful APIs can be developed to provide real-time tracking information to users. For example, a GetShipmentStatus servlet could expose endpoints that return the status of a shipment based on the data stored in the database.

A user-friendly interface could be created to display real-time tracking information, allowing users to monitor their shipments via web application.

Operational Efficiency

Operational efficiency is another significant benefit of real-time tracking. By having real-time data, company can optimize route planning, reduce fuel consumption, and minimize delays. Predictive analytics and automated alerts can be implemented to foresee potential issues, enabling proactive measures that streamline operations.

Implementation in the application structure would include, business logic to process tracking data, optimize routes, and manage exceptions. Services like RouteOptimizationWithOrder and RouteOptimizationWithVehicle can be extended to include functionalities that analyze real-time data and optimize logistics operations.

Efficient data management strategies, including proper indexing and data retrieval methods, are crucial for handling real-time tracking data.

Route Optimization with Interceptor Classes

Concepts of Route Optimization in Logistics

Route optimization in logistics refers to the process of determining the most efficient path for transportation to minimize costs, reduce transit times, and enhance service quality. This involves leveraging advanced algorithms and real-time data to optimize routes based on various factors such as distance, vehicle capacity, and delivery time windows. Effective route optimization can lead to significant improvements in fuel efficiency, reduced carbon emissions, and better utilization of resources, thereby enhancing the overall efficiency and sustainability of logistics operations.

Interceptor Methods and Their Contribution

In the context of route optimization, interceptors can be particularly useful for ensuring that the routes and orders being processed adhere to certain business rules before the actual processing logic is executed.

Order Validation Interceptor

The OrderValidationInterceptor ensures that only valid orders are considered for a particular route. This interceptor checks each order against the route's destination, and only allows the method invocation to proceed if there are valid orders that match the route's criteria.

```
@Interceptor
@CheckOrderValidity
public class OrderValidationInterceptor {
   @AroundInvoke
   public Object intercept(InvocationContext ic) throws Exception {
      Object[] parameters = ic.getParameters();
      List<OrderToManage> allOrderList = (List<OrderToManage>) parameters[0];
      Route route = (Route) parameters[1];
      List<OrderToManage> optimizedOrderList = new ArrayList<>();
      for (OrderToManage order : allOrderList) {
         if (isValidOrder(order, route)) {
               optimizedOrderList.add(order);
         }
      }
      if (optimizedOrderList.size() > 0) {
         return ic.proceed();
      } else {
         throw new EmptyOptimizeDataListException("Empty optimize order List");
      }
   }
   private boolean isValidOrder(OrderToManage order, Route route) {
      String customer_address = order.getCustomer_id().getAddress().trim().toLowerCase();
      String route_destination = route.getDestination().trim().toLowerCase();
      return customer_address.contains(route_destination); }
}
```

Vehicle Validation Interceptor

The VehicleValidationInterceptor ensures that the route to be taken by the vehicle is valid based on certain criteria, such as the total weight of products and the route distance. This interceptor fetches the route details, calculates the total weight of the products, and proceeds with the method invocation only if the criteria are met.

```
@Interceptor

@CheckVehicleValidity

public class VehicleValidationInterceptor {

   @PersistenceContext

   private EntityManager em;

   @AroundInvoke

   public Object intercept(InvocationContext ic) throws Exception {

      Object[] parameters = ic.getParameters();

      Route route = (Route) parameters[0];

      double tot = 0;

      TypedQuery<Route> query = em.createQuery(

            "SELECT DISTINCT r FROM Route r LEFT JOIN FETCH r.orderToManages WHERE r.route_id
= :routeId",  Route.class);

      query.setParameter("routeId", route.getRoute_id());

      List<Route> routes = query.getResultList();

      for (Route fetchedRoute : routes) {

         for (OrderToManage order : fetchedRoute.getOrderToManages()) {

            for (Product p : order.getProducts()) {

               tot += p.getTotal_weight();

            } }}

      if (tot > 0 && route.getDistance()>10) { return ic.proceed();

      } else { throw new IncorrectRouteDataException("Incorrect Route Data!");

      }}

}
```

Integration and Benefits

These interceptors can be integrated into the EJB module where business logic is implemented. The OrderManageService can use these interceptors to validate orders and routes during the route planning process.

Benefits

- Interceptors provide a centralized approach to validation, ensuring that the business rules are consistently applied across multiple EJB methods.
- By using interceptors, validation logic is separated from business logic, making the codebase cleaner and easier to maintain.
- Interceptors enable dynamic validation based on current data, allowing the system to adapt to changing conditions and optimize routes in real-time.

Dynamic Transaction Management

Critical Role of Transaction Demarcation Management in Logistics Systems

Transaction demarcation management is crucial in logistics systems for ensuring the consistency and reliability of operations that span multiple resources or actions. In logistics, where processes are interconnected and often interdependent, transaction management ensures that a series of operations are executed successfully and consistently, or not at all, maintaining the integrity of the system.

In the provided logistics management system, transaction management plays a pivotal role in operations such as adding shipments, updating shipment statuses, assigning orders to routes, and integrating vehicle information. Each of these operations may involve multiple database updates and validations that must be completed successfully to maintain the correctness of the logistics data.

Key Aspects of Transaction Management in Logistics

- Atomicity- Ensures that all steps in a transaction are completed successfully. If any step fails, the transaction is rolled back, leaving the system in its original state. For example, when adding a new shipment, updating the order status, and assigning the shipment to a route must all succeed together.
- Consistency- Guarantees that a transaction transforms the system from one consistent state to another, adhering to predefined rules and constraints. For instance, updating the status of a shipment to "DELIVERED" should only occur if the shipment has been properly tracked and verified.

- Isolation- Ensures that concurrently executing transactions do not interfere with each other. This is critical in logistics to prevent issues such as double-booking resources or conflicting updates to the same shipment.
- Durability- Ensures that once a transaction is committed, the changes are permanent, even in the event of a system failure. This is essential for maintaining accurate records of shipments and routes.

Potential Impact of Transaction Inaccuracies

Transaction inaccuracies can have significant adverse effects on logistics operations, impacting delivery timelines and resource utilization.

Examples of Potential Impacts:

- If transaction management fails to ensure the correct update of shipment statuses, shipments may be incorrectly marked as delivered or pending. This can lead to delays in processing and delivering shipments, disrupting the supply chain.
- Incorrect transaction handling can lead to inaccurate assignment of orders to routes or vehicles. For instance, if an order is not correctly assigned to a route, it may lead to underutilized or overburdened routes, inefficient use of vehicles, and increased operational costs.
- Inaccurate transactions can result in inconsistent data across the system. For example, if a shipment addition transaction fails halfway, the system might show the shipment as added in some parts of the application while missing in others, leading to confusion and potential loss of cargo.

Implementation

OrderManageService and RouteService, transaction demarcation can be managed using container-managed transactions provided by EJB.

```
@Stateless
public class OrderManageServiceImpl implements OrderManageService {
   @PersistenceContext
   private EntityManager em;
   @Override
   @RolesAllowed("admin")
   @TransactionAttribute(TransactionAttributeType.REQUIRED)
   public Long addShipment(Long customer_id) {
      OrderToManage orderToManage = new OrderToManage();
      orderToManage.setCustomer_id(em.find(Customer.class, customer_id));
      orderToManage.setStatus(OrderStatus.PENDING);
      Date currentDate = new Date(System.currentTimeMillis());
      Timestamp currentTimestamp = new Timestamp(currentDate.getTime());
      orderToManage.setCreated_at(currentTimestamp);
      try {
         em.persist(orderToManage);
         em.flush();
         em.refresh(orderToManage);
         Long orderId = orderToManage.getOrder_id();
         return orderId;
      } catch (Exception e) {
         e.printStackTrace();
         return 0L;
      }
   }
```

```
@Override
@RolesAllowed({"admin", "employee"})
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public boolean updateStatus(Long orderId, String status) {
   OrderToManage order = em.find(OrderToManage.class, orderId);
   if (order == null) {
      return false;
   }
   switch (status) {
      case "PENDING":
      case "PROCESSING":
      case "SHIPPED":
      case "DELIVERED":
      case "CANCELLED":
      case "ON_HOLD":
      case "RETURNED":
      case "REFUNDED":
      case "COMPLETED":
         order.setStatus(OrderStatus.valueOf(status));
         break;
      default:
         return false;
   }
try {

          em.merge(order);

          return true;

       } catch (Exception ex) {

          return false;

       }

   }

}
```

In this implementation, the @TransactionAttribute(TransactionAttributeType.REQUIRED) annotation ensures that each method runs within a transaction. If any exception occurs, the transaction will be rolled back, preventing partial updates and maintaining data consistency.

Cargo Data Security

Critical Analysis of Security Measures

Cargo data security in logistics management systems is paramount, given the sensitive nature of the data involved. Protecting this data involves addressing various challenges such as data privacy concerns, securing communication channels, and safeguarding against cyber threats.

Data Privacy Concerns

Data privacy is crucial in logistics, as it involves handling sensitive information about cargo, customers, routes, and transactions. Ensuring that only authorized personnel can access and modify this data is vital to maintaining confidentiality and trust.

Role-Based Access Control

The use of @RolesAllowed annotations in the OrderManageServiceImpl class ensures that only users with specific roles (e.g., admin, employee) can access certain methods.

```
@RolesAllowed({"admin", "employee"})

public List&lt;OrderToManage&gt; getShipments() {

}

@RolesAllowed("admin")

public Long addShipment(Long customer_id) {

}
```

This restricts access to sensitive operations based on the user's role, thereby protecting data from unauthorized access.

Secure Communication Channels

Secure communication channels are essential to prevent data interception during transmission between clients and the server.

Web Application Security Configuration

The web.xml configuration file defines security constraints and form-based authentication, ensuring that only authenticated users can access certain URLs.

```
<security-constraint>

   <web-resource-collection>

      <web-resource-name>Basic</web-resource-name>

      <url-pattern>/home/*</url-pattern>

      <url-pattern>/home.jsp</url-pattern>

      <http-method>GET</http-method>

      <http-method>POST</http-method>

   </web-resource-collection>

  <auth-constraint>

      <role-name>admin</role-name>

      <role-name>employee</role-name>

   </auth-constraint>

</security-constraint>
```

```
<login-config>

   <auth-method>FORM</auth-method>

   <realm-name>file</realm-name>

   <form-login-config>

      <form-login-page>/index.jsp</form-login-page>

      <form-error-page>/error.jsp</form-error-page>

   </form-login-config>

</login-config>
```

```
<error-page>

   <error-code>403</error-code>

   <location>/403.jsp</location>

</error-page>
```

This setup ensures that only authenticated and authorized users can access specific parts of the application.

Protection Against Cyber Threats

Protecting against cyber threats involves implementing measures to detect, prevent, and respond to various types of attacks such as SQL injection and other vulnerabilities.

Entity Validation and Sanitization

Implementing proper validation and sanitization in the data layer is essential. For example, in the OrderManageServiceImpl class, methods interacting with the database should include checks and validations.

```
@Override

@RolesAllowed({"admin", "employee"})

public boolean updateStatus(Long orderId, String status) {

   OrderToManage order = em.find(OrderToManage.class, orderId);

   if (order == null) {

      return false;

   }



   // Validate status

   if (!EnumUtils.isValidEnum(OrderStatus.class, status)) {

      return false;

   }
order.setStatus(OrderStatus.valueOf(status));
    try {
        em.merge(order);
        return true;
    } catch (Exception ex) {
        return false;
    }
}
```

Challenges Specific to Cargo Data

01. Interoperability and Data Sharing

Securely sharing data between different stakeholders like suppliers, carriers, and customers. Without exposing it to unauthorized access requires robust encryption and secure API design.

02. Scalability of Security Measures

As the volume of data and the number of transactions grow, the security measures must scale accordingly without compromising performance.

Exception Handling in Logistics Operations

Importance of Quick and Effective Response Mechanisms

In logistics operations, errors can have significant consequences, disrupting the smooth flow of goods and services, leading to delays, increased costs, and reduced customer satisfaction. Effective exception handling mechanisms are crucial to swiftly address and mitigate these errors, ensuring that operations can continue with minimal interruption.

Consequences of Errors in Logistics

1. **Delays and Inefficiencies**
    - Errors in route planning or order optimization can cause significant delays. For instance, the EmptyOptimizeDataListException is thrown when there are no valid orders to optimize, potentially stalling the entire shipping process.

```
if (optimizedOrderList.size() > 0) {

   return ic.proceed();

} else {

   throw new EmptyOptimizeDataListException("Empty optimize order List");

}
```

This scenario highlights the need for robust validation checks before proceeding with logistics operations.

2. **Resource Misallocation**
   - Incorrect data or route information can lead to inefficient use of resources. The IncorrectRouteDataException is thrown when the route data is invalid, preventing further processing of potentially flawed operations.

```
if (tot > 0 && route.getDistance() > 10) {

   return ic.proceed();

} else {

   throw new IncorrectRouteDataException("Incorrect Route Data!");

}
```

3. **Customer Dissatisfaction**
   - Errors that result in delayed or incorrect deliveries directly impact customer satisfaction. Prompt error detection and resolution can prevent these negative outcomes, maintaining trust and reliability in the logistics service.

Effective Response Mechanisms

Proactive Exception Handling

- By implementing interceptors such as OrderValidationInterceptor and VehicleValidationInterceptor, potential issues are caught early in the processing pipeline

```
@AroundInvoke

public Object intercept(InvocationContext ic) throws Exception {

   // Validation logic

}
```

Clear Error Messaging

- Custom exceptions like EmptyOptimizeDataListException and IncorrectRouteDataException provide clear and specific error messages. This aids in quick diagnosis and resolution of issues, as the exact nature of the problem is immediately apparent.

```
public class EmptyOptimizeDataListException extends RuntimeException {

   public EmptyOptimizeDataListException(String message) {

      super(message);

   }

}



public class IncorrectRouteDataException extends RuntimeException {

   public IncorrectRouteDataException(String message) {

      super(message);

   }

}
```

Modular Component Organization for Logistics Software

Best Practices for Component Organization

Separation of Concerns

- EAR- The ear module should handle the overall packaging and deployment of the application. It integrates different components like EJB, Web, and resources, ensuring that the entire application is packaged cohesively.
- EJB- The ejb module is responsible for business logic and transactional operations. It includes services, entities, exceptions, interceptors, and timers.
- Web- The web module handles the presentation layer, managing servlets, JSP files, and other web resources. It ensures that all web-related functionalities are kept separate from business logic.

Layered Architecture

- Organize code into distinct layers- Presentation, Business Logic, and Data Access. This separation enhances maintainability and scalability.
- Presentation Layer- Managed in the web module, it includes servlets, JSPs, and static resources like CSS and JS files.
- Business Logic Layer- Encapsulated in the ejb module, it includes services, entities, and business rules.
- Data Access Layer- Integrated within the ejb module, it includes entity classes and persistence logic.

Modularity

- Break down functionalities into distinct, reusable modules. This makes it easier to manage, test, and update individual components without affecting the entire system.
- For example, within the ejb module, separate packages for annotation, entity, exception, impl, interceptor, remote, and timer provide clear boundaries for different functionalities.

Naming Conventions and Structure

- Use meaningful and consistent naming conventions for packages and classes. This improves code readability and maintainability.

Advantages of Modular Component Organization

Ease of Maintenance

- Modular components make it easier to identify and fix issues. For example, changes to the business logic in the ejb module do not affect the presentation layer in the web module.

- Clear separation of concerns reduces the complexity of each module, making it easier for developers to understand and maintain the codebase.

Scalability

- Modularity allows the application to scale efficiently. As the logistics requirements evolve, new features can be added to specific modules without disrupting existing functionalities.
- For example, adding a new service in the ejb module or a new servlet in the web module can be done independently, facilitating easier scaling.

Reusability

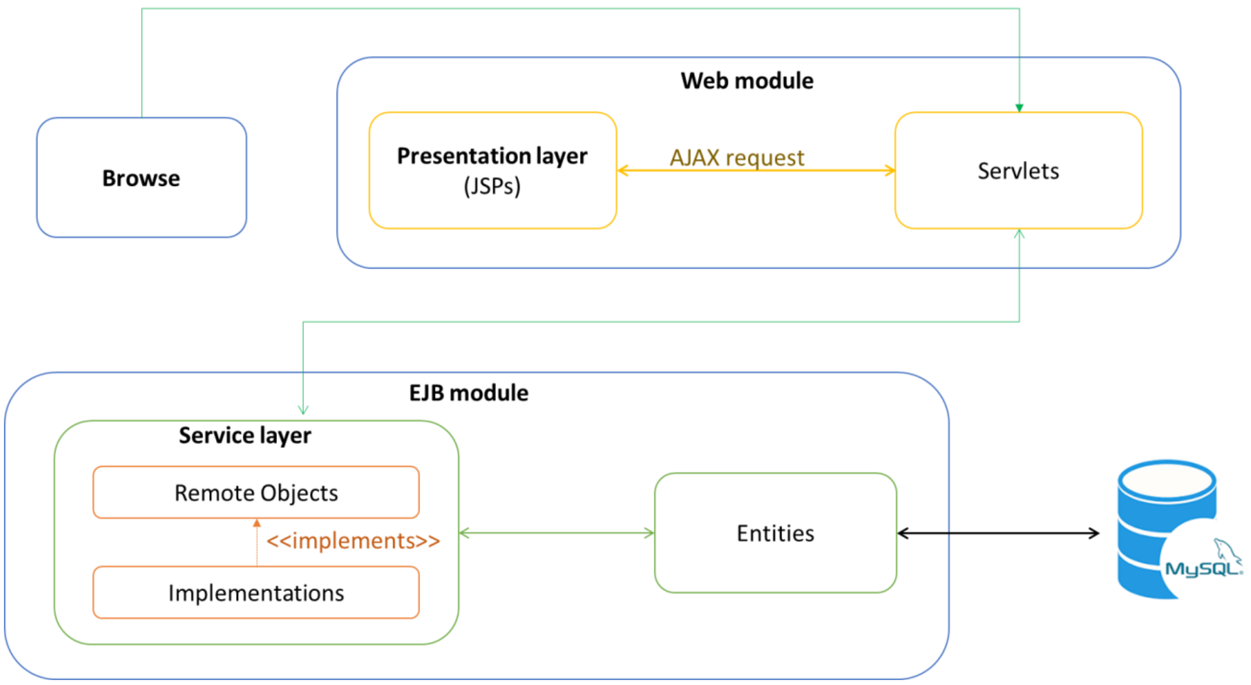
- Well-defined modules can be reused across different projects or within different parts of the same project. This reduces redundancy and promotes code reuse.
- For example, the annotation and interceptor packages in the ejb module can be reused in other parts of the application or even in other projects.

Improved Collaboration

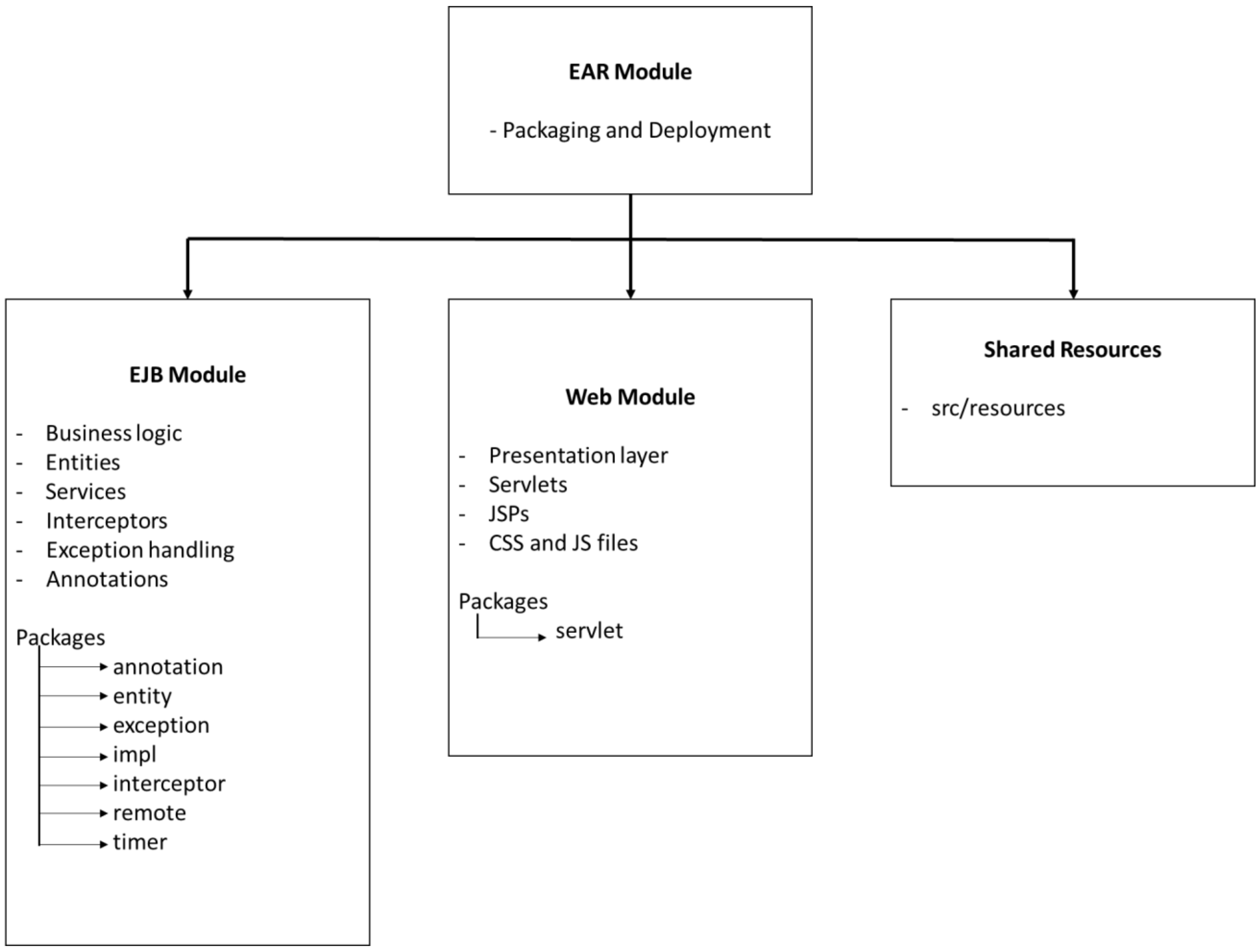
- Modular organization facilitates better collaboration among development teams. Teams can work on different modules simultaneously without interfering with each other's work.
- Clear boundaries between modules ensure that teams can focus on their specific areas, improving productivity and reducing conflicts.

Architectural Diagrams

01.


Browse
Web module
Presentation layer (JSPs)
AJAX request
Servlets
EJB module
Service layer
Remote Objects
<<implements>>
Implementations
Entities
MySQL

02.


EAR Module
- Packaging and Deployment
EJB Module
- Business logic
- Entities
- Services
- Interceptors
- Exception handling
- Annotations
Packages
annotation
entity
exception
impl
interceptor
remote
timer
Web Module
- Presentation layer
- Servlets
- JSPs
- CSS and JS files
Packages
servlet
Shared Resources
- src/resources