Java Institute for Advanced Technology

UNIT NAME - Business Component Development I

UNIT ID – JIAT/BCD I

ASSIGNMENT ID – JIAT/BCD I/EX/01

NAME – J.D Deruwan Chalithanga
SCN NO - 207979414
NIC - 200025902440
BRANCH – Colombo

Task 1

1. Analysis of the Scenario:

a. Challenges of Urban Traffic Clog and Pollution: Urban zones face various challenges related to traffic blockage and pollution. These include:

- Traffic blockage leading to expanded travel times, fuel utilization, and air pollution.
- Difficulty in managing traffic stream, particularly amid peak hours and extraordinary events.
- Lack of real-time data for effective decision-making and traffic management.
- Environmental concerns due to tall levels of vehicle emissions.

b. Key Requirements for the Smart Urban Traffic Management System: To address these challenges, the Smart Urban Traffic Management System should have the following key requirements:

- Real-time observing of traffic conditions utilizing IoT devices such as sensors, cameras, and GPS trackers.
- Data collection and analysis to identify traffic patterns, congestion hotspots, and potential bottlenecks.
- Intelligent traffic control mechanisms to optimize traffic flow and decrease congestion.
- Integration with existing transportation systems and infrastructure.
- Scalability to handle large volumes of data and activity management tasks.
- High accessibility to ensure persistent operation and minimal downtime.

c. Enterprise JavaBeans (EJB) can help address the versatility and availability challenges of the Smart Urban Traffic Management System in the following ways:

- EJB gives a component-based architecture that allows for the modular development of the system, making it easier to scale by including or removing components as needed.
- EJB containers manage the lifecycle of EJB components, providing features such as pooling, caching, and load balancing, which help improve scalability and availability.
- EJB supports clustering, permitting the system to be deployed across multiple servers to distribute load and ensure high availability.
- EJB's transaction management capabilities help ensure data consistency and integrity, which is crucial for a real-time traffic management system.

2. Design of the EJB Solution:

a. Architectural Design:

- Overall Architecture:

    1. Central Server: Acts as the core component, receiving data from IoT devices and coordinating traffic management activities.

    2. IoT Devices: Deployed throughout the urban range to collect real-time traffic data and communicate with the central server.

    3. Analytical Server: Analyzes information collected from IoT gadgets to recognize activity designs and give proposals to the central server.

- Main Components:

    1. Message-Driven Beans (MDBs): Process messages from IoT devices, activating actions such as traffic rerouting.

    2. Session Beans: Manage application state and process business logic, handling requests from the central server or analytical server.

    3. Entities: Represent the data model, storing persistent data such as traffic light states and vehicle locations.

b. Component Design

- Role of Each EJB Component:

    1. MDBs: Get sensor data from IoT gadgets and trigger actions based on the received messages.

    2. Session Beans: Handle requests from outside substances (e.g., central server, expository server) and arrange activity administration activities.

    3. Entities: Store and oversee tireless information related to activity administration, such as activity light states and vehicle information.

- Interactions Between Components and Outside Entities:

    1. IoT Gadgets: Send information to MDBs, which at that point handle the information and trigger actions.

    2. Central Server: Communicates with session beans to ask activity management actions based on data received from IoT devices.

3. Analytical Server: Gives insights and recommendations to the central server based on data analysis.

c. Functional Design

• MDBs: Receive sensor data, process it, and trigger actions such as altering traffic signal timings or starting traffic rerouting.

• Session Beans: Handle demands from external entities, coordinate traffic management activities, and ensure information consistency and integrity.

• Entities: Store and oversee determined information related to traffic management, such as traffic light states, vehicle information, and traffic patterns.

Task 2

1. Usage of Enterprise Application Model

Real-world Example:

In our Smart Urban Traffic Management System, we apply the Enterprise Application Model to improve the maintainability and scalability of our application. We utilize the Model-View-Controller (MVC) pattern to partitioned concerns and manage the interaction between components effectively.

o Model: Represents the data and business logic of the application. In our system, the model incorporates entities such as traffic lights, vehicles, and traffic patterns. These entities are managed by Enterprise JavaBeans (EJB) to ensure scalability and availability.

o View: Represents the introduction layer of the application. We use JavaFX for our user interface, which interacts with the model through controllers. The view displays real-time traffic information, such as traffic flow, congestion levels, and suggested routes.

o Controller: Manages the flow of data between the model and the view. Controllers handle user input, update the model accordingly, and refresh the view to reflect the changes. We use session beans as controllers in our application to manage traffic management activities and ensure data consistency.

```
4 usages
public class TrafficData {
    no usages
    private int vehicleSpeed;
    no usages
    private String trafficLightStatus;
    no usages
    private String gpsCoordinates;

}


no usages
public class TrafficController {
    no usages
    public void handleUserInput(String userInput) {

    }
}


no usages
public class TrafficView {
    no usages
    public void displayTrafficInfo(TrafficData trafficData) {

    }
}
```

2. Usage of Containers and Connectors

- Explanation of Containers and Connectors:
  Containers, such as EJB containers, and connectors, such as JDBC connectors, play crucial roles in our Smart Urban Traffic Management System to ensure scalability and availability.

  o Containers:
    EJB containers manage the lifecycle of EJB components, providing services such as transaction management, security, and concurrency control. They abstract the complexity of managing these services from the

application developer, allowing them to focus on business logic. Containers also handle the pooling and caching of EJB instances, which improves performance and scalability by reusing instances instead of creating new ones for each request.

o Connectors:

JDBC connectors provide a standardized way to connect to a database, enabling EJB components to access and manipulate persistent data. They handle database connections, transactions, and error handling, guaranteeing that database operations are performed reliably and efficiently.

```
@Stateless
public class TrafficDataProcessorBean {
    no usages
    public void processTrafficData(TrafficData trafficData) {
        System.out.println(trafficData);
    }
}
```

Features Enhancing Scalability

o Clustering:

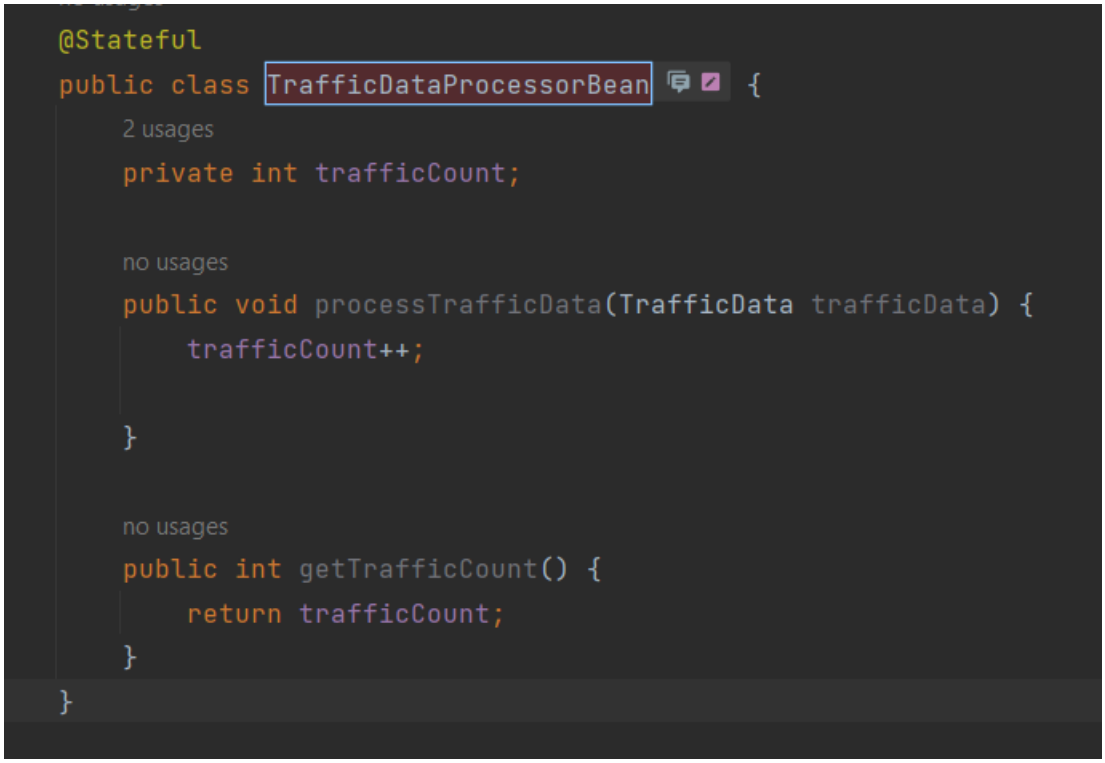
EJB containers can be configured in a clustered environment to disseminate traffic and improve performance. Clustering allows multiple instances of an application to run simultaneously, with requests distributed among them. This improves scalability by enabling the application to handle a larger number of requests.

o Load Balancing:

Load balancing is another feature that enhances scalability by distributing incoming traffic across multiple servers. EJB containers can be configured with a load balancer that redirects requests to different instances of the application based on factors such as server load and availability. This ensures that no single server is overwhelmed with requests, improving overall system performance and scalability.

o Failover and High Availability:

EJB containers can also be configured for failover and high availability. In case of a server failure, EJB containers can automatically redirect requests to another available server, ensuring that the application remains accessible and minimizing downtime. This improves the availability of the application and enhances scalability by reducing the impact of server failures on the overall system.



```
@Stateful
public class TrafficDataProcessorBean {
    2 usages
    private int trafficCount;

    no usages
    public void processTrafficData(TrafficData trafficData) {
        trafficCount++;

    }

    no usages
    public int getTrafficCount() {
        return trafficCount;
    }
}
```

3. Advantages of Venture JavaBeans Component Model

o Scalability:

EJB gives a scalable component model that permits the application to handle extending traffic and data volume. EJB containers manage the lifecycle of comp nonets, enabling the application to be conveyed over multiple servers in a clustered environment. This guarantees that the application can scale horizontally by including more servers to the cluster.

Example: In our traffic management system, EJB components such as session beans can be conveyed across different servers in a clustered environment. This permits the system to handle a large number of requests from IoT gadgets and analytical servers, making strides scalability.

o Transaction Management:

EJB gives built-in back for exchange administration, guaranteeing that operations are performed molecularly and dependably. This is significant for the traffic management system, where data keenness and consistency are paramount.

o Security:

EJB gives a strong security demonstrate, allowing engineers to characterize security arrangements and get to controls for components. This guarantees that touchy operations and data are secured from unauthorized access.

o Resource Management:

EJB containers manage resources such as database connections and thread pools, optimizing asset usage and moving forward execution. This helps in efficiently managing the traffic data and analytical operations in the system.

Task 3

1. Dependency Injection

- Dependency Injection (DI) simplifies the management of dependencies, making the code more maintainable and extensible. For example, in our solution, DI allows us to easily swap implementations of traffic data processors without modifying the client code.

- Initial Context Lookup is used for resource management, such as obtaining database connections. However, it can introduce scalability and configuration challenges due to the overhead of creating and maintaining the Initial Context. In our solution, we minimized the use of Initial Context Lookup by leveraging DI for resource injection.

- DI vs Initial Context Lookup:

Aspect	DI	Initial Context Lookup
• Coupling	Loosely	Tightly

• Code Complexity	Reduce code complexity	Can lead to higher code complexity
• Modularity	Promotes a more modular architecture	Can leas to monolithic design if not carefully managed.

2. JMS API

    - JMS API provides a standardized way to send and receive messages, enhancing communication in our application.
        - For ex, JMS allows us to send traffic data from IoT devices to the central server asynchronously, improving responsiveness.

    - JMS plays a vital role in facilitating messaging and asynchronous communication in our solution. For instance, JMS is used to notify the central server of traffic incidents detected by IoT devices, enabling real-time incident management. These are the Key Use Cases of JMS

    - JMS API integrates seamlessly into our solution, providing a reliable messaging system. We faced challenges in configuring JMS providers but overcame them by following best practices. Code snippets demonstrating JMS usage include setting up JMS connections and sending/receiving messages.

3. Message-driven beans (MDB)

    - MDBs are integrated into our solution to handle asynchronous message processing, such as processing traffic data from IoT devices. Code snippets demonstrate how MDBs listen for incoming messages and process them asynchronously.

    - MDBs play a crucial role in achieving efficient asynchronous message processing in our solution. They allow us to,
        - Offload time-consuming tasks, such as traffic data analysis, to background processes, improving system responsiveness.

    - MDBs contribute to message handling efficiency by providing a scalable and efficient way to process messages asynchronously. They help in managing the flow of messages within our application, ensuring timely processing of traffic data. These are the benefits

Task 4

- A Singleton Session Bean provides a single instance per application, making it suitable for managing shared resources such as traffic data caches or configuration settings.

This ensures that critical resources are centrally managed and can be accessed efficiently by multiple components, contributing to scalability by reducing resource contention and improving availability by providing a single point of access and control.

- Advantages include centralized management of shared resources, simplified access to application-wide services, and improved performance due to reduced overhead of creating multiple instances.

  Compared to Stateful and Stateless Session Beans, Singleton Session Beans offer better resource management for shared data and configuration, as they maintain their state throughout the application's lifecycle.

- Examples in UTMS

  - Singleton Session Bean can be used to manage a central cache of traffic data, ensuring that all components access the most up-to-date information.
  - It can also be used to coordinate traffic light control across intersections, ensuring that timing sequences are synchronized and optimized for traffic flow.
  - Additionally, a Singleton Session Bean can handle system-wide events, such as coordinating emergency vehicle routes or managing system-wide alerts for traffic incidents.