BUSINESS COMPONENT DEVELOPMENT - I
(CMP6222)

Student First Name	**: ABDUL RAHUMAN**
Student Last Name	**: MUHAMMED HAZEEM**
BCU No.	**: 22178960**
NIC No.	**: 199835410549**

Contents

1. Usage of Enterprise Application Model……………………………………………………………3
2. Usage of Containers and Connectors…………………………………………………………………4
3. Advantages of Enterprise JavaBeans Component Mode…………………………………………5
4. The difference between dependency injection and initial context lookup……………6
5. The importance of JMS API for the enterprise-level application……………………………7
6. Usage of the message-driven beans (MDB)…………………………………………………………8
7. What is a suitable Session bean type for the implementation of a web application representing the number of times a web page has been accessed? Compare and contrast your implementation among other bean types…………………………………………9
8. Referance……11

1. Usage of Enterprise Application Model

In the scenario, the implementation involves the use of an IoT device to capture and transmit sensor data to a messaging server for processing with an analytical server. The Enterprise Application Model (EAM) can be used to design and implement the application by defining the business processes, entities, services, and infrastructure components.

Business Processes: The business processes may include the following activities:

Data collection: Collecting sensor data from the IoT device.

Data processing: Transforming and aggregating the sensor data.

Data analysis: Analyzing the sensor data to generate insights and predictions.

Business Entities: The business entities may include the following objects and data:

Sensor data: Voltage, frequency, and current readings from the IoT device.

Analytical data: Calculated values such as total power consumption, average current, average voltage, and average frequency.

User data: User information such as login credentials, preferences, and settings.

Services: The services may include the following functionality and interfaces:

Data access service: Providing access to the sensor data and analytical data.

Messaging service: Transmitting sensor data from the IoT device to the messaging server.

Analytics service: Analyzing the sensor data and generating analytical data.

Infrastructure: The infrastructure may include the following hardware, software, and network components:

IoT device: Capturing and transmitting sensor data.

Messaging server: Receiving and routing sensor data to the analytical server.

Analytical server: Processing and analyzing the sensor data to generate analytical data.

Database server: Storing and retrieving sensor data, analytical data, and user data.

Network components: Providing connectivity between the IoT device, messaging server, analytical server, and database server.

Using the EAM can help to ensure that the application is designed and built according to best practices and standards. It can also help to ensure that the application is scalable, maintainable, and extensible. Additionally, the EAM can help to facilitate communication and collaboration between different teams and stakeholders involved in the application development process.

2. Usage of Containers and Connectors

Containers and connectors are essential components for building a scalable, reliable, and flexible system. Here's how they could be used in the solution for the scenario of a Smart Energy Management System:

Containers:

Containers are lightweight, portable, and self-contained runtime environments that can be easily deployed and scaled. In the Smart Energy Management System, containers can be used to host various microservices or components of the system, including the messaging server, analytical server, and IoT devices. Each container can be configured with its own resources, dependencies, and environment variables, which makes it easier to manage and update the system.

For example, the messaging server can be deployed in a container that is isolated from other components, such as the analytical server or the database, to ensure fault isolation and scalability. Similarly, each IoT device can run in its own container with a predefined set of resources and configurations, which makes it easier to manage and update the devices.

Connectors:

Connectors are software components that facilitate communication and data exchange between different parts of the system. In the Smart Energy Management System, connectors can be used to link the IoT devices to the messaging server, the messaging server to the analytical server, and the analytical server to the database or other systems.

For example, a connector can be used to enable communication between the IoT devices and the messaging server. The messaging server can then use a connector to queue and distribute the data to the analytical server, which can use analyze and process the data. Finally, the analytical server can store the results in a database such as MySQL using a connector.

In summary, containers and connectors can be used to build a robust, scalable, and flexible Smart Energy Management System that can efficiently collect, process, and analyze data from IoT devices. By using containers, the system can be easily managed, updated, and scaled, while connectors can enable seamless communication and data exchange between different components.

3. Advantages of Enterprise JavaBeans Component Mode

EJB has several advantages as a component model for enterprise applications:

Component-based architecture: EJB is a component-based architecture that allows developers to build modular, reusable components for their applications. This can help to simplify application development, testing, and maintenance, and can make it easier to build and scale large enterprise applications.

Transaction management: EJB provides built-in transaction management capabilities, which can help to ensure that database updates and other operations are atomic, consistent, isolated, and durable (ACID). This can help to improve data consistency and reliability, and can make it easier to manage complex business logic.

Security: EJB provides built-in security features, such as authentication, authorization, and role-based access control (RBAC), which can help to ensure that only authorized users can access sensitive data or perform critical operations. This can help to improve the overall security and compliance of enterprise applications.

Distributed computing: EJB supports distributed computing, which means that components can be deployed across multiple servers or clusters and communicate with each other seamlessly. This can help to improve application scalability and availability, and can make it easier to build and maintain complex distributed systems.

Interoperability: EJB is based on industry standards, such as the Java EE platform, which means that components built using EJB can interoperate with other Java-based technologies and frameworks. This can help to ensure that applications can integrate with existing systems and can be easily extended or customized as needed.

4. The difference between dependency injection and initial context lookup

Dependency injection and initial context lookup are two different approaches for managing object dependencies in Java EE applications. Here's a brief explanation of each approach and their differences:

Dependency Injection:

Dependency injection (DI) is a design pattern that allows objects to be loosely coupled by injecting their dependencies (i.e., other objects they need to function) at runtime. In other words, instead of an object creating its dependencies, it receives them from an external source. This makes objects more modular, reusable, and easier to test.

DI can be implemented in various ways, such as constructor injection, setter injection, or field injection. In a Java EE application, DI can be implemented using the CDI (Contexts and Dependency Injection) framework or the Spring Framework.

Initial Context Lookup:

Initial Context Lookup is a way to locate objects or resources in a Java EE application by performing a lookup on the application server's naming and directory service (JNDI). When an object is looked up, the application server returns a reference to the object, which can be used to invoke its methods or access its properties.

Initial context lookup can be used to obtain a wide range of resources in a Java EE application, such as data sources, EJBs, JMS queues, and JNDI-defined objects. However, it is not a good approach for managing object dependencies since it tightly couples objects to the JNDI environment and makes them harder to test and reuse.

5. The importance of JMS API for the enterprise-level application

Java Message Service (JMS) API is a key component of the Java Enterprise Edition (Java EE) platform, and it provides a standardized way to exchange messages between distributed applications in a reliable, asynchronous, and loosely coupled manner. JMS is a messaging middleware technology that enables communication between different components of an enterprise-level application or between different applications in a distributed system. Here are some of the key benefits of using JMS API in enterprise-level applications:

Asynchronous communication:

JMS API allows for asynchronous communication between applications, which means that the sender and receiver applications do not have to be running at the same time. This allows for better scalability, fault-tolerance, and responsiveness of the system.

Reliability:

JMS API provides a reliable messaging system that ensures that messages are delivered to the intended recipients, even if there are failures or network issues. This is achieved through features such as message persistence, transactional messaging, and message acknowledgment.

Loose coupling:

JMS API enables loose coupling between different components of an enterprise-level application by providing a standardized interface for communication. This allows for greater flexibility, modularity, and maintainability of the system.

Scalability:

JMS API provides support for different messaging models, such as point-to-point (queue) and publish-subscribe (topic), which allows for better scalability and performance of the system.

Integration:

JMS API provides a standardized interface for integrating with other messaging systems, such as IBM MQ, Apache ActiveMQ, and RabbitMQ. This makes it easier to integrate with other systems in a heterogeneous environment.

Here's how we can implement this using JMS API:

- Create a JMS Producer to send messages from the IoT device to the messaging server.
- JMS Producer to send the sensor data to the messaging server.
- Create a JMS Consumer to receive messages from the messaging server on the analytics server.

6. Usage of the message-driven beans (MDB).

message-driven beans (MDBs) can be used to handle the communication between the IoT device and the messaging server of the Smart Energy Management System (SEMS). The MDBs can be used to receive and process messages from the IoT device, which can include sensor data and status updates.

The following are some ways in which MDBs can be used in the SEMS solution:

Receive sensor data: MDBs can be used to receive the sensor data from the IoT device, which can include voltage, frequency, and current readings. The MDB can be configured to listen to a specific message queue or topic, where the device sends the sensor data.

Process sensor data: Once the MDB receives the sensor data, it can be processed and analyzed to calculate the total power consumption, average current, average voltage, and average frequency. The MDB can use the session beans or entity beans to perform the necessary calculations and store the results in the database.

Trigger alerts: The MDB can also be used to trigger alerts or notifications based on specific sensor readings or conditions. For example, if the voltage reading falls below a certain threshold, the MDB can send an alert message to the messaging server or to a specific user or group.

Manage communication: MDBs can manage the communication between the IoT device and the SEMS messaging server, ensuring reliable and efficient communication. The MDB can handle message retries, timeouts, and errors, and can provide feedback and status updates to the device and the messaging server.

By using MDBs, the SEMS solution can provide an efficient and scalable architecture for handling the messaging and communication between the IoT device and the analytical server. The use of MDBs allows for asynchronous and reliable communication, while providing a high level of fault tolerance and scalability.

7. **What is a suitable Session bean type for the implementation of a web application representing the number of times a web page has been accessed? Compare and contrast your implementation among other bean types.**

For the implementation of a web application that represents the number of times a web page has been accessed, a suitable Session bean type is the Stateful Session Bean.

Stateful Session Beans maintain state for a specific client across multiple requests. In this case, we can maintain the number of times a web page has been accessed as a state variable in the bean. Each time the web page is accessed, we can increment the state variable and return the updated count to the client.

Stateful Session Beans provide a number of advantages over other bean types:

- Client-specific state: Stateful Session Beans maintain state for a specific client across multiple requests. This makes them suitable for use cases where we need to maintain state across multiple requests, such as counting the number of times a web page has been accessed.
- Efficient communication: Stateful Session Beans allow for efficient communication between the client and the server, as they maintain a single instance of the bean for each client. This reduces the overhead of creating and destroying beans for each request.
- Optimized performance: Stateful Session Beans can optimize performance by caching data that is frequently used by the client. This can reduce the number of database or network calls required, improving overall performance.

In contrast, Stateless Session Beans do not maintain state across requests, and are typically used for performing lightweight operations that do not require state. They are ideal for use cases where multiple clients can access the same bean instance simultaneously, and the operations performed by the bean do not rely on any previous state.

Finally, Entity Beans are used to represent persistent data, and are typically used for accessing data from a database. While Entity Beans can maintain state, they are not suitable for use cases where the state needs to be maintained across multiple requests, as they do not have a specific client associated with them.

In summary, the Stateful Session Bean is the most suitable bean type for the implementation of a web application that represents the number of times a web page has been accessed, as it allows for the efficient maintenance of client-specific state across multiple requests.

8. References

https://docs.oracle.com/cd/E24329_01/web.1211/e24446/ejbs.htm#INTRO315