BUSINESS COMPONENT DEVELOPMENT - I
(CMP6222)

Student First Name : M.R.P.N.Tharuksha

Student Last Name : Rajapaksha

BCU No. : 22178965

NIC No. : 200019401866

Acknowledgment

First and foremost, I would like to publicly thank Mr. Achintha Chamara, my lecturer, and adviser for the business component development I subject, for his unwavering encouragement, inspiration, and guidance in helping me finish this report.

I am extremely grateful to my parent for their love, care, and sacrifices they made to build a better future for me. And the encouragement they have given me to complete this report.

Finally, I want to express my gratitude to my friends, the management of Java Institute, and everyone who supports me to complete this report.

Abstract

This research is about the EJB (Enterprise JavaBeans) & the J2EE (Java 2 Platform, Enterprise Edition) technologies and describes the EJB concepts, principles, terminology, and components.

Furthermore, this research discusses the usage of the enterprise application model, usage of containers and connectors, and advantages of the Enterprise JavaBeans Component Model, with real-world examples and snippet codes. And, critically analyzes the difference between dependency injection and initial context lookup, the importance of JMS API for the enterprise-level application, and the usage of the message-driven beans (MDB). And also, critically describes the suitable session bean type for the implementation of a web application representing the number of times a web page has been accessed and compares and contrasts it among other bean types.

Content

1. Glossary

- **CRM:** Customer Relationship Management
- **EAM:** Enterprise Application Model
- **EJB:** Enterprise JavaBeans
- **ERP:** Enterprise Resource Planning
- **IoT:** Internet of Things
- **J2EE:** Java 2 Platform, Enterprise Edition
- **JMS:** Java Messaging Service
- **JNDI:** Java Naming and Directory Interface
- **JPA:** Java Persistence API
- **JSP:** Java Server Pages
- **JTA:** Java Transaction API
- **JVM:** Java Virtual Machine
- **MDB:** Message-Driven Beans
- **MOM:** Message-Oriented Middleware
- **SFSB:** Stateful Session Bean

2. Introduction:

This research discusses the usage of the enterprise application model, usage of containers and connectors, advantages of the Enterprise JavaBeans Component Model, the difference between dependency injection and initial context lookup, the importance of JMS API for the enterprise-level application, usage of the message-driven beans (MDB), the suitable session bean type for the implementation of a web application representing the number of times a given web page has been accessed.

3. Task 2 – Descriptive Explaining

3.1. Usage of the Enterprise Application Model

A design pattern that is frequently used when creating enterprise-level apps is called Enterprise Application Model (EAM). It is a thorough model that outlines the components, interactions, and data flow of an application's structure.

The fundamental purpose of EAM is to make it simpler to design, implement, and maintain the system by giving a clear understanding of the application architecture. For creating intricate systems with numerous components, such as large-scale web applications, enterprise resource planning (ERP) systems, and customer relationship management (CRM) systems, EAM is very helpful.

Online banking is one example of a real-world enterprise application that makes use of EAM. A web-based user interface, a database server, and several back-end services to handle transactions, account management, and authentication are typical components of such a system. The EAM would specify exactly how these components interacted with one another.

Here is an example of J2EE code that uses EAM to demonstrate its use:

```
public class BankService {
    private AccountDAO accountDao;
    private TransactionDAO transactionDao;
    private AuthenticationService authService;

    public BankService(AccountDAO accountDao, TransactionDAO
transactionDao, AuthenticationService authService) {
        this.accountDao = accountDao;
        this.transactionDao = transactionDao;
        this.authService = authService;
    }

    public void transferMoney(String fromAccount, String toAccount,
double amount) throws InsufficientFundsException {

        User user = authService.authenticate();
        if (user == null) {
            throw new UnauthorizedAccessException();
        }
```

```
        Account from = accountDao.getAccount(fromAccount);
        Account to = accountDao.getAccount(toAccount);

        if (from.getBalance() < amount) {
            throw new InsufficientFundsException();
        }

        from.setBalance(from.getBalance() - amount);
        to.setBalance(to.getBalance() + amount);
        transactionDao.createTransaction(from, to, amount);
    }
}
```

The BankService class in this illustration depicts a back-end service that manages transactions for the online banking system. The class is dependent on the AccountDAO, TransactionDAO, and AuthenticationService subsystems. The dependency injection design pattern is used to inject these dependencies through the constructor.

Money transfer between two accounts is likewise a feature of the BankService class. The function fetches the accounts using the AccountDAO, authenticates the user using the AuthenticationService, and then verifies that there are enough funds. The method executes the transaction and adds a new record to the TransactionDAO if all requirements are satisfied.

With explicitly defined components and relationships between them, this example shows how EAM may be used to define the structure of an enterprise application.

3.2. Usage of Containers and Connectors

“Two key elements of the Java 2 Platform, Enterprise Edition (J2EE) architecture that aid in building scalable, portable, and distributed applications are containers and connectors.”

An environment called a container offers the architecture and management tools required to run and maintain J2EE components including servlets, JSPs, and EJBs. The container offers features including resource pooling, lifecycle management, transaction management, and security.

A connector is a pluggable component that offers a defined interface for fusing J2EE applications with legacy systems and databases. By abstracting away the specifics of the underlying systems and offering a standard API for the J2EE application, connectors make integration simpler.

Here is an illustration of how to use connectors and containers:

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.*;

public class MyServlet extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse
response) {
      Context ctx = new InitialContext();
      DataSource ds = (DataSource)
ctx.lookup("java:comp/env/jdbc/mydb");
   }
}
```

Consider a J2EE web application that demands access to a legacy database architecture. By using a connector, we can hide the internal workings of the database and offer a uniform interface for accessing it. To give the application a runtime environment and control its lifecycle, we can utilize a container.

Here is a sample of code that shows how to use a connector and a container in a J2EE application:

The lifecycle of our servlet is managed in the code snippet above utilizing a servlet container (supplied by the J2EE server). Additionally, we are utilizing a connector (described in the deployment descriptor) to connect to a datasource that hides the internal workings of the database system. As a result, we are able to develop application code using a standardized API without having to worry about the specifics of the supporting systems.

3.3. Advantages of the Enterprise JavaBeans Component Model

“A component approach for creating distributed, scalable, and secure corporate applications is called Enterprise JavaBeans (EJB).” Developers can create reusable business logic components using the interfaces and contracts defined by the EJB specification. An EJB container can then be used to deploy and control these components.

Here are some of the advantages of the EJB component model:

- Scalability: EJB components are easily scalable to accommodate large numbers of concurrent requests. As necessary to fulfill the needs of the application, the EJB container can generate and delete instances to manage the lifecycle of the components.
- Security: At the component level, EJB components can be configured to implement security policies like authentication and authorization. As a result, creating secure applications that safeguard sensitive data is made simpler.
- Reusability: EJB components may be created so that they can be used in different applications. This may result in shorter development cycles and less expensive maintenance.
- Persistence: Java Persistence API can be utilized by EJB components to communicate with databases and other data sources (JPA). This makes it possible for programmers to create apps with effective data storage and retrieval capabilities.

Here is an illustration of a basic EJB component:

```
@Stateless
public class ExampleBean implements ExampleRemote {
    public String hello() {
        return "Hello, world!";
    }
}

@Remote
public interface ExampleRemote {
    public String hello();
}
```

In this illustration, we've developed an EJB component called ExampleBean that implements ExampleRemote, a remote interface. The EJB container should manage this component as a stateless session bean, according to the @Stateless annotation. The component's business logic is implemented in the hello() method, which only outputs a string message.

The @Remote annotation instructs the container that this interface should be made available to clients executing on separate machines as a remote interface. The hello() method on the ExampleBean component can be called by clients using the ExampleRemote interface.

In general, the EJB component paradigm offers a strong and adaptable means of creating enterprise applications. Developers can create scalable, secure, and reusable components that are simple to incorporate into intricate systems by utilizing the advantages offered by the EJB container.

4. Task 3 – Critically Analysis

4.1. The difference between dependency injection and initial context lookup

Both of these two techniques are methods for giving objects the resources they require to work properly. Their methods and the resources they offer, however, are different.

Through the use of the programming design pattern known as dependency injection, a component's dependencies can be defined outside as opposed to being hardcoded inside the component. At runtime, the dependencies are then made available to the component, generally by constructor injection, method injection, or property injection. It is simpler to test and manage components when there is loose connectivity between them thanks to dependency injection.

In contrast, Java Enterprise Edition (Java EE) offers the Initial Context Lookup capability to let components access resources including database connections, JMS (Java Messaging Service) queues, and EJB (Enterprise Java Beans) components. “To

locate the resources and get a reference to them, the Java Naming and Directory Interface is used (JNDI).” The resources are often bound to a logical name and configured in configuration files like web.xml or ejb-jar.xml.

In conclusion, Initial Context Lookup is a feature offered by Java EE that enables components to access resources by looking them up in a naming and directory service. Dependency Injection is a programming design pattern that encourages loose coupling between components by externalizing their dependencies.

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

“JMS (Java Message Service) API is a key tool for developing enterprise-level systems that require dependable and asynchronous communication between numerous software components.” Some reasons why JMS API is important for EE applications:

- Reliable messaging: JMS API offers reliable messaging by assuring that messages are delivered to the intended recipients and that none are lost. For enterprise systems that are mission-critical and demand reliable message delivery, this is crucial.
- Asynchronous messaging: JMS API supports asynchronous messaging, enabling programs to send and receive messages independently of one another. Because messages can be handled concurrently without interfering with one another, this enables more effective and flexible communication between software components.
- Platform independence: As JMS API is platform-independent, applications created with it can be used with any JMS-compliant messaging system. For enterprise-level applications, this offers greater flexibility and portability.
- Scalability: The JMS API is made to be scalable, enabling applications to handle the high message and user volumes without degrading performance or dependability.
- Standards-based: A broad variety of messaging systems and suppliers support the JMS API because it is a standards-based API. Enterprise-level apps benefit from increased interoperability and flexibility as a result.

Enterprise-level applications that require dependable and asynchronous interactions must use JMS API. It is a crucial technology for creating intricate and mission-critical systems due to its support for platform independence, scalability, and standards-based messaging.

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

The asynchronously receiving and processing EJB utilized in Java EE applications are referred to as MDB. For MOM applications, where various applications exchange messages, MDBs are employed.

Some typical MDB use cases:

- Processing messages asynchronously: MDBs can be used to receive and process messages asynchronously, which can enhance the application's performance and scalability. This is especially helpful when processing a message can take a while and the sender does not want a quick answer.
- Integration with messaging platforms: Java EE applications can be integrated with messaging platforms like JMS (Java Messaging Service) or IBM MQ using MDBs. This makes it possible for the apps to communicate with other applications that adhere to the same messaging standards.
- Event-driven architectures: In event-driven architectures, MDBs can be utilized to react to events produced by other system components. In an Internet of Things (IoT) system, for instance, an MDB could be used to receive and handle events produced by sensors or other devices.
- Asynchronous communication between distributed components: In a Java EE application, MDBs can be utilized to enable asynchronous communication between distributed components. This can lessen coupling between components and enhance the application's performance and dependability.

Overall, MDBs offer a potent method for creating resilient, distributed, and scalable applications that can deal with high message volumes in an asynchronous and non-blocking way.

5. Task 4 – Critically Describe

“The best session bean type for building a web application that displays the number of times a web page has been accessed is a singleton session bean.”

A single instance of a bean for the duration of the entire application is maintained by a singleton session bean, which can be used to track application-wide state data like the total number of times a web page has been accessed. This is due to the fact that it is not connected to a single client's session and can be viewed by several clients at once.

“A Stateful Session Bean, in contrast, is made to keep track of the state data for a particular client's session.” It can be used to maintain track of details particular to a client, like a login information or preferences. As each client's session is unique and individual, it might not be appropriate for managing application-wide state information.

“A Stateless Session Bean is intended to offer a service to a client without maintaining any state data in between requests.” It can be used for tasks like straightforward calculations or database queries that don't need session-specific data. Therefore, it might not be appropriate for a recording session- or application-specific status data.

In conclusion, a Singleton Session Bean is the best sort of Session Bean to employ when building a web application that counts how many times all users have accessed a particular web page. While a Stateful Session Bean excels at preserving session-specific state data, the Stateless Session Bean is suited for stateless activities.

6. Conclusion

The J2EE platform's EJB module is a strong component that offers a solid framework for creating and delivering scalable, distributed, and transactional enterprise applications. EJB makes application development easier, increases application security, and boosts performance by supporting a variety of programming paradigms and standards. The J2EE platform's other elements, such as JSP, Servlets, JMS, and JTA, work together to form a comprehensive platform for creating and deploying enterprise-level applications.

7. References:

Tyson, M. (2019). *What is EJB? The evolution of Enterprise JavaBeans*. [online] InfoWorld. Available at: https://www.infoworld.com/article/3432125/what-is-ejb-the-evolution-of-enterprise-javabeans.html#:~:text=The%20EJB%20architecture%20consists%20of [Accessed 5 Mar. 2023].

docs.oracle.com. (n.d.). *Understanding Enterprise JavaBeans*. [online] Available at: https://docs.oracle.com/cd/E11035_01/wls100/ejb/understanding.html.

www.javatpoint.com. (n.d.). *EJB Architecture - javatpoint*. [online] Available at: https://www.javatpoint.com/ejb-architecture-java.

www.tutorialspoint.com. (n.d.). *EJB - Overview - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/ejb/ejb_overview.htm.

GeeksforGeeks. (2019). *Enterprise Java Beans (EJB)*. [online] Available at: https://www.geeksforgeeks.org/enterprise-java-beans-ejb/.

docs.oracle.com. (n.d.). *J2EE Platform Overview (Sun Java System Application Server 9.1 Deployment Planning Guide)*. [online] Available at: https://docs.oracle.com/cd/E19159-01/819-3680/abfar/index.html.

www.javatpoint.com. (n.d.). *Java EE | Java Enterprise Edition - Javatpoint*. [online] Available at: https://www.javatpoint.com/java-ee.

www.informit.com. (n.d.). *J2EE Platform | J2EE Platform Overview | InformIT*. [online] Available at: https://www.informit.com/articles/article.aspx?p=23573&seqNum=3 [Accessed 5 Mar. 2023].

flylib.com. (n.d.). *Enterprise Application Model | Microsoft Corporation - Analyzing Requirements and Defining Solutions Architecture. MCSD Training Kit*. [online] Available at: https://flylib.com/books/en/2.894.1.19/1/ [Accessed 5 Mar. 2023].

Chakray. (2018). *7 benefits of Enterprise Application Integration (EAI)*. [online] Available at: https://www.chakray.com/7-benefits-of-enterprise-application-integration-eai/.

www.oracle.com. (n.d.). *Java 2 Platform, Enterprise Edition (J2EE) Overview*. [online] Available at: https://www.oracle.com/java/technologies/appmodel.html.

docs.oracle.com. (n.d.). *J2EE Containers*. [online] Available at: https://docs.oracle.com/cd/E17802_01/j2ee/j2ee/1.4/docs/tutorial-update6/doc/Overview3.html [Accessed 5 Mar. 2023].

Edureka. (2019). *What is EJB? Enterprise Java Beans Tutorial*. [online] Available at: https://www.edureka.co/blog/ejb-in-java/#:~:text=Advantages%20of%20EJB [Accessed 5 Mar. 2023].

docs.jboss.org. (n.d.). *Dependency injection and programmatic lookup*. [online] Available at: https://docs.jboss.org/weld/reference/latest/en-US/html/injection.html [Accessed 5 Mar. 2023].

docs.oracle.com. (n.d.). *Overview of the JMS API - The Java EE 6 Tutorial*. [online] Available at: https://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html.

javaee.github.io. (n.d.). *Overview of the JMS API*. [online] Available at: https://javaee.github.io/tutorial/jms-concepts001.html [Accessed 5 Mar. 2023].

www.ibm.com. (n.d.). *Developing message-driven beans*. [online] Available at: https://www.ibm.com/docs/en/was-nd/9.0.5?topic=beans-developing-message-driven [Accessed 5 Mar. 2023].

docs.oracle.com. (n.d.). *What Is a Message-Driven Bean? - The Java EE 6 Tutorial*. [online] Available at: https://docs.oracle.com/javaee/6/tutorial/doc/gipko.html.

www.javatpoint.com. (n.d.). *Message Driven Bean - javatpoint*. [online] Available at: https://www.javatpoint.com/message-driven-bean [Accessed 5 Mar. 2023].

www.tutorialspoint.com. (n.d.). *EJB - Message Driven Beans*. [online] Available at: https://www.tutorialspoint.com/ejb/ejb_message_driven_beans.htm [Accessed 5 Mar. 2023].

docs.jboss.org. (n.d.). *Chapter 1. Getting started with Web Beans*. [online] Available at: https://docs.jboss.org/webbeans/reference/current/en-US/html/intro.html [Accessed 5 Mar. 2023].

www.javatpoint.com. (n.d.). *Types of EJB - javatpoint*. [online] Available at: https://www.javatpoint.com/types-of-ejb.

www.javatpoint.com. (n.d.). *Session Bean - javatpoint*. [online] Available at: https://www.javatpoint.com/session-bean [Accessed 5 Mar. 2023].

www.tutorialspoint.com. (n.d.). *EJB - Stateless Bean*. [online] Available at: https://www.tutorialspoint.com/ejb/ejb_stateless_beans.htm.