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 report is about the EJB (Enterprise JavaBeans) & the J2EE (Java 2 Platform, Enterprise Edition) technologies and describes the EJB concepts, principles, terminology, and components. It gives the EJB application solution for the given situation using the J2EE platform and explains it.

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:

“The Smart Energy Management System is going to implement an IoT device to capture and transmit its sensors' data (voltage, frequency, and current) to the messaging server which tries to process data with an analytical server, where their requirement is to monitor the (total power consumption, average current, average voltage, and average frequency) from that data.”

This report is written for given an EJB application solution for the above situation using the J2EE platform and explains it. Also, here the diagrams are given and the efficiency and correctness of the solution are analyzed.

3. Task 1 – EJB Application

3.1. Code Solution

3.1.1. IOTAnalyticsServerBean

IOTAnalyticsBean.java

```java
package com.ntr.iot;

import static com.ntr.iot.IOTMessageBean.iotMessageData;
import com.ntr.iot.device.IOTMessageData;
import javax.ejb.Stateless;

@Stateless
public class IOTAnalyticsBean implements IOTAnalyticsBeanRemote {

    @Override
    public String analyze() {
        double averageVoltage = 0.0;
        double averageFrequency = 0.0;
        double averageCurrent = 0.0;

        double totalVoltage = 0.0;
        double totalFrequency = 0.0;
        double totalCurrent = 0.0;

        for (IOTMessageData message : iotMessageData) {
            totalVoltage = totalVoltage + message.getVoltage();
            totalFrequency = totalFrequency + message.getFrequency();
            totalCurrent = totalCurrent + message.getCurrent();
        }

        if (!iotMessageData.isEmpty()) {
            averageVoltage = totalVoltage / iotMessageData.size();
            averageFrequency = totalFrequency / iotMessageData.size();
            averageCurrent = totalCurrent / iotMessageData.size();
        }

        return "Average Voltage = "+averageVoltage+" V\n"
                + "Average Frequency = "+averageFrequency+" Hz\n"
                + "Average Current = "+averageCurrent+" A";
    }
}
```

This is a Java class called IOTAnalyticsBean that contains a method called analyze() that calculates and returns the average voltage, frequency, and current from a collection

of IOTMessageData objects. The analyze() method overrides the method in an IOTAnalyticsBeanRemote interface. The method calculates the average voltage, frequency, and current by iterating over the iotMessageData collection of IOTMessageData objects and summing up the values of these properties. The method checks if the iotMessageData collection is empty before calculating the averages to avoid division by zero errors. The method returns a string that contains the calculated averages, formatted as human-readable text.

IOTMessageBean.java

```
package com.ntr.iot;

import com.ntr.iot.device.IOTMessageData;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationLookup",
propertyValue = "iotQueue"),
    @ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Queue")
})
public class IOTMessageBean implements MessageListener {

    static final List<IOTMessageData> iotMessageData = new
ArrayList<>();

    @Override
    public void onMessage(Message message) {
        try {
            IOTMessageData iOTMessageData =
message.getBody(IOTMessageData.class);
            System.out.println("Message Received: " +
iOTMessageData.getVoltage());
            iotMessageData.add(iOTMessageData);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

This code is a Java class that implements an MDB in a JMS application. This class will be automatically created and managed by the EJB container to consume messages from a JMS queue because of the "@MessageDriven" annotation. The "@ActivationConfig" annotation is used to specify the configuration properties for this MDB. It specifies the name of the JMS destination (iotQueue) and the type of destination "(javax.jms.Queue)". The class implements the MessageListener interface, which means that it has a single method: "onMessage()". This method is called by the JMS provider when a message is received on the iotQueue destination. Inside the "onMessage()" method, the code attempts to extract an object of type IOTMessageData from the incoming JMS message using the "getBody()" method. If successful, it prints the voltage value of the message to the console and adds the IOTMessageData object to a static list called iotMessageData. If an exception occurs while processing the message, the catch block logs the exception to the console.

3.1.2. IOTAnalyticsServerLibrary

IOTAnalyticsBeanRemote.java

```
package com.ntr.iot;

import javax.ejb.Remote;

@Remote
public interface IOTAnalyticsBeanRemote {
    String analyze();
}
```

This code is a remote interface for an IoT analytics bean that provides a method for analyzing data and returning a result as a String. This code defines a Java interface called IOTAnalyticsBeanRemote. It has a analyze() method that returns a String. The "@Remote" annotation indicates that this interface is intended to be used by remote clients, meaning it can be accessed from a different JVM.

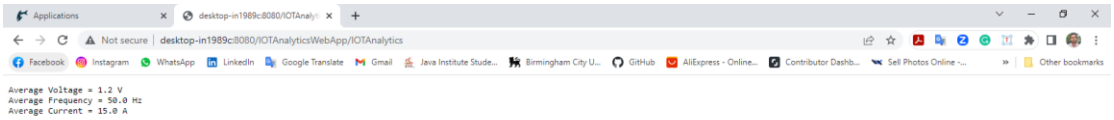
3.1.3. IOTAnalyticsWebApp

IOTAnalytics.java

```
package com.ntr.iot.webapp;

import java.io.IOException;
import javax.ejb.EJB;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ntr.iot.IOTAnalyticsBeanRemote;

@WebServlet(name = "IOTAnalytics", urlPatterns = {"/IOTAnalytics"})
public class IOTAnalytics extends HttpServlet {

    @EJB
    IOTAnalyticsBeanRemote IOTAnalyticsBeanRemote;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        resp.getWriter().write(IOTAnalyticsBeanRemote.analyze());
    }
}
```

This is a Java Servlet class named IOTAnalytics that extends the HttpServlet class. It receives GET requests and responds with the result of calling a method from a remote EJB named "IOTAnalyticsBeanRemote". The "@WebServlet" annotation specifies the URL pattern that maps to this servlet. The "@EJB" annotation injects a reference to the remote EJB named IOTAnalyticsBeanRemote into the class variable of the same name. The "doGet()" method is overriden to handle incoming GET requests, which simply calls the analyze() method from the injected IOTAnalyticsBeanRemote EJB, and writes the result to the response output stream using the "getWriter()" method.



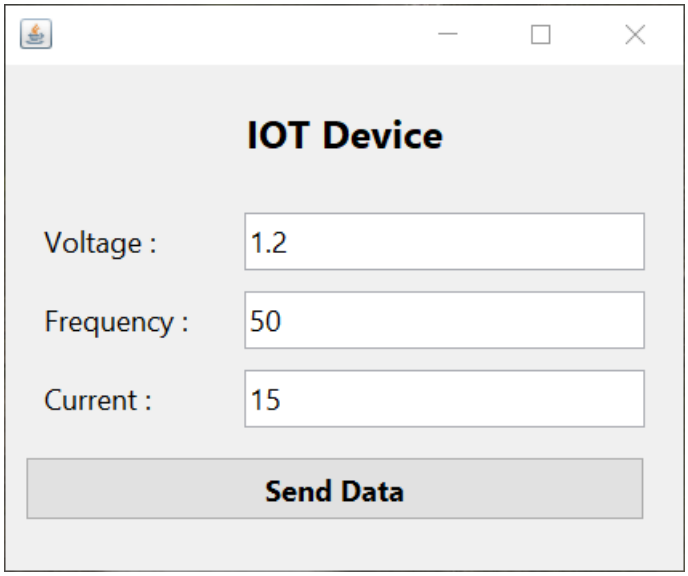
3.1.4. IOTDevice

IOTDevice.java

```
package com.ntr.iot.device;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jms.JMSException;
import javax.naming.NamingException;

public class IOTDevice extends javax.swing.JFrame {

    public IOTDevice() {
        initComponents();
    }

    private void
sendDataButtonActionPerformed(java.awt.event.ActionEvent evt) {

        String voltage = voltageTextField.getText();
        String frequency = frequencyTextField.getText();
        String current = currentTextField.getText();

        IOTMessageData iotMessageData = new IOTMessageData();
        iotMessageData.setVoltage(Double.parseDouble(voltage));
        iotMessageData.setFrequency(Double.parseDouble(frequency));
        iotMessageData.setCurrent(Double.parseDouble(current));

        try {
            IOTJMSSender jsmSender = new
IOTJMSSender("iotQueueConnectionFactory", "iotQueue");
            jsmSender.sendMessage(iotMessageData);
```

```
        } catch (NamingException | JMSException e) {
            Logger.getLogger(IOTDevice.class.getName()).log(Level.SEVER
E, null, e);
        }
    }

    public static void main(String args[]) {…}
}
```

This is a Java program that defines a graphical user interface (GUI) for an IoT device. The code defines a class named "IOTDevice" that extends the "javax.swing.JFrame" class, which is a top-level container that represents the window of the application. When the sendDataButtonActionPerformed button is clicked, it retrieves the values from three text fields for voltage, frequency, and current. It then creates an instance of the IOTMessageData class, sets the values from the text fields on the IOTMessageData object, and sends the IOTMessageData object to a JMS queue using an IOTJMSSender object. The try-catch block is used to catch any potential exceptions that might be thrown during the JMS message-sending process. The Logger class is used to log any exceptions that are caught.



IOTJMSSender.java

```
package com.ntr.iot.device;

import javax.jms.JMSException;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
```

```
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class IOTJMSSender {
    private final QueueConnectionFactory queueConnectionFactory;
    private final Queue queue;

    public IOTJMSSender(String connectionFactory, String destination)
throws NamingException {
        Context context = new InitialContext();
        queueConnectionFactory = (QueueConnectionFactory)
context.lookup(connectionFactory);
        queue = (Queue) context.lookup(destination);
    }

    public void sendMessage(IOTMessageData obj) throws JMSException {
        QueueConnection queueConnection = null;
        QueueSession queueSession = null;
        QueueSender queueSender = null;
        try {
            queueConnection =
queueConnectionFactory.createQueueConnection();
            queueSession = queueConnection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
            queueSender = queueSession.createSender(queue);
            ObjectMessage message = queueSession.createObjectMessage();
            message.setObject(obj);
            queueSender.send(message);
        } finally {
            if (queueSender != null) {
                queueSender.close();
            }
            if (queueSession != null) {
                queueSession.close();
            }
            if (queueConnection != null) {
                queueConnection.close();
            }
        }
    }
}
```

This Java code defines a class called IOTJMSSender that is responsible for sending JMS messages containing IOTMessageData objects to a JMS queue. The class has a constructor that takes two arguments, connectionFactory and destination, which are the JNDI names of the queue connection factory and queue destination, respectively. The sendMessage method takes an IOTMessageData object and sends it to the JMS queue specified by the destination argument. The method creates a queue connection, a queue session, and a queue sender using the queue connection factory and destination obtained from the JNDI context. It then creates an object message and sets its payload to the IOTMessageData object. Finally, the message is sent to the JMS queue. The sendMessage method uses a try-finally block to ensure that the JMS resources (queue sender, queue session, and queue connection) are properly closed, regardless of whether or not an exception is thrown during message sending.

3.1.5. IOTDeviceLibrary

IOTMessageData.java

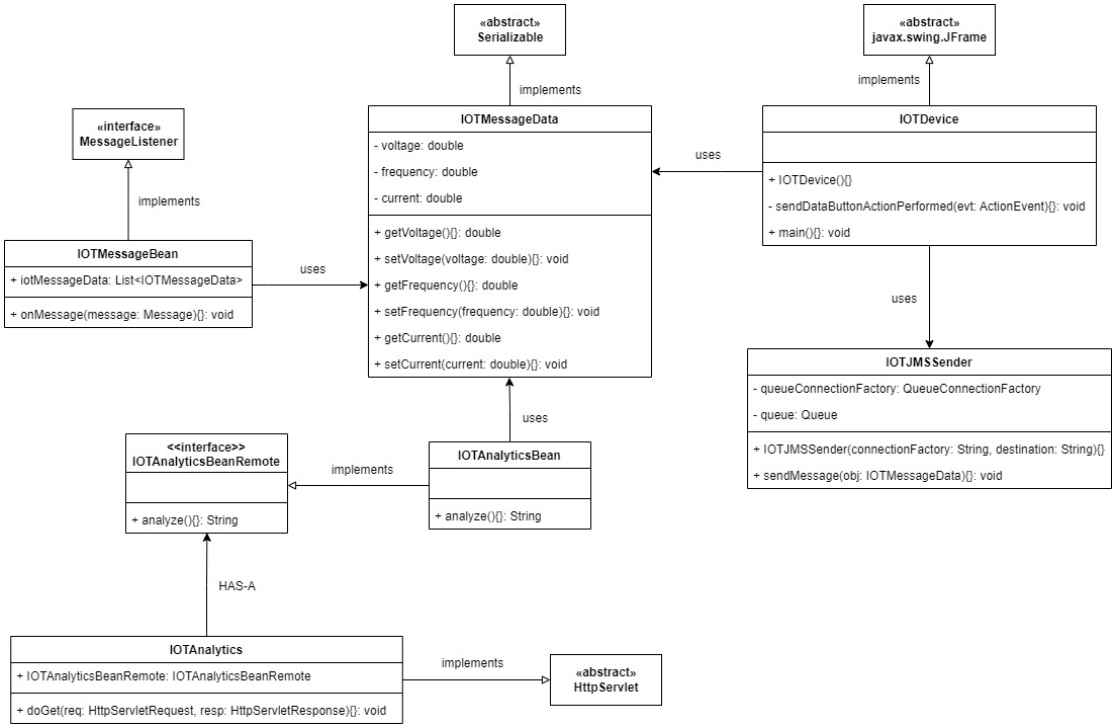
```
package com.ntr.iot.device;

import java.io.Serializable;

public class IOTMessageData implements Serializable{
    private double voltage;
    private double frequency;
    private double current;

    public double getVoltage() {
        return voltage;
    }

    public void setVoltage(double voltage) {
        this.voltage = voltage;
    }

    public double getFrequency() {
        return frequency;
    }

    public void setFrequency(double frequency) {
        this.frequency = frequency;
    }

    public double getCurrent() {
        return current;
```

```
    }

    public void setCurrent(double current) {
        this.current = current;
    }
}
```

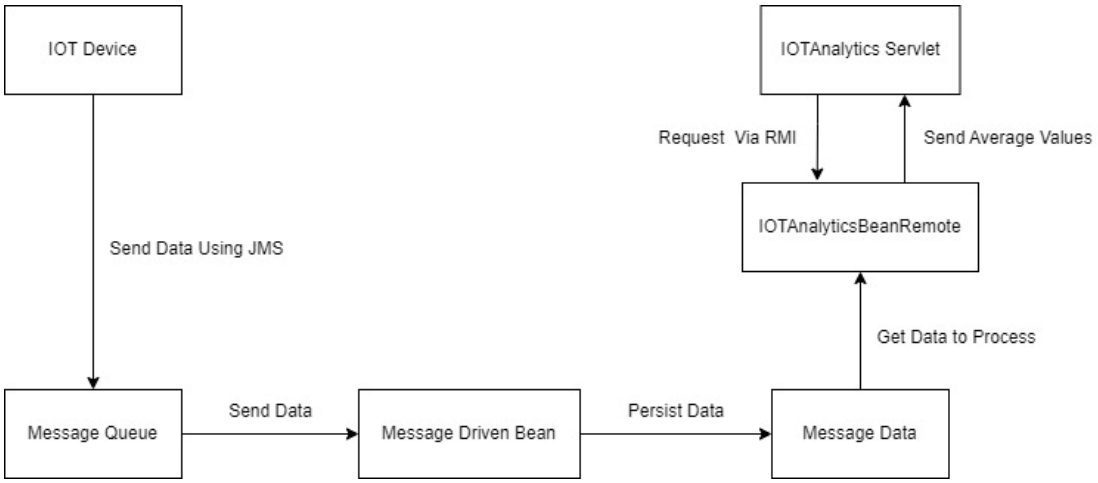
This is a Java class named IOTMessageData that implements the Serializable interface. It has three private instance variables: voltage, frequency, and current, all of which are of type double. Each instance variable in the class contains public getter and setter methods. This class is used as a data object to hold information about an IoT device's voltage, frequency, and current measurements. The Serializable interface is implemented so that instances of this class can be serialized and sent over a network or stored in a file.

3.2. Diagrams

3.2.1. Class Diagram



3.2.2. Flow Diagram



3.3. Efficiency and Correctness Analysis

The solution created using Java EE for processing IoT device data provides an efficient and proper technique to manage data collecting and processing.

The usage of JMS API and Message-Driven Beans provides an efficient means of handling data collection by decoupling the message producers from the consumers. The implementation of Queue and QueueConnectionFactory instead of Topic and TopicConnectionFactory is appropriate as it ensures that data is not lost when the destination is down. The usage of a separate EJB module for saving and processing the messages enhances the application's scalability as it simplifies the development of future enhancements.

The use of a static variable for storing the ArrayList enhances the efficiency of the solution as it provides a means to store and retrieve data without requiring the overhead of creating and destroying objects.

The EJB module's implementation of the analyze() method offers a quick and accurate way to compute and return the average values of the gathered data. An effective method of processing the data is to use a for-loop to iterate through the ArrayList and calculate the total values of the data. The technique delivers accurate results even when no data has been collected because a conditional statement is used to verify if the ArrayList is empty.

By keeping the client web app from seeing the source code, the implementation of the RMI interface to use the analyze() method in the EJB module improves the solution's correctness.

Overall, the solution created using Java EE shows how to manage data gathering and processing in an effective and proper manner, improving the application's scalability and maintainability.

4. 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.

5. 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.