JAVA INSTITUTE for Advanced Technology

INNOVATIONS FOR THE KNOWLEDGE SOCIETY

ISO 9001:2008 Certified Institute

BUSINESS COMPONENT DEVELOPMENT - II
(JIAT/BCD II)

Student First Name	**: M.R.P.N.Tharuksha**
Student Last Name	**: Rajapaksha**
BCU No.	**: 22178965**
NIC No.	**: 200019401866**
Branch	**: Colombo**

BIRMINGHAM CITY University

SQA Approved Centre Scottish Qualifications Authority

scqf scottish credit and qualifications framework

PCJT The Software Engineer

Acknowledgment

First and foremost, I would like to publicly thank Mr. Achintha Chamara, my lecturer, and adviser for the business component development II 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 designing, developing and deploying an enterprise Java application for a chosen business scenario. This application has included Time services, Interceptor classes, Different types of transactions, Robust security architecture, Exception handling, EJB components, and Testing tools, and comprehensively describes all of them in this report.

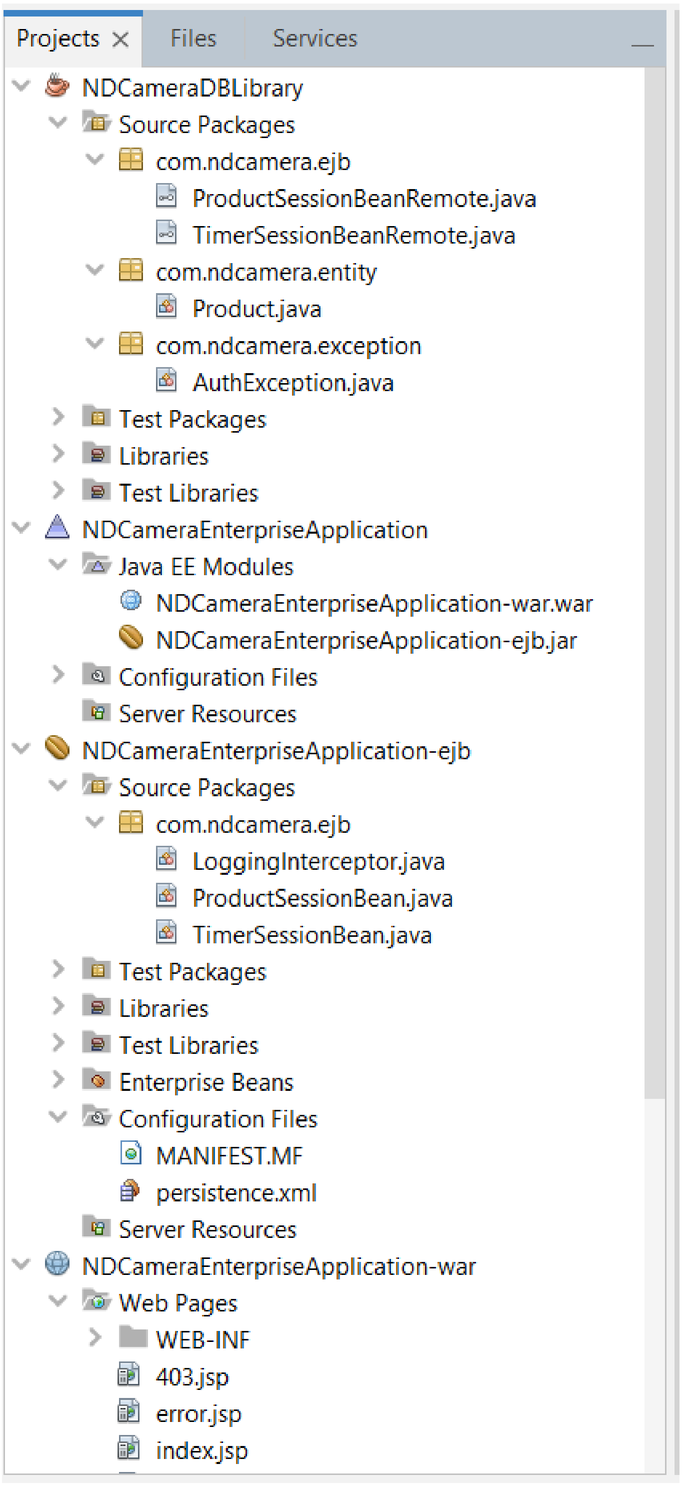
Content

1. Introduction

This report is written for describing the designing, developing and deploying of an enterprise Java application for a chosen business scenario (NDCamera Project Manegement Part). The application uses best practices and standards, and it is user-friendly, efficient, and secure. This application has included Time services to manage task scheduling, Interceptor classes to log user activity, Different types of transactions to ensure data consistency and reliability, Robust security architecture to protect user data and ensure privacy, Exception handling to optimize application performance, EJB components in a split directory structure, and Testing tools to test the application works. Furthermore, this report comprehensively describes all of the above-included things.

2. Comprehensive

2.1. Design



```
Projects × | Files | Services
NDCameraDBLibrary
    Source Packages
        com.ndcamera.ejb
            ProductSessionBeanRemote.java
            TimerSessionBeanRemote.java
        com.ndcamera.entity
            Product.java
        com.ndcamera.exception
            AuthException.java
    Test Packages
    Libraries
    Test Libraries
NDCameraEnterpriseApplication
    Java EE Modules
        NDCameraEnterpriseApplication-war.war
        NDCameraEnterpriseApplication-ejb.jar
    Configuration Files
    Server Resources
NDCameraEnterpriseApplication-ejb
    Source Packages
        com.ndcamera.ejb
            LoggingInterceptor.java
            ProductSessionBean.java
            TimerSessionBean.java
    Test Packages
    Libraries
    Test Libraries
    Enterprise Beans
    Configuration Files
        MANIFEST.MF
        persistence.xml
    Server Resources
NDCameraEnterpriseApplication-war
    Web Pages
        WEB-INF
        403.jsp
        error.jsp
        index.jsp
```

The project consists of two components: NDCameraDBLibrary and NDCameraEnterpriseApplication. NDCameraDBLibrary is a Java Class Library, while NDCameraEnterpriseApplication is an Enterprise Application. NDCameraEnterpriseApplication is further divided into NDCameraEnterpriseApplication-ejb, which contains EJB (Enterprise JavaBeans) modules, and NDCameraEnterpriseApplication-war, which contains the web application.

NDCameraDBLibrary includes the following components:

- ProductSessionBeanRemote and TimerSessionBeanRemote: These are Session Bean Remotes that provide remote access to the ProductSessionBean and TimerSessionBean components.
- Product entity class: This class represents the Product entity in the project's database.
- AuthException Exception class: This is an exception class that handles run time-related exceptions.

NDCameraEnterpriseApplication-ejb includes the following components:

- LoggingInterceptor: This is an Interceptor class that handles logging for the project.
- ProductSessionBean and TimerSessionBean: These are Session Beans that provide business logic for the project.
- persistence.java file: This file contains configuration settings for the Java Persistence API (JPA) and is located in the NDCameraEnterpriseApplication-ejb module.

NDCameraEnterpriseApplication-war includes the following components:

- 403.jsp, error.jsp, index.jsp, login.jsp: These are JSP files that define the user interface for the web application.
- DeleteProduct, InsertProduct, Logout, SearchProduct, UpdateProduct Servlets: These are Servlets that handle various requests from the user interface.

- web.xml file: This file contains configuration settings for security types in the web application.

2.2. Development

2.2.1. NDCameraDBLibrary

2.2.1.1. ProductSessionBeanRemote.java

```
package com.ndcamera.ejb;

import com.ndcamera.entity.Product;
import com.ndcamera.exception.AuthException;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface ProductSessionBeanRemote {

    void insertProduct(Product product) throws AuthException;

    void updateProduct(Product product) throws AuthException;

    void deleteProduct(int id) throws AuthException;

    List<Product> getAllProducts() throws AuthException;

    Product findById(int id) throws AuthException;

    Product findByProductName(String productName) throws AuthException;

    List<Product> findByBrand(String brand) throws AuthException;

    List<Product> findByType(String type) throws AuthException;

    List<Product> findBySensorType(String sensorType) throws
AuthException;
}
```

In a Java Enterprise Edition (Java EE) application, the code specifies a remote interface called "ProductSessionBeanRemote" that stands in for a session bean. Clients in a distributed system are supposed to access the session bean remotely. Several operations related to managing products are provided through the interface's various methods. These methods consist of:

- Inserts a new product into the system with the help of insertProduct(Product product). It takes a "Product" object as a parameter.
- Changes a current product in the system with updateProduct(Product product). It takes a "Product" object as a parameter.
- Using the integer ID, deleteProduct(int id) removes a product from the database. It takes an integer ID as a parameter.
- Gets a list of every product in the system using the getAllProducts() function. It returns a list of "Product" objects.
- findById(int id) method using the ID of a product, retrieves it from the database. It accepts an integer ID as a parameter, returns a "Product" object.
- Finds a product from the system using its name using findByProductName(String productName). It accepts the string "productName" as a parameter, returns a "Product" object.
- findByBrand(String brand) pulls a list of items from the database according to their brands. It accepts the string "brand" as a parameter, returns a list of "Product" objects.
- findByType(String type) retrieves a list of items from the system according to the kind of those products. It accepts a string parameter named "type," returns a list of "Product" objects.
- findBySensorType(String sensorType) pulls a list of items from the system according to the sensor type that each one has. It receives a string parameter named "sensorType," returns a list of "Product" objects.

If run time related fails, the methods in the interface may throw a "AuthException,".

2.2.1.2. TimerSessionBeanRemote.java

```java
package com.ndcamera.ejb;

import javax.ejb.Remote;
import javax.ejb.Timer;

```

```java
@Remote
public interface TimerSessionBeanRemote {

    void scheduler(Timer Timer);

    void createTimer(long duration);
}
```

The aforementioned code defines a remote enterprise JavaBean (EJB) interface called "TimerSessionBeanRemote" that clients in a distributed Java application can access remotely. The interface offers two methods:

- Schedule actions or events to take place at predetermined intervals or periods using the scheduler(Timer Timer) method, which accepts a Timer object as an argument. A timer service that may be used to create and manage timers in an EJB container is represented by the Timer object.
- Create a timer that will expire after the specified amount of time by using the createTimer method with a long value reflecting the duration in milliseconds. The timer can be used to plan actions or occasions to happen when a predetermined period of time has passed.

This interface is intended to be accessed remotely by clients, as indicated by the "@Remote" annotation, enabling them to call the methods specified in this interface on a remote EJB instance. A session bean in an EJB application is supposed to implement this interface so that the timer-related functionality can actually be implemented.

2.2.1.3. Product.java

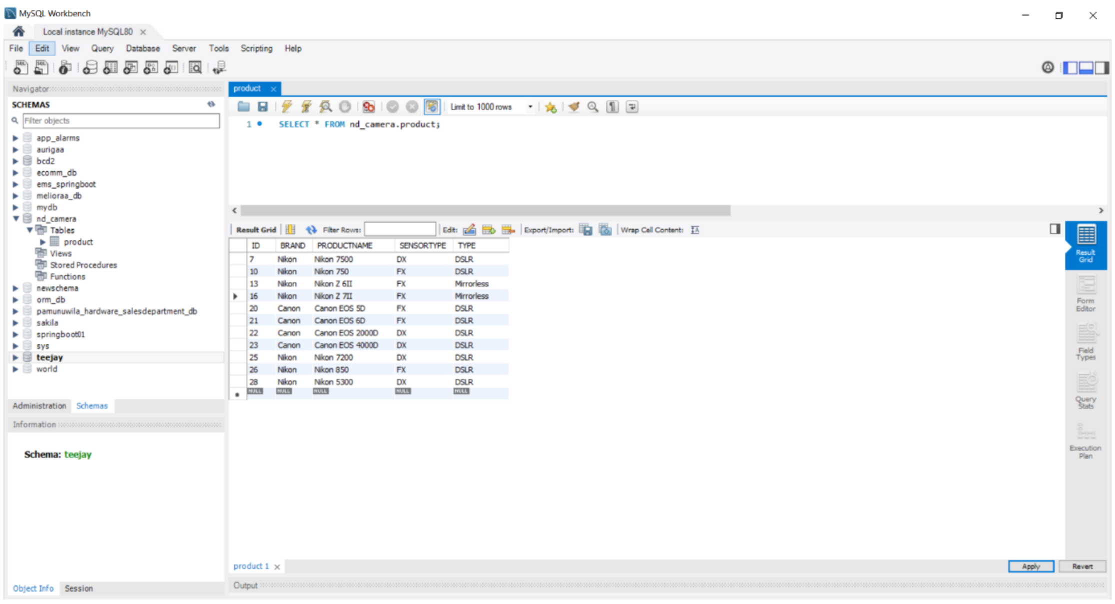
```java
package com.ndcamera.entity;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Product implements Serializable {
```

```java
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
private String productName;
private String brand;
private String type;
private String sensorType;

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getProductName() {
    return productName;
}

public void setProductName(String productName) {
    this.productName = productName;
}

public String getBrand() {
    return brand;
}

public void setBrand(String brand) {
    this.brand = brand;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public String getSensorType() {
    return sensorType;
}

public void setSensorType(String sensorType) {
        this.sensorType = sensorType;
    }
}
```

This Java code creates a class called Product in the com.ndcamera.entity package. This class represents a product entity in a fictitious camera application and is designed to be used with an object-relational mapping (ORM) implementation of the Java Persistence API (JPA).

- This class can be mapped to a database table because it is an entity, as shown by the @Entity annotation.
- The id field is designated as the entity's primary key via the @Id annotation.
- The @GeneratedValue annotation indicates that the database will use an identity approach to produce the values for the id field.
- The class has four private attributes that correspond to different aspects of a camera product: id, productName, brand, type, and sensorType.
- For each field in the class, there are matching getter and setter methods that allow you to access and change its values.
- The class implements the Serializable interface, enabling objects of this class to be serialized and deserialized in order to make them appropriate for storage in a persistent medium or network transport.

In conclusion, this code creates the Product class, which can be used with a JPA implementation to store and retrieve data from a database, and represents a camera product with a variety of attributes.



```sql
SELECT * FROM nd_camera.product;
```

ID	BRAND	PRODUCTNAME	SENSORTYPE	TYPE
7	Nikon	Nikon 7500	DX	DSLR
10	Nikon	Nikon 750	FX	DSLR
13	Nikon	Nikon Z 6II	FX	Mirrorless
16	Nikon	Nikon Z 7II	FX	Mirrorless
20	Canon	Canon EOS 5D	FX	DSLR
21	Canon	Canon EOS 6D	FX	DSLR
22	Canon	Canon EOS 2000D	DX	DSLR
23	Canon	Canon EOS 4000D	DX	DSLR
25	Nikon	Nikon 7200	DX	DSLR
26	Nikon	Nikon 850	FX	DSLR
28	Nikon	Nikon 5300	DX	DSLR
NULL	NULL	NULL	NULL	NULL

2.2.1.4. AuthException.java

```java
package com.ndcamera.exception;

import javax.ejb.ApplicationException;

@ApplicationException(rollback = true)
public class AuthException extends RuntimeException {

}
```

The code shown above creates a Java class called "AuthException" that extends the "RuntimeException" class. This exception is marked with the annotation "@ApplicationException," which denotes that it should only be used in application-level contexts, such as Java EE (Enterprise Edition) applications.

The attribute "rollback" of the "@ApplicationException" annotation likewise has the value "true." This signifies that any active transaction should be rolled back when this exception is raised. This is frequently used to ensure that any modifications made to the transaction are undone when an exception should cause a rollback of the transaction.

In conclusion, this code creates a special exception called "AuthException" that can be used in Java EE applications and is set up to cause a rollback of the current transaction when it is thrown. When handling runtime-related problems and making sure that any modifications made within a transaction are undone in the event of a run time failure, this can be helpful.

2.2.2. NDCameraEnterpriseApplication

2.2.2.1. NDCameraEnterpriseApplication-ejb

2.2.2.1.1. LoggingInterceptor.java

```java
package com.ndcamera.ejb;

import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

public class LoggingInterceptor {

    @AroundInvoke
    public Object intercept(InvocationContext invocation) throws
Exception {
        long startTime = System.currentTimeMillis();
        System.out.println("Calling method: " +
invocation.getMethod().getName());
        Object result = invocation.proceed();
        long endTime = System.currentTimeMillis();
        System.out.println("Execution time: " + (endTime - startTime) +
"ms");
        return result;
    }
}
```

This code is a Java class that uses the Java EE (Enterprise Edition) javax.interceptor package to implement an interceptor. In Java EE applications, interceptors are used to intercept and process method invocations, enabling the addition of new functionality to methods without directly changing their source code.

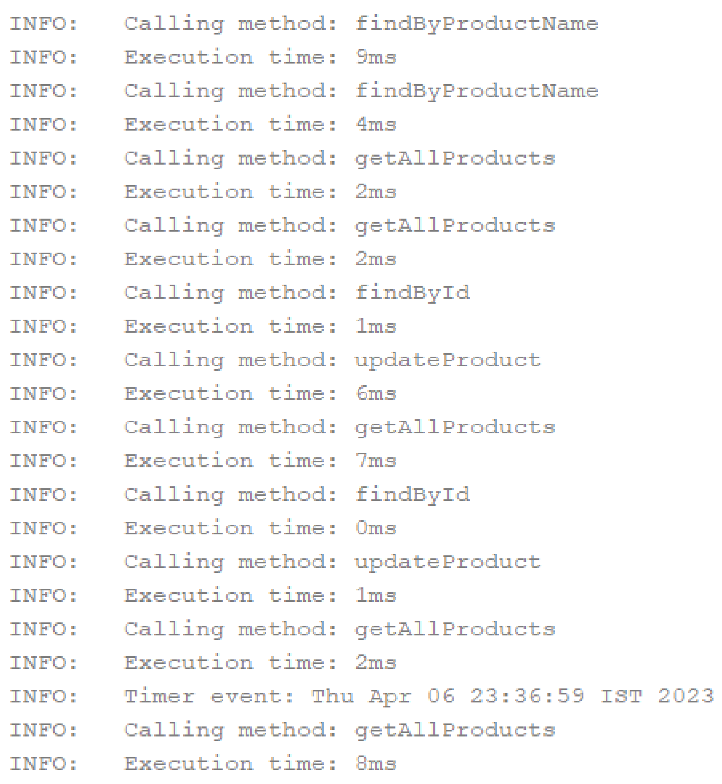
The LoggingInterceptor class in this instance has a single method intercept() that is annotated with @AroundInvoke, indicating that this method will be called around method invocations that are intercepted.

The intercept() method of the LoggingInterceptor is automatically invoked when a method is intercepted. It accepts an InvocationContext parameter that describes the method being called, its parameters, and the target object on which the method is being invoked in the context of the intercepted method invocation.

Inside the intercept() method, System.currentTimeMillis() is used to record the method invocation's start time, and invocation.getMethod().getName is used to print the name of the method being called to the console (). The intercepted method is then given permission to carry on with its operation by calling the invocation.proceed() method after that. The result variable holds the output of the intercepted method invocation.

The execution time is determined by deducting the start time from the end time after the intercepted method has finished running. The end time of the method invocation is captured using System.currentTimeMillis(). The console is then printed with the execution time. The outcome of the method call that was intercepted is then returned to the result variable.

In short, this code creates a logging interceptor that records the name of the intercepted method, the time it was executed, and permits the method to continue running while recording its outcome. With the aid of this interceptor, methods in a Java EE application can be given logging capabilities without requiring any direct code modifications.



```
INFO:   Calling method: findByProductName
INFO:   Execution time: 9ms
INFO:   Calling method: findByProductName
INFO:   Execution time: 4ms
INFO:   Calling method: getAllProducts
INFO:   Execution time: 2ms
INFO:   Calling method: getAllProducts
INFO:   Execution time: 2ms
INFO:   Calling method: findById
INFO:   Execution time: 1ms
INFO:   Calling method: updateProduct
INFO:   Execution time: 6ms
INFO:   Calling method: getAllProducts
INFO:   Execution time: 7ms
INFO:   Calling method: findById
INFO:   Execution time: 0ms
INFO:   Calling method: updateProduct
INFO:   Execution time: 1ms
INFO:   Calling method: getAllProducts
INFO:   Execution time: 2ms
INFO:   Timer event: Thu Apr 06 23:36:59 IST 2023
INFO:   Calling method: getAllProducts
INFO:   Execution time: 8ms
```

2.2.2.1.2. ProductSessionBean.java

```java
package com.ndcamera.ejb;

import com.ndcamera.entity.Product;
import com.ndcamera.exception.AuthException;
import java.util.List;
import javax.annotation.Resource;
import javax.annotation.security.PermitAll;
import javax.annotation.security.RolesAllowed;
import javax.ejb.Stateless;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;
import javax.interceptor.Interceptors;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import javax.transaction.UserTransaction;

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class ProductSessionBean implements ProductSessionBeanRemote {

    @PersistenceContext(unitName = "NDCameraEnterpriseApplication-
ejbPU")
```

```java
    private EntityManager em;

    @Resource
    private UserTransaction ut;

    public EntityManager getEm() {
        return em;
    }

    public void setEm(EntityManager em) {
        this.em = em;
    }

    public UserTransaction getUt() {
        return ut;
    }

    public void setUt(UserTransaction ut) {
        this.ut = ut;
    }

    @Interceptors(LoggingInterceptor.class)
    @RolesAllowed({"superadmin", "admin"})
    @Override
    public void insertProduct(Product product) throws AuthException {
        try {
            ut.begin();
            em.persist(product);
            ut.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Interceptors(LoggingInterceptor.class)
    @RolesAllowed({"superadmin", "admin"})
    @Override
    public void updateProduct(Product product) throws AuthException {
        try {
            ut.begin();
            em.merge(product);
            ut.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @Interceptors(LoggingInterceptor.class)
    @RolesAllowed({"superadmin"})
    @Override
    public void deleteProduct(int id) throws AuthException {
        Product product = findById(id);
        try {
            ut.begin();
            if (!em.contains(product)) {
                product = em.merge(product);
            }
            em.remove(product);
            ut.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Interceptors(LoggingInterceptor.class)
    @PermitAll
    @Override
    public List<Product> getAllProducts() throws AuthException {
        return em.createQuery("SELECT p FROM Product p",
Product.class).getResultList();
    }

    @Interceptors(LoggingInterceptor.class)
    @PermitAll
    @Override
    public Product findById(int id) throws AuthException {
        return em.find(Product.class, id);
    }

    @Interceptors(LoggingInterceptor.class)
    @PermitAll
    @Override
    public Product findByProductName(String productName) throws
AuthException {
        TypedQuery<Product> query = em.createQuery("SELECT p FROM
Product p WHERE p.productName=:productName", Product.class);
        query.setParameter("productName", productName);
        Product product = query.getSingleResult();
        return product;
    }

    @Interceptors(LoggingInterceptor.class)
    @PermitAll
    @Override
    public List<Product> findByBrand(String brand) throws AuthException
{
        TypedQuery<Product> query = em.createQuery("SELECT p FROM
Product p WHERE p.brand=:brand", Product.class);
        query.setParameter("brand", brand);
        List<Product> product = query.getResultList();
        return product;
    }

    @Interceptors(LoggingInterceptor.class)
    @PermitAll
    @Override
    public List<Product> findByType(String type) throws AuthException {
        TypedQuery<Product> query = em.createQuery("SELECT p FROM
Product p WHERE p.brand=:brand", Product.class);
        query.setParameter("type", type);
        List<Product> product = query.getResultList();
        return product;
    }

    @Interceptors(LoggingInterceptor.class)
    @PermitAll
    @Override
    public List<Product> findBySensorType(String sensorType) throws
AuthException {
        TypedQuery<Product> query = em.createQuery("SELECT p FROM
Product p WHERE p.brand=:brand", Product.class);
        query.setParameter("sensorType", sensorType);
        List<Product> product = query.getResultList();
        return product;
    }
}
```

In an enterprise Java program, this Java class represents a stateless session bean. It offers ways to manage products in a database, including ways to add, update, remove, and retrieve products. A high-level summary of the code is provided below:

- A stateless session bean is indicated by the annotation @Stateless on the class.

- The bean will manage its own transactions, as indicated by the @TransactionManagement annotation with "TransactionManagementType.BEAN".
- An instance of EntityManager that is linked to the persistence unit known as "NDCameraEnterpriseApplication-ejbPU" is injected using the @PersistenceContext annotation.
- To inject a UserTransaction instance for handling transactions, use the @Resource annotation.
- The ProductSessionBeanRemote interface, which provides the remote interface for accessing the methods offered by this session bean, is implemented by the class.
- Several methods for managing products are included in the class, including insertProduct, updateProduct, deleteProduct, getAllProducts, findById, findByProductName, findByBrand, findByType, and findBySensorType. Each of these methods is annotated with @RolesAllowed or @PermitAll to indicate the access permissions necessary for calling the method.
- The @Interceptors annotation is used to indicate that all methods in this session bean should have a logging interceptor, represented by the LoggingInterceptor class, applied for logging purposes.
- The methods carry out different tasks on the EntityManager instance, including persisting, merging, deleting, and requesting Product entities, which are used to represent products in the database.
- Try-catch blocks are used to implement exception management, catching and printing any exceptions that may arise while the methods are being executed.
- Overall, this session bean offers a collection of transactional methods for managing items. Logging is implemented via an interceptor for monitoring, and access rights are enforced depending on user roles.

2.2.2.1.3. TimerSessionBean.java

```
package com.ndcamera.ejb;

import com.ndcamera.entity.Product;
import java.util.ArrayList;
```

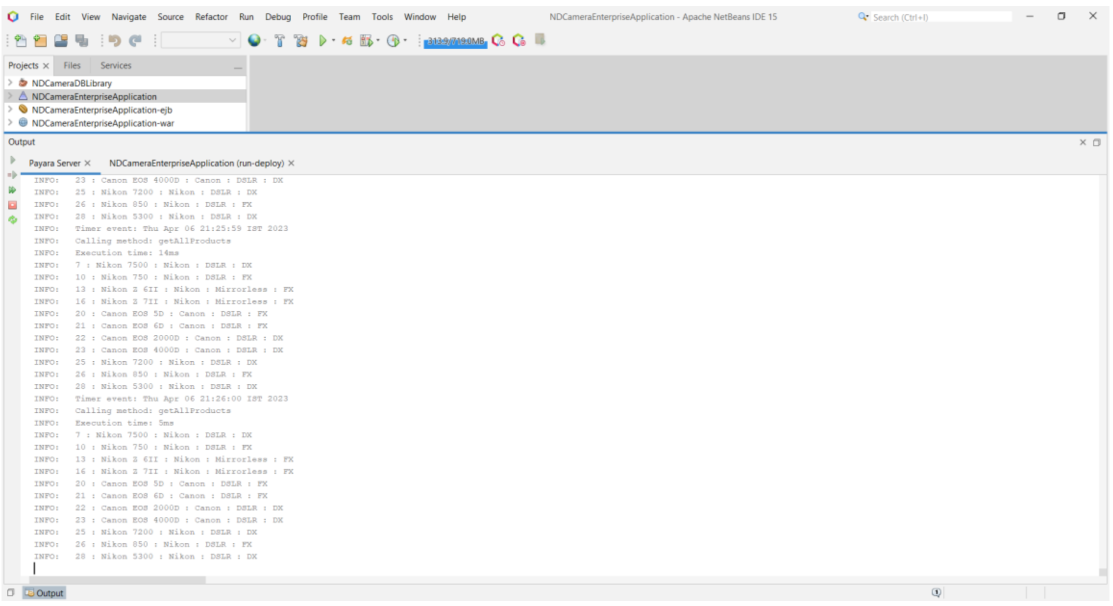
```java
import java.util.Date;
import java.util.List;
import javax.annotation.Resource;
import javax.ejb.EJB;
import javax.ejb.Schedule;
import javax.ejb.ScheduleExpression;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;
import javax.ejb.Timeout;
import javax.ejb.Timer;

@Stateless
public class TimerSessionBean implements TimerSessionBeanRemote {

    @EJB
    private ProductSessionBeanRemote psbr;

    @Resource
    private SessionContext context;

    @Schedule(dayOfWeek = "*", month = "*", hour = "*", dayOfMonth =
"*", year = "*", minute = "*", second = "*/59")
    @Override
    public void scheduler(Timer Timer) {
        System.out.println("Timer event: " + new Date());
        List<Product> products = new ArrayList<>();
        products = psbr.getAllProducts();
        for (Product product : products) {
            System.out.println(product.getId() + " : " +
product.getProductName() + " : " + product.getBrand() + " : " +
product.getType() + " : " + product.getSensorType());
        }
//          timer.cancel();
    }

    @Override
    public void createTimer(long duration) {
        ScheduleExpression se = new ScheduleExpression();
        se.hour("*");
        se.minute("*");
        se.second("*/" + duration);
        Timer timer =
context.getTimerService().createCalendarTimer(se);
        //timer.cancel();
    }
    @Timeout
    public void timeoutHandler() {
        System.out.println("Timeout...............");
    }
}
```

A stateless session bean in an enterprise Java application is represented by the Java class in the aforementioned code. Events related to timers are scheduled and handled by the session bean. Here's a quick breakdown of the code:

- A stateless session bean is indicated by the annotation @Stateless, which is included in the class.
- The TimerSessionBeanRemote interface, which establishes the remote interface for the session bean, is implemented by the class.
- The @EJB annotation is used to inject ProductSessionBeanRemote, an EJB that is a dependent of the session bean.
- The @Resource annotation is used to inject the SessionContext interface, which represents the context of the current session, as a dependency for the session bean.
- The scheduler(Timer timer) method of the session bean includes the annotation @Schedule. According to the supplied cron-like phrase, which is setup using different parameters like dayOfWeek, month, hour, etc., this procedure is set to run on a regular basis. In this scenario, the procedure should be called once every second according to the expressions dayOfWeek = "*", month = "*", hour = "*", dayOfMonth = "*", year = "*", minute = "*," and second = "*/59."
- The session bean pulls a list of Product objects from the injected ProductSessionBeanRemote EJB and prints some of its characteristics inside the scheduler() method.
- A calendar-based timer is created using the SessionContext's TimerService via the session bean's method createTimer(long duration). The method receives a duration parameter that specifies the timer's expiration date and time.

- The session bean's timeoutHandler() method includes the annotation @Timeout. When the timer generated by createTimer() expires, this function is called.

A few commented lines of code demonstrate how to use the cancel() function on the Timer object to stop a timer.


File
Edit
View
Navigate
Source
Refactor
Run
Debug
Profile
Team
Tools
Window
Help
NDCameraEnterpriseApplication - Apache NetBeans IDE 15
Search (Ctrl+I)
Projects
Files
Services
NDCameraDBLibrary
NDCameraEnterpriseApplication
NDCameraEnterpriseApplication-ejb
NDCameraEnterpriseApplication-war
Output
Payara Server
NDCameraEnterpriseApplication (run-deploy)
Output

```
INFO:   23 : Canon EOS 4000D : Canon : DSLR : DX
INFO:   25 : Nikon 7200 : Nikon : DSLR : DX
INFO:   26 : Nikon 850 : Nikon : DSLR : FX
INFO:   28 : Nikon 5300 : Nikon : DSLR : DX
INFO:   Timer event: Thu Apr 06 21:25:59 IST 2023
INFO:   Calling method: getAllProducts
INFO:   Execution time: 14ms
INFO:   7 : Nikon 7500 : Nikon : DSLR : DX
INFO:   10 : Nikon 750 : Nikon : DSLR : FX
INFO:   13 : Nikon Z 6II : Nikon : Mirrorless : FX
INFO:   16 : Nikon Z 7II : Nikon : Mirrorless : FX
INFO:   20 : Canon EOS 5D : Canon : DSLR : FX
INFO:   21 : Canon EOS 6D : Canon : DSLR : FX
INFO:   22 : Canon EOS 2000D : Canon : DSLR : DX
INFO:   23 : Canon EOS 4000D : Canon : DSLR : DX
INFO:   25 : Nikon 7200 : Nikon : DSLR : DX
INFO:   26 : Nikon 850 : Nikon : DSLR : FX
INFO:   28 : Nikon 5300 : Nikon : DSLR : DX
INFO:   Timer event: Thu Apr 06 21:26:00 IST 2023
INFO:   Calling method: getAllProducts
INFO:   Execution time: 5ms
INFO:   7 : Nikon 7500 : Nikon : DSLR : DX
INFO:   10 : Nikon 750 : Nikon : DSLR : FX
INFO:   13 : Nikon Z 6II : Nikon : Mirrorless : FX
INFO:   16 : Nikon Z 7II : Nikon : Mirrorless : FX
INFO:   20 : Canon EOS 5D : Canon : DSLR : FX
INFO:   21 : Canon EOS 6D : Canon : DSLR : FX
INFO:   22 : Canon EOS 2000D : Canon : DSLR : DX
INFO:   23 : Canon EOS 4000D : Canon : DSLR : DX
INFO:   25 : Nikon 7200 : Nikon : DSLR : DX
INFO:   26 : Nikon 850 : Nikon : DSLR : FX
INFO:   28 : Nikon 5300 : Nikon : DSLR : DX
```

2.2.2.1.4. persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="NDCameraEnterpriseApplication-ejbPU"
transaction-type="JTA">
    <jta-data-source>ndCameraDbConnection</jta-data-source>
    <class>com.ndcamera.entity.Product</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-
generation.database.action" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

An sample XML configuration file for a Java Persistence API (JPA) persistence unit may be seen in the code above. It specifies the configuration options for the "NDCameraEnterpriseApplication-ejbPU" persistence unit, a named entity inside a Java Enterprise Edition (EE) application, to manage persistence and connect to a database. The elements and attributes in the code are broken down as follows:

- persistence: This element, which is the root of the XML configuration file, specifies the XML namespace and the JPA version that is being used (2.1).
- persistent-unit: This element depicts a JPA persistence unit, which is a named collection of entity classes maintained collectively. It has two characteristics:
    - name: The persistence unit's name, in this case "NDCameraEnterpriseApplication-ejbPU," is specified here.
    - The Java Transaction API (JTA) is utilized in this instance to manage distributed transactions; the transaction-type specifies the type of transaction management to be employed.
- JNDI name of the data source to be utilized while establishing a connection to the database is specified by the jta-data-source element. The data source in this illustration is known as "ndCameraDbConnection."
- class: The fully qualified class name of an entity class that is a part of the persistence unit is specified by this element. The entity class "com.ndcamera.entity.Product" is shown in this instance.
- exclude-unlisted-classes: This element's boolean value indicates whether or not the persistence unit should only contain the classes that are listed. Only the classes explicitly listed using the class element will be included in this case because it is set to "true."
- properties: The persistence unit's additional properties can be specified using this element. In this scenario, the property "javax.persistence.schema-generation.database.action" is set to "create," meaning that when the persistence unit is deployed, the database schema will be generated automatically.

The "NDCameraEnterpriseApplication-ejbPU" persistence unit's parameters, including the data source, entity classes, and schema generation activity, are all defined in this XML configuration file.

2.2.2.2. NDCameraEnterpriseApplication-war

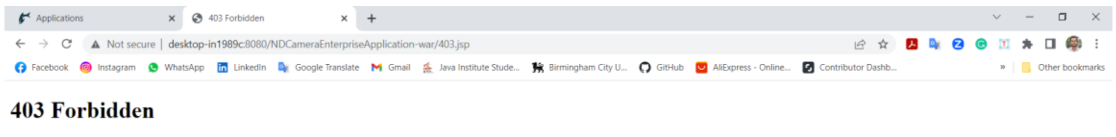
2.2.2.2.1. 403.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>403 Forbidden</title>
    </head>
    <body>
        <h1>403 Forbidden</h1>
    </body>
</html>
```

This code produces a "403 Forbidden" error page and appears to be a straightforward HTML document. Let's deconstruct it:

- The viewable content of the page is contained in the body> element.
- A header containing the words "403 Forbidden" is represented by the "h1" element inside the "body" element. This is often used to show the page's primary message or problem description.

In conclusion, this code creates a straightforward HTML page with the heading "403 Forbidden," denoting that the user does not have authorization to access the requested site.

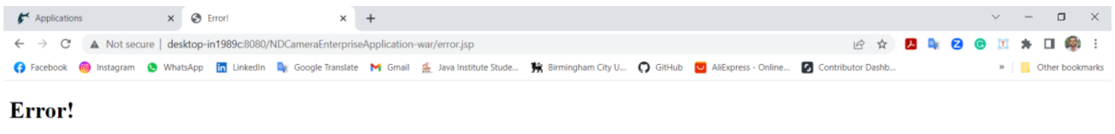


403 Forbidden

2.2.2.2.2. error.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Error!</title>
    </head>
    <body>
        <h1>Error!</h1>
    </body>
</html>
```

Java Server Pages (JSP) syntax is used in the aforementioned code to create a simple HTML page. The text "Error!" will be shown as a heading with a larger font size on an HTML page when this code is run on a web server that supports JSP. It could serve as a straightforward error page in a web application to alert users to a problem or error.


Error!

2.2.2.2.3. index.jsp

```
<%@page import="java.util.ArrayList"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page import="com.ndcamera.entity.Product"%>
<%@page import="java.util.List"%>
<%@page import="com.ndcamera.ejb.ProductSessionBeanRemote"%>
<%@page import="javax.naming.InitialContext"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>NDCamera Home</title>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"
crossorigin="anonymous">
        <style>
            a{
                text-decoration: none;
                color: white;
            }
        </style>
    </head>
    <body>
        <div class="container-fluid text-center bg-danger p-2">
            <div class="row">
```

```
                <div class="col align-self-start"></div>
                <div class="col align-self-center text-
light"><h1>NDCamera</h1></div>
                <div class="col align-self-end"><button class="btn btn-
primary"><a href="Logout">Logout</a></div>
            </div>
        </div>


        <div class="row align-items-start p-5">
            <div class="col">
                <div class="container-sm px-4">
                    <h3>Product Management</h3>
                </div>
                <div class="container-sm mt-4 px-4">
                    <form action="InsertProduct" method="post">
                        <div class="form-group">
                            <label for="productName" class="form-
label">Product Name:</label>
                            <input type="text" class="form-control"
name="productName" required>
                        </div>
                        <div class="form-group mt-2">
                            <label for="productName" class="form-
label">Brand:</label>
                            <input type="text" class="form-control"
name="brand">
                        </div>
                        <div class="form-group mt-2">
                            <label for="productName" class="form-
label">Type:</label>
                            <input type="text" class="form-control"
name="type">
                        </div>
                        <div class="form-group mt-2">
                            <label for="productName" class="form-
label">Sensor Type:</label>
                            <input type="text" class="form-control"
name="sensorType">
                        </div>
                        <br>
                        <button type="submit" class="btn btn-
danger">Save</button>
                    </form>
                </div>
            </div>
            <div class="col">
```

```
                <div class="container-sm px-4">
                    <form action="index.jsp">
                        <div class="input-group"><input type="text"
class="form-control" name="value"></td>
                            <button type="submit" class="btn btn-
danger">Search</button></div>
                    </form>
                    <br>
                    <table class="table table-striped">
                        <tr>
                            <th>Product Name</th>
                            <th>Brand</th>
                            <th>Type</th>
                            <th>Sensor Type</th>
                            <th colspan="2">Action</th>

                        </tr>
                        <%
                            InitialContext ic = new InitialContext();
                            ProductSessionBeanRemote psbr =
(ProductSessionBeanRemote)
ic.lookup("com.ndcamera.ejb.ProductSessionBeanRemote");
                            List<Product> products = new ArrayList<>();
                            if
(pageContext.getRequest().getParameter("value") != null &&
!pageContext.getRequest().getParameter("value").equals("") &&
psbr.findByProductName(pageContext.getRequest().getParameter("value"))
!= null) {
                                products.add(psbr.findByProductName(pag
eContext.getRequest().getParameter("value")));
                            } else {
                                products = psbr.getAllProducts();
                            }
                            pageContext.setAttribute("products",
products);
                        %>
                        <c:forEach var="product" items="${products}">
                            <tr>
                            <form action="UpdateProduct" method="post">
                                <td><input type="text"
name="productName" value="${product.productName}" required></td>
                                <td><input type="text" name="brand"
value="${product.brand}"></td>
                                <td><input type="text" name="type"
value="${product.type}"></td>
```

```
                                <td><input type="text"
name="sensorType" value="${product.sensorType}"></td>
                                <input type="hidden" name="id"
value="${product.id}">
                                <td>
                                    <button type="submit" class="btn
btn-primary">Update</button>
                                </td>
                            </form>
                            <td><button class="btn btn-danger"><a
href="DeleteProduct?id=${product.id}">Delete</a></button></td>
                            </tr>
                        </c:forEach>
                    </table>
                </div>
            </div>
        </div>
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4ddQ3GYNS7NTKfAdVQSZe"
crossorigin="anonymous"></script>
    </body>
</html>
```

A web page for managing products in an online photography store is represented by the JavaServer Pages (JSP) file in this example. The page includes a form for adding new products as well as a table for listing those that are already there.

ArrayList, Product, and other essential Java classes and libraries are imported in the JSP file's initial import declarations. Also, a taglib declaration is included for the page's use of JSTL (JavaServer Pages Standard Tag Library) tags.

The page's HTML content is contained within the html> elements. The page's title and CSS styles are defined in the head> part of the code. The CSS styles specify how the page will look, including the font color and button styles. The Bootstrap framework provides pre-designed styles for this web page.

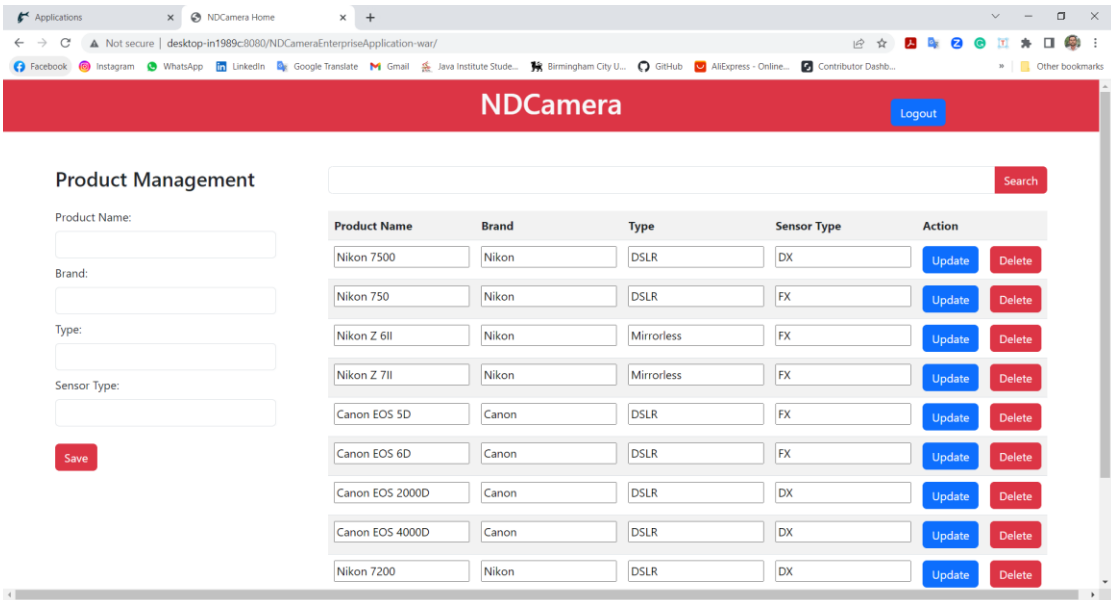
The primary material is located in the page's body> section. It has a header section called "NDCamera," a logout button, a row with two columns for managing products, and other features. A form for adding new products can be found in the first column, and a form for searching already-existing products and a table showing the products can be found in the second column.

The product name, brand, kind, and sensor type are only a few of the input fields on the form for adding new products. Moreover, there is a submit button for the data entered.

A search value can be entered into the input field on the form for existing products, and a submit button sends the search request to the server.

Using JSTL tags, data from a Java remote session bean called ProductSessionBeanRemote is used to populate the table used to show products. The InitialContext class is used to look up the session bean. The table features two action buttons for updating and deleting products, as well as columns for showing product name, brand, kind, and sensor type. A form's input fields show the product information, which the user can change and then submit for updating. The delete button connects to a servlet called DeleteProduct that deletes the product from the database by passing the product ID as a parameter.

The JSP file also contains Java scriptlets that retrieve product data from the session bean and set it as an attribute in other Java code contained in %%> tags.



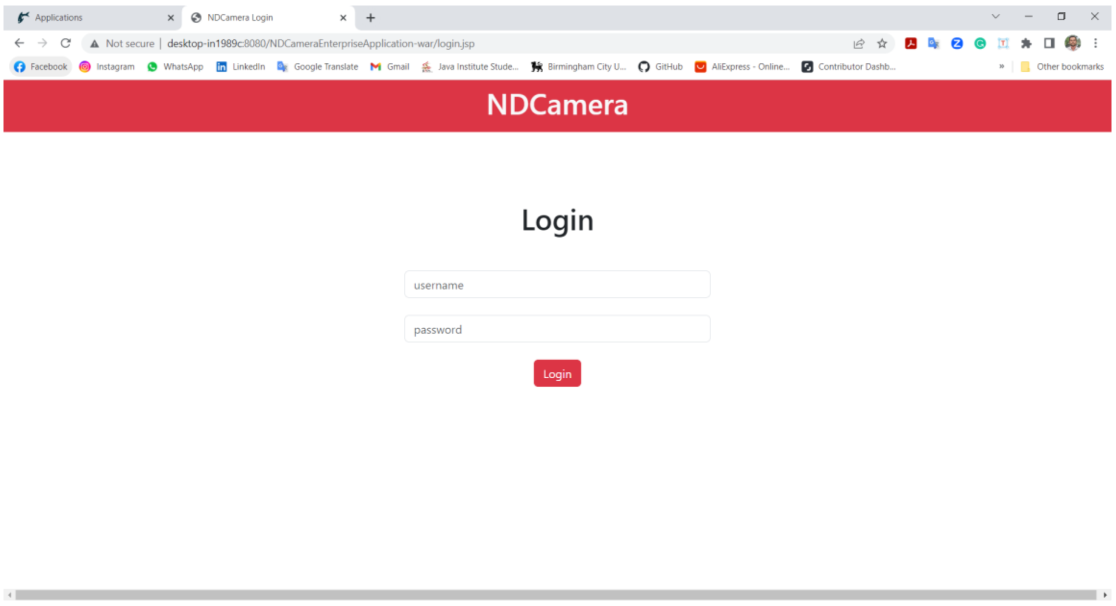
Product Name	Brand	Type	Sensor Type	Action
Nikon 7500	Nikon	DSLR	DX	Update Delete
Nikon 750	Nikon	DSLR	FX	Update Delete
Nikon Z 6II	Nikon	Mirrorless	FX	Update Delete
Nikon Z 7II	Nikon	Mirrorless	FX	Update Delete
Canon EOS 5D	Canon	DSLR	FX	Update Delete
Canon EOS 6D	Canon	DSLR	FX	Update Delete
Canon EOS 2000D	Canon	DSLR	DX	Update Delete
Canon EOS 4000D	Canon	DSLR	DX	Update Delete
Nikon 7200	Nikon	DSLR	DX	Update Delete

2.2.2.2.4. login.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>NDCamera Login</title>
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"
crossorigin="anonymous">
    </head>
    <body>
        <div class="container-fluid text-center bg-danger p-2">
            <div class="row">
                <div class="col-12 text-light"><h1>NDCamera</h1></div>
            </div>
        </div>

        <div class="row m-5 p-5 text-center">
            <div class="col-4 m-auto">
                <h1>Login</h1>
                <form action="j_security_check" method="post"
class="mt-5">
                    <div class="form-outline mb-4">
                        <input type="text" name="j_username"
id="form2Example1" class="form-control" placeholder="username"
required/>
                    </div>
                    <div class="form-outline mb-4">
                        <input type="password" name="j_password"
id="form2Example2" class="form-control" placeholder="password"
required/>
                    </div>
                    <button type="submit" class="btn btn-
danger">Login</button>
                </form>
            </div>
        </div>
    </form>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4ddQ3GYNS7NTKfAdVQSZe"
crossorigin="anonymous"></script>
```

```
    </body>
</html>
```

An HTML page for a login form for an application called NDCamera is created using the provided code. The Bootstrap framework provides pre-designed styles for this web page. This form element defines the HTTP method used for submission as well as the action URL (j security check) where the form data will be transmitted (post).The name property of this username input field is set to "j username." The name property of this password input field is set to "j password."



2.2.2.2.5. DeleteProduct.java

```
package com.ndcamera.web;

import com.ndcamera.ejb.ProductSessionBeanRemote;
import java.io.IOException;
import javax.ejb.EJB;
import javax.ejb.EJBAccessException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "DeleteProduct", urlPatterns = {"/DeleteProduct"})
```

```java
public class DeleteProduct extends HttpServlet {

    @EJB
    private ProductSessionBeanRemote psbr;

    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String id = request.getParameter("id");
        try {
            psbr.deleteProduct(Integer.parseInt(id));
            response.sendRedirect("index.jsp");
        } catch (EJBAccessException ex) {
            ex.printStackTrace();
            response.sendRedirect("403.jsp");
        } catch (Exception ex) {
            ex.printStackTrace();
            response.sendRedirect("error.jsp");
        }
    }
}
```

A Java servlet is the piece of given code that manages removing a product from a web application. This is a quick breakdown of the code:

- The servlet is called "DeleteProduct" and has the URL pattern "/DeleteProduct," therefore it will be used whenever the web application receives a request with this URL.
- The @EJB annotation was used to inject a reference to a remote "ProductSessionBeanRemote" EJB (Enterprise Java Bean) into the servlet. The deletion of the product is carried out using this EJB.
- The handler for the HTTP GET method, doGet, is overridden by the servlet. When a client makes a GET request to the servlet's URL, this method is invoked.
- The "id" parameter of the request, which must include the ID of the product to be deleted, is retrieved by the servlet.
- The product is then deleted from the system by calling the deleteProduct method on the injected EJB and supplying the ID as an integer argument.

- If the deletion is successful, the servlet will use response.sendRedirect("index.jsp") to send a redirect response to the page, which will direct the user to the web application's main page.
- The servlet sends a redirect response to the "403.jsp" page, which is probably a customized error page for access prohibited, if an EJBAccessException is thrown, indicating that the user does not have the required access rights to conduct the deletion.
- The "error.jsp" page, which is probably a customized error page for general issues, receives a redirect answer if any more exceptions are encountered during the deletion process.

This servlet is in charge of handling the deletion of a product from a web application. To carry out the deletion itself, it employs an injected EJB, and depending on how the deletion operation turns out, it also handles probable exceptions and redirects to the proper pages.

2.2.2.2.6. InsertProduct.java

```java
package com.ndcamera.web;

import com.ndcamera.ejb.ProductSessionBeanRemote;
import com.ndcamera.entity.Product;
import java.io.IOException;
import javax.ejb.EJB;
import javax.ejb.EJBAccessException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

    @EJB
    private ProductSessionBeanRemote psbr;

    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        try {
            String productName = request.getParameter("productName");
            String brand = request.getParameter("brand");
            String type = request.getParameter("type");
            String sensorType = request.getParameter("sensorType");

            Product product = new Product();
            product.setProductName(productName);
            product.setBrand(brand);
            product.setType(type);
            product.setSensorType(sensorType);
            psbr.insertProduct(product);
            response.sendRedirect("index.jsp");
        } catch (EJBAccessException ex) {
            ex.printStackTrace();
            response.sendRedirect("403.jsp");
        } catch (Exception ex) {
            ex.printStackTrace();
            response.sendRedirect("error.jsp");
        }
    }
}
```

This Java servlet code uses the Enterprise JavaBeans (EJB) technology to add a product to a web application. The code is broken down as follows:

The fundamental class for building Java servlets that manage HTTP requests and responses is HttpServlet, which the InsertProduct class extends.

The name and URL patterns of the servlet are defined using the @WebServlet annotation.

An instance of the ProductSessionBeanRemote EJB is injected into the servlet using the @EJB annotation. This enables interaction between the servlet and the EJB-encapsulated business logic.

Handling HTTP POST requests begins with the doPost function. In order to input the data for the product, such as the name, brand, kind, and sensor type, it receives the parameters from the request object.

The retrieved parameter values are used to create a new Product object, and its properties are set appropriately.

The Product object is passed as an argument to the insertProduct method of the injected ProductSessionBeanRemote EJB, which is in charge of persisting the product in the database.

The response is delivered to the index.jsp page if the insertion is successful, signifying a successful operation.

The answer is routed to the 403.jsp page to signify an unauthorized access attempt if an EJBAccessException is caught, which signifies the servlet lacks the requisite permissions to access the EJB.

The answer is routed to the error.jsp page to indicate a general error in the operation if any other exception occurs.

In general, this servlet code is in charge of managing the EJB technology-based insertion of a product into a web application and handling various circumstances, including successful insertion, unauthorized access, and common faults.

2.2.2.2.7. Logout.java

```
package com.ndcamera.web;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        request.logout();
        response.sendRedirect("login.jsp");
    }
}
```

The "Logout" functionality in a web application is handled by the Java servlet code in the aforementioned code. It extends the Java foundation class for servlet development known as HttpServlet. The @WebServlet annotation links the servlet to the URL pattern "/Logout".

The HttpServlet class's doGet() method, which is overridden, manages HTTP GET requests. The doGet() method is called whenever a GET request is made to the URL associated with this servlet.

The request.logout() method is called within the doGet() method. With this technique, the currently logged-in user can log out and the current session can be nullified. It is frequently employed in web applications that implement authentication and authorization using Java EE security.

The response.sendRedirect("login.jsp") method is used to send the user to a login page after using request.logout() (in this case, "login.jsp"). This is typically carried out after logging out to remind the user to log in again if they wish to access the web application's protected resources.

In conclusion, the aforementioned code represents a servlet that manages the functionality of logging out by invalidating the session and directing the user to a login page.

2.2.2.2.8. SearchProduct.java

```
package com.ndcamera.web;

import com.ndcamera.ejb.ProductSessionBeanRemote;
import com.ndcamera.entity.Product;
import java.io.IOException;
import javax.ejb.EJB;
import javax.ejb.EJBAccessException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

    @EJB
    private ProductSessionBeanRemote psbr;

    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String productName = request.getParameter("productName");
        try {
            Product product = psbr.findByProductName(productName);
            response.getWriter().write(product.getProductName());
        } catch (EJBAccessException ex) {
            ex.printStackTrace();
            response.sendRedirect("403.jsp");
        } catch (Exception ex) {
            ex.printStackTrace();
            response.sendRedirect("error.jsp");
        }
    }
}
```

This code is for the Java servlet "SearchProduct," which allows users to look up products using their names. A session bean called "ProductSessionBeanRemote" is used for remote communication using the Enterprise JavaBeans (EJB) technology.

The servlet's name and URL pattern are specified by the annotation @WebServlet. To handle HTTP requests and responses, it extends HttpServlet.

When the servlet receives a GET request, the single doGet method is invoked. The name of the product being searched for is retrieved from the request's "productName" field. The ProductSessionBeanRemote session bean's findByProductName method is then invoked in order to look up the product in the backend. If a matching item is discovered, response is used to add the name of that item to the response. getWriter(). write(product.getProductName()).

In the event that an EJBAccessException is encountered, the stack trace is printed and the response is forwarded to a "403.jsp" page, signifying a banned access issue. A generic error is indicated by sending the response to a "error.jsp" page and printing the stack trace if any other exception is caught.

2.2.2.2.9. UpdateProduct.java

```
package com.ndcamera.web;

import com.ndcamera.ejb.ProductSessionBeanRemote;
import com.ndcamera.entity.Product;
import java.io.IOException;
import javax.ejb.EJB;
import javax.ejb.EJBAccessException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

    @EJB
    private ProductSessionBeanRemote psbr;

    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        try {
            String id = request.getParameter("id");
            String productName = request.getParameter("productName");
            String brand = request.getParameter("brand");
            String type = request.getParameter("type");
            String sensorType = request.getParameter("sensorType");

            Product product = psbr.findById(Integer.parseInt(id));
            product.setId(Integer.parseInt(id));
            product.setProductName(productName);
            product.setBrand(brand);
            product.setType(type);
            product.setSensorType(sensorType);
            psbr.updateProduct(product);
            response.sendRedirect("index.jsp");
        } catch (EJBAccessException ex) {
            ex.printStackTrace();
            response.sendRedirect("403.jsp");
        } catch (Exception ex) {
            ex.printStackTrace();
            response.sendRedirect("error.jsp");
        }
    }
}
```

A Java servlet that manages HTTP POST requests to update a product in a web application is provided in the code. Here is a summary of how the code works:

- The HttpServlet class, a basic class for building servlets in Java online applications, is what the servlet extends.
- The servlet is mapped to the URL pattern "/UpdateProduct" using the @WebServlet annotation. This indicates that this servlet will process requests that follow this pattern of URLs.
- An instance of the ProductSessionBeanRemote EJB (Enterprise JavaBean) is injected into the servlet using the @EJB annotation. A remote EJB used to communicate with the business logic for managing products is called the ProductSessionBeanRemote.
- To support HTTP POST requests, the doPost method has been overridden. When a client submits a form to the servlet using the HTTP POST method, this function is invoked.
- The values of the changed product properties (such as id, productName, brand, type, and sensorType) are retrieved by the servlet using the request inside the doPost method. Method getParameter().
- The ProductSessionBeanRemote EJB's findById function is then invoked by the servlet to obtain the product with the specified id.
- The new attribute values are then added to the obtained product using the respective setter methods.
- To save the changed product in the database, the updateProduct method on the ProductSessionBeanRemote EJB is invoked.
- The response.sendRedirect() method of the servlet redirects the response to the "index.jsp" page, which is often the home page or the product listing page, if the update is successful.

- The servlet captures any EJBAccessExceptions and sends the response to the "403.jsp" page, which is often an access refused page, if the caller does not have authorization to access the EJB.
- The answer is forwarded to the "error.jsp" page, which is normally an error page to handle unexpected exceptions, if there is any additional exception.

This code requires that the web application contains JSP pages (such as "index.jsp," "403.jsp," and "error.jsp") that manage the client-side rendering of HTML content in accordance with the results of the servlet's processing. The provided code sample does not contain the real ProductSessionBeanRemote EJB implementation or the related JSP pages.

2.2.2.2.10. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="jakarta.ee/xml/ns/jakartaee
jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd" version="5.0"
>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Security</web-resource-name>
            <url-pattern>/InsertProduct</url-pattern>
            <url-pattern>/UpdateProduct</url-pattern>
            <url-pattern>/DeleteProduct</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>superadmin</role-name>
            <role-name>admin</role-name>
            <role-name>user</role-name>
        </auth-constraint>
    </security-constraint>
    <login-config>
```

```
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/login.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>
</web-app>
```

This web application's XML configuration file lists different security options. Let's go over the various parts of the code:

- This establishes the XML namespace for the Jakarta EE (formerly Java Enterprise Edition) standards. xmlns="jakarta.ee/xml/ns/jakartaee"
- xmlns:xsi="w3.org/2001/XMLSchema-instance": The XML namespace for the XML Schema Instance, which is used to indicate schema locations, is declared in this.
- xsi:schemaLocation="jakarta.ee/xml/ns/jakartaee jakarta.ee/xml/ns/jakartaee/web-app 5 0.xsd": The location of the XML schema that will be used to validate the structure of this XML document is specified here. It relates to the Jakarta EE web application schema version 5.0 in this instance.
- This indicates the version of the Jakarta EE specification that this web application is made to abide by, which is version="5.0".
- This specifies configuration options for web application sessions, such as the length of the session timeout (in this case, 30 minutes).
- This specifies a security constraint for particular online resources (in this case, URLs) that only allows users with specific responsibilities access. The GET and POST HTTP methods are supported for accessing the URLs "/InsertProduct," "/UpdateProduct," and "/DeleteProduct" by the roles "superadmin," "admin," and "user."

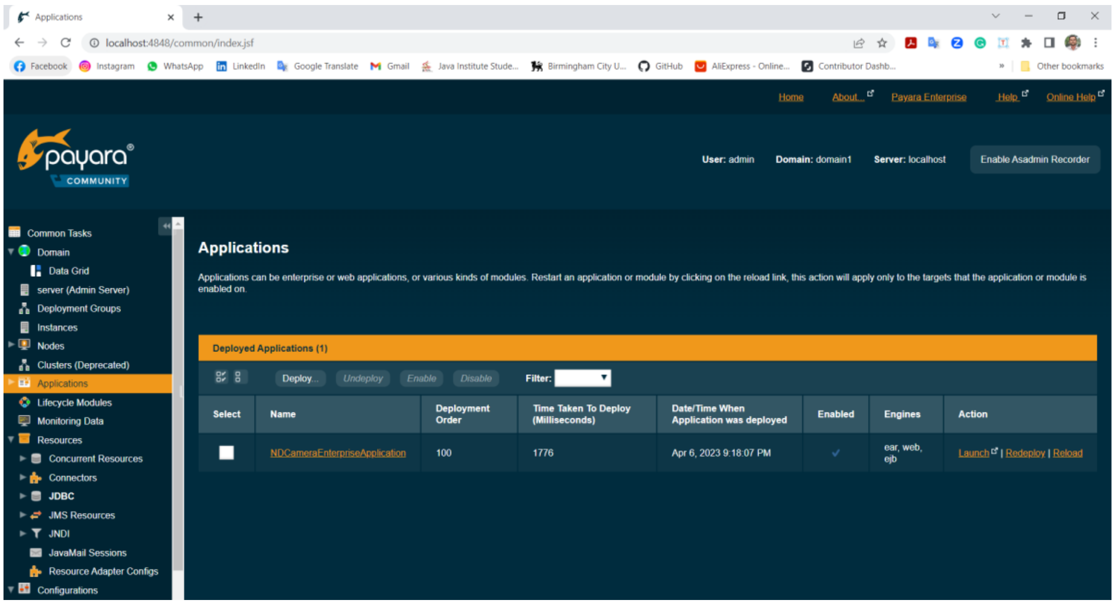
- The authentication mechanism is "FORM" and the login settings for the web application is defined by the login-config> tag. Also, it provides the URLs for the error page ("/error.jsp") and the login page ("/login.jsp") in the event that authentication is unsuccessful.
- When a 403 HTTP status code (Forbidden) is detected, this specifies the error page that will be shown. The location of the error page is set to "/403.jsp."

In general, this XML configuration file configures security settings for a Jakarta EE web application, including login configuration, session timeout, and access control based on roles.

2.3. Deployment

I used the Payara server, a Java application server that offers a framework for hosting Java EE applications, to launch this project. I made ndCameraConnectionPool JDBC Connection Pool and ndCameraDbConnection JDBC Resources and configured them using the persistence.xml file to configure the database. This makes it possible for my program to access the database and carry out the required database activities.

I added three user ids—superadmin, admin, and user—to the server-config security of the Payara server to provide a strong security architecture. As a result, it is possible to implement role-based access management, in which users are given particular roles with variable levels of access privileges according to their user type. The highest level of access is granted to the superadmin, who also ensures that appropriate security measures are in place to safeguard sensitive data and application functionality.



User: admin **Domain:** domain1 **Server:** localhost

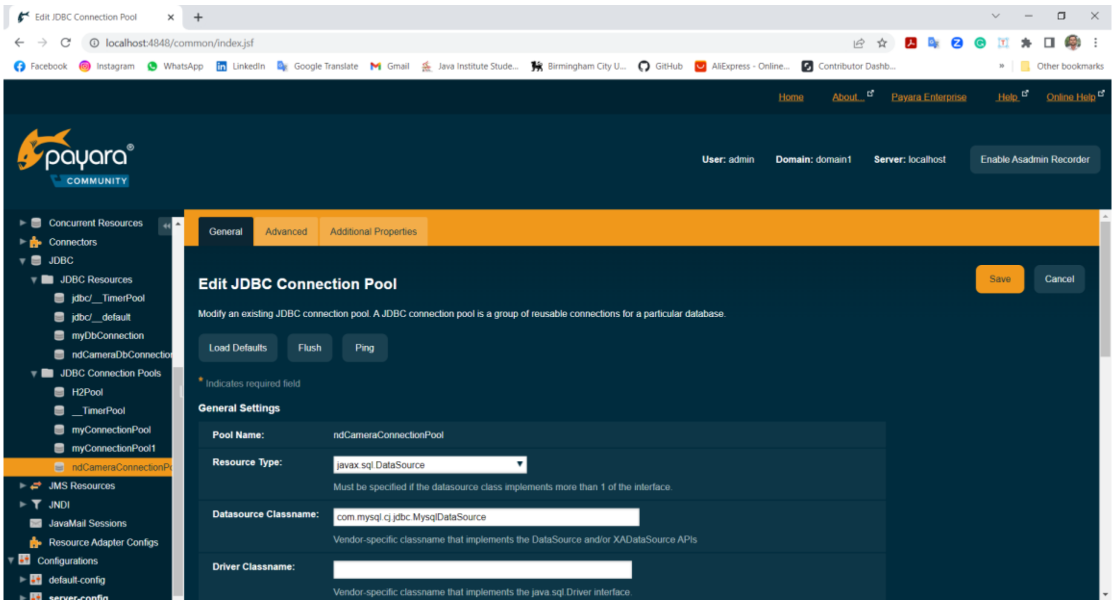
Applications

Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

Deployed Applications (1)

Filter:

Select	Name	Deployment Order	Time Taken To Deploy (Milliseconds)	Date/Time When Application was deployed	Enabled	Engines	Action
☐	NDCameraEnterpriseApplication	100	1776	Apr 6, 2023 9:18:07 PM	✓	ear, web, ejb	Launch | Redeploy | Reload



User: admin **Domain:** domain1 **Server:** localhost

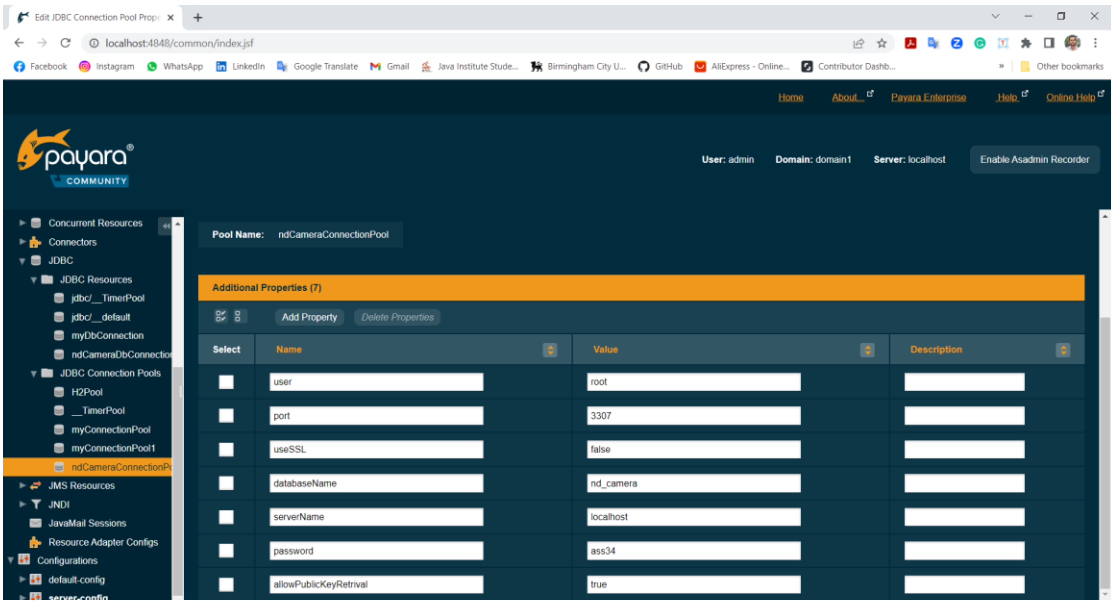
Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

* Indicates required field

General Settings

Pool Name:	ndCameraConnectionPool
Resource Type:	javax.sql.DataSource Must be specified if the datasource class implements more than 1 of the interface.
Datasource Classname:	com.mysql.cj.jdbc.MysqlDataSource Vendor-specific classname that implements the DataSource and/or XADataSource APIs
Driver Classname:	 Vendor-specific classname that implements the java.sql.Driver interface.

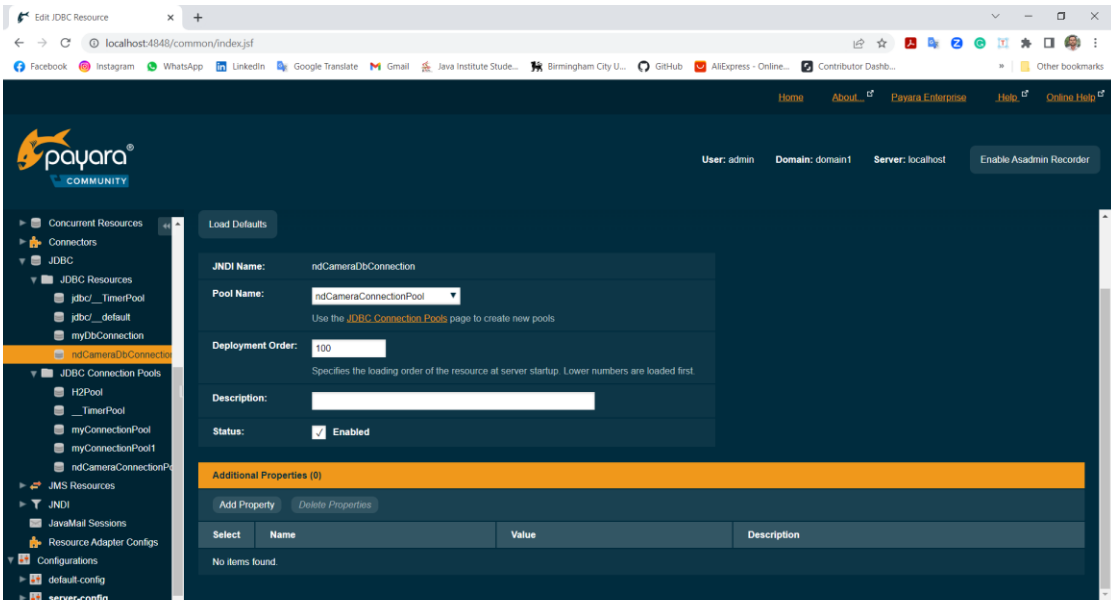


User: admin **Domain:** domain1 **Server:** localhost

Pool Name: ndCameraConnectionPool

Additional Properties (7)

Select	Name	Value	Description
☐	user	root	
☐	port	3307	
☐	useSSL	false	
☐	databaseName	nd_camera	
☐	serverName	localhost	
☐	password	ass34	
☐	allowPublicKeyRetrival	true	



User: admin **Domain:** domain1 **Server:** localhost

JNDI Name: ndCameraDbConnection

Pool Name: ndCameraConnectionPool

Use the JDBC Connection Pools page to create new pools

Deployment Order: 100

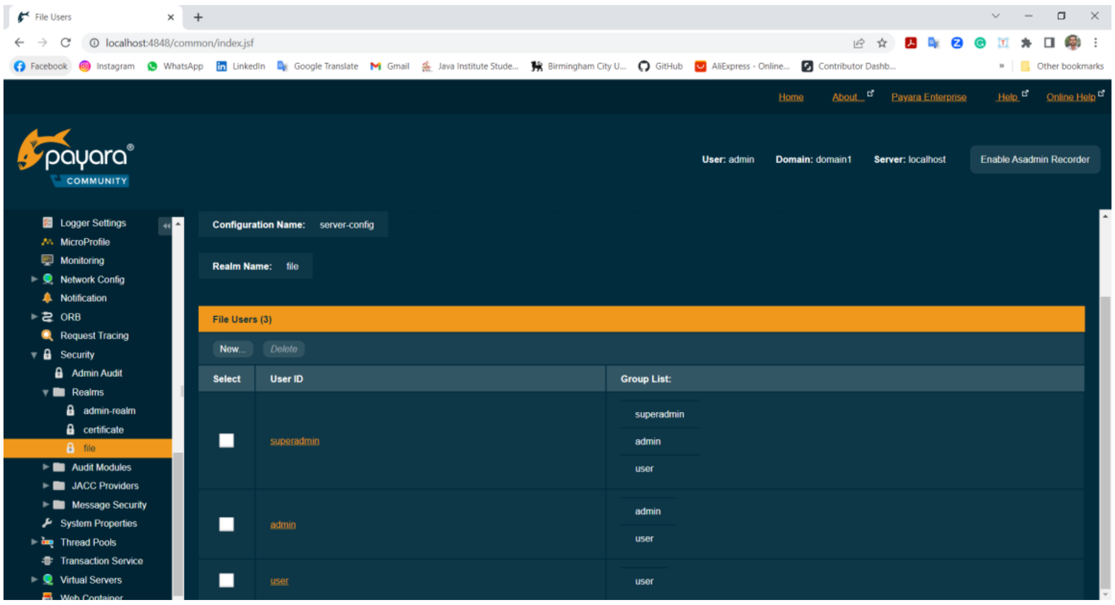
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Status: ✓ Enabled

Additional Properties (0)

Select	Name	Value	Description
No items found.			


Configuration Name: server-config

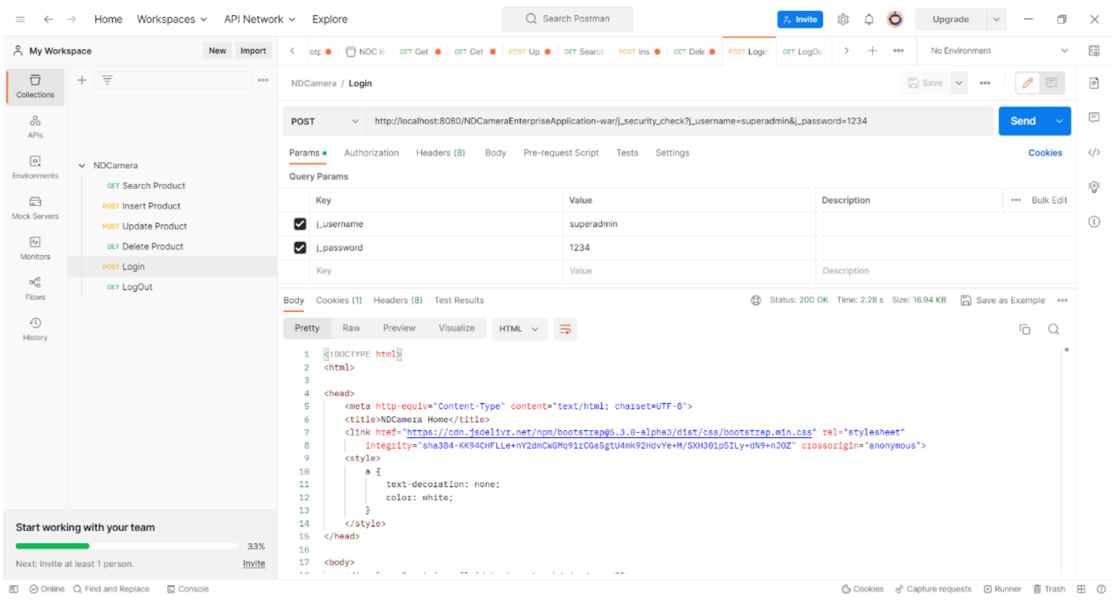
Realm Name: file

File Users (3)

Select	User ID	Group List:
☐	superadmin	superadmin admin user
☐	admin	admin user
☐	user	user

2.4. Testing

I used Postman to Test this application.


NDCamera / Login

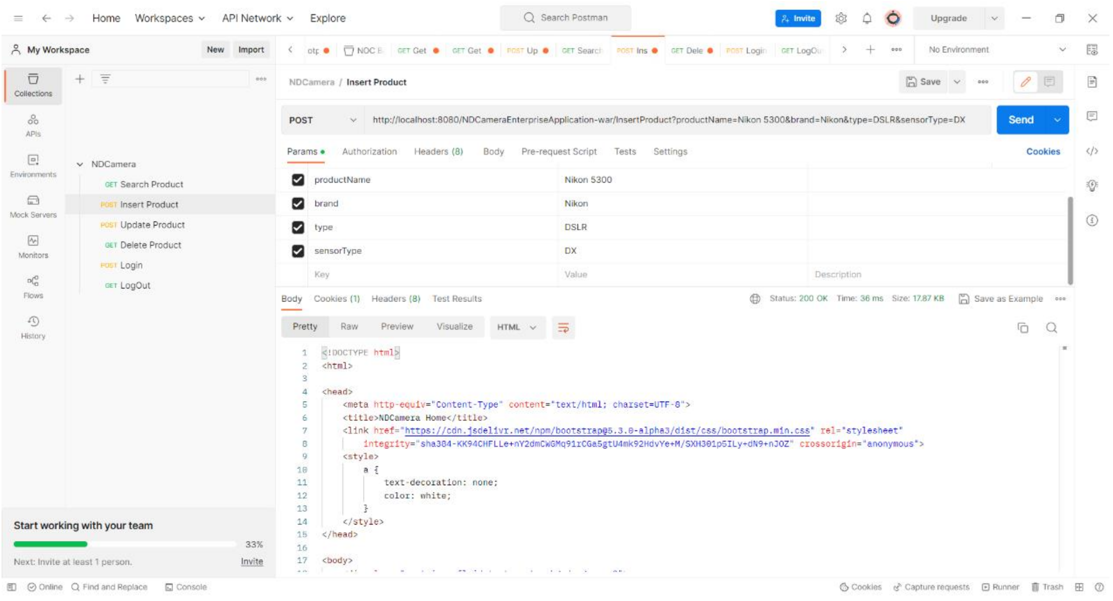
POST http://localhost:8080/NDCameraEnterpriseApplication-war/j_security_check?j_username=superadmin&j_password=1234

Query Params

Key	Value	Description
j_username	superadmin	
j_password	1234	

Status: 200 OK Time: 2.28 s Size: 16.94 KB

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>NDCamera Home</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ" crossorigin="anonymous">
    <style>
        a {
            text-decoration: none;
            color: white;
        }
    </style>
</head>

<body>
```



NDCamera / Insert Product

POST http://localhost:8080/NDCameraEnterpriseApplication-war/InsertProduct?productName=Nikon 5300&brand=Nikon&type=DSLR&sensorType=DX

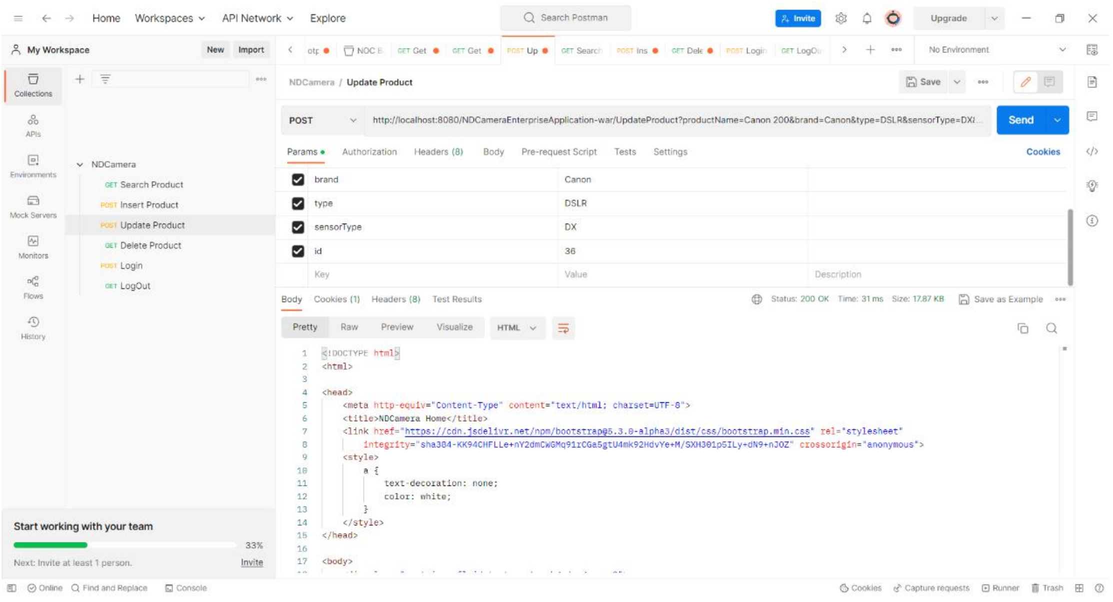
Params • | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

	Key	Value	Description
✓	productName	Nikon 5300	
✓	brand	Nikon	
✓	type	DSLR	
✓	sensorType	DX	

Body | Cookies (1) | Headers (8) | Test Results — Status: 200 OK Time: 36 ms Size: 17.87 KB

Pretty | Raw | Preview | Visualize | HTML

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>NDCamera Home</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ" crossorigin="anonymous">
    <style>
        a {
            text-decoration: none;
            color: white;
        }
    </style>
</head>

<body>
```



NDCamera / Update Product

POST http://localhost:8080/NDCameraEnterpriseApplication-war/UpdateProduct?productName=Canon 200&brand=Canon&type=DSLR&sensorType=DX&...

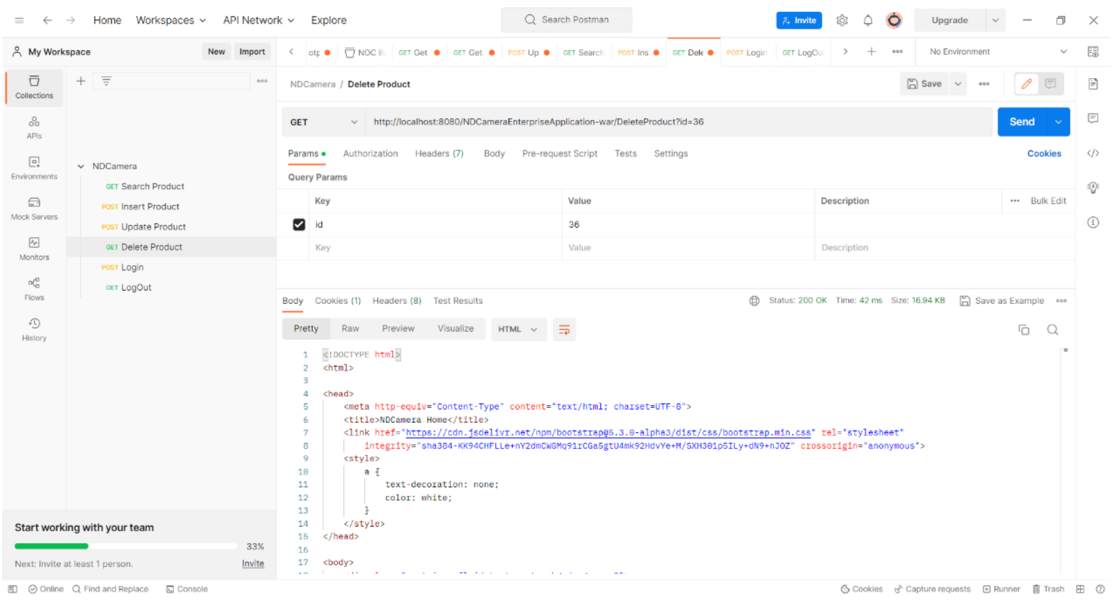
Params • | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

	Key	Value	Description
✓	brand	Canon	
✓	type	DSLR	
✓	sensorType	DX	
✓	id	36	

Body | Cookies (1) | Headers (8) | Test Results — Status: 200 OK Time: 31 ms Size: 17.87 KB

Pretty | Raw | Preview | Visualize | HTML

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>NDCamera Home</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ" crossorigin="anonymous">
    <style>
        a {
            text-decoration: none;
            color: white;
        }
    </style>
</head>

<body>
```



NDCamera / Delete Product

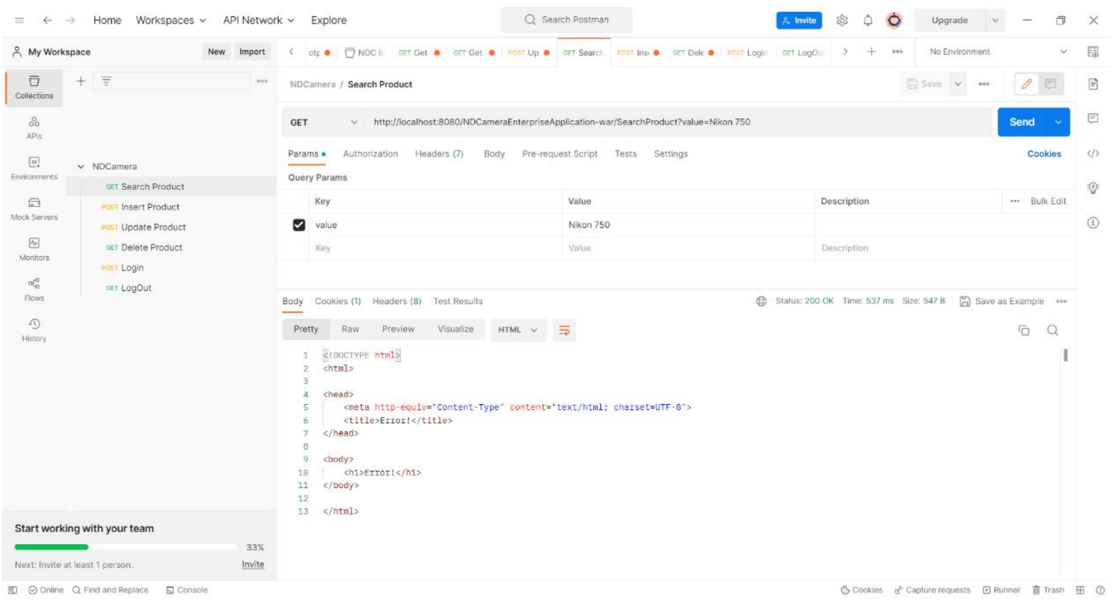
GET http://localhost:8080/NDCameraEnterpriseApplication-war/DeleteProduct?id=36

Query Params

Key	Value	Description
id	36	

Status: 200 OK Time: 42 ms Size: 16.94 KB

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>NDCamera Home</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ" crossorigin="anonymous">
    <style>
        a {
            text-decoration: none;
            color: white;
        }
    </style>
</head>

<body>
```



NDCamera / Search Product

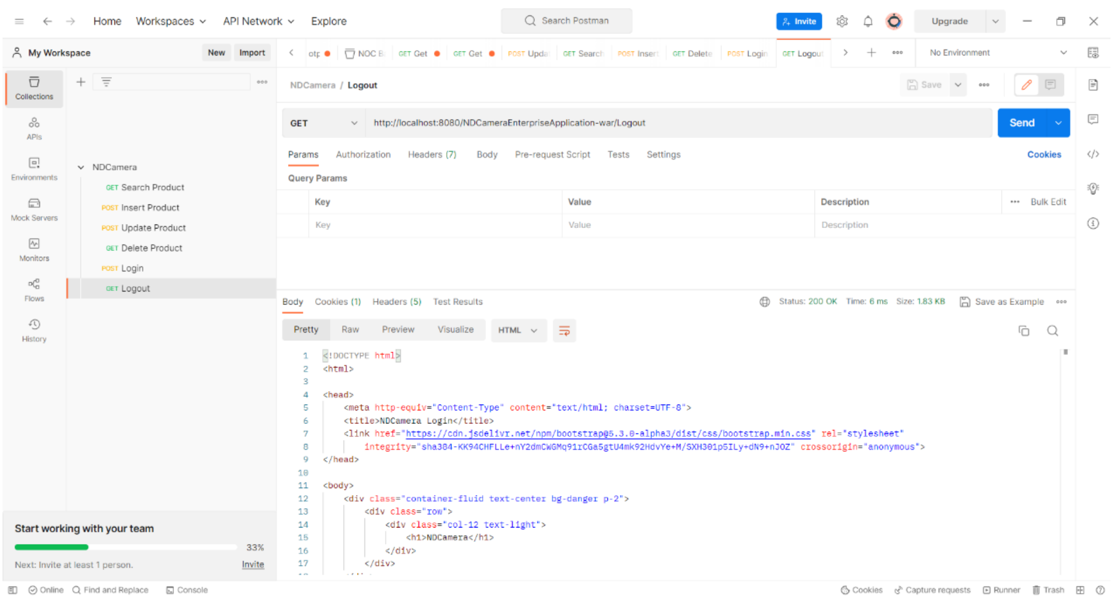
GET http://localhost:8080/NDCameraEnterpriseApplication-war/SearchProduct?value=Nikon 750

Query Params

Key	Value	Description
value	Nikon 750	

Status: 200 OK Time: 537 ms Size: 547 B

```
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Error!</title>
</head>

<body>
    <h1>Error!</h1>
</body>

</html>
```



GET http://localhost:8080/NDCameraEnterpriseApplication-war/Logout

Status: 200 OK Time: 6 ms Size: 1.83 KB

```html
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>NDCamera Login</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ" crossorigin="anonymous">
</head>

<body>
    <div class="container-fluid text-center bg-danger p-2">
        <div class="row">
            <div class="col-12 text-light">
                <h1>NDCamera</h1>
            </div>
        </div>
```

3. Conclusion

Time services to manage task scheduling, Interceptor classes to log user activity, Different types of transactions to ensure data consistency and reliability, Robust security architecture to protect user data and ensure privacy, Exception handling to optimize application performance, EJB components in a split directory structure, and Testing tools to test the application works, are using to design, develop and deploy of an enterprise Java application is very easy user-friendly, efficient, and secure.

4. References:

www.javatpoint.com. (n.d.). *What is EJB - javatpoint*. [online] Available at: https://www.javatpoint.com/what-is-ejb.

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

Wikipedia. (2022). *Jakarta Enterprise Beans*. [online] Available at: https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans [Accessed 6 Apr. 2023].

www.tutorialspoint.com. (n.d.). *EJB Tutorial*. [online] Available at: https://www.tutorialspoint.com/ejb/index.htm [Accessed 6 Apr. 2023].

docs.oracle.com. (n.d.). *What Is an Enterprise Bean? - The Java EE 5 Tutorial*. [online] Available at: https://docs.oracle.com/javaee/5/tutorial/doc/bnblt.html.

docs.oracle.com. (n.d.). *Creating a Split Development Directory Environment*. [online] Available at: https://docs.oracle.com/cd/E13222_01/wls/docs90/programming/splitcreate.html [Accessed 6 Apr. 2023].

flylib.com. (n.d.). *Split Directory Development | Packaging and Deployment*. [online] Available at: https://flylib.com/books/en/2.107.1/split_directory_development.html.

Oracle.com. (2019). *Java 2 Platform, Enterprise Edition (J2EE) Overview*. [online] Available at: https://www.oracle.com/java/technologies/appmodel.html#:~:text=The%20J2EE%20platform%20provides%20choices [Accessed 6 Apr. 2023].

Wikipedia. (2021). *Jakarta EE*. [online] Available at: https://en.wikipedia.org/wiki/Jakarta_EE.

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

docs.oracle.com. (n.d.). *Understanding EJB Timer Services*. [online] Available at: https://docs.oracle.com/cd/E16439_01/doc.1013/e13981/undejdev012.htm [Accessed 6 Apr. 2023].

www.tutorialspoint.com. (n.d.). *EJB - Timer Service*. [online] Available at: https://www.tutorialspoint.com/ejb/ejb_timer_service.htm [Accessed 6 Apr. 2023].

docs.oracle.com. (n.d.). *25.6 Using Interceptors in CDI Applications - Java Platform, Enterprise Edition: The Java EE Tutorial (Release 7)*. [online] Available at: https://docs.oracle.com/javaee/7/tutorial/cdi-adv006.htm [Accessed 6 Apr. 2023].

F.Marchioni (2021). *How to code EJB interceptors like a pro*. [online] Mastertheboss. Available at: http://www.mastertheboss.com/java-ee/ejb-3/ejb-interceptors-in-depth/ [Accessed 6 Apr. 2023].

Packt (2013). *Using Events, Interceptors, and Logging Services*. [online] Packt Hub. Available at: https://hub.packtpub.com/using-events-interceptors-and-logging-services/ [Accessed 6 Apr. 2023].

Developer.com. (2017). *Managing Transactions with EJB*. [online] Available at: https://www.developer.com/java/data/managing-transactions-with-ejb/.

docs.oracle.com. (n.d.). *Chapter 28 Transactions (The Java EE 6 Tutorial)*. [online] Available at: https://docs.oracle.com/cd/E19798-01/821-1841/bncih/index.html [Accessed 6 Apr. 2023].

docs.oracle.com. (n.d.). *Chapter 24 Introduction to Security in the Java EE Platform (The Java EE 6 Tutorial)*. [online] Available at: https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpig/index.html.

docs.oracle.com. (n.d.). *Fusion Middleware Tuning Performance of Oracle WebLogic Server*. [online] Available at: https://docs.oracle.com/middleware/12213/wls/PERFM/ejb_tuning.htm#PERFM238 [Accessed 6 Apr. 2023].

www.tutorialspoint.com. (n.d.). *EJB - Exception Handling*. [online] Available at: https://www.tutorialspoint.com/ejb/ejb_exception_handling.htm [Accessed 6 Apr. 2023].