

# ENAPP 2012 - WEBSHOP

IWAN PAOLUCCI

## 1. INTRO

In the module ENAPP.H12 a Webshop has to be developed. This documentation covers the most necessary information and a short overview of the architecture.

### 1.1. Technology

A short overview over the technologies used.

- J2EE 1.6 (Java Enterprise Edition)
- Java 7
- Java Server Faces (JSF)
- Java Message Service (JMS)
- SOAP Webservices
- Rest Webservices
- JDBC / JPA (MySQL)

### 1.2. Usage

The application is more or less self-explanatory or should be. There are separate users for administration and shopping. A already installed user can be used or a new one can be created.

#### **Webshop**

0160.intra015.el.campus.intern:8080/enapp12-tapaoluc-web/index.xhtml

User/Password: dude/dude

#### **Adminpanel**

0160.intra015.el.campus.intern:8080/enapp12-tapaoluc-web-admin/index.xhtml

User/Password: admin/admin

## 2. INFRASTRUCTURE

This section covers information about the Infrastructure.

### 2.1. Integration

Description of the integrationplatform in the EnterpriseLab @ HSLU.

#### 2.1.1. OperatingSystem

Type: Solaris

User: tapaoluc (EL User)

root Password: ENAPP\_H12

#### 2.1.2. Database

Type: MySQL

Server: s0160.intra015.el.campus.intern

User: enapp

Password: enapp

#### 2.1.3. Applicationserver

Type: Glassfish 3+

Adminpanel: <https://s0160.intra015.el.campus.intern:4848>

User: admin

Password: ENAPP\_H12

#### 2.1.4. /etc/hosts

For the connection to the Navision service an additional entry to the hosts file is needed.

```
#navision
```

```
10.29.2.12      icompanydb01.icompany.intern
```

## 2.2. Configuration.

### 2.2.1. Database

*JDBC Connection Pool*

Poolname: EnappWebshopTapaolucPool

Resource Type: javax.sql.DataBase

Datasource Classname: com.mysql.jdbc.jdbc2.optional.MysqlDataSource

*Additional Properties on Connection Pool*

password = enapp

user = enapp

servername = s0160.intra015.el.campus.intern

roleName = com.mysql.jdbc.Driver

datasourceName = jdbc:mysql://s0160.intra015.el.campus.intern:3306/enappwebshop

databaseName = enappwebshop

portNumber = 3306

*Ressource*

JNDI Name: jdbc/enappwebshoptapaoluc

Poolname: EnappWebshopTapaolucPool

### 2.2.2. Java Message Service

*Queuefactory*

Poolname: jms/purchasequeuefactory

Ressource Type: javax.jms.QueueConnectionFactory

Transaction: XATransaction

AdditionalProperty: AddressList = mq://10.29.3.152:7676/jms

*Queue*

JNDI Name: jms/purchasequeue

Physical Name: EnappQueue

Resource Type: javax.jms.Queue

### 2.2.3. Security *Security - Realm*

Configuration: server-config

Realm Name: enappwebshoprealm

Classname: com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm

JAAS Context: jdbcRealm

JNDI Name: jdbc/enappwebshoptapaoluc

Usetable: customer

Usercolumn: username

Passwordcolumn: password

Grouptable: customergroups

Grouptable-Usercolumn: username

Groupcolumn: groupname

Digest algorithm: none

Encryption algorithm: none

**Attention: There is no encryption so do not use your own passwords for tests!**

### 2.2.4. JNDI - Custom Ressources

This setting sets the stage of the application to production.

JNDI Name: javax.faces.PROJECT\_STAGE

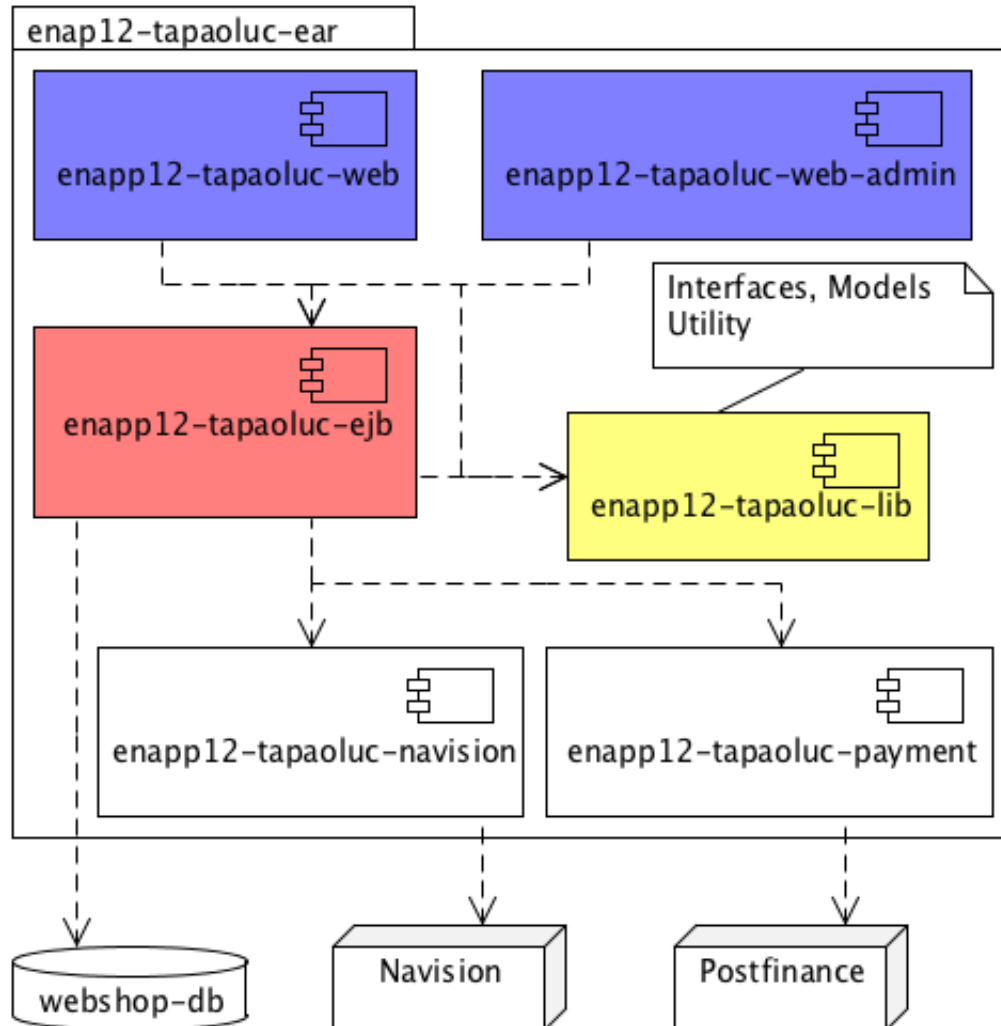
Ressource Type: java.lang.String

Factory Class: org.glassfish.resources.custom.factory.PrimitivesAndStringFactory

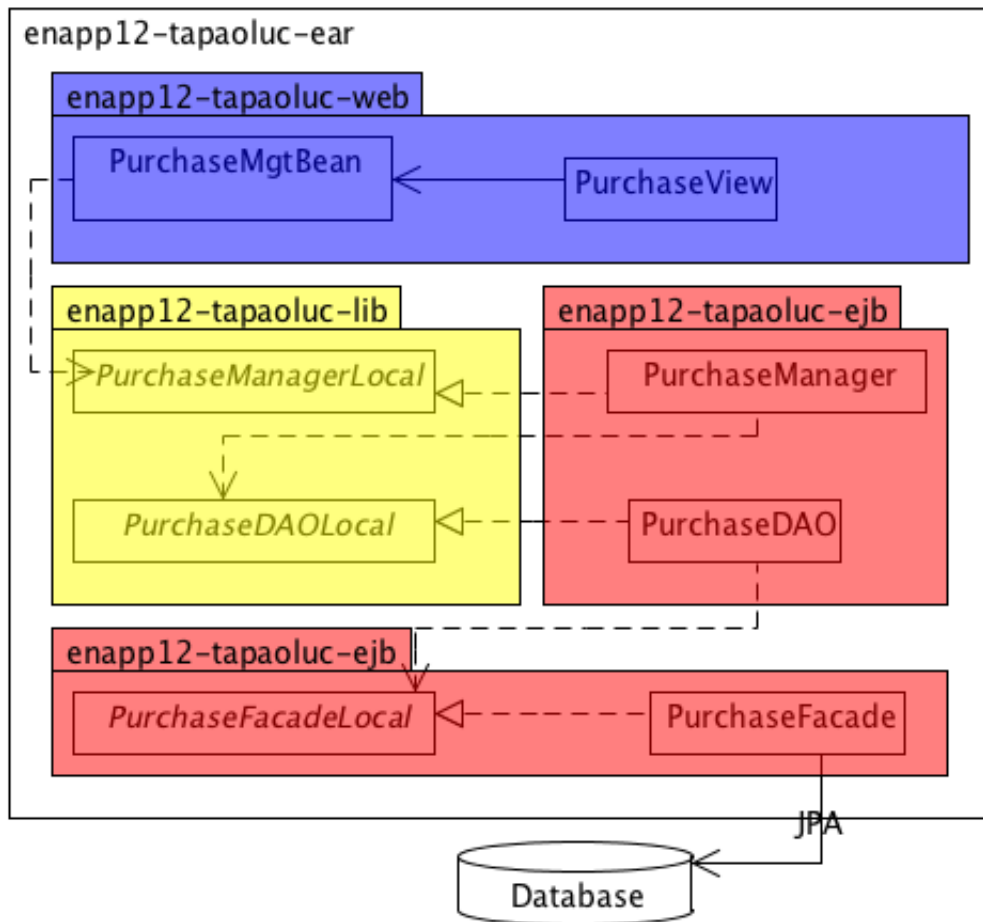
Property: stage = production

### 3. ARCHITECTURE

Below is an overview of the webshop. It is a component based architecture. All accesses to other components are done via Interfaces defined in `enapp12-tapaoluc-lib`. The general idea of this architecture is to avoid any changes in business logic when the datasource changes. For example when the purchase is sent to a JMS queue insted of stored in a relational database, in general there shouldn't be any changes insted of some annotations tellig CDI that another implementation is needed.



Inside the EJB component a "Entity Controller Boundary" pattern is implemented. Below there is a sample from the Purchase.



The JSF Managed Bean accesses the `PurchaseManager` via the Interface `PurchaseManagerLocal`. The `PurchaseManager` has the BusinessLogic implemented and gets hist Data from `PurchaseDAO` via `PurchaseDAOLocal`. The implementations are injected via the Context and Dependency Injection.

In code it looks like this (this is not the actual code but it shows the general idea):

```

/* PurchaseMgmtBean.java (web)*/
@Inject
private PurchaseManagerLocal pml;

public void checkoutPurchase(){
    // collect data entered from user and send it to ejb
    this.pml.checkoutPurchase(purchase);
}

/* PurchaseManager.java (ejb)*/
@Inject
@JMSPurchaseDAO
private PurchaseDAOLocal dao;

@Inject
@PostfinancePayment
private CreditCardPayment payment;

private boolean anythingWentWrong;

public void checkoutPurchase(Purchase purchase){
    // do the payment
    payment.pay();
    // store purchase
    dao.storePurchase(purchase);
    // send feedback if necessary
    if(anythingWentWrong){
        throw new BusinessException("something wrong");
    }
}

/* PurchaseDAO (JMS impl) */
@StatelessBean
@JMSPurchaseDAO
public class JMSPurchaseDataAccess implements PurchaseDAOLocal{}

```

When the Interface to the Purchase service changes for example to a Rest service there is only the Annotation `@JMSPurchaseDAO` changed to `@RestPurchaseDAO` which points to the Rest implementation.

### 3.1. Project Structure

Below the structure of the project is visualized. At the end everything is packed into one EAR (Enterprise Archtive). In the `enapp12-tapaoluc-lib` project in packages `boundary` and `dataaccess` are only interfaces not implementations.

