IMPORT------

Definition.xml

<action-definition id="com.hybris.backoffice.widgets.actions.excel.import"

                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

                 xsi:noNamespaceSchemaLocation="http://www.hybris.com/schema/cockpitng/action-definition.xsd">

    <name>Import CSV</name>

    <description>Import Payment Transaction Entry</description>

    <author>hybris</author>

    <version>0.1</version>

    <actionClassName>com.hybris.backoffice.widgets.actions.excel.ExcelImportAction</actionClassName>

    <inputType>java.util.Map</inputType>

    <iconUri>icons/icon\_action\_import\_excel\_default.png</iconUri>

    <iconDisabledUri>icons/icon\_action\_import\_excel\_disabled.png</iconDisabledUri>

    <iconHoverUri>icons/icon\_action\_import\_excel\_hover.png</iconHoverUri>

    <settings>

        <setting key="viewMode" default-value="iconAndText"/>

    </settings>

</action-definition>

Handler -   
  
**package** com.fabindia.backoffice.paymenttransactionentry;

**import** com.fabindia.backoffice.data.RowData;

**import** com.fabindia.excel.FabIndiaExcelService;

**import** com.google.common.base.Preconditions;

**import** com.hybris.backoffice.excel.imp.wizard.ExcelImportHandler;

**import** com.hybris.backoffice.excel.imp.wizard.ExcelImportWizardForm;

**import** com.hybris.backoffice.excel.jobs.FileContent;

**import** com.hybris.backoffice.widgets.notificationarea.event.NotificationEvent;

**import** com.hybris.cockpitng.config.jaxb.wizard.CustomType;

**import** com.hybris.cockpitng.widgets.configurableflow.FlowActionHandlerAdapter;

**import** de.hybris.platform.core.model.user.CustomerModel;

**import** de.hybris.platform.payment.model.PaymentTransactionEntryModel;

**import** de.hybris.platform.servicelayer.model.ModelService;

**import** de.hybris.platform.servicelayer.search.FlexibleSearchQuery;

**import** de.hybris.platform.servicelayer.search.FlexibleSearchService;

**import** de.hybris.platform.servicelayer.search.SearchResult;

**import** org.apache.commons.collections4.CollectionUtils;

**import** org.apache.commons.lang3.StringUtils;

**import** org.apache.poi.openxml4j.exceptions.InvalidFormatException;

**import** javax.annotation.Resource;

**import** java.io.IOException;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Objects;

**public** **class** PaymentTransactionEntryUploadHandler **extends** ExcelImportHandler

{

    @Resource(name = "readExcelFileServices")

**private** FabIndiaExcelService excelService;

**private** Map<String, String> attributeValueMapping;

**private** FlexibleSearchService flexibleSearchService;

**private** ModelService modelService;

**public** PaymentTransactionEntryUploadHandler(Map<String, String> attributeValueMapping,

            FlexibleSearchService flexibleSearchService, ModelService modelService)

    {

**this**.attributeValueMapping = attributeValueMapping;

**this**.flexibleSearchService = flexibleSearchService;

**this**.modelService = modelService;

    }

**protected** **static** **final** String ORDERID = "orderId";

**protected** **static** **final** String CUSTOMERID = "customerId";

**protected** **static** **final** String PAYMENTPROVIDER = "paymentProvider";

**protected** **static** **final** String TRANSACTIONSTATUS = "transactionStatus";

**protected** **static** **final** String AMOUNT = "amount";

**protected** **static** **final** String TRANSACTIONID = "transactionId";

**protected** **static** **final** String PHONENUMBER = "phoneNumber";

**protected** **static** **final** String REVIEW = "REVIEW";

**protected** **static** **final** String ACCEPTED = "ACCEPTED";

**public** **static** **final** String FETCH\_BY\_CODE = "Select {pk} from {" + PaymentTransactionEntryModel.\_TYPECODE + "} where {"

            + PaymentTransactionEntryModel.CODE + "}=?" + PaymentTransactionEntryModel.CODE;

    @Override

**public** **void** perform(CustomType customType, FlowActionHandlerAdapter adapter, Map<String, String> parameters)

    {

        ExcelImportWizardForm data = (ExcelImportWizardForm) adapter.getWidgetInstanceManager().getModel()

                .getValue("paymentTransactionExcelForm", ExcelImportWizardForm.**class**);

**if** (data == null)

        {

**this**.getNotificationService().notifyUser("excelImport", "excelMissingExcelFormInModel", NotificationEvent.Level.FAILURE,

**new** Object[0]);

        }

**else**

        {

            FileContent excelFile = **this**.toFileContent(data.getExcelFile());

**if** (excelFile == null)

            {

**this**.getNotificationService().notifyUser("excelImport", "excelMissingExcelFile", NotificationEvent.Level.FAILURE,

**new** Object[0]);

            }

**else**

            {

**try**

                {

                    readExcel(excelFile);

                }

**catch** (IOException | InvalidFormatException e)

                {

**throw** **new** RuntimeException(e);

                }

                adapter.done();

            }

        }

    }

**private** **void** readExcel(FileContent excelFile) **throws** IOException, InvalidFormatException

    {

        List<RowData> rowList = excelService.importExcel(excelFile);

**for** (RowData rowData : rowList)

        {

            setTransactionEntryStatusBasedOnCode(rowData);

        }

    }

**private** **void** setTransactionEntryStatusBasedOnCode(RowData rowData)

    {

        Integer transactionValue = Integer.valueOf(getAttributeValueMapping().get(TRANSACTIONID));

        Integer orderValue = Integer.valueOf(getAttributeValueMapping().get(ORDERID));

        Integer paymentProvider = Integer.valueOf(getAttributeValueMapping().get(PAYMENTPROVIDER));

        Integer transactionStatus = Integer.valueOf(getAttributeValueMapping().get(TRANSACTIONSTATUS));

        Integer amount = Integer.valueOf(getAttributeValueMapping().get(AMOUNT));

        Integer customerId = Integer.valueOf(getAttributeValueMapping().get(CUSTOMERID));

        Integer phoneNumber = Integer.valueOf(getAttributeValueMapping().get(PHONENUMBER));

**if** (Objects.nonNull(rowData.getEntryMap()))

        {

            rowData.getEntryMap().forEach((key, value) -> {

                String[] strings = value.split(",");

                String transactionId = strings[transactionValue];

                PaymentTransactionEntryModel paymentTransactionEntryModel = getEntryFromCode(transactionId);

**if** (paymentTransactionEntryModel != null)

                {

                    CustomerModel user = (CustomerModel) paymentTransactionEntryModel.getPaymentTransaction().getOrder().getUser();

**if** (paymentTransactionEntryModel.getPaymentTransaction().getOrder().getCode().equals(strings[orderValue])

                            && user.getUid().equals(strings[customerId])

                            && paymentTransactionEntryModel.getCode().equals(strings[transactionValue])

                            && paymentTransactionEntryModel.getTransactionStatus().equals(strings[transactionStatus])

                            && strings[transactionStatus].equals(REVIEW)

                            && paymentTransactionEntryModel.getPaymentTransaction().getPaymentProvider().equals(strings[paymentProvider])

                            && user.getContactNumber().equals(strings[phoneNumber]) && paymentTransactionEntryModel.getAmount().toString().equals(strings[amount]))

                    {

                        paymentTransactionEntryModel.setTransactionStatus(ACCEPTED);

                        modelService.save(paymentTransactionEntryModel);

                    }

                }

            });

        }

    }

**private** PaymentTransactionEntryModel getEntryFromCode(**final** String code)

    {

        Preconditions.checkArgument(StringUtils.isNotBlank(code), "code is null");

        Map<String, Object> paramsMap = **new** HashMap<>();

        paramsMap.put(PaymentTransactionEntryModel.CODE, code);

        FlexibleSearchQuery query = **new** FlexibleSearchQuery(FETCH\_BY\_CODE, paramsMap);

        SearchResult<PaymentTransactionEntryModel> result = **this**.getFlexibleSearchService().search(query);

**return** CollectionUtils.isNotEmpty(result.getResult()) ? result.getResult().get(0) : null;

    }

**public** FabIndiaExcelService getExcelService()

    {

**return** excelService;

    }

**public** FlexibleSearchService getFlexibleSearchService()

    {

**return** flexibleSearchService;

    }

**public** ModelService getModelService()

    {

**return** modelService;

    }

**public** Map<String, String> getAttributeValueMapping()

    {

**return** attributeValueMapping;

    }

}

CONFIG.xml – PaymentTransactionEntry—Context wizard

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

*<!--*

*Copyright (c) 2019 SAP SE or an SAP affiliate company. All rights reserved.*

*-->*

<config xmlns:na="http://www.hybris.com/cockpitng/config/notifications"

        xmlns:wz="http://www.hybris.com/cockpitng/config/wizard-config"

        xmlns="http://www.hybris.com/cockpit/config"

>

    <context merge-by="module" component="editorareaactions" type="Consignment">

        <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

            <y:group qualifier="common">

                <y:label>actiongroup.common</y:label>

                <y:action action-id="com.fabindia.backoffice.actions.confirmpickupaction" property="currentObject"/>

            </y:group>

        </y:actions>

    </context>

*<!-- Notification Area-->*

    <context component="notification-area">

        <na:notification-area>

            <na:notifications eventType="fabindiabackoffice.confirmpickup.event" level="SUCCESS">

                <na:message>hmc.action.confirmpickup.success</na:message>

            </na:notifications>

        </na:notification-area>

    </context>

    <context component="explorer-tree" merge-by="module">

        <explorer-tree:explorer-tree xmlns:explorer-tree="http://www.hybris.com/cockpitng/config/explorertree">

            <explorer-tree:navigation-node id="hmc\_treenode\_fab">

                <explorer-tree:type-node code="FabFranchiseEnquiry" id="hmc\_typenode\_fabfranchiseenquiry"/>

            </explorer-tree:navigation-node>

        </explorer-tree:explorer-tree>

    </context>

    <context type="FabFranchiseEnquiry" component="listviewactions" merge-by="type">

        <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

            <y:group qualifier="common">

                <y:action action-id="com.hybris.cockpitng.listview.action.export.csv" property="exportColumnsAndData"/>

            </y:group>

        </y:actions>

    </context>

    <context merge-by="type" parent="GenericItem" type="FabFranchiseEnquiry" component="listview">

        <list-view:list-view xmlns:list-view="http://www.hybris.com/cockpitng/component/listView">

            <list-view:column qualifier="fullName"/>

            <list-view:column qualifier="dateOfBirth"/>

            <list-view:column qualifier="email"/>

            <list-view:column qualifier="address"/>

            <list-view:column qualifier="mobile"/>

            <list-view:column qualifier="eduQualifications"/>

            <list-view:column qualifier="profession"/>

            <list-view:column qualifier="firmName"/>

            <list-view:column qualifier="firmNature"/>

            <list-view:column qualifier="experience"/>

            <list-view:column qualifier="turnover"/>

            <list-view:column qualifier="city" spring-bean="nullSafeListCellRenderer"/>

            <list-view:column qualifier="location"/>

            <list-view:column qualifier="state" spring-bean="nullSafeListCellRenderer"/>

            <list-view:column qualifier="investmentBudget"/>

            <list-view:column qualifier="bandsInVicinity"/>

            <list-view:column qualifier="property"/>

            <list-view:column qualifier="carpetArea"/>

            <list-view:column qualifier="frontage"/>

            <list-view:column qualifier="noOfFloors"/>

        </list-view:list-view>

    </context>

    <context merge-by="module" type="FabFranchiseEnquiry" component="editor-area">

        <editorArea:editorArea xmlns:editorArea="http://www.hybris.com/cockpitng/component/editorArea">

            <editorArea:tab name="hmc.properties">

                <editorArea:section name="hmc.section.personal.profile">

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="fullName"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="dateOfBirth"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="email"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="address"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="mobile"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="eduQualifications"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="profession"/>

                </editorArea:section>

                <editorArea:section name="hmc.section.business.profile">

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="firmName"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="firmNature"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="experience"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="turnover"/>

                </editorArea:section>

                <editorArea:section name="hmc.section.proposal.details">

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="city"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="location"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="state"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="investmentBudget"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="bandsInVicinity"/>

                </editorArea:section>

                <editorArea:section name="hmc.section.property.details">

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="property"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="carpetArea"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="frontage"/>

                    <editorArea:attribute xmlns="http://www.hybris.com/cockpitng/component/editorArea" qualifier="noOfFloors"/>

                </editorArea:section>

            </editorArea:tab>

        </editorArea:editorArea>

    </context>

    <context merge-by="module" parent="auto" type="Product" component="editor-area">

        <ea:editorArea xmlns:ea="http://www.hybris.com/cockpitng/component/editorArea">

            <ea:essentials>

                <ea:essentialSection name="hmc.essential" merge-mode="replace">

                    <ea:attribute qualifier="code"/>

                    <ea:attribute qualifier="ecomName"/>

                    <ea:attribute qualifier="catalogVersion"/>

                    <ea:attribute qualifier="approvalStatus"/>

                </ea:essentialSection>

            </ea:essentials>

            <ea:tab name="hmc.tab.product.properties" position="2">

                <ea:section name="hmc.product.descriptions">

                    <ea:panel colspan="2" name="panel.proddescription" rowspan="2">

                        <ea:attribute editor="com.hybris.cockpitng.editor.localized(com.hybris.cockpitng.editor.wysiwyg)" qualifier="additionalDetails"/>

                    </ea:panel>

                </ea:section>

            </ea:tab>

        </ea:editorArea>

    </context>

    <context component="explorer-tree" merge-by="module">

        <explorer-tree:explorer-tree xmlns:explorer-tree="http://www.hybris.com/cockpitng/config/explorertree">

            <explorer-tree:navigation-node id="AsmReport" position="1">

            </explorer-tree:navigation-node>

        </explorer-tree:explorer-tree>

    </context>

    <context type="AsmReport" component="create-wizard">

        <wz:flow id="exportExcel" title="Export excel" show-breadcrumb="false">

            <wz:prepare id="prepare">

                <wz:initialize property="asmReport" type="AsmReport"/>

            </wz:prepare>

            <wz:step id="step1" label="Export"

                     sublabel="my.wizard.sublabel">

                <wz:content>

                    <wz:property-list root="asmReport">

                        <wz:property qualifier="startDate" label="Start Date"/>

                        <wz:property qualifier="endDate" label="End Date"/>

                        <wz:property qualifier="pointsOfServices"/>

                    </wz:property-list>

                </wz:content>

                <wz:navigation id="step1.navigation">

                    <wz:custom handler="orderReportExcelExportHandler" label="Download excel"/>

                    <wz:cancel/>

                </wz:navigation>

            </wz:step>

        </wz:flow>

    </context>

    <context component="notification-area" source="validPointOfService" merge-by="source">

        <na:notification-area>

            <na:notifications eventType="datesAreEmpty" level="FAILURE">

                <na:message>hmc.dates.not.found</na:message>

            </na:notifications>

        </na:notification-area>

    </context>

    <context type="PaymentTransactionEntry" component="listviewactions" merge-by="type">

        <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

            <y:group qualifier="common">

                <y:action action-id="com.fabindia.backoffice.actions.exportPaymentTransactionEntry" position="30">

                </y:action>

            </y:group>

            <y:group qualifier="import" show-group-header="false" show-separator="false" position="40">

                <y:action action-id="com.hybris.backoffice.widgets.actions.excel.import" property="pageable.typeCode"

                         position="40"/>

            </y:group>

        </y:actions>

    </context>

    <context type="StockLevel" component="listviewactions" merge-by="type">

        <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

            <y:group qualifier="common">

                <y:action action-id="com.fabindia.backoffice.actions.exportStockLevel" position="30">

                </y:action>

            </y:group>

        </y:actions>

    </context>

    <context type="Category" component="listviewactions" merge-by="type">

        <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

            <y:group qualifier="common">

                <y:action action-id="com.fabindia.backoffice.actions.exportcategory" position="30">

                </y:action>

            </y:group>

        </y:actions>

    </context>

    <context type="AbstractOrder" component="listviewactions" merge-by="type">

        <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

            <y:group qualifier="common">

                <y:action action-id="com.fabindia.backoffice.actions.exportOrder" position="30">

                </y:action>

                <y:action action-id="com.hybris.cockpitng.listview.action.export.csv" merge-mode="REMOVE"/>

            </y:group>

        </y:actions>

    </context>

    <context type="AbstractOrder" component="listviewactions" merge-by="type">

        <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

            <y:group qualifier="common">

                <y:action action-id="com.fabindia.backoffice.actions.exportOrder" position="30">

                </y:action>

                <y:action action-id="com.hybris.cockpitng.listview.action.export.csv" merge-mode="REMOVE"/>

            </y:group>

        </y:actions>

    </context>

    <context component="explorer-tree" merge-by="module">

        <explorer-tree:explorer-tree xmlns:explorer-tree="http://www.hybris.com/cockpitng/config/explorertree">

            <explorer-tree:navigation-node id="hmc\_treenode\_payment">

                <explorer-tree:type-node code="PaymentTransactionEntry" id="hmc\_typenode\_paymenttransactionentry"/>

            </explorer-tree:navigation-node>

        </explorer-tree:explorer-tree>

    </context>

    <context type="PaymentTransactionEntry" component="excel-import-wizard">

        <wz:flow id="excelImportWizard" title="excel.import.wizard.title" show-breadcrumb="false">

            <wz:prepare id="excelImportWizardPrepare">

                <wz:initialize property="paymentTransactionExcelForm" template-bean="excelImportWizardForm"/>

            </wz:prepare>

            <wz:step id="step1" label="">

                <wz:content>

                    <wz:property qualifier="paymentTransactionExcelForm.excelFile"

                                 type="com.hybris.cockpitng.editor.defaultfileupload.FileUploadResult"

                                 editor="com.hybris.cockpitng.editor.dndfileupload" validate="false">

                        <wz:editor-parameter>

                            <wz:name>accept</wz:name>

                            <wz:value>application/vnd.openxmlformats-officedocument.spreadsheetml.sheet</wz:value>

                        </wz:editor-parameter>

                        <wz:editor-parameter>

                            <wz:name>singleLineMode</wz:name>

                            <wz:value>true</wz:value>

                        </wz:editor-parameter>

                    </wz:property>

                </wz:content>

                <wz:navigation id="step1.navigation">

                    <wz:cancel/>

                    <wz:custom label="excel.import.wizard.button.label" handler="paymentTransactionEntryExcelImportHandler"

                             visible="paymentTransactionExcelForm.excelFile!=null"/>

                </wz:navigation>

            </wz:step>

        </wz:flow>

    </context>

</config>

<alias name="paymentTransactionEntryExcelImportHandler" alias="paymentTransactionEntryExcelImportHandler"/>

[24](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT24)

+Added: <bean id="paymentTransactionEntryExcelImportHandler" class="com.fabindia.backoffice.paymenttransactionentry.PaymentTransactionEntryUploadHandler" parent="abstractExcelImportHandler">

[25](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT25)

+Added: <constructor-arg name="attributeValueMapping">

[26](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT26)

+Added: <map>

[27](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT27)

+Added: <entry key="orderId" value="0"/>

[28](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT28)

+Added: <entry key="customerId" value="1"/>

[29](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT29)

+Added: <entry key="paymentProvider" value="2"/>

[30](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT30)

+Added: <entry key="transactionStatus" value="3"/>

[31](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT31)

+Added: <entry key="amount" value="4"/>

[32](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT32)

+Added: <entry key="transactionId" value="5"/>

[33](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT33)

+Added: <entry key="phoneNumber" value="6"/>

[34](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT34)

+Added: </map>

[35](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT35)

+Added: </constructor-arg>

[36](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT36)

+Added: <constructor-arg name="flexibleSearchService" ref="flexibleSearchService"/>

[37](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT37)

+Added: <constructor-arg name="modelService" ref="modelService"/>

[38](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1475/upload-payment-transaction-entry-through?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-spring.xmlT38)

+Added: </bean>

READ AND WRITE EXCEL---  
  
**package** com.fabindia.excel.impl;

**import** **static** com.fabindia.core.constants.FabindiaCoreConstants.DATE\_FORMAT;

**import** java.io.\*;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.\*;

**import** javax.annotation.Resource;

**import** de.hybris.platform.tx.Transaction;

**import** org.apache.commons.collections.CollectionUtils;

**import** org.apache.commons.lang3.BooleanUtils;

**import** org.apache.commons.lang3.StringUtils;

**import** org.apache.log4j.Logger;

**import** org.apache.poi.openxml4j.exceptions.InvalidFormatException;

**import** org.apache.poi.ss.usermodel.Cell;

**import** org.apache.poi.ss.usermodel.CellType;

**import** org.apache.poi.ss.usermodel.Row;

**import** org.apache.poi.xssf.usermodel.XSSFSheet;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**import** com.fabindia.backoffice.data.RowData;

**import** com.fabindia.core.order.FabOrdersReportService;

**import** com.fabindia.excel.FabIndiaExcelService;

**import** com.hybris.backoffice.excel.jobs.FileContent;

**import** de.hybris.platform.category.model.CategoryModel;

**import** de.hybris.platform.core.model.order.OrderEntryModel;

**import** de.hybris.platform.core.model.product.ProductModel;

**import** de.hybris.platform.core.model.user.UserModel;

**import** de.hybris.platform.ordersplitting.model.ConsignmentEntryModel;

**import** de.hybris.platform.ordersplitting.model.ConsignmentModel;

**import** de.hybris.platform.storelocator.model.PointOfServiceModel;

**public** **class** DefaultFabindiaExcelFileServices **implements** FabIndiaExcelService

{

    @Resource(name = "fabOrderReportService")

    FabOrdersReportService fabOrderReportService;

    Logger logger = Logger.getLogger(DefaultFabindiaExcelFileServices.**class**);

**private** **static** **final** Object[] headers =

    { "Web OrderID", "Store codes", "Store names", "User IDs", "Order Type", "Order Date", "Country", "State", "City",

            "CustomerName", "Shipping Address1", "Shipping Address2", "Shipping Address3", "Shipping Address4", "Shipping PinCode",

            "Billing CustomerName", "Billing Address1", "Billing Address2", "Billing Address3", "Billing Address4", "Billing State",

            "Billing Country", "Billing City", "Billing Pincode", "Item Code", "Item Name", "Category", "Dept", "MRP", "Sale price",

            "Discount Amount", "Order qty", "Total Price", "Amt", "Discount Code", "Invoice No", "Invoice Qty", "Shipping Warehouse",

            "Shipping provider", "Tracking Number", "Return Qty", "Sale status", "Net QTY", "Net Sale" };

**public** List<RowData> readExcel(FileContent excelFile) **throws** IOException, InvalidFormatException

    {

        List<RowData> rowDataList = **new** ArrayList<>();

        File file = File.createTempFile("sizechart", "xlsx");

        OutputStream os = **new** FileOutputStream(file);

        os.write(excelFile.getData());

        os.close();

**try**

        {

            FileInputStream readFile = **new** FileInputStream(file);

            XSSFWorkbook workbook = **new** XSSFWorkbook(file);

            XSSFSheet sheet = workbook.getSheetAt(0);

            Iterator<Row> rowIterator = sheet.iterator();

**while** (rowIterator.hasNext())

            {

                RowData rowData = **new** RowData();

                Map<Integer, String> entryMap = **new** HashMap<>();

                Row row = rowIterator.next();

                Iterator<Cell> cellIterator = row.cellIterator();

**while** (cellIterator.hasNext())

                {

                    Cell cell = cellIterator.next();

**if** (cell.getCellType().equals(CellType.NUMERIC))

                    {

**long** numericCellValue = (**long**) cell.getNumericCellValue();

                        entryMap.put(cell.getColumnIndex(), Long.toString(numericCellValue));

                    }

**else** **if** (cell.getCellType().equals(CellType.STRING))

                    {

                        entryMap.put(cell.getColumnIndex(), String.valueOf(cell.getRichStringCellValue()));

                    }

**else** **if** (cell.getCellType().equals(CellType.BOOLEAN))

                    {

                        entryMap.put(cell.getColumnIndex(), String.valueOf(cell.getBooleanCellValue()));

                    }

**else**

                    {

                        entryMap.put(cell.getColumnIndex(), " ");

                    }

                    rowData.setEntryMap(entryMap);

                }

                rowDataList.add(rowData);

            }

            readFile.close();

            file.deleteOnExit();

        }

**catch** (Exception e)

        {

            logger.error(e);

        }

**return** rowDataList;

    }

    @Override

**public** File writeExcel(Date startDate, Date endDate, List<PointOfServiceModel> pointOfServiceModel, UserModel userModel)

**throws** ParseException

    {

        List<OrderEntryModel> ListOfOrder = getFabOrderReportService().allOrdersForReport(startDate, endDate, pointOfServiceModel,

                userModel);

        XSSFWorkbook workbook = **new** XSSFWorkbook();

        XSSFSheet sheet = workbook.createSheet("Orders");

        TreeMap<Integer, Object[]> data = **new** TreeMap<>();

**int** count = 1;

        data.put(count, headers);

**for** (OrderEntryModel orderEntry : ListOfOrder)

        {

            String shippingWarehouse = StringUtils.EMPTY;

            String trackingNumber = StringUtils.EMPTY;

            String shippingProvider = StringUtils.EMPTY;

            String invoiceNumber = StringUtils.EMPTY;

            count = count + 1;

            String l1 = StringUtils.EMPTY;

            String l2 = StringUtils.EMPTY;

            List<String> category = setCategoryOptional(orderEntry.getProduct());

**if** (CollectionUtils.isNotEmpty(category))

            {

                l1 = category.get(category.size() - 1);

**if** (category.size() > 2)

                {

                    l2 = category.get(category.size() - 2);

                }

            }

**if** (CollectionUtils.isNotEmpty(orderEntry.getConsignmentEntries()))

            {

                Optional<ConsignmentEntryModel> optionalConsignmentEntry = orderEntry.getConsignmentEntries().stream().findAny();

**if** (optionalConsignmentEntry.isPresent())

                {

                    ConsignmentModel cs = optionalConsignmentEntry.get().getConsignment();

                    shippingWarehouse = StringUtils.defaultIfBlank(cs.getWarehouse().getName(), StringUtils.EMPTY);

                    trackingNumber = StringUtils.defaultIfBlank(cs.getTrackingID(), StringUtils.EMPTY);

                    shippingProvider = StringUtils.defaultIfBlank(cs.getCarrier(), StringUtils.EMPTY);

                    invoiceNumber = StringUtils.defaultIfBlank(cs.getInvoiceNumber(), StringUtils.EMPTY);

                }

            }

            data.put(Integer.valueOf(String.valueOf(count)), **new** Object[]

            { orderEntry.getOrder().getGuid(),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getStore().getSubmitOrderProcessCode(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getStore().getName(), StringUtils.EMPTY),

                    orderEntry.getOrder().getUser().getUid(),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentType().getCode(), StringUtils.EMPTY),

**new** SimpleDateFormat(DATE\_FORMAT).format(orderEntry.getOrder().getCreationtime()),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getUser().getDefaultShipmentAddress().getCountry().getName(),

                            StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getUser().getDefaultShipmentAddress().getRegion().getName(),

                            StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getUser().getDefaultShipmentAddress().getTown(),

                            StringUtils.EMPTY),

                    orderEntry.getOrder().getUser().getName(),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getDeliveryAddress().getLine1(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getDeliveryAddress().getLine2(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getOrder().getDeliveryAddress().getLongitude()),

                            StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getDeliveryAddress().getNickName(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getDeliveryAddress().getPostalcode(), StringUtils.EMPTY),

                    orderEntry.getOrder().getUser().getName(),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentAddress().getLine1(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentAddress().getLine2(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getOrder().getPaymentAddress().getLongitude()),

                            StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentAddress().getNickName(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentAddress().getRegion().getName(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentAddress().getCountry().getName(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentAddress().getTown(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getPaymentAddress().getPostalcode(), StringUtils.EMPTY),

                    orderEntry.getProduct().getCode(),

                    StringUtils.defaultIfBlank(orderEntry.getProduct().getEcomName(), StringUtils.EMPTY), l1, l2,

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getBasePrice()), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getTotalPrice()), StringUtils.EMPTY),

                    orderEntry.getDiscountValues().size() > 0 ? String.valueOf(orderEntry.getDiscountValues().get(0).getValue())

                            : StringUtils.EMPTY,

                    String.valueOf(orderEntry.getQuantity()),

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getOrder().getTotalPrice()), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getBasePrice()), StringUtils.EMPTY),

                    orderEntry.getDiscountValues().size() > 0 ? orderEntry.getDiscountValues().get(0).getCode() : StringUtils.EMPTY,

                    StringUtils.defaultIfBlank(invoiceNumber, StringUtils.EMPTY), String.valueOf(orderEntry.getQuantityShipped()),

                    shippingWarehouse, shippingProvider, trackingNumber,

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getQuantityReturned()), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(orderEntry.getOrder().getStatus().getCode(), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getQuantityShipped()), StringUtils.EMPTY),

                    StringUtils.defaultIfBlank(String.valueOf(orderEntry.getTotalPrice()), StringUtils.EMPTY) });

        }

        Set<Integer> keySet = data.keySet();

**int** rowNo = 0;

**for** (Object key : keySet)

        {

            Row row = sheet.createRow(rowNo++);

            Object[] objArr = data.get(key);

**int** cellNo = 0;

**for** (Object obj : objArr)

            {

                Cell cell = row.createCell(cellNo++);

**if** (obj **instanceof** String)

                    cell.setCellValue((String) obj);

**else** **if** (obj **instanceof** Integer)

                    cell.setCellValue((Integer) obj);

            }

        }

        File file = null;

**try**

        {

            file = File.createTempFile(Long.toString(System.currentTimeMillis()), "temp");

            FileOutputStream out = **new** FileOutputStream(file);

            workbook.write(out);

            out.close();

        }

**catch** (Exception e)

        {

            logger.error(e);

        }

**return** file;

    }

**public** List<RowData> importExcel(FileContent excelFile) **throws** IOException, InvalidFormatException

    {

        List<RowData> rowDataList = **new** ArrayList<>();

        File file = File.createTempFile("paymentTransactionEntry", "xlsx");

**final** Transaction tx = Transaction.current();

        tx.begin();

**try** (OutputStream os = **new** FileOutputStream(file))

        {

            os.write(excelFile.getData());

            XSSFWorkbook workbook = **new** XSSFWorkbook(file);

            XSSFSheet sheet = workbook.getSheetAt(0);

            Iterator<Row> rowIterator = sheet.iterator();

            rowIterator.next();

**while** (rowIterator.hasNext())

            {

                RowData rowData = **new** RowData();

                Map<Integer, String> entryMap = **new** HashMap<>();

                Row row = rowIterator.next();

                Iterator<Cell> cellIterator = row.cellIterator();

**while** (cellIterator.hasNext())

                {

                    Cell cell = cellIterator.next();

**if** (cell.getCellType().equals(CellType.NUMERIC))

                    {

**long** numericCellValue = (**long**) cell.getNumericCellValue();

                        entryMap.put(cell.getColumnIndex(), Long.toString(numericCellValue));

                    }

**else** **if** (cell.getCellType().equals(CellType.STRING))

                    {

                        entryMap.put(cell.getColumnIndex(), String.valueOf(cell.getRichStringCellValue()));

                    }

**else** **if** (cell.getCellType().equals(CellType.BOOLEAN))

                    {

                        entryMap.put(cell.getColumnIndex(), String.valueOf(cell.getBooleanCellValue()));

                    }

**else**

                    {

                        entryMap.put(cell.getColumnIndex(), " ");

                    }

                    rowData.setEntryMap(entryMap);

                }

                rowDataList.add(rowData);

            }

            tx.commit();

            file.deleteOnExit();

        }

**catch** (**final** Exception e)

        {

            logger.error("Failed to Import Payment Transaction Entry"+e.getMessage());

            tx.rollback();

        }

**return** rowDataList;

    }

**public** FabOrdersReportService getFabOrderReportService()

    {

**return** fabOrderReportService;

    }

**public** List<String> setCategoryOptional(ProductModel product)

    {

        List<String> target = **new** ArrayList<>();

        Optional<CategoryModel> categoryOptional = product.getSupercategories().stream().filter(**this**::isEComLevelCategory)

                .findAny();

**if** (categoryOptional.isPresent())

        {

            CategoryModel eComCategoryModel = categoryOptional.get();

            target.add(eComCategoryModel.getCode());

            fetchSuperEComCategories(eComCategoryModel, target);

        }

**return** target;

    }

**private** **void** fetchSuperEComCategories(CategoryModel sourceCategoryModel, List<String> target)

    {

        Optional<CategoryModel> categoryOptional = sourceCategoryModel.getSupercategories().stream()

                .filter(**this**::isEComLevelCategory).findAny();

**if** (categoryOptional.isPresent())

        {

            CategoryModel eComCategoryModel = categoryOptional.get();

**if** (StringUtils.isNotEmpty(eComCategoryModel.getName()))

            {

                target.add(eComCategoryModel.getCode());

            }

**if** (CollectionUtils.isNotEmpty(sourceCategoryModel.getSupercategories()))

            {

                fetchSuperEComCategories(eComCategoryModel, target);

            }

        }

    }

**protected** **boolean** isEComLevelCategory(CategoryModel categoryModel)

    {

**return** categoryModel **instanceof** CategoryModel && BooleanUtils.isTrue(categoryModel.isEcomVisible())

                && BooleanUtils.isTrue(categoryModel.isPdpVisible());

    }

}

EXPORT ----  
  
Definition.xml--  
<action-definition id="com.fabindia.backoffice.actions.exportPaymentTransactionEntry"

                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

                 xsi:noNamespaceSchemaLocation="http://www.hybris.com/schema/cockpitng/action-definition.xsd">

    <name>Export CSV</name>

    <description>Export Payment Transaction Entry</description>

    <author>hybris</author>

    <version>0.1</version>

    <actionClassName>com.fabindia.backoffice.actions.ExportPaymentTransactionEntry</actionClassName>

    <inputType>java.util.Map</inputType>

    <iconUri>icons/icon\_action\_export\_csv\_default.png</iconUri>

    <iconDisabledUri>icons/icon\_action\_export\_csv\_disabled.png</iconDisabledUri>

    <iconHoverUri>icons/icon\_action\_export\_csv\_hover.png</iconHoverUri>

    <settings>

        <setting key="viewMode" default-value="iconAndText"/>

    </settings>

</action-definition>

Handler -   
  
**package** com.fabindia.backoffice.actions;

**import** com.hybris.cockpitng.actions.ActionContext;

**import** com.hybris.cockpitng.actions.ActionResult;

**import** com.hybris.cockpitng.actions.CockpitAction;

**import** de.hybris.platform.core.model.user.CustomerModel;

**import** de.hybris.platform.payment.model.PaymentTransactionEntryModel;

**import** de.hybris.platform.servicelayer.search.FlexibleSearchQuery;

**import** de.hybris.platform.servicelayer.search.FlexibleSearchService;

**import** de.hybris.platform.servicelayer.search.SearchResult;

**import** org.apache.commons.lang3.StringUtils;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.zkoss.zul.Filedownload;

**import** javax.annotation.Resource;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** **static** org.apache.commons.lang3.StringUtils.EMPTY;

**public** **class** ExportPaymentTransactionEntry **implements** CockpitAction<Map, Object>

{

**protected** **static** **final** String CSV\_HEADERS = "Order Id,Customer Id,Payment Provider,TransactionStatus,Amount,TransactionID,Phone Number\n";

**public** **static** **final** String FIND\_PAYMENT\_TRANSACTIONS = "SELECT {" + PaymentTransactionEntryModel.PK + "} " + "FROM {" + PaymentTransactionEntryModel.\_TYPECODE

+ "} where {" + PaymentTransactionEntryModel.TRANSACTIONSTATUS+"}=?"+PaymentTransactionEntryModel.TRANSACTIONSTATUS+"";

**protected** **static** **final** String REVIEW = "REVIEW";

**private** **static** String COMMA = ",";

**private** **static** **final** Logger LOG = LoggerFactory.getLogger(ExportCategory.**class**);

@Resource

**private** FlexibleSearchService flexibleSearchService;

@Override

**public** ActionResult<Object> perform(**final** ActionContext<Map> actionContext)

{

**final** Map<String, Object> params = **new** HashMap<>(1);

params.put(PaymentTransactionEntryModel.TRANSACTIONSTATUS, REVIEW);

**final** FlexibleSearchQuery query = **new** FlexibleSearchQuery(FIND\_PAYMENT\_TRANSACTIONS);

query.addQueryParameters(params);

**final** SearchResult<PaymentTransactionEntryModel> result = getFlexibleSearchService().search(query);

**final** List<PaymentTransactionEntryModel> model = result.getResult();

**final** String csvContent = createCsv(model);

writeBinaryResponse(csvContent);

**return** **new** ActionResult("success");

}

**protected** **void** createCsvContent(**final** StringBuilder stringBuilder, **final** List<PaymentTransactionEntryModel> paymentTransactionEntryModels)

{

**for** (**final** PaymentTransactionEntryModel paymentTransactionEntryModel : paymentTransactionEntryModels)

{

buildCsv(stringBuilder, paymentTransactionEntryModel);

}

}

**protected** **void** buildCsv(**final** StringBuilder stringBuilder, **final** PaymentTransactionEntryModel paymentTransactionEntryModel)

{

**final** CustomerModel customer = (CustomerModel) paymentTransactionEntryModel.getPaymentTransaction().getOrder().getUser();

stringBuilder.append(StringUtils.defaultIfBlank(paymentTransactionEntryModel.getPaymentTransaction().getOrder().getCode(), EMPTY));

stringBuilder.append(COMMA);

stringBuilder.append(StringUtils.defaultIfBlank(paymentTransactionEntryModel.getPaymentTransaction().getOrder().getUser().getUid(), EMPTY));

stringBuilder.append(COMMA);

stringBuilder.append(StringUtils.defaultIfBlank(paymentTransactionEntryModel.getPaymentTransaction().getPaymentProvider(), EMPTY));

stringBuilder.append(COMMA);

stringBuilder.append(StringUtils.defaultIfBlank(paymentTransactionEntryModel.getTransactionStatus(), EMPTY));

stringBuilder.append(COMMA);

stringBuilder.append(StringUtils.defaultIfBlank(String.valueOf(paymentTransactionEntryModel.getAmount()), EMPTY));

stringBuilder.append(COMMA);

stringBuilder.append(StringUtils.defaultIfBlank(paymentTransactionEntryModel.getCode(), EMPTY));

stringBuilder.append(COMMA);

stringBuilder.append(StringUtils.defaultIfBlank(customer.getContactNumber(), EMPTY));

stringBuilder.append('\n');

}

**protected** **void** writeBinaryResponse(**final** String csvContent)

{

Filedownload.save(csvContent, "text/xls", "PaymentTransactionEntry.xls");

LOG.info("Payment Transaction Entry Export Excel generated");

}

**protected** String createCsv(**final** List<PaymentTransactionEntryModel> paymentTransactionEntryModel)

{

**final** StringBuilder stringBuilder = **new** StringBuilder();

stringBuilder.append(CSV\_HEADERS);

createCsvContent(stringBuilder, paymentTransactionEntryModel);

**return** stringBuilder.toString();

}

@Override

**public** **boolean** canPerform(**final** ActionContext<Map> ctx)

{

**return** CockpitAction.**super**.canPerform(ctx);

}

@Override

**public** **boolean** needsConfirmation(**final** ActionContext<Map> ctx)

{

**return** CockpitAction.**super**.needsConfirmation(ctx);

}

@Override

**public** String getConfirmationMessage(**final** ActionContext<Map> ctx)

{

**return** CockpitAction.**super**.getConfirmationMessage(ctx);

}

**public** FlexibleSearchService getFlexibleSearchService()

{

**return** flexibleSearchService;

}

}

Config.xml  
  
 <context type="PaymentTransactionEntry" component="listviewactions" merge-by="type">

[163](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT163)

+Added: <y:actions xmlns:y="http://www.hybris.com/cockpit/config/hybris">

[164](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT164)

+Added: <y:group qualifier="common">

[165](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT165)

+Added: <y:action action-id="com.fabindia.backoffice.actions.exportPaymentTransactionEntry" position="30">

[166](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT166)

+Added: </y:action>

[167](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT167)

+Added: </y:group>

[168](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT168)

+Added: </y:actions>

[169](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT169)

+Added: </context>

[170](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT170)

+Added:

[162171](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlF162T171)

<context component="explorer-tree" merge-by="module">

[207](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT207)

+Added: <explorer-tree:explorer-tree xmlns:explorer-tree="http://www.hybris.com/cockpitng/config/explorertree">

[208](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT208)

+Added: <explorer-tree:navigation-node id="hmc\_treenode\_payment">

[209](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT209)

+Added: <explorer-tree:type-node code="PaymentTransactionEntry" id="hmc\_typenode\_paymenttransactionentry"/>

[210](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT210)

+Added: </explorer-tree:navigation-node>

[211](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT211)

+Added: </explorer-tree:explorer-tree>

[212](https://bitbucket.org/fabindiab2c/fabcommerce/pull-requests/1434/export-review-payment-transaction-entry?link_source=email#Lcore-customize/hybris/bin/custom/fabindia/fabindiabackoffice/resources/fabindiabackoffice-backoffice-config.xmlT212)

+Added: </context>