? Sistemul

- Managerul de magazin comandă marfă
- Managerul de magazin creează produsele
- Managerul de magazin recepționează factura
- Managerul reconciliază extrasul bancar
- Vânzătorul vinde marfa
- Import data from legacy app
- Programul de fidelizare Partener Colibri

Managerul de magazin comand? marf?

Data mapping

The suppliers for a product are distinct for each facility. There is a party for each Facility. Suppliers are stored in mantle.product.ProductPrice, using the following fields:

- **productPriceId**: supplierId+"_"+productId+"_"+facilityPartyId
- **productId**: productId
- **vendorPartyId**: supplierId
- **customerPartyId**: facilityPartyId
- **priceTypeEnumId**: "PptCurrent"
- **pricePurposeEnumId**: "PppPurchase"
- **price**: price
- **priceUomId**: "RON"
- **preferredOrderEnumId**: "SpoMain" or "SpoAlternate"
- **otherPartyItemId**: supplierProductId
- **otherPartyItemName**: supplierProductName

Product Pareto category is stored as:

```
mantle.product.category.ProductCategoryMember
```

- **productCategoryId**: facilityPartyId+paretoCategory (eg.: L2A)
- **productId**: productId

```
mantle.product.category.ProductCategory
```

- **productCategoryId**: facilityPartyId+paretoCategory (eg.: L2A)
- **productCategoryTypeEnumId**: PctPareto
- **categoryName**: "Colibri Pareto A"
- **ownerPartyId**: L2

Products that should be kept in stock and replenished when sold are stored as:

```
mantle.product.Product
```

- **requireInventory**: Y or null

```
mantle.facility.ProductFacility:
```

- **productId**: productId
- **facilityId**: facilityId
- **minimumStock**: minimumStock

Product ordering requirements are stored as:

```
mantle.request.requirement.Requirement
```

- **requirementTypeEnumId**: RqTpInventory
- **statusId**: statusId
- **facilityId**: facilityId ?: ec.user.getPreference('FacilityActive') ?: ec.user.getPreference('FacilityGeneralDefault')
- **productId**: productId
- **quantity**: quantity
- **description**: supplier.organizationName

Screen outline

Necesar

- search widgets
- table
  - cod
  - denumire
  - UM
  - pareto
  - ultimul preț achiziție fără TVA
  - preț vânzare
  - furnizori
  - stocAchCuTVA (hidden)
  - DIO(zile epuizare stoc) (hidden)
  - stocAchCuTVA * DIO (hidden)
  - stocMinim
  - stoc L1
  - stoc L2
  - comanda recomandată L2
  - comenzi furnizor L2
  - adauga
- refresh button

Aproba

- search widgets
- Comanda
- Delete
- table
  - Cod
  - Denumire
  - UM
  - ULPfTVA
  - preț vânzare
  - Furnizori
  - Necesar
- Comanda popup
  - furnizor
  - trimite pe

Comenzi

- search widgets
- Delete
- table
  - Cod
  - Denumire
  - UM
  - Furnizor
  - Comandat

Screen data mapping

Necesar

Table	**Products** WHERE Product.ACTIV_FIELD = true AND Product.HIDE_WHEN_ORDERING_FIELD = false **mantle.product.ProductPrices** WHERE requireInventory = Y/null AND customerPartyId = this gestiune partyId(L2) AND priceTypeEnumId = PptCurrent AND pricePurposeEnumId = PppPurchase
cod	Product.BARCODE_FIELD
denumire	Product.NAME_FIELD
UM	Product.UOM_FIELD
Pareto	mantle.product.category.ProductCategoryMember.productCategoryId

ultimul preț achiziție fără TVA	Product.LAST_BUYING_PRICE_FIELD
preț vânzare	Product.PRICE_FIELD
furnizori	**Organization.organizationName** from ProductPrice.vendorPartyId WHERE preferredOrderEnumId = SpoMain
stocAchCuTVA	product.stocLX * lastBuyingPriceWithVAT
DIO	stocAchCuTVA / dailyCogs dailyCogs = (Product.getTotalSalesPeUltimulAn - Product.getTotalProfitPeUltimulAn) / 250(zile lucratoare in an)
stocMinim	PF.minimumStock ProductFacility PF where PF.facilityId = gest.id AND PF.productId = product.id
For each gestiune	
stoc gest.getImportName	p.stoc(gest)
comanda recomandată L2	PF.minimumStock-p.stoc(gest) ?. p.recommendedOrder(gest)
comenzi furnizor L2	SUM(R.quantity) mantle.request.requirement.Requirements R WHERE R.productId = product.id
adauga	new Requirement with statusId = RqmtStCreated

Aproba

Table	**Requirements R** WHERE R.statusId = RqmtStCreated AND R.facilityId = L2 **JOIN Product P** **JOIN ProductPrice PP**
Cod	P.BARCODE_FIELD
Denumire	P.NAME_FIELD
UM	P.UOM_FIELD
ULPfTVA	P.LAST_BUYING_PRICE_FIELD
preț vânzare	P.PRICE_FIELD
Furnizori	**Organization.organizationNames** from PP.vendorPartyId ORDER BY PP.preferredOrderEnumId, PP.price
Necesar	R.quantity

Comanda popup	**Selected rows R**
furnizor(**S**)	Auto select most common supplier from selected rows
trimite pe	Update on Supplier change Printeaza, ${PartyContactMech PCM WHERE PCM.contactMechPurposeId = 'PhoneShippingOrigin'}, IF supplier is internal organization THEN Transfera

Comenzi

Table	**Requirements R** WHERE R.statusId = RqmtStOrdered AND R.facilityId = L2 **JOIN Product P**
Cod	P.BARCODE_FIELD
Denumire	P.NAME_FIELD
UM	P.UOM_FIELD
Furnizor	R.description
Comandat	R.quantity

Managerul de magazin creeaz? produsele

Data mapping

Prices are stored in mantle.product.ProductPrice, using the following fields:

- **productPriceId**: autoincrement
- **productId**: productId
- **pricePurposeEnumId**: "PppPurchase"
- **priceUomId**: "RON"
- **priceTypeEnumId**: "PptList" or "PptCurrent"
- **price**: price

Data statements

- Regular price is **ProductPrice** of type **PptList.**
- Promotional price is **ProductPrice** of type **PptCurrent.**
- Quantity promotion is **ProductPrice** of type **PptCurrent** with extra field: **minQuantity.**
- Customer specific pricing is **ProductPrice** of type **PptCurrent** with extra field: **customerPartyId.**
- Customer specific quantity promotion is **ProductPrice** of type **PptCurrent** with extra fields: **minQuantity, customerPartyId.**

Managerul de magazin recep? ioneaz? factura

Când **Managerul de magazin** inregistrează o Factură de intrare si produsul este Marfa, **Sistemul** actualizează prețul furnizorului și îl setează ca furnizor principal pentru marfa recepționată.
Când **Managerul de magazin** înregistrează o linie de intrare și cantitatea e mai mare ca 0, **Sistemul** șterge comenzile puse pentru produsul respectiv.

Managerul receptioneaza eFactura

Povestea

Sistemul descarca facturile primite de pe ANAF. **Managerul** incarca eFacturile de pe o anumita perioada, apoi selecteaza o factura care nu a fost receptionata. **Sistemul** gaseste partenerul dupa CUI si il selecteaza. **Managerul** face receptia cu unul din modurile:

1. **Receptioneaza**. **Sistemul** deschide fereastra Manager -> Receptii si precompleteaza campurile cu datele de pe factura, apoi deschide dialogul Adauga si precompleteaza campurile cu datele de la prima linie de pe factura. Dupa ce **Managerul** salveaza linia, **Sistemul** actualizeaza codul furnizorului pentru produsul respectiv, marcheaza linia de pe eFactura ca receptionata si continua cu urmatoarele linii de pe factura. Dupa ce ultima linie a fost receptionata, **Sistemul** marcheaza eFactura ca receptionata.
2. **Fara linii**. **Sistemul** deschide fereastra Urmarire Parteneri, selecteaza partenerul de pe factura si modul CUMPARARE, apoi deschide Adauga Document si precompleteaza campurile. Dupa ce **Managerul** salveaza factura, **Sistemul** marcheaza eFactura ca receptionata.
3. **Doar marcheaza**. **Sistemul** marcheza eFactura ca receptionata.

Data mapping

A received ANAF eInvoice is stored as:

moqui.service.message.SystemMessage

- **systemMessageId:** AnafReceivedMessage.id
- **systemMessageTypeId**: ANAFReceivedInvoice
- **statusId**: SmsgReceived

- **isOutgoing**: N
- **initDate**: now
- **processedDate**: confirmed date
- **messageText**: rawXml
- **senderId**: AccountingSupplierParty.Party.PartyTaxScheme.CompanyID
- **receiverId**: AccountingCustomerParty.Party.PartyTaxScheme.CompanyID
- **messageId**: cbc:ID(invoice number)
- **messageDate**: IssueDate
- **docType**: "Invoice" or "CreditNote"
- **docSubType**: InvoiceTypeCode or CreditNoteTypeCode
- **docControl**: LegalMonetaryTotal.TaxInclusiveAmount.value

moqui.service.message.SystemMessageType

- **systemMessageTypeId**: ANAFReceivedInvoice
- **description**: "Invoice received through a message from ANAF eFactura"

mantle.account.invoice.InvoiceSystemMessage

- **invoiceId**: AccountingDocument.id
- **systemMessageId**: AnafReceivedMessage.id

And the invoice lines are split into one message each:

moqui.service.message.SystemMessage

- **systemMessageId:** AnafReceivedMessage.id+"_"+InvoiceLine.id
- **systemMessageTypeId**: ANAFReceivedInvoiceLine
- **statusId**: SmsgReceived
- **isOutgoing**: N
- **initDate**: now
- **processedDate**: confirmed date
- **parentMessageId**: AnafReceivedMessage.id
- **messageText**: InvoiceLineXml

moqui.service.message.SystemMessageType

- **systemMessageTypeId**: ANAFReceivedInvoiceLine
- **description**: "Invoice line split from an ANAF eInvoice"

Data statements

- A received(posted in the books) ANAF eInvoice is **SystemMessage** with **statusId** =SmsgConfirmed.
- A received ANAF eInvoice line is a **SystemMessage** with **statusId**=SmsgConfirmed

Service data mapping

Managerul incarca eFacturile de pe o anumita perioada

Managerul interogheaza **Moqui** pe o perioada intre from si to.

Moqui interogheaza **cloud-anaf-connector** la /invoices/search/between (from, to). **Cloud-anaf-connector** returneaza List<ReceivedInvoice>. Pentru
fiecare ReceivedInvoice.id=systemMessageId care nu exista **Moqui** creeaza un SystemMessage. **Moqui** consuma SystemMessages noi.

Moqui returneaza **Managerului** efacturile intre from si to.

find#AnafInvoices

SystemMessage SM
JOIN InvoiceSystemMessage ISM on systemMessageId
WHERE SM.systemMessageTypeId=ANAFReceivedInvoice AND SM.messageDate between(from, to)

ReceivedMessage RM
WHERE RM.creationDate between(from, to)

IN

- **start**: start
- **end**: end

OUT

- **id**: SM.systemMessageId ?: RM.id
- **senderId**: SM.senderId ?: RM.taxId
- **senderName**: AccountingSupplierParty.Party.PartyLegalEntity.RegistrationName
- **issueDate**: SM.messageDate :? RM.creationDate
- **invoiceNumber**: SM.messageId
- **invoiceTotal**: LegalMonetaryTotal.TaxInclusiveAmount.value
- **taxTotal**: TaxTotal.TaxAmount.value
- **taxExclusiveAmount**: LegalMonetaryTotal.TaxExclusiveAmount.value
- **invoiceId**: ISM.invoiceId
- **statusId**: SM.statusId
- **rawXml**: messageText
- **messageType**: "AnafRecMsgBillReceived" ?: RM.statusId
- **messageTypeL10n**: ec.l10n.localize("AnafRecMsgBillReceived") ?: ec.l10n.localize(RM.statusId)
- **details**: RM.details

find#AnafInvoiceLines

SystemMessage SML WHERE SML.systemMessageTypeId=ANAFReceivedInvoiceLine AND SML.parentMessageId=systemMessageId

IN

- **systemMessageId**: systemMessageId

OUT

- **id**: SML.systemMessageId
- **lineId**: ID
- **itemId**: Item.SellersItemIdentification.ID
- **name**: Item.Name + Item.Description
- **price**: Price.PriceAmount
- **priceCurrency**: Price.PriceAmount(currencyID)
- **quantity**: InvoicedQuantity or CreditedQuantity
- **uom**: UNECERec20ToDisplay(InvoicedQuantity(unitCode) or CreditedQuantity(unitCode))
- **total**: LineExtensionAmount
- **totalCurrency**: LineExtensionAmount(currencyID)
- **statusId**: SML.statusId

Screen data mapping

Table	find#anafInvoices
Id	id
Cif emitent	senderId
Nume emitent	senderName
Tip mesaj	messageTypeL10n
Data	issueDate
Mesaj	details
Numar	invoiceNumber
ValCuTVA	invoiceTotal
Id factura	invoiceId
Receptionat	**true** if statusId=SmsgConfirmed else **false**

Receptie popup	selected AnafInvoice AI FILTER AI.messageType = "AnafRecMsgBillReceived"
Partener	Partner P WHERE P.codFiscal=AI.senderId
Factura	AI.invoiceNumber
Data	AI.issueDate

ValFaraTVA	AI.taxExclusiveAmount
ValTVA	AI.taxTotal
ValCuTVA	AI.invoiceTotal
lines summary	foreach find#AnafInvoiceLines IL IL.lineId IL.name IL.price IL.priceCurrency X IL.quantity uom = IL.total IL.totalCurrency

Managerul reconciliaz? extrasul bancar

Data mapping

foreach line WHERE line = debit

- **IF** line starts with ignore case 'COMIS' **THEN** doc.gestiune=gestiuneId | Partner='RAIFF gestiune.name.upper' | TipDoc.PLATA | doc=CARD | docNr=NC | date=closingDate | doc.name=COM | doc.total=sum(lines) | banca=contBancarId
- **ELSE IF** line.partner = 'LINIC SRL' **THEN** doc.gestiune=L1 | Partner='RAIFF gestiune.name.upper' | TipDoc.PLATA | doc='ORDIN PLATA' | docNr=OP | date=line.date | doc.name=line.description | doc.total=line.total | banca=contBancarId
- **ELSE** Partner=line.partner | TipDoc.PLATA | doc='ORDIN PLATA' | date=line.date | doc.total=line.total | banca=contBancarId

foreach line WHERE line = credit

- **IF** line contains 'Depunere numerar' **THEN** doc.gestiune=gestiuneId | Partner='RAIFF gestiune.name.upper' | TipDoc.INCASARE | doc='CHITANTA' | date=line.date | doc.name=INC | doc.total=line.total | banca=contBancarId
- **ELSE IF** line starts with 'AK' AND contains 'POS' **THEN** doc.gestiune=gestiuneId | Partner='CARD INCASARE' | TipDoc.INCASARE | doc='CARD' | docNr='NC' | date=line.date | doc.name='INC' | doc.total=line.total | banca=contBancarId **OR** doc.gestiune=gestiuneId | Partner not 'CARD INCASARE' or 'RAIFF...' | TipDoc.INCASARE | doc='CARD' | date=line.date | doc.total=line.total | banca=contBancarId

Vânz?torul vinde marfa

Cand **Vanzatorul** întregistrează o linie de vânzare în interfața nouă și prețul din Moqui este diferit, **Sistemul** adaugă diferența de preț ca și o linie nouă de discount.

Cand **Vanzatorul** întregistrează o linie de vânzare în interfața nouă **și** închide bonul **și** furnizorul principal al produsului este Dunca **și** stocul intră în negativ, **Sistemul** trimite comandă la furnizor pentru produsul respectiv.

Data statements

A POS sale is an OrderHeader with **salesChannelEnumId = POS** and Customer = **_NA_** and **CarrierShipmentMethod.shipmentMethodEnumId = ShMthPickUp**.

A POS cash payment is represented by an **Invoice** of type **InvoiceSimplified** with an attached payment of type **PtInvoicePayment** and **paymentInstrumentEnumId** = PiCash OR PiCod and **toPaymentMethodId** is a PaymentMethod with **paymentMethodTypeEnumId** = PmtCash and **ownerPartyId** = facility/store party id.

A POS card payment is represented by an **Invoice** of type **InvoiceSimplified** with an attached payment of type **PtInvoicePayment** and **paymentInstrumentEnumId** = PiDebitCard and **toPaymentMethodId** is a PaymentMethod with **paymentMethodTypeEnumId** = PmtBankAccount and **ownerPartyId** = facility/store party id, and an attached BankAccount(subtype of PaymentMethod).

NOTE: we use BankAccount for pos card payments because the money goes directly to our BankAccount and we don't store card details. The CreditCard payment method subtype with the type PmtCreditCard is only used for online card payments, where we need the card details.

```
Scenario A: Customer pays an invoice using store credit
  paymentTypeEnumId    = PtInvoicePayment
  paymentInstrumentEnumId = PiFinancialAccount
  paymentMethodId      → PaymentMethod (PmtFinancialAccount)
                              └─ finAccountId → FinancialAccount (CustomerCredit)
  Payment.finAccountId → same FinancialAccount
  finAccountAuthId     → FinancialAccountAuth (holds the funds)
  finAccountTransId    → FinancialAccountTrans (the debit from the account)

Scenario B: Loading money onto a gift card (funding the FinancialAccount)
  paymentTypeEnumId    = PtFinancialAccount
```

```
paymentInstrumentEnumId = PiCreditCard  (how the customer pays to load it)
paymentMethodId      → PaymentMethod (PmtCreditCard)
Payment.finAccountId → FinancialAccount (GiftCard) being loaded
```

Import data from legacy app

Import Partners

Data mapping

Company company;
private Delegat delegat;
Integer termenPlata;
List<Document> documents;
Boolean platitorTva;
Boolean tvaLaIncasare;
LocalDate dataInceputTvaInc;
LocalDate dataSfarsitTvaInc;
Boolean splitTva;
LocalDate dataInceputSplitTVA;
LocalDate dataAnulareSplitTVA;
FidelityCard fidelityCard;
Boolean inactiv;
Set<PartnerGrupaInteresMapping> grupeInteres = new HashSet();
boolean notifyAppointment = false;

Add Customer and Supplier role

```
mantle.party.Party
partyId = id
  isPerson = isEmpty(legacyP.getCodFiscal())
partyTypeEnumId = isPerson ? "PtyPerson" : "PtyOrganization"
disabled = !activ

mantle.party.Person
firstName = name.remove("^A - ").split(" ")[1+" "+..i]
lastName = name.remove("^A - ").split(" ")[0]
nickname = name

mantle.party.Organization
organizationName = name
```

```
mantle.party.PartyIdentification
partyIdTypeEnumId = "PtidTaxId"
idValue = codFiscal

mantle.party.PartyIdentification
partyIdTypeEnumId = "PtidTradeReg"
idValue = regCom

mantle.party.contact.ContactMech
contactMechId = legacyId + "_PHONE"
contactMechTypeEnumId = "CmtTelecomNumber"
mantle.party.contact.TelecomNumber
contactNumber = phone
mantle.party.contact.PartyContactMech
contactMechPurposeId = "PhonePrimary"
fromDate = y2000

mantle.party.contact.ContactMech
contactMechId = legacyId + "_EMAIL"
contactMechTypeEnumId = "CmtEmailAddress"
infoString = email
mantle.party.contact.PartyContactMech
contactMechPurposeId = "EmailPrimary"
fromDate = y2000

mantle.party.contact.ContactMech
contactMechId = legacyId + "_ADDR"
contactMechTypeEnumId = "CmtPostalAddress"
mantle.party.contact.PostalAddress
countryGeoId = legacyAddr.getCountry()
countyGeoId = legacyAddr.getJudet()
city = legacyAddr.getOras()
address1 = legacyAddr.getStrada()
postalCode = legacyAddr.getNr()
mantle.party.contact.PartyContactMech
contactMechPurposeId = "PostalPrimary"
fromDate = y2000

mantle.party.contact.ContactMech
contactMechId = legacyId + "_DELIV_ADDR"
```

```
contactMechTypeEnumId = "CmtPostalAddress"
mantle.party.contact.PostalAddress
address1 = deliveryAddress
directions = indicatii
mantle.party.contact.PartyContactMech
contactMechPurposeId = "PostalShippingDest"
fromDate = y2000

mantle.account.method.PaymentMethod
paymentMethodId = legacyId + "_BANK"
paymentMethodTypeEnumId = "PmtBankAccount"
ownerPartyId = legacyId
mantle.account.method.BankAccount
bankName = banca
accountNumber = iban
```

Programul de fidelizare Partener Colibri

Data mapping

A Colibri Partner is:

```
mantle.party.PartyRole
partyId = partner.id
roleTypeId = "Affiliate"

mantle.party.PartyIdentification
partyId = partner.id
partyIdTypeEnumId = "PtidAffiliateId"
idValue = partnerCode

mantle.party.contact.ContactMech
contactMechId = partner.id + "_PHONE"
contactMechTypeEnumId = "CmtTelecomNumber"
mantle.party.contact.TelecomNumber
contactMechId = contactMechId
contactNumber = phone
mantle.party.contact.PartyContactMech
partyId = partner.id
contactMechId = contactMechId
contactMechPurposeId = "PhonePrimary"
fromDate = now

mantle.account.financial.FinancialAccount
finAccountId = partner.id
finAccountTypeId = "ServiceCredit"
statusId = "FaActive"
finAccountName = "Partener Colibri"
organizationPartyId = L2
ownerPartyId = partner.id
```

```
mantle.party.agreement.AgreementParty
agreementId = "PartenerColibri"
partyId = partner.id
roleTypeId = "Affiliate"

mantle.party.agreement.Agreement
agreementId = "PartenerColibri"
agreementTypeEnumId = "AgrCommission"
organizationPartyId = L2
organizationRoleTypeId = "OrgInternal"

mantle.party.agreement.AgreementTerm
agreementTermId = "PartenerColibri_1..10"
agreementId = "PartenerColibri"
termTypeEnumId = "TtCommission"
termNumber = 50,150,300,500,750,1500,2500,3500,6000,10000
minQuantity = 500,5000,10000,15000,20000,30000,40000,50000,75000,100000
```

A received payment from a Colibri partner is:

```
mantle.account.payment.Payment
paymentId = legacyIncasare.id
paymentTypeEnumId = "PtInvoicePayment"
fromPartyId = legacyIncasare.partner.id
toPartyId = L2
statusId = "PmntDelivered"
effectiveDate = legacyIncasare.dataDoc
amount = legacyIncasare.getTotal()

mantle.account.payment.PaymentParty
paymentId = paymentId
partyId = affiliatePartnerId
roleTypeId = "Affiliate"
```

When a new threshold is reached the customer credit is replenished like this:

```
mantle.account.financial.FinancialAccountTrans
finAccountTransId = "PC${partner.id}_1..10"
finAccountTransTypeEnumId = "FattDeposit"
```

```
finAccountId = partner.id
fromPartyId = L2
toPartyId = partnerAffiliate.id
transactionDate = now
amount = termNumber - previous termNumber
```

Service mapping

```
WHEN
foreach incaseazaDocsOrPartner().paidDocs AS legacyIncasare
closeBonCasa().get(InvocationResult.ACCT_DOC_KEY) AS legacyIncasare
closeFacturaBCAviz().get(InvocationResult.CHITANTA_KEY) AS legacyIncasare
THEN
createPayments()

seca: checkPartnerAgreements after createPayments()
affiliatePartyId = PaymentParty.partyId WHERE PaymentParty.roleTypeId = "Affiliate"
if null affiliatePartyId = Payment.fromPartyId
agreementParty = AgreementParty WHERE agreementId = "PartenerColibri" AND roleTypeId =
"Affiliate" AND partyId = affiliatePartyId
if agreementParty is null then EXIT
lastThresholdReached = FinancialAccountTrans WHERE toPartyId = affiliatePartyId AND
finAccountTransTypeEnumId = "FattDeposit" AND fromPartyId = "L2" AND finAccountId =
affiliatePartyId ORDER BY finAccountTransId.split("_")[1].toInt GET LARGEST
nextThreshold = AgreementTerm WHERE agreementTermId = "PartenerColibri_${lastThresholdIndex +
1}" ?: "PartenerColibri_1"
lastReachedTerm = AgreementTerm WHERE agreementTermId =
"PartenerColibri_${lastThresholdIndex}" ?: null
if nextThreshold is null then EXIT
if affiliatePartyPaymentsTotal > nextThreshold.minQuantity AND NOT EXISTS
FinancialAccountTrans WHERE finAccountTransId = "PC${partner.id}_${lastThresholdIndex + 1}"
  create FinancialAccountTrans
    finAccountTransId = "PC${affiliatePartyId}_${lastThresholdIndex + 1}"
    finAccountTransTypeEnumId = "FattDeposit"
    finAccountId = partner.id
    fromPartyId = "L2"
    toPartyId = affiliatePartyId
    transactionDate = now
```

```
    amount = nextThreshold.termNumber - (lastReachedTerm.termNumber ?: 0)
  create legacy DiscountDoc incasare
  send threshold reached SMS to affiliate
else
  send payment received SMS to affiliate
```