Sinkronizacija podataka između sustava za upravljanje odnosima s klijentima i platforme za e-poslovanje

Otočan, Luka

Sveučilište u Rijeci – Odjel za informatiku

Informacijski i komunikacijski sustavi

Luka Otočan

Sinkronizacija podataka između sustava za upravljanje odnosima s klijentima i platforme za e-poslovanje

Diplomski rad

Mentor: izv. prof. dr. sc. Ana Meštrović

Komentor: Toni Miletić

Rijeka, 24.5.2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 28.5.2021.

Zadatak za diplomski rad

Pristupnik: Luka Otočan

Naziv diplomskog rada: Sinkronizacija podataka između sustava za upravljanje odnosima s klijentima i platforme za e-poslovanje

Naziv diplomskog rada na eng. jeziku: Data synchronisation between customer relationship management system and e-commerce platform

Sadržaj zadatka:

Zadatak diplomskog rada je definirati i opisati konceptualni model sinkronizacije podataka o vaučerima između sustava za upravljanje odnosa s klijentima i platforme za e-poslovanje. U radu je potrebno istražiti probleme u komunikaciji između postojećeg CRM sustava i platforme za e-poslovanje Valfresco, te dati prijedlog mogućeg rješenja koje će unaprijediti sinhronizaciju između sustava. Potrebno je objasniti tehnologije koje se koristite u novom načinu sinkronizacije i definirati arhitekturu novog sustava.

Mentor:

Izv. prof. dr. sc. Ana Meštrović

Voditelj za diplomske radove:

Izv. prof. dr. sc. Ana Meštrović

Komentor:
Toni Miletić

Zadatak preuzet: 28.5.2021.

Luka Otočan

(potpis pristupnika)

1. Sažetak

Cilj ovog diplomskog rada je razviti konceptualni model sinkronizacije podataka o vaučerima između CRM-a i platforma za e-poslovanje.

CRM je razvijen u Microsoft Dynamicsu dok je platforma za e-poslovanje razvijena od strane privatne tvrtke pomoću .NET frameworka.

CRM sustav sadrži podatke o klijentima, izrađuje kampanje te generira vaučer kôdove i podatke o vaučerima koji se dodjeljuju klijentima kroz određene promotivne kampanje. Platforma za e-poslovanje koristi vaučere kako bi klijenti dobili određene popuste za kupovinu unutar platforme te kako bi se korisnici iznova vratili na platformu radi naručivanja proizvoda.

Budući da, zbog potrebe platforme za e-poslovanje, već postoji komunikacija između dva sustava, kroz ovaj rad prikazani su trenutni procesi komunikacije između dva sustava te su izloženi problemi u svakodnevnom radu platforme i njegovih vaučera. Izrađen je i konceptualni model koji prikazuje kako bi trebala teći nova komunikacija između dva sustava.

Unutar koncepta biti će vidljiva API[1] komunikacija CRM Web servisa i Backend sustava Platforma za e-poslovanje, Frontend prikaz upravljanja vaučerima u administraciji na Platforma za e-poslovanje platformi i Apache Kafka sustav koji nadzire komunikaciju između dva sustava i dostavu poruka.

Ključne riječi: CRM, Platforma za e-poslovanje, Apache Kafka, Confluent Platform, sinkronizacija vaučera, API

[1] Engl. Application Programming Interface

2. Sadržaj

3. Uvod

Posljednji semestar studija informatike na Odjelu za informatiku sastoji se od stručne prakse.

Praksu sam odradio u turističkoj tvrtki Valamar Riviera u odjelu digitalizacije. Radi se o vrlo malom odjelu koji je nedavno otvoren zbog potrebe za digitalizacijom procesa i proizvoda unutar same tvrtke.

Prošla 2020. godina jedna je od izazovnijih godina zbog novonastale situacije izazvane pandemijom COVID-19. Valamar Riviera je pravovremeno reagirala i uočila kako bi se u novonastaloj situaciji mogla okušati u novoj djelatnosti, platforma za e-poslovanje.

Platforma za e-poslovanje Valfresco nastala je s ciljem okupljanja obiteljskih poljoprivrednih gospodarstava (OPG) s područja Istre i ostalih dijelova Hrvatske kako bi lokalno stanovništvo moglo kupovati domaće proizvode na jednom mjestu, iz udobnosti svoga svog doma. S obzirom na novonastalu situaciju izazvanu COVID-om 19 i propisane mjere fizičkog distanciranja, Valfresco je svojim nastankom omogućio potrošačima da i dalje kupuju kvalitetne i zdrave proizvode bez odlaska u trgovinu ili OPG. Valfresco nudi široki asortiman proizvoda uz što je uključena i dostava na kućni prag. Valfresco dostavu vrši sam, bez posrednika, što ubrzava sam proces dostave i proizvode je moguće dobiti već sljedeći dan.

Valamar Riviera je u vrlo kratkom roku, od samo 3 mjeseca, uspjela razviti „online“ platformu. Platforma se neprestano razvija kako bi svojim korisnicima omogućila bolje korisničko iskustvo, a djelatnicima što efikasnije i preciznije izvođenje poslova.

Nakon pokretanja online platforme Valfresco, jedna od stavki koju je bilo potrebno implementirati jest kupovina pomoću vaučera. Vaučeri su kodovi koje kupci dobivaju putem promotivnih kampanja na društvenim mrežama (Facebook i Instagram), e-mailom ili SMS porukom.

Vaučeri prvenstveno služe kako bi se korisnici Valfresca nastavili koristiti njegovim uslugama ili kako bi se privukli novi korisnici. Također, Valamar Riviera mjesečno daje vaučer kôdove svojim zaposlenicima ukoliko su u prošlom mjesecu izvršili barem jednu kupovinu.

Valamar Riviera je turistička tvrtka koja se prvenstveno bavi iznajmljivanjem turističkih objekata i pružanjem turističkih usluga zbog čega ima veliku bazu stalnih klijenata. Klijenti se nagrađuju kroz program vjernosti (engl. Loyalty program). Radi vođenja programa vjernosti i privlačenja novih klijenata tvrtka ima razvijen vlastiti Customer Relationship Management (CRM) sustav.

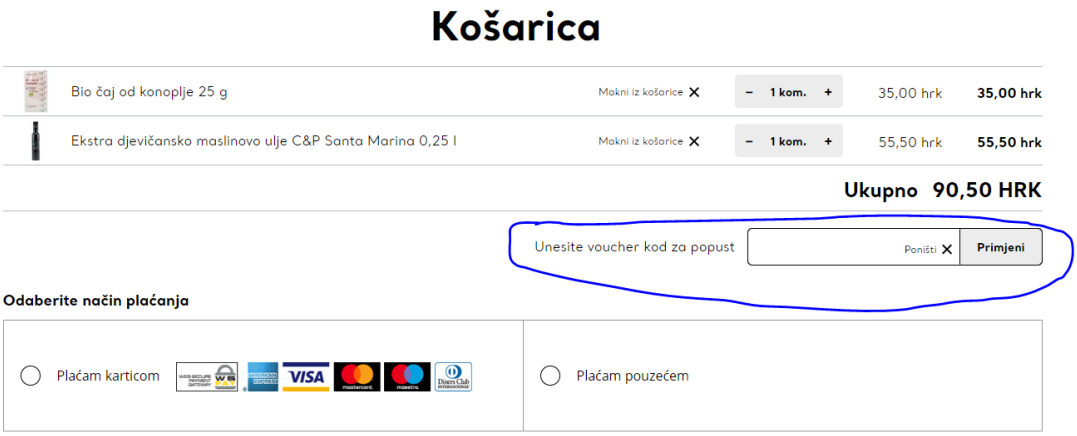
Zbog unaprjeđenja usluga koje nudi platforma Valfresco, koristi se postojeći CRM sustav. S obzirom na to da je implementacija sustava dobivanja vaučer podataka iz CRM-a uvedena u vrlo kratkom vremenskom roku, sustav je funkcionalan, ali s mnogim pogreškama u pozadini koje zahtijevaju ljudsko ispravljanje, odnosno mnogi procesi nisu automatizirani.

Ovim diplomskim radom izložit ćemo kako trenutno funkcioniranje sustava, pogreške unutar sustava te načine na koje ih možemo ispraviti. Rješenje sustava ostat će na konceptualnom modelu s obzirom na to da tvrtka nema pravo uređivanja procesa na CRM strani, a i samo uređivanje zahtjeva financijske izdatke koji u ovom trenutku nisu mogući. Konceptualni model sadržava način izvođenja sinkronizacije podataka o vaučerima iz CRM-a prema platformi za e-poslovanje te informaciju koji dodatni sustav moramo implementirati za upravljanje komunikacijom između dva navedena sustava. Dodatni sustav koji će upravljati komunikacijom između dva sustava bit će Apache Kafka.

Rad je strukturiran u tri cijeline. U prvoj cjelini opisani su trenutni procesi nastanka vaučera na platformi Valfresco i CRM-u. U tom dijelu objašnjena je trenutna sinkronizacija i komunikacija između ta dva sustava, njihove strukture podataka, baze i nedostatke. U drugoj cjelini objašnjava se sva tehnologija koju ćemo koristiti na novom načinu sinkronizacije platforme Valfresco i CRM-a. U zadnjoj cjelini prikazan je novi koncept sinkronizacije, odnosno, detaljno su objašnjeni svi elementi novoga koncepta koji će biti prikazan kroz arhitekturu visoke razine.

4. Postojeći proces unosa vaučera u ValFresco bazu

Proces unosa vaučera u Valfresco bazu započinje na frontendu[2]. Nakon što odaberemo sve proizvode koje želimo kupiti odlazimo u prikaz košarice. U tom prikazu košarice možemo vidjeti sve proizvode koje smo odlučili kupiti, odabrati način plaćanja (gotovinsko ili kartično) te unijeti vaučer kôd za popust (slika 1).



Slika 1. Izgled košarice Valfresca

Kako bismo ostvarili popust pri kupovini, u označeno polje unosimo vaučer kôd koji smo dobili kroz neku od promotivnih kampanja. Nakon što unesemo vaučer u označeno polje, pritiskom na gumb „Primjeni“ pozivamo API poziv za provjeru nalazi li se uneseni vaučer kôd u ValFresco bazi, **GetPromoCode** (URL: **/api/merchandise/promocode/{vaučer_kod})** .

Ako vaučer postoji u Valfresco bazi, korisniku se na frontendu prihvaća vaučer kôd i umanjuje se ukupni iznos košarice za iznos koji stoji na vaučeru.

Ako vaučer nije pronađen u Valfresco bazi, poziva se funkcija **CheckVoucherCodeAsync** na backendu koja unutar toga poziva CRM-ov API poziv **GetVoucherInfo**.

GetVoucherInfo je API poziv kojega šaljemo na server CRM-a. URL se sastoji od: **{{server}}/api/GetVoucherInfo?code={{apiKey}}==**, gdje je {{server}} DNS adresa servera, a {{apiKey}} autorizacijski token za pristup API-u koji smo dobili od providera CRM-a.

U tijelu (engl. Body) poziva (slika dolje) šaljemo sljedeće: **VoucherCode** što je vaučer kôd koji smo unijeli na frontendu i **ClientId** koji je jedinstven za Valfresco kojega smo dobili od pružatelja usluga CRM-a.

[2] Link do stranice: https://www.valfresco.com/

```
{
    "VoucherCode": "{vaučer_kod}",
    "ClientId": "{ClientId}}"
}
```

Nakon slanja API poziva prema CRM-u dobivamo sljedeći poziv:

```
{
    "response": {
        "success": true,
        "errorCode": null,
        "message": "Successfully retrieved voucher info",
        "voucherPromotionCode": "V23",
        "expiryDate": "2020-12-31T07:00:00",
        "voucherStatus": 100000001,
        "minimumValueEur": null,
        "minimumValueKn": null,
        "usageCount": 1,
        "maxUsageCount": null,
        "voucherProductCode": null,
        "promotionValidFrom": "2020-05-27T00:00:00",
        "promotionValidTo": "2020-12-31T08:00:00",
        "value": "15",
        "voucherPromotionType": 100000001,
        "currencyCode": "HRK",
        "currencyValue": null,
        "minimumCurrencyValue": 150.0
    }
}
```

Dobiveni podaci o vaučeru se spremaju u tablicu [Billing].[Coupon]. Dobiveni podaci se na slijedeći način mapiraju u navedenu tablicu:

JSON	[Billing].[Coupon]
voucherPromotionCode	[VoucherPromotionCode]
expiryDate	[ExpiryDate]
voucherStatus	[CouponStatusId] -> vrijednost dobiva iz tablice [Billing].[CouponStatus]

minimumValueEur	[MinimumValueEur]
minimumValueKn	[MinimumValueKn]
usageCount	[UsageCount]
maxUsageCount	[MaxUsageCount]
voucherProductCode	[VoucherProductCode]
promotionValidFrom	[ValidFrom]
promotionValidTo	[ValidTo]
value	[Ammount]
voucherPromotionType	[CouponTypeId] -> vrijednost dobiva iz tablice [Billing].[CouponType]
currencyCode	[CurrencyCode]
currencyValue	empty
minimumCurrencyValue	[MinimumValueCurrency]

Tablica 1. Mapiranje polja JSON-a i tablice Billing.Coupon

Na frontendu se tada aktivira vaučer koji dodjeljuje popust ili na cjelokupni iznos košarice ili na pojedini proizvod, ovisno o vrsti vaučera.

Ako unesemo vaučer, kôd koji ne postoji niti u CRM, niti u Valfresco bazi dobiti ćemo odgovor prikazan na slici 2:

- Frontend odgovor:

Slika 2. Poruka stranice ako uneseni vaučer kôd ne postoji

- JSON odgovor:


```
{
    "PromoCode": "V271-OHD8-ZKO",
    "IsValid": false,
    "ErrorCode": 100,
    "ErrorMessage": "Vaučer koji ste unijeli ne postoji",
    "MinCartValue": 0.0,
    "MinCartValueHrk": 0.0,
    "ValidFrom": null,
    "ValidTo": null,
    "Ammount": 0.0,
    "AmmountPercent": 0.0
```

```
}
```

U slučaju pogreške sustav javlja korisniku poruke prikazane u tablici 2:

Naziv metode za poruku	Poruka
VoucherAmountIsLessThanMinimumValue	Vaučer koji ste unijeli nije validan za odabrane uvjete
VoucherCannotBeUsedMoreThan2Times	Vaučer koji ste unijeli nije validan za odabrane uvjete
VoucherDoesntExist	Vaučer koji ste unijeli ne postoji
VoucherIsExpired	Vaučer koji ste unijeli nije aktivan
VoucherIsNotActive	Vaučer koji ste unijeli nije aktivan
VoucherIsNotValid	Vaučer koji ste unijeli nije validan za odabrane uvjete
VoucherProductIsUndefined	Vaučer koji ste unijeli nije validan za odabrane uvjete
VoucherPromotionDoesntExistOnVoucher	Vaučer koji ste unijeli nije validan za odabrane uvjete
VoucherStatusIsNotAvailable	Vaučer koji ste unijeli nije aktivan
VoucherUsageCountIsInvalid	Vaučer koji ste unijeli nije validan za odabrane uvjete

Tablica 2. Vrste poruka koje se mogu prikazati korisniku

Postojeće rješenje dobro funkcionira za novo unesene vaučere u bazu. Problem nastaje pri ažuriranju vaučera koji već postoji u Valfresco bazi. Trenutno ne postoji servis ažuriranja postojećih vaučera u bazi. Neke od postojećih situacija u kojima postoji potreba za izradom servisa ažuriranja vaučera su:

- Izmjena datuma trajanja kampanje → želimo smanjiti ili povećati datum trajanja kampanje ovisno o potrebama marketinga,
- povećanje/smanjenje ukupnog broja vaučera,
- greška pri unosu u CRM-u → ako dođe do pogreške pri izradi vaučera, a vaučer se već prenese u Valfresco bazu, tada ne postoji mogućnost njegova ažuriranja,
- razno.

Postojeći problemi se najčešće rješavaju SQL upitima direktno na bazu. Za postojeće vaučere u bazi, koji se trebaju ažurirati, dolazi do promjene stupca **IsActive** iz 1 u 0 te se tada pri novom upitu na frontendu ponovno šalje poziv na CRM za taj vaučer te se izradi novi zapis u bazi. Takav način izmjene nije optimalan iz razloga što nije automatiziran zbog čega nemamo unikatne zapise za svaki vaučer u bazi.

5. Postojeći proces iskorištavanja vaučera na CRM strani

Nakon što se vaučer iskoristi na Valfresco platformi, informacija o iskorištenosti vaučera šalje se CRM-u. CRM za to ima API poziv UseVoucher.

Method: POST

Naziv polja	Opis polja
VoucherCode	• Promo kod vaučera • Npr. V27-X123-Y858
AmountKn	• Iznos košarice u kunama
AmountEur	• Iznos košarice u eurima
UsageDateTime	• Timestamp korištenja vaučera
CurrencyAmount	• Iznos u glavnoj valuti, u ovom slučaju HRK
VoucherProductCode	• Promo kod vaučera od proizvoda
ConsumerAccountId	• Guid korisnika koji koristi vaučer • Guid je Guid iz CRM Baze
CosumerLoyaltyProfileId	• Guid Loyalty profila korisnika koji koristi vaučer
ConsumerValfrescoProfileId	• Guid Valfresco profila korisnika koji koristi vaučer
ClientId	• Guid sustava koji šalje obavijest o korištenju vaučera

Tablica 3. Opis JSON-a za UseVoucher

S obzirom na to da Valfresco još uvijek nema izrađenu funkciju prijave korisnika na stranicu, kao i funkciju praćenja prošlih narudžbi i unošenja osobnih podataka, polja **ConsumerAccountId**, **ConsumerLoyaltyProfileId**, **ConsumerValfrescoProfileId** koriste unaprijed definirane vrijednosti za ta polja. CRM ima ValfrescoWEB profil u svojoj bazi tako da za svaki vaučer koji se iskoristi na Valfrescu poprima vrijednosti ValfrescoWEB profila. Valfresco sprema svako korištenje vaučera u tablicu Billing.CouponUsed. Da bi se poslala informacija CRM-u o iskorištenosti vaučera, kreirana je metoda koja svake dvije minute prikupi svako novo korištenje vaučera te za svako korištenje iskoristi API poziv **UseVoucher**. Da bi se metoda aktivirala svake dvije minute koristi se alat koji se zove HangFire.

HangFire je alat unutar platforme koji koristimo za pokretanje automatskih ili zakazanih procesa. Pokretanjem metode provjerava se ima li u tablici **Billing.CouponUsed** vaučera koji su u statusu **Reserved**. Ako ima vaučera koji su u statusu **Reserved,** za svakoga od njih pokuša poslati API poziv **UseVoucher** kojim javlja CRM-u da je vaučer iskorišten. Ako poziv vrati HTTP kod 200, vaučer prelazi u status **Used** i polje **IsAlreadyUsed** dobiva vrijednost 1. Nakon toga provjerimo je li API poziv vratio poruku da je vaučeru iskorištenost uspješno odrađena. Ako nije, polju **IsCrmStatusSyncError** se promjeni vrijednost u 1.

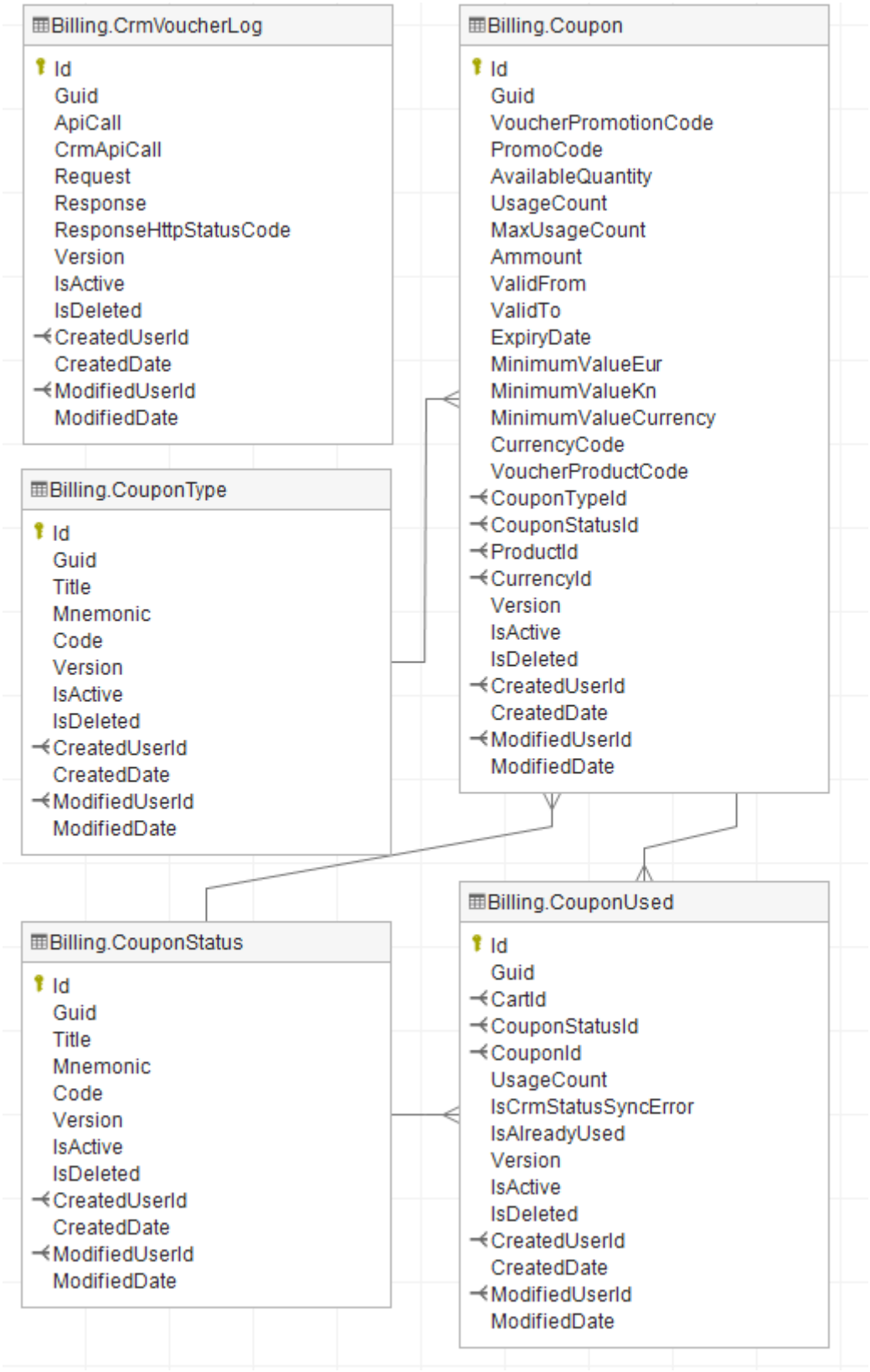
Dok se ova tablica nije izradila, kontrola nad rezerviranim ili trenutnim kuponima se vršila u tablici **Billing.Coupon**. Takva praksa je bila neučinkovita jer se vaučer prebacio u status **Reserved** ako bi ostao na jednom korištenju, te je korisnik morao čekati 10 minuta da se vaučer resetira. No s ovom tablicom korisnik ne mora čekati da se vaučer resetira. Status vaučera u tablici **Billing.CouponUsed** za tu košaricu će zauvijek ostati **Available** jer vaučer nije iskorišten to jest košarica nije finalizirana. Te se onda taj zapis pri računanju **UsageCounta** za određeni vaučer ne uzima.

6. Postojeće Valfresco tablice za vaučere

Trenutne tablice koje imamo u Valfresco bazi za vaučere su:

- Billing.Coupon
  - Tablica koja sadrži sve vaučere koji se nalaze u Valfresco bazi
- Billing.CouponType
  - Tipovi vaučera koji mogu stići iz CRM-a
  - Iz CRM-a mogu stići pet vrsta vaučera, iako se na Valfresco platformi trenutno koriste samo dva tipa:
    - Apsolutni iznos (engl. Absolute value)
      - Ovo su tipovi vaučera koji imaju iznos u nekoj valuti, npr. 200 kuna. Kada se unese ovaj tip vaučera korisniku se ukupan iznos košarice umanjuje za gore navedeni iznos. Što znači da ako imamo košaricu u vrijednosti od 500 kuna nakon unosa vaučera iznos košarice će iznositi 300 kuna.
    - Postotak (engl. Percent)
      - Ovo su tipovi vaučera na kojima je iznos označen u određenom postotku, npr. 20%. Kada se unese ovaj tip vaučera korisniku se ukupan iznos košarice smanjuje za gore navedeni postotak. Ukoliko imamo košaricu u vrijednosti od 500 kuna nakon unosa vaučera košarica će iznositi 400 kuna.
- Billing.CouponStatus
  - U bazi trenutno imamo 3 statusa vaučera:
    - Available: vaučer je aktivan te nije još iskorišten
    - Used: kod ovog vaučera možemo imati više situacija kada je vaučer Used:
      - Vaučer je iskorišten maksimalan broj puta (**maxUsageCount** = **usageCount**)
      - Vaučer je iskorišten, ali može se još koji put iskoristiti (**usageCount** != 0 && **usageCount** < **maxUsageCount**)[3]
    - Expired: datum do kada vaučer vrijedi je manji od današnjeg datuma (**expiryDate** < **todaysDate**)
- Billing.CouponUsed
  - U ovoj tablici bilježimo iskorištenost vaučera. Svakom vaučerom pridružujemo i njegovu košaricu da znamo za koju kupovinu je iskorišten koji vaučer.
  - Također, bilježimo je li se iskorištenost vaučera uspješno poslala prema CRM bazi
- Billing.CrmVoucherLog
  - U ovoj tablici bilježimo svaki API poziv prema CRM bazi

[3] usageCount je različit od 0 ali manji od maxUsageCount)



Slika 3. Relacijski dijagram trenutnih tablica za vaučere na platformi Vafresco

6.1. Billing.Coupon

U ovoj tablici zapisujemo podatke o vaučerima što je čini primarnom tablicom o vaučerima. U nastavku će biti opisano svako polje koje se nalazi u tablici 4:

Naziv stupca	Primjer podatka	Dodatni opis
Id	1	Primarni ključ retka u tablici.
Guid	9dbcbeba-dce0-4a8d-83d6-a329e61a4a79	GUID identifikator retka u tablici.
VoucherPromotionCode	V27	Promocijski kod kampanje kojoj pripada vaučer.
PromoCode	V277-848A-WVM4	• Prve tri znamenke su oznake kampanje • Ostalo je nasumičan izbor brojeva i slova
AvailableQuantity	2	Oznaka slobodne količine vaučera, to znači ako je AvailableQuantity = 2, vaučer se može još dvaputa iskoristiti.
UsageCount	1	Oznaka koliko puta je vaučer iskorišten.
MaxUsageCount	3	Oznaka koliko puta se vaučer može maksimalno iskoristiti. Zbroj AvailableQuantity i UsageCount-a.
Ammount	200.00	Iznos vaučera u određenoj valuti.
ValidFrom	2020-05-27T00:00:00	Datum otkada vrijedi kampanja kojoj pripada vaučer.
ValidTo	2020-05-27T00:00:00	Datum dokada vrijedi kampanja kojoj pripada vaučer.
ExpiryDate	2020-05-27T00:00:00	Datum isteka vaučera. ExpiryDate i ValidTo ne moraju imati jednake vrijednosti.
MinimumValueEur	200.00	Minimalni iznos košarice u eurima.
MinimumValueKn	200.00	Minimalni iznos košarice u kunama.
MinimumValueCurrency	200.00	Ista vrijednost kao MinimumValueKn.
CurrencyCode	HRK	Oznaka valute vaučera.
VoucherProductCode	V27	Ista vrijednost kao i VoucherPromotionCode.
CouponTypeId[4]	1	ID koji referencira na vrijednost iz tablice Billing.CouponType.

[4] Tip vaučera

CouponStatusId[5]	1	ID koji referencira na vrijednost iz tablice Billing.CouponStatus.
ProductId[6]	1	ID proizvoda koji je vezan uz ovaj vaučer.
CurrencyId	1	ID koji referencira na vrijednost iz tablice Common.Currency.
Version	1	Oznaka verzije retka. Svakom modifikacijom ova vrijednost se inkrementalno poveća za 1.
IsActive	1	Oznaka da li je redak u tablici aktivan.
IsDeleted[7]	0	Oznaka da li je redak u tablici izbrisan.
CreatedUserId	1	ID koji referencira na vrijednost iz tablice Common.User. Oznaka korisnika koji unio vaučer u tablicu.
CreatedDate	2020-05-27T00:00:00	Datum unosa podatka u tablicu.
ModifiedUserId	1	ID koji referencira na vrijednost iz tablice Common.User. Oznaka korisnika koji je zadnji promijenio vaučer u tablici.
ModifiedDate	2020-05-27T00:00:00	Datum zadnje promjene vaučera u tablici.

Tablica 4. Opis tablice Billing.Coupon

Neke od ovih podataka ne dobivamo putem CRM API-a **GetVoucherInfo**, no izrađuju se pomoću backend procesa. To su:

- AvailableQuantity
  - On se izračunava pomoću: **AvailableQuantity** = **MaxUsageCount** – **UsageCount**
- CouponTypeId i CouponStatusId
  - Ovi podaci dolaze sa šiframa iz CRM-a
  - Nakon dolaska vaučera na Valfresco stranu backend proces mora mapirati točne ID-eve statusa i tipova vaučera.
  - Tablice Billing.CouponType i Billing.CouponStatus sadrže stupce **Code**
    - U ovaj stupac se sprema vrijednost tipa ili statusa koji dolazi iz CRM-a.
    - Kada backend proces mapira statuse i tipove vaučera on uspoređuje vrijednost iz JSON polja **voucherStatus** i **voucherPromotionType** s poljem **Code** iz gornje navedenih tablica.

[5] Status vaučera

[6] Ova funkcionalnost se trenutno ne koristi, no njezin značaj bi bio sljedeći: ako želimo iskoristiti neki vaučer, a on kao ProductId ima Id Brašno 1 kg tada se u košarici mora nalaziti produkt Brašno 1 kg jer bez njega nije moguće iskoristiti vaučer.

[7] Ovo se u praksi zove SoftDelete. Kada želimo korisniku na prikazu reći da vaučer nije aktivan, odnosno da je izbrisan ali on u bazi i dalje postoji samo što ima oznaku kao da je izbrisan.

- Ako taj **Code** postoji u tablicama onda se vraća njihov ID iz tablice i taj ID se zapisuje u polja **CouponTypeId** i **CouponStatusId** u tablicu Billing.Coupon

Primjeri podataka koji se mogu mijenjati nad vaučerom:

- AvailableQuantity i UsageCount
  - Ako dođe do korištenja vaučera na frontendu backend proces će **AvailableQuantity** smanjiti za 1 dok će se **UsageCount** povećati za 1
- AvailableQuantity i MaxUsageCount
  - U slučaju da se za navedeni vaučer poveća ili smanji broj mogućih korištenja tada će se ove dvije vrijednosti izmijeniti
- Ammount
  - Ako smanjimo ili povećamo iznos ili postotak vaučera
- ValidFrom, ValidTo i ExpiryDate
  - Ako smanjimo ili povećamo vremenski interval trajanja kampanje
  - Ako smanjimo ili povećamo vremenski interval trajanja vaučera
- CouponTypeId
  - Ako vaučer prebacimo iz tipa **AbsoluteValue** u tip **Percent** ili obrnuto
- CouponStatusId
  - Ako se vaučer iskoristi maksimalni broj puta (**MaxUsageCount** = **UsageCount**) tada vaučer prelazi u status **Used**
  - Ako je vaučeru **ExpiryDate** manji od današnjeg datuma, tada je vaučer istekao

6.2. Billing.CouponUsed

Ova tablica nam služi za spremanje informacija o tome koji je vaučer iskorišten za koju kupovinu. Također, ova tablica omogućava da se vaučer može iskoristiti više puta u slučaju da dođe do neke greške na frontendu.

Prije uvođenja ove tablice vaučer bi se rezervirao za korištenje na CRM-u pomoću njegovog API-a čim se je unio u polje „Unesi promo kod“ na frontendu. Ukoliko bi došlo do problema, npr. kupovina je neuspješno završena zbog pada WSPay[8] sustava, korisnik nije mogao ponovno izvršiti kupovinu jer mu je vaučer bio zauzet sljedećih 10 minuta te je morao čekati kako bi mu se vaučer ponovo oslobodio. Ova praksa se pokazala izuzetno lošom jer je klijent bio frustriran zbog činjenice da mora čekati 10 minuta kako bi ponovo pokušao obaviti kupovinu što je moglo rezultirati gubitkom kupaca.

Stupci tablice Billing.CouponUsed:

Naziv stupca	Opis stupca
Id	**ID stupca**
Guid	**GUID stupca**
CartId	**Referencira na Id košarice u kojoj iskorišten vaučer**

[8] Tip Internet Payment Gateway-a

CouponStatusId	**Status vaučera u košarici**
UsageCount	**Koliko puta je iskorišten vaučer u košarici**
IsCrmStatusSyncError	**Da li je poziv prema CRM-u da je vaučer iskorišten uspješno prošao zapisujemo u ovu tablicu**
IsAlreadyUsed	**Da li je vaučer već iskorišten**

Tablica 5. Opis tablice Billing.CouponUsed

Zapis se u tablici **Billing.CouponUsed** izrađuje tek prilikom izrade košarice. Kada se izradi košarica, izradi se zapis i pridruži se **Id** vaučera iz tablice **Billing.Coupon**. Status vaučera u tablici **Billing.CouponUsed** prelazi u status **Available**. On je u statusu **Available** jer kupovina još uvijek nije finalizirana pa ne možemo smatrati kupon iskorištenim. Ako se kupovina ne finalizira, status vaučera će u ovome zapisu ostati **Available**. Ako se kupovina finalizira, vaučer prelazi u status **Reserved**. Kupon je iskorišten na ValFresco platformi i **UsageCount** je povećan za 1.

7. Vaučeri u sustavima za upravljanje odnosima s klijentima

7.1. Sustav za upravljanje odnosima s klijentima

CRM je alat za upravljanje odnosima s klijentima. CRM analizira podatke svojih sadašnjih i prijašnjih kupaca te pomoću ovih podataka želi potaknuti kupce da i dalje ostanu njihovi kupci [1]. Svaka tvrtka želi zadržati svoje kupce, ali i proširiti bazu kupaca s ciljem povećanja prodaje, a samim time i profita [2].

CRM podatke o kupcima može prikupiti s više kanala:

- WEB,
- E-mailovi,
- Online chatovi,
- Telefonski pozivi,
- Društvene mreže.

CRM kao alat je nastao 80-ih godina prošlog stoljeća. I prije razvijanja CRM-a, pokušao se razviti odnos s kupcima pomoću godišnjih anketa ili personalnim tehnikama. Razvijanjem tehnologije 80-ih godina 20. stoljeća, snaga računala je porasla dok je njihova cijena padala što je potaknulo brojne tvrtke da krenu koristiti računala za upravljanje podatcima o kupcima.

Prvi počeci prikupljanja podataka o kupcima su:

- Kategoriziranje kupaca i
- Skupljanje njihovi podataka u liste ili proračunske tablice.

Današnji CRM alati korištenjem modernih tehnologija poput brze internetske veze, društvenih mreža, pametnih telefona, cloud-a i sl., postali su još moćniji.

CRM danas nudi velike mogućnosti:

- Organiziranje i automatizacije prodaje,
- Marketing,
- Upravljanje podacima o kupcima.

CRM može odjednom reklamirati različite proizvode na različitim kanalima pomoću čega možemo pratiti s kojeg se kanala kupac odlučio za kupnju određenog proizvoda. Istovremeno, kada se kupac odluči za određeni proizvod, prikupljamo podatke o kupcima pomoću kojih možemo pratiti što najčešće kupuje. Ova informacija nam je vrlo važna jer pomoću nje možemo „targetirati“ određene klijente s proizvodima koje najčešće kupuju kako bi učinkovitost marketinga proizvoda bila što veća.

CRM u sklopu Valfresco platforme trenutno pokušava zadržati postojeće ili privući nove kupce na kupovinu s različitim vaučerima. Kroz daljnjem razvoju u planu je implementirati i vezu s kupovinama kupca i CRM-om kako bi CRM mogao bilježiti sve kupovine kupca na Valfrescu kako bi se kupac mogao još bolje informirati o određenim proizvodima.

7.2. Proces unosa vaučera i kampanja u CRM-u

Životni ciklus vaučera počinje u CRM-u. CRM koji se koristi je izrađen od tvrtke Be-terna (bivša Adacta). CRM je izrađen u Microsoft Dynamics 365.

U CRM-u se prvotno izrađuje kampanja jer vaučer mora pripadati nekoj kampanji. Kampanje se izrađuju u dva oblika:

1. Unosi se lista unaprijed određenih klijenata koji će dobiti vaučer, te se za svakoga klijenta unaprijed definira jedinstveni promo kod
2. Izrađuje se kampanja i određuje se maksimalni broj vaučera koji se može iskoristiti i onda se izrađuje newsletter koji se šalje klijentima ili promovira na marketing kanalima: društvene mreže, mail i itd.
    a. Ako se šalje preko društvenih mreža onda će svaka osoba imati isti promo kod
    b. Ako se šalje mailom onda svi klijenti mogu ali ne moraju imati isti promo kod

Svaka kampanja sadrži sljedeće podatke:

- Kod kampanje (u tablici Billing.Coupon stupac **VoucherPromotionCode**)
- Naziv kampanje
- Otkad do kad vrijedi kampanja (Promotion Valid From i Promotion Valid To)
- Max Usage Number
    - Unosi se kod kampanje tipa 2, kod tipa 1 nije potreban
- Tip vaučera (apsolutna vrijednost, postotak…)
- Valuta (kuna ili euro)
- Iznos vaučera (npr. 10% ili 100 kuna)
    - Unosi se samo broj, postotak ili novčanu vrijednost prepoznaje po tipu vaučera
- Minimalna vrijednost kupovine (nije obavezno, jedino ako je dogovoreno)
- „Recipient list“ – lista primatelja vaučera, koristi se kod kampanje tipa 1
    - Kod kampanje tipa 2 lista primatelja se izrađuje unutar drugog programa (Responsys) i tek kada se iz toga programa pošalje vaučer ispuni se automatski i ova lista
    - U ovoj listi možemo vidjeti jedinstveni promo kod za svakoga klijenta ako kampanja ima vaučere jedinstvene za svakoga primatelja

Nakon što se iz generiraju promo kodovi šalju se klijentima koji su određeni za primiti ga. Valfresco platforma ne dobiva nikakvu obavijest o vaučeru dok ga netko ne pokuša iskoristiti na frontendu.

7.3. Problemi s CRM-om

CRM ima nekakvih nedostataka gdje je logika oko vaučera nejedinstvena za pozive na isti servis. Primjer: vaučeru je **ExpiryDate** prošao, ali kada se pošalje API poziv na CRM **GetVoucherInfo** dobije se status **Available** ili **Used** ovisno o iskorištenosti.

Nedostaci takvog rješenja su:

1. Izrada datuma kampanje s opcijom „Sljedećih 30 dana“ ne šalje **ExpiryDate** preko **GetVoucherInfo** poziva
2. Mijenjanjem **ExpiryDate** nad vaučerom ne ažurira i automatski status vaučera ili obrnuto
3. Kampanja tipa 1 ne izrađuje vrijednost **MaxUsageCount**
4. **GetVoucherInfo** kao odgovor daje krive vrijednosti
5. CRM nema opciju filtra samo Valfresco vaučera

7.3.1. Nedostatak 1.

Pri izradi kampanje biramo između dvije opcije vremena trajanja kampanje:

1. Fiksni period – datumom određeno otkad do kad
2. Sljedećih 50 dana – kampanja vrijedi od danas 50 dana

Ako za trajanje kampanje odaberemo opciju pod brojem 1 onda nam se otvara mogućnost da izaberemo točan datum i vrijeme od kad do kad kampanja traje te u tom slučaju nam **ExpiryDate** svakoga vaučera postaje jednak datumu do kada vrijedi kampanja. No ako za trajanje kampanje odaberemo opciju 2 onda nam sustav automatski dodijeli datum trajanja kampanje, ali iz nekoga razloga nad svakim vaučerom ne stvori **ExpiryDate**. Općenito, **ExpiryDate** je moguće mijenjati samo za svaki vaučer pojedinačno, a ne nad cijelom kampanjom. Kampanja nema **ExpiryDate**.

Prijedlog rješenja: unositi trajanje kampanje uvijek pomoću opciju 1 ili s providerom CRM-a razriješiti problem zašto za opciju 2 ne izrađuje se **ExpiryDate** te ispraviti grešku.

7.3.2. Nedostatak 2.

Vaučer bez **ExpiryDate** će se upisati u Valfresco bazi i izradit će se kupovina, no problem nastaje kada se prema CRM-u šalje API poziv **UseVoucher** kako bi se javilo CRM-u da je vaučer iskorišten. Od **UseVoucher** poziva dobiti ćemo odgovor da je vaučer istekao (jer nema **ExpiryDate**) te CRM ne može dobiti informaciju da je vaučer iskorišten stoga **UsageCount** u CRM-u neće biti točan.

ExpiryDate se može mijenjati svakom vaučeru posebno. **ExpiryDate** vaučera kao standardnu vrijednost podrazumjeva datum do kada vrijedi kampanja. Status svakoga vaučer može se izmijeniti za svaki vaučer pojedinačno ili nad cijelom kampanjom. Ako kampanja istekne, odnosno ako je datum završetka kampanje prošao, status kampanje odlazi u status **Expired** i svi vaučeri s njom. Ako smo izmijenili **ExpiryDate** vaučera da npr. traje duže nego kampanja, moramo i status vaučera promijeniti u **Available** ili **Used** Kada prođe **ExpiryDate** vaučera, vaučer nije moguće iskoristiti. Ukoliko bismo odradili API poziv nad tim vaučerom, on bi stajao u statusu **Available** ili **Used**, a ne **Expired**. Razlog tome jest da se status vaučera ne ažurira automatski kad mu **ExpiryDate** prođe. Status treba ručno ažurirati.

Prijedlog rješenja: dogovoriti s providerom CRM-a da se status vaučera automatski ažurira. Ako je **ExpiryDate** vaučera jednak datumu završetka kampanje, potrebno je sinkronizirati status vaučera sa statusom kampanje. Ako je **ExpiryDate** različiti od datuma završetka kampanje potrebno je ažurirati status u **Expired**, ukoliko je **ExpiryDate** prošao. Ažuriranje statusa **Available** i **Used** ostaje kao i do sada pri čemu ono ovisi o **UsageCount**.

7.3.3. Nedostatak 3.

Kampanje koje se izrađuju pomoću tipa 1 ne dobivaju **MaxUsageCount**. Razlog tome je što su u takvoj kampanji svi vaučeri u obliku da je promo kôd jedinstven za svakog klijenta, to jest može se iskoristiti samo jedanput. Tada je za sve te vaučere **MaxUsageCount** 1, što se sa

stajališta CRM-a podrazumjeva te im taj podatak nije važan, no sa stajališta Valfresa taj je podatak od iznimne važnosti.

Platforma Valfresco, kako smo opisali u prijašnjem dijelu teksta, **AvailableQuantity** računa tako da **MaxUsageCount** oduzme od **UsageCounta**, to jest maksimalan broj korištenja oduzme se od broja iskorištenosti vaučera. S obzirom da CRM-ov API **GetVoucherInfo** za vaučere bez **MaxUsageCounta** nema definiranu vrijednost, Valfresco ne može izračunati koliko puta se može vaučer iskoristiti. U slučaju da je **MaxUsageCount** ne definiran, **AvailableQuantity** računa se 1 minus **UsageCount**. Ako je **UsageCount** ne definiran, on postaje 0 te dobijemo da **AvailableQuantity** postaje 1.

Ova praksa nije dobra, općenito nije dobro da je ovaj podatak ne definiran. Jer ako nam je **MaxUsageCount** ne definiran, mogli bismo shvaćati da nam je neka greška s API pozivom. Bolje da se zapiše 1. Isti problem postoji i kod **UsageCounta**. Bolje rješenje bi bilo da se zapiše 0 kada vaučer nije nijednom iskorišten nego da ostane nedefiniran.

Prijedlog rješenja: treba dogovoriti da se pri izradi takve kampanje dodijeli **MaxUsageCount** 1 te će se onda svakom vaučeru u kampanji prepisati ta vrijednost ili da sustav prepozna o kakvom se tipu kampanje radi te on automatski dodijeli **MaxUsageCount** 1.

7.3.4. Nedostatak 4.

GetVoucherInfo daje krive vrijednosti za pojedine vaučere. Za vaučere koji su istekli javlja da su u statusu **Available** ili **Used**. Tek kada se proba pozvati **UseVoucher** ili **CanUseVoucher** dobijemo poruku od CRM-a da je vaučer istekao.

Prijedlog rješenja: treba ispraviti **GetVoucherInfo** da odmah dobijemo informaciju je li vaučer u statusu **Expired**.

7.3.5. Nedostatak 5.

Valamar, osim vaučera za Valfresco, ima i vaučere za smještajne objekte koji se također nalaze u CRM-u. CRM u svom sustavu nema nikakvo polje gdje se sprema vrijednost za koji brend je izrađen vaučer. Oni nisu nikako odvojeni, to jest ne postoji podatak po kojemu se prepoznaje je li kampanja za Valfresco ili smještajne objekte.

Prijedlog rješenja: izraditi polje gdje se sprema naziv brenda za koga vrijedi kampanja. U CRM-u treba postojati neka oznaka da se može znati koji vaučeri su za Valfresco, a koji nisu.

8. Prijedlog sinkronizacija CRM-a i Valfresca

Nakon što su istraženi sustavi CRM i Valfresco treba pronaći način kako riješiti problem sinkronizacije vaučera između ta dva sustava. Inicijalni problem radi kojega je pokrenut ovaj koncept je nemogućnost ažuriranja postojećih vaučera.

Također možemo reći da trenutni proces unosa vaučera nije najbolje rješenje i nije u realnom vremenu. Trenutno rješenje u slučaju da se vaučer ne nalazi u Valfresco bazi mora provjeriti CRM bazu pomoću **GetVoucherInfo** API-ja. Korištenjem CRM API-ja se uočilo da često zna imati probleme po pitanju brzine dobivanje odgovora (u početku se znao čekati odgovor API-ja i do pola minute) i nekonzistentnost podataka. **GetVoucherInfo** za neke vaučere ima jednu informaciju, dok **CanUseVoucher** ima drugačiju informaciju za te iste vaučere.

Dolazimo do zaključka da s ovakvim rješenjima pomoću API-ja nikad nećemo imati sve vaučere u Valfresco bazi jer postoji mogućnost da se neki vaučeri neće nikada iskoristiti na Valfresco strani.

Ako želimo koristiti API-je za pratiti izmjene vaučera to znači da moramo u HangFire-u kreirati proces koji će se pokretati na dnevnoj bazi te možemo imati dvije opcije:

1. Gledati pomoću API-a **GetVoucherInfo** ima li promjena na svakom vaučeru u Valfresco bazi
2. CRM mora izraditi API kojim možemo dobivati vaučere koji su se promijenili ili nastali od pokretanja procesa dan ranije.

Ovakve procese želimo izbjeći jer ćemo s opcijom 1. na dnevnoj bazi opterećivat sustav s provjerama vaučera. Vaučeri se ne mijenjaju na dnevnoj bazi, većinom se ne mijenjaju niti nakon prve izrade.

Opcija broj 2 iziskuje velike tehničke implementacije od strane pružatelja usluga CRM-a, što sa sobom povlači i velike financijske izdatke koje firma mora uložiti da bi se ovaj proces izradio.

Moramo izraditi rješenje koja će nam omogućiti:

- Dobivanje novih i ažuriranih vaučera u „skoro stvarnom vremenu“
- Dobivanje konzistentnih podataka o vaučerima
- Smanjiti što je više moguće financijske izdatke koje firma mora napraviti da bi se sinkronizacija izvršila
- Automatsko ažuriranje statusa vaučera

Rješenje koja nam omogućava ove funkcionalnosti moguće je izraditi uz pomoć Apache Kafka.

8.1. Apache Kafka

Apache Kafka jest platforma za dijeljenje događaja (engl. event streaming platform) koja je open source[9]. Event stream processing (ESP) je metoda koju pokrećemo nad izvorištima podataka koji kontinuirano kreiraju neke podatke [3]. Pojam event označava izvorište podataka dok „stream“ (hrv. tok) označava kontinuirani priljev podataka kroz te evente. Akcije koje se često izvode na tim podacima su:

- Agregacije (sumiranje, standardne devijacije i itd.)
- Analitika (izrada izvješća iz dolazećeg priljeva podataka)
- Transformacija (dobivene podatke iz evenata transformirati u drugačije formate podataka)
- Obogaćivanje (kombiniranje podataka iz različitih evenata kako bismo dobili konkretnije podatke ili informaciju)

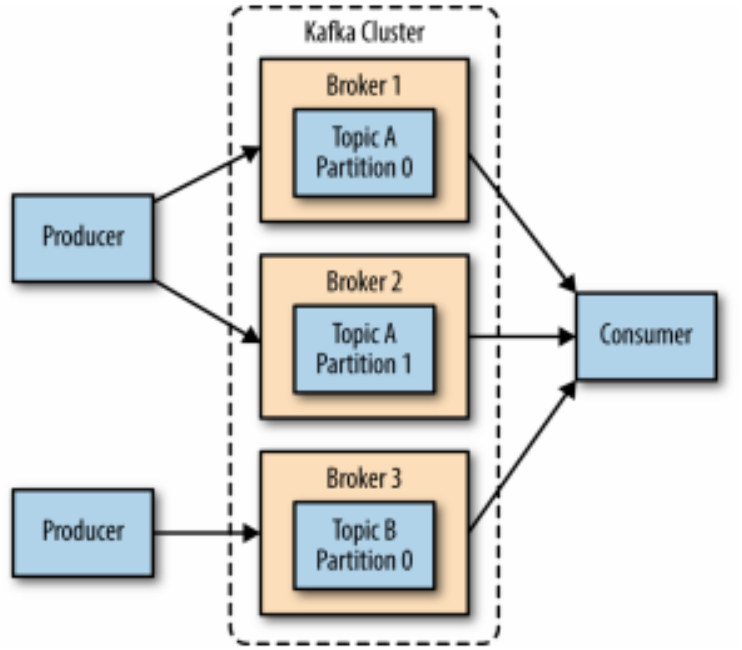
Suprotnost event streamingu podataka je batch (hrv. hrpa) procesiranje podataka. Batch procesiranje podataka je tip procesiranja gdje se nekoliko puta dnevno odjednom uzima veliki skup podataka i obradi se, dok event streaming uzima podatke kako dolaze i odmah se obrađuju u stvarnom vremenu.

Primarni cilj Kafke je prikupiti podatke s različitih izvorišta, izvesti jednu ili više od gore navedenih akcija te nove podatke proslijediti sustavima kojima su potrebni određeni podaci. Kafka funkcionira na principu „publish/subscribe“ metode. Publish/subscribe poruke funkcioniraju na način da pošiljatelja poruke ne zanima kome šalje poruke, nego ih kategorizira u određene klase. Nakon toga primatelj poruke (engl. subscriber) sam određuje koja klasa poruka ga zanima te čita poruke iz određene klase.

Kafka se sastoji od tri primarna dijela:

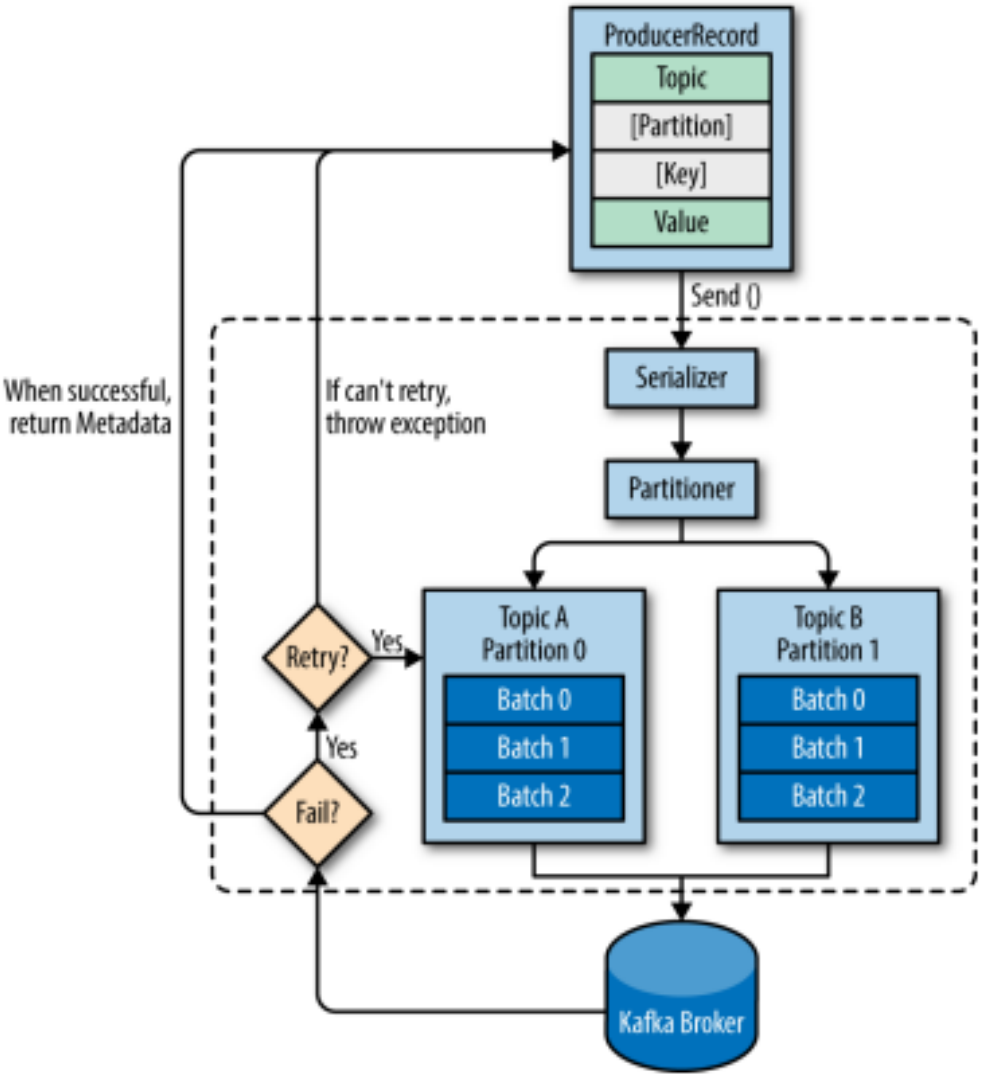
- Proizvođač (engl. Producer)
- Posrednik (engl. Broker)
- Potrošač (engl. Consumer)

[9] Odnosi se na softver čiji je izvorni kôd dostupan svim korisnicima. To znači da bilo tko može mijenjati, prepravljati ili poboljšati njegov izvorni kôd.



Slika 4. Primjer jednostavnog Kafka klastera (slika preuzeta iz [4])

Proizvođača smatramo izvorištem podataka [4]. Proizvođač prikuplja podatke s jednog izvorišta podataka, u našem slučaju to može biti CRM baza, te ih šalje u **topic**. **Topic** je kanal podataka između proizvođača i potrošača, odnosno jedan od oblika queue (eng. red) gdje podaci stoje dok ih potrošač ne preuzme. Proizvođač podatke u **topic** šalje u obliku poruke. **Topic** može imati više particija kako bi sustav mogao biti skalabilan. Proizvođač ne šalje poruke na određenu particiju, nego se posrednik brine o tome.



Slika 5. High level pregled producer komponenti (slika preuzeta iz [4])

Kafka posrednik je u suštini Kafka server [4]. Posrednik je zadužen za slijedeće radnje:

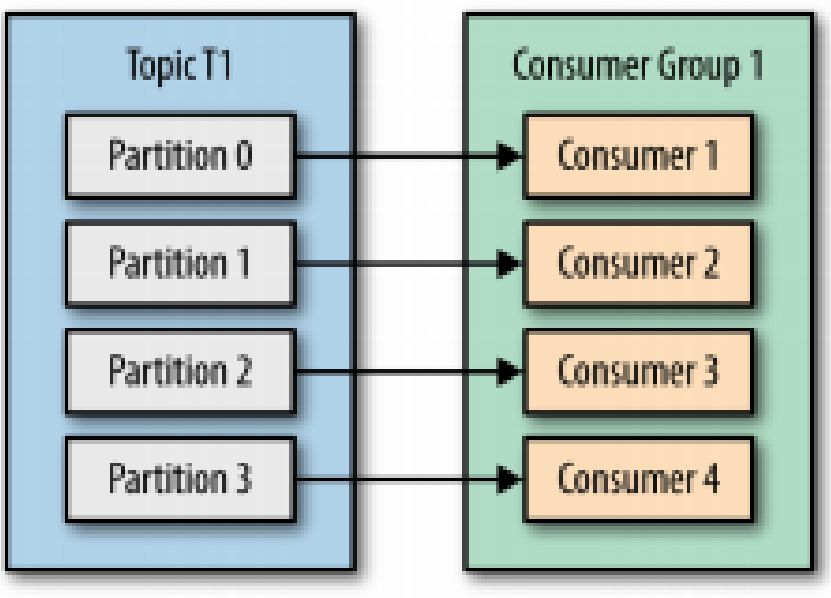
- Prima poruke s proizvođača
- Sprema oznaku do koje je poruke potrošač stigao u **topicu** (offset)
- Sprema poruke na disk.

Posrednik za svaki **topic** ima definiran **retention policy**. **Retention policy** nam označava za koje vremensko razdoblje želimo imati poruke spremljene u **topic-ima**. To jest ako nam je **retention policy** sedam dana, poruke koje su starije više od 7 dana se brišu u **topic-u**. Također, **retention policy** nam definira i koliko maksimalno veliki **topic** može biti. Ukoliko kažemo da nam je **retention policy** 1 GB, onda će nam posrednik početi brisati najstarije poruke ako **topic** pređe definirani limit od 1 GB.

Potrošač je primatelj poruka, on čita poruke s određenog **topic-a**. Potrošač je dio potrošačke grupe (engl. Consumer group). Potrošačke grupe služe za brže čitanje poruka s jednog **topic-a**. Recimo da imamo **topic** s dvije particije, proizvođač producira poruke u **topic** i posrednik ravnomjerno raspoređuje poruke u te dvije particije. Ako imamo samo jednog potrošača u potrošačkoj grupi, onda taj potrošač mora čitati poruke s obje particije. No, ako imamo dva

potrošača iz iste potrošačke grupe tada će se potrošači podijeliti tako da svaki čita s jedne particije te će nam onda i čitanje poruka biti dvostruko brže.

Definiranje više particija unutar **topic-a** je pogodna za **topic-e** koji će biti konstantno pod velikim pritokom podataka i kada imamo u paraleli više potrošača [4]. Broj potrošača i particija unutar **topic-a** treba definirati shodno količini podataka koja prolazi kroz **topic** kako nam se ne bi dogodilo da imamo više potrošača nego što nam je potrebno.



Slika 6. Primjer consumer grupe s ravnomjernim brojem consumera i particija u topicu (slika preuzeta iz [4])

8.2. Confluent Platform

LinkedIn[10] je poslovna socijalna mreža. Prvenstveno služi kao platforma na kojoj korisnici mogu pronaći oglase za posao, regruteri mogu pronaći potencijalne nove radnike za svoju tvrtku. LinkedIn je razvio Kafku kao rješenje za problem koji su imali sa prikupljanjem određenih metrika i podataka s različitih izvorišta nakon čega su analizirali prikupljene informacije.

Jedan od primjera je sakupljanje podataka o korisnikovoj aktivnosti na LinkedIn-u. Da bi izradio analitiku o korisnikovoj aktivnosti, sustav je morao periodički prikupiti podatke s frontend servera u **XML-u**[11] formatu, obraditi **XML** i izraditi analitiku s podacima iz **XML-a**. Ovakvo prikupljanje podataka zove se batch procesiranje. Ovakvo procesiranje bilo je loše iz dva razloga:

- **XML** podatak nije imao konzistentne podatke te je često bacao greške
- Svaka obrada **XML-a** je računski skupo to jest nepotrebno korištenje procesorske snage servera

Tim koji je radio na Kafki unutar tvrtke LinkedIn odvojio se i osnovao tvrtku naziva Confluent. Confluent se primarno bavi razvojem Kafke, podrškom za tvrtke i treningom za korištenje Kafke [5]. Jednu od tehnologija koje je Confluent razvio je Confluent Platform koji ćemo mi koristiti u ovome konceptu.

[10] Link na stranicu: www.linkedin.com
[11] Extensible Markup Language

Confluent Platform je Apache Kafka event streaming platforma koja je skalabilna te donosi dodatne servise koji će nam olakšati korištenje Kafke. To su:

- Control center
    - Control center je GUI[12] sučelje koje se koristi za nadzor, kontrolu i modificiranje Kafke
    - Control center nam nudi:
        - Pregled **topic-a** i njegovih poruka
        - Pregled potrošačkih grupa i njihove oznake do koje je poruke pročitao u **topic-u**
        - Postavljanje Kafka konektora (engl. Connector)
- Kafka Connect
    - Kafka Connect je alat koji nam služi za sigurno i skalabilno slanje podataka između Apache Kafke i ostalih sustava
    - Kafka Connect postoji i bez Confluenta, no pomoću Confluenta dobivamo dodatne konektore koje možemo koristiti
    - Konektori su ti koji se brinu za slanje podataka između Apache Kafke i ostalih sustava
    - Jedan od „use caseova“ konektora je:
        - Slanje cijele baze podataka u Kafku

Confluent Platform možemo instalirati na sljedeće načine:

- On – premise instalacija:
    - Instalacija Confluent platforme direktno na Linux mašinu
    - Docker kontejneri
- Cloud instalacija:
    - Confluent platforma kao servis (PaaS) unutar Confluent Clouda

Confluent platforma je besplatna za jedan klaster tako da će se u ovome konceptu nalaziti jedan posrednik. Instalacija će se izvršiti pomoću Docker kontejnera.

Docker [6] (hrv. kontejner) je poput virtualnog računala. Možemo reći da je svaki kontejner kao svaki operacijski sustav za sebe [7]. Svaki kontejner nam funkcionira neovisno o radu ostalih kontejnera [8]. No, velika prednost kontejnera naspram standardnih virtualnih računala je ta što se svi kontejneri izvode na istom operacijskom sustavu te se time smanjuje potreba za višestrukom upotrebom operacijskih sustava.

Ako želimo razviti aplikaciju koja će imati odvojeni razvoj frontenda i backenda, primjerice Frontend u Reactu, a backend u .NET Core, pri podizanju aplikacije na produkcijski ili testni server Docker nam uveliko može olakšati stvar. Svaki dio aplikacije frontend i backend zapakiramo u zasebni kontejner. Kada onda želimo podignuti aplikaciju na neki server, preuzeti ćemo kontejnere na taj server i pokrenuti ih. Kontejneri ne zahtijevaju nikakvu dodatnu konfiguraciju jer smo sve namjestili u konfiguraciji kontejnera. Ovakvim principom si možemo dodatno ubrzati razvoj i puštanje proizvoda u produkciju.

Za izradu našeg koncepta imat ćemo slijedeće kontejnere:

- Zookeeper

[12] Graphical User Interface

    - Zookeeper je centralizirani servis za održavanje, izradu nazivlja i distribuirane sinkronizacije za distribuirane aplikacije[13]
    - Kafka koristi Zookeeper za spremanje meta podataka o Kafka klasteru
- Kafka connect
- Control center
- MySQL
    - MySQL server koji će nam imitirati CRM i Valfresco bazu

Kontejneri imaju još jednu zgodnu mogućnost koja se zove Docker Compose. Pomoću ove mogućnosti možemo konfigurirati naše kontejnere zajedno u jednoj **.yaml** datoteci te ih pokretati sve odjednom pomoću jedne naredbe.

Pomoću kontejnera dobivamo na fleksibilnosti u implementaciji Kafke i ostalih servisa te onda nije potrebna ručna konfiguracija servisa u operacijskom sustavu gdje ubrzavamo proces implementacije i dobivamo na skalabilnosti sustava.

[13] Distribuirane aplikacije su aplikacije koje se nalaze u cloudu i pokreću se na različitim sistemima simultano

8.3. Konfiguracija kontejnera

Svaki kontejner će nam sadržavati standardnu konfiguraciju i njegove dodatne varijable za okruženje ili terminalske naredbe.

Pod standardnu konfiguraciju smatramo:

- **Image**
  - **Image** je nepromjenjiva datoteka koja sadrži kôd, biblioteke, ovisnosti i alate koji trebaju aplikaciji da se pokrenu
- **Container_name**
  - Naziv kontejnera kako bismo ga znali kasnije pozivati za ostale servise
- **Restart**
  - Definira pokreće li se ponovno kontejner za neke slučajeve
  - Za restart imamo sljedeće vrijednosti
    - **No** → ne pokreće se ponovno za niti jedan slučaj
    - **Always** → za bilo koji slučaj se ponovno pokreće
    - **On-failure** → ponovno se pokreće ako kontejner izbaci neku grešku
    - **Unless-stopped** → ponovno se pokreće uvijek osim u slučaju kada je kontejner već zaustavljen
- **Depends_on**
  - Oznaka da li je određeni kontejner ovisan o nekim drugima kontejnerima
- **Ports**
  - Oznaka porta nad kojem je moguće pristupiti kontejneru, npr. 8080

Za sve kontejnere **Restart** će biti postavljen na vrijednost „**always“** u slučaju da radimo neke izmjene na konfiguraciji kontejnera da nam se one ažuriraju.

8.3.1. Zookeeper

- Image
  - confluentinc/cp-zookeeper:5.4.0
  - Confluentova slika Zookeeepera
- Container_name
  - zookeeper
- **Port** za Zookeepera definiramo unutar varijabla za okruženje te njegov standardni port je 2181

8.3.2. Kafka

- Image
  - confluentinc/cp-enterprise-kafka:5.4.0
  - Confluentova slika Kafke
- Container_name
  - kafka
- Depends_on
  - zookeeper
  - Kafka ne može raditi bez Zookeepera
- Ports:
  - 9092

- Unutar varijabli okruženja definiramo:
    - Bootstrap server za Kafku
        - Bootstrap server služi kao oznaka na koji Kafka posrednik nam se proizvođači ili potrošači spajaju kako bi producirali ili konzumirali poruke s Kafka **topic-a**
    - Broker_ID
        - U ovom slučaju imamo samo jedan broker te nam je **broker_id** = 1

8.3.3. Kafka Connect

- Image
    - confluentinc/cp-kafka-connect:5.4.0
    - Confluent slika Kafka Connecta
- Container_name
    - kafka-connect
- Depends_on:
    - Zookeeper
    - Kafka
- Ports: 8083

Unutar varijabli okruženja imamo pojedine standardne vrijednosti koje su potrebne za pokretanje Kafka Connecta. Jedna od bitnijih za naš koncept je **CONNECT_PLUGIN_PATH**. U ovoj varijabli postavljamo putanju gdje ćemo spremati dodatke (engl. Pluginove) koje skinemo za Kafka Connect na lokalnoj mašini. Dodatak koji ćemo mi koristiti je JDBC Source Connector koji ćemo objasniti kasnije. Preuzimanje i instaliranje datoteka za instalaciju konektora izvodimo pomoću Linux naredbi koje unosimo u sekciju **command** za konfiguriranje Kafka Connecta.

8.3.4. MySQL

- Image
    - mysql
    - Službena slika MySQL servera
- Container_name
    - mysql
- Ports: 3306

Unutar varijabli okruženja definiramo lozinku za **root** korisnika te korisničko ime i lozinku za standardnog korisnika.

Pod sekciju **command** definiramo vremensku zonu MySQL servera kojoj mi pripadamo. Ovo je ključno kako bi nam servisi i MySQL server bili u istoj vremenskoj zoni.

Pod sekciju **volumes** se definira putanja na lokalnoj mašini ako želimo neke podatke vezano uz kontejner trajno pohraniti. U ovom slučaju spremat ćemo podatke koji se nalaze u bazama podataka koji se nalaze u MySQL serveru. Ovo je bitno u slučaju da želimo ugasiti kontejner od MySQL servera, a želimo da nam podaci u bazama podataka ostanu na lokalnoj mašini pri ponovnom pokretanju kontejnera, a ne da se izgube.

8.3.5. CRM

- Image
    - crm-view
    - Slika Laravel aplikacije konceptualnog CRM-a

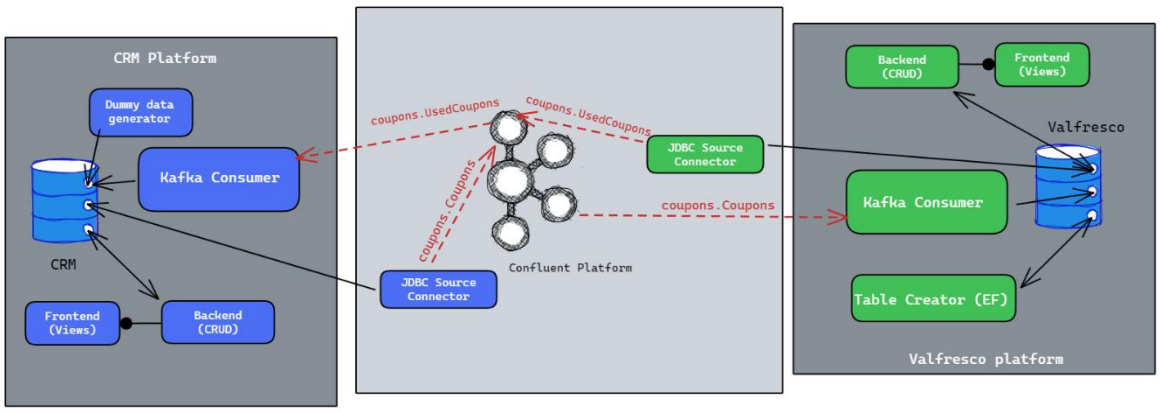
- Container_name
  - crm-view
- Ports
  - 8000

8.3.6. Valfresco

- Image
  - valfresco-view
  - Slika Laravel aplikacije konceptualnog Valfresca
- Container_name
  - alfresco-view
- Ports: 8001

Pri pokretanju **.yaml** datoteke dobiti ćemo Kafku platformu s GUI-em to jest Control Centrom, Kafka Connect za spajanje Kafke s vanjskim sistemima i MySQL server koji će nam biti potreban za izradu CRM i Valfresco baze. Unutar .yaml datoteke nalaze se i kontejneri za konceptualni CRM i Valfresco. Potrošači obje platforme biti će instalirani kao servis na virtualnom računalu pošto ne možemo pokrenuti potrošače prije nego što kreiramo **topic-e** s kojih mogu čitati poruke. Ako to napravimo potrošači će se pokrenuti s greškom.

8.4. Arhitektura visoke razine



Slika 7. High - level arhitektura

Arhitektura modela je podijeljena u tri dijela (sav kôd se nalazi na BitBucket privatnom repozitoriju):

- CRM platforma
  - Konceptualna CRM platforma koja se sastoji 5 dijela:
    - MySQL baza
    - Dummy data generator i Table Creator (EF)
      - Pisano u .NET 5.0
    - Laravel aplikacija za prikaz CRM-a
      - Sadrži backend i frontend
    - Kafka Consumer
      - Pisano u .NET 5.0
- Confluent platform
  - Dockerizirana Confluent platforma sa svojim komponentama:
    - Zookeeper
    - Kafka
    - Kafka Connect
      - JDBC Source Connector
    - Control center
- Valfresco platforma
  - Konceptualna Valfresco platforma koja se sastoji od 4 dijela:
    - Table Creator (EF)
      - Pisano u .NET 5.0
    - Laravel aplikacija za prikaz Valfresca
      - Sadrži backend i frontend
    - Kafka Consumer
      - Pisano u .NET 5.0

8.5. CRM platforma

8.5.1. MySQL baza

Baza CRM se sastoji 8 tablica:

- Coupons
  - Tablica koja sadrži podatke o vaučerima
- CouponStatus
  - Šifranik sa svim mogućim statusima vaučera
- CouponTypes
  - Šifranik sa svim mogućim tipovima vaučera
- CouponCampaigns
  - Osnovni podaci o kampanjama
  - Svaki vaučer mora pripadati nekoj kampanji
- Users
  - Standardna tablica s podacima o administrativnim korisnicima
- CouponUsed
  - Tablica u kojoj se bilježi svako korištenje vaučera
- ConsumerValfrescoProfiles
  - Tablica u kojoj se nalaze profili osoba koje su iskoristile vaučer na platformi Valfresco
  - Ova funkcionalnost ne postoji trenutno niti na pravoj platformi niti u konceptualnoj
  - Trenutna vrijednost ove tablice je ValfrescoWeb
  - Za svaki vaučer koji se iskoristi na platformi Valfresco kao ConsumerProfil poslat će se ValfrescoWeb
- _EFMigrationsHistory
  - Povijest migracija tablica
  - Entity Framework tablica

Slika 8. Relacijski dijagram tablica za vaučere u konceptualnoj CRM bazi

8.5.1.1.

8.5.2. Table Creator (EF)

Izradu tablica, njezinih stupaca, ključeva i relacije izrađujemo pomoću Entity Frameworka (u nastavku EF). EF je ORM[14] framework. Pomoću Entity Frameworka možemo izraditi tablice, primarne i vanjske ključeve te relacije između tablica pomoću programskog kôda u ovom slučaju C#. U osnovi definiramo model podataka: stupce u tablici i tip podatka (broj, tekst, datum...). Nakon toga definiramo relacije s ostalim tablicama (1:1, 1:M, M:1) pomoću programskih funkcija. Nakon toga možemo pozvati naredbu u terminalu **dotnet ef migrations add Naziv_migracije** te nam EF izradi skriptu za kreiranje strukture i relacije u bazi. Prije kreiranja migracije moramo definirati putanju do baze i autorizacijsku konfiguracija kako bi se program mogao spojiti na bazu. Moramo i odrediti tip baze (MySQL, SQL server, Postgres) kako bi EF znao koji tip skripte za migraciju mora izraditi.

Nakon toga pozovemo još jednu naredbu: **dotnet ef database update**. Ovom će se naredbom skripta izvršiti u bazi, te nam je baza podataka spremna za korištenje. Ovakvim načinom dobivamo lakši pristup nad uređivanjem baze jer se o tome brine programski kôd te ne moramo sami pisati skripte gdje dobivamo na uštedi vremena.

[14] Object–relational mapping. Programska tehnika koja nam omogućava oblikovanje strukture podataka pomoću objektno orijentirano programskog jezika [9].

8.5.3. Dummy data generator

Ovaj generator će nam služiti kako bismo izradili podatke u CRM-u. Izradit ćemo nekolicinu kampanja, vaučera. Testne podatke izradit ćemo pomoću .NET Core dodatka koji se zove Bogus. Bogus nam omogućava da za neki model podataka izradi nekolicinu testnih podataka pomoću par linija kôda. Ovo nam olakšava izradu testnih podataka jer za svako polje u modelu podataka možemo definirati kakve bismo podatke, količinu podataka te pomoću EF-a ih izradimo, ne ovisimo o ručnoj izradi podataka ili nekoj SQL skripti.

8.5.4. Laravel aplikacija

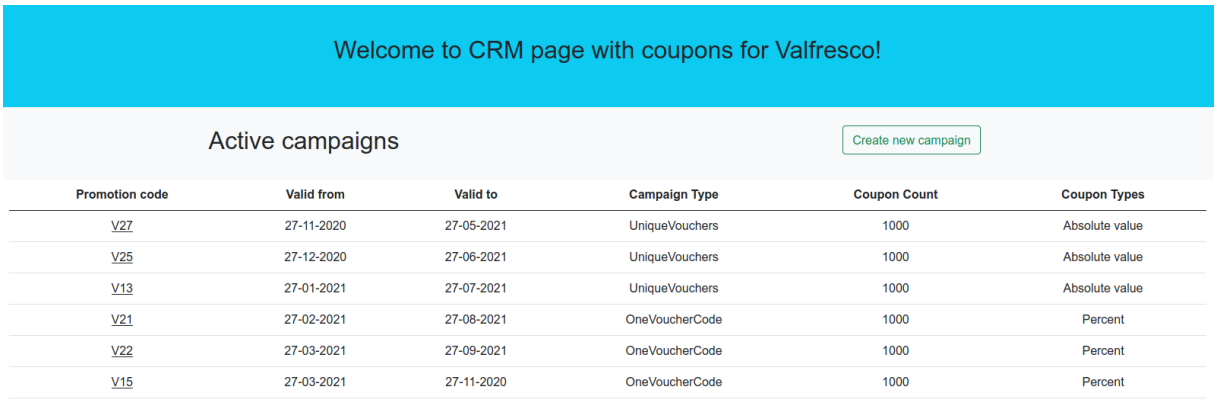
Laravel je PHP framework koji se primarno koristi za WEB aplikacije. Laravel je praktičan i developeri ga često koriste jer se može istovremeno razvijati frontend i backend aplikacije u jednom okruženju [10]. Jedna od prednosti Laravela je što ne zahtijeva mnogo programiranja za funkcionalnosti koje sadrži svaka WEB aplikacija: autorizacija, usmjeravanje (engl. routing), sesije, predmemoriranje (engl. caching) itd. Ove funkcionalnosti sam Laravel ima već izrađene te se mogu pozivati unutar koda ili dodatno konfigurirati.

Pošto se trenutni CRM nalazi unutar platforme Microsoft Dynamics mogućnosti za njegovim izmjena, kako na frontendu tako i na backendu, nisu moguće. Radi izrade ovoga koncepta CRM platforma je izrađena u Laravel-u radi jednostavnosti i brzine izrade koncepta. Ovaj CRM sadržava CRUD (Create, Read, Update, Delete) operacije nad vaučerima i njihovim kampanjama.

Laravel spada u softvere s MVC arhitekturom: Model, View, Controller. S obzirom da smo modele podataka izradili pomoću EF-a, modele podataka nećemo koristiti u Laravelu nego ćemo koristiti druge dvije funkcionalnosti: View i Controller. View je zapravo prikaz podataka kakve korisnik vidi unutar Internet preglednika. Za izradu Viewa Laravel ima svoj tip datoteka koje se zovu Blade. Blade nam je zapravo HTML, CSS i JS template u kojemu dinamički definiramo kako će nam se prikazivati podatci koje definiramo na backend strani. O tome koje podatke će View dobiti na prikaz, brine se Controller.

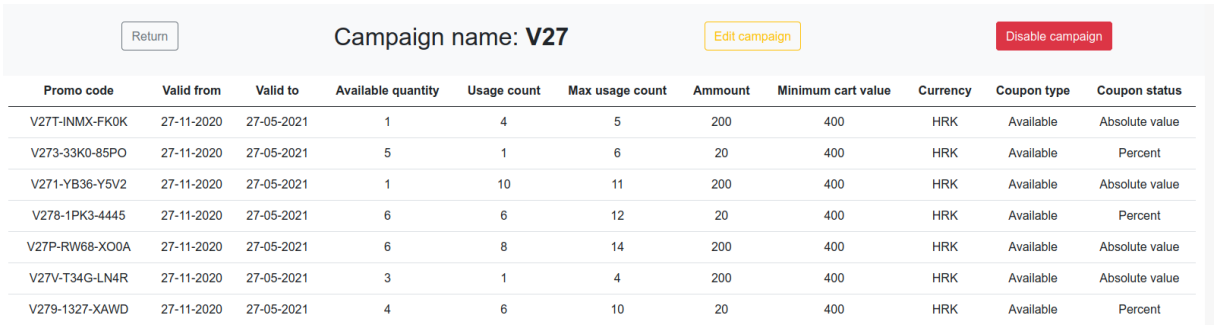
Controller nam je zapravo jedna od komponenti backend sustava. Unutar Controllera definiramo API-je koji će se pozivati na View-u. Logiku koju će nam sadržavati jedan API poziv npr. upit na bazu „daj mi sve vaučere iz XY kampanje“ nam je izrađena u jednoj od metoda koju definiramo unutar Controllera.

U našem konceptu imat ćemo jedan Controller u kojemu se nalaze sve metode CRUD operacije nad vaučerima i njihovim kampanja. Za prikaz vaučera i kampanja imat ćemo četiri Viewa: prikaz svih kampanja, prikaz vaučera unutar jedne kampanje, izmjena i unos kampanje.



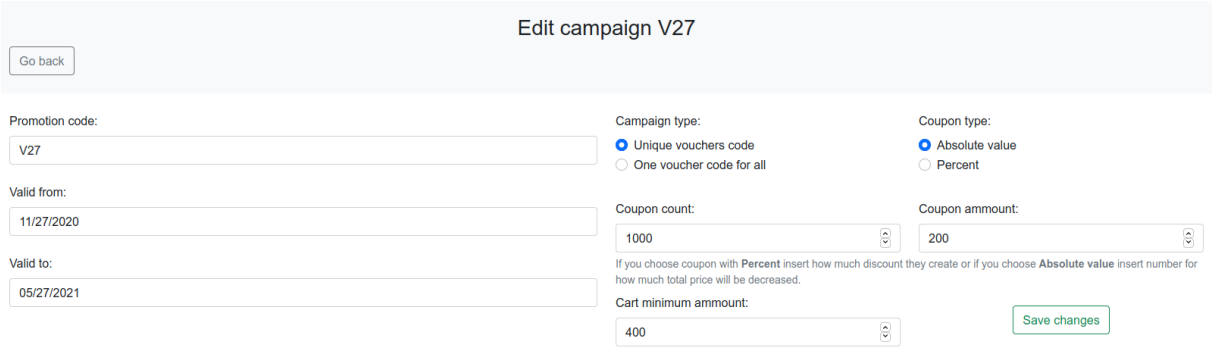
Promotion code	Valid from	Valid to	Campaign Type	Coupon Count	Coupon Types
V27	27-11-2020	27-05-2021	UniqueVouchers	1000	Absolute value
V25	27-12-2020	27-06-2021	UniqueVouchers	1000	Absolute value
V13	27-01-2021	27-07-2021	UniqueVouchers	1000	Absolute value
V21	27-02-2021	27-08-2021	OneVoucherCode	1000	Percent
V22	27-03-2021	27-09-2021	OneVoucherCode	1000	Percent
V15	27-03-2021	27-11-2020	OneVoucherCode	1000	Percent

Slika 9. Početna stranica CRM-a

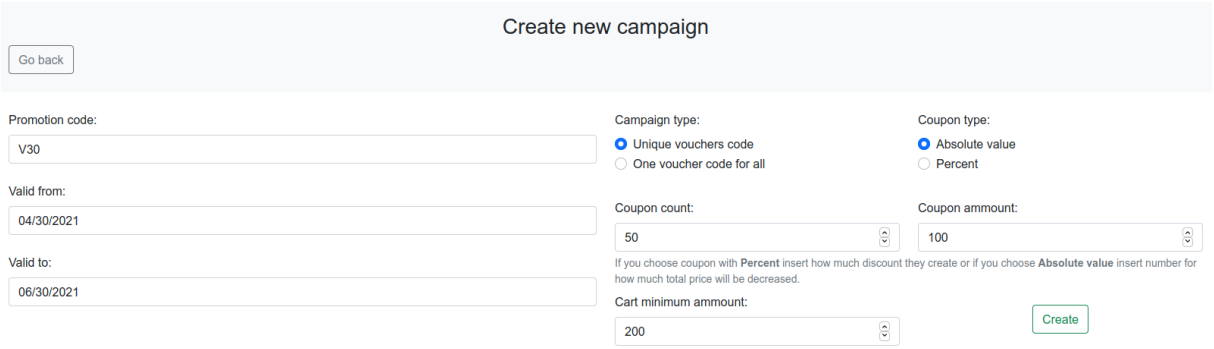


Promo code	Valid from	Valid to	Available quantity	Usage count	Max usage count	Ammount	Minimum cart value	Currency	Coupon type	Coupon status
V27T-INMX-FK0K	27-11-2020	27-05-2021	1	4	5	200	400	HRK	Available	Absolute value
V273-33K0-85PO	27-11-2020	27-05-2021	5	1	6	20	400	HRK	Available	Percent
V271-YB36-Y5V2	27-11-2020	27-05-2021	1	10	11	200	400	HRK	Available	Absolute value
V278-1PK3-4445	27-11-2020	27-05-2021	6	6	12	20	400	HRK	Available	Percent
V27P-RW68-XO0A	27-11-2020	27-05-2021	6	8	14	200	400	HRK	Available	Absolute value
V27V-T34G-LN4R	27-11-2020	27-05-2021	3	1	4	200	400	HRK	Available	Absolute value
V279-1327-XAWD	27-11-2020	27-05-2021	4	6	10	20	400	HRK	Available	Percent

Slika 10. Prikaz kampanja u CRM-u



Slika 11. Izmjena postojeće kampanje



Slika 12. Unos nove kampanje

Unutar datoteke web.php definiramo putanje (eng. Endpoint) za CRUD operacije:

- Prikaz svih kampanja
  - URL: /campaigns
  - Method: GET
  - Funkcija unutar Controllera: index
- Prikaz određene kampanje
  - URL: /campaigns/{name}
  - Method: GET
  - Funkcija unutar Controllera: show
- View za izradu kampanje
  - URL: /campaign/create
  - Method: GET
  - Funkcija unutar Controllera: createCampaign
- Izrada kampanje
  - URL: /campaign/create
  - Method: POST
  - Funkcija unutar Controllera: storeCampaign
- View za izmjenu kampanje
  - URL: /campaigns/{name}/edit
  - Method: GET
  - Funkcija unutar Controllera: editCampaign
- Izmjena kampanje
  - URL: /campaigns/{name}/edit
  - Method: PUT

    - Funkcija unutar Controllera: updateCampaign
- Deaktiviranje kampanje
    - URL: /campaigns/{name}/disable
    - Method: GET
    - Funkcija unutar Controllera: disableCampaign

Radi potreba koncepta izradili smo i dva tipa kampanje:

- Unique vouchers code
    - Kampanja gdje svaki vaučer ima samo jedno moguće korištenje to jest jedinstven je za svakog konzumenta kampanje
    - Ovo je konceptualni oblik kampanje 1 koje sada nalazimo u CRM-u
- One voucher code for all
    - Kampanja koja sadrži samo jedan vaučer kôd za sve konzumente kampanje
    - Broj korištenja vaučera se definira kao ukupni broj vaučera unutar kampanje
    - Ovo je konceptualni oblik kampanje 2 koje sada nalazimo u CRM-u

Unutar Controllera definiramo funkcije za određene URL i dodatne funkcije potrebne za izradu vaučera:

- Index
    - Upit na bazu za svim kampanja koje su aktivne i slanje podataka na View
- Show
    - Upit na bazu za odabranu kampanju na prethodnom View-u
- CreateCampaign
    - Pozivanje Viewa za izradu kampanje
    - Za izraditi kampanju s Viewa dobivamo sljedeće podatke
    - Promo kod kampanje
    - Datum početka trajanja kampanje
    - Datum završetka trajanja kampanje
    - Količina kupona u kampanji
    - Iznos kupona
    - Tip kupona
    - Tip kampanje
    - Minimalni iznos košarice kako bi se vaučer mogao iskoristiti
    - Pomoću dobivenih podataka pozivamo metodu koja izrađuje vaučere za tu kampanju ovisno o izabranom tipu kampanje
- StoreCampaign
    - Spremanje podataka dobivenih iz CreateCampaign-a
    - Izrada kampanje u bazi
- EditCampaign
    - Pozivanje View-a gdje se izmjenjuje odabrana kampanja na Viewu s svim kampanjama
- UpdateCampaign
    - Spremanje ažuriranih podataka o kampanji u bazu podataka
    - Pomoću novo dobivenih podataka ažuriramo kampanju i vaučere unutar te kampanje
    - Možemo imati sljedeće situacije:

        - Promjena tipa kampanje iz jedne u drugu
            - Kod ove situacije deaktiviramo sve vaučere u kampanji i izradimo nove vaučere
        - Broj vaučera u kampanji se smanjio
            - Vaučere koji nisu iskorišteni deaktiviramo
        - Broj vaučera se povećao
            - Izradimo nove vaučere kako bi nam nova količina vaučera u kampanji bila točna
        - Iznos vaučera se promijenio
            - Svim vaučerima izmijenimo iznos vaučera
        - Minimalni iznos košarice kako bi se vaučer mogao iskoristiti
            - Svim vaučerima izmijenimo minimalni iznos košarice
        - Datum trajanje kampanje se je izmijenio
            - Datum trajanje kampanje izmijenimo svim vaučerima
            - Ako je datum trajanje kampanje prošao, prebacimo sve vaučere u Expired status
        - Tip vaučera se je izmijenio
    - Svim vaučerima u kampanji izmijenimo tip
- DisableCampaign
    - Deaktiviranje kampanje u bazi podataka

Također, unutar CRM aplikacije napravili smo jedan proces koji se pokreće jednom dnevno. Laravel nudi planer zadataka (engl. Task Scheduler) unutar već postojećeg kôda. S obzirom na to da se u postojećem CRM-u vaučeri, kada im istekne datum trajanja, ne prebacuju automatski u status **Expired**, moramo izraditi proces koji će to raditi na dnevnoj bazi. Proces će nam jednom dnevno provjeriti imamo li vaučera koji su aktivni, također provjerava da status vaučera nije status **Expired** te kojima je datum trajanja istekao. Ako u bazi imamo takvih vaučera prebaci ih u status **Expired**. Da bismo ovo izradili moramo napraviti sljedeće:

- Metodu i upit na bazu koji provjerava imamo li takvih vaučera
- Pozvati metodu u Laravel-ovom **Task Scheduleru**

Nakon što izradimo gore navedene korake preostaje samo aktivirati Laravel-ov **Task Scheduler** pošto se on ne pokreće automatski.

8.5.5. CRM Consumer

CRM Consumer na CRM platformi primati će podatke o iskorištenosti vaučera na drugim platformama, u ovom slučaju platformi Valfresco. CRM Consumer je napisan kao .NET Console App. CRM Consumer i Table Creator se nalaze u istom programskom okruženju tako da već imamo spremne modele podataka za vaučere.

CRM Consumer je vrlo jednostavna aplikacija. Prije nego što pokrenemo potrošača moramo mu definirati konfiguracijske podatke:

- **GroupId**
  - Naziv potrošačke grupe kojoj pripada
  - Ovo nam je bitno ako imamo više potrošača koji čitaju s jednog **topic-a** kako bi nam Kafka mogla ravnomjerno raspodijeliti broj particija unutar **topic-a** koje dobije svaki potrošač unutar potrošačke grupe
  - U našem slučaju imamo jednu particiju te ćemo imati i samo jednog potrošača
- **BootstrapServers**
  - **URL** do Kafka Brokera
  - Potrošač mora znati **URL** kako bi znao gdje se nalazi Kafka posrednik
- **AutoOffsetReset**
  - Postavka kojom definiramo od koje poruke da nam potrošač počne čitati s **topic-a**
  - Definirana vrijednost: **AutoOffsetReset.Earliest**
    - Pomoću ove vrijednosti, ako se potrošač po prvi put spaja na **topic**, početak će nam čitati od prve poruke koja postoji u **topic-u**
    - Ako je potrošač već pročitao neke poruke u **topic-u**, početak će nam čitati od zadnje poruke koju je pročitao i nastaviti dalje
- **Subscribe**
  - Unutar ove funkcije definiramo s kojeg će **topic-a** potrošač čitati poruke
  - Prije uključivanja potrošača moramo biti sigurni da ovaj **topic** postoji jer će u protivnom potrošač javljati pogrešku da **topic** ne postoji

Nakon definiranja ovih podataka spremni smo za čitanje poruka s Kafka **topic-a**. CRM Consumer je zapravo beskonačna petlja koju mora korisnik prekinuti. Pomoću ove petlje potrošač ostaje uvijek upaljen i „stoji“ nad **topic-om** i čeka nove poruke. Unutar potrošača možemo definirati logiku što napraviti s porukom koju je pročitao. U našem slučaju potrošač sadrži logiku što napraviti kada dobije poruku o iskorištenosti nekoga vaučera.

Poruka dolazi u JSON formatu te sadrži sljedeće podatke:

Naziv polja	**Opis polja**
PromoCode	Promo kôd vaučera
CartAmmount	Iznos košarice
UsageDateTime	Timestamp iskorištenja vaučera
ConsumerProfileId	Guid korisnika koji je iskoristio vaučer. U ovom slučaju ova vrijednost je konstatna. Konstastno se šalje Guid korisnika ValfrescoWEB.

Tablica 6. Opis JSON-a u topicu coupon.CouponUsed

Nakon što pročitamo gore navedene podatke radimo sljedeće:

1. Provjeravamo postoji li vaučer
    a. Ako vaučer ne postoji sprema se zapis u tablicu **CouponUsed** i poruka da vaučer ne postoji
    b. Ako je vaučer istekao spremamo zapis u tablicu **CouponUsed** i poruku da je vaučer istekao
    c. Ako vaučer nema više slobodnih korištenja zapis se sprema u tablicu **CouponUsed** i poruku da kupon nema više mogućih korištenja
2. Ako vaučer postoji i zadovoljava gornje uvjete radimo sljedeće:
    a. Postojećem kuponu broj mogućih korištenja smanjujemo za 1
    b. Količinu korištenosti kupona povećavamo za 1
    c. Ako kupon nema više mogućnost korištenja prebacujemo ga u status **Used**
    d. Ažuriramo postojeći kupon u bazi
    e. U tablicu **CouponUsed** zapisujemo informaciju o iskorištenosti vaučera i poruku da je kupon uspješno iskorišten

8.6. Confluent platform

Confluent platforma [11] se sastoji od četiri Docker kontejnera za slijedeće stvari:

- Zookeeper
- Kafka
- Kafka Connect
- Control center

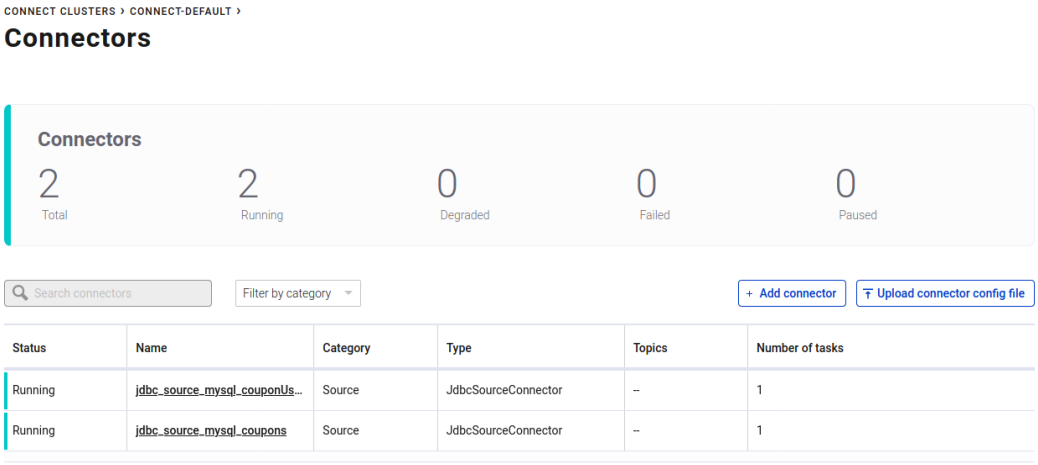
Ove elemente platforme već smo opisali u ranijim poglavljima. Rekli smo da za slanje podataka između Apache Kafke i ostalih sustava koristimo Kafka Connect, to jest njezine connectore. Kako bismo poslali podatke o vaučerima iz CRM-a u Apache Kafku koristit ćemo JDBC Connector. JDBC Connector ćemo koristiti i za slanje podataka o iskorištenosti vaučera na Valfresco strani prema CRM-u. Pošto je naša konceptualna baza CRM-a i Valfresca u MySQL-u koristit ćemo konkretno JDBC MySQL Connector.

Kako bismo mogli koristiti ovaj konektor moramo ga instalirati unutar Kafka Connect-a. Instalaciju izvršavamo u **docker-compose** datoteci. Servis **kafka-connect** u sekciji **volumes** moramo definirati gdje će nam se spremiti instalacija connectora. Connector spremamo na sljedećoj putanji: `/my/local/folder/with/jdbc-driver.jar:/usr/share/java/kafka-connect-jdbc/jars/.`

Nakon toga za servis **kafka-connect** pod sekciju **command** izvršavamo sljedeće naredbe:

- **/bin/bash**
- **-c**
- **|**
- **cd /usr/share/java/kafka-connect-jdbc/**
- **curl https://cdn.mysql.com/Downloads/Connector-J/mysql-connector-java-8.0.23.tar.gz | tar xz**
- **sleep infinity &**
- **/etc/confluent/docker/run**

Pri pokretanju **docker-compose** naredbe, kada se bude podigao Kafka Connect kao jedan od Docker kontejnera, usput će se instalirati i JDBC Connector za MySQL bazu.



Slika 13. Prikaz Kafka Connecta i Connectora u Control Centru

Konfiguraciju konektora možemo izvršiti na dva načina:

- Kroz Control center ili
- Pomoću Postman-a ili CURL-a kreirati API poziv za konfiguraciju konektora

Mi ćemo konfiguraciju izvršiti pomoću Postman-a. Za slanje API poziva imamo definirane sljedeće stvari:

- Method: POST
- URL: http://localhost:8083/connectors
- Body: JSON

JSON konfiguracija konektora nad CRM bazom izgleda sljedeće [12]:

```
{
        "name": "jdbc_source_mysql_coupons",
        "config": {
        "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
        "connection.url": "jdbc:mysql://mysql:3306/crm",
        "value.converter": "org.apache.kafka.connect.json.JsonConverter",
        "value.converter.schemas.enable": false,
        "connection.user": "root",
        "connection.password": "Admin123",
        "topic.prefix": "coupons-Coupons",
        "mode": "timestamp+incrementing",
        "incrementing.column.name": "Id",
        "query": "SELECT Id, GUID, VoucherPromotionCode, PromoCode, AvailableQuantity, UsageCount, MaxUsageCount, Ammount, ValidFrom, ValidTo, ExpiryDate, MinimumValueCurrency, CurrencyCode, CouponTypeCode, CouponStatusCode, Version, IsActive, IsDeleted, CreatedDate, ModifiedDate FROM crm.Coupons",
        "timestamp.column.name": "ModifiedDate, CreatedDate",
        "validate.non.null": "false",
        "poll.interval.ms": 1000
        }
}
```

JSON konfiguracija konektora nad Valfresco bazom izgleda sljedeće [12]:

```
{
        "name": "jdbc_source_mysql_couponUsed",
        "config": {
        "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
```

```
"connection.url": "jdbc:mysql://mysql:3306/valfresco",
"value.converter": "org.apache.kafka.connect.json.JsonConverter",
"value.converter.schemas.enable": false,
"connection.user": "root",
"connection.password": "Admin123",
"topic.prefix": "coupons-CouponUsed",
"mode": "incrementing",
"incrementing.column.name": "Id",
"query": "SELECT Id, GUID, PromoCode, CartAmmount, UsageDateTime, ConsumerProfileGuid FROM valfresco.CouponUsed",
"validate.non.null": "false",
"poll.interval.ms": 1000
}
}
```

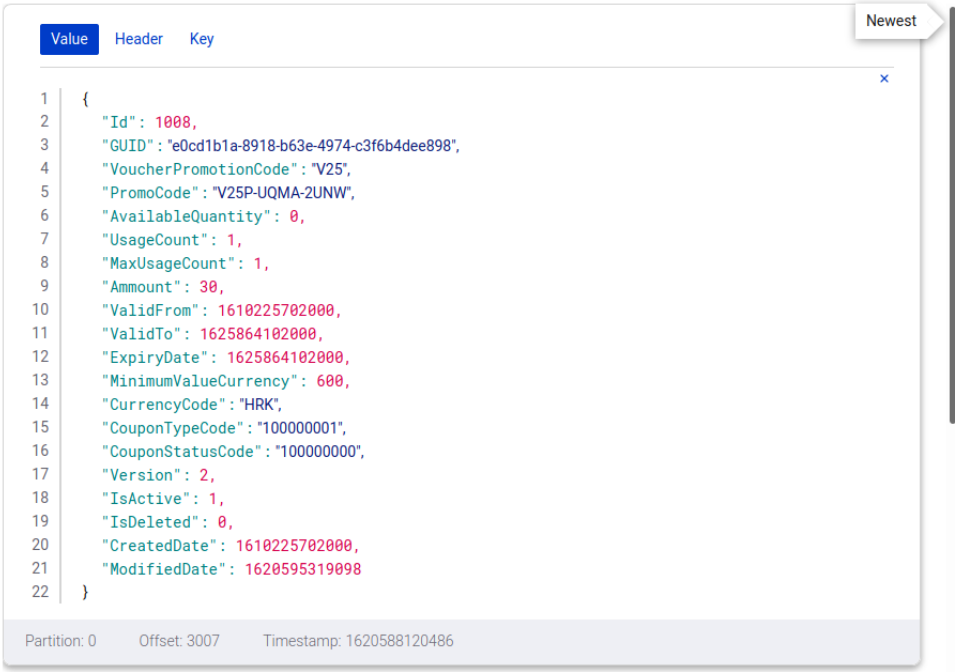
Naziv polja	Opis polja
Name	• Naziv konektora unutar Kafka Connecta • Sami definiramo naziv
Connector.class	• Klasa konektora kojoj konektor pripada
Connection.url	• URL do baze na koju konektor priključi • URL se definira u JDBC formatu • jdbc:tip_baze://url_MySQLServera/naziv_baze
Value.converter	• Definiramo u kojem formatu želimo poruku poslati u topic • org.apache.kafka.connect.json.JsonConverter definiramo da poruke dolaze u JSON formatu
value.converter.schemas.enable	• Definiramo da li želimo uz svaku poruku poslati shemu tablice iz koje dolazi poruka
table.whitelist	• Lista tablica u bazi za koje želimo da nam konektor šalje poruke u Apache Kafku • Ako ovo ne definiramo šalje poruke za sve tablice u bazi
connection.user	• Korisnik kojega konektor koristi za prijavu na bazu • Većinom je to SQL korisnik
connection.password	• Lozinka koju konektor koristi za prijavu u bazu
topic.prefix	• Prefiks naziva topica • Svaka tablica u bazi za koju želimo da se šalju podaci u Apache Kafku dobiva svoj vlastiti topic

	• Naziv topica se definira od kombinacije prefiksa naziva topica i naziva tablice
mode	• Način rada konektor • Konektor ima četiri načina rada • **Bulk** Ovim načinom rada konektor će svaki puta pročitati cijelu tablicu i cijelu je poslati u Kafka topic. Ovaj način rada nam nije pogodan jer ako imamo tablicu s velikim brojem redaka, svi reci će završiti u Kafka topicu. • **Incrementing** Ovim načinom konektor će poslati samo nove retke iz tablice u Kafka topic. Ovaj način rada nam je pogodan ako ćemo slati samo nove retke u Kafka topic. Ovim načinom ne možemo slati izmijenjene retke u Kafka topic. • **Timestamp** Konektoru je zadano timestamp polje kojim provjerava je li došlo do promjene u retku. Ako se polju izmijenila vrijednost, konektor će poslati redak u Kafka topic. Ovaj način rada je pogodan ako ćemo slati samo izmijenjene retke u tablici. Ovim načinom ne možemo slati nove retke u Kafka topic. • **Timestamp+incrementing** Kombinacija načina incrementing i timestamp. Konektor će poslati sve retke u tablici u Kafka topic ako su reci novi ili im se je timestamp polju izmijenila vrijednost. Ovim načinom rada možemo slati nove i izmijenjene retke u tablici.
Query	• SQL upit na bazu • Ako ne želimo cijelu tablicu sa svim poljima ili ako želimo spojiti podatke iz više tablica u jedan Kafka topic onda moramo izraditi SQL upit koji će to odraditi
Incrementing.column.name	• Ako koristimo incrementing način rada onda moramo postaviti koji stupac u tablici nam je inkrementirajući • Najčešće je to ID stupac

Timestamp.column.name	• Ako koristimo timestamp način rada moramo definirati polja nad kojima pratimo promjene u retku • U našem slučaju datum kreiranja retka i datum izmjene retka
Validate.non.null	• Ovime označavamo da li nam stupci iz definiranih vrijednosti u incrementing.column.name i timestamp.column.name mogu biti prazni u nekim recima • Ako mogu onda nam ovo polje dobiva vrijednost false
Poll.interval.ms	• Vremenska oznaka u milisekundama to jest nakon koliko vremena konektor provjerava u bazi podataka da li ima promijenjenih redaka • Ako postavimo ovo na 1000, to znači da će nam konektor provjeravati svaku sekundu da li je došlo do novih ili izmijenjenih redaka u tablici, ovisno koji način rada smo izabrali pod polje mode

Tablica 7. Opis JSON-a za kreiranje Connectora

Nakon što pokrenemo poziv dobivamo poruku o uspješnom pokretanju JDBC Connectora. Konektor istoga trena započinje s radom te kako je prvi put pokrenut pročitati će sve retke u tablici. Redci će završiti u definiranom Kafka **topic-u** i potrošači ih mogu početi konzumirati.



Slika 14. Prikaz poruke u topicu coupon.Coupon

Slika 15. Prikaz poruke u topicu coupon.CouponUsed

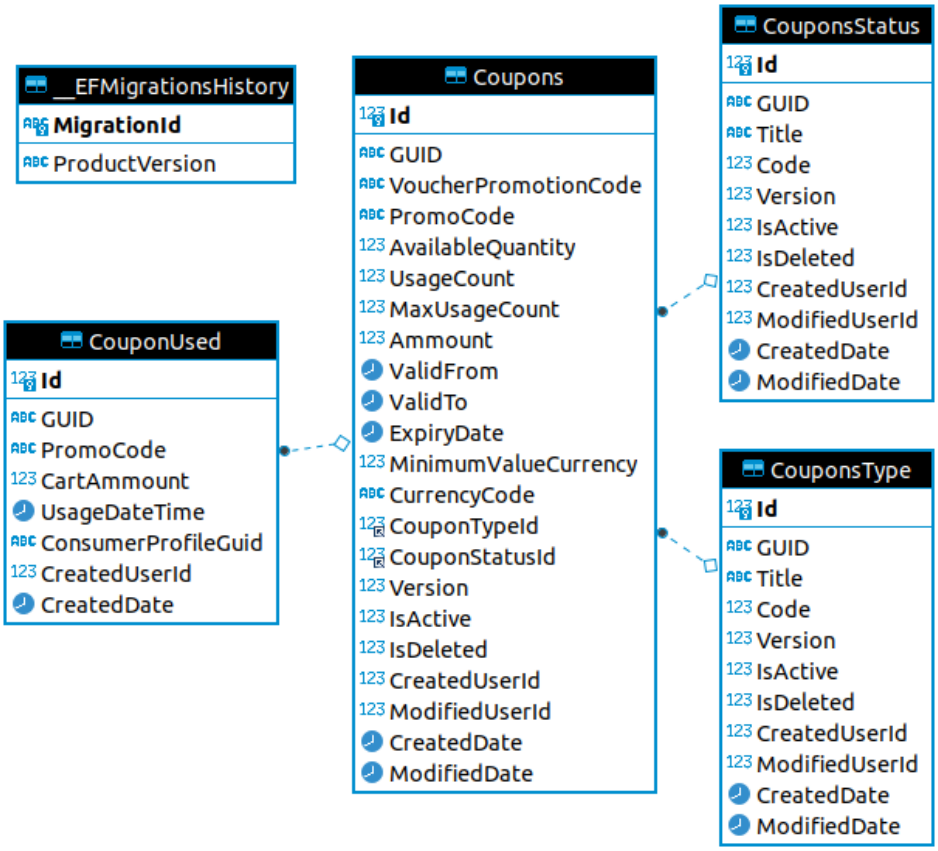
8.7. Konceptualna platforma Valfresco

8.7.1. MySQL baza i Table Creator (EF)

Valfresco baza se sastoji 4 tablice:

- Coupons
  - Tablica koja sadrži sve vaučere koji su došli iz CRM-a
- CouponStatus
  - Tablica sa svim statusima koje vaučer može imati
- CouponTypes
  - Tablica sa svim tipovima vaučera koji postoje u CRM-u
- CouponUsed
  - Tablica koja sadrži podatke o iskorištenosti vaučera

Izrada ovih tablica, primarnih i vanjskih ključeva te relacija među tablicama isto je određena pomoću okvira Entity Framework (EF). Pomoću EF-a smo kroz programski kôd izradili modele podataka, relacije i ključeve, nakon čega smo također pomoću EF naredbe izradili migracijsku skriptu i naredbom **dotnef ef database update** izradili tablice u MySQL bazi Valfresca. Za izradu baze Valfresco također smo morali definirati konfiguracijske varijable od MySQL baze kako bi se EF mogao spojiti na bazu i odraditi svoj proces (URL do baze, SQL korisnika i njegovu lozinku).



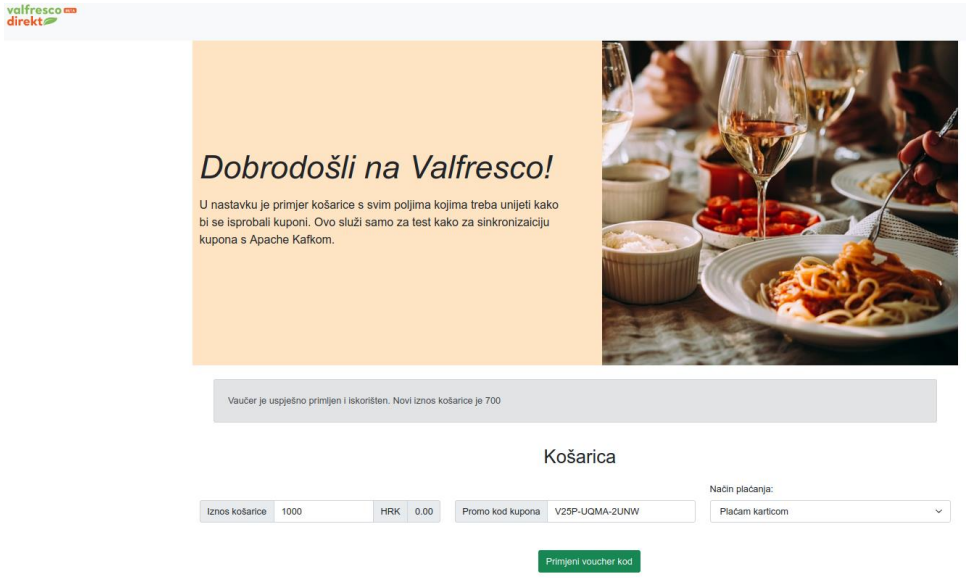
Tablica 8. Relacijski dijagram konceptualne baze platforme Valfresco

8.7.2. Laravel aplikacija

Konceptualni Valfresco će također biti izrađen pomoću Laravela. Za izradu koncepta koristit ćemo Laravel Controller i View. Unutar Controllera definirat ćemo dvije metode:

- **Index**
  - Metoda koja nam poziva View u browseru
  - View nam imitira košaricu na Valfrescu kako bismo mogli probno iskoristiti vaučere na Valfresco strani
  - Za testiranje vaučera moramo unijeti slijedeće podatke:
    - Iznos košarice (HRK)
    - Promo kôd vaučera
    - Način plaćanja: karticom ili pouzećem
  - Nakon unesenih podataka pritisnemo na gumb „Primjeni voucher kôd“ gdje će nam metoda sendCart tada provjeriti unesene podatke
- **sendCart**
  - unutar ove metode je napravljena logika koja se brine o korištenju vaučera
  - kako bi se iskoristio vaučer, vaučer mora zadovoljiti sljedeće uvjete:
    - Vaučer mora postojati u Valfresco bazi
      - Ako ne postoji, metoda vrati poruku na View: Uneseni vaučer ne postoji!
    - Vaučeru je AvailableQuantity > 1
      - Vaučer ima još mogućih korištenja
      - Ako uvjet nije zadovoljen, metoda vrati poruku na View: Uneseni vaučer je iskorišten maksimalan broj puta! Molimo Vas probajte unijeti drugi vaučer.
    - Vaučer nije istekao
      - ExpiryDate vaučera ne smije biti manji od današnjeg datuma
      - Ako je vaučer istekao, metoda vrati poruku na View: Uneseni vaučer je istekao! Molimo Vas probajte unijeti drugi vaučer.
    - Iznos košarice je veći ili jednak od dopuštenog
      - Iznos košarice je veći ili jednak od vrijednosti MinimumValueCurrency nad vaučerom
      - Ako je iznos košarice manji od dopuštene vrijednosti, metoda vrati poruku na View: Iznos košarice je manji od dozvoljenog. Za korištenje ovog vaučera iznos košarice mora biti minimalno $MinimumValueCurrency_vaučera.

Ako su svi gore navedeni uvjeti zadovoljeni metoda **sendCart** vraća sljedeću poruku: Vaučer je uspješno primljen i iskorišten. Novi iznos košarice je XY. XY nam je novi iznos košarice koji je umanjen za vrijednost vaučera.



Slika 16. Prikaz konceptualnog Valfresca

Istovremeno metoda spremi zapis o iskorištenosti vaučera u tablicu **CouponUsed**. Metoda spremi sljedeće podatke:

- Promo kod vaučera
- Iznos košarice
    - Iznos košarice je originalni iznos košarice, tj. iznos prije smanjenja pri primjenjivanju vaučera
- Timestamp kada je iskorišten vaučer
- **Guid** korisnika koji je iskoristio vaučer
    - Pošto trenutno imamo implementiranu samo mogućnost da imamo samo jednog korisnika, **ValfrescoWEB**, tada će nam se uvijek spremati **Id** od vrijednosti **ValfrescoWEB**

Nakon što se spremi podatak o iskorištenosti vaučera u bazu, JDBC Source Connector će preuzeti taj zapis iz baze i poslati u **topic** koji sadrži sve poruke o iskorištenosti vaučera. Nakon toga, ako je CRM Consumer uključen, potrošač će preuzeti poruku i obraditi je s logikom koju ima definiranu. Istovremeno kako će se zapis vaučera ažurirati u CRM bazi tako će JDBC Connector koji gleda na promjene vaučera u CRM bazi, preuzeti ažurirani vaučer i poslati ga u **topic** koji sadrži poruke o vaučerima. Nakon toga će Valfresco Consumer obraditi vaučer i ažurirati vaučer u Valfresco bazi.

8.7.3. Valfresco Consumer

Valfresco Consumer na platformi Valfresco primati će podatke o novim ili postojećim vaučerima na CRM platformi. Ako CRM izradi ili ažurira vaučere, oni će završiti u određenom Kafka **topic-u** te će ih ovaj potrošač obraditi i spremiti u Valfresco bazu. Valfresco Consumer je napisan kao .NET Console App. Kafka potrošač i **Table Creator** se nalaze u istom programskom okruženju tako da već imamo spremne modele podataka za vaučere.

Kafka Consumer je vrlo jednostavna aplikacija. Prije nego što pokrenemo potrošača moramo mu definirati konfiguracijske podatke:

- **GroupId**
  - Naziv potrošačke grupe kojoj pripada
  - Ovo nam je bitno ako imamo više potrošača koji čitaju s jednog **topic-a** kako bi nam Kafka mogla ravnomjerno raspodijeliti broj particija unutar **topic-a** koje dobije svaki potrošač unutar potrošačke grupe
  - U našem slučaju imamo jednu particiju te ćemo imati i samo jednog potrošača
- **BootstrapServers**
  - **URL** do Kafka posrednik
  - potrošač mora znati **URL** kako bi znao gdje se nalazi Kafka posrednik
- **AutoOffsetReset**
  - Postavka kojom definiramo od koje poruke da nam potrošač počne čitati s **topic-a**
  - Definirana vrijednost: **AutoOffsetReset.Earliest**
    - Pomoću ove vrijednosti ako nam se potrošač po prvi put spaja na **topic** počet će nam čitati od prve poruke koja postoji u **topic-u**
    - Ako nam je potrošač već pročitao neke poruke u **topic-u**, počet će nam čitati od zadnje poruke koju je pročitao i nastaviti dalje
- **Subscribe**
  - Unutar ove funkcije definiramo s kojega **topic-a** će potrošač čitati poruke
  - Prije uključivanja potrošač moramo biti sigurni da ovaj **topic** postoji jer inače će nam potrošač javljati pogrešku da **topic** ne postoji

Nakon definiranja ovih podataka spremni smo za čitanje poruka s Kafka **topica**. Valfresco Consumer nam je zapravo beskonačna petlja koju mora korisnik prekinuti. Pomoću ove petlje potrošač ostaje uvijek upaljen i „stoji“ nad **topic-om** i čeka nove poruke. Unutar potrošača možemo definirati logiku što ćemo napraviti s porukom koju je pročitao. U našem slučaju potrošač sadrži logiku što napraviti kada dobije poruku o iskorištenosti nekoga vaučera.

Poruka dolazi u JSON formatu te sadrži sljedeće podatke:

Naziv polja	**Opis polja**
Id	Id vaučera
GUID	GUID vaučera
VoucherPromotionCode	Kôd kampanje kojoj pripada vaučer
PromoCode	Promo kôd vaučera
AvailableQuantity	Koliko puta se još vaučer može iskoristiti
UsageCount	Koliko puta je vaučer iskorišten
MaxUsageCount	Maksimalan broj korištenja vaučera. Zbroj AvailableQuantity i UsageCount-a
Ammount	Iznos vaučera
ValidFrom	Datum početka kampanje
ValidTo	Datum završetka kampanje
ExpiryDate	Datum isteka vaučera

MinimumValueCurrency	Minimalni iznos košarice kako bi se vaučer mogao iskoristiti
CurrencyCode	Valuta iznos vaučera
CouponTypeCode	Šifra tipa vaučera
CouponStatusCode	Šifra statusa vaučera
Version	Verzija vaučera
CreatedDate	Datum izrade vaučera
ModifiedDate	Datum zadnje izmjene vaučera

Tablica 9. Opis JSON u topicu coupon.Coupon

Nakon što potrošač pročita poruku s Kafka **topic-a** uradi sljedeće:

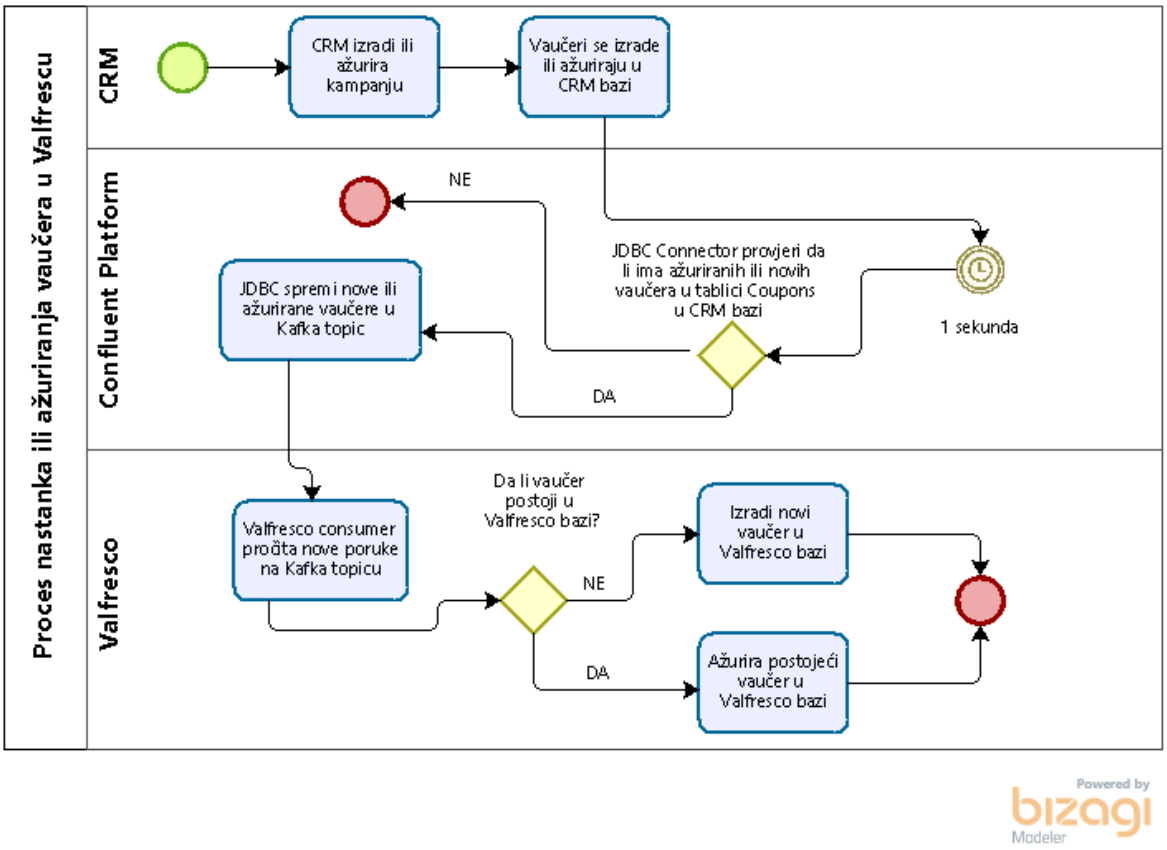
- Ako vaučer postoji u Valfresco bazi, ažurirati će ga s novim podacima koji su došli s Kafka **topica**
- Ako vaučer ne postoji u tablici izraditi će novi zapis u Valfresco bazi

8.8. Tok podataka

S novo izrađenim konceptom imamo dva toka podataka:

- CRM → Confluent Platform → Valfresco
- Valfresco → Confluent Platform → CRM

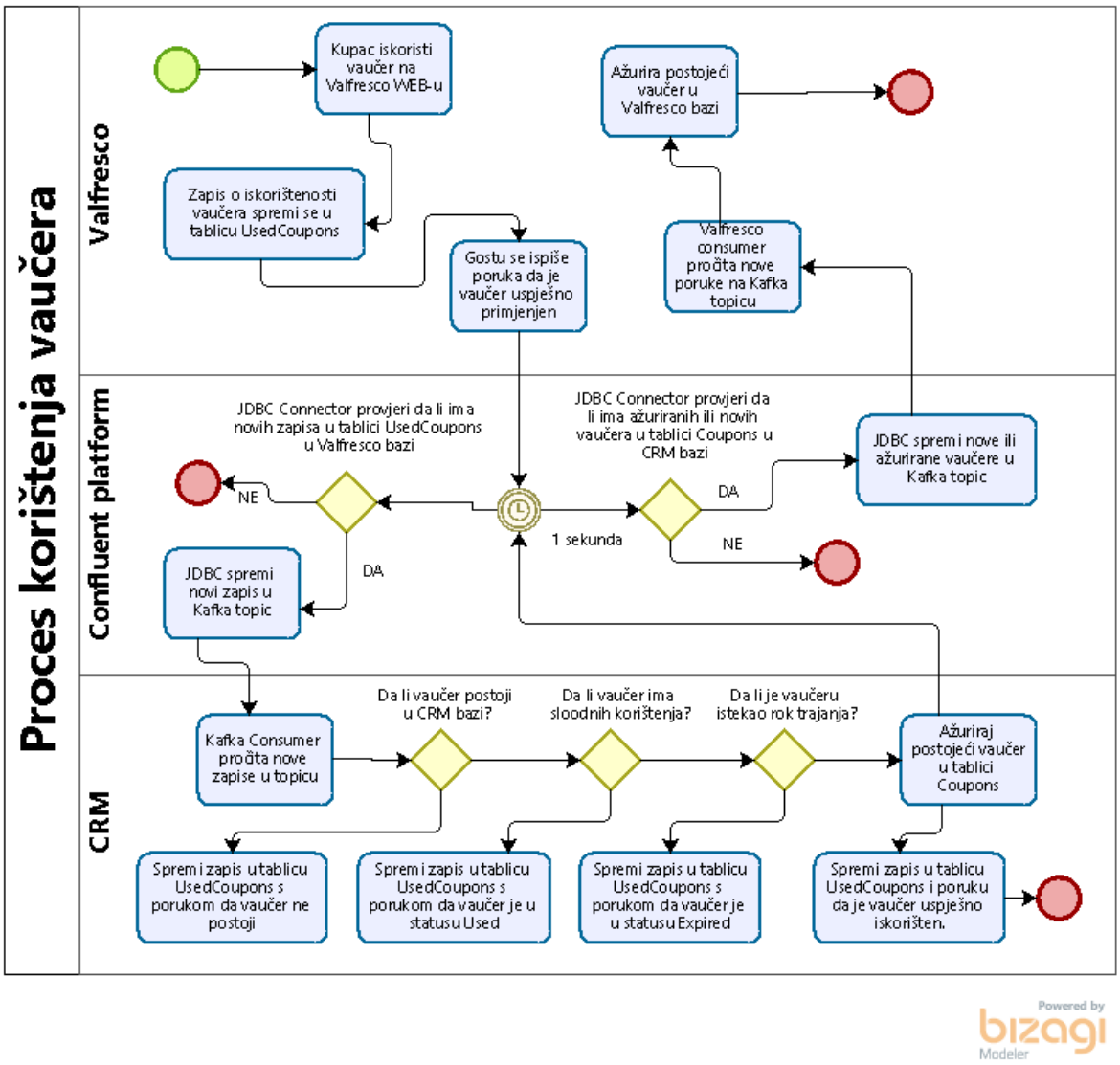
Prvi tok podataka nam kreće kada se izradi ili ažurira neki vaučer u CRM-u. JDBC Source Connector povuče nove ili ažurirane zapise o vaučerima iz CRM-a u Kafka **topic** te ih Valfresco Consumer, ako je aktivan, može odmah pročitati i spremiti u svoju bazu.



Slika 17. Dijagram toka podataka pri izradi ili ažuriranju vaučera

Drugi tok podataka nam se dešava kada kupac na Valfresco stranici iskoristi vaučer. Informacija o iskorištenosti vaučera se sprema u tablicu CouponUsed u Valfresco bazi. JDBC pročita nove zapise u toj tablici i spremi ih u Kafka **topic**. CRM-ov Consumer pročita nove poruke na Kafka **topic-u** te u CRM bazu spremi informaciju o iskorištenosti vaučera. Ako je vaučer moguće iskoristiti, potrošač će AvailableQuantity nad vaučerom smanjiti za 1, a UsageCount povećati za 1.

S obzirom na to da se vaučer ažurirao u CRM bazi, JDBC Connector koji stoji nad vaučer tablicom u CRM, poslat će ažurirani vaučer u Kafka **topic** kako bi Valfresco Consumer mogao ažurirati vaučer u Valfresco bazi.



Slika 18. Dijagram toka iskorištenosti vaučera

8.9. Prednosti koncepta

Konceptom smo dobili sljedeće stvari:

- Vaučeri su u skorom stvarnom vremenu ažurni u izvorištu podataka (CRM) i u sustavima kojima su ti podaci potrebni (Valfresco)
- Konzistentnost podataka s izvorišta podataka
- Trajno skladište podataka o vaučerima
- Ispravljena logika oko korištenja vaučera

Nakon što smo prošli kroz kratki opis koncepta, izložit ćemo koje smo nedostatke i probleme riješili u vezi tim konceptom. Trenutni proces sinkronizacije vaučera ne postoji tako da smo samim konceptom postigli primarni cilj, a to je sinkronizacija vaučera iz CRM-a s Valfrescom i obrnuto.

Prvi nedostatak kod CRM-a je bio taj da pri izradi jednoga tipa kampanje, svi vaučeri iz te kampanje ne dobivaju **ExpiryDate**. Novi koncept CRM-a dodjeljuje svakom vaučeru njegov **ExpiryDate** neovisno o tipu kampanje.

Drugi nedostatak kod CRM-a nam nastaje kada vaučeri pređu u **Expired** stanje. Tada na vaučeru datum **ExpiryDate** postane stariji od današnjeg datuma, a njegov status u CRM-u se ne prebaci u status **Expired**. Razlog tomu je što u CRM-u ne postoji neki proces koji prebacuje vaučere u status **Expired**. Koncept CRM-a unutar Laravel aplikacije sadrži planirani proces koji jednom dnevno provjeri ima li vaučera s **ExpiryDate** starijim od današnjeg datuma. Ako ima, prebaci ih u status **Expired**. Istovremeno se vaučeri ažuriraju u novi status i promjena se pošalje na Kafka **topic**. Također, ovim rješenjem smo otklonili i nedostatak 4, tj. situaciju kada **GetVoucherInfo** nije javljao da je vaučer istekao već nam je tu informaciju javio tek API **CanUseVoucher**. Novim konceptom ne moramo više pitati CRM da li nam je vaučer istekao, već će se promjena automatski izvršiti i na Valfresco strani.

Tip kampanje kod koje svaki vaučer ima maksimalno jedno korištenje, to jest svaki vaučer ima jedinstveni promo kôd, CRM-ov API **GetVoucherInfo** za te vaučere nije slao **MaxUsageCount**, odnosno broj maksimalnih korištenja vaučera. Treći nedostatak je riješen novim konceptom jer novi koncept za svaki vaučer u bilo kojoj kampanji između ostalog šalje i **MaxUsageCount**.

Također, s ovim principom rada Valfresco se više ne mora brinuti o ažuriranju količine moguće iskorištenosti vaučera u njihovoj bazi. Trenutno Valfresco na svojoj platformi ima napravljen proces računanja korištenosti vaučera. Ovim konceptom iz CRM-a dolaze tri podatka o korištenju vaučera:

- **AvailableQuantity** → koliko puta se još može iskoristiti vaučer
- **UsageCount** → koliko puta je iskorišten vaučer
- **MaxUsageCount** → koliko puta se može maksimalno vaučer iskoristiti

CRM API **GetVoucherInfo** kao informaciju šalje samo **UsageCount** i **MaxUsageCount**, dok se **AvailableQuantity** računao na Valfresco strani. Dodavanjem polja **AvailableQuantity** u CRM-u izbjegli smo računanje koliko se puta još vaučer može iskoristiti. Valfrescu preostaje samo postaviti logiku oko korištenja vaučera da mu je validna:

- Vaučer postoji u bazi
- Vaučer nije istekao

- Vaučer ima slobodnih korištenja
- Iznos košarice je veći ili jednako od dopuštenog za korištenje vaučera

Što se tiče statusa vaučera, CRM-ova logika oko toga je trenutno nejasna. Nejasan je dio oko statusa **Used**. Vaučer može biti u statusu **Used** ili ako nema više slobodnih korištenja ili ako je vaučer barem jednom iskorišten. To znači da status **Available** vrijedi samo ako vaučer nije ni jednom iskorišten. Novim konceptom je ova logika izmijenjena. Vaučer je u statusu **Used** samo ako nema više slobodnih korištenja. Ako se vaučer još uvijek može iskoristiti, to jest ako je **AvailableQuantity** veći ili jednak 1 onda je vaučer u statusu **Available**.

Valfresco je prije svako korištenje vaučera morao poslati prema CRM-u API-em **UseVoucher**. Često se dešavalo da ovakav API ne prolazi zato što bi vaučer došao bez **ExpiryDate**, a API nije mogao prihvatiti takav vaučer jer on nije bio validan. Također Valfresco nije imao nikakvu kontrolu što napraviti ako CRM ne prihvati korištenje vaučera. S novim konceptom se Valfresco o tome ne mora više brinuti. Poruke o iskorištenosti vaučera se šalju u Kafka **topic** i CRM **Consumer** ih obradi i sprema kod sebe. Prema tome, ukoliko se dogodi neka pogreška, primjerice da je vaučer krivo obrađen ili sl., CRM tu informaciju ima kod sebe preko tablice **CouponUsed** te može napraviti neki frontend prikaz gdje bi mogli vidjeti sva korištenja vaučera koja nisu uspješno odrađena.

Ovakvim pristupom sinkronizacije, ali i integracije između dva sustava izbjegli smo korištenje API-a s obje strane. Svaki sustav ima svoj mikro servis koji se brine o primanju poruka i logici oko spremanja poruka u sustav, dok se u Kafki nalaze sve poruke između dva sustava. Ovim principom smo dobili i rješenje problema u slučaju da nam jedan od sustava ne radi. Trenutnom metodom, npr. ako CRM API-ji nisu radili, a korisnik je htio iskoristiti vaučer koji još uvijek ne postoji u Valfresco bazi jer nije bio nijednom iskorišten u Valfrescu, Valfresco ne može pitati pomoću API-a postoji li ovaj vaučer jer CRM API ne radi. Valfresco više ne mora pitati CRM za nove vaučere jer čim se izrade preko Kafke Valfresco Consumer ih može preuzeti.

Ovime smo dobili na sinkronizaciji vaučera iz CRM-a prema Valfrescu, ali u isto vrijeme dobili smo i ažuriranje vaučera iz Valfresca prema CRM-u. Kafka se bavi slanjem poruka između dva sustava, dok dva sustava koji komuniciraju imaju dva mikro servisa koji se brinu o primanju poruka iz Kafke i spremanju poruka u svoje baze. Kafka Connect se brine o tome da svaki novi ili ažurirani vaučer odmah završi u Kafka **topic-u** kako bi potrošač mogao imati najsvježije informacije. S druge strane također brine da svako korištenje vaučera u stvarnom vremenu završi u Kafki, nakon čega CRM Consumer obradi poruku i spremi određene podatke kod sebe.

9. Zaključak

Ovaj rad prikazuje kako integracija između dva sustava može unaprijediti cjelokupni sustav, pri čemu same integracije postaju puno jednostavnije i brže, a ne više duge, mukotrpne i skupe.

Dosadašnja je praksa kod integracija bila takva da je svaki sustav morao pitati drugi sustav za nove ili ažurirane podatke te ih onda spremati kod sebe. S Apache Kafkom i njezinim konektorima ovo više nije slučaj. U Kafka Connectu možemo izraditi konektor koji će čitati određenu grupu podataka, to jest podatke koji trebaju drugom sustavu, nakon čega ih on proslijedi u **topic** i drugi sustav ih pomoću svojega mikroservisa obradi i spremi.

Ovakvim principom izbjegavamo da sustav kojemu su potrebni podaci svako određeno vrijeme pita drugi sustav za promjene, već on sam dobiva promjene čim se one dogode u tom sustavu. Također ovime smo izbjegli izradu i održavanje API-ja jer API-ji nam sada više nisu potrebni. Izradimo **topic-e** koji su potrebni drugom sustavu i on sve potrebne podatke dobiva s njih.

Smatram da će se ubuduće integracije sustava raditi po ovome principu jer je jednostavniji, brži, skalabilniji i sigurniji. Uvođenjem Kafke između dva sustava osiguravamo da će svi podaci doći do drugih sustava, odnosno osiguravamo da se podaci neće izgubiti ako dođe do pada jednoga od sustava.

Još jednu od prednosti za ovakav način izgradnje platformi vidjeli smo kroz kontejnere. Kontejneri omogućavaju bržu konfiguraciju određene platforme i servisa te brže podizanje platforme ili servisa online. Također, prije su se mnogi sustavi gradili kao monoliti, to jest sve funkcionalnosti platforme bile su spremljene u jedno programsko okruženje.

Današnjim načinom izrade platforme težište se stavlja prema tome da se platforme podijele u manje funkcionalne cjeline, to jest mikro servise. S mikroservisima dobijemo puno jednostavnije održavanje sustava jer možemo sustav izgraditi na način da je skup nekih procesa u jednom mikroservisu, a skup drugi procesa u drugom mikroservisu. Time dobijemo na neovisnosti procesa o drugim procesima, to jest možemo razvijati nove procese ili ažurirati postojeće s većom sigurnosti da ostale procese ne remetimo.

Kontejner je pojednostavio cijeli proces razvoja, testiranja i podizanja ovakvih mikroservisa online. Nekada su developeri sustava više vremena trošili na konfiguriranje i podizanje sustava nego na razvoj. S kontejnerima ovo više nije slučaj.

Predloženi konceptualni model je koncept na kojemu možemo vidjeti u kojem smjeru bi trebala ići sinkronizacija vaučera između dva sustava. Ovaj model je dobra podloga za daljnje razvijanje novih funkcionalnosti, također može poslužiti kao primjer gdje se još može Apache Kafka iskoristiti za bržu i jednostavniju integraciju.

10. Literatura

[1] Luka Otočan, 'Upravljanje odnosima s klijentima (Customer Relationship Managment, CRM)', Sveučilište u Rijeci - Odjel za informatiku, Rijeka, 2019.

[2] 'Što je CRM i što se iza njega krije', Jan. 28, 2016. https://www.poslovni.hr/lifestyle/sto-je-crm-i-sto-se-iza-njega-krije-307951

[3] Hazelcast, 'Event Stream Processing'. https://hazelcast.com/glossary/event-stream-processing/

[4] N. Narkhede, G. Shapira, and T. Palino, *Kafka: the definitive guide: real-time data and stream processing at scale*, First edition. Sebastopol, CA: O'Reilly Media, 2017.

[5] *Quick Start for Apache Kafka using Confluent Platform (Docker)*. Confluent. [Online]. Available: https://docs.confluent.io/platform/current/quickstart/ce-docker-quickstart.html

[6] *Docker*. [Online]. Available: https://www.docker.com/

[7] Aleksandar Skendžić, 'Docker kontejnerska virtualizacija', Feb. 06, 2019. [Online]. Available: https://www.vidi.hr/Racunala/Novosti/Docker-kontejnerska-virtualizacija

[8] Sofija Simic, 'Docker Image vs Container: The Major Differences', Oct. 31, 2019. https://phoenixnap.com/kb/docker-image-vs-container

[9] 'Hibernate - ORM Overview'. [Online]. Available: https://www.tutorialspoint.com/hibernate/orm_overview.htm

[10] *The PHP Framework for Web Artisans*. [Online]. Available: https://laravel.com/

[11] Robin Moffatt, *Demo-scene*. 2019. [Online]. Available: https://github.com/confluentinc/demo-scene

[12] Robin Moffatt, 'Kafka Connect Deep Dive – JDBC Source Connector', Feb. 12, 2019. [Online]. Available: https://www.confluent.io/blog/kafka-connect-deep-dive-jdbc-source-connector/

11. Popis tablica

12. Popis slika

13. Prilozi

a. Link na web stranicu Platforma za e-poslovanje platforme Valfresco: https://www.valfresco.com/
b. Link na konceptualni Valfresco: http://134.209.238.191:8001/
c. Link na konceptualni CRM: http://134.209.238.191:8002/
d. Link na konceptualni Control Center: http://134.209.238.191:9021/
e. Link do koda cijeloga koncepta: Repozitorij je privatan, treba zatražiti zahtjev