VARAUSJONON TAUSTAKÄSITTELY DOMAIN-TAPAHTUMIEN AVULLA

Kristian Toivonen

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2024

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Kristian Toivonen	**Vuosi**	2024
Ohjaaja	Toni Westerlund		
Toimeksiantaja	Sanoma Media Finland		
Työn nimi	Varausjonon taustakäsittely Domain-tapahtumien avulla		
Sivumäärä	28		

Opinnäytetyön keskeisenä tavoitteena oli pyrkiä parantamaan varausjärjestelmän luotettavuutta. Tämä pyrittiin saavuttamaan integroimalla toteutukseen domain-tapahtumia, jotka hallitsivat monimutkaista ja pitkää asynkronista prosessia.

Opinnäytetyössä tutkittiin puhtaan arkkitehtuurin periaatteita ja domain-tapahtumia, joiden avulla pyrittiin kehittämään parannusehdotus varausjonolle. Työssä keskityttiin puhtaan arkkitehtuurin keskeisten periaatteiden läpikäyntiin sekä perehdyttiin käsitteeseen domain-tapahtuma.

Tutkimuksen ja kehitystyön tuloksena luotiin geneerinen hallinta domain-tapahtumille, mikä tarjosi ratkaisun varausjonon parannusehdotukseen. Täydentäen tätä kehitettiin taustapalvelu, jonka tehtävänä oli käsitellä ja välittää domain-tapahtumia. Käytettyjen MediatR- ja Rebus-kirjastojen avulla pystyttiin tehokkaasti käsittelemään tapahtumia ja niiden välitystä.

Avainsanat C#, domain-tapahtumat, puhdas arkkitehtuuri

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Study Programme in Information
and Communication Technology
Bachelor of Engineering

Author	Kristian Toivonen	**Year**	2024
Supervisor	Toni Westerlund		
Commissioned by	Sanoma Media Finland		
Title	Booking Queue Background Processing by Using Domain Events		
Number of pages	28		

The main objective of this thesis study was to try to improve the reliability of the commissioner's booking system, which was attempted by integrating domain events into the implementation controlling the complex and long asynchronous process.

The principles of clean architecture and domain events were investigated with the aim to develop a proposal for improving the booking queue. The focus of the study was on going through the key principles of clean architecture and looking into the concept of domain events.

As a result of the study and development work, a generic management system for domain events was created, providing a solution for the improvement proposal for the booking queue. In addition to this, a background service was developed whose task was to handle and transmit domain events. The use of the MediatR and Rebus libraries allowed efficient handling and transmission of events. Utilizing these libraries was crucial in ensuring the scalability of the system.

Keywords: C#, clean architecture, domain events

SISÄLLYS

1 JOHDANTO 5
2 PUHDAS ARKKITEHTUURI 6
2.1 Yleisesti ohjelmistoarkkitehtuurista 6
2.2 Puhtaan arkkitehtuurin rakenne 6
2.2.1 Entiteetit 7
2.2.2 Käyttötapaukset 8
2.2.3 Liitäntäsovittimet 9
2.2.4 Kehykset ja ajurit 9
3 DOMAIN-TAPAHTUMAT 11
3.1 Domain-tapahtumien määrittely 11
3.2 Domain-tapahtumien suunnittelu 12
3.3 Domain-tapahtumien käsittely 12
3.4 Domain-tapahtumien nimeäminen 14
4 KEHITYSTYÖ 15
4.1 Käytetyt teknologiat 15
4.1.1 C# 15
4.1.2 PostgreSQL 16
4.2 Alkumäärittely 17
4.3 Suunnittelu 18
4.4 Toteutus 19
4.4.1 Domain-tapahtumien implementointi ja käsittely 20
4.4.2 Taustapalvelu 22
4.4.3 Testaus 23
4.5 Jatkokehitys 24
5 POHDINTA 25
LÄHTEET 27

1 JOHDANTO

Opinnäytetyön tavoitteena on luoda parannusehdotus, joka hyödyntää domain-tapahtumia taustakäsittelyssä, tarjoten tehokkaamman ja luotettavamman tavan käsitellä vaativia kirjoitusoperaatioita. Työssä tarkastellaan nykyistä taustakäsittelyä API:ssa ja kehitetään ratkaisu, joka pyrkii vastaamaan API:n suorituskyvyn parantamisen tarpeisiin. Työhön sisältyy koodimuutoksia C#-pohjaisessa API-projektissa, ja toteutuksen vaatimuksena on, että se toimii API:n päästä päähän testauksessa ja pystyy käsittelemään uudelleenyritykseen epäonnistuneita kirjoitusoperaatioita.

API:n suorituskyvyn parantaminen on välttämätöntä, jotta asiakkaille voidaan tarjota tehokas ja luotettava palvelu. Palvelun käytön lisääntyminen ja kasvava kuorma vaativat uusia ratkaisuja ja lähestymistapoja.

Työssä tutkitaan puhdasta arkkitehtuuria ja domain-tapahtuma käsitettä. Tämän pohjalta luodaan konseptitodistus (*Proof Of Concept eli POC*), jossa domain-tapahtumia hyödynnetään rajapinnassa. Tämä prosessi mahdollistaa käytännön kokeilun ja arvioinnin uuden lähestymistavan toimivuudesta.

Tämä on monimutkainen haaste, joka edellyttää tietoa ja taitoja C#-ohjelmoinnista, DevOps-käytännöistä, Domain-driven designista ja taustakäsittelyprosessista. Haasteena on myös Salesforcen määrittelemät rajoitukset. Kaikki kirjoitusprosessit on järjestettävä jonoon, jotta vältetään Advendio-tilauksen ylittävät rajat. Tämä on välttämätöntä, sillä liian monta samanaikaista pyyntöä aiheuttaa ongelmia ja yhteyskatkoksia API:n ja Salesforcen välillä.

Opinnäytetyön tilaajana toimii Sanoma Media Finland, Suomen johtava monimedialiiketoiminnan tarjoaja. Yhtiö tarjoaa laajan valikoiman erilaisia medioita ja palveluita, jotka kattavat uutiset, viihteen, mainonnan ja muut sisältöalueet. Sanoma Media Finlandin tuotteisiin kuuluvat sanomalehdet, aikakauslehdet, verkkosivustot, mobiilisovellukset sekä erilaiset sisältöpalvelut. (Sanoma 2023.)

2 PUHDAS ARKKITEHTUURI

2.1 Yleisesti ohjelmistoarkkitehtuurista

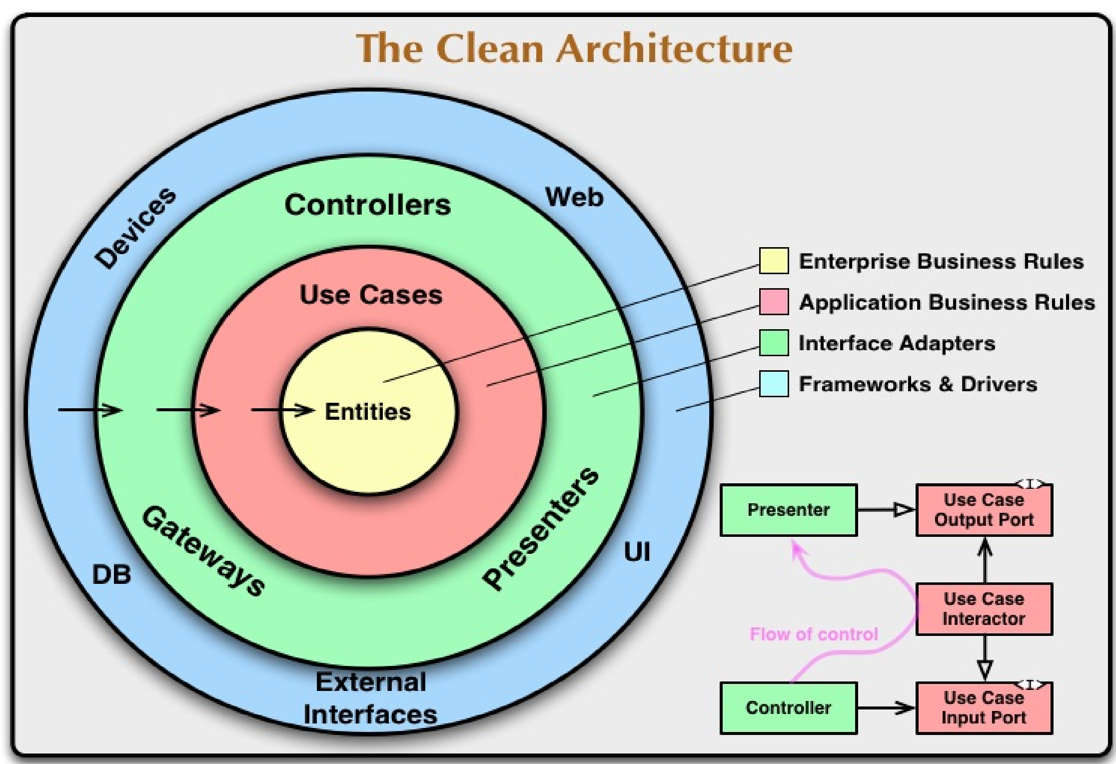
Ohjelmistoarkkitehtuuri on suunnitelma tai ohjeistus siitä, miten ohjelmiston osat tulee järjestää, jotta ohjelmisto olisi ymmärrettävä, ylläpidettävä ja laajennettava. Ohjelmistoarkkitehtuurin pääperiaate on yksinkertaistaa sovelluksen rakenne, sillä ohjelmiston ikääntyessä ja kasvaessa sen hallinnasta tulee haastavaa. Yksinkertainen ja selkeä arkkitehtuuri voi merkittävästi parantaa ohjelmiston laatua ja ylläpidettävyyttä. (Martin 2008, kappale 1.)

Puhdas arkkitehtuuri on ohjelmistoarkkitehtuurin tyyppi, joka on suunniteltu pitämään ohjelmiston liiketoimintalogiikka erillään ulkoisista tekijöistä, kuten käyttöliittymästä, tietokannasta tai ulkoisista rajapinnoista. Tämä tekee ohjelmistosta joustavamman ja helpommin ylläpidettävän, koska muutokset yhdessä osassa eivät vaikuta muihin osiin. Puhdasta arkkitehtuuria käytetään useista syistä, kuten huolenaiheiden erottamiseen, testattavuuden parantamiseen, ylläpidettävyyden parantamiseen sekä joustavuuden ja skaalautuvuuden parantamiseen. (Jovanovic 2022.)

2.2 Puhtaan arkkitehtuurin rakenne

Vuonna 2012 Robert C. Martin, esitteli Puhtaan arkkitehtuurin -konseptin (*Clean Architecture*) omassa blogissaan. Arkkitehtuurimallin idea on jakaa sovellus kerroksiin, jossa jokaisella kerroksella on oma tehtävänsä. Tämä tarjoaa lähestymistavan ohjelmiston suunnitteluun ja rakentamiseen, jossa keskitytään komponenttien selkeyteen, ylläpidettävyyteen ja riippuvuuksien hallintaan. (Martin 2012.)

Kuviossa 1 sovellus on jaettu neljään eri kerrokseen, mutta kerroksien määrä ei ole vakio, vaan voi mukautua sovelluksen tarpeisiin. Kerroksia voi tarvita enemmän, ja niitä voi lisätä jakamalla olemassa olevia kerroksia pienempiin kokonaisuuksiin ja luomalla niille tarkempia rajauksia. Myös uusien osioiden luominen on mahdollista, mikäli se tukee sovelluksen kokonaisrakennetta. (Martin 2012.)



Kuvio 1. Robert C. Martinin idea puhtaan arkkitehtuurin rakenteesta (Martin 2012)

Puhtaan arkkitehtuurin alkuperäisen konseptin esittelyn jälkeen se on kehittynyt ja ansainnut laajan hyväksynnän ohjelmistokehitysyhteisössä. Nykypäivänä se on suosittu tapa suunnitella ohjelmistoja. Nykyään puhtaan arkkitehtuurin käytön yhteydessä usein hyödynnetään myös Domain-Driven Designia (*DDD).* DDD:n avulla saavutetaan vieläkin kattavampi lähestymistapa ohjelmistosuunnitteluun. (Karabulut 2023.)

2.2.1 Entiteetit

Kuvion 1 keltainen entiteettikerros (*Enterprise Business Rules*) on olennainen osa puhdasta arkkitehtuuria, sillä se muodostaa sovelluksen ytimen. Entiteettikerros voi sisältää tietorakenteita tai objekteja, kuten yksinkertaistettu tietorakenne kuviossa 2. Entiteeteillä on liiketoimintalogiikkaan liittyviä funktioita. Erityisen tärkeää on, että entiteetit ovat riippumattomia sovelluksen tilasta tai käyttötapauksista. Tämä tarkoittaa sitä, että ne eivät ole sidoksissa esimerkiksi käyttöliittymään tai tietokantaan. Eristämällä entiteetit muusta sovellusarkkitehtuurista saavutetaan useita etuja. Niiden muokkaaminen ja lisääminen helpottuvat, koska

muutosten tekeminen ei vaikuta muihin sovelluksen osiin. Lisäksi entiteettejä voidaan testata erikseen ja ne tarjoavat selkeän rajapinnan. (Martin 2012.)

```
namespace DomainEventExample;

public class Film
{
    1 usage
    public string? Title { get; set; }
    1 usage
    public string? Director { get; set; }
    1 usage
    public int ReleaseYear { get; set; }

    public string GetInfo()
    {
        return $"{Title} ({ReleaseYear}) by {Director}";
    }
}
```

Kuvio 2. Esimerkki entiteetistä

2.2.2 Käyttötapaukset

Kuvion 1 punainen käyttötapauskerros (*Application Business Rules*) käsittelee liiketoimintaprosesseja, sääntöjä ja logiikkaa. Käyttötapaus on yksittäinen tehtävä, jonka sovellus suorittaa vastauksena käyttäjän toimintaan tai johonkin tapahtumaan. Se voi olla esimerkiksi tilauksen tekeminen, käyttäjän sisäänkirjautuminen tai tuotteen lisääminen ostoskoriin. Käyttötapaukset toimivat rajapintoina liiketoimintalogiikan ja muiden osien, kuten käyttöliittymän ja tietokannan välillä. Käyttötapaukset kiteyttävät suorittamisen kannalta olennaiset säännöt ja logiikan. (Martin 2012.)

Käyttötapauksien eristäminen omaan kerrokseen tarjoaa useita etuja. Se tekee sovelluksesta modulaarisemman, mikä helpottaa yksittäisten toimintojen ylläpitoa ja laajentamista. Kun käyttötapaukset ovat eristettyinä, niitä on helpompi testata ja varmistaa, että logiikka toimii odotetusti. Lisäksi, jos liiketoiminnan vaatimukset muuttuvat, käyttötapauksia voidaan päivittää erillään muista osista, mikä

vähentää riskejä, sekä mahdollistaa joustavan kehityksen, ylläpidon ja laajentamisen ajan myötä.

2.2.3 Liitäntäsovittimet

Kuvion 1 vihreä liitäntäsovitinkerros (*Interface Adapters*) toimii välittäjänä sisäisten ja ulkoisten kerrosten välillä puhtaassa arkkitehtuurissa. Kerroksen päätehtävä on helpottaa eri kerrosten, kuten käyttöliittymän ja sovelluslogiikan välistä kommunikaatiota. Sovittimet toimivat ikään kuin tulkkeina ja työkaluina, jotka mahdollistavat kerrosten välisen yhteistyön. Tässä kerroksessa on kahdenlaisia sovittimia: tulosovittimia (*Input Adapters*) ja lähtösovittimia (*Output Adapters*). (Martin 2012.)

Tulosovittimet vastaanottavat käyttöliittymästä tulevat pyynnöt ja muuntavat ne muotoon, joka on ymmärrettävä sovelluslogiikalle. Ohjaimet (*Controllers*) vastaavat käyttöliittymän pyyntöjen käsittelystä. Ne toimivat käyttöliittymän ja sovelluslogiikan välisenä rajapintana, sekä ohjaa käyttöliittymästä tulleen pyynnön oikealle käyttötapaukselle ja käynnistävät sen. Esittimet (*Presenters*) puolestaan hoitavat käyttöliittymässä esitettävien tietojen muotoilun ja valmistelun. (Ushakov 2021.)

Lähtösovittimet tarjoavat rajapinnan sovelluksen ydinlogiikalle kommunikoida ulkoisten järjestelmien, kuten tietokantojen ja ulkoisten palveluiden kanssa. Ne huolehtivat tiedon hakemisesta ja tallentamisesta ulkoisista lähteistä. Tärkein periaate on yhteistyö ulkoisten järjestelmien kanssa. (Ushakov 2021.)

2.2.4 Kehykset ja ajurit

Kuvion 1 sininen kehykset ja ajurikerros (*Frameworks & Drivers*) ovat osa ulointa kerrosta puhtaassa arkkitehtuurissa. Tämä kerros on vuorovaikutuksessa ulkoisten järjestelmien, työkalujen ja infrastruktuurin kanssa. Se on myös suoraan yhteydessä ulkomaailmaan ja vastaa siitä, miten kommunikaatio tapahtuu ulkomaailman kanssa, sisältäen käyttöliittymän, tietokannat ja verkkopalvelut. (Martin 2012.)

Nykypäivänä käyttöliittymän kehityksessä on erittäin suosittua hyödyntää käyttöliittymäkehystä. Tällainen kehys voi olla esimerkiksi verkkokehys, kuten Vue ja React tai mobiilisovelluskehys, kuten Flutter ja Swift. Modernit käyttöliittymäkehykset nopeuttavat ja helpottavat kehitystyötä. Aikaisemmin ongelmalliseksi havaittiin kehyksien nopea vaihtuminen, sekä niiden suhteellisen lyhyt elinkaari. Nykyään osa kehyksistä on saavuttanut niin sanotun de facto -aseman, mikä tarkoittaa, että ne ovat vakiintuneita ja laajalti hyväksyttyjä. Vuoden 2023 Stack Overflow'n kehittäjäkyselyn mukaan React, Next.js ja Vue olivat kolme suosituinta verkkokehystä. (Stack Overflow 2023a.)

Tietokannat kuuluvat myös tähän kerrokseen. Yksinkertaistettuna tietokanta on jäsennelty digitaalinen arkisto, joka on suunniteltu tallentamaan, järjestämään ja hallitsemaan tietoja. Käyttäjä voi hakea, muokata ja analysoida tietokannan tietoja. (Oracle 2023.)

Aiemmin viittaamassani Stack Overflow'n kehittäjäkyselyssä neljä suosituinta tietokantaa olivat PostgreSQL, MySQL, SQLite ja MongoDB. Näistä kolme ensimmäistä ovat relaatiotietokantoja, kun taas MongoDB on NoSQL-tietokanta. (Stack Overflow 2023c.)

Relaatiotietokannat perustuvat SQL-kieleen (*Structured Query Language*) ja noudattavat relaatiotietokantojen periaatteita, kuten tietojen tallentamista taulukkomuodossa ja SQL-kyselyiden käyttöä tietojen hallintaan. MongoDB kuuluu NoSQL-tietokantoihin. NoSQL (*Not Only SQL*) on laaja käsite, joka kattaa erilaiset tietokantajärjestelmät. NoSQL-tietokannat, kuten MongoDB, käyttävät erilaisia tietomalleja, kuten dokumenttitallennetta, avain-arvo-pareja tai graafisia rakenteita, ja ne eivät vaadi ennaltamääriteltyä rakennetta. NoSQL-tietokannat tarjoavat joustavuutta ja skaalautuvuutta erityisesti suurten ja monimuotoisten tietojoukkojen käsittelyssä. (MongoDB 2023.)

3 DOMAIN-TAPAHTUMAT

3.1 Domain-tapahtumien määrittely

Domain-tapahtuma on ikään kuin ilmoitus jostain tapahtumasta tai tilan muutoksesta. Sitä voidaan käyttää tilanteissa, joissa halutaan muun sovelluksen osan olevan tietoinen jostain tietystä tapahtumasta. Kooditasolla domain-tapahtuma voidaan nähdä yksinkertaisena tietorakenteena, kuten kuviossa 3. Tapahtumia käytetään monipuolisesti pitkäkestoisten prosessien hallintaan ja viestintään eri osien välillä. Kun pitkäkestoinen prosessi on suoritettu loppuun, domain-tapahtuma voi ilmoittaa muille osille prosessin valmistumisesta, mahdollistaen seuraavien vaiheiden käynnistämisen. Lisäksi prosessin aikana kerättävät tärkeät tiedot voidaan välittää sovelluksen toisille osille domain-tapahtuman kautta, jotta ne voidaan käsitellä ja tallentaa tarpeen mukaan. Domain-tapahtuman avulla voidaan myös seurata ja valvoa prosessin etenemistä eri vaiheissa, samalla ilmoittaen mahdollisista virheistä ja poikkeustilanteista, jotta asianmukaiset toimenpiteet voidaan suorittaa. (Microsoft 2022.)

C# Copy

```csharp
public class OrderStartedDomainEvent : INotification
{
    public string UserId { get; }
    public string UserName { get; }
    public int CardTypeId { get; }
    public string CardNumber { get; }
    public string CardSecurityNumber { get; }
    public string CardHolderName { get; }
    public DateTime CardExpiration { get; }
    public Order Order { get; }

    public OrderStartedDomainEvent(Order order, string userId, string userName,
                                   int cardTypeId, string cardNumber,
                                   string cardSecurityNumber, string cardHolderName,
                                   DateTime cardExpiration)
    {
        Order = order;
        UserId = userId;
        UserName = userName;
        CardTypeId = cardTypeId;
        CardNumber = cardNumber;
        CardSecurityNumber = cardSecurityNumber;
        CardHolderName = cardHolderName;
        CardExpiration = cardExpiration;
    }
}
```

Kuvio 3. Domain-tapahtuma-esimerkki (Microsoft 2022)

3.2 Domain-tapahtumien suunnittelu

Domain-tapahtumien suunnittelu alkaa yleensä määrittelemällä tapahtumat, jotka ovat merkittäviä liiketoiminnan kannalta. Esimerkiksi, kun tilaus aloitetaan, voidaan julkaista orderStartedDomainEvent domain-tapahtuma. Tämä tarkoittaa, että kun domain-tapahtuma tapahtuu, se voi lähettää tiedon muille järjestelmän osille, joita kutsutaan aggregaateiksi. Sen jälkeen järjestelmän toiset osat voivat käsitellä tiedon ja käynnistää seuraavan vaiheen. (Microsoft 2022.)

Lisäksi on tärkeää määrittää, milloin domain-tapahtumat laukaistaan. Tämä voi riippua liiketoimintaprosessista ja siitä, milloin tietty toiminto tai tila on saavutettu. Esimerkiksi, jos on tapahtumasarja, joka edustaa tilauksen käsittelyprosessia, tapahtumien on tapahduttava tietyssä järjestyksessä esimerkiksi orderPlaced-tapahtuma ennen orderShipped-tapahtumaa. (Microsoft 2022.)

On myös tärkeää miettiä, miten tapahtumat käsitellään. Tämä voi sisältää tapahtumien tallentamisen tietokantaan, tapahtumien lähettämisen viestijonoon tai tapahtumien käsittelemisen sovelluksen sisällä. Tapahtumien seuranta on tärkeää, erityisesti kun ne lähetetään asynkronisesti. Tämä voi sisältää tapahtumalokien ylläpitämistä tai tapahtumien seuraamista. (Microsoft 2022.)

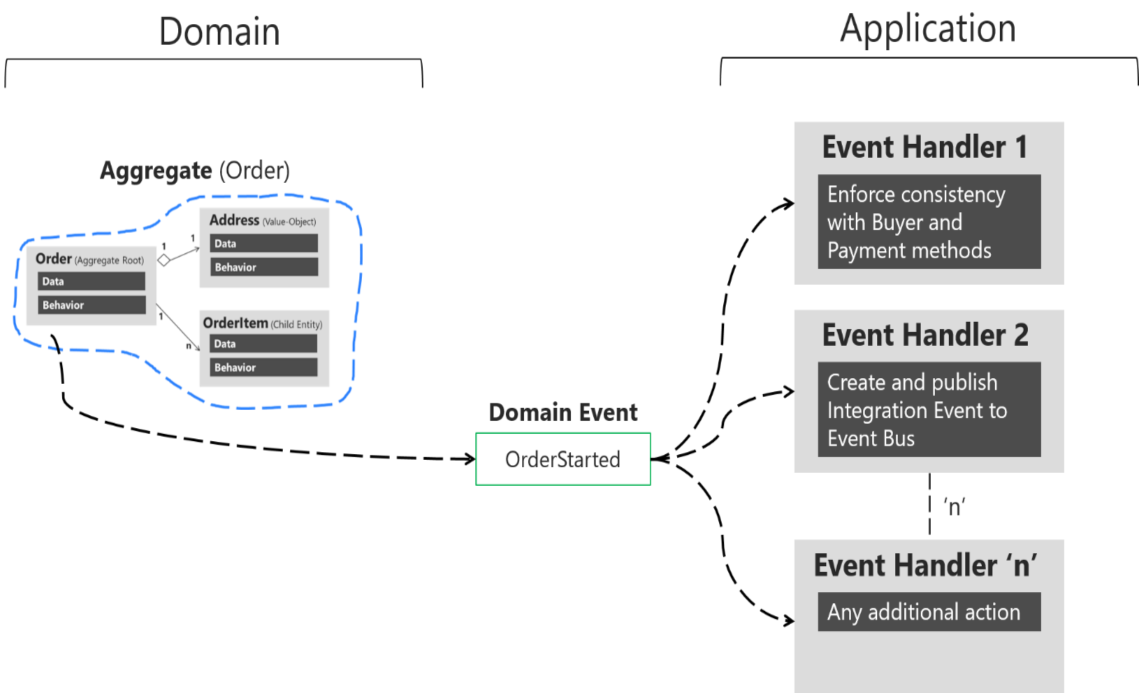
Lopuksi on olennaista ottaa huomioon, mitä seurauksia on, jos tapahtuman käsittely ei onnistu. Tämä voi tarkoittaa uudelleenyrityslogiikan toteuttamista, virheilmoitusten lähettämistä tai virhetilanteiden korjaamista. Kaikkein tärkeintä on, että suunnitteluprosessi vastaa liiketoimintavaatimuksia ja auttaa luomaan järjestelmän, joka täyttää nämä vaatimukset. (De La Torre 2017.)

3.3 Domain-tapahtumien käsittely

Domain-tapahtumien käsittelyyn on useita lähestymistapoja. Ensinnäkin niitä voidaan käsitellä in-process-tyyppisesti, mikä tarkoittaa tapahtumien käsittelyä suoraan sovelluksen sisällä ilman erillisiä palveluja tai prosesseja. Tämä lähestymistapa voi olla tehokas pienissä tai yksinkertaisissa järjestelmissä, joissa tapahtumien käsittely tapahtuu suoraan sovelluskoodin sisällä. (Microsoft 2022.)

Toinen vaihtoehto on käsitellä domain-tapahtumia taustapalvelun avulla. Tässä tapauksessa tapahtumien käsittely siirretään erilliseen taustaprosessiin tai palveluun, mikä voi tarjota useita etuja, kuten paremman suorituskyvyn ja eristetymmän ympäristön tapahtumien käsittelylle. Tämä lähestymistapa on erityisen hyödyllinen suurissa järjestelmissä, joissa on suuri määrä tapahtumia tai joissa tarvitaan monimutkaista tapahtumien käsittelyä. (Jovanovic 2023.)

Kumpikaan lähestymistapa ei ole ehdottomasti parempi kuin toinen, vaan niiden sopivuus riippuu sovelluksen vaatimuksista, käyttötapauksista ja arkkitehtuurista. Domain-tapahtumien käsittelyyn liittyy myös kuvion 4 mukainen käyttötapauskerroksen (Application) käsittelijä, joka vastaa tapahtumien jatkotoimenpiteistä. Kun domain-tapahtuma laukeaa, käsittelijä havaitsee sen ja suorittaa tarvittavat toimenpiteet sen perusteella. Tämä voi sisältää esimerkiksi tietyn liiketoimintalogiikan suorittamisen, tietojen päivittämisen tietokantaan tai muiden osien ilmoittamisen tapahtuman tapahtumisesta. (Microsoft 2022.)



Kuvio 4. Domain-tapahtumalla voi olla useita käsittelijöitä (Microsoft 2022)

3.4 Domain-tapahtumien nimeäminen

Domain-tapahtumien nimeäminen on tärkeä osa niiden suunnittelua ja käyttöä. Yleinen käytäntö on, että tapahtumat nimetään menneessä aikamuodossa kuvaamaan, mitä on tapahtunut. Esimerkiksi, kun tilaus on tehty, voitaisiin luoda OrderCreatedDomainEvent-domain-tapahtuma. (Microsoft 2022.)

On myös tärkeää huomata, että tapahtumien nimet ovat kirjainkoosta riippuvaisia. Yksi yleinen nimeämiskäytäntö on camel case. Se on muuttujien nimeämiskonventio, jossa jokainen paitsi ensimmäinen sana alkaa isolla kirjaimella, esimerkiksi camelCase. Lähes vastaavanlainen käytäntö on pascal case, jossa myös jokainen sana alkaa isolla kirjaimella. Erotuksena camel caseen on se, että pascal casessa myös ensimmäinen sana aloitetaan isolla kirjaimella. Esimerkiksi PascalCase on tällainen nimeämiskonventio. Näiden käytäntöjen avulla ohjelmointikoodin lukeminen ja ymmärtäminen helpottuvat, kun nimet ovat yhdenmukaisia ja noudattavat selkeitä sääntöjä. (Lemonaki 2022.) Domain-tapahtumien nimeämisessä käytetään usein Camel Casea tai Pascal Casea riippuen organisaation tai kehittäjäyhteisön käytännöistä. Tärkeintä on, että nimeäminen on yhdenmukaista ja helposti ymmärrettävää, jotta kehittäjät voivat selkeästi tunnistaa tapahtumat.

4 KEHITYSTYÖ

4.1 Käytetyt teknologiat

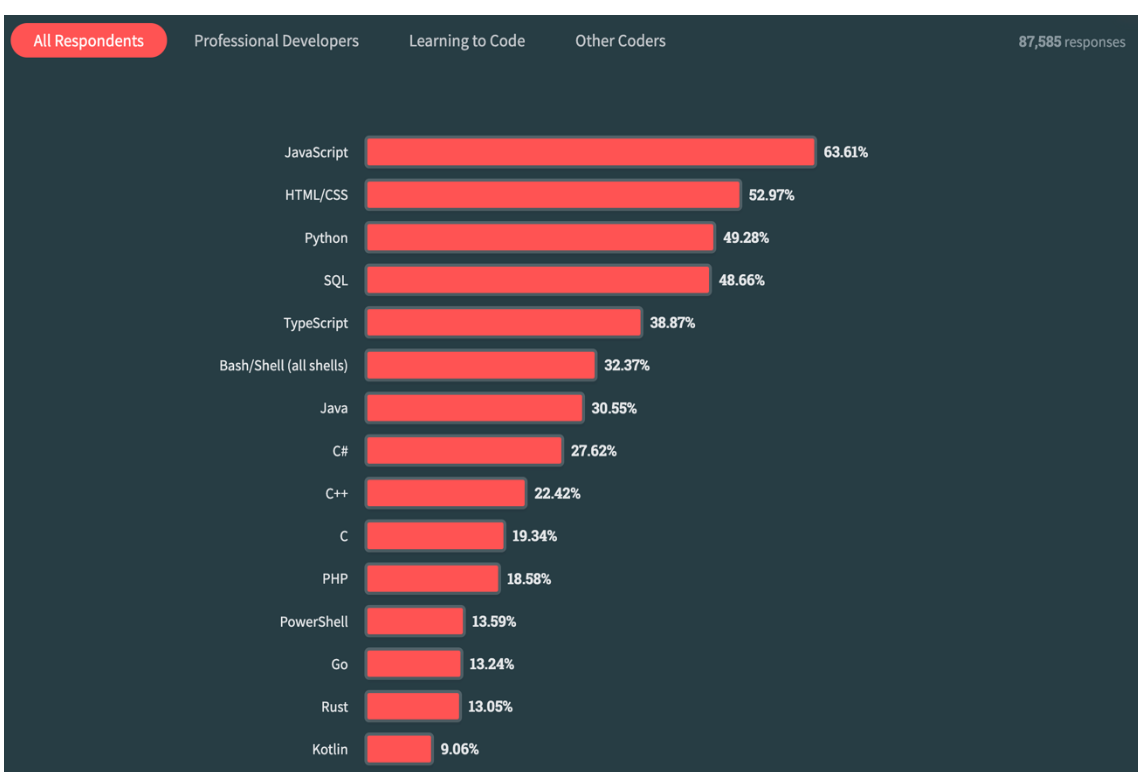
Erilaiset teknologiat ovat olennainen osa ohjelmistokehitystä ja niiden valinta voi vaikuttaa suuresti projektin lopputulokseen. Käytetyt teknologiat ovat huolella valittuja työkaluja, joiden avulla pyritään saavuttamaan projektin tavoitteet tehokkaasti ja luotettavasti.

4.1.1 C#

Microsoft kehitti C#-ohjelmointikielen 2000-luvun alussa, ja se esiteltiin ensimmäisen kerran julkisuudessa vuonna 2000. C# suunniteltiin alun perin .NET Frameworkin yhteyteen, ja siitä tuli merkittävä osa Microsoftin ohjelmistokehityksen ekosysteemiä. (Microsoft 2023a.)

Se on monipuolinen ja laajasti käytetty ohjelmointikieli, joka soveltuu erilaisiin sovelluskehityksen tarpeisiin. Sitä käytetään laajasti web-sovellusten, mobiilisovellusten, palvelinsovellusten ja pelikehityksen parissa. C# on suosittu ohjelmointikieli useista syistä. Se tarjoaa tehokkaan, laajan dokumentaation ja selkeän syntaksin, joka on helppo oppia ja ymmärtää. Kielen suunnittelussa on panostettu vahvasti helppokäyttöisyyteen ja ohjelmointivirheiden minimoimiseen, mikä tekee siitä houkuttelevan sekä aloitteleville että kokeneille kehittäjille. C# on vahvasti tyypitetty kieli, mikä auttaa kehittäjiä havaitsemaan ja korjaamaan virheet jo koodin kirjoitusvaiheessa ennen ohjelman suorittamista. (Microsoft 2023b.)

Viimeisimmässä Stack Overflow’n toteuttamassa kehittäjäkyselyssä C# osoittautui erittäin suosituksi ohjelmointiteknologiaksi, sijoittuen kyselyn tuloksissa kahdeksanneksi suosituimmaksi ohjelmointikieleksi (kuvio 5). Tämä vahvistaa C#-kielen vakiintunutta asemaa ja laajaa käyttöä ohjelmistokehityksessä. Kyselyt kuten Stack Overflow’n toteuttama vuosittainen kehittäjäkysely, ovat arvokkaita työkaluja ohjelmointiyhteisölle, koska ne heijastavat alan trendejä ja suuntauksia. Tulos antaa vahvaa tukea sille, että C# on kieli, joka on edelleen tärkeässä asemassa ohjelmistokehityksen maailmassa. (Stack Overflow 2023b.)



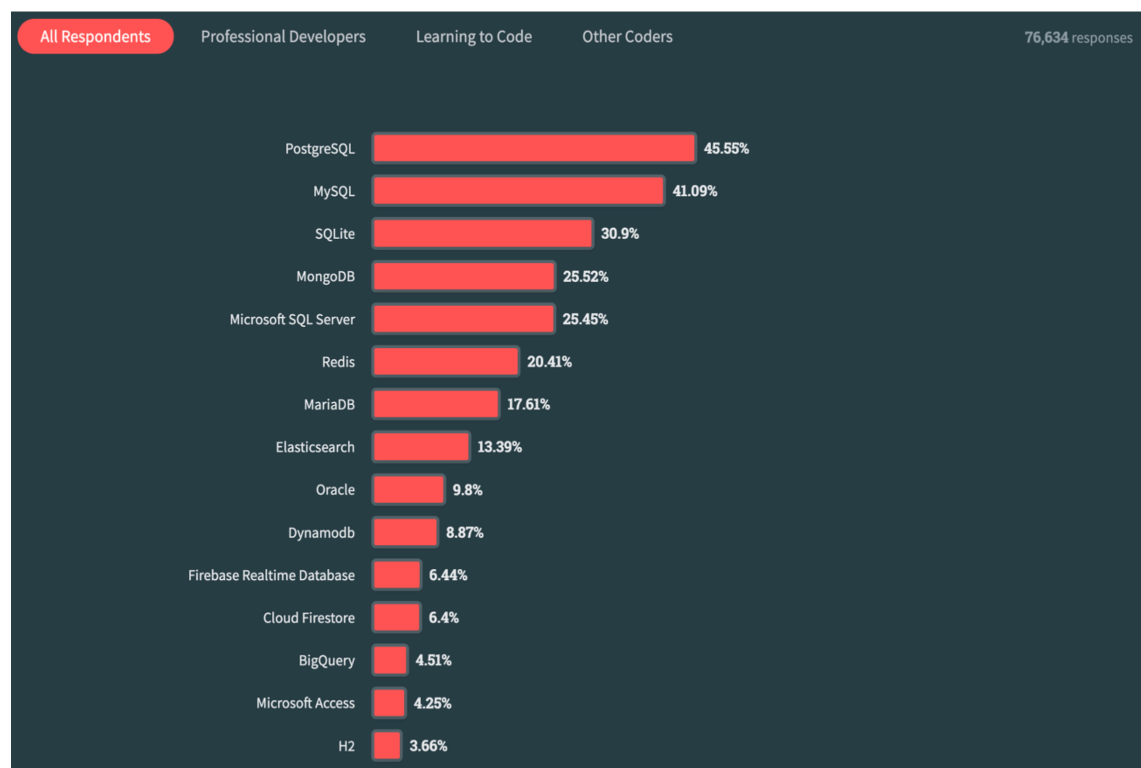
Kuvio 5. Stack Overflow’n kehittäjäkyselyn tulos suosituimmista ohjelmointikielistä (Stack Overflow 2023b)

4.1.2 PostgreSQL

PostgreSQL edustaa voimakasta ja avoimen lähdekoodin objekti-relaatiotietokantajärjestelmää, joka pohjautuu SQL-kieleen, rikastuttaen sitä monipuolisilla ominaisuuksilla. Tämän tietokantajärjestelmän alkuperät juontavat juurensa vuoteen 1986, kun se oli osa POSTGRES-projektia Kalifornian yliopistossa. Projektilla on vaikuttava yli 35 vuoden kehityshistoria. Sen maine perustuu vahvaan arkkitehtuuriin, luotettavuuteen, laajaan toiminnallisuuteen, laajennettavuuteen ja avoimen lähdekoodiyhteisön tinkimättömään sitoutumiseen tarjota suorituskykyisiä ja innovatiivisia ratkaisuja. (PostgreSQL 2023.)

Vuonna 2023 PostgreSQL vakiinnutti asemansa suosituimpana tietokantana Stack Overflow’n kehittäjäkyselyssä (kuvio 6). Tämä vahvistaa PostgreSQL:n kasvavaa suosiota ja laajaa käyttöä ohjelmistokehityksen maailmassa. PostgreSQL:n asema suosituimpana tietokantana vuonna 2023 korostaa sen

vahvuuksia luotettavuudessa, suorituskyvyssä ja laajennettavuudessa, mikä tekee siitä houkuttelevan vaihtoehdon monille kehittäjille ja organisaatioille. (Stack Overflow 2023c.)



Kuvio 6. Stack Overflow'n kehittäjäkyselyn tulos suosituimmista tietokannoista (Stack Overflow 2023c)

4.2 Alkumäärittely

Opinnäytetyö keskittyi ratkaisemaan seuraavan ongelman: kuinka integroida domain-tapahtumat varausjonon käsittelyn osaksi. Tämän tavoitteen saavuttamiseksi suoritettiin muutoksia C#-pohjaiseen API-projektiin, joka toimii .NET-ympäristössä. Tarkoituksena oli erottaa erilaiset asynkroniset komennot toisistaan ja luoda yleinen käsittely domain-tapahtumille. Lisäksi tutkittiin, voiko Rebus-kirjastoa käyttää PostgreSQL-tietokannan kanssa tallennusratkaisuna domain-tapahtumille.

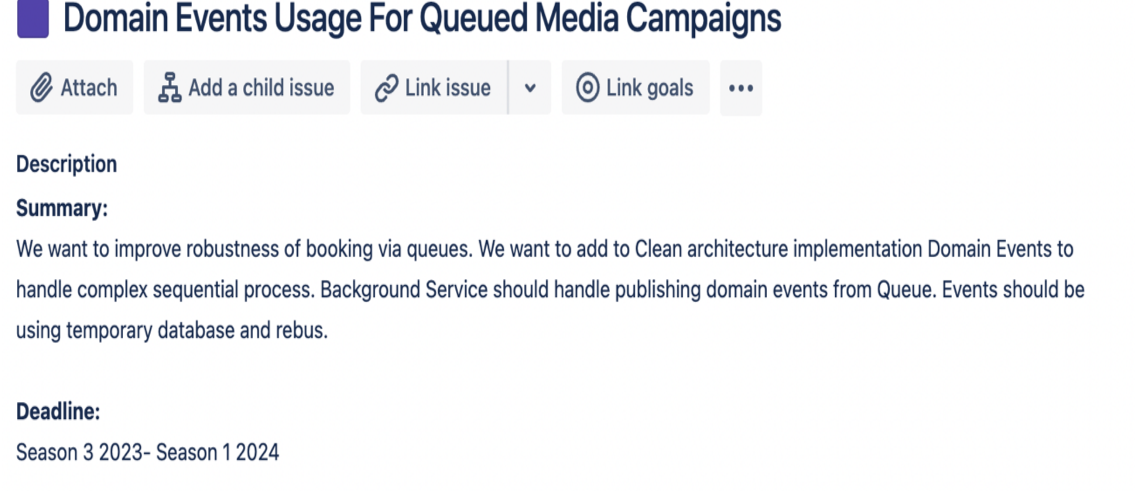
Opinnäytetyön avulla pyritään selkeästi määrittelemään ja ratkaisemaan nämä haasteet. Lisäksi pyritään parantamaan järjestelmän suorituskykyä ja valmistelemaan se tuleviin kehitysvaiheisiin.

4.3 Suunnittelu

Kehitystyön suunnittelu oli olennainen vaihe opinnäytetyön prosessissa, ja se käynnistyi alkumäärittelyiden jälkeen. Tavoitteena oli luoda selkeä ja rakenteellinen pohja kehitystyön etenemiselle. Suunnittelu aloitettiin suorittamalla perusteellinen ja lähdekriittinen tietojen keruu erilaisista tietolähteistä, kuten verkkosivustoista, tieteellisistä artikkeleista ja virallisesta dokumentaatiosta. Tiedonhankinta oli välttämätöntä varmistaakseni opinnäytetyön laadun ja luotettavuuden.

Riittävän tiedonhankinnan jälkeen suoritettiin useita iteraatioita, joissa käytiin läpi opinnäytetyöhön liittyviä teknisiä muutosehdotuksia, kerta kerralta parannellen muutosehdotusta. Tämän jälkeen luotiin Jira projektinhallintatyökaluun kehitystyön ”epic” ja tarvittavat ”tiketit”.

Kuvion 7 mukainen Jira epic on projektinhallinnan termi, joka viittaa suuriin työkokonaisuuksiin tai päätehtäviin. Se toimii yhtenä korkeimmalla tasolla olevista tehtävistä ja auttaa organisoimaan työkokonaisuudet pienempiin, hallittavampiin alatehtäviin eli tiketteihin. Jira on suosittu Atlassian-yhtiön kehittämä projektinhallinta- ja seurantatyökalu, jota käytetään erityisesti ohjelmistokehityksessä. Jira tarjoaa useita työkaluja, joiden avulla organisaatiot voivat hallita projektejaan tehokkaasti ja seurata niiden etenemistä. (Atlassian 2023.)



Kuvio 7. Kehittämistyön Jira epic

4.4 Toteutus

Kehitystyössä hyödynnettiin ketteriä menetelmiä, jotka mahdollistivat joustavan ja tehokkaan työskentelyn. Työskentely suoritettiin sprinttityöskentelynä, joka on yksi ketterän kehityksen peruspilareista. Sprintit ovat toistuvia, aikarajoitettuja työjaksoja, joiden aikana tiimi pyrkii saavuttamaan tiettyjä ennalta määriteltyjä tavoitteita. Sprinttien avulla pystyttiin jakamaan työn hallittaviin osiin ja seuraamaan edistymistä säännöllisesti.

Vetopyynnöt olivat myös tärkeä osa työprosessia. Ne mahdollistivat koodin tarkastelun ja arvioinnin ennen sen yhdistämistä päähaaraan. Tämä auttoi varmistamaan koodin laadun ja ehkäisemään virheitä.

Testaus oli myös olennainen osa kehitysprosessia. Automaattisia testejä käytettiin varmistamaan, että koodi toimi odotetusti. Testit ajettiin automaattisesti aina, kun vetopyyntö tehtiin, mikä auttoi havaitsemaan ja korjaamaan mahdolliset ongelmat nopeasti.

Kaiken kaikkiaan ketterät menetelmät ja hyvät käytännöt, kuten sprinttityöskentely, vetopyynnöt ja automaattiset testiajot, auttoivat toteuttamaan kehitystyön tehokkaasti ja laadukkaasti. Ne mahdollistivat jatkuvan parantamisen, joustavuuden ja korkean tuottavuuden, mikä on olennaista nykypäivän nopeasti muuttuvassa ohjelmistokehitysympäristössä.

Kehitystyön keskiössä oli domain-tapahtumien implementointi ja käsittelyn toteuttaminen. Seuraavissa luvuissa avaan tarkemmin prosessin etenemistä sekä niitä haasteita, joita kohtasin kehitystyön aikana. Valitsin toteutukseen MediatR-kirjaston hyödyntäen INotification-rajapintaa. Lisäksi kokeilin Rebus-viestinvälittäjää syventyäkseni näiden teknologioiden soveltuvuuteen sovelluksen tarpeisiin.

Valitsin MediatR-kirjaston sen yksinkertaisuuden ja joustavuuden vuoksi. MediatR tarjoaa selkeän tavan käsitellä ja hallita domain-tapahtumia INotification-rajapinnan ansiosta. INotication-rajapinta tarjoaa yksinkertaisen tavan välittää

tapahtumat käsittelijöille ja mahdollistaa tapahtumien välittämisen eri osien välillä. (GitHub Inc 2024.)

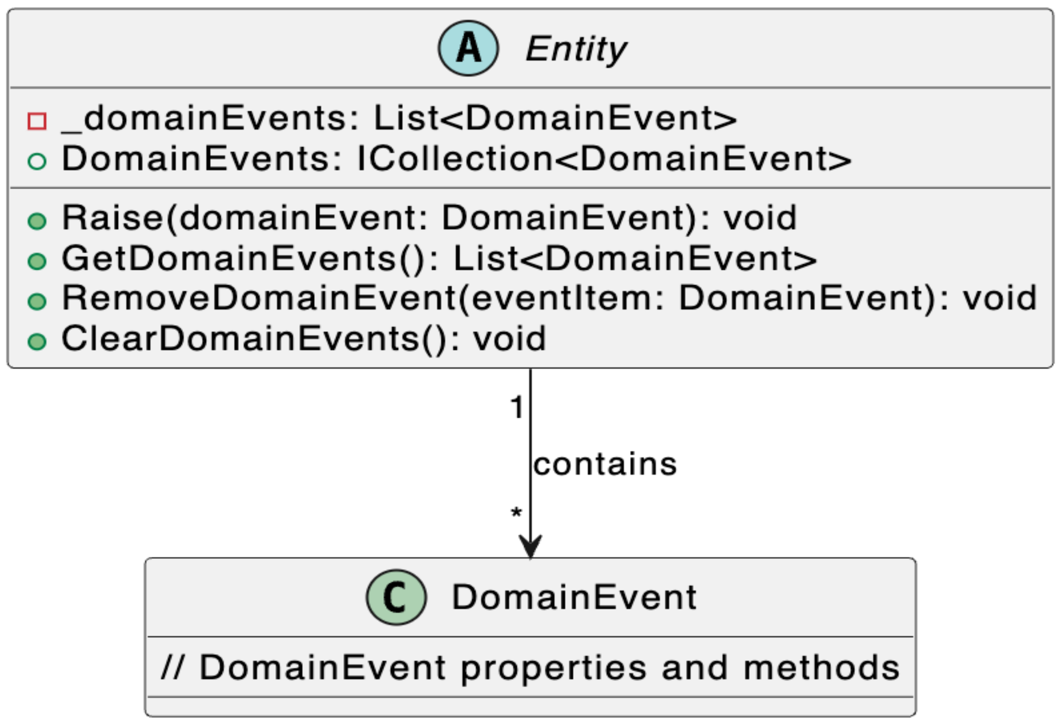
Rebus taas on hajautetun viestinvälityksen kirjasto, joka tarjoaa ratkaisuja viestien välittämiseen eri sovelluskomponenttien välillä. Se tarjoaa laajan valikoiman erilaisia konfiguraatiomahdollisuuksia, mikä tekee siitä monimutkaisemman kuin MediatR-kirjasto. Kuitenkin oikein käytettynä Rebus voi osoittautua erityisen hyödylliseksi vaativimmissa tapauksissa. (Heller Grabe 2023.)

Toteutin Rebus-kirjaston avulla myös domain-tapahtumien käsittelyn ja hallinnan, mutta on kuitenkin huomattava, että jotta Rebusista saataisiin enemmän hyötyä kuin MediatR-kirjastosta, edellyttäisi se viestijonon konfigurointia, kuten Azure Service Busin, ja viestien tallennuspaikan, kuten PostgreSQL-tietokannan, käyttöönottoa. Kun Rebusin infrastruktuurielementeille ei ole tarvetta, MediatR tarjoaa paljon suoraviivaisemman vaihtoehdon. Se tarjoaa yksinkertaisen ja suoran tavan käsitellä tapahtumia sovelluksessa ilman ylimääräistä infrastruktuurin asentamista tai konfigurointia. MediatR keskittyy pääasiassa tapahtumien käsittelyyn ja erottaa liiketoimintalogiikan selkeästi muusta sovelluksesta, mikä tekee koodista helpommin ylläpidettävää ja ymmärrettävää. Kehitetty taustapalvelu käytti Rebus-kirjastoa.

4.4.1 Domain-tapahtumien implementointi ja käsittely

Kehitystyö aloitettiin toteuttamalla suunnitteluvaiheessa määritetyt domain-tapahtumat. Näin ollen siirryin käytännön kooditasolle ja aloin toteuttamaan näitä tapahtumia sovelluksessa. Luodessani perusluokat pyrin varmistamaan, että ne vastasivat suunniteltuja tapahtumarakenteita. Integroin tapahtumat osaksi sovellusta noudattaen suunnitteluvaiheessa tehtyjä ratkaisuja. Tapahtumaluokkien tueksi luotiin domain-tapahtumakantaluokka, jonka kaikki tapahtumaluokat perivät. Tämä ratkaisu mahdollisti yhtenäisen ja jäsennellyn lähestymistavan tapahtumien hallintaan, kun kaikki tapahtumaluokat jakoivat yhteisen perusrakenteen domain-tapahtumakantaluokan kautta.

Domain-tapahtumat toteutettiin noudattaen puhtaan arkkitehtuurin periaatteita ja näin ollen perusluokat sijoitettiin entiteetti kerrokseen. Asettamalla domain-tapahtumat tähän kerrokseen varmistetaan, että ne ovat osa sovelluksen ydintä. Tämä lisää liiketoimintalogiikan selkeyttä ja yksinkertaisuutta. Tapahtumien jälkeen kehitettiin yleinen kuvion 8 mukainen hallintaluokka tapahtumille, se tarjoaa yhtenäisen rajapinnan tapahtumien käsittelyyn sovelluksen eri osissa. Tämä luokka mahdollistaa tapahtumien nostamisen, etsimisen ja poistamisen yhdessä keskitetyssä paikassa.



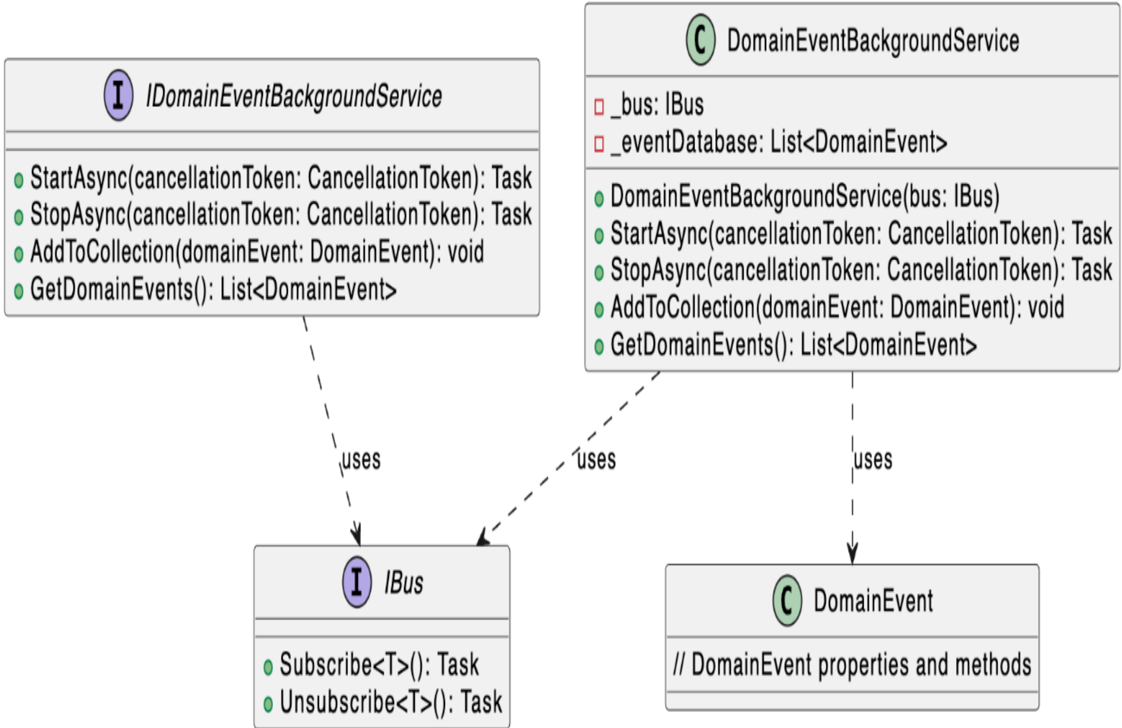
Kuvio 8. Geneerinen hallintaluokka domain-tapahtumille

Jokainen tapahtumaluokka vastaa tiettyä liiketoimintatapahtumaa. Kun sovelluslogiikassa syntyy uusi tilaus, käytetään MediatR-kirjastoa tapahtumaluokan luomiseen ja julkaisemiseen. MediatR-kirjasto mahdollistaa tapahtuman käsittelyn tilaamalla siihen liittyviä käsittelijöitä. Käsittelijät rekisteröidään, ja niille määritellään vastuu tietyn tapahtuman käsittelystä. Käsittelijät ovat vastuussa liiketoimintalogiikasta ja muiden toimintojen suorittamisesta tapahtuman syntymisen yhteydessä.

4.4.2 Taustapalvelu

Yleisesti taustapalveluita hyödynnetään C#-rajapinnassa erilaisten tehtävien suorittamiseen asynkronisesti. Taustapalvelu mahdollistaa sovelluksen jatkuvan toiminnan ilman pääsäikeen tukkeutumista.

Kun toteutin kuvion 9 domain-tapahtumien hallintaan perustuvaa taustapalvelua, hyödynsin Rebus-kirjastoa viestinvälitykseen ja käsittelyyn. Ensisijainen tavoite oli käsitellä domain-tapahtumia, jotka syntyvät sovelluksessa. Käyttämällä Rebus-kirjastoa voitiin helposti kapseloida nämä tapahtumat viesteiksi, jotka taustapalvelu voi käsitellä.

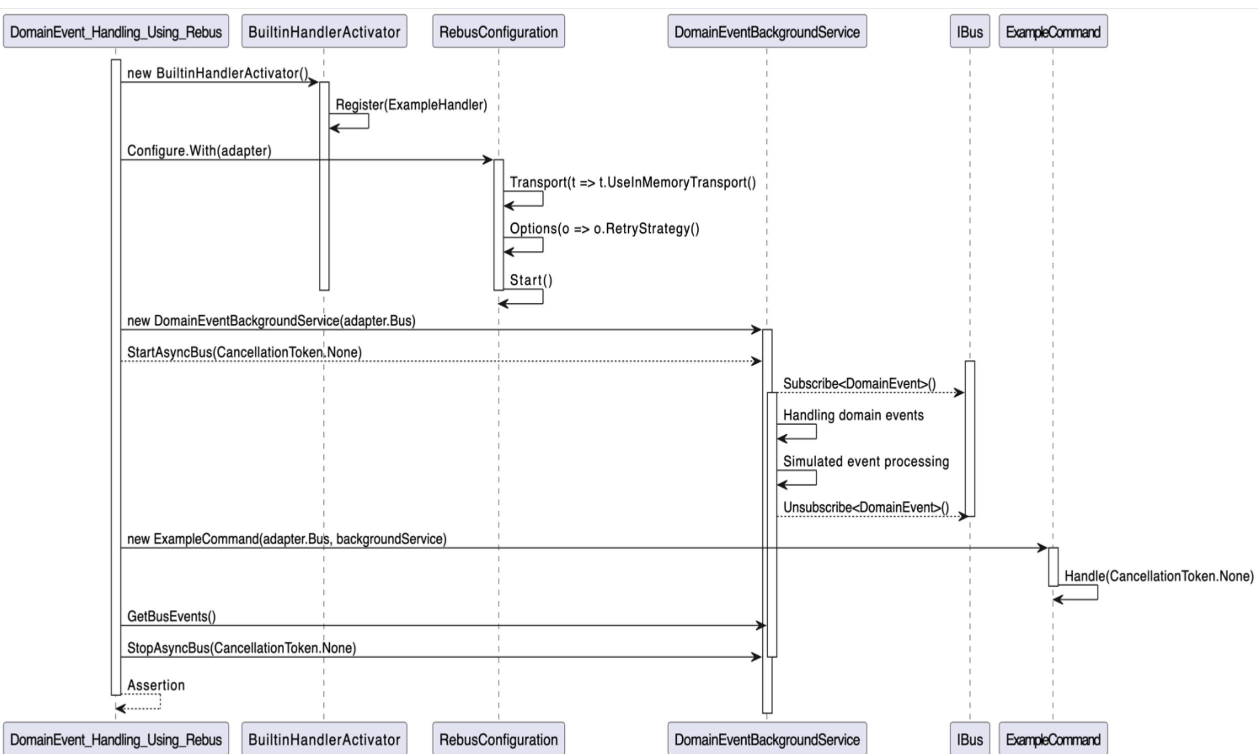


Kuvio 9. Taustapalvelu

Vaikka taustapalvelun alkuperäinen tarkoitus oli lähettää käsitellyt tapahtumat viestijonoon ja tallentaa ne postgreSQL-tietokantaan, aikataulurajoitteiden vuoksi nämä toiminnallisuudet jäivät konfiguroimatta. Kuitenkin taustapalvelun toimivuuden varmistamiseksi tapahtumien käsittelyä ja tallennusta simuloitiin testausvaiheessa.

4.4.3 Testaus

Testauksessa hyödynnettiin xUnit-testikehystä, joka tarjoaa tehokkaan ja joustavan työkalun automatisoidulle testaukselle. Taustapalvelun testauksessa hyödynnettiin Rebusin in-memory-kuljetusta. Tämä valinta mahdollisti testien suorittamisen ilman tarvetta käyttää ulkoista viestinvälittäjää tai tietokantaa, mikä yksinkertaisti testausta ja nopeutti testisykliä. Testitapauksissa pystyin simuloimaan erilaisia skenaarioita, kuten onnistuneita tapahtumien käsittelyjä, virhetilanteita ja uudelleenyrityksiä, varmistaen taustapalvelun vakaan toiminnan eri tilanteissa. Kuviossa 10 on sekvenssidiagrammi yhdestä kehitetystä testimetodista.



Kuvio 10. Sekvenssidiagrammi yhdestä testitapauksesta

Jokainen testimetodi noudattaa "Arrange, Act, Assert" (*AAA*) -rakennetta varmistaen, että testit ovat selkeitä ja ymmärrettäviä. Testit myös varmistavat, että taustapalvelun in-memory-tallennusjärjestelmään tallennetaan odotetusti tapahtumat.

4.5 Jatkokehitys

Ensimmäinen olennainen askel jatkokehityksen kannalta olisi Rebusin käyttöä varten pilvipalveluiden konfigurointi, jolla saataisiin domain-tapahtumille tallennuspaikka ja viestijono tallentamista varten. Pilvipalveluiden konfigurointi mahdollistaisi Rebusin täyden hyödyntämisen.

Suuremman kokoluokan kehitys-epic on suunnitteilla myöhemmin tänä vuonna keskittyen domain-tapahtumien jatkokehitysteemoihin. Tämä laaja kokonaisuus pyrkii tuomaan merkittäviä parannuksia järjestelmän reagointikykyyn, virheenhallintaan ja integraatioihin, joihin olen vahvasti sitoutunut opinnäytetyöni myötä.

Opinnäytetyön edetessä on käynyt ilmi, että domain-tapahtumien laajentuminen muihin liiketoimintaobjekteihin voisi olla olennainen askel järjestelmän kehityksessä. Tulevaisuudessa odotetaan palvelujen käytön kasvavan merkittävästi, ja tässä kehityksessä Rebusista olisi todellakin suurta hyötyä. Järjestelmän laajentuessa kohti mikropalveluarkkitehtuuria integraatiotapahtumat muodostavat luonnollisen askeleen tapahtumien hallinnassa.

Integraatiotapahtumat ovat samankaltaisia kuin domain-tapahtumat, mutta niiden konteksti poikkeaa domain-tapahtumista. Domain-tapahtumat kuvaavat järjestelmän sisäisiä tapahtumia, kun taas integraatiotapahtuman tarkoituksena on välittää tietoa järjestelmästä ulospäin.

5 POHDINTA

Opinnäytetyöni avulla minulla oli mahdollisuus tutustua domain-tapahtumiin sekä osallistua aktiivisesti niiden suunnitteluun ja toteutukseen. Keskittyessäni liiketoimintaprosessien parantamiseen ja järjestelmän luotettavuuden kasvattamiseen, pyrin tuomaan esille ratkaisuja, jotka tukevat liiketoimintatarpeita ja vastaavat organisaation vaatimuksia.

Ammatillisesti tämän prosessin aikana olen oppinut syventämään ymmärrystäni puhtaasta arkkitehtuurista ja domain-tapahtumien merkityksestä ohjelmistokehityksessä. Osallistuminen käytännön työskentelyyn näiden käsitteiden parissa on antanut minulle arvokasta kokemusta ja taitoja, joita voin varmasti hyödyntää tulevissa työtehtävissäni.

Alkuperäisen suunnitelman mukaan Azuren pilvipalveluiden hyödyntäminen oli houkutteleva vaihtoehto. Kuitenkin aikataulurajoitteiden vuoksi tämä mahdollisuus jäi toteuttamatta. Vaikka pilvipalveluiden käyttö jäi saavuttamatta, työssä päästiin silti tavoitteisiin, kun kehitin parannusehdotuksen taustakäsittelyn tehokkuuden ja luotettavuuden parantamiseksi. Olisin kuitenkin voinut tarkentaa työn rajausta, mikä olisi auttanut keskittymään olennaisiin seikkoihin ja saavuttamaan vieläkin parempia tuloksia.

Vaikka domain-tapahtumat eivät välttämättä ole viimeisin trendi tämän hetken ohjelmistoalalla. Ollaan silti tekemisissä tärkeiden perusasioiden kanssa, tämä on viisasta pitää mielessä ohjelmistojen kehityksessä ja suunnittelussa. Vaikka uusimmat teknologiat ja kehityssuuntaukset saavat usein eniten huomiota, domain-tapahtumat voivat olla olennainen osa ohjelmiston perusrakennetta.

Näiden perusasioiden ymmärtäminen ja asianmukainen soveltaminen voivat tuoda konkreettista arvoa sekä lyhyellä että pitkällä aikavälillä. Ne auttavat varmistamaan, että ohjelmistojärjestelmät ovat joustavia, helposti ylläpidettäviä ja skaalautuvia. Tämä tasapaino uusimpien trendien ja perusasioiden välillä on avain onnistuneeseen ohjelmistokehitykseen, sillä ne tarjoavat kestävän perustan innovatiivisten ratkaisujen kehittämiselle. Jatkuva keskittyminen näissä

perusasioissa osoittautuu pitkällä aikavälillä usein ratkaisevaksi tekijäksi onnistuneessa ohjelmistoprojektissa.

LÄHTEET

Atlassian 2023. Welcome to Jira Software. Viitattu 22.10.2023 https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software.

De La Torre, C. 2017. Domain Events vs. Integration Events in Domain-Driven Design and microservices architectures. Microsoft DevBlog 7.2.2017. Viitattu 25.8.2023 https://devblogs.microsoft.com/cesardelatorre/domain-events-vs-integration-events-in-domain-driven-design-and-microservices-architectures/.

GitHub Inc. 2024. MediatR Wiki. GitHub Inc 13.1.2024. Viitattu 18.2.2024 https://github.com/jbogard/MediatR/wiki.

Heller Grabe, M. 2023. Rebus Wiki. GitHub Inc 6.3.2023. Viitattu 18.2.2024 https://github.com/rebus-org/Rebus/wiki.

Jovanovic, M. 2022. Clean architecture and the benefits of structured software design. Milan Jovanovic 24.12.2022. Viitattu 7.12.2023 https://www.milanjovanovic.tech/blog/clean-architecture-and-the-benefits-of-structured-software-design.

Jovanovic, M. 2023. How to use domain events to build loosely coupled systems. Milan Jovanovic 22.7.2023. Viitattu 22.2.2024 https://www.milanjovanovic.tech/blog/how-to-use-domain-events-to-build-loosely-coupled-systems.

Karabulut, A. 2023. Understanding clean architecture and domain driven design. Medium 27.5.2023. Viitattu 11.8.2023 https://medium.com/bimar-teknoloji/understanding-clean-architecture-and-domain-driven-design-ddd-24e89caabc40.

Lemonaki, D. 2022. Snake Case VS Camel Case VS Pascal Case VS Kebab Case – What’s the Difference Between Casings? FreeCodeCamp 29.11.2022. Viitattu 19.11.2023 https://www.freecodecamp.org/news/snake-case-vs-camel-case-vs-pascal-case-vs-kebab-case-whats-the-difference/.

Martin, R. 2008. Clean architecture, A Craftsman’s Guide to Software Structure and Design. E-kirja. Viitattu 4.8.2023.

Martin, R. 2012. The Clean Architecture. The Clean Code Blog 13.8.2012. Viitattu 4.8.2023 https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

Microsoft. 2022. Domain events: Design and implementation. Microsoft 29.12.2022. Viitattu 25.8.2023 https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/domain-events-design-implementation.

Microsoft. 2023a. The history of C#. Viitattu 1.10.2023 https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history.

Microsoft. 2023b. A tour of the C# language. Microsoft 4.5.2023. Viitattu 1.10.2023 https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

MongoDB. 2023. NoSQL vs. SQL Databases. Viitattu 20.8.2023 https://www.mongodb.com/nosql-explained/nosql-vs-sql.

Oracle. 2023. What is a database? Viitattu 20.8.2023 https://www.oracle.com/database/what-is-database/.

PostgreSQL. 2023. What is PostgreSQL? Viitattu 1.10.2023 https://www.postgresql.org/about/.

Sanoma. 2023. Mitä teemme Media Finland. Viitattu 10.8.2023 https://www.sanoma.com/fi/mita-teemme/media-finland/.

Stack Overflow. 2023a. Developer survey 2023: Admired and desired web frameworks and technologies. Viitattu 19.8.2023 https://survey.stackoverflow.co/2023/#section-admired-and-desired-web-frameworks-and-technologies.

Stack Overflow. 2023b. Developer survey 2023: Most popular technologies and languages. Viitattu 1.10.2023 https://survey.stackoverflow.co/2023/#most-popular-technologies-language.

Stack Overflow. 2023c. Developer survey 2023: most popular technologies and databases. Viitattu 1.10.2023 https://survey.stackoverflow.co/2023/#section-most-popular-technologies-databases.

Ushakov, G. 2021. Clean architecture: domain entities and interface Adapters. Medium 18.7.2021. Viitattu 11.8.2023 https://medium.com/@gushakov/clean-architecture-domain-entities-and-interface-adapters-4152b9ee22d2.