# Глава 1. Проблеми на информационното осигуряване при управление на поръчките от клиенти

## 1.1. Същност и принципи на информационните системи, поддържащи дейността на производствено предприятие

### 1.1.1. Характеристика на ERP системите

ERP информационните системи позволяват ефективното планиране на дейността на предприятие, в т.ч. разходи за обновяване на оборудването и инвестициите в производството на нови изделия (Банабакова, В. К, 2019). Те произхождат от RP технологията (Requirements/resource planning - планиране на потребностите/ресурсите). Основната цел на RP е съкращаване на количеството на запасите от материали, незавършено производство и готова продукция, съгласуване на графика на доставките с работата на отделните производствени звена и процеса на закупуване (Филипов, Ст. Г., 2019). С логистичните технологии и тяхното приложение се цели осигуряване на оптимални решения в логистичната система (Благоев, Бл., 2010). Логистичната технология се определя като стандартизирана последователност (алгоритъм) на изпълнение на отделни логистични функции, и/или процеси в логистичната система или в отделни нейни функционални области (Сергеева, В., 2004). Някои от тези алгоритми и поддържащите ги информационно управляващи системи са получили и нормативна регламентация. Такива са MRP I и MRP II, за които са разработени и утвърдени международни стандарти ISO (Стоянов, Ст. Хр., 2019). ER спомагат преодоляването на проблеми свързани с управление на складовите наличности, като:

• Забавяне на постъпването на материали;

• Натрупване на излишни складови запаси;

• Нарушаване баланса на доставката;

• Намаление на ефективността на производство;

• Сложност при процеса на контрол и анализ на производствената дейност;

Подобренията от въвеждането на ERP се изразяват в увеличаване броя на изпълнените поръчки, повишаване качеството на логистичното обслужване към клиентите, възможности за промени в обема на поръчките, съкращаване на времето от поръчката до доставката и други (Банабакова, В. К, 2019).

Нека разгледаме SAP, който бива пуснат за първи път преди 50 години в Германия. В рамките на тази система се управляват всички функционални области на даден бизнес: от човешки ресурси, до финанси и функции за закриване на отчетен период, продажби, управление на клиенти, фактуриране и задължения, управление на инвентара, логистика, и други. Всяка отделна функция, от която едно предприятие може да има нужда, е напълно достъпна и интегрирана в SAP. Както беше отбелязано, SAP е водещата ERP система на пазара, като около 87% от глобалните организации го използват. SAP може да поддържа производствена (например Harley Davidson), нефтената и газовата индустрия (Shell), държавни органи.

В зависимост от компанията и много други фактори, внедряването на SAP може да отнеме дълго време и много ресурси. За сметка на това, SAP предлага и автоматизация, инструменти, които автоматично ще получават, съгласуват и извършват определени действия. Това позволява съсредоточаването върху други задачи, които могат да помогнат за растежа на компанията. SAP разполага с огромни количества данни, които да бъдат използвани за вземане на ефективни и сигурни бизнес решения.

SAP е разделен на различни области, които работят заедно, наречени модули. В SAP има два вида модули: технически и функционални модули. В техническите модули работят програмисти или ИТ персонал. Функционалните модули, които се използват от потребителите на системата, съдържат вградени транзакции, съответстващи на бизнес процесите. SAP има много функционални модули. Следната фигура представя 3 от най-използваните. 

***Фигура 1.1****: Модули в САП.*

Първата група е от модул Логистика. Sales and Distribution е насочен към обработка и доставка на поръчки за продажби. Модулът за управление на материалите включва закупуване и управление на инвентара. Производственото планиране съдържа основни данни като спецификация на материалите, маршрути и изпълнение на планирането на материалните изисквания. SAP Quality Management управлява контрола на качеството и несъответствията. Финансовото счетоводство на SAP включва главната книга, дължимите сметки и вземанията. SAP Controlling е мястото, където се извършва анализът на себестойността на продукта и рентабилността. Project Systems управлява счетоводните аспекти на планирането на проекта, мониторинга на проекта и функциите за изчисляване на разходите по проекта. Управлението на човешките ресурси обхваща целия цикъл на един служител в една компания, от наемане до прекратяване, включително ведомост. Функционалните модули са тясно интегрирани. Те ефективно изпращат информация помежду си, което е една от най-силните страни на SAP ERP системата.

SAP се състои от два типа данни: основни и транзакционни. Основните данни са градивните елементи за всички транзакции, като клиенти, доставчици, активи, материали и други. Те са относително статични. Необходими са специални разрешения за да се манипулират или създават основните данни. Данните за транзакциите, като продажби, покупки и фактуриране, се променят непрекъснато. Всеки SAP модул има своя собствена независима организационна структура, която определя взаимоотношенията между различните работни групи и отдели. Следващата таблица подчертава някои, но не всички аспекти на организационните структури.

***Таблица 1.2****:* Организационни структури *в САП.*



Модула за финанси съдържа **сметкоплан**, който изброява всички сметки и се основава на счетоводни правила, определени от държавата. Следва **компания**, на чието ниво могат да се създават индивидуални финансови отчети. По-долу са **фирмените кодове**. Една компания може да има множество кодове и всеки фирмен код може да има **множество бизнес области**. Пример за бизнес област в рамките на фирмен код би било **производство**.

При продажбите и дистрибуцията **търговската организация** е на най-високо ниво и цялото отчитане на продажбените дейности се извършва на ниво търговска организация. След това е **каналът за дистрибуция**, представляващ начин, по който се достига до клиентите. Следващият компонент е **дивизия**, която обработва конкретна продуктова линия. Една компания може да има едно подразделение, което продава потребителски продукти и отделно подразделение за консултантски услуги. Комбинацията от търговска организация, дистрибуционен канал и подразделение се нарича **търговска зона**.

На първо ниво в модула за управление на материалите стои **заводът**. Той може да бъде производствено съоръжение, дистрибуторски център или дори офис. **Местата за съхранение** в заводите са физическите места, където се складират запасите. **Организациите за закупуване** водят преговори и дейности по доставки от доставчици. Организациите за закупуване могат да се справят с доставките за множество фирмени кодове или могат да бъдат ограничени и да извършват покупките за конкретен завод. **Организациите за покупки** често се разделят на групи за покупки, които се занимават със специфични аспекти, като специфични материали в рамките на процеса на закупуване.

В управлението на човешките ресурси отново срещаме **фирмен код**, който е собствена независима счетоводна единица. **Персоналните области** стоят в рамките на фирмения код, като те самите се разделят на **подобласти**. Организационните структури определят взаимоотношенията между различните работни групи и отдели. Те биват конфигурирани първоначално и обикновено остават статични, освен ако дадена компания не придобие друга или се откаже от части от своя бизнес.

Фокусът на дисертацията е върху модула за продажби и дистрибуция. Цялата функция на този модул е да „продава стоки и услуги на клиентите на предприятието“. Основен обект на модула за продажби и дистрибуция е „материал“. Той може да се закупи, произведе, продаде, върне и/или прехвърли. Фигура 1.1. представя данните, които са част от този запис: продажби и логистика, количества за материали и доставки и други:



***Фигура 1.1****: Данни, част от записа за материал в модул за продажби и дистрибуция.*

Вторият най-критичен обект от гледна точка на продажбите и дистрибуцията е главният запис на клиента. Той представлява субект, на който се продават стоки и/или услуги. Част от атрибутите на запис на клиент са: името и адреса на клиента, условия на плащане, специфични опции за ценообразуване, които може да се прилагат само за този клиент, както и различните партньорски функции. ERP системите поддържат няколко версии на клиент. Първата партньорска функция представлява субектът, на който продаваме стоки и услуги (sold‑to party). Партньор за доставка (sold‑to partner) представлява мястото, където изпращаме стоки или услуги. С други думи, това е адреса за доставка на клиента, който може да е различен от адреса на купувача. Партньорът за фактуриране представя къде трябва да бъде изпратена фактура. Местоположението, на което се изпращат фактури, може отново да бъде напълно различно от това на адреса за доставка. Функцията партньор на платеца представлява субектът, който отговаря за плащането на фактура. В много случаи и четирите функции може да имат едни и същи данни, но в някои случаи тези четири функции са не само различни физически адреси, но и напълно различни обекти. Фигура 1.2. илюстрира на кратко четирите вида:



***Фигура 1.2****: Видове партньорски функции.*

*TODO: ценообразуване / поръчките / продажби / доставки*

### 1.1.2. Характеристика на управлението на веригата за доставки

Управлението на веригата за доставки представлява надзора над информацията за материалите и финансите, докато те се движат в процеса от производител до потребител. Състои се от взаимосвързани мрежи, канали и предприятия, комбинирани в предоставянето на продукти и услуги, изисквани от крайния клиент. Практиката за управление на веригата за доставки черпи до голяма степен от областите на индустриалното, системното инженерство, управлението на операциите, логистиката, информационните технологии. Ролята на този модел е насочена към продажбата на продукти, увеличаване на производителността, ефективността, навременността, намаляване на разходите за преработване.

ERP системите дават възможност за комуникация с доставчици и клиенти, по начин, чрез който обменяме информация, подобряваща преминаването на материали, консумативи и услуги. Тази взаимосвързаност бива разгледана от гледна точка на клиентското обслужване. Управлението на веригата за доставки се дефинира като проектиране, планиране, изпълнение, контрол, наблюдение на дейностите по веригата за доставки с цел създаване на нетна стойност, изграждане на конкурентна инфраструктура, използване на световната логистика, синхронизиране на предлагането с търсенето и измерване на производителността в световен мащаб. Глобалните компании се стремят към пазари, където имат предимството както с доставките, така и с клиентите. Разглеждайки предлагането с търсенето, осигурявайки поток от информация, който определя, в кой момент трябват повече запаси или например кои продукти трябва да биват предлагани в определен момент. Интегрираното планиране и изпълнение на процеса, необходим за оптимизиране на потока от материали, информация и финансов капитал, включва планиране на търсенето, снабдяване, производство, управление на запасите и съхранение, транспортиране или логистика, връщане на излишни или дефектни продукти.

Управление на веригата за доставки се използва за да осигуряване на непрекъснато снабдяване, управление на договорните задължения и предотвратяване на прекъсвания в доставките. Също така управление на риска, спазването на организационните разпоредби и споразумения. В зависимост от естеството на бизнеса, трябва да се спазват индустриални и правителствени изисквания, да се извличат анализи на поръчките.

Управление на веригата за доставки служи за увеличаване на печалбата, увеличаване на паричния поток, подобряване обслужването на клиентите, намаляване на оперативните разходи, подобряване на финансовите позиции.

За планиране на търсенето, приемаме търсенето от клиента, поддържано в ERP системите. Също така помагат в планирането за производство или снабдяване с материали, транспорта и логистиката. Тези модули помагат за изграждането на веригата. SD, MM, PP съдържат тези градивни елементи. Анализирането на информацията идваща от ERP, в частност взаимоотношенията с клиенти и доставчици, полага добра основа за изграждането на персоналната верига за доставки. Тъй като процесът е сложен, ERP не поддържа такъв подмодул по подразбиране. В случая разглеждането на информацията предпоставя изготвянето на стратегия и план. Веригата за доставки започва с управлението на материалите, включва логистично изпълнение и производствено планиране.

Както беше споменато в точка 1.1.1, логистичния модул, поддържащ оферти, обработката на поръчки, доставки и транспорт. Свързани с веригата за доставки, прогнозите за продажбите са основно разбиране какво биха поискали клиентите в бъдеще. Тази прогноза бива въведена в производственото планиране. Друг аспект на логистичния модул е функционалността за доставки към клиенти. Логистичното изпълнение, като част от модула за управление на материалите, е мястото, където ще открием нашите складови операции. Ключов момент в логистиката са трансферите до местата за съхранение, при постъпване на материалите. Планирането на производството е друг подмодул в рамките на логистиката. Използва се за създаване на графици за производство и доставки за готов продукт и компонентни материали.

На фигура е представен модел, показващ потока от информация по отношение на търсенето чрез реалното издаване на готовите материали за клиент. Основните данни са част от почти цялата ERP: материали, доставчици, клиенти се създават под формата на **вътрешни,** основни записи. Фигурата илюстрира търсене от MRP, ( клиенти, които имат нужда от определено количество продукт). От гледна точка на MRP, това е планиране на изискванията за материали. Ако разполагаме с продукта или услугата, можем да го изпратим директно на клиента. В противен случай трябва да изпратим вътрешна предложение за закупуване на материал. След като бъде одобрено, материалът бива получен в инвентар. Ако компанията има на склад, този материал се премества там. Ако част от процес подлежи на инспекция, се насочваме към модула за управление на качеството, където, след като материалите бъдат инспектирани, те се освобождават за неограничена употреба. Въз основа на поръчките, материалите биват издадени и след това започва изходящия логистичен процес. 

***Фиг 1.5****: Принципи за проектиране на облачни системи.*

## 1.2. Възможности за дигитализация на процесите по управление чрез прилагане на облачни технологии

В последните години облачните технологии се превърнаха във водеща тенденция в софтуерната индустрия. Те предоставят нов начин за изграждане на големи и сложни системи, като по този начин използват пълноценно съвременните практики за разработка на високо-качествен софтуер и налична инфраструктура. Това променя начина на проектиране, интегриране и внедряване на системите. Облачно базираните решения са проектирани да приемат бързо промените, да обслужват голям мащаб от хора и да бъдат устойчиви на всякакъв вид натоварване или хакерски атаки (Vettor, 2022).

Организацията Cloud Native Computing Foundation предлага следното определение: "*Технологиите, базирани на облак, дават възможност на организациите да създават и изпълняват приложения в модерни, динамични среди като публични, частни и хибридни облаци, чрез мрежи от услуги и микроуслуги. Качества на системите са устойчивост, висока наличност и достъпност, мащабируемост и управляемост, които са от критично значение за много от бизнес единиците. Автоматизацията на тези процеси позволява на инженерите да правят промени, с голямо въздействие, но с минимални усилия."*

Приложенията стават все по-сложни, като изискванията, от страна на потребителите, стават все повече и повече, главно насочени към бърза реакция и иновативни функции. Проблеми с производителността или повтарящи се грешки вече не са приемливи.

Предимствата на облачните системи поставят бизнеса една стъпка пред конкурентите. Бизнес системите се развиват от способностите на бизнеса да бъдат инструменти за стратегическа трансформация, която ускорява растежа на компанията. Облачно базираните системи се свързват главно с бързина (Smith, 2022). Незабавното пускане на иновативните идеи на пазара е важна тема за всички модерни  компании, например следните компании са приложили успешно тези техники:

• Netflix има над 600 услуги в производствена среда. Стотици пъти на ден се изпълняват нови внедрявания и разгръщания на съществуващи;

• Uber има над 1000 услуги в производствена среда. Обновяват се няколко хиляди пъти всяка седмица;

Както е видно, бизнесът на тези две компании се базира на системи, които се състоят от стотици независими микроуслуги. Този архитектурен стил им позволява бързо да реагират на пазарните условия като постоянно актуализират малки, но важни области. Скоростта на облачния носител се дължат на редица фактори, като на първо място е инфраструктурата на изчислителните ресурси.

На фигура 1.3. са показани пет основополагащи стълба, осигуряващи основата за базирани в облак системи.



***Фиг 1.6****: Фундаментни стълбове на облачните системи.*

Проектирани да процъфтяват в динамична, виртуализирана облачна среда, облачните системи използват широко „платформата като услуга“ (PaaS) изчислителна инфраструктура и управлявани услуги. Те третират основната инфраструктура като за еднократна употреба - осигуряват се за минути и се преоразмеряват, мащабират или унищожават при поискване - чрез автоматизация.

Нека разгледаме DevOps концепция наречена: „Pets vs. Cattle“ (Menchaca, 2018). В традиционния център за данни сървърите се третират като домашни любимци (pets), като всеки един от тях представлява физическа машина, която трябва да бъде поддържана. Мащабирането се случва като се добавят ресурси към нея. Възникването на проблем в сървър, рефлектира върху всички потребители. Моделът на услугата Cattle е по-различен. При него всеки ресурс се предоставя като виртуална машина или контейнер. Те са идентични и присвояват системни идентификатори като Service-01, Service-02 и т.н. Мащабирането се случва като се създадат нови екземпляри. Ако един от тях стане недостъпен, друг поема неговата роля. Облачните системи поддържат този модел. Те продължават да работят, независимо от машините.

Дванадесетфакторното приложение (Wiggins, A., 2017), представено в таблица 1.2, е известна методология за конструиране на облачно базирани приложения. Изготвена от разработчици в Heroku, компания, преглагаща, платформа като услуга, описва набор от принципи и практики, които разработчиците следват, за да създават приложения, оптимизирани за модерни облачни среди. Много практици смятат Twelve-Factor за солидна основа за изграждане на облачни приложения, защото е приложим за всяко уеб-базирано приложение. Системите, изградени на този принцип, могат да се внедряват и мащабират бързо, както и да добавят нови или да променят съществуващи функции, за да реагират бързо на пазарните промени.

***Таблица 1.6****: Описание на методологията на дванадесетте фактора.*

| **Factor** | **Explanation** |
| --- | --- |
| 1 - Code Base | A single code base for each microservice, stored in its own repository. Tracked with version control, it can deploy to multiple environments (QA, Staging, Production). |
| 2 - Dependencies | Each microservice isolates and packages its own dependencies, embracing changes without impacting the entire system. |
| 3 - Configurations | Configuration information is moved out of the microservice and externalized through a configuration management tool outside of the code. The same deployment can propagate across environments with the correct configuration applied. |
| 4 - Backing Services | Ancillary resources (data stores, caches, message brokers) should be exposed via an addressable URL. Doing so decouples the resource from the application, enabling it to be interchangeable. |
| 5 - Build, Release, Run | Each release must enforce a strict separation across the build, release, and run stages. Each should be tagged with a unique ID and support the ability to roll back. Modern CI/CD systems help fulfill this principle. |
| 6 - Processes | Each microservice should execute in its own process, isolated from other running services. Externalize required state to a backing service such as a distributed cache or data store. |
| 7 - Port Binding | Each microservice should be self-contained with its interfaces and functionality exposed on its own port. Doing so provides isolation from other microservices. |
| 8 - Concurrency | When capacity needs to increase, scale out services horizontally across multiple identical processes (copies) as opposed to scaling-up a single large instance on the most powerful machine available. Develop the application to be concurrent making scaling out in cloud environments seamless. |
| 9 - Disposability | Service instances should be disposable. Favor fast startup to increase scalability opportunities and graceful shutdowns to leave the system in a correct state. Docker containers along with an orchestrator inherently satisfy this requirement. |
| 10 - Dev/Prod Parity | Keep environments across the application lifecycle as similar as possible, avoiding costly shortcuts. Here, the adoption of containers can greatly contribute by promoting the same execution environment. |
| 11 - Logging | Treat logs generated by microservices as event streams. Process them with an event aggregator. Propagate log data to data-mining/log management tools like Azure Monitor or Splunk and eventually to long-term archival. |
| 12 - Admin Processes | Run administrative/management tasks, such as data cleanup or computing analytics, as one-off processes. Use independent tools to invoke these tasks from the production environment, but separately from the application. |

В книгата Beyond the Twelve-Factor App авторът Кевин Хофман описва подробно всеки от оригиналните 12 фактора, като добавя три допълнителни, които отразяват модерен дизайн на облачни приложения.

***Таблица 1.7****:ТОДО.*

| **New Factor** | **Explanation** |
| --- | --- |
| 13 - API First | Make everything a service. Assume your code will be consumed by a front-end client, gateway, or another service. |
| 14 - Telemetry | On a workstation, you have deep visibility into your application and its behavior. In the cloud, you don't. Make sure your design includes the collection of monitoring, domain-specific, and health/system data. |
| 15 - Authentication/ Authorization | Implement identity from the start. Consider [RBAC (role-based access control)](https://learn.microsoft.com/en-us/azure/role-based-access-control/overview) features available in public clouds. |

Проектирането и внедряването на облачно базирани работни натоварвания може да бъде предизвикателство. Microsoft Well-Architected Framework (Stanford D. et al, 2022) предоставя набор от ръководни принципи, които се използват за подобряване качеството на работното натоварване. Следната таблица представя пет важни стълба на добра архитектурата.

***Таблица 1.8****: Стандартни за добри практики на облачаната индустрията.*

| **Tenets** | **Description** |
| --- | --- |
| [Cost management](https://learn.microsoft.com/en-us/azure/architecture/framework/#cost-optimization) | Focus on generating incremental value early. Apply Build-Measure-Learn principles to accelerate time to market while avoiding capital-intensive solutions. Using a pay-as-you-go strategy, invest as you scale out, rather than delivering a large investment up front. |
| [Operational excellence](https://learn.microsoft.com/en-us/azure/architecture/framework/#operational-excellence) | Automate the environment and operations to increase speed and reduce human error. Roll problem updates back or forward quickly. Implement monitoring and diagnostics from the start. |
| [Performance efficiency](https://learn.microsoft.com/en-us/azure/architecture/framework/#performance-efficiency) | Efficiently meet demands placed on your workloads. Favor horizontal scaling (scaling out) and design it into your systems. Continually conduct performance and load testing to identify potential bottlenecks. |
| [Reliability](https://learn.microsoft.com/en-us/azure/architecture/framework/#reliability) | Build workloads that are both resilient and available. Resiliency enables workloads to recover from failures and continue functioning. Availability ensures users access to your workload at all times. Design applications to expect failures and recover from them. |
| [Security](https://learn.microsoft.com/en-us/azure/architecture/framework/#security) | Implement security across the entire lifecycle of an application, from design and implementation to deployment and operations. Pay close attention to identity management, infrastructure access, application security, and data sovereignty and encryption. |

Облачните системи поддържат ориентирания към микроуслуги архитектурен стил за конструиране на системи. Това е подход за изграждане на сървърно приложение като набор от малки, но високо-качествени подуслуги. Съотвено, клиентите, на сървърните услуги, могат да бъдат отделни приложения, които да се поддържат и управляват самостоятелно. Всяка услуга работи в собствен процес и комуникира с други процеси, използвайки различен тип и вид протоколи като: HTTP/HTTPS, WebSockets, AMQP и мн други. Всеки микросървис притежава специфична бизнес способност. Предимства на това архитектурен стил са:

• Всяка микроуслуга може да бъде проектирана, разработена и внедрена независимо една от друга, което осигурява възможно за независима работа по отделни области на приложението;

• Работата може да бъде дистрибутирана между отделни екипи;

• Проблемите са по-изолирани;

• Позволява използването на различни технологии;

Микроуслугите насърчават фактор #6 от принципите на дванадесетфакторното приложение.

…….

***Таблица 2.2****: Принципи за проектиране на облачни системи.*

|  |  |  |
| --- | --- | --- |
| име на български | име на английски | Описание (тук ще има доста цитати) |
| Разделяне на грижите | Separation of Concerns | всеки обект и модул трябва да бъде в своя собствена грижа и контекст |
| Капсулиране | Encapsulation |  |
| Инверсия на зависимостта | Dependency Inversion |  |
| Изрични компоненти | Explicit Components |  |
| Единична отговорност | Single Responsibility |  |
| Не се повтаряйте | Don’t Repeat Yourself |  |
| Устойчивост и невежество относно инфраструктурата | Presentation Ignorance |  |
| Ограничени контексти | Bounded Contexts |  |

Всяка микроуслуга притежава своя собствена логика и данни, в рамките на автономен жизнен цикъл. Концептуалните модели се различават между подсистемите или микроуслугите. Този принцип е заложен в дизайнът, управляван от домейн (DDD), където всяка услуга притежава свой модел на домейн (данни плюс логика и поведение).

Традиционният (монолитен) подход, използван в много приложения, притежава единична централизирана база данни (или няколко бази), която често е нормализирана SQL база, използвана за цялото приложение и всички негови вътрешни подсистеми. Този подходът изглежда по-прост, позволява повторно използване на обекти. В крайна сметка се стига до огромни таблици, които обслужват много различни подсистеми, включващи атрибути и колони, които в повечето случаи не са необходими. Монолитните приложения: ACID транзакции и SQL език, като и двете работят във всички таблици и данни, свързани с приложението. Това допринася за сравнително лесно комбиниране данни от множество таблици.

Достъпът до данни, става много по-сложен в архитектурата на микроуслуги. Дори когато използвате ACID транзакции в рамките на микроуслуга или ограничен контекст, е изключително важно да се има предвид, че данните, притежавани от всяка микроуслуга, са частни за тази и трябва да бъдат достъпвани синхронно, чрез нейните API крайни точки (REST, gRPC, SOAP и т.н.) или асинхронно чрез съобщения (AMQP). Капсулирането на данните гарантира, че микроуслугите са слабо свързани и могат да се развиват независимо една от друга. Ако множество услуги имат достъп до едни и същи бази данни, актуализациите на схемата ще изискват координация. Това би нарушило автономността на жизнения цикъл. Когато един бизнес процес обхваща множество микроуслуги трябва се да използва т.н. „евентуална последователност“. Това е много по-трудно за изпълнение от обикновените SQL съединения. Различните микроуслуги често използват различни видове бази данни. Съвременните приложения съхраняват и обработват различни видове данни. За някои случаи на употреба NoSQL база данни като Azure CosmosDB или MongoDB може да има по-удобен модел, както и да предлага по-добра производителност и мащабируемост от SQL база данни. Микроуслугите често използват смесица от SQL и NoSQL бази данни, което се нарича подход на „полиглотна устойчивост“ (Polyglot Persistence).

Концепцията за микроуслуга произлиза от модела на ограничен контекст (Bounded Context) в управлявания от домейн дизайн (DDD). DDD се занимава с големи модели, като ги разделя на множество BC и определя техните граници. Всеки BC има собствен модел и база данни. По същия начин всяка микроуслуга притежава свързаните с нея данни. В допълнение, BC притежават така нареченият повсеместен език, който помага на комуникацията между разработчици на софтуер и експерти в бизнеса. Ограничен контекст може да бъде отделна услуга или е просто логическа граница.

## 1.3. Специфики при управление на поръчките от клиенти в производствено предприятие

Софтуерните приложения трябва да имат познание за потребителя или процеса, който ги извиква. Потребителят или процесът, взаимодействащ с приложение, е известен като принципал за сигурност, а процесът на удостоверяване и упълномощаване на тези принципали е известен като управление на самоличността (Vettor, 2022). Простите приложения могат да включват цялото им управление на самоличността в приложението, но този подход не се мащабира добре с много приложения и много видове принципали за сигурност. Windows, например, поддържа използването на Active Directory за предоставяне на централизирано удостоверяване и оторизация. Въпреки че това решение е ефективно в рамките на корпоративни мрежи, то не е предназначено за използване от потребители или приложения, които са извън домейна.

Удостоверяването е процес на определяне на самоличността на потребител. Упълномощаването е актът на предоставяне на удостоверено разрешение за извършване на действие или достъп до ресурс (Wike R, 2022).

Съвременните решения за самоличност обикновено използват токени за достъп, които се издават от защитена услуга/сървър (STS) на принципал за сигурност, след като тяхната самоличност бъде определена (Anil N, 2022).

Изискванията към STS:

• Защитават ресурсите;

• Удостоверяват потребителите;

• Осигуряват управление на сесии;

Важни функции на сървъра за самоличност са:

• услуга за удостоверяване, която да работи в централизиран процес;

• Единично влизане/излизане за множество приложения;

• Покрива индустриалните стандарти OpenID Connect и OAuth 2.0;

• Шлюз към Google, Facebook и др;

Удостоверяването чрез токени е механизъм без състояние, тъй като никаква информация за потребителя не се съхранява в паметта на сървъра или базaтa от данни, за разлика от бисквитките (Gichuhi , 2021).

Стандарт (RFC 7519) за уеб приложенията е JSON уеб токен (JWT). Той се състои от три части:

• Заглавна - JSON обект, кодиран във формат base64. Съдържа информация за типа на токена и алгоритъма за криптиране;

• Полезен товар - съдържа информация за текущия потребител (потребителско име, роля и др). Тук не трябва да се включват чувствителни данни, защотп лесно се могат да бъдат декодирани със публични сайтове като jwt.io;

• Подпис – Използва се от сървъра, за да провери дали токенът е валиден. Той се генерира чрез комбиниране на двете части (заглавна и полезен товар) заедно. Базира се на таен ключ, който само сървърът за удостоверяване знае. По този начин злонамерен потребител не може да фалшифицира валиден токен;

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiI0YTY2ZWNmNC1iZDdjLTQ3ODQtYmViOS1jZGM0MzQzZGY3MWYiLCJ1bmlxdWVfbmFtZSI6Im15QG15LmNvbSIsIm5iZiI6MTU5NjEzMzk3OCwiZXhwIjoxNTk2NzM4Nzc4LCJpYXQiOjE1OTYxMzM5Nzh9.W7k3UXA1g3TKxt-hR9a-mgCAcCsKjEyGTxBv5Dt79y8

***Фигура 1.3.1****: Пример за токен.*