|  |  |
| --- | --- |
|  | ИКОНОМИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА  ФАКУЛТЕТ ИНФОРМАТИКА  КАТЕДРА ИНФОРМАТИКА |

**Йордан Иванов Йорданов**

**Облачна информационна система за управление на поръчките от клиенти в производствено предприятие**

**ДИСЕРТАЦИЯ**

за присъждане на образователна и научна степен „доктор” по докторска програма   
„Информатика”, професионално направление "Информатика и компютърни науки“

**Научен ръководител:** доц. д.н. Павел Петров

ВАРНА, 2024

СЪДЪРЖАНИЕ

[Списък на използваните съкращения 3](#_Toc161493489)

[Въведение 4](#_Toc161493490)

[Глава 1. Проблеми на информационното осигуряване при управление на поръчките от клиенти 7](#_Toc161493491)

[1.1. Управление на веригите от поръчки и доставки и тяхното приложение в системите за планиране на ресурси 7](#_Toc161493492)

[1.2. Рационализиране на процесите чрез персонализиранa информационна система 17](#_Toc161493493)

[1.3. Възможности за централизация на процесите по управление чрез прилагане на облачни технологии 20](#_Toc161493494)

[1.4. Управление на бизнес процесите чрез ориентиран към домейн дизайн 31](#_Toc161493495)

[Глава 2. Архитектура на облачна система за управление на поръчки от клиенти 48](#_Toc161493496)

[2.1. Концептуален модел на облачната система за управление на поръчките 48](#_Toc161493497)

[2.2. Логически модел на облачна система за управление на поръчки 57](#_Toc161493498)

[2.2.1. Модули, поддържащи поръчки и доставки 57](#_Toc161493499)

[2.2.2. Детайлизиране на модулите за поръчки и доставки 62](#_Toc161493500)

[2.2.3. Модул за управление на потребителските профили 69](#_Toc161493501)

[2.3. Kомуникационни модели между модулите 74](#_Toc161493502)

[2.4. Функционалност и потребителски интерфейс 84](#_Toc161493503)

[Глава 3. Изграждане и използване на облачна система за производствено предприятие "Хейделберг Цимент Девня" АД 89](#_Toc161493504)

[3.1. Обща характеристика на дейността на компанията 89](#_Toc161493505)

[3.2. Избор на технологични средства за реализация на системата 92](#_Toc161493506)

[3.3. Физическа реализация на системата 99](#_Toc161493507)

[3.5. Oценка на приблизителните разходи 109](#_Toc161493508)

[3.6. Мониторинг и системен дневник 110](#_Toc161493509)

[3.7. Техники за анализ чрез когнитивни услуги и машинно обучение 114](#_Toc161493510)

[Заключение 116](#_Toc161493511)

[Списък с фигури и таблици 117](#_Toc161493512)

# Списък на използваните съкращения

|  |  |
| --- | --- |
| Съкращение | Пълно Наименование |
| API | Application Programming Interface |
| ERP | Enterprise Resource Planning |
| SCM | Supply Chain Management |

# Въведение

В съвременната ера на глобализация, производствените компании се сблъскват с проблеми, свързани с обработването на клиентски поръчки в рамките на своите вериги за доставка, въпреки наличието на модерни системи за управление на ресурсите и информационна логистика. Проблемите имат различен характер, обхващайки управлението на множество доставчици, транспортни маршрути, спецификации и качество на продуктите и много други, които водят до риск от забавяния и увеличение на оперативните разходи. Подобни проблеми ограничават способността на компаниите да бъдат гъвкави в търсенето и предлагането и могат да се отразят негативно на удовлетвореността на клиентите, когато поръчките се бавят или не се изпълняват изобщо.

Допълнителни проблеми могат да възникнат при интеграция между различни информационни системи, което може да доведе до трудности свързани с трансфера на данни, различни формати на данните и други, които могат да повлияят на процеса във веригата на доставки. Например, при интегрирането на корпоративни системи за планиране на ресурсите (ERP) с технологии като Интернет на нещата, е възможно да възникнат проблеми със сигурността. Освен това производствените предприятия са длъжни да се съобразяват със законите и стандартите на юрисдикцията в която работят и това налага освен доброто познаване на нормативните актове, тяхното интегриране в информационните системи.

Актуалността на изследваната тема се обуславя от тенденцията облачните технологии да се превръщат в инструмент със стратегическо значение за бъдещ растеж, модернизация и цифрова трансформация на производствени предприятия. Тази тенденция продължава да се развива, тъй като все по-голям брой компании използват възможностите на облачните платформи, за да въвеждат иновативни идеи, повишавайки своята конкурентоспособност. В тази връзка, проучването изследва проблемите и решенията, свързани с внедряването на информационни системи в техните икономически и технологични аспекти.

**Тезата** на разработката е, че облачните системи биха могли да подобрят процеса на управление на поръчките за продажби и цялостната верига за доставки на производствено предприятие, като предоставят адаптивни софтуерни решения с интегриран потребителски контрол и взаимодействие с крайния клиент. Докторска теза се основава на начините, по които производствените компании управляват информацията за поръчките на клиентите – как тя се събира, пази, обработва и предава. Тази информация е важна, тъй като е основата за организиране на доставката на продукти, включително тяхното зареждане, транспортиране и разтоварване. Изследването обхваща всички системи и процедури, които са част от управлението на веригата за доставки и съпътстващата ги информация от етапа на производство до доставянето на продуктите на крайните клиенти, с фокус върху ефективността.

**Обект** на изследване в дисертацията са процесите във веригите за доставки в производствените предприятия. Това обхваща всички системи и процеси, които са включени в управлението на потока от стоки и информация от момента на тяхното производство до момента, в който те достигат до крайния потребител и включва всички аспекти на логистиката, като се вземат предвид нуждите на клиентите и оптималното използване на ресурсите

**Предметът** на изследване включва технологиите и научните методи за автоматизиране на логистичните процеси, използвайки съвременните постижения на дигиталните платформи и средства.

**Целта** на изследванетое да се разработи облачна информационна система за управление на поръчките от клиенти и да се оцени въздействието и върху действия и процесите по планиране на ресурсите и управлението на веригите за доставки в производствено предприятие, като се имат в предвид проблемите, решенията и технологичните средства, свързани с киберсигурността, защитата на данните, логистична синхронизация и други.

За постигане на поставената цел са решени следните задачи:

1. Изследване на взаимосвързаността между системите за управление на веригата за доставки, електронна логистика и планиране на ресурсите в производствени предприятия, включително динамиката на тяхното взаимодействие и интегрирането на тези компоненти за повишаване на ефективността на бизнес процесите;
2. Анализ на съвременните мобилни и уеб технологии и техния потенциал да участват в рационализирането на процесите, свързани с получаването и обработката на клиентски заявки, доставката на готови продукти, избор на канали за дистрибуция и управление на логистичните услуги;
3. Изследване на необходимите компоненти за разработване на детайлна архитектура, свързана с проектирането на облачна информационна система;

В изследването са използвани исторически, сравнителен, системен и комплексен подход. Също така са приложени методи на логически анализ, създаване, моделиране и алгоритмизация на облачна информационна система.

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

## Управление на веригите от поръчки и доставки и тяхното приложение в системите за планиране на ресурси

В последните десетилетия, управлението на веригите от поръчки и доставки се превръща във важен компонент за успеха на производствените предприятия в условията на нарастваща конкуренция в глобалния световен пейзаж и за това веригите за доставки са обект на засилено изследване с цел оптимизиране. В исторически аспект организациите са разчитали предимно на ръчни процеси и прости инструменти за планиране и координиране на своите операции, а тези ранни методи често водят до грешки, което затруднява ефективното управление.

В научната литература съществуват множество различни дефиниции за термина „*верига на доставките*“. В своята книга, авторите (Chopra & Meindl, 2016) разглеждат веригата за доставки като „*етапите, които пряко или непряко участват в изпълнението на заявките на клиента. Веригата на доставки включва не само производителя и доставчиците, но и превозвачите, складовете, търговците на дребно и самите клиенти*“.

Други автори (Khan & Yu, 2019) дефинират веригата за доставки, като: „*мрежа от съоръжения и възможности за дистрибуция, която изпълнява функциите на доставка на материали, превръщането на тези материали в междинни и готови продукти и разпространението на тези готови продукти на клиентите.*“

Друга дефиниция, която се предлага от (Matinheikki, et al., 2022)е, че веригата за доставки представлява „*съвкупност от процеси и ресурси, необходими за извършване и доставка на продукт на крайния потребител*“ или също „*канал за ефективно движение на материали, продукти, услуги или информация от доставчици към клиенти*“.

В настоящото изследване се приема определението на (Jamaluddin & Saibani, 2021), дефиниращо понятието, като „*ясно очертана верига от свързани двойки логистични звена „доставчик – получател“ (структурирани подразделения на фирмата и/или логистичните й партньори), по която конкретната стока и/или услуга се доставя на крайния потребител в съответствие с неговата заявка и изисквания*“.

Транспортирането на продуктите по верига на доставки, се осъществява от производителя до крайния потребител. Обратната логистика представлява процеса на движение на стоките в обратна посока, а именно от крайния потребител обратно към производителя. Обратната логистика включва различни дейности, като връщане на стоки от страна на потребителите, които не отговарят на техните очаквания, връщане на стоки с цел ремонт, рециклиране, замяна и други (Plaza-Úbeda, et al., 2021).

В книгата си (Gupta, 2013) прави сравнителна характеристика между права и обратната верига за доставки, като извежда основните аспекти, които ги отличават (Таблица 1.1).

Таблица 1.1.  
Сравнение между права и обратна верига за доставки   
Източник: Gupta, 2022

|  |  |  |
| --- | --- | --- |
| **Показатели** | **Права верига за доставки** | **Обратна верига за доставки** |
| Оптимизация | Базирана на оптимизиране на печалбата и разходите. | Базирана на екологичните принципи и закони, както и на оптимизирането на печалбите и разходите. |
| Прогнозиране | Сравнително по-лесно прогнозиране на търсенето на продукти. | По-трудно прогнозиране за връщане на продукти. |
| Качество на продукта | По-малко вариации в качеството на продукта. | Големи различия. |
| Време за обработка | Времето и стъпките за обработка са добре дефинирани. | Времето и стъпките за обработка зависят от състоянието от върнатия продукт. |
| Транспорт | Стоките се транспортират от едно място до много други места. | Върнатите продукти се събират от много места и пристигат в едно. |
| Оценка на разходите | Сравнително лесна, разчитаща на счетоводни системи. | Сравнително сложно определяне и представяне на разходи. |
| Опаковка | Стандартна структура на продукт. | Модифицирана структура на продукт. |
| Прозрачност на процесите | Проследяването на движението в реално време. | Липсата на възможности за обратна връзка. |

Може да се обобщи, че правите вериги за доставки се фокусират върху печалбата и оптимизирането на разходите, докато обратните вериги за доставки дават приоритет на спазването на законите и обратната връзка с клиента. Те се различават значително по показателите прогнозиране на търсенето, вариации в качеството на продукта, време за обработка, транспортиране на стоки, оценка на разходите, конкурентно предимство, опаковка на продукта, структура на продукта и прозрачност на процеса.

Важен елемент в управлението на веригата за доставки е логистиката. В своите документи, (Европейска Логистична Асоциация, 2021) дефинира логистиката като "*организация, планиране, контрол и реализация на придвижването на стоковия поток от проектирането и закупуването, през производството и разпределението до крайния потребител с цел удовлетворяване изискванията на пазара с минимални операционни и капиталови разходи*".

Тези дейности се идентифицират по различни параметри (TM Le, 2020) като размерност, начална и крайна точка на движение, дължина на пътуването, скорост, време на движението, време на престой, вид на използваните транспортни средства, условия на транспортиране. В тази смисъл, (Wichaisri & Sopadang, 2013) разглеждат логистичната система като устойчива мрежа от звена, които са взаимно свързани и управлявани централно чрез административни системи, които подпомагат управлението на целия логистичен процес. Целта е да се удовлетворят заявките и нуждите на клиентите, като се поддържа баланс между предлагането и търсенето.

В тази връзка се среща понятието „*информационен поток*“, което представлява обмен на данни, чрез документи или по друг начин като следствие на логистичния поток от стоки. Този информационен обмен играе роля в логистичната система и е от значение за управлението на веригата за доставки. Един от най-често срещаните модели на информационни потоци в логистиката е моделът на „*поток на поръчки от клиенти*“ (Zając & Swieboda, 2023). Всеки бизнес организира този процес по индивидуален начин, съобразено със специфичните си нужди и процедури.

В сборника си (Appolloni, et al., 2024) свързват стратегията и планирането на веригата за доставки с конкурентоспособността на производствени предприятия на световния пазар за строителни материали, докато редица автори и изследователи (Barata, et al., 2022) провеждат проучвания и откриват, че изследването на информацията за търсенето и предлагането може да помогне при определянето кога са необходими по-големи запаси или кои продукти трябва да бъдат предлагани в определен момент.

Според (Alzoubi & Yanamandra, 2020), интегриране на планирането и изпълнението на процесите в рамките на веригата за доставки включва комплекс от дейности по планиране на търсенето, управление на доставките, производство, контрол на запасите, складиране, транспортиране и други логистични операции. Този интегриран подход помага за оптимизиране на потока от материали, информация и финансови ресурси, както и за управление на връщането на излишни или дефектни продукти.

Според (Anca, 2019) логистичният мениджмънт включва планиране, организиране, координация и контрол на всички операции, които да удовлетворят изискванията на клиентите, като осигурят ефективно движение на стоките от точката на зареждане до точката на доставка. От тази гледна точка, логистичният мениджмънт включва вземане на стратегически, тактически и оперативни решения, свързани с развитието на логистичната дейност и взаимодействието с доставчиците и другите участници във веригата за доставки. (Bardakçi, 2020) допълва, че стратегическият логистичен план има за цел да реализира поставената стратегия и да осигури ефективното функциониране на логистичната мрежа. Освен това редица автори и изследователи (Calabrò, et al., 2020) посочват че, оптимизационните задачи в логистичното планиране могат да се разграничат на различни функционални области, като една от тях е управлението на поръчките. Този аспект включва регламентиране и оптимизация на всички етапи в цикъла на изпълнение на поръчките, включително приемането, обработката и доставката им (Tukamuhabwa, et al., 2021).

В този контекст, изборът на подходящи технически средства и технологии за приемане, обработка, въвеждане на електронен обмен на данни и установяването на параметри за качество на обслужване също са аспекти на управлението на поръчките. Според (Lee, et al., 2022) ефективното управление на материалните и съпътстващите ги потоци изискват координирано изпълнение на разнообразни функции и операции в рамките на логистичната система. Тази координация се изпълнява както на стратегическо, така и на оперативно ниво, тъй като влияе както на ритмичността на бизнес дейността, така и на ефективността на самата логистика.

Според (Hahn, 2020) за по-нататъшно изясняване на компонентите на верига за доставки, следва производството да се разгледа като процедура на трансформация на суровини и материали в готови стоки. Тази процедура е насочена към създаване на стойност, както за производителите, така и за потребителите. Съществуват различни стратегии за управление на производството, като например: проектиране по поръчка, производство по поръчка, сглобяване по поръчка, производство на склад и други. Всяка от тези стратегии има свои предимства и недостатъци, но целта им е обща - да създадат стойност както за клиентите, така и за самата организация. С прилагането на подходящата стратегия, предприятията могат да подобрят качеството и времето за доставка на своите продукти, да увеличат ефективността, като същевременно се намаляват излишните запаси и разходите, което в крайна сметка е от полза както за клиентите, така и за бизнеса. Планирането на производството съгласува търсенето с производствения капацитет и определя график за доставка на готови продукти (Chen, et al., 2009).

Възможните компоненти, които влияят върху управлението, могат да бъдат обобщени в следните групи: логистика, стратегия за веригата за доставки, планиране на веригата за доставки, доставки и управление на активи, управление на жизнения цикъл на продукта и закупуване, като корпоративните приложения подпомагат управлението на информационния поток. Те са представени графично на фиг. 1.1.



Фиг 1.1. Модел на елементите, съставящи управлението на веригите за доставки. Източник: Meehan & Pinnington, 2021, Адаптирано от автора

Моделът на фигурата представлява обобщение на движението на материали и финансовите процеси от момента на производство до достигането на продуктите и услугите до крайните клиенти, включвайки свързани мрежи, канали и предприятия, които си сътрудничат чрез няколко фундаментални аспекта на този модел:

* Увеличаване на печалбата: Оптимизирането на процесите във веригата за доставки може да доведе до по-голяма ефективност и по-ниски оперативни разходи, което накрая води до увеличаване на печалбата (Goodman & Broetzmann, 2019);
* Увеличаване на паричния поток: Управление на веригата за доставки може да подобри паричния поток на организацията, като ускори оборота на стоките и оптимизира управлението на финансовите ресурси;
* Подобряване на обслужването на клиентите: Операциите в рамките на веригата за доставки са насочени към предоставянето на добро обслужване на клиентите с цел продуктите да се доставят навреме и да отговарят на изискванията;
* Намаляване на оперативните разходи: Ефективното управление на веригата за доставки може да намали оперативните разходи, свързани с транспорта, складирането и обработката.

Като обобщение на посочените по-горе фактори SCM се свързва с поддържането на непрекъснати доставки, управлението на договорните задължения, запазването на връзките с клиенти, както и поддържането на конкурентно предимство на пазара. Според (Katsaliaki, et al., 2021) чрез внедряването на SCM системи, производствените предприятия могат да подобрят производителността си. Модулите на тези системи се интегрират взаимно, за да допринесат за ефективното управление на модела, представен по-горе. В крайна сметка контролът на разходите и качественото обслужване на клиентите, подобрява конкурентоспособността на пазара.

На базата на някои от последните класации на S&P Global Ratings за компании за строителни материали към 1 декември 2023 г., внедряването на системи за SCM е от първостепенно значение за поддържане и интеграция на бизнес процесите (Cataldo, et al., 2022) в редица компании като например CRH plc, Vulcan Materials Company, Martin Marietta Materials, Inc., Anhui Conch Cement и Heidelberg Materials AG използват SCM софтуер, предназначен за индустрията като SAP S/4HANA, Oracle SCM Cloud, Blue Yonder, Microsoft Dynamics 365, Kinaxis RapidResponse като същевременно използват и персонализирани решения, съобразени с техните логистични и оперативни проблеми. Тези системи са проектирани да оптимизират доставките, да рационализират производствените графици, да осигурят ефективни дистрибуторски мрежи, да подобряват вземането на решения, да насърчават сътрудничеството на доставчици, диспечери и клиенти и стимулират инициативи за устойчивост (Xia, et al., 2-23).

Основната цел на ЕRP 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 и доставка. Логистичната технология e стандартизирана последователност (алгоритъм) на изпълнение на отделни логистични функции, и/или процеси в логистичната система или в отделни нейни функционални области. 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. 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.

Една от водещите ERP системи в света е SAP, която е пусната за първи път преди 50 години в Германия. Според (Baumgartl, et al., 2021) в рамките на тази система се управляват всички функционални области на даден бизнес: човешки ресурси, 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 производствено предприятие може да има нужда, е достъпна и интегрирана в SAP. 92% от компаниите от Forbes Global 2000 и 86% от компаниите от Fortune 500 са клиенти на SAP. Освен това над 400 000 фирми в 180 различни държави използват софтуера на SAP и 77% от световните приходи от транзакции преминават през тази система (Duff, 2022).

В зависимост от компанията и много други фактори, внедряването на SAP може да отнеме дълго време и много ресурси. За сметка на това той предлага инструменти, които автоматично получават, съгласуват и извършват определени действия. Според статистически изследвания, SAP позволява събирането на големи количества данни, които могат да бъдат използвани за вземане на ефективни бизнес решения, които да помогнат за растежа на компанията.

Модулът за дистрибуция, заедно с модулът за управление на материали, дават възможност за изпълнение на поръчки, осигурявайки контрол на инвентара, операции по снабдяване, оптимални нива на запаси и навременна наличност на материали. В допълнение към тях е модулът за планиране на производството, в който се разпределят ресурси и се планират производствените дейности. Като допълнение, модулът за управление на качеството е част от защитата на веригата за доставки, внедрявайки проверки на качеството във всяка фаза от жизнения цикъл на продукта (Kumar, et al., 2022). Тези модули образуват архитектурна рамка в SAP за подобряване на прозрачността, гъвкавостта и устойчивостта на веригата за поръчки и доставки.

Значението на основните и транзакционни данни е предмет на изследване от редица автори. Според (Bier, et al., 2019)основните данни са градивните елементи за всички транзакции, като клиенти, доставчици, активи, материали и други. Те са относително статични, докато данните за транзакциите, като продажби, покупки и фактури, се променят непрекъснато. Всеки SAP модул има своя собствена независима организационна структура, която определя взаимоотношенията между различните работни групи и отдели. В са подчертани някои, но не всички аспекти на организационните структури.

Таблица 1.3.  
 Организационни структури в SAP  
(Business Process Integration with SAP ERP, 2013)

| **Финанси** | **Продажби и Дистрибуция** | **Управление на Материали** |
| --- | --- | --- |
| Сметкоплан (Chart of Accounts) | Организация на продажбите (Sales Organization) | Завод (Plant) |
| Компания (Company) | Дистрибуционен канал (Distribution Channel) | Местоположение на склада (Storage Location) |
| Код на компания (Company Code) | Дивизия (Division) | Организация на покупките (Purchasing Organization) |
| Бизнес област (Business Area) | Продажбена област (Sales Area) | Група покупки (Purchasing Group) |

В своята книга, (Becker & Hirn, 2017) описват модула за финанси, съдържащ сметкоплан, който изброява всички сметки и се основава на счетоводни правила, определени от държавата, следван от компания, на чието ниво могат да се създават индивидуални финансови отчети и 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 е производство(Baumgartl, et al., 2021). При продажбите и дистрибуцията търговската организация е на най-високо ниво и цялото отчитане на продажбените дейности се извършва на ниво търговска организация (Baumgartl, et al., 2021). Друг модул управлява каналът за дистрибуция, представляващ начин, по който се достига до клиентите. Следващият компонент е дивизия, свързана с управлението на движението на продуктите в 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 в SAP 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. Закупчиците водят преговори и дейности по доставки от доставчици и те могат да се справят с доставките за множество фирмени кодове или могат да бъдат ограничени и да извършват покупките за конкретен завод. Те се разделят на групи за покупки, които се занимават със специфични аспекти, като специфични материали в рамките на процеса на закупуване.

Проучвания (Knolmayer, et al., 1999) на софтуерните решения за управление на поръки установяват, че въпреки че системите за планиране на ресурсите осигуряват основна рамка, те често не успяват да се справят „самостоятелно“ с динамичния характер на съвременните вериги за доставки и се нуждаят от интегрирането на допълнителни софтуерни и хардуерни продукти. Проблеми като неефективност при обработката на данни в реално време и адаптиране към променящите се пазарни изисквания, определят необходимостта от персонализирани решения за централизация на процесите по вземане на решения. Рационализирането на процесите чрез персонализирани софтуерни решения не само преодоляват споменатите проблеми, но също така въвеждат и нови възможности за дигитализация, оптимизация и комуникация (Петров, et al., 2020).

## Рационализиране на процесите чрез персонализиранa информационна система

Според проучване на (Hasim, et al., 2020), мултинационалните компании за строителни материали срещат проблеми с ефективното управление на веригите за поръчки и доставки на своите многобройни търговски организации и канали за дистрибуция. От друга страна, остаряла практика е ERP системите да контролират основните операции в производствените компании (Verwijmeren, 2004). Въпреки това, в теорията и практиката се пренебрегва процесът на интегриране на ERP със системи за управление на взаимоотношенията с клиенти (Александрова, 2015), електронни устройства за обмен на данни, технологии за Интернет на Нещата (IoT), системи за управление на складове (WMS) и системи за управление на транспорт (Sullivan & Kern, 2021). Това изследване показва, че липсва единно становище относно използването на персонализирани софтуерни решения, които консолидират данни от посочените системи, за да предоставят на крайните клиенти достъп до тези данни докато целта е да се оптимизира оперативната ефективност, да се подобрят възможностите за вземане на решения, да се оптимизира комуникацията и координацията между заинтересованите страни, както и цялостната видимост на веригата за доставки.

В изследваните литературни и интернет източници липсва специално разработен модел на логистичните вериги и въз основа на литературен анализ (Моллов, 2022) може да се конструира прототип на подобен процесен модел. На Фигура 1.2 е представена централизирана система, която е свързана с всички други вътрешни системи на производствено предприятие, като същевременно предоставя публично достъпни данни на крайните клиенти.



Фигура 1.2. Процесен модел на веригата за доставки  
(Разработка на автора)

Диаграмата илюстрира серия от взаимосвързани системи, които улесняват транспортирането на материали и свързаните с тях информационни потоци. Проектирането на такава система следва да бъде съобразено с изисквания на конкретно предприятие (Luo, 2010), тъй като се вземат предвид растежът на цялостно разширяване на бизнеса, както и справянето с глобалните проблеми.

Унифицираната спедиторска платформа за логистични услуги, свързва различни заводи, географски райони и бизнес единици. Поддържа основни до сложни логистични изисквания (Петров, et al., 2020), позволявайки на бизнеса да се мащабира, докато се разраства. Помага при планирането и ефективното изпълнение на доставките, като по този начин спестява разходи. Същевременно, подобрява сътрудничеството между клиентите, доставчиците и диспечерите чрез предоставянето на единен комуникационен формат (Василев & Cristescu, 2020). Позволява на бизнеса и крайните клиентите да имат видимост и контрол върху пратките, което им помага проактивно да управляват всички проблеми.

Интеграция с ERP дава възможност за споделяне на модели на данни и работните потоци. Технически модули на ERP, като SAP Netweaver Gateway (Drees, et al., 2014) предоставят на персонализираната система както входящи, така и изходящи интерфейси. Например поръчките за транспортиране от външно приложение се получават чрез входящ интерфейс, а данните за пратката се връщат чрез изходящ интерфейс след планиране на доставката. Чрез централизираната система различните организационни единици следва да работят в съответствие с регионалните и международни стандарти и изисквания.

В съответствие с модела на Dickens (2019), персонализираното решение прилага рамка, основаваща се на две предварително определени фази: планиране и изпълнение. В началната фаза системата улеснява регистрацията и обработката на поръчките за продажба. Определя най-ефективните източници на изпълнение и подготвя маршрути за доставка. Във втория етап се назначават доставки, които същевременно могат да се проследят в реално време. Значението на тази функционалност е свързано със ISO стандартите и споразуменията за ниво на обслужване, като например ISO 9001, който е свързан с управление на качеството, ISO 28000, специфичен за управлението на сигурността . Софтуерът предлага транспортни данни, включващи визуални представяния, насочени известия относно предстоящи събития, както и преглед на жизнения цикъл на пратките. Същевременно, системата има за цел да подобри жизнения цикъл чрез внимателен избор на превозвачи и оптимизиране на маршрутите за доставка. Системата подготвя определени маршрути въз основа на дестинацията, следвайки методи от компании като FedEx (Frey, 2023).

При регистриране на нови поръчки, промяна на съществуващи или назначение на доставка през платформата, актуализациите се публикуват във вътрешните системи, което води до постоянни актуализации. Системата е насочена към глобална аудитория с милиони потребители и хиляди служители, като следва да се справя с пикове на търсенето, мащабирайки се според нуждите. Чрез IoT сензорите, системата обновява местоположенията на доставчиците на всяка секунда, което генерира огромен брой ежедневни съобщения. В тази връзка е необходима висока изчислителна производителност на обработка на данни, която да минимизира прекъсванията на услугите. Времето за отговор трябва да бъде бързо, в рамките на няколко милисекунди, тъй като системата свързва клиентите с доставчици и диспечери. В края на процеса, на базата на извършени доставки, се издават фактури, които се изпращат на клиентите, като този етап може да бъде следваща доработка на системата.

## 1.3. Възможности за централизация на процесите по управление чрез прилагане на облачни технологии

През последните години редица автори и изследователски компании (Partida, 2023; Roy, 2023; Microsoft Research, 2023) подчертават значението на облачните технологии за оптималното функциониране на веригите от доставки. Облачните изчисления, както са определени от (Huang, et al., 2014) представляват изграждането на корпоративни системи, използвайки практики за разработка на високо-качествен софтуер и инфраструктура. Фактори като проектиране, интегриране и внедряване на система пряко влияят на процеса по управление на поръчките от клиенти. Базираните в облак решения са проектирани като технологични иновации, с възможности за обслужване голям брой потребители и устойчивост на натоварване или хакерски атаки (Илиева, 2020).

Концепцията за облачни изчисления варира, като например организацията Cloud Native Computing Foundation (2018) предлага следната дефиниция: "*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."* От друга страна National Institute of Standards and Technology (2011) определя облачните изчисления като „*модел за позволяване на мрежов достъп, при поискване, до споделен пул от конфигурируеми изчислителни ресурси, които могат бързо да бъдат предоставени и внедрени с минимални усилия.“* Посочените определения дават различни тълкувания, като преобладаващото е схващането, че базираните на облак системи са свързани предимно с висока производителност, ниско ниво на латентност (Smith, 2022), които следва да бъдат разгледани в детайли.

Облачните системи се характеризират с висока производителност, а производителността измерва времето между заявката на потребителя и последващия отговор на системата. Следователно производителността действа като показател за ефективност, който е свързан с удовлетвореността на потребителя. Бързото време за реакция обикновено означава оптимална производителност на системата, което води до положително потребителско изживяване, докато забавянето може да е показател за неефективност. Heusser представя общ метод за концептуализиране на производителността, чрез следното уравнение:

*Време за отговор = Време за обработка + Време на изчакване*

,където, *време за отговор* е общото време, от момента, в който потребителят изпрати заявка до момента, в който получи отговор. Или интервал от време, което потребителят чака да види резултат след започване на действие. *Време за обработка* е време, необходимо на системата за изчисляване на резултата след получаване на заявката. То включва задачи като заявка към база от данни, обработка и всякакви други действия, които системата извършва, за да изпълни заявката. От друга страна, *време на изчакване* представлява времето, в което заявката се намира в „опашка“, преди да бъде обработена. В система с голям трафик от данни могат да постъпят няколко заявки едновременно. Ако системата не може да ги обработи наведнъж, някои заявки трябва да изчакат, като по този начин се увеличава времето за изчакване, така че чрез разделянето на времето за отговор на неговите компоненти, системните администратори и разработчиците могат да определят областите за подобрение. Например, ако времето за обработка е дълго, може да е необходима оптимизация на алгоритми или код. Ако времето за изчакване е дълго, това може да служи като показател, че системата се нуждае от по-добро балансиране на натоварването или увеличен капацитет за обработка.

В тази връзка, много автори фаворизират нивото на латентност, тъй като то дава информация дали част от клиентите получават последователно обслужване. В софтуерните системи 95% (означено в техническата литература като P95, Kleppmann, 2017) от заявките се обработват в сравнително оптимално време, докато 5% отнемат повече. В литература, латентността се свързва с ефикасността на заявките в проценти (като P95, P99 и P99.9), като се акцентира върху „слабите“ заявки. Голямото забавяне може да означава проблеми, които възникват само при определени условия, като конкуренция за ресурси, хардуерни проблеми или други.

Според проучване на Google, (Winters, 2020), 53% от потребителите пренебрегват сайтове, зареждането на които отнема повече от 3 секунди. Имайки в предвид, че мрежовият трафик и натоварването са динамични променливи, влияещи пряко върху производителността на системата и възможността за разширяване на бизнеса, в съвременната ситуация мащабируемостта е възможно решение на тези проблеми. Проучвания на източници в областта (Betts, D. et al. 2012), показват че способността на система да управлява ефективно увеличеното работно натоварване се отнася до мащабируемостта. В литературата се описват две измерения на мащабируемостта: вертикална и хоризонтална. Според (Henning et al. 2022), вертикална мащабируемост представлява надграждане на физическия хардуер като процесор, памет или пропускателна способност на мрежата. За сметка на това, хоризонтална мащабируемост се постига чрез добавяне на ресурсни единици. Вместо да се подобрява един сървър, множество виртуални сървъри се създават, за да се разпредели натоварването. Този подход осигурява висока достъпност и толерантност към грешки, но същевременно въвежда сложност при координацията между ресурсите.

Висока наличност е първостепенен атрибут на качество, предоставен от облачните доставчици, а наличността на система се определя като частта от времето, през което дадена услуга е функционална и достъпна. Според (Atchison 2020), наличността може да бъде изразена като процент от времето на работа (uptime) спрямо сумата от времето на работа и времето в престой (downtime):

*Availability = uptime / (uptime + downtime)*

Счита се, че за повечето облачни услуги абсолютната 100% наличност е нереалистична поради необходимостта от поддръжка и надстройки (Davis, 2019). Статистически, 90% наличност се равнява на над 2 часа ежедневен престой или 36 дни годишно. 95% се равнява на около час дневно или 18 дни годишно, в които системата е офлайн докато според рекламите на доставчиците, времената за готовност за работа са около 99% (99,9% наричани още „три деветки“), при престой по-малко от 1,5 минути дневно.

В тази връзка, споразуменията за ниво на обслужване(SLA), което представляват договорни споразумения между облачни доставчици и компании. Според (Debski et al. 2018) SLA включват гореспоменатите ангажименти за производителност, латентност и време за реакция. От друга страна, индивидуалните цели, определени за една система се наричат цел за ниво на обслужване (SLO). Всеки SLO показва целева стойност или диапазон за специфични системни аспекти, като време за реакция под 100 ms на 90-ия процент. В тази връзка, индикаторът за ниво на обслужване (SLI) е количествена мярка за определяне на спазването на SLO. Той представлява данните за ефективността в реално време, които се събират и оценяват дали се постигат SLO. Според нас SLA, SLO и SLI са основни за осигуряване на качество на облачна услуга. Докато SLA често се определят от юридически екипи, SLO и SLI попадат в обсега на софтуерните архитекти.

Изследователи в областта (Laszewski et al. 2019) анализират редица фактори и разработват методология, наречена „дванадесет фактора“ (Twelve-Factor), представена в Таблица 1.3.

Тази методология предоставя набор от принципи и практики, към които разработчиците да се придържат, когато създават приложения, оптимизирани за съвременни облачни среди. Практици (Grafiati et al. 2023) смятат Twelve-Factor за солидна основа за изграждане на облачни системи, защото е приложима за всяко уеб, десктоп или мобилно базирано решение. Системите, изградени по тези принципи, могат да се внедряват и мащабират, като същевременно позволяват добавяне на нови или промяна на съществуващи функционалности.

Таблица 1.3.  
 Обобщение на методологията на дванадесетте фактора  
(адаптирано от автора)

| **Фактор** | **Описание** |
| --- | --- |
| **Code Base** | Единична база за сорс кода на всяка микроуслуга, съхранявана в собствено хранилище към GitHub, GitLab, Azure DevOps и други. Чрез контрол на версиите, всяка микроуслуга може да се внедри в множество среди (QA, Staging, Production). |
| **Dependencies** | Всяка микроуслуга изолира и пакетира свои собствени зависимости, като обхваща промени, които да не засягат цялата система. |
| **Configurations** | Конфигурационната информация се управлява чрез инструмент, извън кода на микроуслугата. Тя може да бъде различна за различните страни. |
| **Backing Services** | Допълнителните ресурси (хранилища за данни, кешове, брокери на съобщения) трябва да бъдат изложени чрез адресируем URL адрес. Това отделя ресурса от приложението, което му позволява да бъде взаимозаменяем. |
| **Build, Release, Run** | Всяка нова версия следва да премине през няколко етапа на изграждане и изпълнение, чрез използване на технологии за автоматизация. Резултатът от това е минимизирането на възможностите за допускане на човешки грешки и стандартизиране на цялостния процес. |
| **Processes** | Всяка облачна услуга трябва да се изпълнява в свой собствен процес, изолиран от другите. |
| **Port Binding** | Всяка услуга трябва да бъде самостоятелна със своите интерфейси и насочена на определен порт. |
| **Concurrency** | Когато капацитетът на услуга трябва да се увеличи, мащабирането следва да бъде от хоризонтален тип, ориентирано към увеличение на множеството идентични процеси |
| **Disposability** | Екземплярите на услугите трябва да благоприятстват бързото стартиране, както и изключване. Контейнерите заедно с приложение-оркестратор, по своята същност, отговарят на това изискване. |
| **Dev/Prod Parity** | Различните среди е необходимо да се поддържат възможно най-сходни, през целия жизнен цикъл на приложението. Тук контейнеризацията може значително да допринесе чрез насърчаването на същата среда за изпълнение. |
| **Logging** | Регистрационните файлове, генерирани от различните услуги, следва да се третират като потоци от информация. Инструменти за управление на логове (като Azure Monitor или Splunk) се препоръчват за публикуване на посочените данни и тяхното архивиране. |
| **Admin Processes** | Изпълняване на административни задачи, като почистване на вътрешни данни или рестартиране на услуга. |

(Hoffman 2016) описва подробно всеки от оригиналните 12 фактора, като добавя три допълнителни, които отразяват модерен дизайн на облачни приложения (Таблица 1.4).

Таблица 1.4.  
 Допълнение на методологията на дванадесетте фактора  
(адаптирано от автора по Hoffman, 2016)

| **Фактор** | **Описание** |
| --- | --- |
| **API First** | Всеки ресурс трябва да бъде разгледан като приложно-програмен интерфейс, който да бъде интегриран към основната система. |
| **Telemetry** | Дизайнът на системата трябва да включва събирането на специфични за домейна данни, както и за текущото състояние на системата. |
| **Authentication/ Authorization** | Удостоверяването служи като механизъм за проверка на самоличността на потребител, обикновено чрез идентификационни данни като потребителски имена и пароли. Упълномощаването, от друга страна, определя степента на достъп или привилегии, предоставени на удостоверен обект. |

Като допълнение, Microsoft Research (2023) предоставя набор от ръководни принципи, които се използват за подобряване качеството на работното натоварване което е показано в Таблица 1.5 и представя пет стълба на т.н. „добра архитектурата“.

Таблица 1.5.  
Добри практики на облачната индустрия  
(адаптирано от автора по Microsoft Well-Architected Framework, 2023)

| **Фактор** | **Описание** |
| --- | --- |
| **Управление на разходите** | Обхваща процеса на планиране, оценка, бюджетиране и контрол на разходите, целящ завършване на проект в рамките на одобрен бюджет. Ефективните стратегии за управление на разходите позволяват на организациите да оптимизират използването на облачни ресурси като намаляват ненужните разходи. |
| **Оперативно съвършенство** | Автоматизиране на работната среда и операциите, за да се увеличи общата производителност и да се намалят човешките грешки. |
| **Ефективност** | Отговаряне на изискванията, поставени върху работни натоварвания, чрез тестове за производителност и натоварване, за да се идентифицират потенциалните затруднения. |
| **Надеждност** | Отнася се до концепцията за висока производителност, разгледана в тази глава, както и до функционалностите на мобилните и уеб приложения да „предвиждат“ и справят с неочаквани проблеми. |
| **Сигурност** | Тъй като облачните архитектури по своята същност разпределят ресурси между множество местоположения, проблеми пред сигурността, вариращи от пробиви на данни до неоторизиран достъп, могат да възникнат. Протоколите за криптиране и управление на самоличността подобряват сигурността в облака, но естеството на киберзаплахите налага непрекъсната бдителност. Алгоритми за откриване на аномалии подобряват способността за превантивно идентифициране и смекчаване на потенциални пробиви. |

Изхождайки от казаното до тук, може да обобщим, че облачните изчисления осигурят инфраструктура за приложения, предлагайки ресурси като сървъри, операционни системи, защитни стени, балансьори на натоварването и много други. Хардуерът е разположени в център за данни, поддържан от облачен доставчик като Microsoft, Amazon, Google или други. ИТ специалистите създават виртуални ресурси, без необходимост от закупуване или поддръжка на устройства. Представено на Фигура 1.3, традиционния подход, при който хардуера е собствен и изисква цялостно управление и поддържка от ИТ отдела. Противоположно на това, инфраструктурата като услуга (IaaS) абстрахира физическия хардуер в среда, управлявана външно, позволявайки на организациите да изнесат слоевете на мрежова връзка, съхранение и виртуализация, като същевременно запазват контрол върху операционните системи и приложенията. Като допълнение, платформа като услуга (PaaS) предоставя и управление на операционните системи, междинния софтуер и средите за изпълнение, като по този начин позволява на разработчиците да се съсредоточат единствено върху създаването и внедряването на приложения. Софтуерът като услуга (SaaS), най-абстрактният модел, доставя напълно функционални приложения, изцяло управлявани от доставчика на услуги. Този модел позволява разработчиците да се концентрират върху бизнес полезността на софтуера. Всеки от тези модели очертава различно ниво на контрол, сложност на управление и оперативни разходи, което ги прави подходящи за различни организационни изисквания и възможности.



Фиг. 1. 3. Сравнение моделите на облачни изчислителни услуги (IaaS, PaaS, SaaS) и традиционната локална инфраструктура, определяйки отговорностите за управление. Източник: Mohammed и Zeebaree, 2021

Според изследвания (Kesan et al. 2013), IaaS, PaaS и SaaS моделите са по-ефективни от базовият, тъй като се заплаща само за услугите, които се използват. Ценообразуване, базирано на потреблението, позволява на компаниите да плащат за ресурси само когато са необходими, което намалява първоначалните разходи и позволява по-добро прогнозиране. Освен това има възможност за увеличаване или намаляване на ресурсите в съответствие с работното натоварване в рамките на определен период от време. Друга възможност са т.нар. „изчисления без сървър“ описани от (Kumar & Agnihotri, 2021), които описват отделни функции, които се изпълняват, когато се стартира определено действие. Въпреки че не са подходящи за всяко приложение, това е бърз вариант за внедряване, когато логиката е разделена на независими единици или за автоматизирани задачи. За тях се заплаща на базата на отделните заявки към функцията. Също така облачните доставчици осигуряват архивиране на данни, възстановяване „след бедствие“ и услуги за репликация (Куюмджиев, 2019).

Проучвания на източници в областта (Li, S. et al. 2021) показват, че за конструиране на облачни системи се препоръчва ориентирания към микроуслуги архитектурен стил (microservices). Микроуслугите са интерфейси, които са предназначени за комуникация между приложения, за разлика от уеб сайтовете, които са насочени към взаимодействието с хората и се достъпват през браузър (Smith, 2024). Микроуслугите са подход за изграждане на сървърни приложения като набор от малки, но високо-качествени под-услуги. Съответно, 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, gRPC и други. Всяка микро услуга притежава специфична фунцкия и предимства като това да бъде проектирана, разработена и внедрена независимо от другите. Работата може да бъде разпределена между отделни екипи, осигурявайки възможност за независима работа по отделни области на приложението. Микроуслугите са свързани с **фактор #6** от принципите на дванадесет-факторното приложение, който свързва притежанието на всяка услуга със своя собствена логика и данни, в рамките на автономен жизнен цикъл. Концептуалните модели, технологиите и проблемите се различават между подсистемите или микроуслугите. Този принцип е заложен в дизайнът, управляван от домейн, където всяка услуга притежава свой модел на домейн (данни + логика и поведение).

За разлика, монолитните приложения представляват традиционен модел на софтуерна архитектура, при който всички компоненти на приложението са тясно интегрирани и разгърнати като едно цяло. Тази архитектура, преобладаваща в разработката на софтуер от много години, обхваща унифициран модел, при който различни функции, като въвеждане на данни, обработка и потребителски интерфейс, са интегрирани в една програма. Монолитните програми показват висока степен на вътрешно свързване и взаимозависимост между компонентите, което води до нарастваща сложност с течение на времето. Тази сложност се проявява като сложна, неструктурирана и трудна за поддръжка кодова база. В този смисъл, подобренията или модификациите на една част от системата могат неволно да засегнат други несвързани секции. Следователно, отстраняването на грешки се усложнява, което от своя страна възпрепятства въвеждането на нови функционалности. Това поведение, е описано в различни казуси за разработка на софтуер (Elgheriani & Ahme, 2022; Smith, 2024), където се подчертава недостатъкът на монолитните архитектури в сравнение с подхода на микроуслугите. Целта на микроуслуги е да се достави функционален продукт, изискващ постоянна поддръжка и тясна връзка с клиента.

В исторически план, service-oriented architecture (SOA) често се разглежда като предшественик на архитектурата на микроуслугите. SOA възниква в началото на 2000-та година, поставяйки основата за модулен софтуерен дизайн. Традиционните реализации на SOA използват сложни механизми като Enterprise Service Buses (ESB) и протоколи като SOAP и WS-\* което е усложнено за поддържане. (Radev & Aleksandrova, 2013) От друга страна, микроуслуги използват протоколи, които интернет вече предоставя (обикновено REST API), което ускорява разработката и улеснява поддръжката на приложения. При архитектурата ориентирана към микроуслуги всеки екип е отговорен по отношение на платформата за разработка, базата данни и създаването на регистрационни файлове.

Достъпът до данни е по-труден при дизайн на микроуслуги в сравнение с монолитния. Данните, притежавани от микроуслуга, са поверителни и могат да бъдат извлечени или синхронно чрез API, или асинхронно чрез съобщение. Капсулирането на данните определя, че микроуслугите са слабо свързани и могат да се развиват независимо една от друга. Ако множество услуги получат достъп до едни и същи бази от данни, актуализациите на схемите ще изискват координация, което би нарушило автономността на жизнения цикъл.

## 1.4. Управление на бизнес процесите чрез ориентиран към домейн дизайн

В изследване (De La Torre et al., 2023) представят обстойно изследване на ключовите характеристики на уеб услугите, независимо дали става въпрос за монолитна система или част от разпределена инфраструктура, които включват обема на обработваната информация, бързодействието, бизнес логика и технологично развитие. В други изследвания свързани с „ориентиран към домейн дизайн“ (DDD) се определя ефективността в ситуации, при които има нужда от обработка на сложни бизнес изисквания. С други думи, DDD цели да се справи със сложността на домейна, която обхваща бизнес правила, валидации и изчисления. Изхождайки от определенията и предложенията в Глава1. управление на поръчките от клиенти в производствено предприятие се отличава със сравнително висока сложност и необходимост от изпълнение на поредица от процеси (Parusheva & Pencheva, 2021).

Подходът „*дизайн, управляван от данни*“ (data-driven design), описан от (Т. Ерл 2007) в книгата „*Принципи на дизайна на ориентирана към услуги архитектура“*, определя разделянето на услугите въз основа на данните, с които функционалностите им оперират. От друга страна, (Е. Еванс 2014) твърди, че DDD предоставя ключови концепции за разделяне на уеб услугите, на базата на други фактори след като формулира методология, за DDD и предоставя начин за представяне на реалния свят чрез структурирано решение, което отговаря на изискванията в проблемното пространство. Тези характеристики допринасят за подобрение на качеството на софтуерната архитектура.

В този смисъл, сложността на бизнес логиката представлява индикатор за сложността на проблемната област, която софтуерът е предназначен да реши. Ако приложение, извършва основни операции като създаване, четене, актуализиране и изтриване (CRUD), може да обобщим че то не би съдържало сложна логика и може да бъде реализирано с по-прости методи от DDD (De La Torre, 2023). От друга страна, система за управление на поръчки, която автоматизира голяма част от операциите на компанията и моделира голяма част от логистичните процеси, следователно управлява множество сложни бизнес задачи. Сложността, отнасяща се до броя на алгоритмите и технологиите, които трябва да бъдат внедрени, за да се осигури правилната функционалност на софтуера, може да бъде много висока.

В книгата си „*Patterns of Enterprise Application Architecture*“ (2012), авторът Фаулър представя диаграма (Фигура 1.4), която илюстрира връзката между времето, цената и сложността при проектирането на софтуер. В тази диаграма по оста Y са представени времето и цената, докато по оста X е измерена сложността на проекта. Този модел е свързан с ориентирания към данни подход за проектиране на софтуер и показва, че при този метод, след достигане на определено ниво на сложност, дори незначително увеличение на сложността може да доведе до значително увеличение на разходите и времето, необходимо за разработката.



*Фигура 1.4. Домейн-центрирано срещу данни-центрично в контекста на диаграма за разработка на софтуер, изобразяваща време и сложност.* Източник: *Fowler, M. (2012)*

Срещу това, при ориентирания към домейна подход времето и разходите за проекта имат тенденция да нарастват линейно, като началните разходи обаче биват по-високи. Според принципите на DDD (Zimarev, 2019), случаите на употреба следва да се моделират въз основа на начина, по който реалният бизнес функционира, като взема предвид, че този той постоянно се развива.

DDD предоставя различни технически концепции и модели, които могат да бъдат използвани за внедряването на софтуерни проекти. Представени като контекстна карта на Фигура 1.5, идентифицирането и управлението на взаимозависимостите и сътрудничеството помежду им, се улеснява. Картата има за цел да даде структура на облачната система, както и да бъде пътеводител в по-голямата картина.



Фиг. 1.5. Карта, описаваща връзките в DDD. Източник: Evans, 2014

Тези концепции включват универсален език (UL), ограничен контекст (BC), агрегати, обекти на основния домейн, стойностни обекти и хранилища на данни и много други. Въпреки че, тези технически аспекти са много на брой и според критици са трудни за научаване, те са част от правилното прилагане на DDD методологията.

В различните индустрии се използва специфична терминология, която отразява определен бизнес контекст (Oukes, 2021). В този смисъл, когато се разработва сложна система за управление, е важно да се разбере и използва терминологията, както и да се осигури нейно съответствие в програмния код, за се реализират бизнес целите. Основна характеристика на DDD е улесняване на комуникацията между експертите по домейна и софтуерните инженери, като се дефинира общ, **универсален език** (UL). Това е инструмент, който помага на обединяването на бизнесът, дизайнерите и програмистите, така че те да могат да създадат модели на домейна и да ги приложат в практиката. Когато кодът е написан на UL, той може да даде подсказки за случаи и изисквания, които не са били достатъчно ясни предварително. За да функционира успешно, класовете в кода и таблиците в базата данни трябва да се именуват в съответствие с термините от UL. Тази обща номенклатура улеснява разбирането и съгласуването на изискванията между всички заинтересовани страни. В своето изследване Batista (2022), подчертава важността на универсалния език за предотвратяване на недоразумения и неправилни предположения. UL се използва в различни аспекти на разработката на софтуер, включително в документацията, комуникацията между екипите, кода на приложението и кода за тестване. Според Rademacher и колектив (2017) универсален език се развива и се поддържа с течение на времето, като предоставя средство за събиране и организиране на знанията и бизнес логиката.

В този смисъл, ограниченият контекст (BC), който е друга част от контекстната карта, се счита за малка област в домейна, която дава на всеки елемент от UL собствено значение (Wlaschin, 2018). Според практици в областта, често кодовата база на приложение става неуправляема, когато обемът се увеличи. BC контролира как са структурирани подпрограмите и тяхното развитие. Често BC съответства на под-домейн, който показва как е разделена дейността на бизнеса. Всеки BC се разработва самостоятелно, като може да бъде една или няколко микроуслуги.

В книгата си Vaughn Vernon (2013) казва, че т.н. стойностни обекти следва да са малки, прости обекти, които не се основават на идентичност. Те са елементи, използвани за количествено определяне, измерване или характеризиране на определена тема. Стойностните обекти могат да имат методи и поведение, но никога не трябва да имат странични ефекти. Примери за такива обекти могат да бъдат представените по-горе организационни структури.

От друга страна, сред публикациите в DDD областта, агрегатът е представен като колекция от свързани елементи, които се модифицират като едно цяло (Hippchen, Giessler, Steinegger, & Abeck, 2017). В този смисъл, агрегатите се третират като единица за промени в данните. Те се състоят от един или повече обекти, които се променят заедно. Преди да се направят модификации, е необходимо да се оцени консистенцията на целия агрегат, като може да има правила, които да гарантират, че всички данни на обектите са последователни (Wlaschin, 2018). Промените в данните на агрегатите следва да бъдат атомарни, последователни, изолирани и дълготрайни (ACID).

В своя публикация авторският колектив (Петров, Куюмджиев и Димитров, 2022) представят хранилищата от данни като колекция от елементи от определен тип. Тази колекция предлага унифицирана абстракция за всички проблеми, свързани с четенето и записването на данни, улеснявайки ИТ специалистите. Хранилищата предоставят и регулират достъп до базата с данни, чрез публичният си интерфейс. Счита се, че това е възможност, която прави кода на приложенията по-лесен за тестване в сравнение с традиционния вариант, при който външните ресурси се свързват пряко. В този смисъл, тъй като кодът за достъп до данни е „обгърнат“ в един или няколко програмни класове, той следва да бъде лесен за използване. В тази връзка, Vernon (2016) определя събития в домейна, като средство за записване и част от UL. Събитията служат като индикатори, че определено събитие се е случило. Те могат да бъдат разгледани като съобщение, но същевременно и запис в исторически дневник. Хранилищата от данни следва да управляват събитията в домейна, като „агрегират“ информацията.

Според практици в областта на софтуерното инженерство (De La Torre et al., 2023) Hexagonal, Clean, and Onion архитектурите поддържат високи нива на модулност и разделяне на проблемите. Hexagonal (или шестоъгълна) архитектура, представена на Фигура 1.6 поставя модел, при който ядрото на приложението е отделено от външни системи чрез дефинирани портове и адаптери, като по този начин улеснява взаимозаменяемостта и тестването.



Фиг. 1.6. Модел на Hexagonal архитектурa. Източник: Vieira (2023)

Clean (или чиста) архитектура, представена от Robert C. Martin (2017) и илюстрирана на Фигура 1.7, се базира на Hexagonal, като я допълва чрез разделяне на системата на слоеве с правила за зависимости, което въвежда йерархия, с идеята че вътрешните слоеве следва да останат незасегнати от промените във външните, но за сметка на това, външните са пряко зависими от промените във вътрешните.



Фиг. 1.7. Модел на clean архитектурa. Източник: Lano & Tehrani, 2023

Комбинацията от разгледаните две архитектури с DDD принципите, описани го горе, централизира модела на домейна, обкръжавайки го с приложни и инфраструктурни слоеве. Представена на Фигура 1.8 onion архитектура използва тези слоеве и централно ядро. Горните слоеве зависят от долните, но не и обратно, показвайки, че основните елементи на DDD трябва да работят независимо един от друг.



Фиг. 1.8. Модел на onion архитектурa. Източник: Palermo, 2018

В този смисъл, важен аспект от проектирането и създаването на услуга е определянето на нейните граници. Както беше споменато по-горе, всеки BC идентифицира субектите и стойностните обекти, характеризира ги и ги комбинира. Изборът къде да се направи границата между BC изисква балансиране на две конкуриращи се цели. Създаването на бариера около елементите, които се нуждаят от сплотеност, е първата. Втората цел е да се избегнат „бъбриви“ комуникации между звената. Тези цели могат да противоречат една на друга. Балансът трябва да се постигне чрез разлагане на системата на възможно най-малките единици. Друг начин този аспект да се разгледа е автономността. Една работна единица не е напълно автономна, ако разчита на друга, за да изпълни заявка (Khononov, 2021).

На базата на направени проучвания (Braun, S., Bieniusa, A., & Elberzhager, F., 2021), голямата част от корпоративните приложения имат различни нива, които помагат на разработчиците да управляват сложността на кода. Спазвайки принципите на DDD, елементите могат да бъдат организирани в няколко слоя, както е показано на Фигура 1.9.

Diagram

Description automatically generated

Фигура 1.9. Трислоен архитектурен модел. Източник: Vettor & Smith (2023)

Счита се, че приложният слой координира потока на изпълнение между различни обекти. Той определя случаите на използване и операциите, които могат да бъдат извършени в рамките на услуга като организира взаимодействието между потребителския интерфейс и основните елементи. Обикновено приложният слой се реализира като уеб API или MVC проект (Сълов, 2022). Приложният слой е зависим от другите два. От друга страна, слоят на домейна капсулира бизнес логиката и основните обекти, които съставляват ядрото на услугата (агрегати, стойностни обекти и др). Той се концентрира върху решаването на бизнес проблеми и изразява концепциите и поведението на бизнес домейна. От гледна точка на кода, този слой следва да има напълно отделени класови обекти, за да не зависи от никой друг. За сметка на това, инфраструктурният слой е отговорен за осигуряването на необходимите технически инструменти. Основната му функция е да абстрахира и капсулира технически подробности. Той предоставя реализации за множество проблеми, включително запис на данни, съобщения, мрежова комуникация, интеграция с външни услуги, кеширане и оптимизиране други.

В този смисъл, Грег Йънг представя концепцията за разделяне на отговорността за команди и заявки (CQRS) през 2010 година като разширение на принципите на DDD. Тази идея се базира на принципа на Bertrand Meyer, наречен "разделяне на команди и заявки" (CQS). Съгласно този принцип, всеки метод в API трябва да бъде или команда (command) или заявка (query), но не и двете едновременно. Според Йънг, командите са методи, които извършват операции, променящи състоянието на системата. Те са отговорни за изпълнение на действия, които променят данни или файлове. Заявките, от друга страна, се използват за извличане на информация, като предоставят данни, но без да променят стойностите.

Един от аспектите на CQS е, че методите би трябвало да връщат стойност само ако са "референтно прозрачни" и нямат "странични ефекти, което прави кода по-четлив и предсказуем (Indrasiri & Suhothayan, 2021) . Въпреки това, не винаги е възможно или практично да се съобразяваме стриктно с принципа на CQS. Има сценарии, когато методите трябва да имат както страничен ефект (промяна на състоянието), така и да връщат стойност. Например, при работа със структура от данни "Стек", методът "Pop" премахва и връща последния елемент от стека (Наков, 2022). В този случай разделението на тези задачи на два отделни метода може да стане нелогично. Следователно, важно да се анализират конкретните изисквания и сценарии на приложението, преди да се прилага стриктната парадигма на CQS. В тази връзката, вместо да се фокусира върху методи като CQS, CQRS прилага същите принципи, като се насочва към разделяне на операции: една за управление на записите (командите), а другата за обработка на четенето (заявките). Считаме, че чрез това разделение може да се разработят различни стратегии, които да се фокусират върху конкретните нужди на облачно базираната система. Приложният слой преобразува входните заявки и команда и ги изпраща по споделен комуникационен канал, известен като „манипулатор на съобщения“. В този контекст, командите се използват за да кажат на приложението да извърши определено действие, заявките се използват за да поискат информация или данни от приложението, а събитията представляват информационни съобщения. Командите предизвикват реакции в модела на домейна, а събитията са резултат от тях. Именуването на съобщенията следва стандартизирани указания на UL, като командите винаги са в повелително време, заявките обикновено започват с "GET," а събитията винаги са в минало време.

Според Brewer (2012), теоремата на CAP (или теоремата на Брюър) е основен принцип в областта на разпределените системи, която има тясна връзка с CQRS. Съгласно CAP, че разпределена система не може да гарантира едновременно всички три от следните възможности:

1.Последователност (Consistency): Всички операции на четене връщат запис или грешка;

2.Достъпност (Availability): Всяка заявка получава отговор, дори ако не всички части от системата са достъпни;

3.Производителност (Partition tolerance): Системата продължава да работи дори при загуба или забавяне на комуникацията между различни части в мрежата;

Считаме, че чрез внедряването на CQRS, разработчиците могат да създават облачни услуги, които се справят ефективно с големи натоварвания от заявки, като същевременно гарантират съгласуваност на данните чрез стриктната обработка на командите. CQRS обикновено се използва като междинен етап преди източника на събитие. Извличането на събития допълва CQRS, като събира всички промени в състоянието на системата като поредица от събития, които могат да бъдат използвани за съгласуване и анализ на данните.

Извличането на събития (ES) е техника за проектиране, базирана на концепция, че всички промени в състоянието на приложението, през целия му жизнен цикъл, се записват като поредица от събития. В резултат на това, събитията (events) се превръщат в основен градивен елемент на приложението. При подхода за източник на събития програмите съхраняват транзакции, но не и съответните им състояния. Когато е необходимо да се изтегли текущото състояние от базата, следва да се приложат всички транзакции от началото. Нищо не се изтрива или променя в хранилището за данни. Поради това, не може да се срещне проблеми с едновременната актуализация. На базата на направени проучвания се установи, че повечето приложения работят като съхраняват текущото състояние. Вместо да съхранява цялата информация в колоните на един запис или в свойствата на един обект, чрез ES състоянието на обектите се описва от последователността от събития. Това е т.н. *представяне на субект, базирано на събития*. За да се получи текущото състояние на определенa същност, необходимо е да се повтори времевата линия на програмата от самото начало. Този алгоритъмът включва изследване на данните и използване на логика за извличане на съответната информация. Чрез използването на записаните събития, е възможно да се реконструира състоянието на определен агрегат. Това понякога може да изисква управление на огромни обеми от данни. В такъв случай могат да бъдат записани проекции, които представят състоянието в определен момент от време. Веднъж съхранени, събитията са неизменни. Изхождайки от теоретичните постановки, свързани с концепцията, считаме че, съхранението на събития може да бъде релационно, базирано на документи или базирано на графи, следователно събитията могат да се съхраняват в SQL (Kuyumdzhiev & Nacheva, 2020) или NoSQL база данни като PostgreSQL, MySQL, MongoDB, Apache Cassandra, или могат да се съхраняват с помощта на по-специфични решение като „RavenDB“ или „FaunaDB“.

В този смисъл, разработка, управлявана от тестове (TDD) и DDD са две мощни методологии, които, когато се комбинират, могат да повишат качеството на облачните услуги и процеса на разработка. Използвайки тези практики, разработчиците и инженерите за осигуряване на качеството могат да създадат система, която е по-здрава и надеждна. TDD насърчава строг процес на тестване, при който тестовете се пишат преди кода за внедряване; този процес следва добрите практики, като гарантира, че предвидената функционалност е изпълнена. Съществува процедура в три стъпки, известна като „red, green, refactoring“. Създаването на неуспешен тест за част от функционалността е първата стъпка. Втората фаза е „зелената стъпка“, по време на която се създава достатъчен производствен код, за да премине неуспешният тест. “Рефакторингът” е последната фаза, в която тестовият и производственият код се подобряват, за да се поддържа високо качество. Този цикъл се повтаря за всяка част от функционалността в реда на нарастване на сложността във всеки метод и клас, докато не бъде завършена цялата функция. Използването на TDD гарантира, че процесът на тестване е това, което ръководи дизайна.

В областта на софтуерното тестване съществуват няколко различни вида. Някои тестове са базирани на предмета – например тестване на единица (unit testing), интеграция, услуга на компонент и потребителски интерфейс. Междувременно други се определят от целта на теста – например функционални тестове, тестове за приемане и проучвателни тестове. Други пък се определят от това как се тестват – например автоматизирани, полу-автоматизирани и ръчни тестове.

Пирамидата за автоматизация на тестовете (Фигура 1.10) е представена за първи път от Майк Кон в книгата му Succeeding with Agile: Software Development Using Scrum. Пирамидата изобразява видовете тестове, които трябва да се извършват на различни етапи от жизнения цикъл на разработка на софтуер и колко често следва да се адаптират в тестов пакет, за да се гарантира качеството на програма. Идеята зад пирамидата е, че тестерите трябва да посветят повече усилия на основните тестове, преди да преминат към по-сложни.

Diagram

Description automatically generated

Фиг. 1.10. Пирамидата на тестове. Източник: Succeeding With Agile: Software Development Using Scrum, by Mike Cohn (2010)

На фигурата са идентифицирани четири различни вида тестове:

1) Юнит тестове - автоматизирани тестове, които проверяват колко добре работи отделна част от кода сама по себе си;

2) Сервизни тестове - автоматизирани тестове, които проверяват колко добре работи група от класове и методи, които предоставят услуга на потребителите;

3) UI тестове - автоматизирани тестове, които проверяват дали цялото приложение работи (от потребителския интерфейс до базата данни);

4) Ръчни тестове - тестове, извършвани от лице, което също така проверява пълната функционалност на приложението;

Счита се, че чрез внедряването на TDD програмистите имат способността да идентифицират потенциални проблеми на ранен. В допълнение, итеративният характер на TDD позволява честа обратна връзка, което улеснява непрекъснатото усъвършенстване и адаптивност при разработването на облачни услуги.

Важно да се отбележи, е че техниките, изложени в този раздел, не са подходящи за всички ситуации и следователно имат някои ограничения, като необходимостта от време и усилия за правилното разбиране и прилагане на многобройните DDD слоеве, модели и концепции. Според литературните източници, кривата на обучение за DDD е „стръмна“. В този смисъл, дизайнът на всеки елемент в системата показва свои собствени компромиси и вътрешни решения.

Като заключение, подходите за проектиране, управлявани от домейн, се очертават като методология за изграждане на архитектури на облачни услуги. Капсулирането на основния бизнес домейн в добре дефинирани, ограничени контексти, помага за правилното създаване на подсистеми, модули и обекти. Чрез комбиниране на споменатите подходи производствените организации могат да изградят системи, които са не само технически стабилни, но и съобразени с бизнес целите и изисквания. В крайна сметка, възприемането на управлявания от домейн дизайн и облачни архитектури следва да помогне на организациите с внедряването на иновации, намаленото на разходи, предоставянето на услуги на своите клиенти и конкурентоспособността в бързо променящия се дигитален пейзаж. В тази връзка, неспособността да се поддържа адекватно разделяне на проблемите в корпоративния софтуер е основна причина за претоварени кодови бази, което от своя страна води до забавяния и дори провал на проекти. Тъй като тази част се фокусира най-вече върху теоретичните основи, като продължение във втора и трета глава предстои да се представи практически ориентиран казус върху фактическата реализация.

##### Изводи и обобщения към първа глава

1. От изследване на темата за същността на процесите и софтуерните системи за управлението на веригите за поръчки и доставки, можем да заключим, че проблемите непрозрачност, намалената ефективност и ограничения на мащабируемостта, са значими и изискват технологична намеса;
2. Въз основа на потенциала на облачните технологии да революционизират на традиционните системи, представляват алтернатива за справяне със съществуващите проблеми;
3. Главата подчертава, че разработването на ефективна облачна система изисква приемането и адаптирането на концепции от области на софтуерна архитектура и управление на бизнес;
4. Въпреки че базираните на облак системи предлагат множество предимства, те не са лишени от проблеми, като например сигурността.. Въпреки че съществуват протоколи за централизиране на процесите за удостоверяване на потребителите, технологиите непрекъснато се развиват, изисквайки актуализации и мониторинг.
5. Главата поставя началото на изследване на облачна архитектура за управление на поръчки в производствени предприятия. Идентифицираните тук проблеми следва да послужат като основни точки, около които да се насочи архитектурният дизайн и стратегията за изпълнение.
6. Първа глава представя теоретични основи в областта на стопанската и информационна логистика, управлението на веригите от поръчки и доставки, системите за планиране на ресурси, както и проблемите и решенията по дигитализация на процесите, чрез прилагане на облачни технологии. Тази глава полага основи за следващите фази по проектиране, изграждане и внедряване на облачна информационна система.

# Глава 2. Архитектура на облачна система за управление на поръчки от клиенти

## 2.1. Концептуален модел на облачната система за управление на поръчките

Концептуален модел следва да бъде разгледан като решение от високо ниво, което се съсредоточава върху всички основни потребителски, бизнес и ИТ изисквания (Стоев, 2018). Важна част от тази глава са градивните елементи и интерфейси, изграждащи системата, както и комуникационните модели, които да ръководят композицията. Освен това, дизайнът обхваща функционалност, използваемост, устойчивост, производителност, икономически, технологични ограничения, компромиси и естетически проблеми на клиентските и сървърни приложения. В този смисъл, софтуерният продукт, разглеждан в настоящия труд, се състои от 2 клиентски приложения, които се свързват към разпределена бекенд система, базиранa на микроуслуги, работеща върху множество процеси и сървъри (хостове). Всяка услуга се изпълнява в отделен процес като контейнер, разположен в клъстер от виртуални машини. Това разделение на подсистеми, отделни нива и компоненти цели да постигне разбираемост и лесна поддръжка, стремейки сложността на операциите да бъде сведена до минимум.

В свое изследване Ingeno (2018) разглежда възможността за използване на итеративен процес за проектиране на концептуален модел на бизнес система. Този итеративен процес, изобразен на фиг. 2.1, се състои от циклична поредица от етапи за разработване и усъвършенстване на софтуерни архитектури. Диаграмата представя структурирана рамка, като в основата лежи „Концептуалният модел“, който включва прогнозиран на растеж на системата свързан с основните бизнес сценарии. „Системния преглед от високо ниво“ дава възможност за преход към „логическите модели“ включващи диаграми на същност-връзка (ERD), потребителски последователности и дейности, установявайки цялостно разбиране на системата от гледна точка на логически данни. Втората част на втората половина на диаграмата артикулира аспекта на взаимодействието с потребителя, изобразявайки „Функционалност и интерфейси“, които следва да дадът детайлно представяне за очаваните резултати и възможности на системата. 

Фиг. 2.1. Итерации процес за проектиране на концептуален модел (разработка на автора)

Първата фаза в изграждането на архитектурата на облачна система за управление на поръчки включва **прогнозиран растеж на системата**. Счита се, че софтуерният артефакт заема централна роля в концепцията, затова измерението на капацитета на услугите и необходимата инфраструктура са основен етап от процеса. По отношение на „модела на зрялост“ на архитектурата, Stuckenberg (2014) категоризира етапите на прогнозиране на няколко нива. В съответствие с основните оперативни и стратегически цели, фиг. 2.2. представя адаптиран от автора модел, който определя 4 възможни нива на архитектурата на облачната система. В първото ниво, наречено „при поискване“, всяка организационна единица използва специален екземпляр и сървърна инфраструктура, които могат да бъдат персонализирани според специфичните изисквания. Въпреки че, във второто ниво на „обединена логика“ се стандартизира използвания екземпляр, всяка организационна единица използва специален хардуер с идентични копия на софтуера. В тези два случая, персонализирането е ограничено до предварително дефинирани функционалности. Третото и четвъртото ниво представляват конфигурируеми архитектури, като на тези нива всеки клиент използва едно и също копие на един и същ хардуер. Четвъртото ниво е допълнително подобрено от балансьор на натоварването и клъстер от бази от данни. В този смисъл, последните две нива позволяват постоянни обновления и реализиране на нови версии на системната.

Фиг. 2.2. Модел на зрялост на архитектурата (разработка на автора)

Считаме че, включването на четвъртото ниво на конфигурируемост и мащабируемост в архитектурата е в съответствие с целта за подобряване на ефикасността на механизмите за обработка на поръчките, оптимизиране използването на ресурси и поддържането на ниски оперативните разходи. Множество клиенти следва да използват една и съща версия на система в обединена инфраструктура, което позволява оперативните разходи да бъдат разпределени между отделните организационни единици. В условията на нестабилни пазарни изисквания, оперативната гъвкавост се очертава като важен атрибут за производствените организации. Базираната на облак система, предлага подходи за адаптиране към променливи като повишен потребителски трафик, обеми на поръчки, разнообразни продуктови асортименти и променящи се предпочитания на клиентите.

За да се адаптира облачно базирана система за управление на поръчки в оперативната структура на производствена компания, следва да се представи анализ, който отчита различните бизнес изисквания и идентифицира потенциални случаи на употреба. В тази връзка, втората фаза в разработването на концептуалния модел е дефиниране на бизнес сценарии, които да установят фундаменталното функциониране, като същевременно оценява възможни уязвимости.

Бизнес сценариите описват основните възможности на системата, включвайки „У*правление на потребителски акаунти*“, „*Управление на поръчки от клиенти*“ и „*Управление на доставки до клиенти*“. Също така обхващат участниците, като включените от нас са „Диспечер“, „Клиент“ и „Доставчикът“. Следва да представим диаграма на главен бизнес сценарии на унифицирания език за моделиране (UML). Базирана на нуждите на крайните потребители, диаграма на главен бизнес сценарий се използва за визуално представяне на комуникацията между обектите и събитията, които променят вътрешните състояния на системата. Причислява се към поведенчески диаграми на UML и споделя прилика с "контекстните диаграми на C4", тъй като и се изобразява изглед на структурата на компонент(и) на информационната система, като по този начин се улесни разбирането сред различните заинтересовани страни.

Въз основа на декомпозиция на процеса по управление на поръчките от крайни клиенти, фиг. 2.3. представя диаграма на главен бизнес сценарий: 

Фиг. 2.3. Диаграма на главен бизнес сценарий (разработка на автора)

Дефинираните функции, акцентират върху ролята на диспечера, която контролира всички области на информационната система. Първата от тях е администрирането на потребителите, която включва процедура в няколко стъпки, започваща със създаването на нови акаунти, промени във вече съществуващи, както и промяна или добавяне на права на акаунт. Едновременно с това, диспечерите управляват клиентски поръчки и доставки чрез автоматизиран процес. Възможността на крайните потребители да създават и/или променят поръчки в реално врене, изисква анализ и нужда от постоянна комуникация. За допълнително оптимизиране и надграждане на предоставяните услуги, системата прави постоянен мониторинг и актуализиране на данни, които да допринесат за коректността на информацията, свързана с доставките. Оперативната структура се състои от взаимосвързани фази с двупосочни комуникационни пътища между диспечер, клиент и доставчик. Обединението на тези канали предоставя координация в реално време, осигурявайки последователност в работните процеси и бърза реакция при инцидент или проблем.

Според (Банков, 2023), ясно дефинираните изисквания са основата на успешен проект, тъй като включват набор от процеси като анализ, спецификация и валидиране. В този смисъл, третата фаза от изграждането на концептуален модел е свързано с **формулирането на функционални и нефункционални изисквания**. Функционалните изисквания очертават специфичното поведение и операции на системата, като автоматизиране на обработката на поръчки, интегриране с управление на инвентара за актуализации на наличността в реално време и улесняване на взаимодействията при обслужване на клиенти. Нефункционалните изисквания, от друга страна, определят оперативните атрибути и ограничения на системата, обхващащи показатели за производителност, стандарти за сигурност, мащабируемост, надеждност и достъпност на потребителите.

Функционалните изисквания очертават характеристиките на продукта и са проектирани да посрещнат динамичните изисквания от служителите в производствени предприятия и техните клиенти. Основна част от тези изисквания е способността на системата да интегрира различни бизнес функции, включвайки механизми за проследяване на доставки, междуведомствена координация и автоматизиране на административни задачи, като всички те допринасят за рационализиран.

Изхождайки от анализа на литературата по управление на веригите от поръчки и доставки и данните, получени от разговори с експерти в областта, считаме че основните изисквания включват регистриране и вписване на потребител, преглед на текущите поръчки, разглеждане на детайлите за определена поръчка и доставките към нея. Същевременно, системата следва да поддържа събиране и актуализация на данни в реално време от няколко вътрешни и външни системи като ERP, IoT и др. Като допълнение, са функции като филтриране на елементи от потребителския интерфейс, създаване на нова или промяна на вече съществуваща поръчка, управление на отчети и документи.

Както бе споменато, нефункционалните изисквания уточняват ефективността на системата в различни ситуации, измервайки оперативния капацитет, за разлика от функционалните изисквания, които определят какво прави системата. Считаме че, нефункционални изисквания включват надеждност, която гарантира последователна работа на системата във времето, мащабируемост, която позволява на системата да се справи с растеж на потребителската база, сигурност, която предпазва чувствителните данни от неоторизиран достъп, както и производителност при обработка на голям обем транзакции без забавяне. Тези характеристики са от основно значение за базираните в облак системи, които също така да позволяват на производствените предприятия да постигнат прозрачност във всеки етап от операциите по управление на поръчките от клиенти. В допълнение, мащабируемостта и производителността на системата пряко влияят върху способността й да се адаптира към променящото се търсене и гарантира бърза обработка на поръчките. Нефункционалните изисквания често се наричат ​​„атрибути за качество“ на системата. В този смисъл, следващите точки посочват някои от тях:

* Системата трябва да е високо-достъпна и да може автоматично да разширява мащаба, за да отговори на увеличаващия се трафик (също така да намалява мащаба, след като трафикът спадне);
* Трябва да осигурява диагностични дневници, които помагат при отстраняване на неизправности или други проблеми, които могат да възникнат по време на работа;
* Трябва да поддържа процесите на непрекъсната интеграция и внедряване (continuous integration / deployment);
* Трябва да поддържа междуплатформен хостинг и развитие;
* Системата трябва да връща отговор в рамките на секунди;

Ролята и спецификите на различните изисквания, бизнес сценарий и архитектурни модели полагат основата на цялостно решение. Като заключение на концепцията, в етапа „системен преглед от високо ниво“ фокусът е върху прототип, който адресира идентифицираните проблеми и е в съответствие с установените цели. В тази връзка, Фигура 2.4 илюстрира приложенията, които изграждат системата за управление на поръчките от клиенти. 

Фиг. 2.4. Диаграма от високо ниво на главните приложения. (разработка на автора)

Считаме, че най-подходящи за взаимодействие с крайните потребители са мобилните приложения (Todoranova & Penchev, 2020). Те поддържат функции като местоположение, камера и работят с уеб API. Клиентите на фирмата, които се явяват крайните потребители, управляват и проследяват поръчките и доставките в реално време с мобилно приложение. Целта му е да помага с планирането и логистиката на работната площадка, да въздейства върху крайния резултат с информация и данни. Информацията на смартфона следва „винаги“ да е актуална, тъй като текущото състояние на поръчка и местоположение на доставките се проследява на живо. Други възможности са преглед на история, създаване на нова, промяна или отказване на съществуваща поръчка. Приложението следва да се разпространи чрез Google Play Store и Apple App Store.

От друга страна, уеб порталът е софтуер, насочен към диспечерите и част от цялостната система за управление на транспорта (TMS). Чрез него могат да се създават поръчки и доставки, които същевременно се следят и приспособяват към текущите ресурси. Уеб порталът служи като инструмент за вземане на решения, с предварително зададени предложения, които могат да бъдат одобрени, или отхвърлени и променени, според гледната точка на диспечера на смяна. Вземайки под внимание текущите събития, подсистемите зад уеб портала насрочват за доставка според изискванията от клиента. Те разчитат на правилна информация, която е потвърдена дигитално.

Обхват на уеб портала включва балансиране на работното натоварване на превозните средства, позволява проследяване и коригиране, както на поръчките, така и на доставките, осигурява предварително зададени решения. Диспечерите имат възможност да поправят грешни данни, като комуникират с клиентите или доставчици, или обратно. Същевременно всички промени се отразяват в базата от данни. Уеб портала прилага усъвършенствани техники като оптимизация, която се извършват постоянно на фонов режим. В случай, че поръчки закъснеят, клиентите следва да бъдат уведомени предварително.

Уеб портала предоставя ежедневни отчети, като тези за пробега на превозните средства, отхвърлени поръчки, извършени доставки, въз основа на данните от базата. След приключване на смяна, данните се експортират и сравняват, чрез специални идентификатори към ЕРП. Ако се открие несъответствие, съобщение, съдържащо идентификаторите, се регистрират дневник.

## 2.2. Логически модел на облачна система за управление на поръчки

Преходът от концептуален към логически модел е подход, целящ цялостно разбиране на проблемите и рационалното им решение. Докато концептуалният модел има нетехническите характеристики, логическият модел предоставя по-изчерпателно описание, включително обекти, свойства, връзки и процеси. Тази подход улеснява установяването на колективно разбиране и поддържане на техниките за ориентиран към домейн дизайн и универсален език между всички участващи страни. От техническа гледна точка, архитектурата на облачно базирана система се състои от няколко модула, всеки от които е проектиран да обработва специфични аспекти на процеса на управление на поръчките. Тези модули работят съвместно, за да осигурят ефикасно и ефективно обработване на клиентските поръчки.

### 2.2.1. Модули, поддържащи поръчки и доставки

Въз основа на текущи анализи, установихме, че подпрограмите за управление на поръчки и доставки са взаимосвързани. Следователно е необходимо тези модули да се разгледат заедно, като следва да отговарят на принципите и методите на архитектурния дизайн, описани в началната глава. В тази връзка, UML компонентната диаграма осигурява подробно представяне на двата модула, илюстрирайки взаимодействието им, както с вътрешните системи, така и с външните приложения. Компонентните диаграми са подмножество на структурните UML диаграми, които предават концепциите в системата. Компонентите, показани в тези диаграми, имат прилика със съществителните, открити в естествения език, а връзките, които ги свързват, са или структурни, или семантични по природа. Диаграмите на компонентите са по същество диаграми на микроуслуги, които биват използвани за моделиране на изгледа на статичната реализация и документация на системата. В тази връзка, фиг 2.5. представя UML диаграма на компонентите, която прави преглед на под-системите за управление на поръчки и доставки в обхвата на технологии като интернет на нещата (IoT), мобилни и уеб приложения (UI), както и вътрешни системи (ERP,SCM). 

Фиг. 2.5. UML Диаграма на компонентите представяща структурата и връзките на модулите. (разработка на автора)

Диаграмата очертава модулната структура на системата, идентифицирайки разделянето на отговорностите между различни компоненти, съчетавайки няколко комуникационни протокола и услуги за данни. Графиката илюстрира архитектурния дизайн DDD, прилагайки концепцията за „ограничен контекст“, дефинирайки две „зони“ на отговорност между поръчки и доставки. Както се вижда, дизайнът на системата се характеризира с наличието на три типа микроуслуги, а именно услуги за приемане на съобщения, команди и извличане на информация. По пример от първа глава, услугите за команди променят състоянието на системата, като запазват транзакциите и последователността на събитията. Също така, те приемат промени по асинхронен начин от услугите за съобщения. От друга страна, услугите за извличане на информация oтправят запитвания към базата от данни, без да променят състоянието. Основен компонент е механизмът за синхронизация, който поддържа на съгласуваност на данните в отделните хранилища за запис и четене.

В този смисъл, диаграма дефинира „Вътрешните системи“ като основен компонент, който се свързва с услуга за приемане на съобщения през SOAP протокол, което показва важността за взаимодействия в реално време. Освен това, способността на системата да се свързва с „IoT устройства“ чрез директна TCP връзка демонстрира възможност за адаптиране на различен вид сензори, които набират популярност в съвременните операции на веригите за доставки.

Също така графиката ясно показва четири отделни бази данни: една за записване на информация за поръчките, друга за съхраняване на информация от доставката и две допълнителни за извличане на записите. Всяка база данни е проектирана да отговаря на специфичните оперативни нужди на съответния компонент. Това разграничение подчертава използването на моделa на разделяне на отговорността при запис и четене, както и позволява обработката на голям набор от данни, без да има прекъсване или забавяне поради блокиране на ресурси.

Както бе отбелязано в първа глава, базите от данни следва да поддържат съхранение на събития. Целта да съхраняват събития е да регистрират промени в състоянието на системата. Също така, базите поддържат целостта и проследимостта на транзакциите, служейки като хранилища, в които само се добавят нови данни. Веднъж записани, събитията не могат да бъдат променени, като по този начин се поддържа точността и хронологичният ред. Освен това, реконструкция на състоянието на системата е възможна от всеки даден момент във времето, придобивайки разбирания за поведението на потребителите и позволявайки детайлен одит на всяко действие.

Считаме че, схемата на базите от данни обхваща две таблици: потоци и събития. Потоците служат като основа за организиране и категоризиране на събития. Таблицата по-долу предоставя описание на модела и структурата на таблицата за „Поток“.

Таблица 2.1.  
 Описание на структурата на таблицата за „Поток“   
(разработка на автора)

|  |  |
| --- | --- |
| Поле | Описание |
| Идентификатор (id) | Универсален уникален идентификатор, който представлява първичния ключ за всеки поток. |
| Вид (type) | Указва типа на потока, който може да бъде категоризиран или класифициран. |
| Версия (version) | Обозначава номера на версията на потока. |
| Времеви печат (timestamp) | Улавя точния момент, в който записът е създаден или последно актуализиран. |
| Текущо състояние на потока (snapshot) | Представлява състоянието на потока в определената версия, което позволява бързо извличане на данни. |

От друга страна, събитията са основни компоненти при записването на промени в състоянието и действията, извършвани в системата. Техните характеристики включват историческа неизменност и възможност за проверка. Събитията са динамични обекти, които поддържат последователност, отчетност и адаптивност. Те са част от анализа на данните, позволявайки вземането на информирани решения. Таблицата 2.2. дефинира структура на модела на „Събития“.

Таблица 2.2.  
 Описание на структурата на таблицата за „Събитие“   
(разработка на автора)

|  |  |
| --- | --- |
| Поле | Описание |
| Идентификатор (id) | Универсален уникален идентификатор, който представлява първичния ключ за всяко събитие. |
| Идентификатор на потока (stream\_id) | Свързва събитие със съответния поток, установявайки връзка с таблицата с потоци. |
| Идентификатор на последователнноста (seq\_id) | Последователен идентификатор, представляващ реда, в който се случват събитията. |
| Вид (type) | Указва типа на събитието. |
| Времеви печат (timestamp) | Записва точния момент на създаване на събитието. |
| Данни (data) | Записва данните на събитието. |

Прилагането на концепцията за събиране на събития във всички бази от данни на система за управление на поръчки предлага консистентен метод за съхранение и извличане. Събирането на събития се основава на принципа, че всяка модификация на състоянието на приложението се записва като поредица, улавяйки не само текущото състояние, но и цялата история и всички промени. Този подход предоставя на системата способността да възстановява предишни състояния, да подпомага одити и запитвания и да създаде повторно възпроизвеждане. Според принципите на DDD, записите в таблица „потоци“ на всяка една от базите служи като „агрегатор“ или обект, свързан с потока от събития, обикновено представляващ поръчка или доставка. Това позволява капсулирането на поредицата от събития. Таблицата за „събития“, от своя страна, съхранява подробната информация. Още повече, съгласуваността на структурата на данните позволява създаването на „материализирани изглед“ за подобряване на достъпа до данни и производителността на заявките от микроуслугите. Комбинацията от източници на събития и материализиран изглед позволява на системата да предоставя оптимизирани за четене представяния на данни, които са често използвани. Изгледите са персонализирани, за да изпълнят специфични нужди, като например извличане на поръчка с всички доставки. В случая, те се съхраняват като отделни таблици и това „събиране“ бива осъществено от облачните услуги. Материализирания изглед премахва необходимостта от извършване на съединения и изчисления по време на изпълнение, което води до подобрение в ефективността и отзивчивостта. Освен това, чрез възприемане на събирането и агрегирането на събития се минимизира присъщата сложност при управлението на множество различни таблици и релации, което често има нужда от миграции при промени. Също така, чрез използването на последователни структури от данни, системата за управление на поръчки получава способността да използва исторически данни за анализи от услуги за машинно обучение.

### 2.2.2. Детайлизиране на модулите за поръчки и доставки

След установената обща системна рамка, следва да се проучат оперативните процедури и адаптирани шаблони в микроуслугите. Както бе споменато в предходната глава, компонент на DDD е подхода за разделяне приложението на слоеве, което да контролира сложността на кода. Считаме, че всяка услуга в системата трябва да поддържа сходна структура на пакетите които използва, което да допринесе за функционална съгласуваност. Всеки от компонентите в контекстите, представени на фиг.2.5., се намира на най-високото ниво в йерархията, предоставяйки API между данните и продуктите. Използването на UML диаграма на пакетите създава това йерархично представяне и разделение на слоеве. Тя е вид структурна диаграма, показваща разположението и организацията на елементите в компонентите и техните зависимости. Фиг 2.6. представя предложената UML диаграма на пакетите: 

Фиг. 2.6. UML Диаграма на пакетите представяща структурата и връзките на елементите. (разработка на автора)

На основно ниво в структурата на сглобяване на всяка услуга, стоят две основни под-директории: 'програмен код' (src) и 'тестове' (tests), които съдържат изходния код и компонентните тестове. Оформена е по следния функционален, управляван от домейн дизайн:

├───**Услуга**

│ ├───**src** – индикира път към основния код

│ │ ├───**API**- логика за представяне

│ │ ├───**Business** – бизнес логиката на приложението

│ │ ├───**Core** – експонирани за клиенти модели

│ │ ├───**Domain** – вътрешни модели на приложението

│ │ └───**Persistence** – логика за достъпа до данните

│ └───tests – индикира път към тестовете на кода

│ └───**Tests** – пълен набор от автоматизирани (интеграционни) тестове

Фиг. 2.7. Структурата на слоевете на всяка услуга. (разработка на автора)

Src действа като централен посредник, координиращ взаимодействията между няколко слоя, представени от различни пакети, следвайки принципите на чистата архитектура:

* API - служи като входна точка за комуникация, обработвайки SOAP, HTTP, AMQP, TCP заявки и отговори;
* Основен (Core) - служи като център за команди, заявки и модели за валидиране. Той капсулира основните операции и логиката на домейна, насърчавайки възможността за повторна употреба и поддръжка;
* Бизнес (Business)- съдържа бизнес логика, организираща основните функции на приложението. Също така поддържа манипулаторите на команди и заявки, заедно с интерфейси към външни системи. В този слой се осъществява обмяната на съобщения и изпълнението на CQRS;
* Домейн (Domain)- служи като хранилище за агрегати, обекти и събития. Този слой съдържа основните класове;
* Съхранение (Persistence)– този слой съдържа класове за интеграцията с базите от данни. Тези класове изпълняват извличането и записването на информацията;
* Тестови проект – този слой бива изолиран от src, разположен в под-директорията ‘tests’, съдържайки набор от класове за интеграционни и компонентни тестове, придържайки се към TDD подхода;

За допълнително организиране на пакетите в съответствие с принципите на CQRS, в слоя за бизнес логиката се интегрира шаблонът „медиатор“. Той функционира като междинен обект за насърчаване на комуникацията между различни класове, като по този начин капсулира техните взаимодействия. При този подход се намаляват директните взаимодействия, като същевременно се поддържат принципите за свързване. Уникален входен клас се създава за всяка команда или заявка за извличане на информация в системата, който се съпоставя със съответното потребителско действие и се свързва с индивидуален „манипулатор на съобщения“. Тези манипулатори на медиаторни съобщения демонстрират разнообразна гама от оперативни възможности като управление на валидирането на входните класове. Интерфейсът на медиатор има т.н. „контравариантен характер“, позволяващ приемането и обработка на базови типове. В допълнение, шаблонът управлява рисковете свързани с дублиране на данни, интегриране на услуги за анализ и обща абстракция.

В тази връзка, някои автори (АБВ, 2018) посочват, че UML диаграми на класовете са инструмент, използван за представяне на обектите, съответните им свойства и взаимовръзки. Този тип диаграми идентифицират речника на системата. Счита се, че са често срещани във разработката на софтуерни модули. От гледна точка на системата за управление на поръчки, същностите, които ще изграждат дотук описания модел, са следните (вж. фиг. 2.6.): 

Фиг. 2.8. UML Диаграма на класовете. (разработка на автора)

Лингвистичната рамка UL, посочена в предходната глава и насочена към улесняване на комуникация между членовете на екипа за разработка, се използва за изграждането на висококачествен софтуерен код. Тази рамка поддържа процеса на изграждане на конкретните медиаторни класове. Успешното изпълнение изисква съвместни усилия между екипи за разработка на софтуер и лица със специализирани познания в областта по управление на поръчките и доставките. В ситуацията се очаква всички заинтересовани страни да притежават цялостно разбиране на изходния код, което им позволява да предлагат или одобряват подобрения, както и да откриват възможни проблеми. Лингвистичната рамка позволява и насърчава плавен преход от “псевдо код” към изпълним код, подходящ за производствена среда. Интегрирането на “псевдо код”, който се счита за неформално описание на принципа на работа или алгоритмите в системата за управление. Чрез използване на “псевдо код” в рамките на UL, екипите следва да създадат предварително представяне на логиката на домейна. Също така, този подход рационализира следващите фази на внедряване и поддръжка на системата.

В областта на функционалното програмирането, т.н „Either“ монад се отличава като усъвършенстван инструмент за изразяване на сложна бизнес логика по начин, който съответства на описаните по-горе изисквания. Oбщата структура на „Either“ монад е установена, както следва:

├─── Either<Резултат, Грешка>

│ ├─── променлива IsSuccessful

│ ├─── функция Match

│ ├─── функция Map

│ └─── функция FlatMap

Фиг. 2.9. Структура на „Either“ монад. (разработка на автора)

Съществена част от структурата е състоянието на булевата променливата „IsSuccessful“ и поведението на функцията „Map”. Когато даден „Either“ монад, като тип C<T>, има стойност, представляваща успешен резултат, функцията “Map” прилага трансформация по следния начин:

(C<T>, (T => T2)) => C<T2>

В случай, че възникне изключение или грешка, то ще бъде върнато като стойност на нов „трансформиран“ обек. Като надграждане на тази операция е функцията FlatMap, която е предназначена да приема функция, която връща „Either“ монад, като следва алгоритъма:

(C<T>, (T => C<T2>)) => C<T2>

Както бе отбелязано, функционалният подход предлага последователна методология за представянето на операциите като верига, което следва да направи прави кода насочен към определена индивидуална цел. Следната фиг. 2.10. представя примерен „псевдокод“ за създаване на нова поръчка:

├───**Заявка за създаване на поръчка**

│ Flat Map ├─── валидация на входните данни

│ Flat Map ├─── проверка на инвентара и възможност за доставка

│ Flat Map ├─── запазване на промените в базата от данни

│ Flat Map ├─── изпращане на данни към услугите за анализ

│ Match ├─── връщане на резултат или грешка

Фиг. 2.10. Примерен код за създаване на поръчка чрез използването на „Either“ монад. (разработка на автора)

Използването на монадата „Either“ в кода улеснява разделянето на бизнес логика на отделни, независими функции, привеждайки се в съответствие с представената по-рано идея за UL. Докато императивното писане на код включва програмисти, работещи със вложени кодови структури, верижният подход предлага едно ниво и ред на изпълнението. Този програмен код следва да изгражда “медиатор манипулаторите”.

Според някои автори (АБВ, 2020) допълнително улесняване на комуникацията между технически и не-технически лица може да бъде достигнато чрез UML диаграма на активността, способна да моделира бизнес процеси и работни потоци, описани от UL. Чрез дефиниране на условията и процесите на домейна, диаграма на активността предлага универсално разбираем изглед, много подобен на блок-схемите.

В тази връзка, фиг. 2.11. представя етапите от процеса на управление на поръчките, установен чрез анализ от интервюта с експерти в областта. 

Фиг. 2.11. Диаграма на активност на поръчка. (разработка на автора)

Диаграмата очертава процес на логистично управление, подчертавайки обработката на поръчка и планирането на доставка. Процесът започва със „създаване на нова поръчка“, последван от „възлагане на доставчик“ или „начало на процеса по доставка“, което показва динамичното разпределяне на ресурси. Впоследствие, потокът следва оперативните стъпки на доставчика, включително „пристигане и начално на натоварване“. Следващата процедура включват фазата по транспортиране, а именно „пътуване до мястото за доставка“. При „пристигане“, системата се подготвя за процедура по „разтоварване“, която вътрешно е разделена на „начало на разтоварването“) и завършва със състояние „разтоварено“. Важно да отбележим е, че диаграмата илюстрира веригата за обратна връзка, представена в първа глава, включвайки „Потвърдено разтоварване“ и „Подписано“ доказателство за доставка, гарантирайки целостта и завършването на процеса. Цифровата комуникация е неизменна част, която проследява всяка промяна в процеса по доставка. в реално време. Това осигурява прозрачност и ефективност, които са от съществено значение за оптимизиране на производителността и надеждността на веригата за доставки.

### 2.2.3. Модул за управление на потребителските профили

Управлението на потребителски профили в рамките облачната информационна система играе роля в поддържането на постоянната връзка до услугите, защитата на данните от неоторизиран достъп и кибер заплахи. В основата на функционалността на този модул е неговата връзка с мерките за сигурност на софтуера, предназначени да предотвратят пробиви и несъответствия. Forbes публикуват подобни случаи на изтичания (Brewster, 2018), манипулиране или загуба (Paganini, 2019), водещи до последици върху оперативната работа, представянето и имиджа на компаниите.

В тази връзка, модулът за управление на потребителски профили действа като първа линия на защита, създавайки рамка за удостоверяване и разрешаване на потребителски достъп, като по този начин пряко влияе върху устойчивостта на системата срещу външни и вътрешни заплахи за сигурността. Този модул обхваща набор от практики за сигурност, включително „мрежова архитектура нулево доверие“ (Zero Trust Network Architecture) и Secure Access Service Edge (SASE). Тези практики налагат проверка на виртуалната самоличност на всеки потребител и/или устройство, които се опитват да получат достъп до данни за поръчки или доставки. По подразбиране се приема, че както вътрешният, така и външният мрежови трафик може да бъде потенциална заплаха, поради това достъпът до ресурсите е контролиран. Също така, мрежови услуги за сигурност като защитна стена и виртуална частна мрежа са част от технически стек, с цел последователно прилагане на мерките за сигурност.

Изследването на протоколите за удостоверяване и оторизация в контекста на управление на потребителски профили и контрол на достъпа, дава значение на OAuth 2.0 и OpenID Connect (OIDC). OAuth 2.0 позволява на приложения да изискват и получават ограничен достъп до уеб услуги, като делегира удостоверяване на потребителя, без идентификационните данни да бъдат компрометирани. Използван е онлайн платформи като Facebook, GitHub, DigitalOcean (АБВ, 2020). Като допълнение, OIDC разширява OAuth със слой за идентичност, въвеждайки идентификационните токени като JSON Web Token (JWT), които капсулират удостоверена потребителска информация.

В исторически план, приложенията индивидуално поемат отговорността за удостоверяване на потребителите, което води до разработването на персонализирани интерфейси за влизане, функции за регистрация и политики за пароли. Проучване на научни публикации, показва, че чрез делегиране на удостоверяването на потребителски акаунти на централен доставчик на идентичност, улеснява прилагането на мерки за сигурност, включително защитено съхранение на пароли, многофакторно удостоверяване, интеграция с външни доставчици, редовни актуализации и други (АБВ, 2020). Счита се, че централизираното управление опростява спазването на регулаторните изисквания, като предоставят единна рамка за спазване на стандарти като GDPR и HIPAA.

Централизираният сървър за самоличност внедрява OAuth и OIDC, въвеждайки стандартизиран механизъм за управление на потребителския достъп, удостоверяване, издаване и разчитане на токени за проверка на самоличността.

Таблица 2.3.  
 Видове разрешения на OAuth 2.0 (разработка на автора)

| **Вид** | **Случай на употреба** | **Описание** | **Съображение за сигурност** |
| --- | --- | --- | --- |
| **Идентификационни данни за парола** | Влизане с потребителско име и парола | Позволява на клиентското приложение директно да поиска и използва идентификационните данни на потребителя, за да получи токен за достъп. | Не сигурен поради директното използване на потребителски данни. |
| **Идентификационни данни на клиента** | Удостоверяване на услуга към услуга | Използва се за комуникация между сървъри, при която определено приложение достъпва ресурси въз основа на собствените си идентификационни данни, а не от името на конкретен потребител. | Сигурен при взаимодействия без участието на потребител, насочен към вътрешна връзка. |
| **Код за оторизация (Authorization Code)** | Приложения, работещи на принципа клиент-сървър. | Този вид включва обмен на код за оторизация, след което пренасочва потребителя за токен за достъп. | Висока сигурност, минимизира риска от „фалшифициране на между-сайтови заявки“ (CSRF) |
| **Ключ за доказателство за обмен (PKCE)** | Публични приложения, работещи на принципа клиент-сървър. | Подобрява предходния код за оторизация, като изисква от клиента да създаде таен верификатор, който се използва през целия процес на обмен на токен. Това намалява риска от прихващане на кода за оторизация. | Висока сигурност, предпазвам от aтаки за прихващане на код за оторизация, Man-in-the-Middle и други. |

От представената таблица, PKCE се отличава като подходяща техника за удостоверяване, работеща с верификатор и код за оторизация за да достъпи крайната точка за получаване на токен за достъп, осигурявайки сигурна потребителска сесия. Процедурните стъпки, които информационната система следва да въведе, за да осъществи удостоверяването, са представени на фиг.2.8.



Фиг. 2.8. Диаграма на оторизация чрез код и ключ за обмен (PKCE) (разработка на автора по )

Клиентското приложение инициира началото, като насочва заявка за удостоверяване към крайната точка за оторизация с параметри като `client\_id`, `redirect\_uri`, `scope` и `response\_type`, по пример на OAuth2. При успешно удостоверяване на входните данни и съгласие на потребителя, сървърът издава код на клиентското приложение чрез пренасочване към браузъра. След това клиентското приложение обменя кода за токен за достъп чрез друга заявка към сървъра. Считаме че, тази комуникация е сигурна, тъй като се осъществява на няколко етапа, предлагайки и допълнително развитие към двуфакторно удостоверяване. При издаване на токен клиентското приложение извлича самоличността на потребителя и създава сесия въз основа на тази информация.

Сървърът, от друга страна, поддържа потребителско удостоверяване, упълномощаване, профилни данни и управлява пароли, роли, токени и др. В уеб-базирано удостоверяване има няколко действия, които трябва да бъдат извършени: изисква от потребителя информация (потребителско име и парола) за да създадете самоличност, която записва в базата данни, вписва текущия клиент в сървърната сесия, използвайки HTTP бисквитки и отписва, като премахне тази информация. Елементите от приложението и зависимости, които ще обслужват тази част са визуализирани на фиг. 2. **DbContext** и **ApplicationUser** представляват комбинация от класове, които оперират с базата от данни.  **AccountController** използва тези свойствa чрез **UsersService**, който капсулирана логиката, по безопасен за използване начин, и също така отговаря за визуализацията на потребителския интерфейс, чрез генериране на HTML.

Diagram

Description automatically generated

Фиг. 2.8. Структура на класовете, отговарящи за удостоверяване (разработка на автора )

Базата данни, която обслужва услугата, е базирана на структурирана схема, включваща няколко таблици, като „Users“ за съхраняване на потребителска информация, „Roles“ за дефиниране на роли, „UserRoles“ за свързване на потребители с роли, „UserLogins“ за информация от външни. Диаграма на базата от данни e представена на фиг. 2.9.



Фиг. 2.9. Диаграма на базата от данни за потребителите. (разработка на автора по )

Важно да се отбележи, че последна стъпка, е тестването, ориентирано към сигурността. Считаме, че това е компонент в разработването и поддръжката на базирани на облак информационни системи (АБВ, @), целящ идентифициране и коригиране на потенциални уязвимости, които биха могли да компрометират целостта на системата. Това тестване е многостранно, включващо статични и динамични анализи, тестове за сигурността, често провеждани от „етични хакери“, симулиращи кибератаки в контролирани условия.

## 2.3. Kомуникационни модели между модулите

Комуникационните модели имат важна роля в облачните приложения, тъй като обработват обмена на информация между подсистемите, използвайки технологии за предаване на данни. Тези технологии позволяват на клиентски и сървърни приложения да използват различни видове комуникация, насочени към постигането на различни цели. В предходните диаграми бе използван SOAP протокол за предаване на структурирана информация във вътрешните системи. Освен това протоколите: HTTP, REST, gRPC и AMQP обслужват нуждите на мобилни и уеб клиенти, като предоставят методи за комуникация, които са в съответствие със съвременните стандарти за архитектура на облачни услуги. Като допълнение, инфраструктурата включва IoT устройства, използващи директна TCP/IP мрежова връзка, позволявайки незабавен трансфер и подобрявайки екосистемата със сензорни данни.

В този смисъл, UML диаграма на последователностите идентифицира как продуктите и услугите на система взаимодействат помежду си, за да реализират основна функционалност. Тя визуализира времевата линия и редът, в който се извършват операциите. Въз основа на концепциите на DDD, тази диаграма илюстрира последователната от събития, които се случват в отговор на определенa заявка на потребителя. Основната цел на диаграмата на последователността е да илюстрира интерактивното сътрудничество между различни компоненти на система. Те също играят роля в подпомагането както на техническите, така и на нетехническите заинтересовани страни, което ги прави част от UL. Като част от поведенческите UML диаграми, тя подчертава динамичното поведение на системата

Като резултат, фиг.2.10. представя диаграма на последователностите на основен бизнес сценарий, който е очакван от клиентите да осъществят.

Фиг. 2.10. UML диаграма на последователностите на основен бизнес сценарий. (разработка на автора)

Диаграмата на последователността изобразява комуникацията между клиент и мобилно приложение, включително услугите, които го обслужват. Това е продължение на предходната процедура, след стартиране на мобилното приложение и успешно удостоверяване. Клиентът следва да разгледа информация за настоящи, предстоящи и минали поръчки, както има възможност да започне нова заявка за поръчка, която да бъде обработена и валидирана чрез агрегирани от базата данни и интегриране с вътрешни системи за получаване на резултат. Системата автоматично променя статусите на поръчките и доставките и след това предава тази информация обратно на клиента. Тези промени са свързани и със сензорите и IoT устройствата. Като надграждане на предходните модели, в облачната инфраструктура се внедрява шлюз на приложните интерфейси (API Gateway). Той предоставя входна точка за група микроуслуги, наподобявайки принципа на дизайн „фасадата“. Известен е също като „backend за frontend“, тъй като се изгражда за конкретни нужди на мобилно или уеб клиентско приложение, като по този начин се оптимизира комуникацията и обработката на данни въз основа на контекста на клиента. Освен това, API Gateway позволява на разработчиците да прилагат мерки за сигурност и връзка с протоколите за удостоверяване, съобразени с изисквания на различните нива, като по този начин укрепва цялостната сигурност на системата. Важно да се отбележи, че API Gateway може да се превърне в “анти-модел“ като монолитно приложение, съдържащо твърде много крайни точки и ненужна бизнес логика. В тази връзка, API Gateway трябва да бъдат разделени на логически групи въз основа на бизнес ограничения, показано на фиг. 2.11. 

Фиг. 2.11.. Комуникация чрез шлюз за приложните програмни интерфейси. Източник: Smith, S., 2024

За да се свърже клиентските приложения към сървърната част, API Gateway изпилзва Representational State Transfer (REST). Представен през 2000г. като част от дисертацията на Рой Т. Филдинг, REST е базирани на хипермедия и е независим от протоколите на приложния слой, като концептуалните идеи зад него са взети от HTTP и WWW (Price, 2022). Основно предимство на REST е, че той използва отворени стандарти и не обвързва внедряването на API Gateway или клиентските приложения към конкретна реализация, а бива структуриран въз основа на ресурси бизнес или DDD обекти.

В тази връзка, URI се структурира по йерархичен начин, като например: https://gateway.com/orders извлича набор от поръчки за текущия клиент. Всеки елемент в колекцията има отделна идентификация, за да може да информацията да бъде извлечена индивидуално, като примерен URL адрес: <https://gateway.com/orders/123> .

HTTP 1.1 протоколът определя набор от методи, показани в таблица 2.4, които имат различни значения:

***Таблица 2.4****: Методи на протокола HTTP 1.1.*

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **GET** | Най-разпространеният метод. Използва се за извличане на репрезентации на ресурси. Информацията се съдържа в отговора. |
| **HEAD** | Подобен на GET, но без обект в отговора. |
| **PUT** | Заменя ресурса в посочения URI. Тялото на заявката описва ресурса, който трябва да бъде актуализиран. |
| **DELETE** | Премахва ресурса в посочения URI. |
| **POST** | Създава нов ресурс. Тялото на заявката предоставя подробности за новия ресурс. |
| **OPTIONS** | Предоставя мета данни за ресурс. |
| **PATCH** | Извършва частична актуализация на ресурс. |

За да се свържат клиентските приложения към сървърната част, API Gateway дефинира редица крайни точки, съответстващи на методите от предходната таблица, описани в Таблица 2.5. Те осигуряват различна семантика, когато се прилагат към различните ресурси, като се спазват стандартите на RESTful реализациия.

***Таблица 2.5.***:Крайни точки и техните *REST конвенции.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ресурс** | **GET** | **POST** | **PUT** | **DELETE** |
| **/orders** | Извлича всички поръчки. | Създава нова поръчка. | Общо актуализиране на поръчките. | Премахва всички поръчки. |
| **/orders/id** | Извлича детайли за поръчка по идентификатор. |  | Актуализира данните за поръчка 1, ако съществува. | Премахва поръчка 1. |
| **/orders/id/ deliveries** | Извлича доставките по идентификатор. | Създава нова доставка за поръчка по идентификатор. | Общо актуализиране на доставките за поръчка по идентификатор. | Премахва всички доставки за поръчка по идентификатор. |
| **/deliveries /id** | Извлича детайли за доставка по идентификатор. |  | Актуализира данните за доставка по идентификатор, ако съществува. | Премахва поръчка по идентификатор. |
| **/users** | Извлича всички потребители. |  |  |  |
| **/user/id** | Извлича потребител по идентификатор | Създава нов потребител | Актуализиране на потребител по идентификатор | Премахване на потребител |
| **/auth/login** |  | Вход в системата (създава нов токен) |  |  |

Според HTTP протокола, форматите се определят чрез използване на типове медии (наричани още MIME). Уеб услугите следва да поддържат JSON (application/json) като формат за обмен и представяне на данни. Например, заявка към посочения по-горе ресурс за детайли на поръчкa, ще върне следния отговор във формат JSON:

{

"orderId":eu.123123.231,

"orderValue":99.90,

"productId":1

}

Сървърът информира клиента за резултата от заявка чрез използване на предварително зададени “кодове на състоянието”, представени в Таблица 2.6.

***Таблица 2.6****. Таблица с диапазоните на HTTP кодовете.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Статус код** | **Тип** | **Описание** | **Пример** |
| **2xx** | Успех | Заявката обработена успешно. | 200 OK |
| **3xx** | Пренасочване | Клиентът трябва да изпрати допълнителни заявка. | 301 Redirect |
| **4xx** | Грешки в клиента | Резултат от грешна заявка, причинена от клиента. | 404 Not Found |
| **5xx** | Грешка в сървъра | Грешка от страна на сървъра. | 503 Service Unavailable |

Както бе отбелязано в предходните UML диаграми, клиентските и сървърните приложения използват различни комуникационни модели, като може да разграничим два основни типа комуникация, които се използват между компонентите: синхронна и асинхронна. В синхрония подход клиентът изпраща заявка към услуга, която я обработва и връща обратно HTTP отговор. Клиентският код може да продължи своята задача само след получаване на отговор. Заявка и отговор имат обща структура, включвайки начален ред, описващ текущия HTTP метод, адрес, статус и протокол. Допълнително заглавни редове (headers) дават възможност на клиента и сървъра да предадат допълнителна информация или метаданни, където се предават кодовете и токените за удостоверяване и оторизация. Основна част е тялото на HTTP, съдържащо данни, свързани със заявката или отговора, като фиг. 2.12. илюстрира пример към API Gateway в контекста на управление на поръчки: 

Фиг. 2.11 Примерна HTTP заявка и отговор. (разработка на автора)

REST се счита за удачен избор за иницииране на дейности на API Gateway поради широкото му приемане и поддръжка от популярни рамки като ASP.NET, Symfony, Spring, Node.js и много други. Като надграждане на REST, Google и CNCF представят проект с отворен код, на име gRPC. Той представлява високоефективна рамка, която управлява извикванията на отдалечени процедури (RPC) и обмена на съобщения между микроуслуги. В контекста на информационната система за управление на поръчки, gRPC клиентско приложение е API Gateway, което установява локална функция в REST услуга за изпълнение на конкретно бизнес действие. Този код, който се намира API Gateway извиква друга функция на микроуслуга от система като тези за команди или извличане на информация. Очаква се разработчиците да използват няколко езика за програмиране, рамки и технологии, докато работят върху базираната в облак система. В този смисъл, gRPC предлага хоризонтален слой, който управлява оперативната съвместимост между компонентите, използвайки HTTP/2 като основен транспортен протокол, като се възползва от неговите усъвършенствани функции (Vettor, 2024):

* Пренос на двоични данни, за различава от HTTP 1.1, който е базиран на текст;
* Позволява изпращане на много заявки едновременно през една връзка;
* Компресира съдържанието на съобщенията, като по този начин намалява преноса в мрежата;

Технологията Protocol Buffers и неговите proto файлове осигуряват комуникацията на gRPC и вътрешната платформена структура. Разработчиците следва да използват дефиниране на междуплатформен интерфейс (IDL), за да установят точен „договор“ за всяка микроуслуга. Договорът се дефинира като текстов .proto файл, който определя методите, входните и изходни модели. След това компилаторът изгражда код както за клиента, така и за сървъра за целевата платформа и работна рамка. За да се представи връзката между API Gateway и микроуслугите за поръчки и доставки, order\_delivery.proto файл бива дефиниран, както следва:

syntax = "proto3"; // версия на синтаксиса

package order\_delivery; // идентификатор на пакета

import "google/protobuf/wrappers.proto";

service Orders {

rpc Get (GetRequest) returns (OrderResponse); // метод за извикване.

}

message GetRequest { // формат на съобщението

google.protobuf.StringValue order\_number = 1;

}

message OrderResponse { // формат на отговора

google.protobuf.StringValue order\_number = 1;

}

service Deliveries {

rpc Get (GetRequest) returns (DeliveriesResponse);

}

Фиг. 2.12. Protobuf файл за интегриране на микроуслугата за поръчки. (разработка на автора)

Всички услуги, осъществяващи синхронна комуникация, имат зависимости една към друга, създавайки тясна връзка между различните компоненти, което теоретично нарушава една от предпоставките за използване на ориентирана към микроуслуги архитектура. В тази връзка, добавяне на нови микроуслуги и актуализирането на крайните им точки би въвело нови зависимости, които водят до промени в сорс кода. В тази връзка следната таблица сравнява *REST и gRPC*, като видове на синхронна комуникация.

***Таблица 2.8****. Таблица, описваща разликите между REST и gRPC.*

|  |  |  |
| --- | --- | --- |
|  | **REST** | **gRPC** |
| **организиран** | Към ресурси | Към локално извикване на процедури |
| **формат** | JSON, XML, Text, HTML | Protobuf |
| **случай на** | Публични API; | Вътрешни API; |
| **употреба** | Насочени към мобилни и уеб клиенти; | Високо-производителна комуникация м/у услуги; |
|  | Управлявани от данни; | Ориентирани към команди и действия |

Съществен момент при изграждането на информационната система е интеграцията с вътрешните и IoT системи посредством услугите за приемане на съобщения. Този тип разделение е с цел комуникацията между подсистемите да бъде сведена до минимум, както и всеки микросървис да бъде автономен и достъпен за потребителя, дори ако другите не работят. Силна зависимост, описана в предходния параграф, може да опишем в случай, когато трябва да се извърши повикване от една микроуслуга към друга (като изпълнение на HTTP заявка за получаване на данни), за да може да се обработи отговор. В този случай, програмният продукт няма да бъде устойчив ако някоя от частите се срине. Освен това, създаването на вериги от заявки/отговори, намалява значително производителността на цялото приложение и застрашава SLA.

Когато се използва асинхронна комуникация, управлявана от събития, ERP, IoT или друг вид система публикува интеграционно съобщение към услугата за приемане, задействащо брокер, който препраща информацията към друга услуга за управление на поръчки или доставки. Пример е промяна на статуса на доставка. Микроуслугите се абонират за тези вид събитията, за да могат да ги получават асинхронно. Когато това се случи, получателите могат да актуализират собствените си бази от данни. Тази система за публикуване и абониране се реализира чрез по-горе споменатия брокер на съобщения.

В тази връзка, високата степен на наличност и толерантност към частични проблеми не могат да бъдат гарантирани на 100% в разпределените системи. От гледна точка на основните микроуслуги, това е от важно значение за съхранението в услугите, тъй като тези компоненти запазват текущите състояния на приложението. Важно изискване за последователността, е нито една услуга да включва хранилища за данни от друга, както и да не разчита на директни заявки. Фиг.2.13. представя заключителна архитектурна диаграма на микроуслугите, брокера за съобщения и API Gateway. Graphical user interface, diagram

Description automatically generated

Фиг. 2.13 Mикросървисната архитектура за разработка на информационна система. (разработка на автора)

## 2.4. Функционалност и потребителски интерфейс

Интегрирането на цифровите технологии в различни аспекти на управление на верига за доставки, се базира на цялостна цифрова екосистема, предназначена да рационализира процесите. Чрез мобилното приложение, клиентите на производственото предприятие създават поръчки и проследяват доставките. Скица на интерфейса на началния екран, след вход, е дадена на фиг. 2.11. Представени са основни елементи, включвайки текущия потребител, инструмент за избор на дата и списък на предстоящи, текущи или завършени поръчки. Като детайли са представени статуса на потвърждение, очакваното време на пристигане, карта с текущото местоположение, както и функция за пряк контакт с доставчик или диспечер. Регистрирането на нови поръчки се осъществява чрез екран в главното меню.



Фиг. 2.11 Скица на основен екран на приложението. (разработка на автора)

Както беше споменато, мобилното приложение допринася за бързо изпълнение на процесите, сравнително лесно за използване и удобно за работа, чрез функционалностите за достъп до геолокация, навигация, съобщения, телефон. В този смисъл, използвано на приложението от доставчика, изглежда различно. Водача може да провери списъка с предстоящи доставки, назначени към него.



Фиг. 2.12. Скица на екран за доставчика. (разработка на автора)

Посоченият екран съдържа подробности за доставката, включващ материал, количество, местоположение за товарене, разтоварване и планирани часове. Тъй като определена доставката може да бъде анулирана или пренасочена към друга поръчка, приложението изпраща запитвания към сървъра за актуализации на всяка секунда. Също така доставчикът може да съобщи за повреда към диспечера, като след това превозното средство бива отписано.

За да подпомогне автоматизирането и рационализирането на документацията, приложението поддържа функционалност за електронно доказателство за доставка. Това е процес, който създава документацията, валидираща получаването на стоката от клиента. Обикновено, това се осъществява чрез подпис на клиента на физически документ, като това бива последния етап от процеса по доставка. Когато продуктът бива доставен, от клиента се иска да потвърди получаването чрез подпис на мобилното устройство, след което електронният документа се препраща към ERP. Следната фигура представя скица на екрана за тази функционалност.



Фиг. 2.13. Скица на екран за доказателство за доставка (ePOD). (разработка на автора)

Графичен интерфейс на уеб портала, представен на фиг.2.13, е предназначен за използване от диспечерите за планиране и разпределяне на логистичните задачи. Той представя информация за поръчките, които трябва да бъдат доставени, като също така дава пълен контрол върху всички превозни средства. Целта към обслужването на клиенти е, да даде представа за организацията през работния ден.



Фиг. 2.13. Главен екран в уеб портала. (разработка на автора)

Изгледът предлага списък, в който всеки ред съдържа информация за основните данни, всички получени поръчки и планирани доставки. Данните се актуализират на всяка секунда и след всяка транзакция. В този смисъл, уеб порталът служи като инструмент, използван за актуализации на състоянията в моменти, когато превозвачите са без дистанционно предаване на данни.

Уеб порталът поддържа функционалност за времето, което отнема на превозното средство за да стигне от точка А до точка Б. По този начин, той служи като инструмент за разстояние и продължителност на пътуването. На фиг. 2.14. е визуализиран изглед, който показва текущите местоположения и очакваните времена на пристигане на различните превозни средства.



Фиг. 2.14. Екран за маршрутизиране. (разработка на автора)

Уеб порталът и мобилното приложение предлагат телематична система, която предоставя обратна връзка на диспечерите. Уеб порталът използва събития за местоположение, за да изчисли приблизителната оставаща продължителност на пътуване въз основа на геокоординатите, изпратени от мобилното приложение.

##### Изводи и обобщения към втора глава

1. Разработен е концептуален модел на информационната система, състоящ се от основни бизнес сценарии, прогнозиран растеж и преглед на системата;
2. Разработен е логически модел на софтуерна система, който включва описание от високо ниво, включително обекти, свойства, връзки и процеси. Диаграми от различен вид представят работните единици.

# Глава 3. Изграждане и използване на облачна система за производствено предприятие "Хейделберг Цимент Девня" АД

## 3.1. Обща характеристика на дейността на компанията

"Хейделберг Цимент Девня" АД е най-големият производител на цимент в България, разположен в град Девня, област Варна, в експлоатация от 4 декември 1958 г. "Хейделберг Цимент Девня" АД е част от Heidelberg Materials, основан през 1874 г., който е основен участник в глобалната индустрия за строителни материали. Основната дейност включва производство и дистрибуция на цимент, инертни материали, готови бетонови смеси и асфалт. Компания е специализирана в производството и доставката на бетонова смес, която се произвежда в централно съоръжение за дозиране. Терминът „готови смеси“ произлиза от факта, че тези смеси се произвеждат според спецификациите на клиента, което води до прецизен, висококачествен продукт, който може да се използва веднага след доставката. За дозиране на бетон, компанията използва големи централни съоръжения. Това дава възможност за прецизно и точно смесване, гарантирайки целостта на продукта. "Девня Цимент" произвежда смеси според изискванията за здравина, обработваемост и издръжливост. Често готовата смес се доставя в камиони с миксери и трябва да се използва веднага след пристигането. В тази връзка, проследяването на точната му местоположение е от съществено значение. Също така, превозните средства имат сензори, които изпращат информация за нивото на водата, градусите и други характеристики на сместа в реално време, за да се гарантира високо качество на продукта, тъй като свойствата му могат да се променят междувременно. Счита се, че централизираното смесване е по-благоприятно за околната среда от смесването на работната площадка, тъй като отпадъчния продукт е по-малко, а същевременно позволява по-голям контрол върху използваните материали. В този смисъл, продуктите на компанията се използват за изграждане на къщи, инфраструктура, търговски и промишлени съоръжения, като по този начин отговарят на нуждите на нарастващото световно население за жилища, мобилност и икономическо развитие.

Въз основа на анализ на данните от проведените от нас интервюта с специалисти в областта, могат да се формулират следните бизнес процеси:

* Приемане на поръчка: Считаме, че това е първият етап, който нашата система обхваща. Към момента, поръчките се получават по имейл или телефон и се обработват от диспечери. Това включва разбиране на нуждите на клиента по отношение на типа бетон, обема и времето за доставка, на базата на които се сключва договор. Облачната платформа следва да внедри тези процеси, чрез функциите за онлайн регистрация, съгласяване с общите правила и одобрение от диспечера. След това потребителите могат да регистрират поръчка, както и да променят или отхвърлят съществуваща;
* График: След като поръчката бъде приета, следва да бъде планирана за производство и доставка. Различните специалисти дават различни фактори, включващи производствения капацитет, управление на автопарка, периоди за доставка. Според нас онлайн порталът бива инструментът, който следва да помогне в управлението и автоматизацията на тези задачи, интегрирайки вътрешните системи с облачните услуги;
* Товарене: В този процес бетонът действително се произвежда или „дозира“. Това включва точно измерване и комбиниране на суровините: цимент, инертни материали, вода и други добавки в съответствие с конкретните изисквания. След това сместа се зарежда в превозно средство т.н. „миксер“. Според експерти, поддържането на „интелигентни“ сензори, които да изпращат данни в реално време, е пример за внедряването на иновативни идеи. Същевременно, използването на сензорите за определяне на емисиите на въглероден диоксид, повишава прозрачността в съответствие с тенденциите в индустрията към устойчиви практики, изпълнявайки регулаторните изисквания за околната среда;
* Доставка: Въз основа на стандартите ISO/IEC 27001:2013, ISO 9001:2015 и ISO 28000:2007 проследяването на доставката в реално време е основна функция. Интегрирана в облачно базирана система, тя дава възможност на шофьорите да избират оптимални маршрути, като по този начин повишава ефективността и осигурява навременна доставка. Едновременно с това улеснява непрекъснатата комуникация с клиентите, като се визуализира текущото местоположение на доставчика. След доставката, системата рационализира процеса, като позволява на клиентите да подпишат цифрово и да получат електронно доказателство за доставка, елиминирайки необходимостта от традиционни методи на хартия. Както беше описано по-горе, управлението става чрез мобилно приложение, което пази цялата информация. На работната площадката бетонът се разтоварва и поставя според изискванията;
* Фактуриране: След доставка, на клиента се издава фактура, в която е посочена стойността с начислените данъци и такси. За да се подобри бъдещото обслужване, е важно получаването на обратна връзка от контрагентите (клиенти и доставчици). Фактуриране и извършване на плащане са функционалности, които могат да бъдат интегрирани като допълнение към системата;

Според нас, ефикасността и ефективността на тези процеси пряко влияят върху качеството на предлаганите продукти и услуги, както и на рентабилността на операциите. Следователно, доброто управление е от важно значение за успеха на "Хейделберг Цимент Девня" АД.

Позиционирайки компанията в челните редици на интегрирането на иновативни технологии, облачната система позволява поемането на ангажимент за измерване на замърсяването на околната среда, мерки за пестене на енергия и усилията за намаляване на въглеродния отпечатък, „Хайделберг Цимент Девня” АД отдава значение на екологична отговорност в индустрията като прилага програми за развитие. Отдадеността на компанията на изследователската дейност движи стремежа й към иновативни строителни материали, като набляга на ориентиран към клиента подход, подчертан от персонализираните услуги и поддръжка на дигитални платформи.

## 3.2. Избор на технологични средства за реализация на системата

Изборът на правилните средства за реализация следва да бъде резултат от анализ и оценка на няколко технологични елемента, включващи езици за програмиране, работни рамки, бази данни и доставчици на облачни услуги. Чрез избора на технически инструменти, "Хейделберг Цимент Девня" АД следва да подобри производителността, надеждността и ефективността на процесите по управление на поръчки. Това от своя страна би трябвало да помогне в непрекъснато променящата се област на управление на веригата за доставки.

Таблица 3.1 представя резултати от сравнителен анализ за различни уеб базирани работни рамки, описвайки тяхната производителност.

Таблица 3.1.  
 Сравнение на сървърни технологии за разработка  
(Източник: Тechempower, 29.09.2023)

| **Сървърна Технология** | **Програмен език** | **Брой на едновременни HTTP отговори за секунда** |
| --- | --- | --- |
| ASP .NET Core | C# / .NET | ~ 300 613 |
| NodeJS | JavaScript / C++ | ~ 125 743 |
| Gin | Go | ~ 102 559 |
| Symphony | PHP | ~ 70 382 |
| Spring | Java | ~ 30 891 |

Данните на Techempower показват, че ASP.NET демонстрира по-висока ефективност и производителност в сравнение с други алтернативни платформи за уеб приложения, като се отбелязва, че ASP.NET Core е поне два пъти по-бърз от Node.js, който е втори в класацията. В допълнение на това, всяка година от Microsoft публикуват план за предстоящ напредък и поддръжка на .NET, осигурявайки периодични подобрения. В този смисъл, езици за програмиране C#, F#, и VB (Sulov, 2014), част от еко системата на .NET и Microsoft, споделят индекси в статистиката от Tiobe и Statista. Пример е линейна графика, представена на Фигура 3.1, която очертава оценките на езика за програмиране C#, илюстрирайки тенденция на възход към края на 2023 г.



***Фиг 3.1.****: Оценки на индекса TIOBE за езика за програмиране C#* Източник: Tiobe <https://www.tiobe.com/tiobe-index/> [20.12.2023]

Освен това, GitHub дава информация за над 5,7 милиона месечно активни разработчици по проекти с „отворен код“. От своя страна, Stack Overflow отбелязва .NET Core като „#1 работна рамка“ за годините от 2019 до 2021.

В този смисъл, Microsoft Azure, доставчик на облачни услуги, предлага обширна поддръжка за .NET приложения чрез интегрираната среда за разработка Visual Studio (IDE). Според практици в областта, интеграцията от високо ниво на Azure и .NET подобрява разработката на софтуер и гарантира оперативна съвместимост в рамките на екосистема, поддържана от технологичната корпорация Microsoft. Фигура 3.2 показва тенденциите за използване на различни доставчици на публичен облак в различни предприятия.



***Фиг 3.2.****: Доставчици на облачни услуги, използвани от предприятия* Източник: Flexera State of the Cloud Report [01.12.2023]

Констатациите, получени от извадка от 750 участници, показват, че над 40% от фирмите използват Azure като основна облачна платформа. Като допълнение, данните, представени от Gather за 2023 г, показват темп на растеж от 47% в облачната инфраструктура и платформени услуги, установявайки позицията на Azure като водеща публична облачна платформа (Фигура 3.3).



***Фиг 3.3.****: Категоризираща на публичните облачните компании.* Източник: Gartner Magic Quadrant for Cloud Platforms [01.12.2023]

Според статистически данни, Azure разполага с мрежа от 64 центъра за данни, което надминава броя на другите облачни доставчици. Според данните от проучването се предвижда значителен ръст на приходите на Azure от около 26%, до края на 2024 г., достигайки 70 милиарда щатски долара. Някои от производствените компании, които са клиенти на Azure включват Samsung, Boeing, BMW и много други.

Въз основа на събраните данни може да заключим, че използването на .NET и Azure е благоприятен избор за внедряването на облачно базирана система за управление на поръчките от клиенти в рамките на производствена компания.

Различните микроуслуги, използвани от информационната система, имат различни изисквания за съхранение на данни. Azure предоставя няколко различни вида хранилища за данни, които могат да помогнат за поддръжка и извличане:

* Azure SQL Database - Облачно базиран SQL Server. Поведението му е същото като това на основното изпълнение на базата, но предлага и много предимства: репликира в реално време данни в други географски региони, маскира данни за определени потребители, предоставя пълен одит на всички действия, които са се случили върху данните. Услугата е използвана от подсистемите за удостоверяване и каталогът за продуктите.
* Azure Cosmos DB е нов вид нерелационна база данни, която работи с механизъм за съхранение и предоставяне на данни, който използва свободен модел, също така включва ниска латентност, репликация на данни в други географски региони в реално време, управление на трафика, автоматично индексиране на данните. Услугата е използвана от маркетинговата част.
* Azure Blob представлява хранилище за съхраняване на големи неструктурирани данни. Това могат да бъдат фактури, изображения, видео, файлове и други. Услугата е използвана от подсистемата за поръчки.
* Допълнение, Azure предоставя услуги за бази данни MySQL, PostgreSQL и MariaDB като универсално достъпни, мащабируеми, силно защитени и напълно управлявани.
* Azure предоставя две хранилища за данни, които са много подходящи за съхранение на големи количества с цел анализ: Data Warehouse & Data Lake.

В тази връзка, в таблица 3.2. са съпоставени различните Azure услуги със структурата и характеристиките на съхранените данни.

Таблица 3.2.  
 Сравнение на услуги за данни със структура и характеристика  
(Източник: Azure)

Scatter chart

Description automatically generated with medium confidence

От друга страна, технологиите за уеб и мобилни приложения търпят постоянно развитие, което води до съществуването на различни инструменти за изграждане, които работят ефективно на множество платформи. Рамки като .NET MAUI, Kotlin Multiplatform, React Native предлагат структура с кодова база на един проект, който да споделя логика между Android, iOS, Harmony платформи. В тази връзка, разработката на хибридни приложения включва използването на React, Angular, Vue или Cordova които позволяват на разработчиците да създават мобилни приложения с помощта на HTML, CSS и JavaScript/TypeScript, предлагащи “уеб изглед“ който да визуализира данните по еднакъв начин за всички платформи. От друга страна, прогресивните уеб приложения (PWA) са уеб приложения, които също използват уеб възможности, манифестни файлове (manifest files) и адаптивен дизайн, за да накарат уеб приложенията да работят офлайн и да се представят добре на мобилни устройства.

Таблица 3.2.  
 Сравнение на мобилни и уеб технологии за разработка  
(разработка на автора)

|  | **Eстествени приложения (native)** | **Хибридни приложения** | **Прогресивни уеб приложения (PWA)** | **Междуплатформенпприложения (cross platform)** |
| --- | --- | --- | --- | --- |
| Програмни езици и инструменти за разработка | **iOS** - Objective-C или Swift чрез X-Code & iOS SDK  **Android** – Java или Kotlin чрез  Android Studio | Oбвивка около HTML, JavaScript, CSS чрез Cordova, Onsen. | Работни рамки като Ionic с Angular или Vue и Blazor със C# | NET MAUI, Kotlin Multiplatform, React Native |
| Достъп до функциите на телефона | Пълен контрол, без ограничения | Слаб контрол, доста ограничения | Слаб контрол, доста ограничения | Ограничен, но силно поддържан |
| Потребителско изживяване(UX) | Отлично | Добро, но ограничено | Добро, но ограничено | Добро |

Като заключение, използването на междуплатформени приложения, поддържани от .NET или React, следва да бъдат избор за внедряването на мобилни и уеб приложения, които да работят на различни клиентски устройства и ефективно да комуникират с облачните услуги.

## 3.3. Физическа реализация на системата

За изграждане, доставка и изпълнение на системи, изградени като ориентирани към микроуслуги, експерти в областта препоръчват използването на контейнеризирани технологии. Контейнеризацията е подход, в сферата на разработката на софтуер, при който кодът на приложение, всички негови зависимости и конфигурации са пакетирани в двоичен файл, наречен изображение. По документация, изображенията се съхраняват в регистър, който работи като хранилище или библиотека. Облачната платформа трансформира изображението в работещ екземпляр на контейнер, който може да се стартира, спира, премества или изтрива.

Създават се контейнери за различните части от приложението: уеб услуга, база данни, кеширане и др. Точно както транспортните контейнери позволяват транспортирането на стоки, независимо от товарите вътре, софтуерните контейнери се възприемат като стандартна единица за внедряване на софтуер, която може да съдържа различен код и зависимости. Контейнеризирането на софтуера дава възможност на разработчиците и ИТ специалистите автоматично да подновяват новите промени в различни среди.

Контейнерите също така изолират приложенията едно от друго, в споделена операционна система. Според документацията, приложенията се изпълняват върху хостът на контейнерите. В тази връзка, контейнерите предлагат предимства на изолация, преносимост, гъвкавост и контрол на целия жизнения цикъл на приложението. Според експерти в областта, най-използваната и наложила се като стандарт технология е Docker. Това е проект с отворен код за автоматизиране на внедряването на приложения като преносими, самодостатъчни контейнери, които работят еднакво както локално така и в облака.

Docker контейнерите могат да работят върху Linux или Windows, като на Фигура 3.4 е представено сравнение между компонентите на традиционна виртуална машина и Docker контейнер. Докер контейнерите включват приложението и всички негови зависимости. Те обаче споделят ядрото на ОС с други контейнери, изпълняващи се като изолирани процеси в пространство на хост операционната система, с изключение на Hyper-V контейнери, където всеки контейнер работи вътре в специална виртуална машина. Graphical user interface

Description automatically generated

Фиг. 3.4. Сравнение между Docker и типична виртуална машина. Източник: Docker, Inc. < https://www.docker.com/>, [09.10.2023]

Според техническата литература, виртуалните машини имат три основни слоя: инфраструктура, хост, операционна система, Hypervisor и всички необходими библиотеки. Слоевете в Docker са инфраструктурата, ОС и двигател за контейнери, който поддържа изолация, но споделя основните услуги на ОС. Тъй като контейнерите изискват много по-малко ресурси (например не се нуждаят от пълна ОС), те са лесни за изпълнение, като същевременно гарантират еднакво поведение на всички среди: локална, среда за разработка или продуктивна.

Дейвис (2019) отбелязва, че контейнерите са инструмент на облачния софтуер, чието управление се извършва със специална софтуерна програма, наречена „оркестратор“. В тази връзка, следващата таблица обобщава задачи към оркестратора, описани в теорията и често срещани в практиката.

Таблица 3.3.  
 Обобщение на практиките за управление на контейнерите   
(разработка на автора)

| **Задачи** | **Описание** |
| --- | --- |
| Планиране | Автоматично предоставяне на екземпляри на контейнери. |
| Мониторинг на активността | Автоматично откриване и коригиране на повреди. |
| Failover | Повторно публикуване на неуспешен екземпляр. |
| Мащабиране | Автоматично добавяне или премахване на екземпляр на контейнер, като отговор на повишен трафик. |

При стартиране на приложения в Azure eдно от първите решения, които трябва бъдат вземети, са планираните за използване услуги:

* Azure App Services - eдин от най-лесните и мощни начини за хостване на приложения. Той е предпочитан при монолитната архитектура. Услугите са достъпни и работят в 99,95% от времето. Споделят мощни функции като автоматично мащабиране, внедряване с нулев застой и лесно удостоверяване, позволяват отстраняването на грешки в приложението докато работи в производствена среда (със Snapshot Debugger). По подразбиране приложението ще бъде достъпно в интернет, без да е необходимо да се настройва име на домейн или да се конфигурира DNS. Работи много добре с контейнери;
* Azure Virtual Machines - позволява преместване на съществуващи приложения от виртуални машини, които вече се изпълняват във център за данни. Има много предварително дефинирани изображения, които могат да бъдат използвани като Windows Server, който работи с IIS и има инсталиран и предварително конфигуриран ASP.NET на него, както и собствени софтуерни лицензи (като за SQL Server). Услугата е подходяща за мигриране на т.нар. „наследена система“, която да бъде използва като подсистема или източник на данни;
* Azure Kubernetes (AKS) - водещ инструмент за управление и мащабиране на контейнери, отговарящ за приложения разпределени между микроусуги. Kubernetes предоставя операции от високо ниво, които да бъдат извършени чрез кода на самите микро-услуги. Pаботи с инструкции, които са прехвърлени върху облачните машини, така нареченият „клъстар“: набор от виртуални машини на Linux или Windows (възлови точки), върху които се разполагат самите приложения (но не директно). Kubernetes се грижи за маршрутизирането и логистика на микросервизните (най-често използван в тази архитектура);

Следната таблица представя различните услуги за хостинг на Azure и техните случаи на употреба.

Таблица 3.4.  
 Обобщение, представящо услуги на Azure и съответстващи случаи на употреба   
(разработка на автора)

Chart, scatter chart

Description automatically generated

Според анализ на услугите от таблицата и допълнителни инструменти за оркестрация, Kubernetes, проект за управление на работни натоварвания с отворен код, се очертава като предпочитан избор, главно поради цялостната си екосистема. За разлика от Docker Swarm и Apache Mesos, Kubernetes позволява автоматизация от най-високо ниво, чрез разпределяне на микроуслуги, планиране на контейнери в клъстер, предоставяне на възможности за автоматично рестартиране, повторно планиране и репликация, което от своя страна улеснява хоризонталната мащабируемост. В този смисъл, Azure Kubernetes Service (AKS) е облачната PaaS услуга, която обгръща .NET Core и Docker приложения, като използва силата на Kubernetes, като същевременно се възползва от удобството и функциите на Azure.

Въз основа на разгледаните до тук технологии и инструменти, на Фигура 3.5 е представена архитектурна диаграма, която да съответства на концептуалните решения от втора глава и същевременно да се използват по-горе описаните облачни услуги. 

Фиг. 3.5. Архитектурна диаграма. (разработка на автора)

Важно да се отбележи, е че виртуализацията, контейнеризацията и наблюдението са основни практики, с които се занимава екипът по "Развитие" и "Операции" (DevOps). Основната характеристика се изразява в подкрепа на процеса на автоматизация и съблюдаване на всяка една стъпка от софтуерната разработка – от интеграция, тестване и пускане на пазара до инфраструктурен мениджмънт. В този смисъл непрекъсната интеграция и доставка (CI/CD) представляват систематичен подход за усъвършенстване на процесите на разработка и внедряване (Фигура 3.6).



Фиг. 3.6. Схема на основен DevOps работен поток. Източник: Microsoft< https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/>, [23.10.2023]

Концептуалният работен поток, представен на Фигура 3.5, започва с редактиране на програмен код, който бива качен в хранилище за контрол на версиите. Това задейства процедурите от етапа „непрекъсната интеграция“ (CI), при които се компилира нововъведения програмен код, последван от тестове за валидиране на вече работещите функционалности. При успешна валидация се генерира артефакт или както беше описано по горе – изображение. Следва втората фаза на т.н. „непрекъснато внедряване“, което включва автоматизирани задачи за публикуване на новата версия, които улесняват разпространението на артефакта в желаната среда. Според нас, този рационализиран процес позволява автономни и постепенни актуализации. Всяка микроуслуга следва да премине през изолиран цикъл на изграждане, тестване и внедряване, което позволява ефективно управление на версиите.

В този смисъл, стратегии за внедряване, често срещани в литературата и практиката за осигуряване на актуализации и поддържане на стабилност в системата, са обобщени в Таблица 3.3.

Таблица 3.4.  
 Обобщение на стратегии за внедряване, описани в теорията и често срещани в практиката  
(разработка на автора)

|  |  |
| --- | --- |
| **Стратегия за внедряване** | **Описание** |
| Синьо-зелено внедряване  (blue-green deployment) | Синьо-зеленото внедряване позволява едновременното изпълнение на две идентични производствени среди. „Синята“ представлява активната, докато „зелената“ обозначава новата версия. Идеята на този подход е да се изпробва нова версия в среда, подобна на производствена, без да се прекъсва активната услуга. |
| Постепенно внедряване  (rolling deployment) | При постепенното внедряване новата версия на приложението се актуализира поетапно, като се публикуват няколко контейнера, а не всички наведнъж. Докато определен брой услуги поддържат стара версия, при възникването на проблем, процесът на внедряване може да бъде спрян и проблемът лесно да бъде локализиран. |
| Внедряване на Canary  (Canary release) | При внедряване на Canary, промяна обхваща малка подгрупа от потребители, преди бъде приложена към цялата инфраструктура. Целта е да се тества малка част от трафика, преди да се разпространи към по-широката потребителска база. |

За да допълним темата за софтуерно внедряване, следва да преминем към моделите за производствено тестване. Подобно на стратегиите за внедряване, тестването гарантира, че софтуерът функционира според очакванията в производствена среда. Тези модели могат да помогнат за предотвратяване на софтуерни дефекти, подобряване на устойчивостта на системата и поддържане на качество и надеждност. Според техническата литература, съществуват няколко модела, като A/B тестването е един от тях. В контекста на базирана в облака система за управление на поръчки, A/B тестването дава възможност за вземане на решения, като позволява едновременното внедряване на различни версии на системни подобрения или нови функции за подгрупи от потребители, като по този начин позволява изготвянето на сравнителни оценки на ефективността. Тази процедура, показана на Фигура 3.7, позволява оптимизирани модификации чрез разчитане на емпирични данни.



Фиг. 3.7. Диаграмата илюстрираща внедряване на облачна услуга, използвайки А/Б тестване.. Източник: Stuckenberg (2014)

Според случая на употреба, представен в диаграмата, клиентите биват сегментирани в две отделни групи: група A и B. Трафикът на група A е насочен към оригиналните сървъри (инстанции 1, 2, 3 и 4) чрез балансьор на натоварването. От друга страна, Група B се насочва към отделен клъстер (5 и 6), който е снабден с нова функционалност. Този подход улеснява тестването чрез постепенното внедряване, като по този начин се минимизира риска.

В тази връзка, модела за производствено тестване Chaos Engineering представен на Фигура 3.8, дава възможност за умишлено въвеждане на дефекти в системата по време на работа, като същевременно се наблюдават реакциите. Чрез идентифициране на уязвимости в контролирана среда, екипите могат проактивно да измислят решения за подобряване на устойчивостта на системата.



Фиг. 3.7. Диаграмата илюстрираща Chaos Engineering. Източник: Sharma и колектив. (2020)

Основавайки се на цялостна оценка на системните изисквания, детайлност в тестване и устойчивост към грешки, комбинацията от Canary Release и Chaos Engineering може да осигури балансиран подход за поддържане на стабилност, като същевременно да се подобрява облачно базираната системата за управление на поръчки. Чрез сегментиране на внедряването е възможно да се наблюдава въздействието на промените в системата в реално време, а идентифицирането на грешки следва да адресира уязвимостите на системата.

От гледна точка на изходния код, проектът включва доста отделни решения в Git хранилището си. Фигура 3.8. показва пълното решение на Visual Studio, в което са организирани подпроектите.

Text

Description automatically generated with medium confidence

Фиг. 3.8. Структура на подпроектите във Visual Studio.(разработка на автора)

## Oценка на приблизителните разходи

Ценовият калкулатор на Azure е онлайн инструмент, който помага при преобразуването на прогнозираното използване на облачни услуги в приблизителна оценка на разходите, като по този начин оптимизира процеса на планиране на бюджета за разходи, свързани с услугите. Този инструмент играе роля при вземането на избор относно финансовата стратегия. Калкулаторът има способността да предостави оценка на разходите, която отразява използването на ресурси в Azure, като същевременно взема предвид всички договорени или намалени цени. Следващата таблица предоставя пълен анализ на очакваните разходи, получени от предложената архитектура, включваща изчислителни ресурси, балансиране на натоварването, шлюзове на приложения, механизми за наблюдение, бази данни, и инструменти за управление на контейнери.

Таблица 3.5.  
 Прогнозни месечни разходи за инфраструктура на облачни услуги  
(разработка на автора)

| **Тип услуга** | **Описание** | **Очаквана месечна цена** |
| --- | --- | --- |
| Набори мащаби на виртуална машина | 2 D4s v4 (4 vCPU, 16 GB RAM) (запазени за 3 години), Linux | $127.60 |
| Балансьор на натоварването | Стандартно ниво: 5 правила, 1000 GB обработени данни | $23.25 |
| Шлюз за приложения (Application Gateway) | Ниво на защитна стена за уеб приложения V2, 730 фиксирани часа, 5 GB трансфер на данни | $352.15 |
| Azure Monitor | Ежедневни регистрационни файлове, 1 показател за всяка виртуална машина, допълнителни събития и насочени известия | $46.45 |
| Azure Cosmos DBs | Стандартно осигурена пропускателна способност, запис в няколко региона; 400 RU/s x 730 часа; 4000 GB транзакционно хранилище, периодично архивиране | $1,023.36 |
| Брокер на съобщения | 24 часа пропускателна способност, приблизително 10 милиона входни събития | $153.58 |
| Инструмент за оркестрация (Kubernetes) | Стандартно ниво; Premium v2 P3V2 (16 ядра, 1,75 GB RAM, 50 GB място за съхранение); 24 часа uptime; Linux OS | $273.00 |
|  | **Oбща сума** | **$1999.40** |

*Забележка: Възможно е прогнозните месечни разходи да варират въз основа на различни фактори, като промени в ценообразуването на услугите.*

## Мониторинг и системен дневник

Ефективното водене на системен дневник и мониторинга са основни компоненти на всяка система. Разбирането на различни задължения и огромния набор от налични инструменти е от основно значение за осигуряване на оптимална функционалност на системата. Мониторинг и системен дневник са два термина, които често се бъркат. Поддържането на системен дневник е насочен към ИТ специалисти, помагайки при проследяване на грешки и разбиране на последователността от събития, довели до повреда, системен анализ и одит. В този смисъл, Elasticsearch Logstash Kibana (ELK) е от тип допълваща (поддържаща) система с отворен код, която събира регистрационни файлове от различни източници, съхранява ги за извличане и ги обработва за визуализация. Това полага основата за стабилна, надеждна и ефективна система за управление. ELK позволява не само бързо идентифициране на системни грешки, но и проактивно решаване на проблеми. Воденето на системен дневник чрез ELK включва проследяване на действията на потребителите и поведението на системата чрез документиране на дейността. Регистрират се детайли като дата и час, тип грешка, вътрешно описание и др.

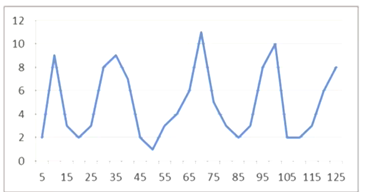
За разлика от ELK, който изисква ръчна настройка и е ориентиран към програмистите, мониторингът е интегриран в облачната платформа Azure, предлагайки услуга за наблюдение, която следва да бъде използвана от различни заинтересовани лица (като бизнес специалисти и мениджъри). В този смисъл, мониторингът може да се раздели на две категории: инфраструктурата и приложенията. Мониторингът на инфраструктурата включва оценка и контрол на системни ресурси като процесор, памет, дисково пространство и мрежов трафик. От друга страна, мониторингът на приложения се фокусира върху наблюдение на функционалността и ефикасността на отделните услуги в системата, разглежда аспекти като време за реакция, честота на грешки и проследяване на транзакции и други. Azure Monitor предоставя услуга за диагностиката в реално време, като на фиг. 3.8. е представен обобщен модел.



Фиг. 3.8. Обобщен модел на Azure Monitor. Източник: Vettor, 2023

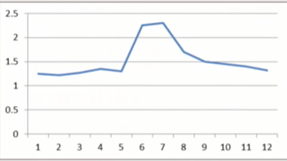
Важен аспект на мониторинга е системата за предупреждения, която се задейства, когато специфични показатели се отклоняват от нормалния си диапазон, като например когато използването на процесора надвишава 90% или ако няма създадена поръчка за последните 24 часа. След това системата уведомява определена група от ИТ специалисти, които следва да разрешат проблема. Значението на това постоянно наблюдение е свързано с очакванията на клиентите, които очакват системата да е достъпна 7 дни в седмицата и 24 часа в денонощието. Както бе отбелязано в първа глава, целите за ниво на обслужване се идентифицират с този инструмент.

Същевременно, чрез Azure Monitor се събират метрики и за бизнеса, като например кои конкретни продукти търсят и поръчват клиентите. Проучване на поведението на крайните клиенти е от основно значение за управлението в областта на веригите за доставки. За да се предвидят нуждите на клиентите, Azure поддържа услуги за използването на исторически данни, анализ и визуализиране в реално време на модел на търсенето. Тенденциите в търсенето не са постоянни, като очакван е цикличен модел, изобразен на фиг. 3.9.



Фиг. 3.9. Пример за цикличност при търсенето с Azure Monitor. (разработка на автора)

Счита се, че е възможна поява на случайни колебания, причинени от непредвидени събития, като смущения на пазара, резки промени в потребителските предпочитания или геополитическа нестабилност. Тези колебания не могат лесно да бъдат предвидени чрез анализ на исторически данни или прогнозно моделиране. Наличието на непредсказуеми вариации в търсенето подчертава необходимостта от адаптивни и стабилни стратегии за веригата за доставки, за да се управляват потенциалните последици от изчерпване или излишък от запаси. Фиг 3.10. представя пример за колебание при търсенето.



Фигура 3.10. Пример за колебание при търсенето

## Техники за анализ чрез когнитивни услуги и машинно обучение

Azure Cognitive Services предлага набор от услуги за изкуствен интелект, предназначени за надграждане на интелигентни приложения с различни възможности. За производствените организации интегрирането на тези услуги в базираната на облак информационна система следва да повиши качеството на операциите, предлагайки предимства в ефективността на обработката, точността на прогнозиране и персонализираното взаимодействие с клиентите. Интеграцията на когнитивните услуги с облачно базирана система включва избор на специфични функционалности, като например използвайки езиковото разбиране на Azure (LUIS) и текстовия анализ, системата може автоматично да интерпретира и обработва клиентски запитвания, като намали обажданията към диспечери. Също така, чрез използване на инструмента Azure Copilot, към клиентите следва да се предлагат персонализирани продуктови препоръки и визуални опции.

Също така, Azure Machine Learning, услуга за създаване и управление на решения за машинно обучение, предлага гама от инструменти и възможности за разработване, обучение и внедряване на прогнозни модели. Използвайки исторически данни и статистически алгоритми, системата следва да помага при прогнозиране на търсенето и оптимизиране на веригата за доставки.

##### Изводи и обобщения към трета глава

1. Oблачната система рационализира различни бизнес процеси, включително приемане на поръчки, планиране, производство и доставка. Тази дигитална трансформация подобрява прецизността, надеждността и скоростта, влияейки пряко върху удовлетвореността на клиентите и оперативната рентабилност;
2. Изборът на основни технологии за разработка и развитие на облачната система е стратегическа цел, която описва стъпките за внедряване на системата в процесите на продуктовата компания;
3. Мониторингът на системата, осъществен чрез инструменти като Elasticsearch Logstash Kibana (ELK) и Azure Monitor, предлага различни подходи за управление на регистрационни файлове на дневник и наблюдение на производителността и бизнес показателите. Тази комбинация осигурява екосистема, адаптираща се към изискванията на пазара и клиентите.

# Заключение

Проучването на базирана в облак информационна система за управление на клиентски поръчки в произведено предприятие разглежда редица проблеми свързани с управление на веригите за доставки в, в частност управлението на поръчки от клиенти. Основно място заемат проблеми като непрозрачност и неефективност, които ограничават оперативните способности. Изследването извежда потенциала на облачните технологии като надграждане над традиционните корпоративни системи с цел да подобри основни бизнес процеси. Очаквана е повишена прецизност, надеждност и бързина от приемането на поръчката до производството и доставката. Тази промяна води до повишена удовлетвореност на клиентите и оперативна рентабилност, давайки на организацията конкурентно предимство на пазара.

Важна част от дисертационен труд е отделена на архитектурата на системата, технически въпроси относно проектиране, програмиране, тестване и интеграция, управление на конфигурации, съображенията за сигурност, непрекъснато променящите се потребителски изисквания, цифрови заплахи, изискващи постоянни актуализации и наблюдение. Въвеждането на концептуален и логически модел, биват ръководство за потребителските и програмни интерфейси. Считаме, че специфичната дейност и изведените проблемите пред разгледаното предприятие покриват определения набор от функционалности. При разработката на моделите сме се ограничили над основни бизнес сценарии, за който са разработени набор от UML диаграми.

Потенциалът за базирани на облак системи в производството е обширен и предразполагащ към по-нататъшно развити. Бъдещи изследвания следва да се съсредоточат върху създаването на допълнителни функционалности, както да се оцени как основните промени влияят на конкурентоспособността и гъвкавостта на организацията в дългосрочен план. Както бе отбелязано в трета глава, стремежът към иновации продължава, насочвайки към подобряването на веригите за доставки чрез цифрови решения.

# Списък с фигури и таблици

[Фигура 1.1. Модел на елементите, съставящи управлението на веригите за доставки 12](#_Toc158991195)

[Фигура 1.2. Процесен модел на веригата за доставки 18](#_Toc158991196)

[Фигура 1.3. Сравнение на моделите на облачни изчислителни услуги (IaaS, PaaS, SaaS) и традиционната локална инфраструктура, подчертавайки отговорностите за управление 28](#_Toc158991197)

[Фигура 1.4. Домейн-центрирано срещу данни-центрично в контекста на диаграма за разработка на софтуер, изобразяваща време и сложност 32](#_Toc158991198)

[Фигура 1.5. Карта, описаваща връзките в DDD 33](#_Toc158991199)

[Фигура 1.6. Модел на Hexagonal архитектурa 36](#_Toc158991200)

[Фигура 1.7. Модел на clean архитектурa 37](#_Toc158991201)

[Фигура 1.8. Модел на onion архитектурa 38](#_Toc158991202)

[Фигура 1.9. Трислоен архитектурен модел 39](#_Toc158991203)

[Фигура 1.10. Пирамидата на тестове 44](#_Toc158991204)

[Фигура 2.1. Итерации процес за проектиране на концептуален модел Ingeno, 2018 48](#_Toc158991205)

[Фигура 2.2. Диаграма на контекстите 49](#_Toc158991206)

[Фигура 2.3. Диаграма на главен бизнес сценарий 50](#_Toc158991207)

[Фигура 2.4. Диаграма от високо ниво на главните приложения 52](#_Toc158991208)

[Фигура 2.5. Диаграма на базата от данни за потребителите 54](#_Toc158991209)

[Фигура 2.6. Диаграма на активност на поръчка 55](#_Toc158991210)

[Фигура 2.7. Диаграма на последователностите 55](#_Toc158991211)

[Фигура 2.8. Data Interaction Sequence Diagram 58](#_Toc158991212)

[Фигура 2.9. Скица на основен екран на приложението 59](#_Toc158991213)

[Фигура 2.10. Скица на екран за доставчика 60](#_Toc158991214)

[Фигура 2.11. Скица на екран за доказателство за доставка (ePOD) 61](#_Toc158991215)

[Фигура 2.12. Главен екран в уеб портала 62](#_Toc158991216)

[Фигура 2.13. Екран за маршрутизиране 63](#_Toc158991217)

[Фигура 3.1. Оценки на индекса TIOBE за езика за програмиране C# 68](#_Toc158991218)

[Фигура 3.2. Доставчици на облачни услуги, използвани от предприятия 69](#_Toc158991219)

[Фигура 3.3. Категоризираща на публичните облачните компании 70](#_Toc158991220)

[Фигура 3.4. Сравнение между Docker и типична виртуална машина 72](#_Toc158991221)

[Фигура 3.5. Архитектурна диаграма 74](#_Toc158991222)

[Фигура 3.6. Схема на основен DevOps работен поток 75](#_Toc158991223)

[Фигура 3.7. Диаграмата илюстрираща внедряване на облачна услуга, използвайки А/Б тестване 77](#_Toc158991224)

[Фигура 3.8. Диаграмата илюстрираща Chaos Engineering 78](#_Toc158991225)

[Фигура 3.9. Пример за тенденция при търсенето 82](#_Toc158991226)

[Фигура 3.10. Пример за сезонни колебания при търсенето 83](#_Toc158991227)

[Фигура 3.11. Пример за цикличност при търсенето 83](#_Toc158991228)

[Фигура 3.12. Пример за колебания при търсенето 84](#_Toc158991229)

[Таблица 1.1. Сравнение между права и обратна верига за доставки 8](#_Toc158991234)

[Таблица 1.2. Организационни структури в SAP 16](#_Toc158991235)

[Таблица 1.3. Обобщение на методологията на дванадесетте фактора 25](#_Toc158991236)

[Таблица 1.4. Допълнение на методологията на дванадесетте фактора 26](#_Toc158991237)

[Таблица 1.5. Стандартни за добри практики на облачната индустрия 26](#_Toc158991238)

[Таблица 2.1. Общи REST конвенции 56](#_Toc158991239)

[Таблица 2.2. Таблица с диапазоните на HTTP кодовете 57](#_Toc158991240)

[Таблица 3.1. Сравнение на сървърни технологии за разработка 67](#_Toc158991241)

[Таблица 3.2. Обобщение на практиките за управление на контейнерите 73](#_Toc158991242)

[Таблица 3.3. Обобщение на стратегии за внедряване, описани в теорията и често срещани в практиката 76](#_Toc158991243)