



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 4.6.
"Информатика и компютърни науки“

Научен ръководител: доц. д.н. Павел Петров

ВАРНА, 2024

СЪДЪРЖАНИЕ

Списък на използваните съкращения	3
Въведение.....	4
Глава 1. Проблеми на информационното осигуряване при управление на поръчките от клиенти.....	7
1.1. Управление на веригите от поръчки и доставки чрез корпоративни системи за планиране на ресурси	7
1.2. Рационализиране на процесите чрез персонализирана информационна система	20
1.3. Възможности за централизация на процесите по управление чрез прилагане на облачни технологии	25
1.4. Управление на бизнес процесите чрез ориентиран към домейн дизайн....	37
Глава 2. Архитектура на облачна система за управление на поръчки от клиенти	54
2.1. Концептуален модел на облачната система за управление на поръчките .	54
2.2. Логически модел на облачна система за управление на поръчки	63
2.2.1. Модули, поддържащи поръчки и доставки	63
2.2.2. Детайлизиране на модулите за поръчки и доставки.....	68
2.2.3. Модул за управление на потребителските профили.....	75
2.3. Комуникационни модели между модулите	81
2.4. Функционалност и потребителски интерфейс.....	90
Глава 3. Изграждане и използване на облачна система за производствено предприятие "Хайделберг Цимент Девня" АД.....	96
3.1. Обща характеристика на дейността на компанията	96
3.2. Избор на технологични средства за реализация на системата	103
3.3. Физическа реализация на системата	109
3.4. Системен мониторинг и надграждащи технологии	124
Заключение.....	134
Списък с фигури и таблици	136
Използвана литература.....	138
Списък с публикации по темата на дисертационния труд.....	144

Списък на използваните съкращения

Съкращение	Пълно наименование
API	Application Programming Interface
ERP	Enterprise Resource Planning
SCM	Supply Chain Management
MRP	Material requirements planning
IoT	Internet of things
SLA	Service Level Agreement
SLO	Service Level Objectives
SLI	Service Level Indicator
SaaS	Software as a service
PaaS	Platform as a service
SOA	Service oriented architecture
ESB	Enterprise service bus
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
DDD	Domain Driven Design
CQS	Conveyancing Quality Scheme
CQRS	Command Query Responsibility Segregation
ES	Event sourcing
CRUD	Create, read, update and delete
UL	Ubitique language
BC	Bounded context
HTTP	Hypertext Transfer Protocol
ACID	Atomicity, consistency, isolation, durability
RPC	Remote procedure call
TDD	Test Driven Development
GUI	Graphical User Interfaces
TMS	Transport Management System
UML	Unified Modeling Language

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, свързани с управлението на множество доставчици, транспортни маршрути, спецификации и качество на продуктите и много други, които водят до риск от забавяния на доставките и увеличение на оперативните разходи. Подобни проблеми ограничават способността на компаниите да бъдат гъвкави в предлагането на своите продукти и услуги 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 процесите във веригата на доставки. Например, при интегрирането на корпоративни системи за планиране на ресурсите 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 държавата, в която работят и това налага освен доброто познаване на нормативните актове и тяхното интегриране в информационните системи.

Актуалността на изследователската теза се обуславя от тенденцията облачните технологии да се превръщат в инструмент от стратегическо значение за бъдещ растеж, модернизация и цифрова трансформация на производствените предприятия. Тази тенденция ще се запази, тъй като все по-голям брой компании използват възможностите на облачните платформи, за да приложат иновативни идеи, повишавайки своята конкурентоспособност. В тази връзка, проучването изследва проблемите и решенията, свързани с

внедряването на информационни системи и техните технологични аспекти.

Тезата, която застъпваме, е, че процесите, свързани с управлението на поръчки от клиенти, поддържани и реализирани посредством различни информационно-технологични решения, могат да се интегрират в персонализирана облачна платформа, която да повиши качеството и да спомогне за усъвършенстване на бизнес процесите в производствено предприятие. Облачните системи могат 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.

В тази връзка са изследвани начините, по които производствените компании управляват информацията за поръчки и доставки – как тя се събира, съхранява, обработва и предава. Тази информация е в основата за организиране на доставката на продукти, включително тяхното товарене, транспортиране и разтоварване. Изследването обхваща основни системи и процедури, които са част от управлението на веригата за доставки и съществуващата ги информация от етапа на производство до доставянето на продуктите на крайните клиенти, с фокус върху ефективността. Някои спомагателни системи и процедури не са разгледани.

Обект на изследване са процесите във веригите за доставки в производствено предприятие, което предлага и доставя собствени търговски продукти и развива дейност чрез отделни организационни единици в множество държави. Това обхваща 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. Също така се анализира практическото прилагане на облачни технологии в основни аспекти на логистиката, като се вземат предвид нуждите на клиентите и оптималното използване на ресурсите.

Предмет на изследване са технологиите и методите за автоматизиране на логистичните процеси, използвайки съвременните постижения на облачни

платформи и средства. На тази основа се базира разработката на персонализирана информационна система с динамично променящи се във времето изисквания и параметри.

Целта на изследването е да се разработи облачна информационна система за управление на поръчките от клиенти и да се оцени въздействието и върху 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 е необходимо да се решат 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 редица научноизследователски методологии като исторически, сравнителен, системен и икономически анализ. Също така са приложени методи на логически анализ, създаване, моделиране и алгоритмизация. При апробацията на резултатите от научното изследване са използвани техники за виртуализация и прототипиране.

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

1.1. Управление на веригите от поръчки и доставки чрез корпоративни системи за планиране на ресурси

В последните десетилетия, управлението на веригите от поръчки и доставки се превърна 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 научната литература съществуват множество различни дефиниции за термина „верига на доставките“. Според някои автори (Vasilev & Stoyanova, 2019) веригата за доставки са „*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 (Jamaluddin & Saibani, 2021) 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 приемаме определението на Matinheikki et al. (2022), 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, 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 (Gupta, 2016). Асоциацията по обратна логистика¹ (RLA) дава дефиницията „Цялата дейност, свързана с продукт или услуга след точката на продажба, крайната цел на която е да оптимизира или да направи по-ефективна следпродажбена дейност, като по този начин спестява пари и екологични ресурси“.

Таблица 1.1. прави сравнителна характеристика между права и обратната верига за доставки, като извежда основните аспекти, които ги отличават.

*Таблица 1.1.
Сравнение между права и обратна верига за доставки
Източник: Gupta, 2016*

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.

¹ Асоциацията по обратна логистика (RLA) е глобална търговска асоциация, създадена през 2002 г. Служи като централен орган за насърчаване на добри практики, предоставяне на сертификати и улесняване на възможностите за работа между производители, компании за търговия и доставчици на услуги.

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. Европейска Логистична Асоциация² 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 (Bisogni et al., 2021) като начална и крайна точка на движение, дължина на пътуването, скорост, време на движението, време на престой, вид на използваните транспортни средства,

² Европейската логистична асоциация (ELA) е федерация на националните логистични асоциации в Европа, създадена през 1984 г. и базирана в Брюксел, Белгия. Фокусирана е върху подобряването на логистиката и веригата за доставки. Ключови функции включват програми за сертифициране, стандарти и осигуряване на качеството.

условия на транспортиране. В тази смисъл, Василев и колектив (2023) разглеждат логистичната система като устойчива мрежа от звена, които са взаимно свързани и управявани 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.

Среща се понятието „информационен поток“, което представлява обмен на данни, чрез документи или по друг начин като следствие на логистичния поток от стоки. Информационния обмен е част от логистичната система и е от съществено 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*“ (Zajac & Swieboda, 2023). Всеки бизнес организира този процес по индивидуален начин, съобразено със специфичните си нужди и процедури.

Милушева (2023) свързва стратегията и планирането на веригата за доставки с конкурентоспособността на производствени предприятия на световния пазар за строителни материали, докато редица автори и изследователи (Парушева & Александрова, 2022; 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.

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 (Alzoubi et al., 2020). 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.

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 (Tseng et al., 2019). 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. Bardakci (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 разграничават 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 (Sulova, 2021; 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 Hahn (2019) ефективното управление на материалните и съществуващите 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 Chen (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 (Lee et al., 2022).

Управлението на веригата за доставки (SCM) се развива в ерата на информационни системи и базираните на облак, софтуерни решения (Le, 2020). SCM представлява надзора и координацията на елементите от веригата, осигурявайки непрекъснат мониторинг, ефикасност и ефективност в работните процеси. Значението на иновациите във SCM е свързано с разнообразието на потребителско търсене. Възможните компоненти, които влияят върху SCM, могат да бъдат координирането в работния процес, работната сила, машините и оборудването и актуализирането в реално време на информационните потоци, връзките с инвеститорите, логистика, стратегия за веригата за доставки, планиране на веригата за доставки, доставки, управление на активи, управление на жизнения цикъл на продукта, снабдяването и корпоративни приложения, които поддържат управлението на информационните потоци (Alhabatah et al., 2023). Описание на компонентите, които считаме че са основни в SCM, е представено в таб. 1.2.

*Таблица 1.2.
Основни компоненти на управлението на веригата за доставки
(SCM)*
Източник: Alhabatah et al., 2023

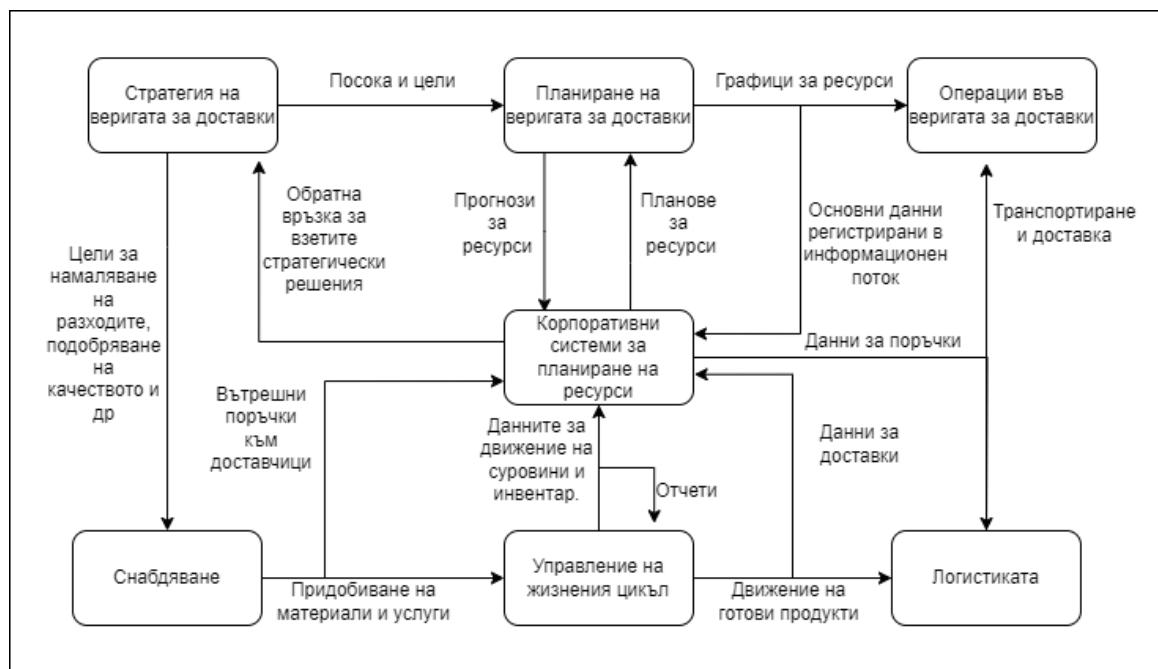
Компонент	Описание
Стратегия на веригата за доставки	Компонент, който установява целите и подхода на

	SCM, насочвайки как планирането, операциите и други процеси следва да се съгласуват.
Планиране на веригата за доставки	Компонент, включващ прогнозиране на търсенето, разпределение на ресурсите и планиране на доставките. Той има за цел ефективно да балансира търсенето и предлагането и да подготви организацията за бъдещи нужди.
Операции във веригата за доставки	Това е компонент, в който се изпълняват SCM планове. Включва ежедневни дейности като изпълнение на поръчки, производство и транспортиране на стоки и продукти.
Корпоративни приложения във веригата за доставки	Технологични инструменти, които поддържат оперативните аспекти на SCM, включително системи за управление на отношенията с доставчици и клиенти, системи за управление на поръчки, системи за управление на инвентара и др.
Снабдяване	Процес на придобиване на стоки и услуги, необходими на SCM, пряко свързан със „стратегията на веригата за доставки“.
Управление на жизнения цикъл на продукта	Това е управлението на продукт от проектирането, производството, обслужването и приключването. Част е от цялостната стратегия и влияе върху „планиране на веригата за доставки“ с информация за състоянието и продуктите.
Логистика	Транспортиране на стоки в рамките на SCM, свързан с „операции във веригата за доставки“. Осигурява физическият поток от продукти, в синхрон с планирането и стратегията.

Фиг. 1.1 представя SCM модел, в който корпоративните системи за планиране на ресурси (ERP) заемат централно място в организацията на потока от стоки, информация и капитал (Türkay et al., 2016). SCM модел фокусира върху осигуряването на ефективност и оптимизация на ресурсите, структурирана около посочените по горе компоненти, които обхващат стратегическо планиране, операционни практики и анализ на данни. Стратегията на веригата за доставки ръководи цялостната посока и целите,

докато планирането превръща тази стратегия в конкретни действия (Sánchez-Flores et al., 2020).. Планирането на веригата за доставки служи като стратегическа рамка за операциите, включвайки създаването на прогнози и графици. В връзка с операциите се генерират данни, които биват администрирани чрез корпоративни системи, които служат като инфраструктура за изпълнението (Chiang et al., 2021).

В тази връзка, вземането на стратегически решения се информира от отчети и анализи, създадени от данните в корпоративните системи. Процедурите по снабдяване се ръководят от характеристики като намаляване на разходите, подобряване на качеството и др. Процедурите по снабдяване са отговорни за получаването на необходимите материали и услуги. Логистичните нужди от сировини, запаси и складиране и транзит на готови продукти се определят от информацията за етапа на продукта. Логистиката се свързва с изпълнение на оперативните цели, със фокус към транспортиране и доставяне (Vasilev & Cristescu, 2019).



*Фиг 1.1. Модел на компоненти, съставящи SCM
Източник: Türkay et al., 2020 Адаптиран 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 аспекти:

- 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;
- 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 (Goodman, 2019);
- 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 ERP 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 (Vasilev, 2015).

На базата на някои от последните класации на S&P Global Ratings³ за

³ S&P Global Ratings предоставя оценка на компаниите в индустрията за строителни

компании за строителни материали към 6 февруари 2024 г., внедряването на системи за ERP 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 и ERP 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 (Tang & Xia, 2023).

Основната цел на ERP 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 е 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 Materials Requirements Planning⁴ (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

материали. Класирането на компаниите става по рейтинг, перспектива, самостоятелен кредитен профил (SACP), профил на бизнес и финансов риск и оценка на ликвидността.

⁴ MRP е система за планиране и управление на материалите, която помага на предприятията да определят колко материали са необходими и кога те трябва да бъдат закупени или произведени. Основната цел на MRP 1 е да осигури материалните нужди в производството, докато MRP 2 включва аспекти като минимизиране на запасите, увеличаване капацитета на производствените мощности, планиране на работната и др.

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 в Германия. Според Gaur (2020) 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 (Templar et al., 2020). Всяка отделна функция, от която едно производствено предприятие може да има нужда, е достъпна и интегрирана в SAP. Към април 2024г. 86% от компаниите от Fortune 500 и 92% от компаниите от Forbes Global 2000 са клиенти на SAP. Освен това, 77% от приходите от транзакции в света преминават през тази система и тя се използва от над 400 000 организации в 180 държави (Duff, 2024).

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 изследвания (Gargeya & Brady, 2005, Ojra et al., 2021), 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 (Schneider, 2020).

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 (Bier et al., 2019). В допълнение към тях е модулът за планиране на производството, в който се разпределят ресурси и се планират производствените дейности. Също така, 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. Тези модули образуват архитектурна рамка в SAP за подобряване на прозрачността,

гъвкавостта и устойчивостта на веригата за поръчки и доставки (González et al., 2024).

Значението на основните и транзакционни данни е предмет на изследване от редица автори (Hildebrand, 2018; Pjp, 2023). Според Dogra et al. (2022) 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. Всички подчертани някои, но не всички аспекти на организационните структури.

*Таблица 1.3.
Организационни структури в SAP
(Magal & Word, 2013)*

Финанси	Продажби и Дистрибуция	Управление на Материали
Сметкоплан (Chart of Accounts)	Търговска организация (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)

Becker et al. (2016) описва 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 фирмени кодове. Една компания може да има множество кодове и всеки фирмрен код може да има множество бизнес области. Пример за бизнес област в рамките на фирмрен код е производство.

При продажбите и дистрибуцията търговската организация е на най-високо ниво и цялото отчитане на продажбените дейности се извършва на ниво търговска организация (Von Aspen, 2020). 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., 2014) 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 (Sulova et al., 2020). 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 (Todoranova & Penchev, 2023).

1.2. Рационализиране на процесите чрез персонализирана информационна система

Според проучване на Hasim et al. (2018) 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, електронни устройства за обмен на данни, технологии за "Интернет на Нещата" (IoT), системи за управление на складове (WMS) и системи за управление на транспорт (Aleksandrova, 2020; 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.

Изследването на разнообразни литературни и интернет източници показва липсата на специално разработен модел на SCM система. Чрез литературен анализ на статии, публикувани в научни списания като 'Journal of Supply Chain Management', 'International Journal of Production Economics', и 'Supply Chain Management: An International Journal', както и на доклади от международни конференции, включително 'International Conference on Logistics and Supply Chain Management', се наблюдават различни подходи и нови модели в областта на логистиката и SCM (Verdouw et al., 2010; Cichosz et al., 2020; Agarwal, 2021). На базата на тези изследвания, предлагаме разработването на централизирана система, която интегрира част от вътрешни

подсистеми на производствено предприятие. Важно да се отбележи, че те са избрани на базата на предходния модел на компонентите, съставящи SCM. Тази система е проектирана така, че да предоставя публично достъпни данни на крайните клиенти, което улеснява прозрачността и ефективността на процесите и взаимовръзките във SCM.



Фигура 1.2. Модел на SCM система (разработка на автора по (Caserio & Trucco, 2018))

Фигурата илюстрира серия от взаимосвързани подсистеми, които са интегрирани помежду си. Проектирането на тази система бива съобразено с изисквания на конкретно предприятие, тъй като се вземат предвид растежът на цялостно разширяване на бизнеса, както и справянето с глобалните проблеми (Luo, 2010). Системата представлява комплексен механизъм, който интегрира различни подсистеми и процеси за оптимизация на потока на материали, информация и финанси между различните звена от производство до крайния потребител. В основата и стои централизирано информационно управление, което позволява рационализация и подобряване на достъпа до актуална информация. Подсистемите, варират от управление на вътрешно фирмени ресурси като логистика и складови наличности, до външни взаимодействия като клиентски връзки и качество на продуктите

(Aleksandrova, 2021). Фигурата показва взаимодействията между различните компоненти, свързани с управление на качеството, идентифициране на проблеми в доставките, поддръжка на обратна връзка с клиентите и анализ на техните предпочтения. Тази интеграция следва да подобри операционната ефективност, стратегическото планиране и реализация на поставените цели. Отличителна черта на SCM системата е нейната способност да внедрява автоматизирани процеси и алгоритми за непрекъснатото подобреие и адаптивност към променящите се пазарни условия и изисквания (Kakhki & Gargeya, 2019).

Унифицираната спедиторска платформа за логистични услуги, свързва различни заводи, географски райони и бизнес единици. Поддържа основни до сложни логистични изисквания (Petrov 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. Позволява на бизнеса и крайните клиенти да имат видимост и контрол върху пратките, което им помага проактивно да управляват всички проблеми.

ERP подсистема заема централна позиция, като обработва данни от повечето други подсистеми (Rajapakse, 2023). Както бе отбелязано, ERP осигурява информация за наличните ресурси, планиране на производството и поддържане на баланс между наличностите и търсенето. Подсистема за управление на качеството осигурява постоянен мониторинг и контрол на качеството на продуктите и процесите в организацията. CRM подсистемата (Александрова, 2020) поддържа връзка с клиентите, управлява информацията свързана с клиентски данни и помага за оптимизиране на продажбите и маркетинговите стратегии. Подсистема за управление на транспорта координира и оптимизира процесите на доставка и транспорт, интегрирайки се с други подсистеми за ефективно планиране и изпълнение на поръчки. Подсистема за управление на склада администрира складовите процеси,

инвентара и управлява наличностите. Подсистемата за бизнес анализ извлича данни от другите подсистеми за да определи производителността, тенденциите на пазара и други ключови бизнес метрики, които могат да подпомогнат стратегическото развитие (Ren et al., 2019; Schniederjans et al., 2020)). Също така предоставя стратегически прогнози, които подпомагат вземането на бизнес решения (Ramakrishna, 2022). Подсистема за мониторинг на производството предоставя възможности за наблюдение и контрол на процесите на производство, информирайки другите подсистеми за състоянието на производствените операции. Изброените подсистеми са свързани така, че да осигуряват ефективен поток на информация в организацията. Те сами по себе си представляват информационни системи, които участват в управлението на поръчки и ресурси на организация. Интеграцията между тези системи позволява обмена на разнообразни данни. В тази връзка, фигурата представя някои от основните типове данни, които се прехвърлят при интеграцията между различните подсистеми (Novais et al., 2019).

В разгледания случай, ERP изисква актуална информация за наличностите от склада, както и данни за текущото състояние на производствените процеси. Информация за продажби и доставки се обменят между системите за управление на склада, транспорта и ERP, за да се поддържа актуално състояние на финансите и да се осигурява точност на данните за фактуриране и счетоводство (Димитров, 2020; Атанасова, 2021). В тази връзка, за координирането на доставките и логистиката, ERP предава информация за поръчките и доставките към системата за управление на транспорта, която оптимизира маршрутите и графиците за доставка. От друга страна, CRM системата предоставя информация за нуждите и предпочитанията на клиентите на ERP, за да помогне при планирането на ресурсите и управлението на поръчките в съответствие с търсенето на пазара. Системата за управление на качеството предоставя информация за съответствието на продуктите със стандартите и изискванията, които биват

разгледани както от служители, така и от клиенти. Информация за състоянието на машините, скоростта на производство, честотата на прекъсванията, които се използват от системите за управление на производството и бизнес анализа за оптимизиране на операциите, биват предадени на ERP и CRM. Данните от системата за управление на околната среда могат да се използват от ERP за подобряване на устойчивостта. ERP, от друга страна, предава информация за потреблението на ресурси и отпадъците към системата за управление на околната среда, която анализира и предлага мерки за намаляване на въздействието.

Технически модули на ERP, като SAP Netweaver Gateway⁵ (Bönnen et al., 2018) 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 за SCM устойчивост (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

⁵ SAP NetWeaver Gateway е технология, която улеснява свързването между SAP приложения и други платформи. NetWeaver предоставя начин за интеграция чрез използване на API.

данни, включващи визуални представления, 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 (Sharma et al., 2020). 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 (Парушева, 2011; Тодоранова, 2015; Partida, 2023; Roy, 2023; Microsoft Research, 2023) подчертават значението на облачните технологии за оптималното функциониране на веригите от доставки. Облачните изчисления, изграждат 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.

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, 2024), които следва да бъдат разгледани в детайли.

В областта на облачните изчисления може да разграничат публични, частни и хибридни облачни архитектури, които имат различно въздействие върху ИТ стратегията. Публичните облаци предоставят изчислителни ресурси чрез интернет, което позволява достъпност в голям мащаб без необходимост от капиталови инвестиции в реална инфраструктура. Частните облаци, от друга страна, са персонализирани среди, проектирани и управлявани специално за конкретни предприятия. Това позволява повече защита и контрол върху данните, но също така води до по-голяма отчетност за администриране и поддръжка на инфраструктурата. Хибридните облаци са резултат от комбинирането на публични и частни, което позволява на предприятиета да

комбинират мащаба и рентабилността на публичните облаци с персонализираната сигурност и контрол на частните. Като обобщение, таблица 1.4 прави обзор на някои от основните характеристики и ограничения на трите вида.

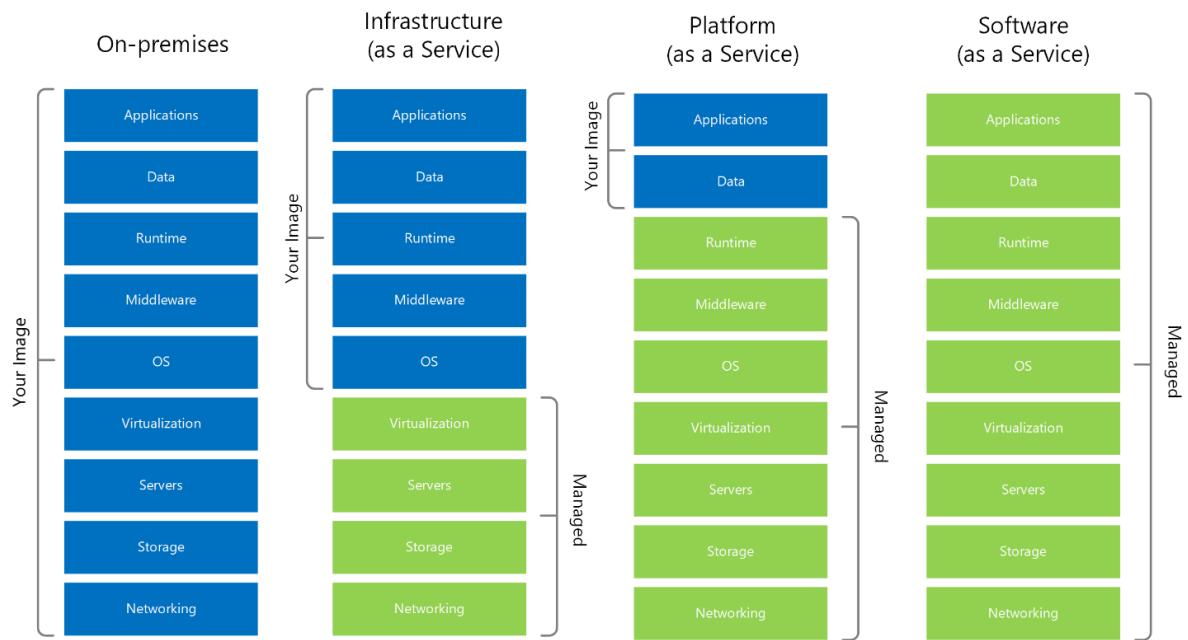
*Таблица 1.4.
Сравнение на публични, частни и хибриден облаци
(Dotson, 2019)*

Характеристика	Публичен	Частен	Хибриден
Хостинг	Услуги, предоставяни извън организацията през Интернет.	Инфраструктура, оперирана за една организация.	Комбинация от обществени и частни облаци, позволяваща обмен на данни и приложения между тях.
Икономическа Ефективност	Висока, поради модела плащане според употреба и липсата на инвестиции във физическо оборудване.	По-ниска в сравнение с обществените облаци поради началната инвестиция в хардуер и разходите за поддръжка.	Предлага баланс, като позволява на организациите да използват икономическите предимства на обществените облаци за операции, които не са критични, а останалите в частни облаци.
Мащабируемост	Висока мащабируемост, с ресурси на разположение от тип „при поискане“ за удовлетворяване на завишени търсения и обратно.	Мащабируемостта е ограничена от капацитета на физическата инфраструктура, изискваща планиране и инвестиции за увеличение.	Подход, при който част от операциите се поддържат в частния облак, а друга част, в публичния, според нуждата от сигурност.
Сигурност	Високи нива на сигурност, основаващи се на водещи доставчици на услуги.	Най-високо ниво на сигурност сред 3те вида, тъй като ресурсите не се споделят с други организации.	Предлага компромис, като поддържа чувствителни данни и приложения в по-сигурния частен облак, докато други операции могат да се възползват от публичния.
Контрол	Ограничено контрол над изчислителната среда и основната инфраструктура.	Пълен контрол над средата и инфраструктурата, позволяващ персонализирани настройки.	Най-сложен за изпълнение и поддръжка, тъй като ресурсите се споделят между различни видове услуги.

Сред трите вида, публичният облак изглежда като подходящ, предвид неговата висока степен на мащабируемост, икономическа ефективност, както и публичния характер на услугите, насочени към клиенти. Публичните облаци предлагат възможността за мащабиране на ресурсите в отговор на променящо се търсене, осигурявайки необходимата компютърна инфраструктура. Не е необходима инвестиция в хардуер. Въпреки че съществуват потенциални

рискове свързани със сигурността, те следва да бъдат избегнати чрез избор на доставчик на облачни услуги, както и сертифициране към ISO 27001. В тази връзка, предимствата, които публичният облак предлага, надхвърлят потенциалните недостатъци.

Изходйки 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 (Филипова и др., 2019). Хардуерът е разположени в център за данни, поддържан от облачен доставчик като Microsoft Azure, Amazon AWS, Google Cloud или много други. ИТ специалистите създават виртуални ресурси, без необходимост от закупуване или поддръжка на устройства. Представено на традиционния подход, при който хардуера е собствен и изиска цялостно управление и поддръжка от ИТ отдела. (Endo et al., 2016) Противоположно на това, инфраструктурата като услуга (IaaS) абстрактира физическия хардуер в среда, управлявана външно, позволявайки на организациите да изнесат слоеовете 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

Фигурата подчертават оперативните и икономическите ползи от моделите за облачни изчисления – инфраструктура като услуга (IaaS), платформа като услуга (PaaS) и софтуер като услуга (SaaS). Fields et al., (2009) посочват, че тези модели предлагат по-ефективно разпределение на ресурсите спрямо традиционната локална инфраструктура, тъй като работят върху модел на ценообразуване, базиран на потреблението. Този подход позволява на организациите да плащат само за ресурсите и лицензите за софтуер, които действително използват, подобрявайки предвидимостта и управлението на разходите. В тази връзка, "изчисленията без сървър" (serverless), както са описани от Garverick и McIver (2023), представляват друг аспект на обачните услуги, при който отделни функции се изпълняват в отговор на конкретни събития. Този модел допълнително намалява необходимостта от правление на ресурси, предлагайки рентабилно решение за приложения с определена функционалност. Необходимо е да се отбележи, че при преминаването към IaaS, PaaS или SaaS от локални центрове, компаниите могат да се сблъскат с проблеми, свързани със суверенитета на данните, съответствието и необходимостта от експерти със специализирани умения за управление на

облачни услуги (Kumar & Agnihotri, 2021). Базираните в облак услуги за възстановяване "след бедствие", описани от Guo (2013), осигуряват основна защитна мрежа, позволяваща възстановяване на операциите в случай на загуба на данни или системни повреди. За справянето с този проблем при локалните решения, компаниите трябва да поддържат дублирани хардуерни и софтуерни среди, които може никога да не бъдат използвани. Докато IaaS, PaaS и SaaS предлагат различни предимства пред локалната поддръжка по отношение на ефективността на разходите и оперативната гъвкавост, те също така въвеждат нови съображения относно сигурността на данните, съответствието и техническият опит, необходим за използване на тези технологии.

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:

$$\text{Време за отговор} = \text{Време за обработка} + \text{Време на изчакване}$$

В посочено „уравнение“, *време за отговор* е общото време, от момента, в който потребителят изпрати заявка до момента, в който получи отговор. Счита се, че това е интервала от време, в което потребителят изчаква за да види резултат след започване на действие (Marinova, 2023). *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 подобрене. 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 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 and Hasselbring (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*):

$$\text{Availability} = \text{uptime} / (\text{uptime} + \text{downtime})$$

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 рекламиите 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., 2018) анализират редица фактори и разработват методология, наречена „дванадесет фактора“ (Twelve-Factor), представена в една от първите облачни платформи - Heroku. Тази методология предоставя набор от принципи и практики, към които разработчиците да се придържат, когато създават приложения, оптимизирани за съвременни облачни среди. Практици (Grafiati, 2022) считат методологията на дванадесетте фактора 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 1.5.
Обобщение на методологията на дванадесетте фактора
(адаптирано от автора)*

Фактор	Описание
Code Base	Единична база за сурс кода на всяка микроуслуга, съхранявана в собствено хранилище към GitHub, GitLab, Azure DevOps и други. Чрез контрол на версии, 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 адресиран 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.

Фактор	Описание
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 наಸърчаването на същата среда за изпълнение.
Logging	Регистрационните файлове, генериирани 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 1.6.
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 таб.1.7. и представя пет стълба на

т.н. „добра архитектурата“.

*Таблица 1.7.
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.

Проучвания на източници в областта (Li 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 функция 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 методологията на дванадесетте фактора, 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 бизнес (или домейн) сложност с течение на времето. Нарастването на бизнес сложността води до неструктуррирана и трудна за поддръжка база от програмен код. 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 се различава 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., 2024) 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

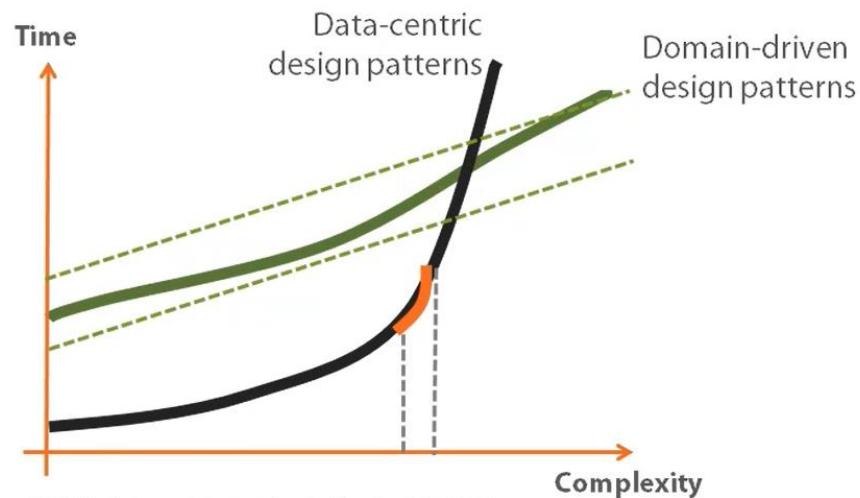
към домейн дизайн“ (ОДД), се определя ефективността в ситуации, при които има нужда от обработка на сложни бизнес изисквания. С други думи, ОДД цели да се справи със сложността в базата от код, която обхваща алгоритмите и структури от данни за бизнес правила, валидации и изчисления (Millett & Tune, 2015). 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, управяван от данни“ (data-driven design), описан от (Erl, 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, Evans (2014) твърди, че ОДД предоставя ключови концепции за разделяне на уеб услугите, на базата на други фактори след като формулира методология, за ОДД и предоставя начин за представяне на реалния свят чрез структурирано решение, което отговаря на изискванията в проблемното пространство. Тези характеристики допринасят за подобреие 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 ОДД (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.

Фиг. 1.4. илюстрира връзката между времето, цената и сложността при проектирането на софтуер (Fowler, 2012). 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)

За разлика, 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 ОДД (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.

ОДД предоставя различни технически концепции и модели (Uludağ et al., 2018), които могат да бъдат използвани за внедряването на софтуерни проекти. Представени като контекстна карта на фиг 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 ОДД. 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 ОДД методологията (Steinegger et al., 2017).

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 необходимо да се разбере и използва терминологията, както и да се осигури нейно съответствие в програмния код, за се реализират бизнес целите. Основна характеристика на ОДД 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. Batista (2022), подчертава необходимостта 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. UL се развива и се поддържа с течение на времето, като предоставя средство за събиране и организиране на знанията и бизнес логиката (Rademacher et al., 2018).

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.

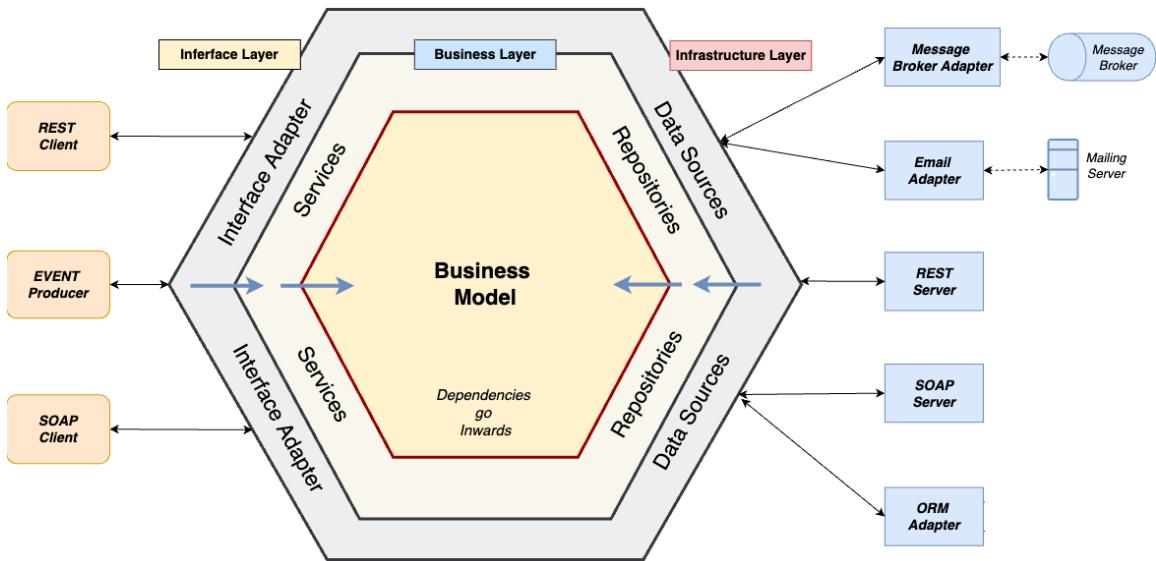
Vernon (2016) описва стойностни обекти, които следва да са малки, прости обекти, които не се основават на идентичност. Те са елементи, използвани за количествено определяне, измерване или характеризиране на определена тема. Стойностните обекти могат да имат методи и поведение, но никога не трябва да имат странични ефекти. Примери за такива обекти могат да бъдат представените по-горе организационни структури.

От друга страна, сред публикациите в ОДД областта, агрегатът е

представен като колекция от свързани елементи, които се модифицират като едно цяло (Hippchen et al., 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).

Stonis (2024) 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 приложението по-лесен за тестване в сравнение с традиционния вариант, при който външните ресурси се свързват пряко (Армянова, 2018). 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; Vieira, 2023) Hexagonal, Clean, and Onion архитектурите поддържат високи нива на модулност и разделяне на проблемите. Hexagonal (или шестоъгълна) архитектура, представена на фиг.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 архитектура. Източник: Cockburn (2022)

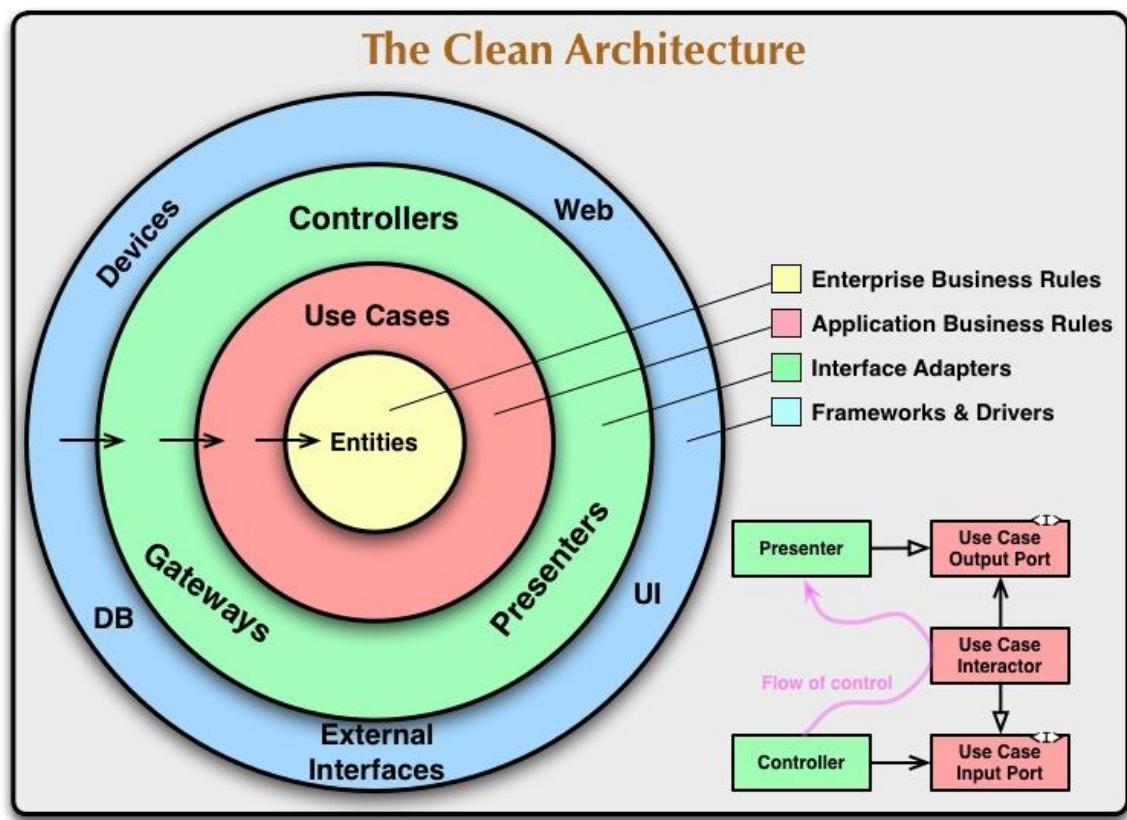
Clean (или чиста) архитектура, представена от Martin (2017) и илюстрирана на фиг.1.7., се базира на Hexagonal⁶, като я допълва чрез разделяне на системата на слоеве с правила за зависимости, което въвежда йерархия, с идеята че вътрешните слоеве следва да останат незасегнати от промените във външните, но за сметка на това, външните са пряко зависими от промените във вътрешните. Основната хипотеза е, че такова разделение улеснява управлението на зависимости, което помага при поддръжане и разширение на софтуера.

Архитектурата, представена на фиг. 1.7, е организирана в „концентрични кръгове“, включвайки:

- Entities: Обекти, които представляват модели или бизнес правила;
- Use Cases: Модули, които организират бизнес правилата за конкретни приложения или функции;
- Interface Adapters: Преобразуват данни между форматите, нужни за бизнес правилата;

⁶ Основната цел на Hexagonal архитектурата е да предостави модулност и възможност за тестване чрез разделяне на два основни типа: портове и адаптори. Тази архитектура е разработена от Alistair Cockburn за разработката и поддръжката на софтуерни системи.

- Frameworks and Drivers: Слой, който включва всички външни компоненти като бази данни, уеб сървъри;



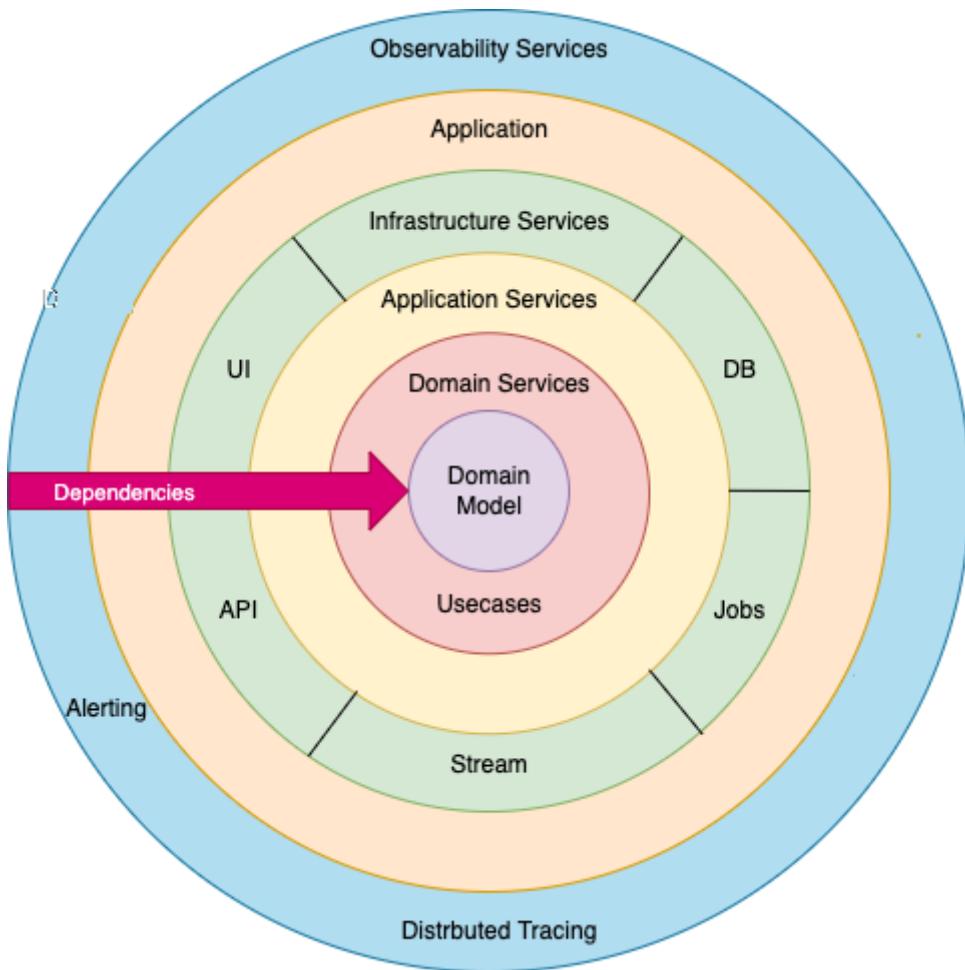
Фиг. 1.7. Модел на Clean архитектура. Източник: Lano & Tehrani, 2023

Clean архитектурата дава насоки за създаване на софтуер, който да бъде разбираем, тестваем и адаптивен в дългосрочен план, като при това се съобразява с различни бизнес изисквания и технологични среди.

Комбинацията от разгледаните две архитектури с ОДД принципите, описани горе, централизира модела на домейна, обкръжавайки го с приложни и инфраструктурни слоеве. Представена на фиг.1.8. „onion архитектура“⁷ използва тези слоеве и централно ядро. Горните слоеве зависят от долните, но не и обратно, показвайки, че основните елементи на ОДД трябва

⁷ Подобно на Hexagonal и Clean, Onion архитектурата се основава на идеята за висока степен на независимост на компонентите, управлявайки поддръжката и развитието на приложенията. Интегрира техническите подходи като Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion.

да работят независимо един от друг.

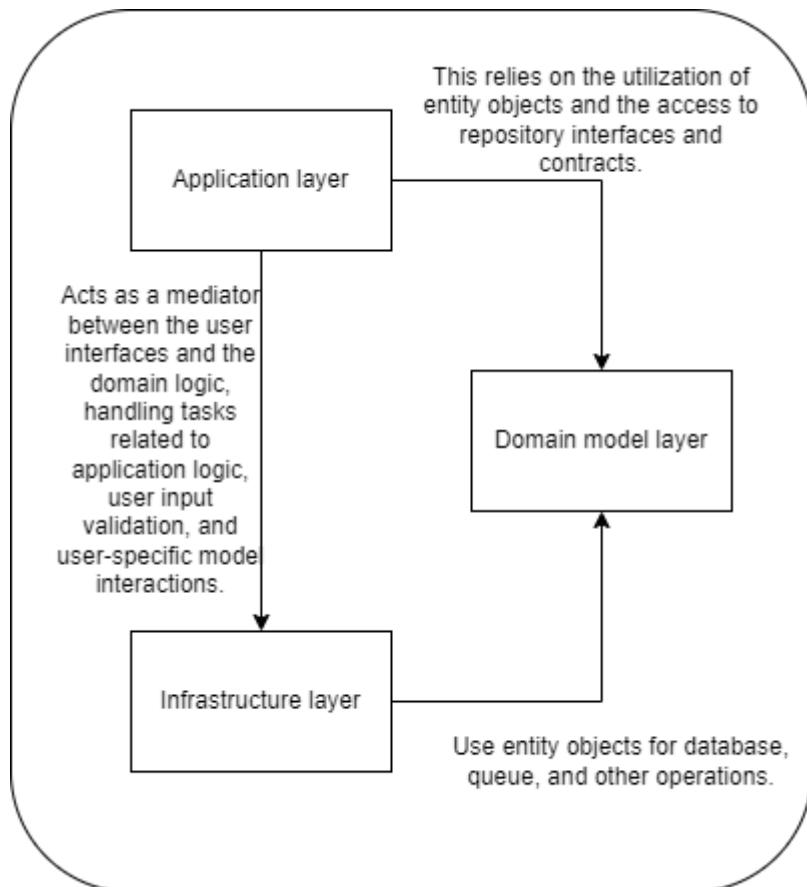


Фиг. 1.8. Модел на onion архитектура. Източник: Palermo, 2018

В този смисъл, важен аспект от проектирането и създаването на услуга е определянето на нейните граници. Както беше споменато по-горе, всеки ВС идентифицира субектите и стойностните обекти, характеризира ги и ги комбинира. Изборът къде да се направи границата между ВС 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 et al., 2021), голямата част

от корпоративните приложения имат различни нива, които помагат на разработчиците да управляват сложността на кода. Спазвайки принципите на ОДД, елементите могат да бъдат организирани в няколко слоя, както е показано на фиг 1.9.



Фигура 1.9. Трислоен архитектурен модел. Източник: Vettor & Smith (2024)

Счита се, че приложният слой координира потока на изпълнение между различни обекти (Армянова, 2018). 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 абстрагира 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 ОДД. 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 извлечане 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 вземат в предвид стриктно принципите 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, 2023). В този случай разделението на тези задачи на два отделни метода може да стане нелогично. Следователно, наложително 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 „манипулятор 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 стандартизириани 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 да осигурят 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

данные.

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 определена същност, необходимо е да се повтори времевата линия на програмата от самото начало. Този алгоритъм включва изследване на данните и използване на логика за извлечение на съответната информация. Чрез използването на записаните събития, е възможно да се реконструира състоянието на определен агрегат. Това понякога може да изиска управление на огромни обеми от данни. В такъв случай могат да бъдат записани „проекции на данните“, които представлят състоянието в определен момент от време. Веднъж съхранени, събитията са неизменни. Изходейки 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 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“ (Kuyumdzhev & Nacheva, 2020).

В този смисъл, разработка, управлявана от тестове (TDD) и ОДД са две

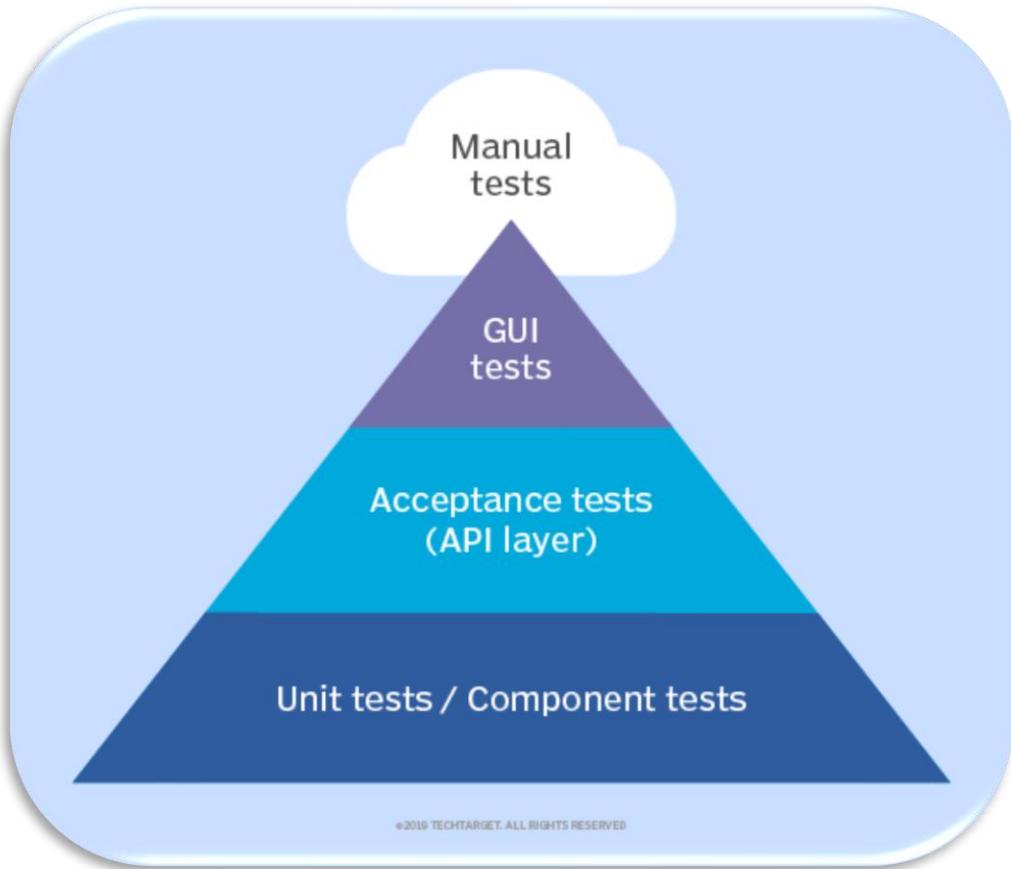
методологии, които при комбинация, могат да повишат качеството на облачните услуги и процеса на разработка. Използвайки тези практики, разработчиците и инженерите за осигуряване на качеството могат да създадат система, която е надеждна. TDD насърчава процес на тестване, при който тестовете се пишат преди кода за внедряване. Този процес следва добрите практики, като кодът, който бива написан за тестовете валидира изпълнението предвидената функционалност. Съществува процедура в три стъпки, известна като „red, green, refactoring“ (Myers, 2022). 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 в процеса на тестване ръководи дизайна на алгоритми, структури от данни, функции и класове в кода на приложенията.

Софтуерното тестване обхваща няколко различни категории. Някои тестове са базирани на предмета – например компонентно тестване (Наков). Други се определят от целта на теста – например функционални тестове, тестове за приемане и проучвателни тестове. Друга категория е определена от вида на тестове – например автоматизирани, полу-автоматизирани и ръчни тестове.

Фиг. 1.10. представя “пирамида за автоматизация на тестовете” (Cohn, 2009). Тя изобразява няколко различни категории тестове, които следва 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 специалистите по осигуряване на качеството⁸ следва да обърнат повече внимание на основните

⁸ Специалистът по осигуряване на качеството (QA) участва в разработването и поддържането на стандарти за продукти или услуги.

тестове, преди да преминат към по-сложните.



Фиг. 1.10. Пирамидата на тестове. Източник: Mike Cohn (2010)

На фигурата са идентифицирани четири различни вида тестове:

- 1) Компонентно тестване (Unit) - автоматизирани тестове, които използват за тестването на отделни компоненти (units) от кода на софтуера. Тези компоненти представляват най-малката част, съществуваща в едно приложение, която позволява да бъде тествана индивидуално. Крайната цел е изолирането на конкретно парче код и установяването дали компонентът работи правилно. Когато става дума за ООП, тези компоненти обикновено са цели класове, а понякога и отделни методи, а в процедурното – най-често са отделни процедури или функции. Компонентното тестване е основен етап от разработката на софтуер, тъй като позволява ранното засичане на проблеми и несъвършенства в кода, чието откриване и отстраняване на по-късен етап може да бъде по-сложно и времеемко. Това е първото ниво в пирамидата на софтуерно тестване и се изпълнява преди всички останали категории. Всички

сценарии се тестват индивидуално в изолирана среда, за да се премахне възможността за зависимости в кода;

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

3) GUI 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;

Итеративният характер на TDD позволява честа обратна връзка, което улеснява непрекъснатото усъвършенстване и адаптивност при разработването на облачни услуги (Ashbacher, 2010).

Трябва да се отбележи, че подходите, представени в този раздел, които включват ОДД, CQRS, ES и TDD, може да не притежават универсална приложимост във всички архитектури. Основно ограничение е значителното време и усилия, необходими за разбирането и прилагането на слоевете, моделите и концепциите, присъщи на ОДД. Кривата на обучение, свързана с овладяването на ОДД, често се описва в литературата като стръмна. Това може да бъде пречка за екипи без предишни познания, което да доведе до неправилно прилагане на методологията. Характеристиките на изброените техники, изискват значителна степен на опит и задълбочено разбиране на бизнес сферата. Това не само удължава първия период на развитие, но също така изисква непрекъснати инвестиции в обучение на екипа и подобряване на процесите. Следователно е важно внимателно да се оцени пригодността на тези подходи по отношение на технологичната зрялост на организацията, както и временевите рамки на проекта.

В заключение, гореспоменатите подходи за проектиране, се налагат 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. Прилагането на ОДД и облачни архитектури следва да помогне на организациите с внедряването на иновации, прогнозиране на разходи, предоставянето на услуги на своите клиенти и конкурентоспособността в бързо променящата се верига от доставки.

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 с намалена ефективност и ограничения на мащабируемостта, са значими и изискват технологична намеса;
2. Предвид потенциала на облачните технологии да усъвършенстват традиционните системи, те представят алтернатива за справяне с посочените по-горе проблеми;
3. В главата се подчертава необходимостта от асимилиране и адаптиране на принципи от сферата на софтуерния дизайн и управлението на бизнеса, за да се създаде оптимално решение в облачна среда;
4. Въпреки че базираните на публичен облак системи предлагат множество предимства, те не са лишени от проблеми, като например сигурността. Динамичният характер на технологиите изиска непрекъснати актуализации на софтуера, както и мониторинг на всички облачни услуги. Важно значение се дава на протоколите за сигурност и проактивните стратегии за защита срещу уязвимости;
5. В главата се 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. В първа глава са представя теоретични основи в областта на стопанската и информационна логистика, управлението на веригите от поръчки и доставки, както и системите за планиране на ресурси.

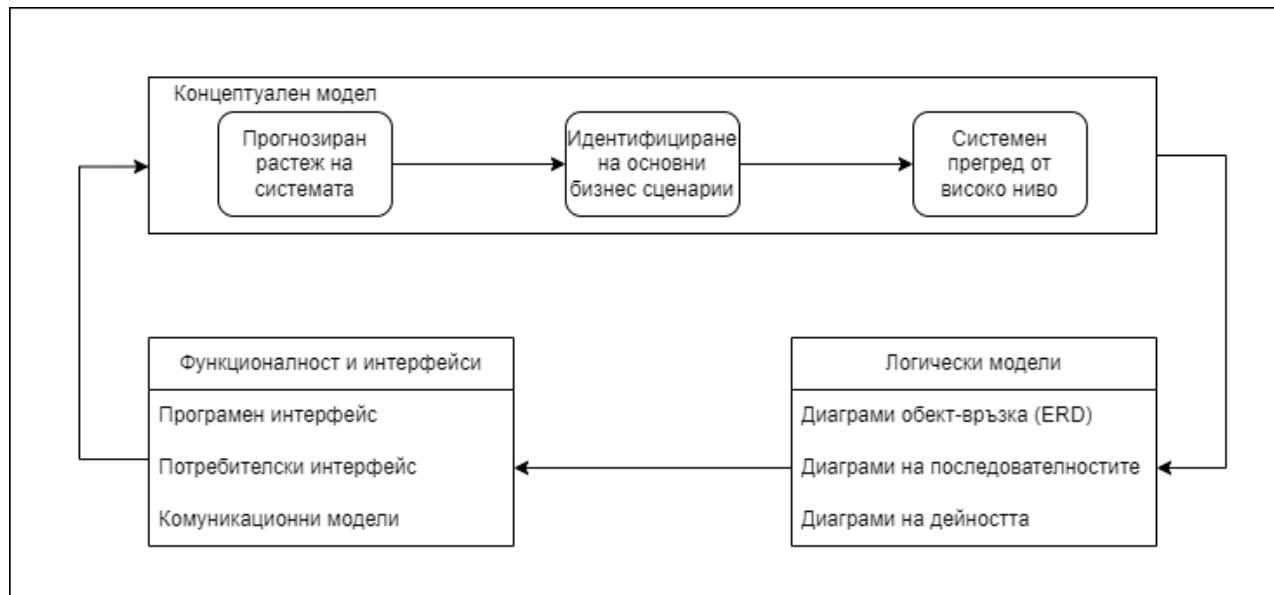
Глава 2. Архитектура на облачна система за управление на поръчки от клиенти

2.1. Концептуален модел на облачната система за управление на поръчките

Концептуалният модел следва да бъде разгледан като решение от високо ниво, което се съсредоточава върху всички основни потребителски, бизнес и ИТ изисквания (Стоев, 2018). Съществена част от тази глава са градивните елементи и интерфейси, изграждащи системата, както и комуникационните модели, които свързват облачните услуги с мобилните и уеб клиенти (Nacheva et al., 2022). 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, базирана на микроуслуги, работеща върху множество процеси и сървъри (хостове). Всяка услуга се изпълнява в отделен процес като контейнер, разположен в кълстер от виртуални машини (Nguyen et al., 2019). Целта на разделянето на системата на подсистеми, индивидуални нива и компоненти е да се подобри разбираемостта и да се улесни поддръжката, с крайната цел да се намали оперативната сложност.

Проектиране на концептуален модел на бизнес система е възможно чрез използване на итеративен процес (Ingeno, 2018). Изобразен на фиг. 2.1, този итеративен процес се състои от циклична поредица от етапи за разработване и усъвършенстване на софтуерни архитектури. Диаграмата представя структурирана рамка, която води началото си с прогнозиран системен растеж, свързан с основните бизнес сценарии. „Системния преглед от високо ниво“ дава възможност за преход към „логическите модели“

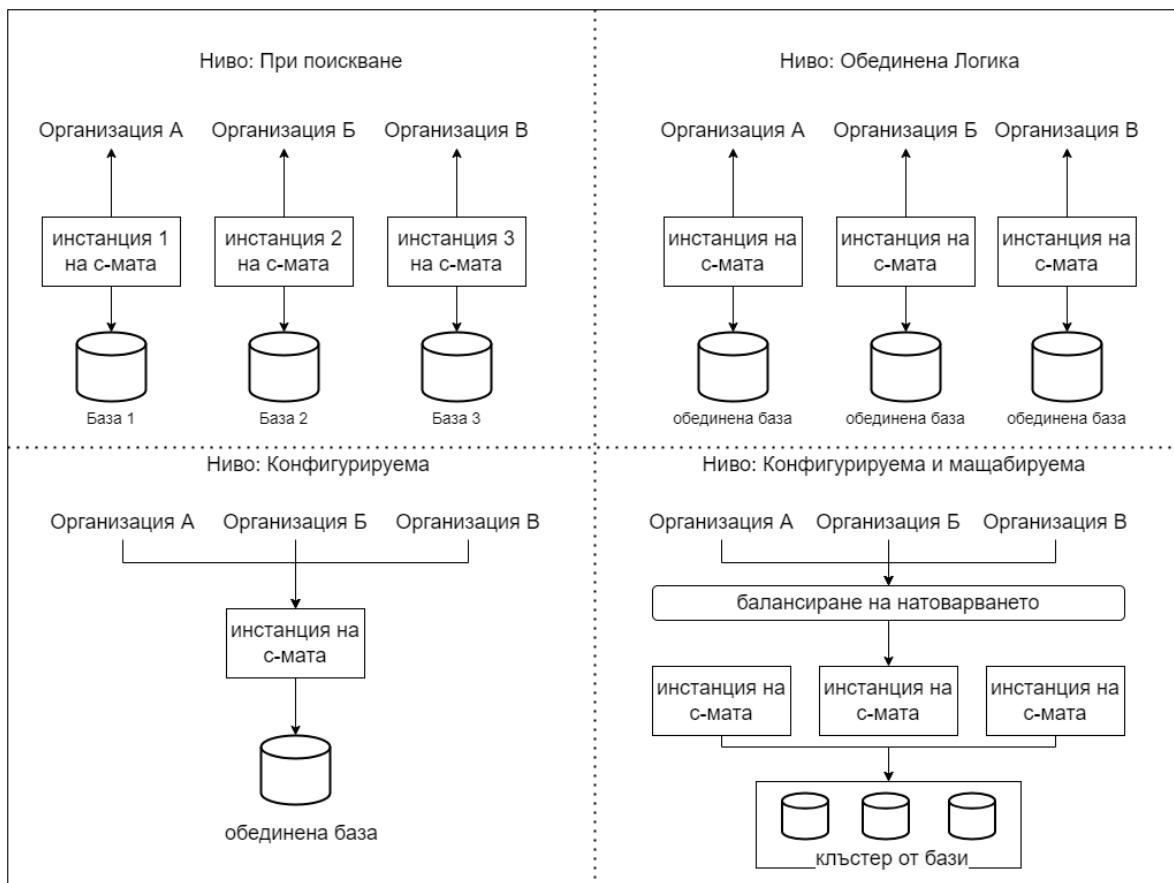
включващи диаграми на същност-връзка (ERD), потребителски последователности и дейности, установявайки базово разбиране на системата от гледна точка на краен потребител на производено предприятие. В тази връзка, втората част на диаграмата представя аспекти на взаимодействието с потребителя, включвайки „Функционалност и интерфейси“, които следва да дадът детайлно представяне за очаваните резултати и възможности на системата.



Фиг. 2.1. Итеративен процес за проектиране на концептуален модел (разработка на автора по Ingino)

Първият етап в изграждането на архитектурата на облачна система за управление на поръчки включва **прогнозиран растеж на системата**⁹. По отношение на „модела на зрялост“ на архитектурата, Stuckenberg (2014) категоризира фазите по прогнозиране на няколко нива. В съответствие с основните цели, фиг. 2.2. представя адаптиран от автора модел, който определя 4 възможни нива на архитектурата на облачната система.

⁹ Прогнозиран растеж се отнася до оценката или предвиждането за бъдещото развитие на дадена система. Тази прогноза може да включва различни аспекти като повишаване на производителността, технологични подобрения и др. Прогнозирането се базира на текущи данни, тенденции, анализ на външни и вътрешни фактори.



Фиг. 2.2. Модел на зрялост на архитектурата (разработка на автора по Stuckenberg, 2014)

В първото ниво, наречено „при поискване“, всяка организационна единица¹⁰ използва специална версия на информационната система, която да бъде пригодена на уникалните изисквания на текущата организационна единица на производственото предприятие. Във второто ниво на „обединена логика“ програмният код се стандартизира, като всяка организационна единица използва идентични копия на софтуера, но адаптирани на специален хардуер.

В тези два случая, персонализирането е ограничено до предварително дефинирани функционалности. За преодоляване на този проблем, третото и четвъртото ниво предоставят по-висока конфигурация, като на тези нива всеки

¹⁰ Терминът "организационна единица" може да бъде дефинирана като подразделение или структурен компонент в рамките на една по-голяма корпорация, като например холдинг. В контекста на модула за Продажби и Дистрибуция на SAP се дефинира като "Търговска организация".

клиент използва едно и също копие на един и същ хардуер. По този начин се дава възможност за постоянни обновления и реализиране на нови версии на системната. В тази връзка, четвъртото ниво е допълнително подобрено от балансъор на натоварването и кълстер от бази от данни.

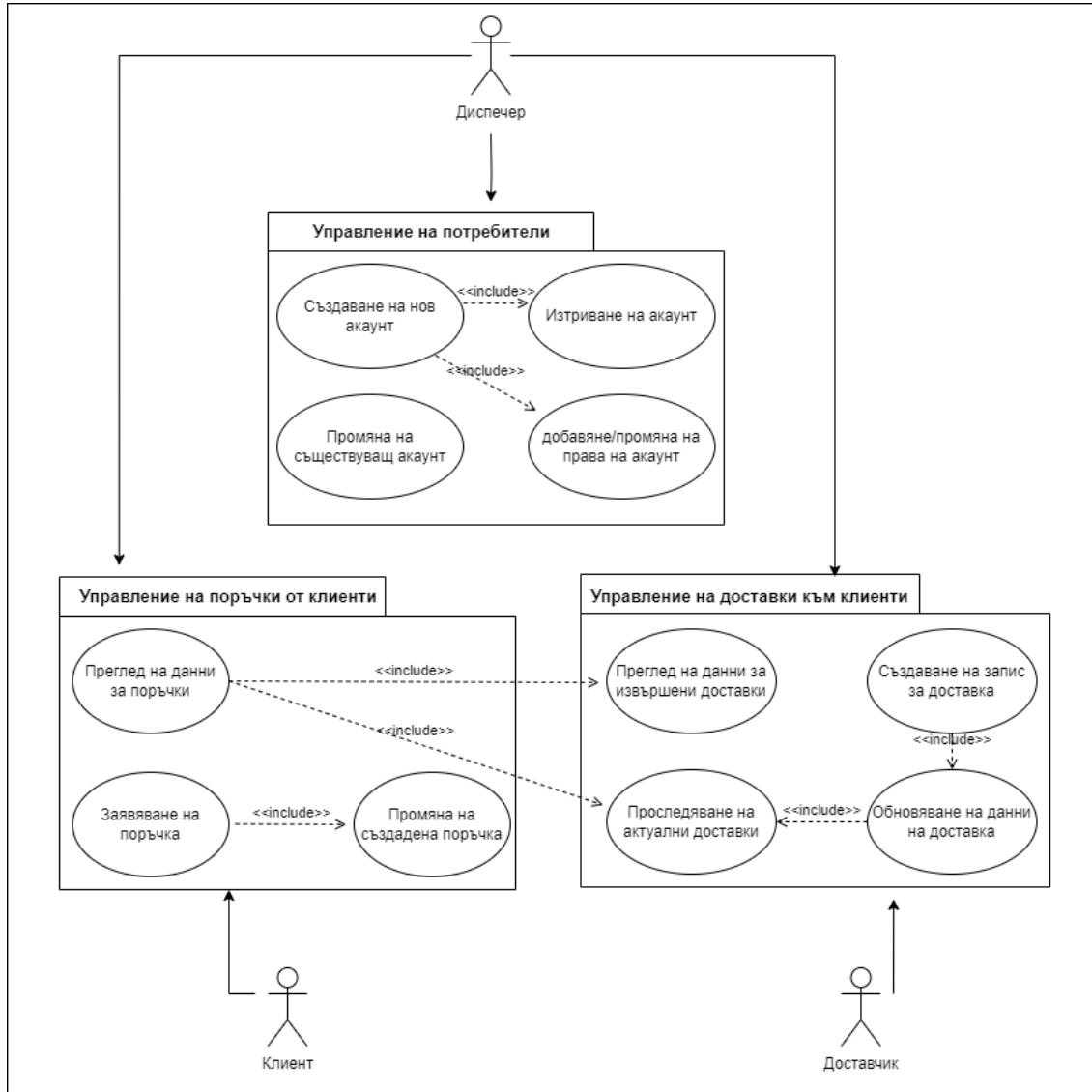
Считаме че включването на четвъртото ниво на конфигурируемост и мащабируемост в архитектурата е в съответствие с целта за подобряване на ефикасността на механизмите за обработка на поръчките, оптимизиране използването на ресурси и поддържането на ниски оперативните разходи. Множество клиенти следва да 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 фактори като повишен или понижен потребителски трафик, обеми на поръчки, разнообразни продуктови асортименти и променящи се предпочитания на клиентите.

За ефективно интегриране на базирана в облака система за управление на поръчки в оперативната рамка на производствена компания е необходимо да се проучат различните бизнес изисквания и да се идентифицират различни сценарии за използване на информационната система. В този смисъл, следващият етап в развитието на концептуалния модел включва очертаването на бизнес сценарии за конструиране на основните операции (Sulova, 2023).

Бизнес сценарийте описват основните възможности на системата, включвайки „Управление 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 сценарий 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 (Parusheva & Pencheva, 2022). 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 диспичера, 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 време, изискава 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, 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 подсистеми като ERP, IoT и т.н. Като допълнение към тези функционалности, системата следва да поддържа 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 се свързва с последователната работа на компонентите на системата, 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 брой уеб заявки. Тези характеристики са от основно значение за базираните в облак системи, за да помогнат на производствените предприятия да постигнат прозрачност във всеки етап от операциите по управление на поръчките от клиенти. В допълнение, мащабируемостта и производителността на системата влияят пряко върху способността ѝ да се адаптира към променящото се търсене и обработка на поръчки.

Нефункционалните изисквания често се наричат „атрибути за качество“ на системата (Toub, 2024). В този смисъл, следващите точки посочват някои от тях:

- Системата трябва да притежава високо ниво на достъпност, което да

й позволява независимо да управлява разпределението на облачните изчислителни ресурси в съответствие с увеличаване на потребителския трафик. Освен това, след намаляване на трафика, използването на облачни ресурси следва автоматично да се върне към предходните нива;

- 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;

Различните изисквания, бизнес сценарии и архитектурни модели са фундаментални за разработването на цялостно решение. В този контекст, „концептуалният модел“ на системата предоставя рамка, като етапът на „преглед от високо ниво“ прави заключение и подчертава значението на прототип, който следва да бъде разгледан в детайли и апробиран в производствено предприятие. Този прототип адресира представените по-горе проблеми и отговаря на предварително установените цели.

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 услуги. Клиентите на фирмата, които се явяват крайните потребители, управляват и проследяват поръчките и доставките в реално време с мобилно приложение. Целта му е да помога 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 разпростири чрез Google

Play Store и Apple App Store.

Фигура 2.4 представя концепцията, която показва как са структурирани приложенията в информационната система за управление на поръчки от клиенти.



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

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 закъсняят, 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.

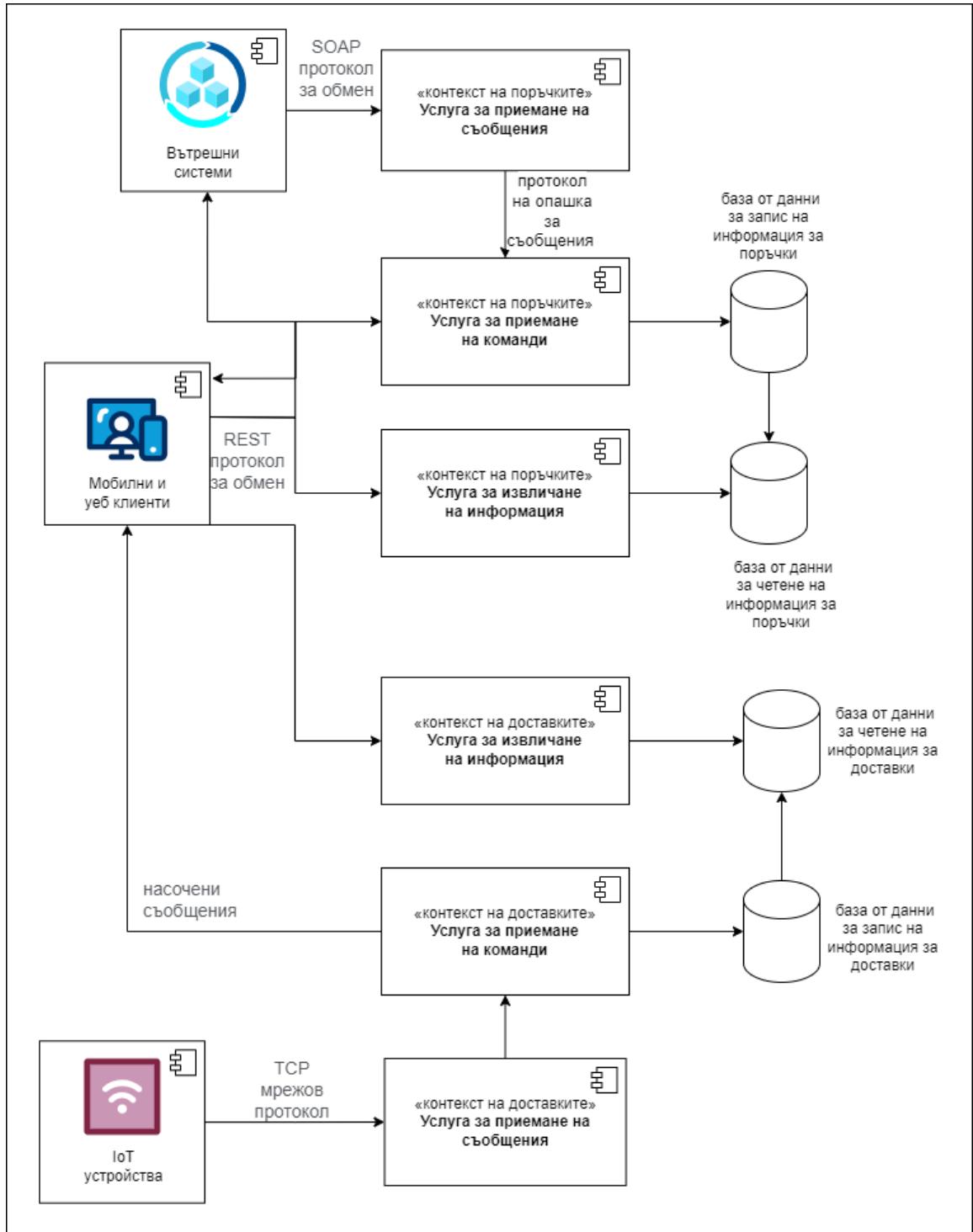
2.2. Логически модел на облачна система за управление на поръчки

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 анализи (Hartley & Sawaya, 2019), установихме, че подпрограмите за управление на поръчки и доставки са взаимосвързани. Следователно е необходимо тези модули да се разглеждат заедно, като следва да отговарят на принципите и методите на архитектурния дизайн, описани в първа 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 IoT (Armiyanova, 2019), мобилни и уеб приложения (UI), както и вътрешни системи (ERP,SCM).



Фиг. 2.5. UML Диаграма на компонентите представляща 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.

На фигурата е показан ОДД архитектурен дизайн, прилагайки концепцията за „ограничен контекст“, дефинирайки две „зоni“ 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 извлечане 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 информация по асинхронен начин от услугите за съобщения. От друга страна, услугите за извлечане на информация отправят запитвания към базата от данни, без да променят състоянието. Основен компонент е механизъмът 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 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 извлечане на записите. Всяка база данни е проектирана да отговаря на специфичните оперативни нужди на съответния компонент. Това разграничение подчертава използването на модела 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 от данни 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 разбиранния за поведението на потребителите и позволявайки детайлена одит на всяко действие на клиента.

Предлагаме, схемата на базите от данни да обхваща две таблици: потоци и събития. Потоците служат като основа за организиране и категоризиране на събития. Таблицата 2.1. 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 извлечане 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, позволяйки вземането на информирани решения. В таблицата 2.2. е описана структура на модела на „Събития“.

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

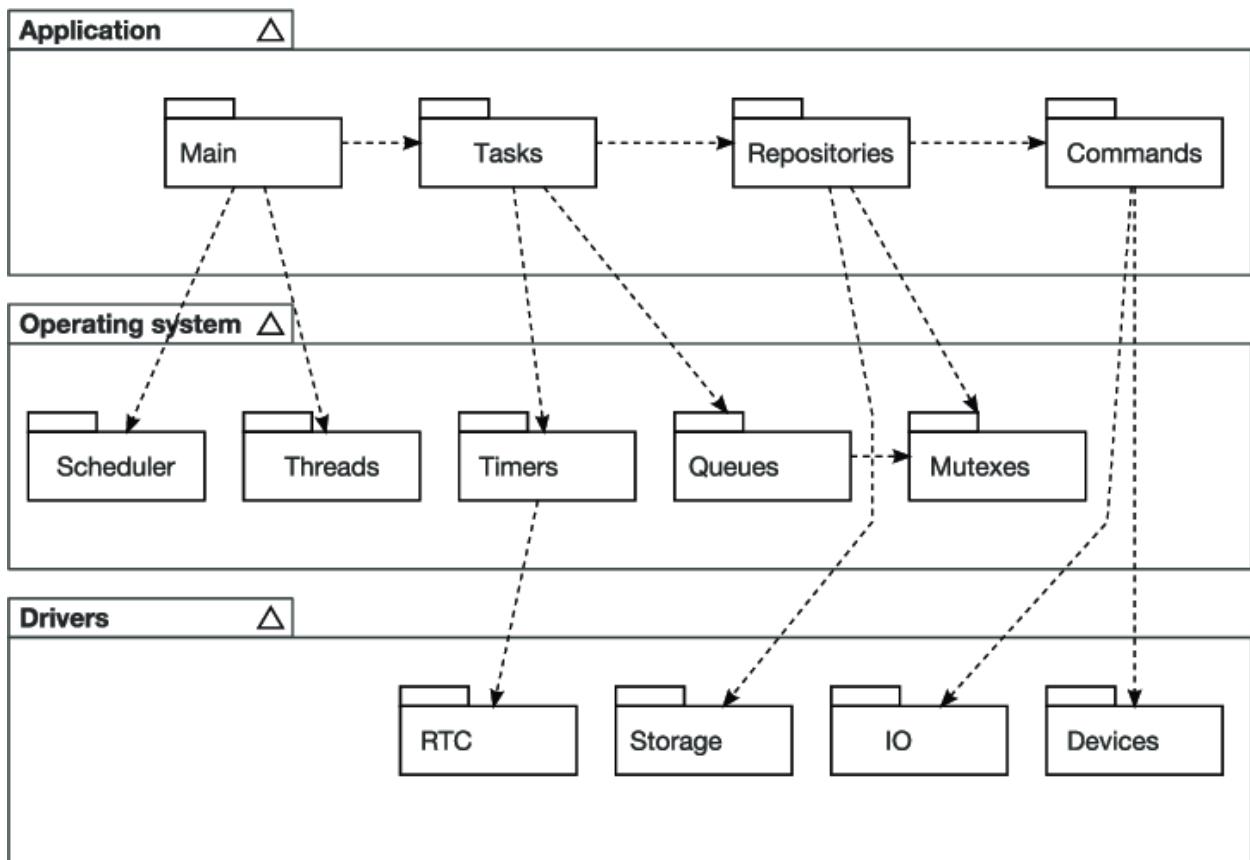
Поле	Описание
Идентификатор (id)	Универсален уникален идентификатор, който представлява първичния ключ за всяко събитие.
Идентификатор на потока (stream_id)	Свързва събитие със съответния поток, установявайки връзка с таблицата с потоци.
Идентификатор на последователността (seq_id)	Последователен идентификатор, 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 извлечане. 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 ОДД, 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 представления на данни, които са често използвани (Villaça et al., 2018). Изгледите са персонализирани, за да изпълнят специфични нужди, като например извлечане 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 подобреие 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.

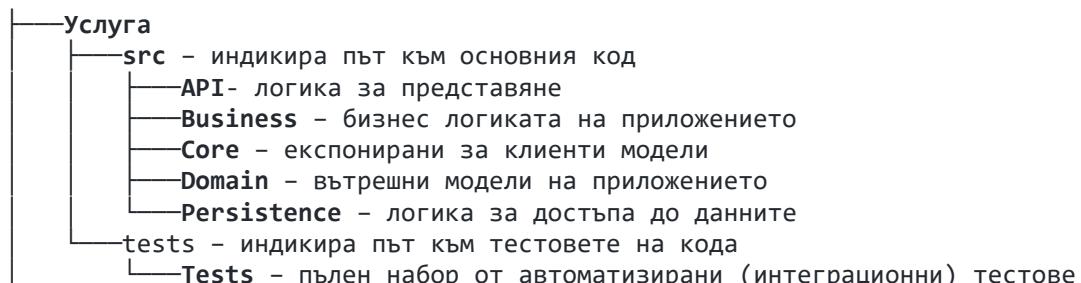
2.2.2. Детайлизиране на модулите за поръчки и доставки

След установената обща системна рамка, следва да се проучат оперативните процедури и адаптирани шаблони в микроуслугите. Както бе споменато в предходната глава, компонент на ОДД 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 юерархията, предоставящи API между данните и продуктите. Използването на UML диаграма на пакетите създава това юерархично 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 „изграждане“ 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:



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

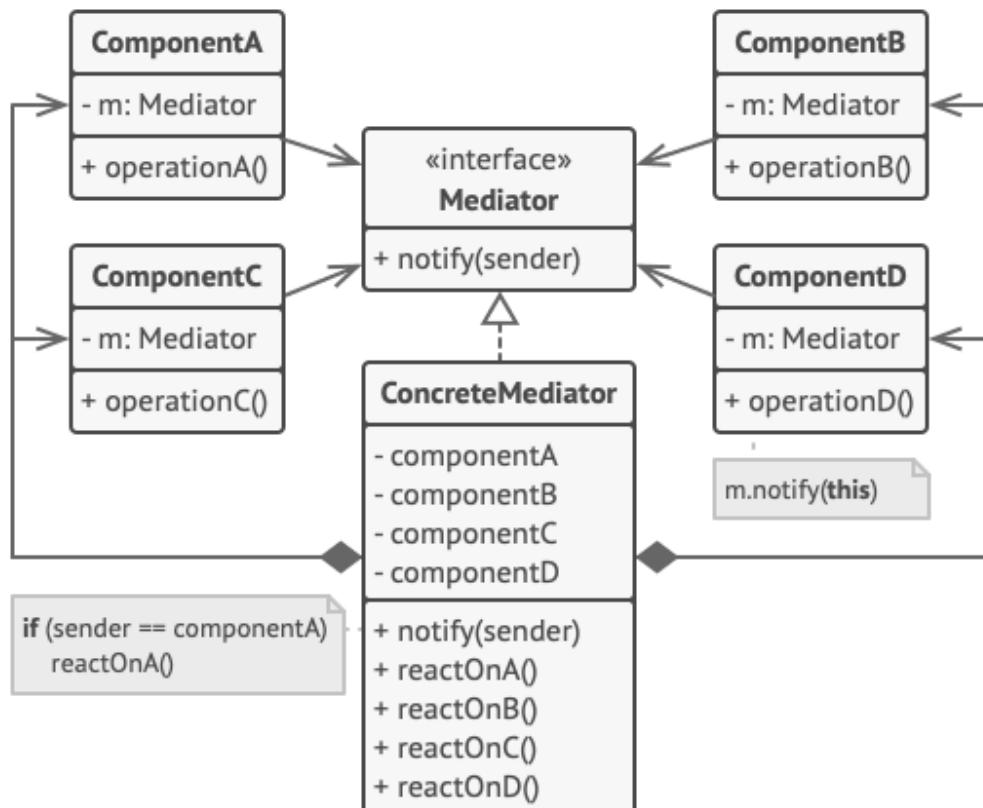
Src действа като централен посредник, координиращ взаимодействията между няколко слоя, представени от различни пакети, следвайки принципите на чистата архитектура:

- 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, наследявайки възможността за повторна употреба и поддръжка;
- Бизнес (Business)- съдържа бизнес логика, организираща основните функции на приложението. Също така поддържа манипуляторите 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 поддиректорията ‘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 наследяване 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 „манипулятор на съобщения“. Тези манипулятори 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.

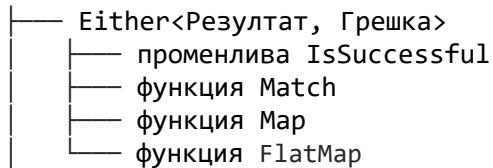
В тази връзка, някои автори (Parusheva & Pencheva, 2022) посочват, че UML диаграми на класовете са инструмент, използван за представяне на обектите, съответните им свойства и взаимовръзки. Този тип диаграми идентифицират речника на системата и се срещат във разработката на софтуерни модули. От гледна точка на системата за управление на поръчки, същностите, които ще изграждат дотук описания модел, са следните (вж. фиг. 2.8):



Фиг. 2.8. Диаграма на медиатор 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 наследчава 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. Общата структура на „Either“ монад е установена, както следва:



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

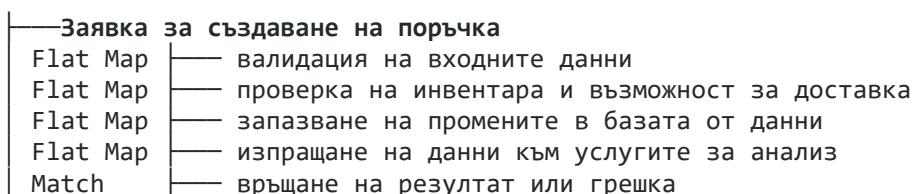
Съществена част от структурата е състоянието на булевата променливата „IsSuccessful“ и поведението на функцията „Map“. Когато даден „Either“ монад, като тип $C<T>$, има стойност, представляваща успешен резултат, функцията “Map” прилага трансформация по следния начин:

$$(C<T>, (T \Rightarrow T2)) \Rightarrow C<T2>$$

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

$(C<T>, (T \Rightarrow C<T2>)) \Rightarrow C<T2>$

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:



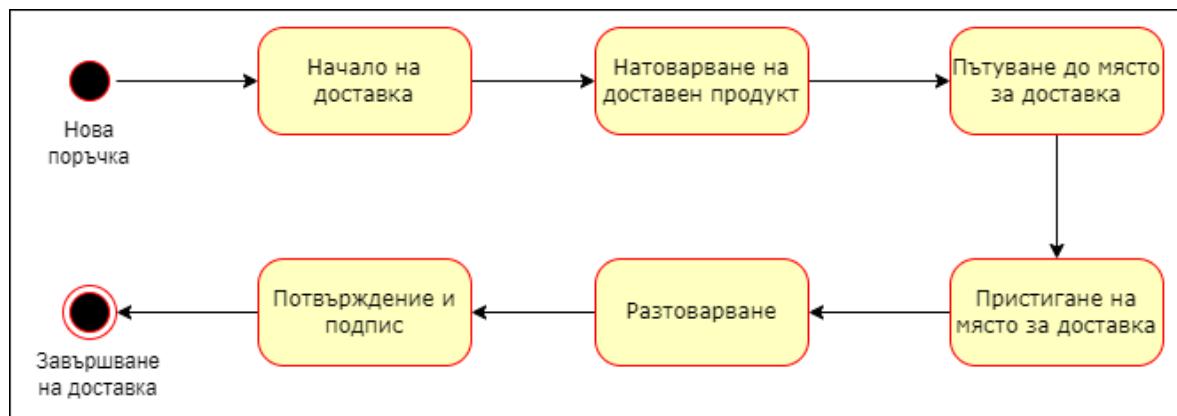
Фиг. 2.10. Примерен код за създаване на поръчка чрез използването на „Either“ монад. (разработка на автора)

Използването на монадата „Either“¹¹ 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 манипуляторите”.

Според някои автори (Hofmann et al., 2019) 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.

¹¹ Either е концепция за обработка на грешки, като представлява тип данни с две възможности, обикновено наричани Left и Right. Традиционно, Left се използва за представяне на грешка или неуспешен резултат, докато Right представлява успешен резултат от операция.

В тази връзка, фиг. 2.11. представя етапите от процеса на управление на поръчките, установен чрез литературния анализ от първа глава.



Фиг. 2.11. Диаграма на процес на логистично управление свързано с активност на поръчка. (разработка на автора)

Диаграмата очертава процес на логистично управление, подчертавайки обработката на поръчка и планирането на доставка (Schachenhofer et al., 2023). Процесът започва със „създаване на нова поръчка“, последван от „възлагане на доставчик“ или „начало на процеса по доставка“, което показва динамичното разпределение 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“. Фактически, 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. Диаграмата представлява опростен модел на процеса, като някои от етапите като управление на рекламации и комуникация с клиента са пропуснати.

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 (Димитров, П, 2018). 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 самостоятелно отговарят 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. Счита се, че централизираното управление опростява спазването на регуляторните изисквания, като предоставят единна рамка за спазване на стандарти като General Data Protection Regulation (Kesan et al., 2013; Василев, 2021).

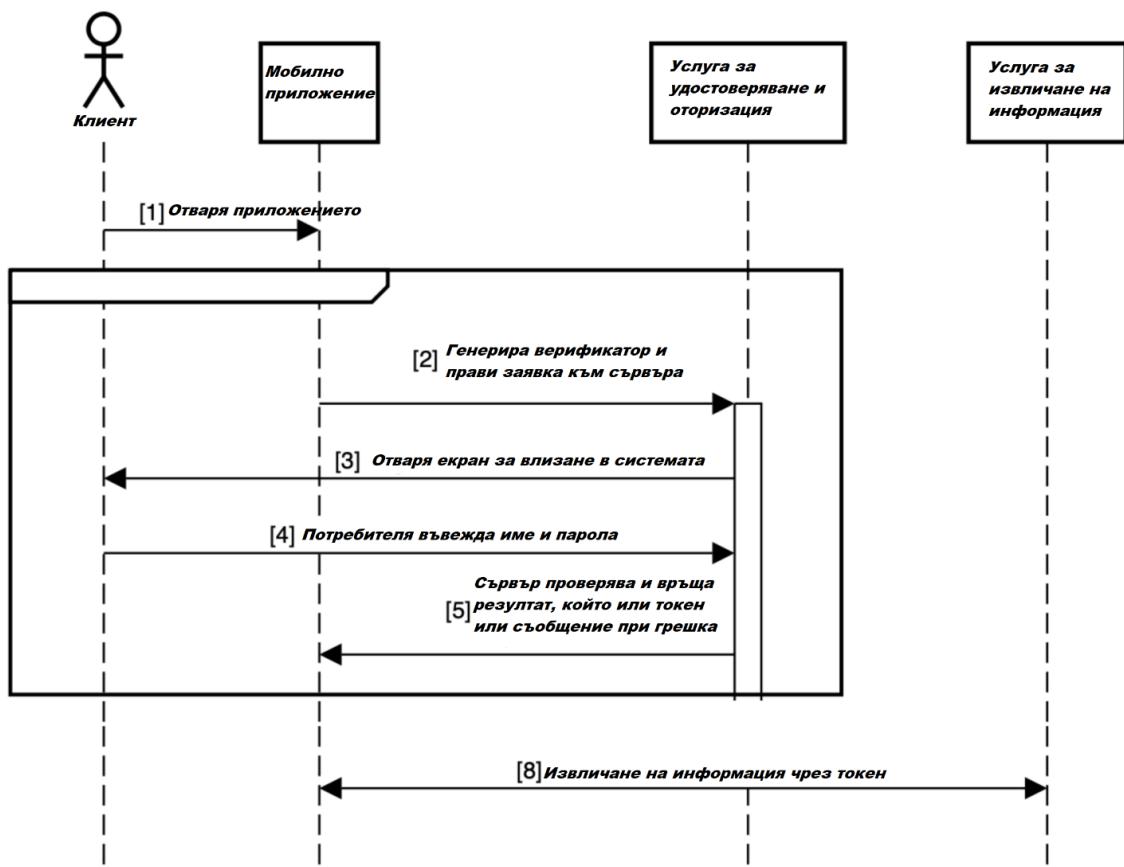
Централизираният сървър за самоличност внедрява OAuth и OIDC, въвеждайки стандартизиран механизъм за управление на потребителския достъп, удостоверяване, издаване и разчитане на токени за проверка на самоличността.

*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.

Вид	Случай на употреба	Описание	Съображение за сигурност
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 атаки за прихващане на код за оторизация, Man-in-the-Middle и други.

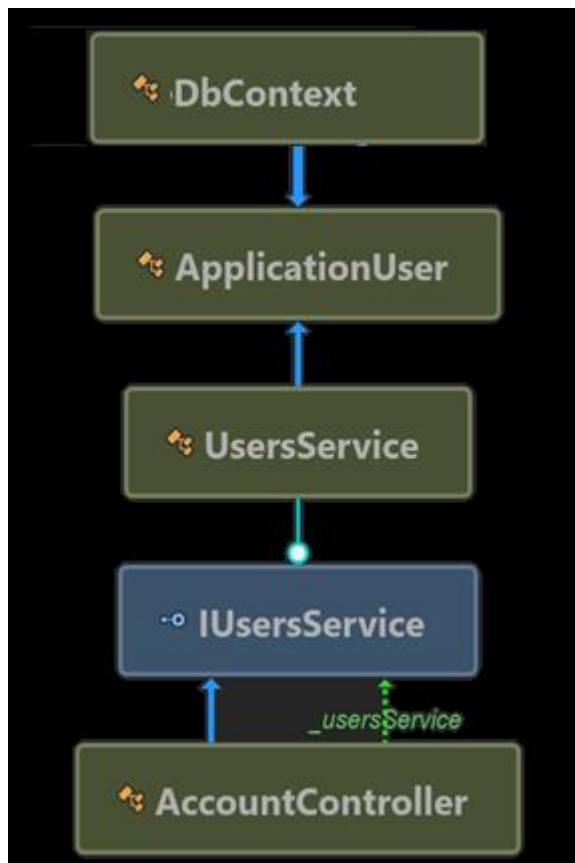
От представената таблица, PKCE се отличава като подходяща техника за удостоверяване, работеща с верификатор и код за оторизация за да достъпи крайната точка за получаване на токен за достъп, осигурявайки сигурна потребителска сесия. Процедурните стъпки, които информационната система следва да въведе, за да осъществи удостоверяването, са представени на фиг.2.12.



Фиг. 2.12. 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. 2.13. **DbContext** и **ApplicationUser** представляват комбинация от класове, които оперират с базата от данни. **AccountController** използва тези свойства чрез **UserService**, който капсулирана логиката, по безопасен за използване начин, и също така отговаря за визуализацията на потребителския интерфейс, чрез генериране на HTML.



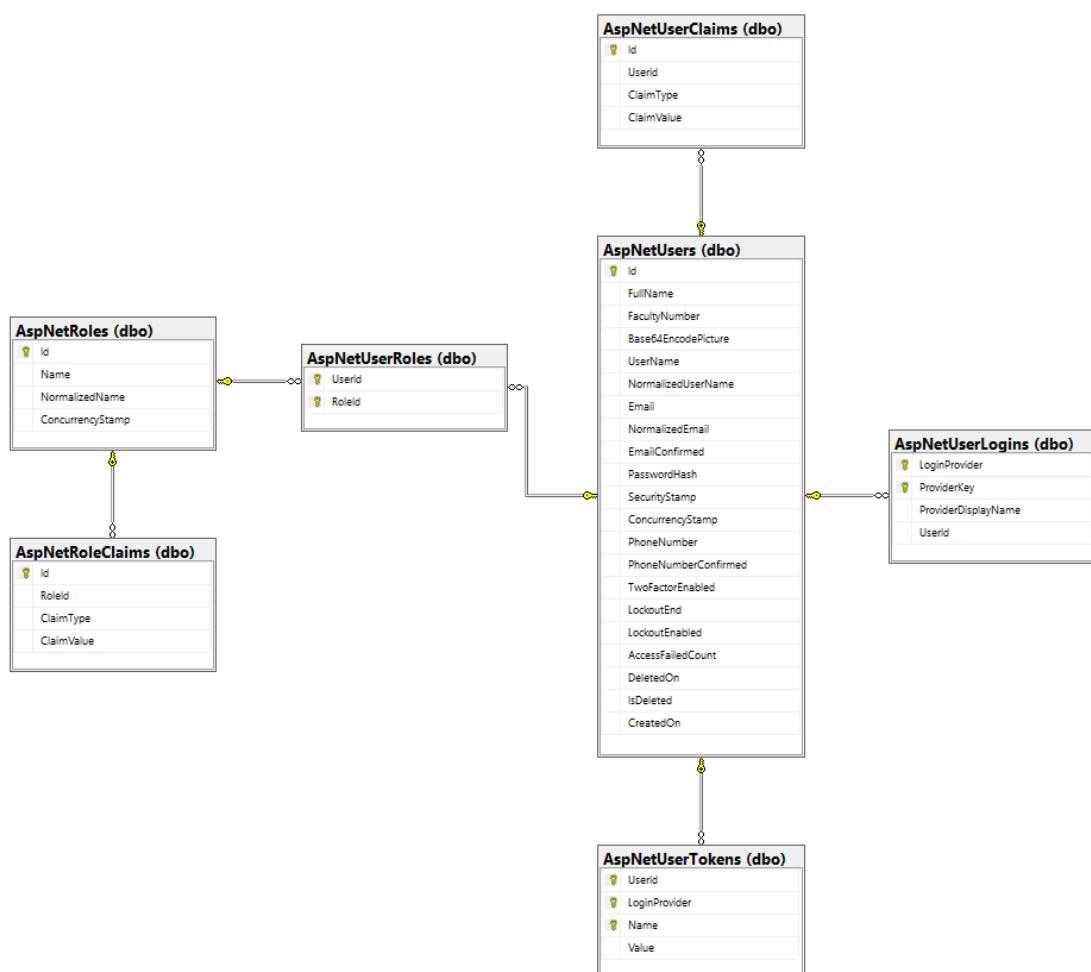
Фиг. 2.13. Структура на класовете, отговарящи за удостоверяване (разработка на автора)

Базата данни, която обслужва услугата, е базирана на структурирана схема, включваща следните таблици:

- AspNetUsers: Основната таблица, съдържаща информация за потребителите, включително имена, имейли и данни за сигурност и идентификация. Таблицата поддържа атрибути за потвърждаване на имайл и телефонен номер, включване на двуфакторна аутентификация и управление на заключване при неуспешен достъп;
- AspNetRoles: Таблицата дефинира ролите, които могат да бъдат придадени на потребители, позволявайки разграничение на потребителски права и привилегии в рамките на приложението.
- AspNetUserRoles: Служи като свързваща таблица между 'AspNetUsers' и 'AspNetRoles', позволяваща многозначна връзка, където един потребител може да има множество роли.

- AspNetUserClaims и AspNetRoleClaims: Тези таблици обработват claims (твърдения) за потребители и роли, които предоставят финно зърнест контрол над идентитета, като например потребителски разрешения или настройки на конфигурация, които не са пряко включени в основните таблици за роли и потребители.
- AspNetUserLogins и AspNetUserTokens: Таблиците управляват външни аутентификационни провайдери, като например Google или Facebook, както и токени за аутентикация, което е от съществено значение за интеграцията със съвременни аутентификационни схеми и API достъп.

Фигура 2.15 представя диаграма на споменатите по-горе таблици.



Фиг. 2.14. Диаграма на базата от данни за потребителите. (разработка на автора по)

От значение 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 (Britch, 2024), 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.

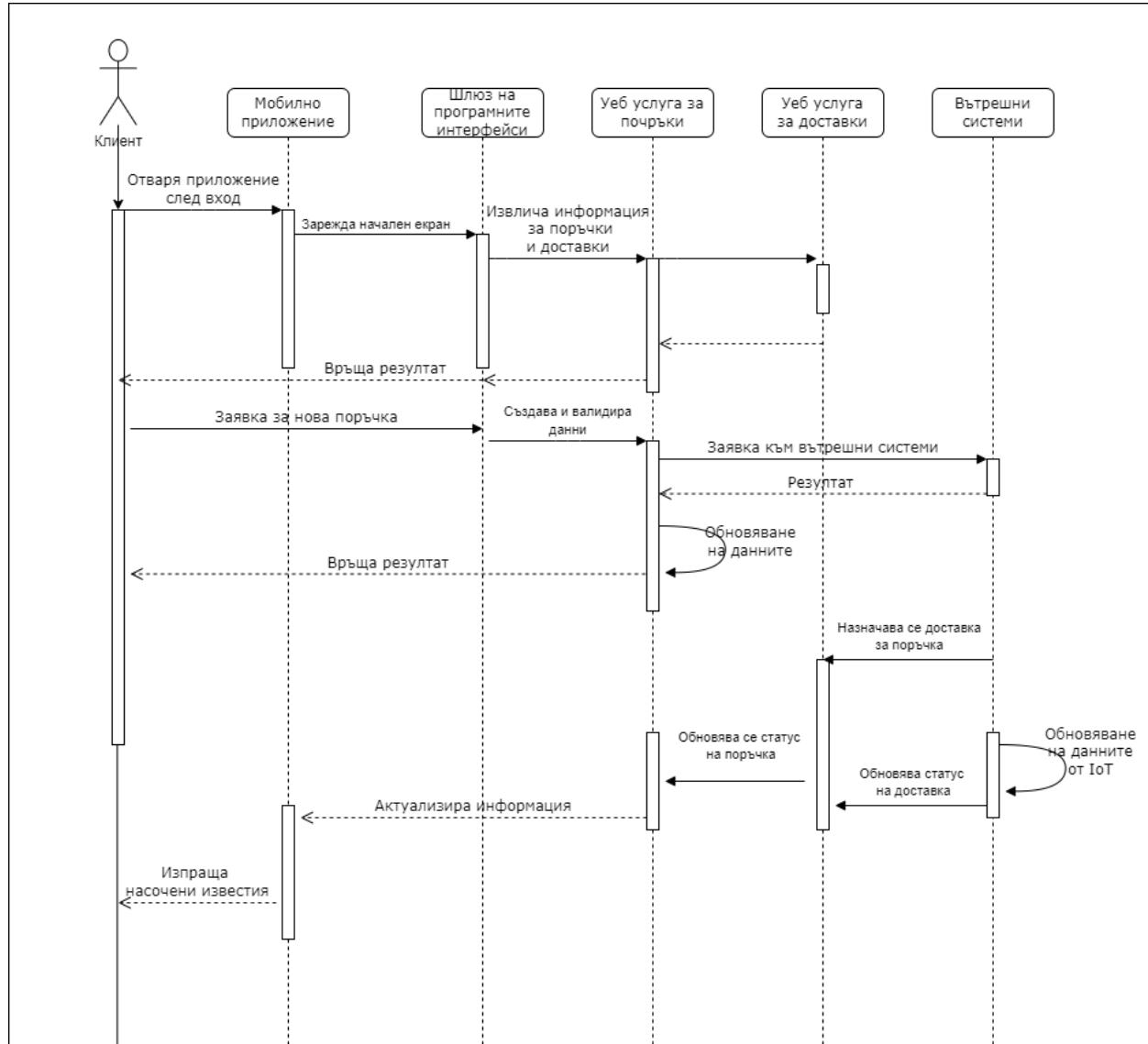
2.3. Комуникационни модели между модулите

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 (Huang et al., 2013). 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 ОДД, тази диаграма илюстрира последователната от събития, които се случват в отговор на определена 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. 2.15. 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.

За да се свърже клиентските приложения към сървърната част, API Gateway използва 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 ОДД 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>.

За да работи с протокола HTTP 1.1, API Gateway следва набор от методи, описани в таблица 2.4, които имат различни значения:

*Таблица 2.4.
Методи на протокола HTTP 1.1. използвани в API Gateway
(разработка на автора по HTTP RFC 2616)*

Метод	Описание
GET	Най-разпространеният метод. Използва се за извлечане 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 (Stonis, 2024). Те осигуряват различна семантика, когато се прилагат към различните ресурси, като се спазват стандартите на RESTful реализации.

*Таблица 2.5.
Крайни точки на API Gateway и техните REST ресурси.
(разработка на автора)*

Крайна точка	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 МИМЕ). 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 поръчка, ще върне следния отговор във формат JSON:

```
{
  "orderId": "eu.123123.231",
  "orderValue": 99.90,
  "orderStatus": "complete",
```

```

    "deliveryId": "3321",
    "deliveryStatus": "complete",
    "deliveryTime": "2h"
}

```

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

*Таблица 2.6.
Предварително зададени HTTP кодове на API Gateway
(разработка на автора)*

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 разгранишим два основни типа комуникация, които се използват между компонентите: синхронна и асинхронна. В синхронния 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:

```
GET
Host: tools.ietf.org
User-Agent: Mozilla/5.0 (Ubuntu; X11; Linux x86_64; rv:9.0.1) Gecko/20100101
           Firefox/9.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
If-Modified-Since: Sun, 13 Nov 2011 21:13:51 GMT
If-None-Match: "182a7f0-2aac0-4b1a43b17a1c0;4b6bc4bba3192"
Cache-Control: max-age=0

HTTP/1.1 304 Not Modified
Date: Tue, 17 Jan 2012 17:02:44 GMT
Server: Apache/2.2.21 (Debian)
Connection: Keep-Alive
Keep-Alive: timeout=5, max=99
Etag: "182a7f0-2aac0-4b1a43b17a1c0;4b6bc4bba3192"
Content-Location:
Vary: negotiate,Accept-Encoding
```

Фиг. 2.17 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 извлечане 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. 2.18. 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 сурс кода. В тази връзка следната таблица сравнява REST и gRPC, като видове на синхронна комуникация.

*Таблица 2.7.
Таблица, описваща разликите между REST и gRPC
(разработка на автора)*

	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 значение за съхранението в услугите, тъй като тези компоненти запазват текущите състояния на приложението. Основно изискване за последователността, е нито една услуга да включва хранилища за данни от друга, както и да не разчита на директни заявки.

2.4. Функционалност и потребителски интерфейс

Интегрирането на облачните технологии в различни аспекти на SCM, се базира на цялостна екосистема, предназначена да рационализира процесите. Чрез мобилното приложение, клиентите на производственото предприятие създават поръчки и проследяват доставките. Скица на интерфейса на началния еcran, 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 еcran в главното меню.



Фиг. 2.19 Скица на основен еcran 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 с по-различен графичен интерфейс. Водачът може да провери списъка с предстоящи доставки, назначени към него.

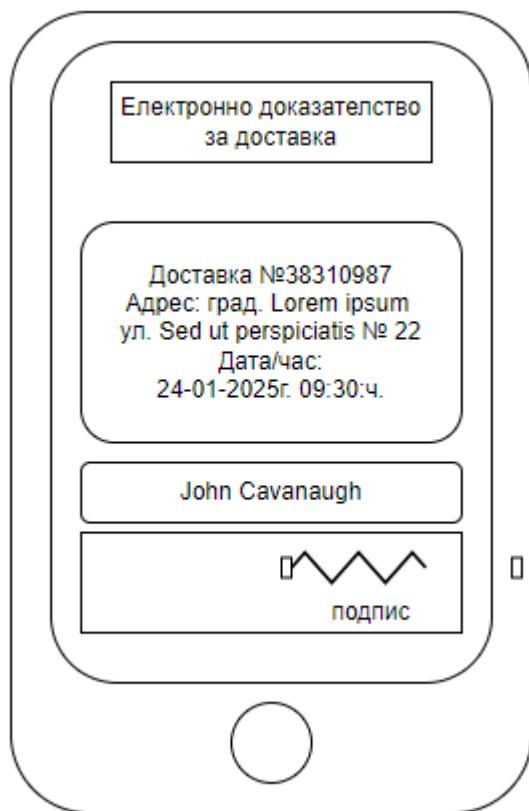


Фиг. 2.20. Скица на еcran за доставчика. (разработка на автора)

Посоченият еcran съдържа подробности за доставката, включващ 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 подпись 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 подпись на мобилното устройство, след което електронният документа се препраща към ERP.

Следната фигура представя скица на экрана за тази функционалност.



*Фиг. 2.22. Скица на екран за доказателство за доставка (ePOD).
(разработка на автора)*

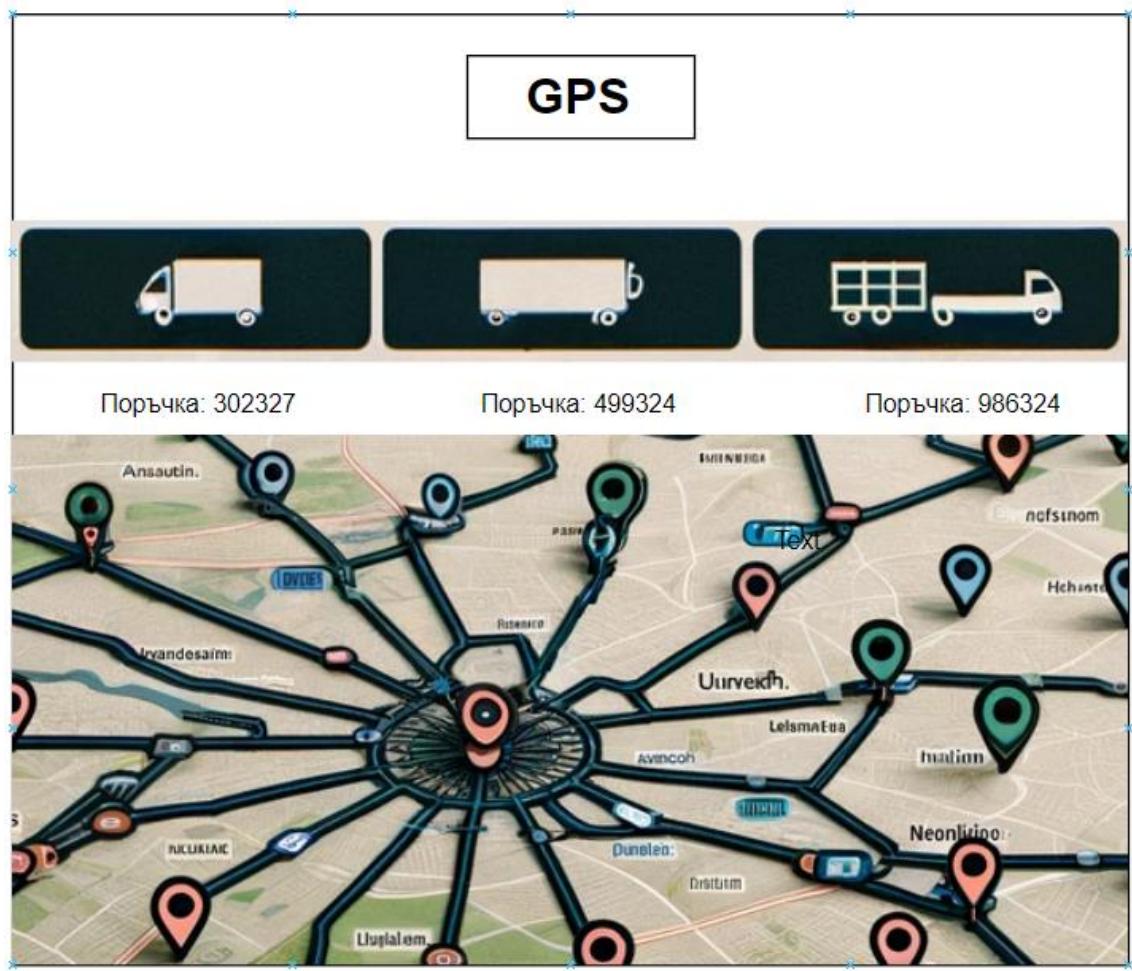
Графичен интерфейс на уеб портала, представен на фиг.2.13, е предназначен за използване от диспечерите за планиране и разпределение на логистичните задачи. Той представя информация за поръчките, които трябва да бъдат доставени, като също така дава пълен контрол върху всички превозни средства. Целта към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



Фиг. 2.23. Главен екран в уеб портала. (разработка на автора)

Главният екран в уеб портала 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 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. 2.24. Екран за маршрутизиране. (разработка на автора)

В заключение, 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 представлят 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, с портфолио от различни продукти, включващо цимент, агрегати, бетон и други специализирани строителни материали. Разположен в град Девня, област Варна, "Хейделберг Цимент Девня" АД в експлоатация от 4 декември 1958 г. Предприятието е част от немска мултинационална компания Heidelberg Materials, основана през 1874 г. С дейност в повече от 50 държави и разполагайки с над 3,000 производствени обекти по света, Heidelberg Materials е основен участник в глобалната индустрия за строителни материали. Поради мащаба на дейността, компанията се стреми постоянно да подобрява своите продукти и процеси. Въвеждането на облачни информационни системи в производството би било част от технологичните иновации, които Heidelberg Materials представя в своята стратегия.

В тази връзка, основната дейност на "Хейделберг Цимент Девня" АД 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 обстоятелството, че тези смеси се произвеждат според спецификациите на клиента, което води до висококачествен продукт, който може да се използва веднага след доставката. За дозиране на бетон, компанията използва големи централни съоръжения. "Хейделберг Цимент Девня" АД 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 (Delnavaz et al., 2022). 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.

В компанията, ефективността и ефикасността на продажбите и операциите по доставка са пряко зависими от ясно дефинираните роли и отговорности на служителите, както и от адаптивността на организационната структура спрямо потребностите на клиентите. Оптимизацията на екипната работа и комуникацията с клиентите, както и вътрешната в рамките на фирмата, е от основно значение за поддържане на производителността и намаляване на времето за доставка.

Разглеждайки структурата и организацията на екипите, на чело стои борд на директорите, който в контекста на търговията с бетон, управлява мрежа от офис диспечери, складови оператори, доставчици, търговски представители и други. Всички те са взаимно свързани, като успехът на тази структура се крие в добре координираната работа.

В процеса на изследване на оперативните процедури в рамките на организацията, е от значение да се разгледа поетапното обработване на клиентски заявки и последващото извършване на логистични дейности за доставка на бетон. При приемането на поръчка от клиент, диспечер въвежда заявката в ERP системата с начален статус "не потвърдена". Този механизъм осигурява възможност за предварително планиране и координация на ресурсите. Следващ етап в този процес е действието на диспечера ден преди

назначената дата за доставка, който следва да се свърже с клиента за потвърждаване на поръчката, променяйки статуса ѝ в "потвърдена" и по този начин назначавайки продукта за доставка. В случай че клиентът откаже доставката, поръчката преминава в статус "отхвърлена", което води до процедури за коригиране на плана за доставки и адаптиране на логистичната верига спрямо нововъзникналите обстоятелства (Văcar, 2019). Понякога, тези случаи водят до пропуски във транспорта и правилните средства и материали, необходими за предстоящите доставки.

След като поръчката бъде потвърдена, диспачерът започва активна комуникация с шофьорите, като следи отблизо за евентуални промени. Неочаквани проблеми като трафик, задръствания или внезапни промени, направени от клиента, се нуждаят от незабавно внимание и коригиране от страна на диспачера. В този контекст, регулярното обновяване на статуса на поръчките, както и предоставянето на текущата информация за очакваното време на доставка са от значение за удовлетвореността на клиента и вътрешното управление на ресурсите.

Последният етап на логистичния процес е доставката на бетона на мястото, указано от клиента, и получаването на потвърждение за приемането му. Необходимо е да се подпишат документи на хартиен носител, като доказателство, че стоките са доставени в оптимално състояние и в уговорения часови диапазон. При успешна доставка, поръчката следва да бъде затворена ръчно в ERP системата, за да може да бъде генерирана фактура за извършване на плащане.

Този преглед дава представа за сложния логистичен процес, свързан с обработката и изпълнението на заявки от клиенти. Той подчертава значението на всеки етап, както и своевременната комуникация в динамична среда, които отговарят на дефинираните от нас процеси. За основен недостатък на обслужването, считаме фактът, че диспачерите ръчно отбелязват на текущото състояние и ниското ниво на автоматизация. В дейността на фирмата се наблюдават някои проблемни области и направления, в които биха могли да

бъдат внесени множество подобрения, които да окажат положителен ефект върху подобряването на цялостната верига.

Предложената от нас облачна информационна система за управление на поръчките от клиенти може да бъде разработена и внедрена в дейността на "Хайделберг Цимент Девня" АД, като по този начин предвиждаме да спомогне за решаване на изложените проблеми. Предлагаме специфични решения, които информационна система осигурява, да бъдат разгледани в реда, който бе представен по-горе, като следва 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 приемане на поръчка от клиента и съобразяване с типа бетон, обема и времето за доставка, на базата на които се сключва договор. Облачната платформа следва да адаптира 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: При този процес бетонът произвежда или „дозира“. Това включва точно измерване и комбиниране на сировините: цимент, инертни материали, вода и други добавки в съответствие с конкретните изисквания. След това сместа се зарежда в превозно средство от тип „миксер“. Поддържането на „интелигентни“ сензори, които да изпращат данни в реално време, е пример за

внедряването на иновативни идеи. Използването на сензорите за определяне на текущите координати на превозното средство, температурата, нивата на вода и др. могат да помогнат при изчисляване на емисиите на въглероден диоксид. Това повишава прозрачността в съответствие с тенденциите в индустрията към устойчиви практики, изпълнявайки регуляторните изисквания за околната среда;

- Доставка: Въз основа на стандарта ISO 9001:2015, който е фокусиран върху качеството и клиентското удовлетворение, проследяването на доставката в реално време може да бъде ключов компонент в предоставянето на висококачествени услуги на клиентите, като се улеснява точната и своевременна информация относно статуса на техните поръчки. Също така, 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 съхранява 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 обезателно 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;

В предложените концептуален и логически модел на системата от втора глава се базира на извършено проучване на организацията на работа в "Хейделберг Цимент Девня" АД, като облачно базираната система позволява интеграция с всякакъв вид платформи и информационни системи, което е от значение за създаването на комплексна среда за SCM. Това обединяване на ресурси и информация в една централизирана система позволява на "Хейделберг Цимент Девня" АД да извлече отчети в реално време, относно поведението на клиентите, тенденциите в търсенето и оперативната ефективност. Така компанията може да прецизира своите стратегии за управление на запасите, да подобри прогнозирането на търсенето и да оптимизира планирането на производствените и логистичните операции.

Освен за България, бордът на директорите отговаря и за управлението и администрирането на операциите в Гърция и Албания. По-конкретно, те са отговорни за заводите Heidelberg Materials Hellas, разположен близо до столицата Атина, както и за Heidelberg Materials Albania в близост до столицата Тирана. В тази връзка, облачната система позволява интегрирането на тези организационни единици в своята инфраструктура и предоставянето на функционалностите до клиентите на тези предприятия. Това би улеснило бързия обмен на информация, която е нужна на мениджърите при отчитане.

Също така, чрез централизираното управление на ресурсите, предприятието може да минимизира излишъците и да оптимизира използването на сировини. Така облачната система допринася за финансовата ефективност, като може да бъде допълнително подобрена, за да предостави на потребителите възможност за достъп и извършвания на плащания по фактури.

Освен това, сигурността и защитата на данните са сред приоритетите на всяка облачна система, което е особено важно в тенденцията на нарастващи заплахи от кибератаки, както и съответни изискванията за защита на корпоративна и клиентска информация. Облачните решения предлагат протоколи за сигурност и редовни актуализации, което допринася данните и системите на "Хейделберг Цимент Девня" АД да останат защитени. Това

укрепва доверието на клиентите в способността на компанията да се грижи за тяхната информация. Така облачната платформа предоставя основа за непрекъснато подобреие и оптимизация на процесите чрез събирането и анализа на големи обеми от оперативни данни. Тази аналитична способност дава възможност на "Хейделберг Цимент Девня" АД да идентифицира тенденции, да прогнозира бъдещи потребности и да внедрява проактивни мерки за повишаване качеството на обслужване.

Въпреки многобройните ползи, които облачната инфраструктура предлага, процесът на нейното внедряване в "Хейделберг Цимент Девня" АД изиска внимателно планиране и управление на промяната, за да се минимизират потенциалните рискове и да се гарантира гладкото преминаване към новата система. Това включва обучение на служителите за работа с новите технологии, адаптиране на вътрешните процеси и политики, както и уверяване, че всички заинтересовани страни са ангажирани и подкрепят промените.

В дългосрочен план, облачната технология отваря вратата на иновационен потенциал за "Хейделберг Цимент Девня" АД, като предоставя платформа за бързо тестване и внедряване на нови идеи и услуги. Същевременно, облачната инфраструктура поддържа висока степен на устойчивост, като осигурява механизми за автоматично архивиране и възстановяване при сривове, което е от значение за минимизиране на потенциалните оперативни прекъсвания (Куюмджиев, 2019).

В тази връзка, от основно значение е да се изберат технологични средства за реализация и доставчик на публични облачни услуги, които да отговорят на специфичните нужди на компанията, както и да осигурят съответствие с местните и международни стандарти за данни и сигурност.

3.2. Избор на технологични средства за реализация на системата

Изборът на 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 SCM.

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

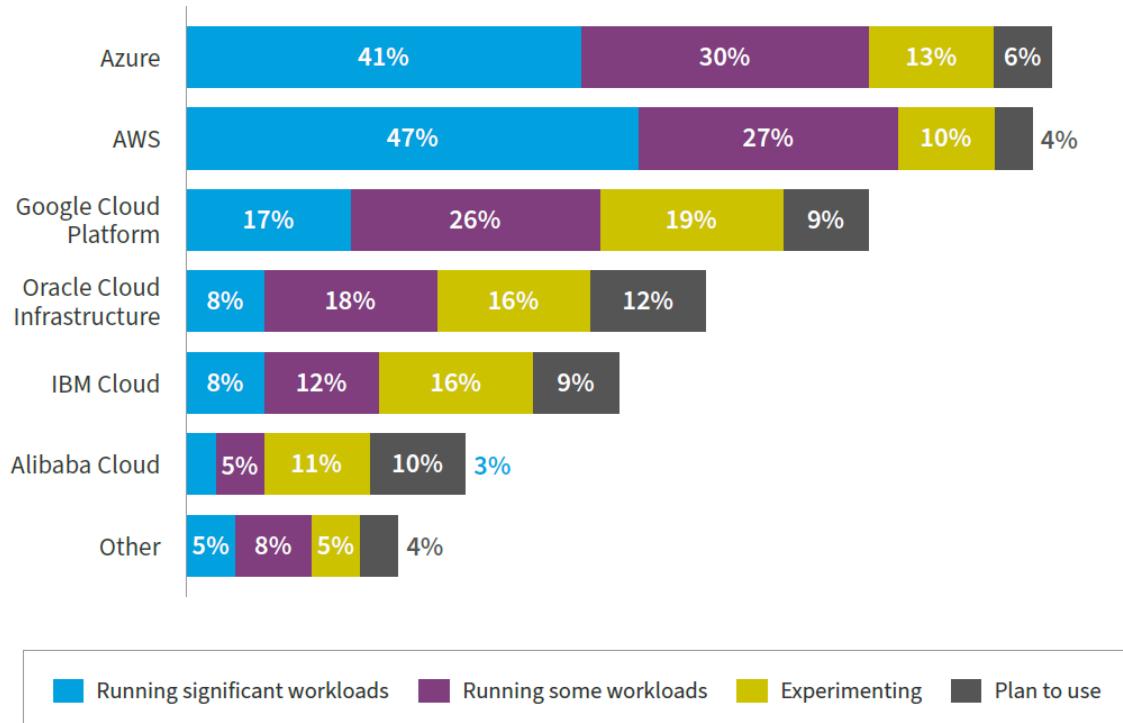
*Таблица 3.1.
Сравнение на сървърни технологии за разработка
(Източник: TechEmpower, 29.09.2023)*

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 отбележва, 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 (Сълов, 2022).

Освен това, GitHub дава информация за над 5,7 милиона месечно активни разработчици по проекти с „отворен код“, използващи езика за програмиране C#, част от еко системата на .NET и Microsoft. От своя страна, Stack Overflow отбележва .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 (Rendle, 2024), 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. На фиг.3.1. са показани тенденциите за използване на различни доставчици на публичен облак в различни предприятия.



Фиг. 3.1. 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 (Likness, 2024)



*Фиг. 3.2. Категоризираща на публичните облачни 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 събранныте 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, машабируеми, 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)*

	Database	Cosmos DB	Blob	Table	File	PostgreSQL, MySQL	SQL Data Warehouse	Data Lake Store
Relational data	✓					✓	✓	✓
Unstructured data		✓	✓					✓
Semistructured data		✓		✓				✓
Files on disk					✓			
Store large data		✓	✓			✓	✓	✓
Store small data	✓	✓	✓	✓	✓	✓	✓	✓
Geographic data replication	✓	✓	✓	✓	✓	✓		
Tunable data consistency		✓						

Сред налични опции, представени на фигуранта по-горе, Azure SQL Database и Azure Cosmos DB се открояват като подходящи решения за съхранение на данни. Със своя релационен модел на данни, Azure SQL Database следва да се интегрира към микроуслугата за управление на потребителите. Azure SQL Database предоставя високо достъпна база от данни като услуга (DBaaS), която поддържа динамични бизнес изисквания и растящи обеми от данни, които са очаквани при централизирано управление на сигурността. Така, изборът на Azure SQL Database не само отговаря на техническите изисквания за съхранение на данни, но и предоставя платформа, която е в състояние да адаптира и мащабира услуги в отговор на изменящите се потребности на потребителите и организацията. От друга страна, Azure

Cosmos DB представя многостранен подход със своята мултимоделна услуга за база от данни, която поддържа ключ-стойност, документи и други. Тази технология борави с полуструктуритирани и неструктуритирани данни, позволявайки на системата да се адаптира към разнообразни изисквания и операции без да прави компромис с производителността.

Технологиите за уеб и мобилни приложения търсят постоянно развитие, което води до съществуването на различни инструменти за изграждане, които работят ефективно на множество платформи (Сълова, и др, 2024). 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 (Сълов, 2024). 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, предлагани "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 онлайн и да се представят добре на мобилни устройства.

*Таблица 3.3.
Сравнение на мобилни и уеб технологии за разработка
(разработка на автора)*

	Естествени приложения (native)	Хибридни приложения	Прогресивни уеб приложения (PWA)	Междуплатформени приложения (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	Обвивка 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	Ограничено, но силно поддържан

	Естествени приложения (native)	Хибридни приложения	Прогресивни уеб приложения (PWA)	Междуплатформени приложения (cross platform)
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.

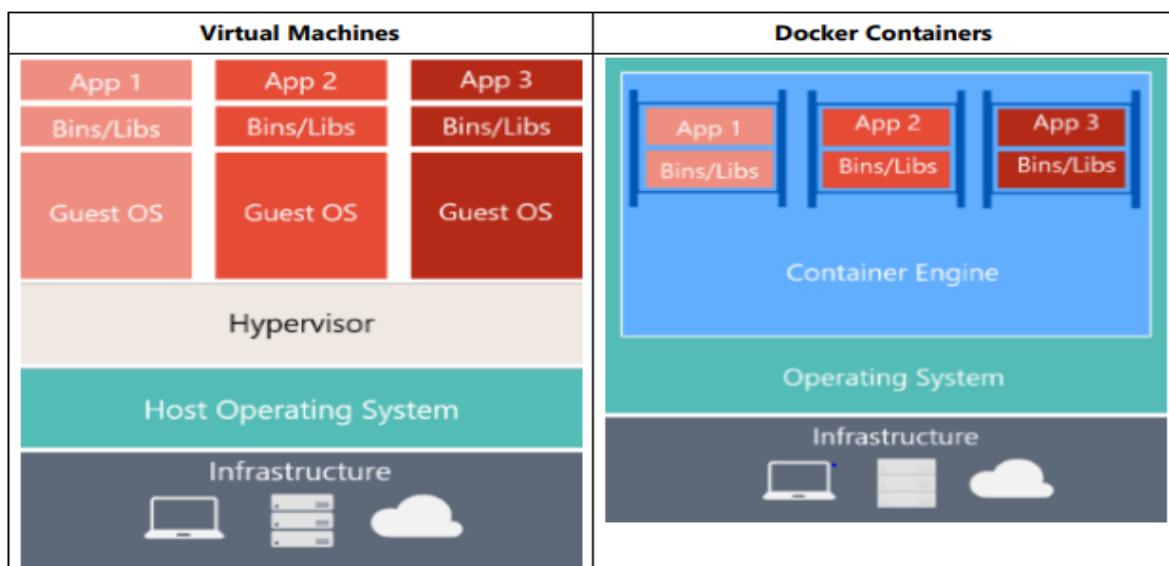
3.3. Физическа реализация на системата

Физическото внедряване на информационната система използва виртуалните машини и инфраструктура на Azure за хостинг на микроуслугите и уеб приложението. Като допълнение, използването на практики по "Развитие" и "Операции" (DevOps), 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 и поддръжка. За изграждане, доставка и изпълнение на системи, изградени като ориентирани към микроуслуги, експерти в областта препоръчват използването на контейнеризирани технологии (De la Torre, 2023). 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 (Toub, 2024). 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 правят промени в различни среди.

Контейнерите също така изолират приложенията едно от друго, в споделена операционна система. 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 (Soper et al., 2024). 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 фиг.3.4. е представено сравнение между компонентите на традиционна виртуална машина и Docker контейнер.



*Фиг. 3.3. Сравнение между Docker и типична виртуална машина.
Източник: Docker, Inc. <<https://www.docker.com/>>, [09.10.2023]*

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. 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, свързани с непрекъсната интеграция и доставка (CI/CD).

*Таблица 3.4.
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 разгръщане на приложения в Azure, един от основните избори, които трябва да имаме предвид, са планираните за използване услуги за хостинг. Този избор е фундаментален за физическа реализация на системата и определя както началните, така и дългосрочните параметри за развитие. В тази връзка, следната таблица представя някои от услугите, част от портфолиото на Azure и техните случаи на употреба.

Таблица 3.5.

*Услуги на Azure за хостинг и съответстващи случаи на употреба
(Източник: Azure)*

	App Service Web Apps	App Service Mobile Apps	Azure Functions	Logic Apps	Virtual Machines	Azure Kubernetes Service (AKS)	Container Instances
Monolithic and N-Tier applications	✓				✓ *		✓
Mobile app back end		✓			✓ *		
Microservice architecture- based applications			✓			✓	
Business process orchestrations and workflows			✓	✓			

Azure App Services е един от начинише за хостване на приложения. Той е предпочитан при монолитната архитектура. При него услугите са достъпни и работят в 99,95% от времето. Тази услуга от тип PaaS предлага функции като автоматично мащабиране, внедряване с нулев застой и удостоверяване чрез активна директория, като също така позволява отстраняването на грешки в приложението докато работи в производствена среда със инструмент, наречен 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. Също така, работи добре и с контейнери.

Друг сходен избор е услугата Azure Virtual Machines, която позволява поддръжката на TCP и SOAP протоколи за обмен на информация, както и преместване на съществуващи приложения от други виртуални машини. Съществуват предварително дефинирани изображения, които да бъдат използвани като Windows Server, който работи с IIS и има инсталзиран и предварително конфигуриран ASP.NET на него, както и собствени софтуерни лицензи (като за SQL Server). Услугата е подходяща за миграция на т.нар.

„наследени системи“, които да бъде използва като източник на данни. В случаите, това се явяват вътрешните системи, които организацията използва и следва да бъдат интегрирани към облачната информационна система.

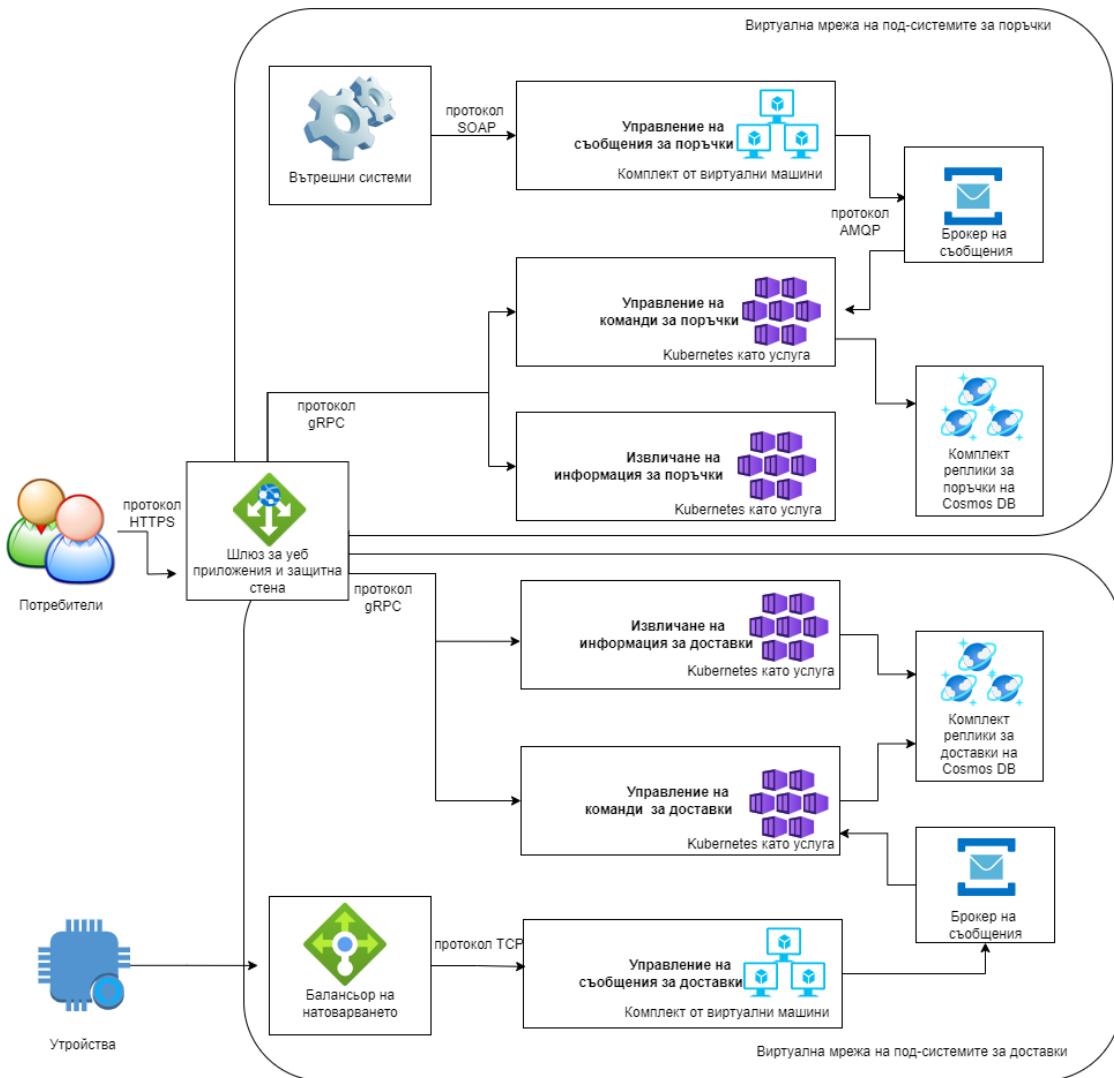
Azure Kubernetes Service (AKS) е водещ инструмент за управление на контейнери, който автоматизира внедряването, мащабирането и операциите свързани с хостване на приложения. Той предоставя на разработчиците услуга от тип PaaS, която се използва за внедряване на облачни приложения и поддръжка на висока достъпност, сигурност и мащабируемост. AKS разпределя ресурсите чрез вътрешния инструмент Azure Availability Zones, позволява автоматично мащабиране на приложения чрез Horizontal Pod Autoscaler. Тези инструменти наблюдават мрежовия трафик към приложения и автоматично коригират броя на работещите екземпляри. По този начин се оптимизират разходите и производителността, като същевременно се следят внезапни пикове в трафика или необичаен растеж в използването (хакерска атака). В този смисъл, сигурността в AKS се разглежда с многопластов подход, който капсулира както инфраструктурата, така и самите приложения. В основата си AKS прилага мрежови политики за регулиране на потока на трафик между модулите, ограничавайки комуникацията чрез оторизация.

AKS е компонент в пакет от услуги на Azure, проектиран да работи съвместно с всякакъв вид услуги, включително Azure SQL Server и CosmosDB (използвани за съхранение), AI и операции за машинно обучение (Sulova et al., 2022), архитектури без сървър като Azure Functions. AKS е базиран на Google проект за управление на работни натоварвания, с отворен код. Kubernetes предоставя операции от високо ниво, които да бъдат извършени чрез кода на самите микроуслуги. Работи с инструкции, които са прехвърлени върху облачните машини, често набор от виртуални машини на Linux или Windows, върху които се разполагат самите приложения (но не директно). Kubernetes организира програмирана автоматизация чрез изчислителни единици (pod), обхващащи един или повече контейнери. Всяка изчислителна единица е проектирана да изпълнява екземпляр на микроуслуга с вътрешен DNS, IP адрес

и порт. Тези единици могат да се създават, унищожават и заменят от Kubernetes контролери, наречени още „задания“ (deployments). Заданията са концепции от по-високо ниво, които управляват изчислителни единици, като определят например кои Docker изображения да бъдат използвани, както и броя на репликите, които да се изпълняват. Изчислителните единици предоставят среда за изпълнение на контейнеризирани микроуслуги, докато заданията управляват жизнения цикъл конфигурацията на системата.

За разлика от Azure App Services и Azure Virtual Machines, AKS позволява автоматизация от най-високо ниво, чрез разпределение на микроуслуги, планиране на контейнери в кълстер, като същевременно се възползва от удобството и функциите на Azure. Считаме, че услугата AKS отговаря на основните изисквания за система за управление на поръчки, базирана на микроуслуги.

Въз основа на разгледаните до тук технологии и инструменти, на фиг.3.6. е представена архитектурна диаграма, която да съответства на концептуалните решения от втора глава и същевременно да се използват по-горе описаните облачни услуги.



Фиг. 3.6. Архитектурна диаграма. (разработка на автора)

Съхранението и поддръжката на програмния код е следваща стъпка в при разработването на софтуерния продукт. Уеб базирани услуги за разполагане на софтуерни проекти и техни съвместни разработки върху отдалечен интернет сървър в т. нар. „ханилище от код“ позволяват контрол на версии и съвместна разработка. Управлението на промените в кода е от значение за информационната система, поради очакванията от постоянни промени и интегриране на нови организационни бизнес единици. За тази цел могат да се използват инструменти като GitHub, GitLab, Bitbucket и Azure DevOps. Въпреки, че главната цел на всички изброени е да поддържат програмния код, основните им характеристики се различават. За сравнение, таблица 3.6, описва

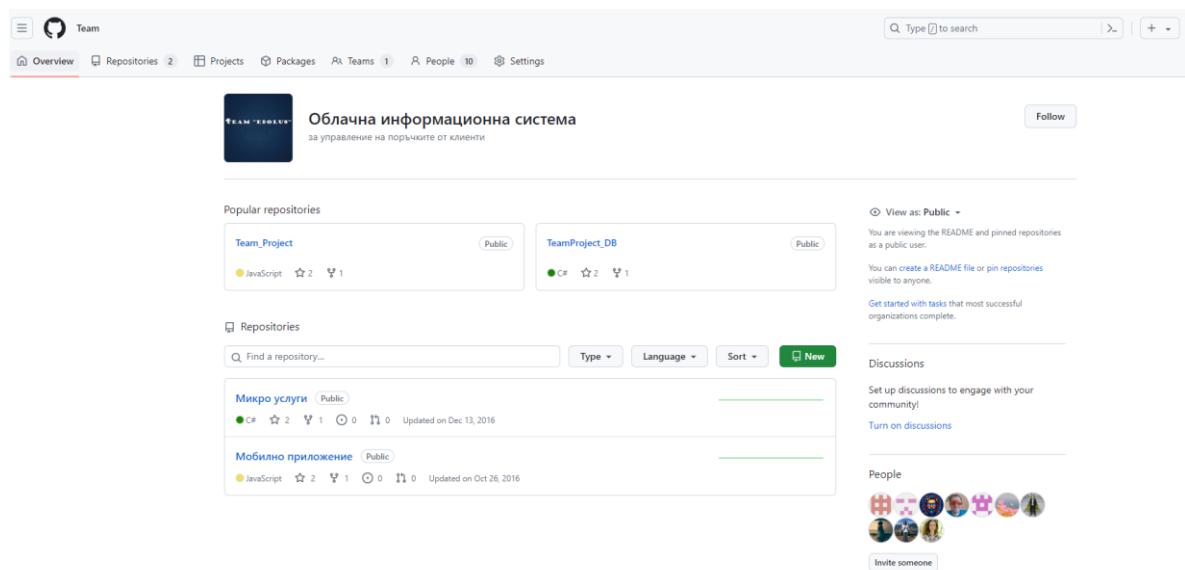
някои от техните силни и слаби страни.

*Таблица 3.6.
Сравнение на уеб базирани услуги за разполагане на софтуер
(разработка на автора)*

	GitHub	GitLab	Bitbucket	Azure DevOps
Контрол на версии	Git	Git	Git/Mercurial	Git/CVS/ Perforce
Интеграция CI/CD	GitHub Actions	Вграден CI/CD, Auto DevOps	Bitbucket Pipelines	Azure Pipelines
Управление на проекти	Задачи, Проекти (табла Kanban)	Задачи, Епици, Важни срокове, Табла Kanban, Пътни карти	Интеграция с Jira, интеграция с Trello, Bitbucket Issues	Azure Boards
Сигурностни функции	Сканиране за сигурност с GitHub Advanced Security	Сканиране за сигурност, Сканиране на зависимости, Съответствие с лицензи	Сигурностни съобщения	Интегрирани сигурностни функции, управление на съответствие
Интеграции	Обширни интеграции с трети страни	Обширни интеграции с трети страни	Силна интеграция с екосистемата на Atlassian	Силна интеграция с услуги и продукти на Microsoft и Azure
Потребителски интерфейс	Интуитивен и лесен за използване	Обширен, но може да бъде сложен	Функционален, но по-малко интуитивен	Обширен, подходящ за потребители на продукти на Microsoft
Общност	Най-голямата общност, обширни ресурси и приложения на трети страни	Силна, особено в предприятията и професионалните среди	Поддържана от Atlassian, по-малка посветена общност	Фокусирана на предприятията, общност ориентирана към Microsoft

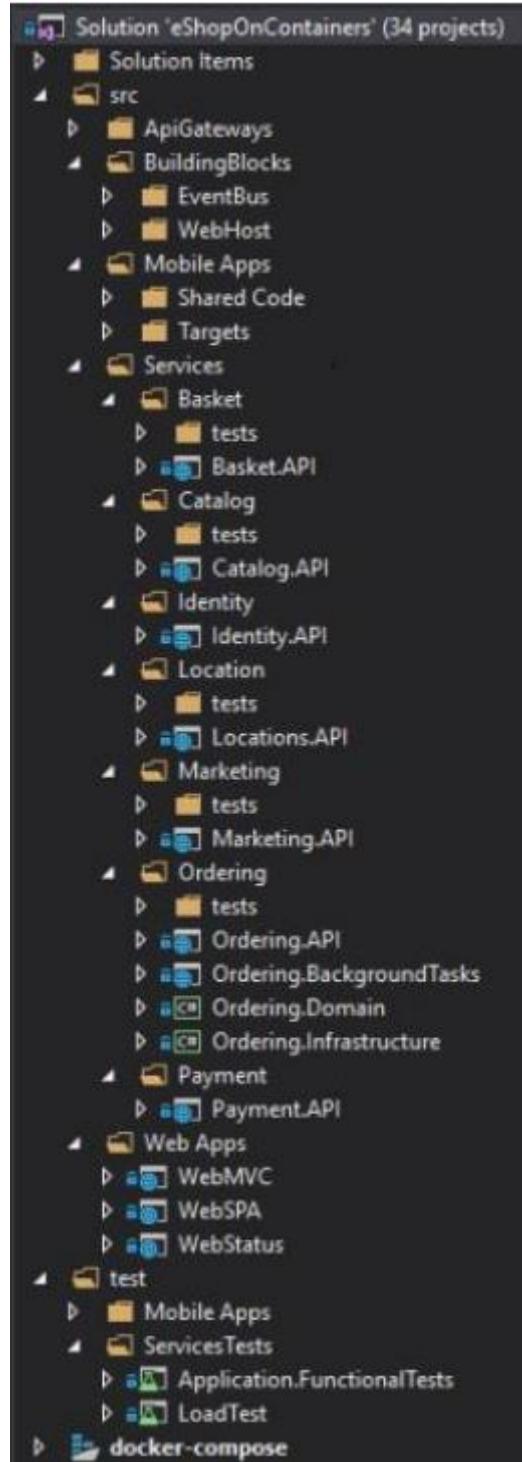
От изброените по-горе, GitHub се отличава с възможностите си за съвместна разработка, CI/CD чрез GitHub Actions, правила за защита, автоматизирани актуализации на зависимостите, проследяването на проблеми и индивидуалните табла за проекти. Освен това, GitHub поддържат продукт, управляван от AI инструмент за допълване на код, наречен Copilot. GitHub Copilot използва езиков модел, който е обучен върху голяма колекция от програмен код. Той предоставя контекстуални препоръки в реално време, което позволява на разработчиците да създават код по-ефективно и с по-малко грешки. Тази технология облекчава психическото бреме върху разработчиците, като предоставя кодови фрагменти в средата за разработка, приспособявайки се към шаблоните за програмиране на текущия проект. Copilot не само ускорява процеса на разработка, но също така поддържа проекта в съответствие с най-актуалните стандарти за кодиране и добри практики.

Разработчиците следва да използват GitHub, като хранилищата на програмния код и настройките по администриране са представени на фиг. 3.7.



*Фиг. 3.7. Хранилищата за програмен код и настройки в GitHub.
(разработка на автора)*

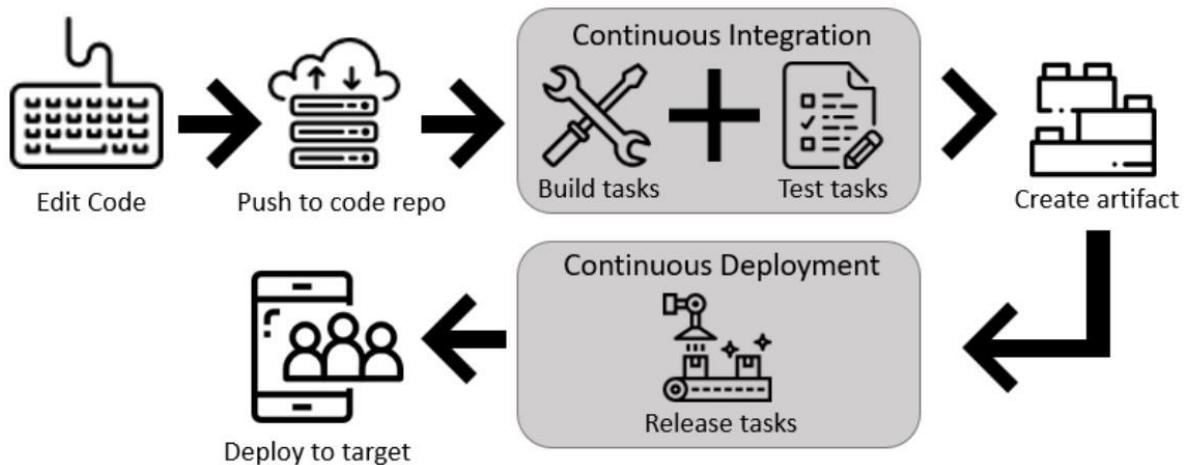
Програмното решение, организирано като под-проекти в интегрираната среда за разработка на Visual Studio, е представено на фиг 3.8.



Фиг. 3.8. Структура на под-проектите във Visual Studio(разработка на автора)

Считаме, че концептуалният работен поток по разработване на информационната система, започва с редактиране на програмен код, който бива качен в GitHub. Това задейства процедурите от етапа по „непрекъсната интеграция“ (CI), при които се компилира нововъведенияния програмен код,

последван от компонентни тестове за валидиране на вече работещите функционалности. При успешна валидация се създава Docker изображение. Втората фаза на „непрекъснато внедряване“ (CD), включва автоматизирани задачи за публикуване на новата версия и съответно създаване на нов контейнер и изчислителна единица в AKS. Всяка микроуслуга, мобилно и уеб приложение следва да премине през изолиран цикъл на изграждане, тестване и внедряване, представено на фиг 3.6.



Фиг. 3.9. Схема на изолиран цикъл на изграждане, тестване и внедряване. Извор: Microsoft <<https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/>>, [23.10.2023]

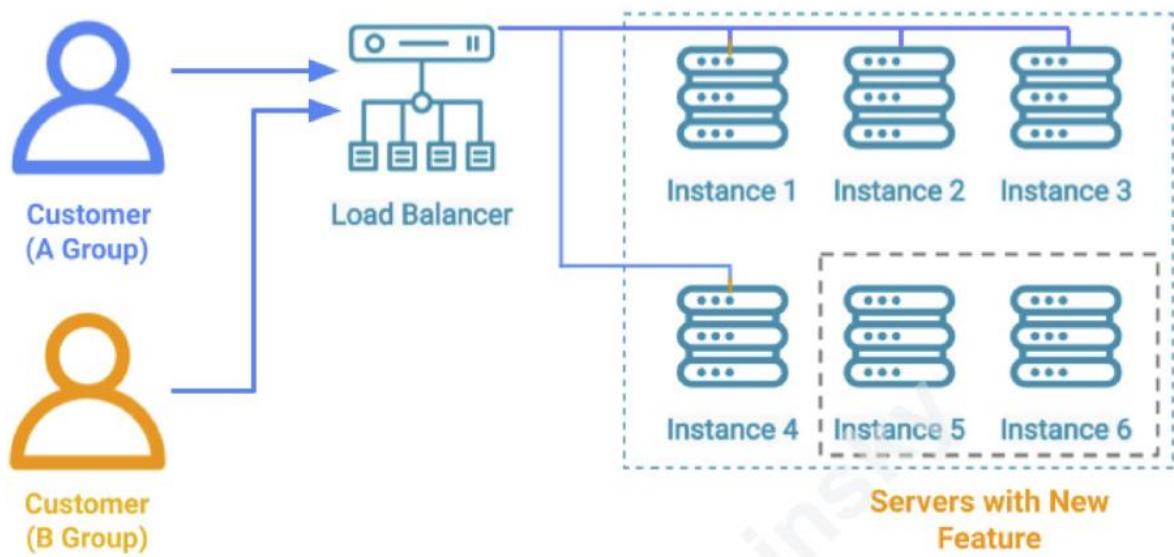
Често срещани в литературата и практиката, са стратегиите за публикуване, внедряване на нови версии и производствено тестване, които представляват процеси по предоставяне на актуализации и запазване на стабилността на системата. Тези стратегии са предназначени да оценят дали системата функционира според очакванията на клиентите в реална среда. В този смисъл, тези стратегии могат да помогнат за предотвратяване на софтуерни дефекти, подобряване на устойчивостта на системата и поддържане на качество и надеждност. Обобщение на някои от тях е представено в таблица 3.7.

Таблица 3.7.

*Обобщение на стратегии за внедряване и производствено тестване
(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 постапно, 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 да бъде приложена към цялата инфраструктура. Целта е да се тества малка част от трафика, преди да се разпространи към по-широката потребителска база.
A/B тестване	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 допринася за изготвянето на сравнителни оценки на ефективността.
Chaos Engineering	Този подход позволява умишлено въвеждане на пропуски в системата по време на нейната работа, като едновременно с това се изследват регистрационните файлове на операциите и се наблюдава системата. Целта е да се открият слабости в регулирана среда, където екипите могат активно да разработват решения за подобряване на устойчивостта на системата.

Като част от апробиране на прототип на информационна система за управление на поръчки, избрахме комбинацията от A/B тестване и Canary release. Считаме, че това е балансиран подход за въведение на нови функционалности, като чрез сегментиране на внедряването е възможно да се наблюдава въздействието на промените в реално време, а идентифицирането на грешки следва да адресира уязвимостите на системата. Както е показано на фиг 3.9, автоматизирани скриптове симулират потребителско поведение в временно създадена облачана среда. Тези скриптове биват сегментирани в две отделни групи: А и В.



Фиг. 3.10. Диаграмата илюстрираща внедряване на облачна услуга, използвайки A/B тестване.. Източник: Stuckenberg (2014)

Трафикът на Група А е насочен към микроуслугите за извлечане на информация, представени като инстанции 1, 2, 3 и 4, чрез балансър на натоварването. Тези инстанции представляват версия на системата, която дава възможност само за „четене“ на информация (read-only) без функционалности за регистриране на нови записи. От друга страна, Група В се насочва към отделен кълстер (инстанции 5 и 6), който поддържа микроузлугите за приемане на команди, надграждащи основните от инстанции 1, 2, 3 и 4. Като резултат от проведените тестове, таб. 3.8. описва основните характеристики.

Таблица 3.8.

Базови характеристики от симулирано внедряване на облачна система, използвайки временна инфраструктура (разработка на автора)

Група	Вид	Функционалности	Намерени програмни грешки	Брой на успешни заявки към сървър	Брой на неуспешни заявки към сървър
A	Възможност за четене	Преглед на данни за поръчка, преглед на данни за доставка	Несъответстващи данни за количество и GPS	25 хиляди GET HTTP заявки	3 хиляди GET HTTP заявки
B	Възможност за четене и регистриране на нови данни	Преглед на поръчка, преглед на доставка, създаване и промяна на поръчка, създаване и промяна на доставка	Неточна дата на новосъздадена поръчка	35 хиляди GET, POST, PUT HTTP заявки	15 хиляди GET, POST HTTP заявки

Вземайки предвид, че Група А е ограничена до възможности само за преглед на данни, броят на неуспешни заявки към сървъра и програмни грешки е по-малък в сравнение с група В. Въпреки това, група В включва цялостни функционалности и по-голям брой обработени заявки. В този смисъл таб. 3.9. представя по-подробен анализ на резултатите от симулацията, като категоризира потребителско изживяване и бъдещи подобрения.

Таблица 3.9.

Допълнителни резултати от симулирано внедряване на облачна система, използвайки временна инфраструктура (разработка на автора)

Група	Вид	Потребителско изживяване	Време за отговор	Намаляване на грешките	Разпределение на ресурсите
A	Възможност за четене	Ограничено до възможности само за преглед на данни без да могат да се правят промени.	Сравнително бързи времена за отговор, допринася за по-ефективно потребителско изживяване.	Подобряване на процесите по синхронизиране, като се подобри точността и надеждността на данните.	Ниски нива на използвани облачни ресурси по операции за четене

Група	Вид	Потребителско изживяване	Време за отговор	Намаляване на грешките	Разпределение на ресурсите
B	Възможност за четене и регистриране на нови данни	Предлага поширова гама от функционалности, която отговаря на динамичните потребителски нужди.	Възможни са по-бавни времена за отговор поради добавената сложност на операциите.	Внедряване на допълнителни проверки за валидиране и механизми за обработка на грешки, за да се намали общият процент на неуспех.	Необходимост от допълнителни ресурси за справяне с натоварване от смесени операции и нови функционалности

В заключение, комбинацията от A/B и Canary release позволява тестване на отделни функционалности както в симулирана, така и в реална среда с минимален риск. Този подход събира информация за въздействието на потребителите върху системата и потенциални проблеми. Чрез този балансиран метод може да се осигури стабилна работа на системата и плавно въвеждане на нови функционалности, минимизирайки риска от прекъсвания в обслужването.

Като допълнение към резултатите от проведените тестове, следва да се направи оценка на приблизителните разходи. За целта може да се използва ценовият калкулатор на Azure, както и стойностите за успешни и неуспешни заявки към сървъра, описани по-горе. Ценовият калкулатор 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 е част от физическа реализация на системата и финансовата стратегия на предприятието. Калкулаторът има функционалността да предостави оценка на разходите, която отразява използването на ресурси в Azure, като същевременно взема предвид всички договорени или намалени цени. На база на предходните анализи, таблица 3.9. предоставя 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 3.10.

*Прогнозни месечни разходи за инфраструктура на облачни услуги
(разработка на автора)*

Тип услуга	Описание	Очаквана месечна цена
Виртуални машини	2 D4s v4 (4 vCPU, 16 GB RAM) (запазени за 3 години), Linux	\$127.60
Балансър 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
Обща сума		\$1999.40

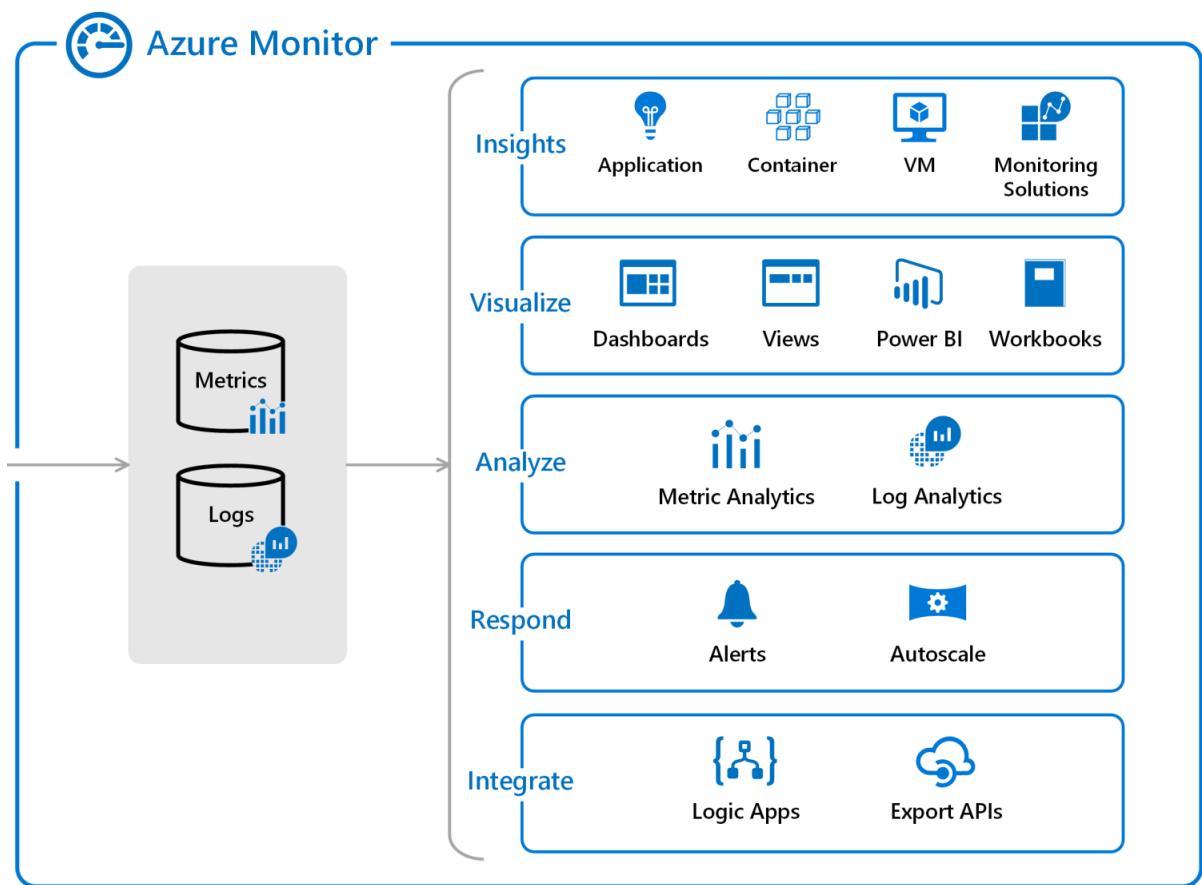
Забележка: Възможно е прогнозните месечни разходи да варират въз основа на различни фактори, като промени в ценообразуването на услугите.

3.4. Системен мониторинг и надграждащи технологии

Ефективното водене на системен дневник и мониторинга са основни компоненти на всяка система, работеща в продукционна среда. Те също така позволяват на допълнителни технологии да се интегрират и да надградят възможностите на съществуваща система или прототип на такава, като предоставят допълнителни предимства за крайните клиенти. Системният дневник (логове) съхранява хронология на събитията, случили се в системата,

като по този начин представлява инструмент за откриване на аномалии, диагностициране на проблеми и предотвратяване на потенциални инциденти. Мониторингът, от своя страна, предоставя възможност за непрекъснато наблюдение на състоянието и производителността на системата в реално време. В този смисъл, мониторинг и системен дневник са два термина, които до определена степен се препокриват. 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. Технологиите, комбинирани като „Elasticsearch Logstash Kibana“ (ELK) биват единица от тип допълваща (поддържаща) система с отворен код, която събира регистрационни файлове от различни източници, съхранява ги за извлечане 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 изискава 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. 3.11. е представен обобщен модел.

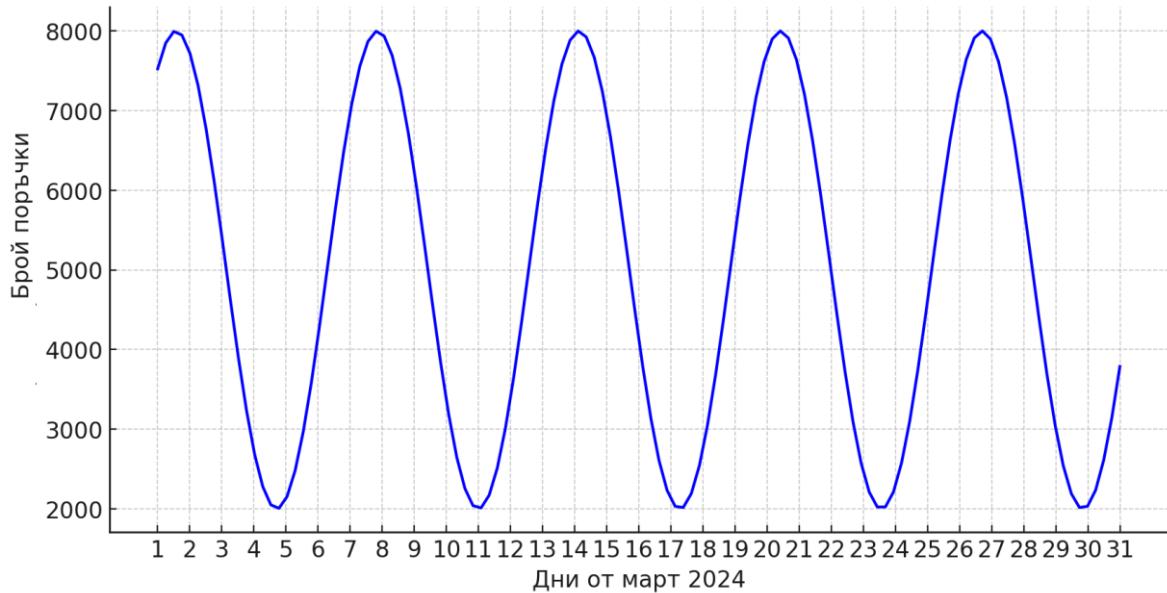


Фиг. 3.11. Обобщен модел на Azure Monitor. Извор: Vettor, 2023

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 система 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.

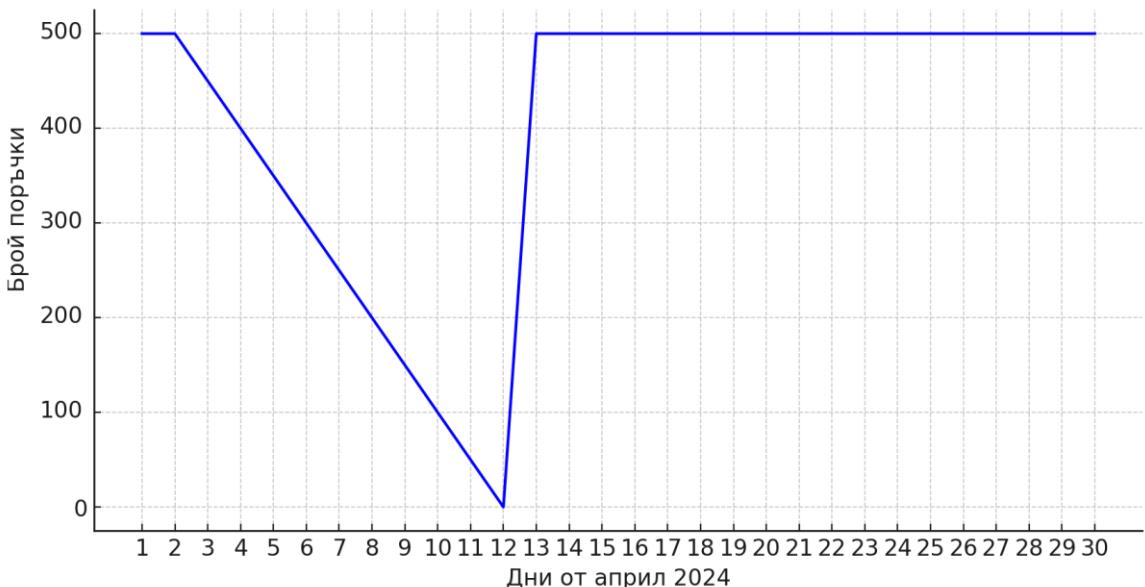
Както бе отбелязано в първа глава, проучване на поведението на крайните клиенти е от основно значение в SCM. За да се предвидят нуждите на клиентите, Azure поддържа услуги за използването на исторически данни, анализ и визуализиране в реално време. По този начин, чрез Azure Monitor се събират бизнес показатели, като например кои конкретни продукти търсят и

поръчват клиентите. За да потвърдим тази възможност, в контролирана среда симулираме тенденции в търсенето. Тези тенденции не са постоянни, като очакван е цикличен модел, изобразен на фиг. 3.9.



*Фиг. 3.11. 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 (Tunç & Büyükkılık, 2017). 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. В тази връзка, такъв пример за колебание при търсенето, осъществен в контролирана среда, е представен на фиг. 3.10. Идентифицирането на такива проблеми в реално време от диспечерите би позволило навременна реакция, която потенциално да предотврати последващи проблеми.



Фиг. 3.12. Пример за колебание при търсенето с Azure Monitor. (разработка на автора)

Системният мониторинг и воденето на дневници са основата, върху която могат да се изграждат и интегрират технологии за подобряване на управлението и ефективността. Някои от тези технологии включват изкуствен интелект (AI) и машинно обучение (ML). Чрез прилагането на AI и ML алгоритми върху данни от мониторинга, системите могат да предсказват потенциални проблеми, оптимизират производителността и осигурят проактивна поддръжка. В тази връзка, Azure и разработени от OpenAI езикови модели, предоставят достъп чрез REST API до инструменти за когнитивни услуги и машинно обучение. Към момента на изследването, Azure е единствения облачен доставчик, който предлага модели на OpenAI, като сред тях са няколко версии на Generative Pre-trained Transformer, включително GPT-3, GPT-4, Codex за вграждане към съществуващи приложения. Други аспекти на Azure OpenAI е инструмент OpenAI Studio, който предоставя визуален интерфейс за тестване и персонализиране на AI модели. Студиото не само улеснява експериментирането с различни конфигурации и параметри, но също така опростява процеса на интегриране в мобилните приложения и работни процеси. Комбинацията от достъпност, сигурност и надеждност прави услугата Azure OpenAI подходящ избор за информационната система.

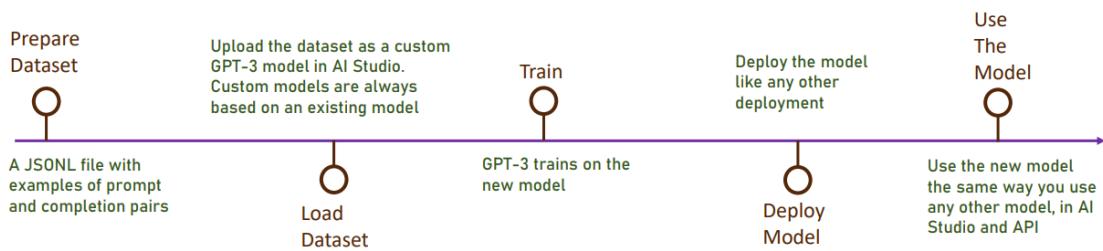
Колекция от услуги за изкуствен интелект, наречени Azure Cognitive Services, предлага уеб APIта, които дават възможност на разработчиците да създават самостоятелни приложения, които да взаимодействат с клиенти и да участват в разсъждения. В таблица 3.5, са описани някои от наличните услуги на Azure, които следва да бъдат интегрирани в оптимизацията на работните процеси:

*Таблица 3.5.
Описание Azure Cognitive Services подходящи за интеграция в
информационната система
(разработка на автора)*

Услуга	Описание	Заявки на минута
Azure AI Search	API, добавящо AI към функционалност за търсене в мобилно или уеб приложение.	120,000
Azure OpenAI Service	Изпълняваща задачи свързани с естествен език.	140,000
Bot Service	Създаване на чат ботове и свързването им към различни канали за комуникация.	160,000
Content Moderator	Откриване на потенциално обидно съдържание от потребителски вход.	220,000
Content Safety	AI услуга, която открива нежелано съдържание.	310,000
Custom Vision	Разпознаване на изображения.	260,000
Document Intelligence	Решение за „екстрактиване“ на данни от снимки на документи.	210,000
Face	Откриване и идентифициране на хора и емоции в изображения.	120,000
Language Understanding	Разбиране на естествения език от вход на потребителя.	120,000
Personalizer	Анализиране на персонализирани съвети за всеки потребител.	140,000
QnA Maker	Симулиране на разговор чрез навигация от очаквани въпроси от потребителя и предварително зададени отговори.	150,000
Speech	Услуга за реч към текст, текст към реч, превод и разпознаване на говор.	180,000
Translator	Използване на технология за превод на над 100 езика.	220,000

Използването на част тези услуги в контекста на облачно базираната система за управление на поръчки, е чрез внедряването на чатбот, който да анализира, разбира и обработва част срещани запитвания от клиенти. Чатботът, разработван като отделно приложение, извлича комбинации от въпроси и отговори от частично организирана информация от ръководства и документи. Също така, достъпът до базата от потребители, поръчки и доставки, не само ускорява обработката на типични запитвания, но също така позволява на диспечерите на смяна да обърнат повече внимание на по-сложни клиентски проблеми, като по този начин се максимизира разпределението на човешкия ресурс. Интеграцията на Bing Search, Cognitive Search, Azure SQL, Cosmos DB и Microsoft Translator, демонстрира отдаденост към подобряване възможностите на AI операции и анализи в съответствие с изискванията на потребителите и техническия прогрес.

Приспособяването на алгоритмите за машинно обучение към конкретни организационни изисквания е задача, която се решава с помощта на масиви от данни, които са достъпни от облачните бази. Понякога те не са достатъчни за да отговарят напълно на специфичните нужди. Чрез използването на методология е възможно постоянно обучение, както е посочено на фиг. 3.14.

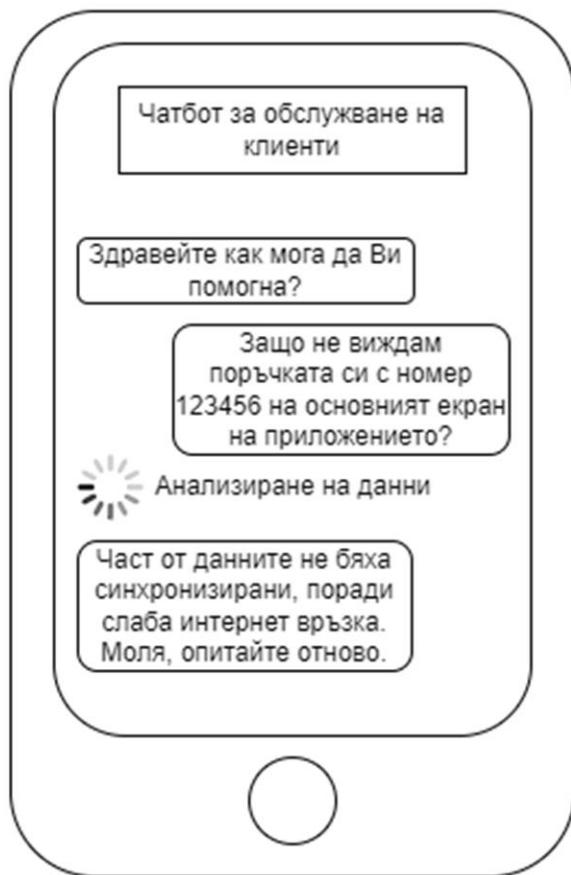


Фиг. 3.14. Процес по приспособяването на алгоритмите за машинно обучение към конкретни организационни изисквания. (разработка на автора)

В тази връзка, интегрирането и на Azure Monitor би помогнало за подобряване на когнитивните възможности чрез наблюдение на използването на чатбота. Azure Monitor създава представа за взаимодействията на потребителите, позволявайки откриване на тенденции, които могат да насочат

отговорите на чатбота и да подобрят неговите възможности. Чрез непрекъснато анализиране на отделните чатове, информационната система може да се коригира и развива, като по този начин подобрява дизайна към променящите се изисквания и очаквания на клиентите.

Частичен пример за употребата на чатбота може да бъде разгледан на фиг. 3.15, като той бива използван от клиент на предприятието за да провери липсваща поръчка. Често, мобилните приложения губят интернет връзка или тя бива недостатъчна за синхронизация с облачния сървър. В такъв случай, е възможно някои от създадените поръчки да липсват на основния экран. Вместо клиентите да търсят диспечерите, те могат да използват чатбота, който има вътрешен достъп до комбинирата информация от сесията на потребителя на текущото устройство, както и базата от данни на поръчки.



Фиг. 3.15. Скица на примерен случай на употреба на чатбот за обслужване на клиенти(разработка на автора)

Допълнителни подобрения на инфраструктурата на приложенията може да бъдат постигнати чрез приемането на „OpenFog“ архитектура, която разширява облачните възможности с периферни изчисления за оптимизиране на производителността, надеждността и обработката в реално време. OpenFog е архитектура, предназначена да доближи възможностите за облачни изчисления до източника на данни чрез използване на крайни устройства. Тази промяна, срещана като fog computing (Tomar et al., 2022), разпределя изчислителните задачи в мрежа от възли, разположени между облака и крайните устройства. По този начин, OpenFog намалява латентността, подобрява времето за реакция, както и цялостната ефективност на системата. За базираната в Azure система, това означава, че данните за поръчките могат бързо да бъдат обработени и валидирани близо до мястото, където са генериирани. Това следва да намали времето, необходимо на данните за трансфер до и от централизирани облачни сървъри, което да доведе до по-бърза обработка на поръчките. Тази способност за локална обработка насочва подходящата информация, като тя бива незабавно достъпна за всички заинтересовани страни. В случай на смущения в мрежата или прекъсвания, крайните възли на OpenFog могат да продължат да работят независимо от Azure, минимизирайки загубата на информация. Също така, модулният характер на OpenFog архитектурата позволява мащабируемост, като с нарастването и увеличаването на броя клиентски поръчки, могат да бъдат разгърнати допълнителни крайни възли, които да се справят с увеличеното натоварване. OpenFog подобрява сигурността и поверителността на данните чрез обработка на чувствителна информация по-близо до нейния източник. То този начин системата намалява излагането на данни на потенциални пробиви по време на предаване.

Изводи и обобщения към трета глава

1. Облачната система рационализира различни бизнес процеси, включително приемане на поръчки, планиране, производство и

доставка. Тази дигитална трансформация подобрява прецизността, надеждността и скоростта, влияйки 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.
4. Интегрирането на допълнителни технологии към облачно базираните системи за управление на поръчките на клиентите представлява допълнителен напредък. Тъй като индустрията се развива непрекъснато, приемането на иновативни подобрения като AI, ML и fog computing следва да бъде от значение за поддържане на оперативни постижения и удовлетворението на клиентите.

Заключение

Проучването на базирана в облак информационна система за управление на клиентски поръчки в произведено предприятие разглежда редица проблеми свързани с SCM. В резултат на изследването на основни научни публикации, както и запознаване с практиката на водещи стопански структури могат, считаме че 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 повищена прецизност, надеждност и бързина от приемането на поръчката до производството и доставката. Тази промяна води до повищена удовлетвореност на клиентите и оперативна рентабилност, давайки на организацията конкурентно предимство на пазара. Натрупаните данни от производствени логистични процеси могат да служат за анализ дейността на производствените предприятия, както и да се правят прогнози на търсене на продукти от крайните клиенти.

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 разгledаното 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 развитие. 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 SCM чрез облачни решения.

Чрез осъщественото изследване по темата считаме, че са постигнати следните резултати с характер на научни и научно приложни приноси, както следва:

1. Разработена е теоретична рамка, която обединява съвременните подходи и технологии за облачно базирано управление на поръчките в контекста на производствените предприятия, предоставящи детайлни модели и алгоритми за оптимизация на процесите във веригата за доставки;
2. Въведена е иновативна архитектура на облачна информационна система, която позволява интеграцията и координацията на различни бизнес функции, включително приемане на поръчки, производство и логистика, подобрявайки оперативната ефективност и намалявайки времето за изпълнение на поръчките;
3. Идентифицирани са основни проблеми и са предложени конкретни методи за тяхното преодоляване чрез внедряване на облачни технологии;
4. Разработени и тествани са нови методи за мониторинг и контрол на процесите по изпълнение на поръчките, базирани на използването на облачни услуги. Също така е извършена емпирична валидация на предложените решения в симулирана среда, която бива копие на производствената;

Списък с фигури и таблици

- Фигура 1.1. Модел на елементите, съставящи управлението на веригите за доставки
- Фигура 1.2. Процесен модел на веригата за доставки
- Фигура 1.3. Сравнение на моделите на облачни изчислителни услуги (IaaS, PaaS, SaaS) и традиционната локална инфраструктура, подчертавайки отговорностите за управление
- Фигура 1.4. Домейн-центрирано срещу данни-центрично в контекста на диаграма за разработка на софтуер, изобразяваща време и сложност
- Фигура 1.5. Карта, описаваща връзките в ОДД
- Фигура 1.6. Модел на Hexagonal архитектура
- Фигура 1.7. Модел на clean архитектура
- Фигура 1.8. Модел на onion архитектура
- Фигура 1.9. Трислоен архитектурен модел
- Фигура 1.10. Пирамидата на тестове
- Фигура 2.1. Итерации процес за проектиране на концептуален модел Ingenuo, 2018
- Фигура 2.2. Диаграма на контекстите
- Фигура 2.3. Диаграма на главен бизнес сценарий
- Фигура 2.4. Диаграма от високо ниво на главните приложения
- Фигура 2.5. Диаграма на базата от данни за потребителите
- Фигура 2.6. Диаграма на активност на поръчка
- Фигура 2.7. Диаграма на последователностите
- Фигура 2.8. Data Interaction Sequence Diagram
- Фигура 2.9. Скица на основен еcran на приложението
- Фигура 2.10. Скица на еcran за доставчика
- Фигура 2.11. Скица на еcran за доказателство за доставка (ePOD)
- Фигура 2.12. Главен еcran в уеб портала
- Фигура 2.13. Еcran за маршрутизиране
- Фигура 3.1. Оценки на индекса TIOBE за езика за програмиране C#
- Фигура 3.2. Доставчици на облачни услуги, използвани от предприятия
- Фигура 3.3. Категоризираща на публичните облачните компании
- Фигура 3.4. Сравнение между Docker и типична виртуална машина
- Фигура 3.5. Архитектурна диаграма
- Фигура 3.6. Схема на основен DevOps работен поток
- Фигура 3.7. Диаграмата илюстрираща внедряване на облачна услуга, използвайки А/Б тестване
- Фигура 3.8. Диаграмата илюстрираща Chaos Engineering
- Фигура 3.9. Пример за тенденция при търсенето
- Фигура 3.10. Пример за сезонни колебания при търсенето
- Фигура 3.11. Пример за цикличност при търсенето

- Фигура 3.12. Пример за колебания при търсенето
- Таблица 1.1. Сравнение между права и обратна верига за доставки
- Таблица 1.2. Организационни структури в SAP
- Таблица 1.3. Обобщение на методологията на дванадесетте фактора
- Таблица 1.4. Допълнение на методологията на дванадесетте фактора
- Таблица 1.5. Стандартни за добри практики на облачната индустрия
- Таблица 2.1. Общи REST конвенции
- Таблица 2.2. Таблица с диапазоните на HTTP кодовете
- Таблица 3.1. Сравнение на сървърни технологии за разработка
- Таблица 3.2. Обобщение на практиките за управление на контейнерите
- Таблица 3.3. Обобщение на стратегии за внедряване, описани в теорията и често срещани в практиката

Използвана литература

1. Александрова, Я. (2020). Analytical customer relationship management system architecture. Monographic Library "Knowledge and Business", Publishing House "Knowledge and Business".
2. Армянова, М. (2019). IoT problems and design patterns which are appropriate to solve them. In Information and Communication Technologies in Business and Education: Proceedings of the International Conference Dedicated to the 50th Anniversary of the Department of Informatics (pp. 291–305). Varna: Science and Economic Publishing House.
3. Армянова, М. (2018). Осигуряване на сигурността в облачна система чрез шаблони за проектиране. Известия на Съюза на учените - Варна. Серия Икономически науки, 7(1), 252–261.
4. Армянова, М. (2018). Patterns classification criteria and scheme. Eastern Academic Journal, 3, 70–89.
5. Атанасова, А. (2021). Управленско счетоводство. Варна: Наука и икономика. Закон за защита на конкуренцията. Държавен вестник, София, бр.17 от 26.02.2021 г.
6. Банков, Б. (2023). Software solutions for responsive and accessible web systems. Научен семинар „Дигитализация, големи данни, изкуствен интелект“, Варна: Наука и икономика, 39–43. ISBN (онлайн) 978-954-21-1145-0.
7. Благоев, Благоева, Желязкова, Василев и др. Стопанска логистика, Варна, 2009., Наука и икономика
8. Димитров, И. (2020). Проблеми на управленското и финансовото счетоводство при приложението на ERP системи. Варна: Наука и икономика.
9. Димитров, П. (2018). Алгоритмични проблеми при внедряване на двуфакторна автентикация в уеб приложения. Научна конференция на младите научни работници - 2018: Сборник с доклади (pp. 126–131). Варна: Стено.
10. Директива 2008/98/EО на Европейския парламент и на Съвета от 19 ноември 2008 г. относно отпадъците и за отмяна на определени директиви
11. Директива 94/62/EО на Европейския парламент и на Съвета от 20 декември 1994 г. относно опаковките и отпадъците от опаковки

- 12.Илиев, П., Сълов, В., & Петров, П. (2010). *Виртуални системи*. Монографична библиотека „Цани Калянджиев”, Варна: Наука и икономика.
- 13.Куюмджиев, И. (2019). *Методологически и технологични аспекти при архивирането на бази от данни*. Варна: Наука и икономика, Бил. Проф. Цани Калянджиев.
- 14.Молов, Д. Теория и практика на управлението на обратните вериги за доставка, София, 2017
- 15.Маринова, О. (2023). Critical factors to succeed at adopting agile software development in a distributed environment. Дигитализация, големи данни, изкуствен интелект: Сборник с доклади от научен семинар (pp. 135–140). Икономически университет - Варна. ISBN (онлайн) 978-954-21-1145
- 16.Милушева, П. (2023). Проблеми при управление на логистиката в строителството: Том 7 (стр. 153 с.). Ико-консулт.
- 17.Начева, Р., Сълова, С., & Пенчев, Б. (2022). Where security meets accessibility: Mobile research ecosystem. In Springer eBooks (pp. 216–231). https://doi.org/10.1007/978-3-031-04238-6_17
- 18.Парушева, С., & Пенчева, Д. (2022). Modeling a business intelligent system for managing orders to supplier in the retail chain with unified model language. In Digital Transformation Technology: Proceedings of ITAF 2020, December 16–17, 2020 (pp. 375–393). New York: Springer Publ. (Lecture Notes in Networks and Systems [LNNS] Book Series; Vol. 224). ISBN 978-981-16-2275-5.
- 19.Парушева, С. (2022). Приложимост и проблеми на облачните услуги в банковия сектор. Информационните технологии - стратегически приоритет в икономиката на знанието: Международна научна конференция, 14-15 октомври 2011 г. (pp. 180–187). Свищов: Академично издателство Ценов.
- 20.Парушева, С., & Александрова, Я. (2022). Дигитализация в строителството в контекста на въздействащи движещи сили и фактори. Списание на Българската академия на науките: Общаакадемично списание на БАН, 135(2), 65–70.
- 21.Пенева, П., Александрова, Я., & Армянова, М. (2013) *Бизнес информационни системи*. Варна: Наука и икономика.
- 22.Радев, М., & Александрова, Я. (2013). Combining virtualization

technologies in SOA-applications.

23. Сълов, В. (2014). *Производителност и ефективност на компютърните системи*. Варна : Унив. изд. Наука и икономика.
24. Сълов, В. (2024). Render modes of .NET Blazor technology. *Journal of Informatics and Innovative Technologies*, 6(1), 9–12. ISSN (печатно) 2682-9517, ISSN (онлайн) 2683-0930.
25. Сълов, В. (2022). Приложение на проектите на платформата .NET при разработка на уеб приложения. *Известия на Икономически университет - Варна*, 66(4), 362–375. ISSN (печатно) 1310-0343, ISSN (онлайн) 2367-6949.
26. Сълова, С. (2023). A conceptual framework for the technological advancement of e-commerce applications. *Businesses*, 3(1), 220–230. <https://doi.org/10.3390/businesses3010015>
27. Сълова, С., Александрова, Я., & Стоянова, М. (2022). A predictive analytics framework using machine learning for the logistics industry. In *CompSysTech'22: 23rd International Conference on Computer Systems and Technologies*, 17-18 June 2022, University of Ruse, Bulgaria (pp. 39–44). New York: Association for Computing Machinery. ISBN 978-145039644-8. <https://doi.org/10.1145/3546118.3546130>
28. Сълова, С. (2021). Big data processing in the logistics industry. *Knowledge and Business Monographic Library*, 7, 6–19.
29. Сълова, С., Петров, П., Радев, М., Александрова, Я., Милева, Л., & Янков, П. (2020). Дигитализация на бизнес процеси в строителството и логистиката. *Knowledge and Business Monographic Library*, 8, April.
30. Сълова, С., Bankov, B., & Stoyanova, M. (2024). Уеб технологии. Варна: Наука и икономика. ISBN 978-954-21-1169-6.
31. Стоев, С. (2018). Приложение на дизайн чрез шаблони при изграждане на информационни системи. *Известия на Съюза на учените - Варна*. Серия Икономически науки, 7(2), 275–282.
32. Тодоранова, Л., & Пенчев, Б. (2023). Digitalization of the public sector. In Сборник с доклади от научен семинар "Дигитализация, големи данни, изкуствен интелект" (pp. 68–73). Варна: Наука и икономика. ISBN 978-954-21-1145-0.
33. Тодоранова, Л. (2015). Публичният сектор в „облака”. *Научни трудове на Русенския университет*, 54, 136–140.

34. Василев, Ю., & Стоянова, М. (2019). Information sharing with upstream partners of supply chains. In Proceedings of the 19th International Multidisciplinary Scientific GeoConference SGEM 2019, Geoinformatics and Remote Sensing (Vol. 19, Informatics, Issue 2.1, pp. 329–336). Sofia: STEF92 Technology Ltd. ISBN 978-619-7408-76-8.
35. Василев, Ю. (2021). Надграждане на банковия софтуер - генериране на декларации по GDPR. Защитата на личните данни и дигитализацията - предизвикателства и перспективи: Сборник с доклади [от кръгла маса, проведена в ИУ - Варна, 1 октомври 2021] (pp. 56–63). Варна: Наука и икономика.
36. Филипова, Н., Парушева, С., & Александрова Я. (2017). *Основи на информационните системи*. Варна: Унив. изд. Наука и икономика,

Интернет източници

1. Smith, S. (2023). *Architecting modern web applications with ASP.NET Core and Microsoft Azure*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 15 януари 2024].
2. Roth, D., Fritz, J., & Southwick, T. (2024). *Blazor for ASP.NET Web Forms developers*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 20 февруари 2024].
3. Toub, S. (2024). *Performance improvements in .NET 7*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 5 март 2024].
4. Vettor, R., & Smith, S. (2024). *Architecting cloud-native .NET apps for Azure*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 18 април 2024].
5. De la Torre, C. (2023). *Containerized Docker application lifecycle with Microsoft platform and tools*. Microsoft Corp [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 22 май 2024].
6. Vettor, R., Molenkamp, S., & van Wijk, E. (2024). *Dapr for .NET developers*. Microsoft. Foreword by M. Russinovich [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 30 юни 2024].
7. Soper, C., Addie, S., & Dembovsky, C. (2024). *DevOps for ASP.NET Core developers*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 12 юли 2024].
8. De la Torre, C., Wagner, B., & Rousos, M. (2024). *.NET microservices: Architecture for containerized .NET applications*. Microsoft Corporation [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 23 август 2024].
9. De la Torre, C. (2024). *Modernize existing .NET applications with Azure cloud and Windows containers*. Microsoft Corp [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 4 септември 2024].
10. Likness, J., & Phillip, C. (2024). *Serverless apps: Architecture, patterns,

and Azure implementation*. Microsoft [онлайн], Достъпен на:
<https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 14 октомври 2024].

11. Britch, D. (2024). *Enterprise application patterns using Xamarin.Forms*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 25 ноември 2024].
12. Stonis, M. (2024). *Enterprise application patterns using .NET MAUI*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 5 декември 2024].
13. Rendle, M., & Steiner, M. (2024). *gRPC for WCF developers*. Microsoft [онлайн], Достъпен на: <https://learn.microsoft.com/en-us/dotnet/architecture>, [Достъпено на 31 декември 2024].

Списък с публикации по темата на дисертационния труд

Статии

1. Jordanov, J., & Petrov, P. (2023). Domain Driven Design Approaches in Cloud Native Service Architecture. *TEM Journal* 12 (4), 1985.
2. Vasilev, J., Petrov, P., & Jordanov, J. (2024). A practical approach of data visualization from geographic information systems by using mobile technologies. *International Journal of Interactive Mobile Technologies*, 18(3).
3. J Jordanov, D Simeonidis, P Petrov. (2024). Containerized Microservices for Mobile Applications Deployed on Cloud Systems. *International Journal of Interactive Mobile Technologies*.

Доклади

1. D Simeonidis, P Petrov, J Jordanov. (2023). Network Intrusion Detection Through Classification Methods and Machine Learning Techniques. *2023 International Conference Automatics and Informatics (ICAI)*, 409-413

Справка за приносните моменти

Основните приносни моменти на настоящето изследване могат да се обобщят по следния начин:

A. В теоретичен план

– направено е изследване на теоретичните аспекти на облачно базираните информационни системи, със специален фокус върху техните преимущества и предизвикателства в контекста на управление на поръчките в производствените предприятия. Това изследване включва детайлен анализ на основните компоненти на такива системи и тяхната роля в оптимизацията на веригите за доставки;

– проведено е изследване на методологическите проблеми, свързани с организацията и управлението на дейността в производствени предприятия с помощта на облачно базирани системи. Анализирани са различни подходи за управление, които включват внедряване на облачни технологии, като се обсъждат техните предимства и недостатъци в контекста на ефективното управление на клиентските поръчки;

B. В приложен план

– с използването на утвърдени формални средства са разработени

концептуален и логически модел на софтуерната система за управление на клиентски поръчки. Тези модели детайлизират структурата и функционирането на системата, като осигуряват основа за нейното последващо реализиране;

– предложен е практически план за реализация на софтуерната система, който включва детайлно описание на етапите на разработка, внедряване и тестване на системата. Този план обхваща и ресурсите, необходими за успешното изпълнение на проекта, както и времевия график за различните фази на реализацията.;

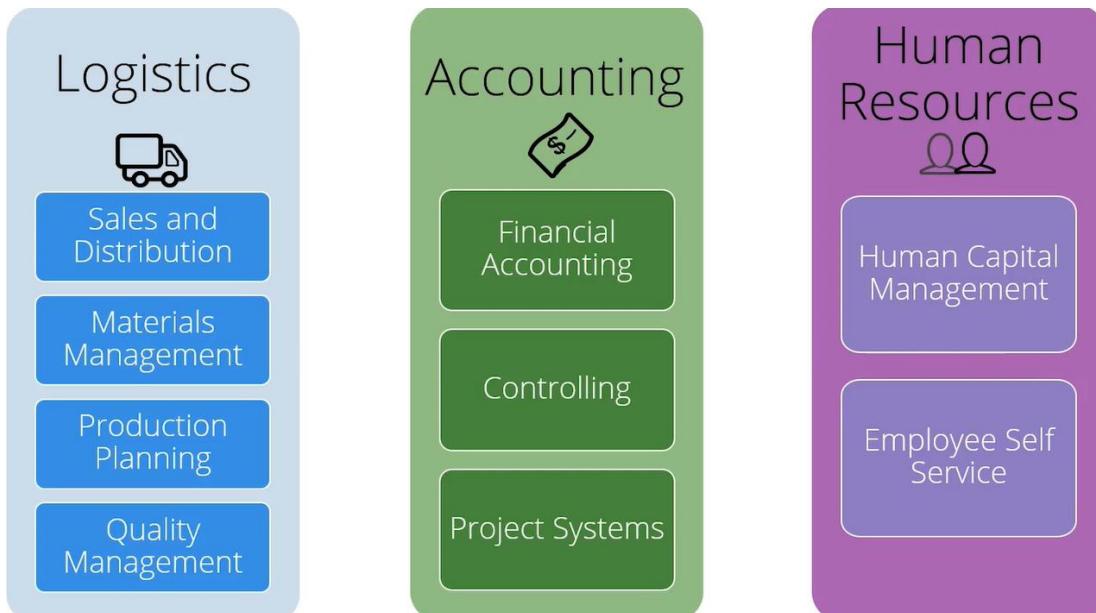
– Аргументирани са софтуерни технологии, които да се използват в разработката на системата, базирайки се на техните технически характеристики и възможности за интеграция с други системи. Това включва избор на програмни езици, рамки и инструменти, които да отговарят на специфичните изисквания на проекта;

– Направено е обобщение на софтуерните инструменти, които могат да се използват в отделните етапи на процеса по създаване на софтуер в облачна среда. Това включва инструменти за разработка, тестване, разгръщане и поддръжка на системата, като се акцентира върху техните предимства и недостатъци в контекста на облачните технологии;

Приложения

Приложение 1. Описание на модула на SAP ERP за продажби и дистрибуция

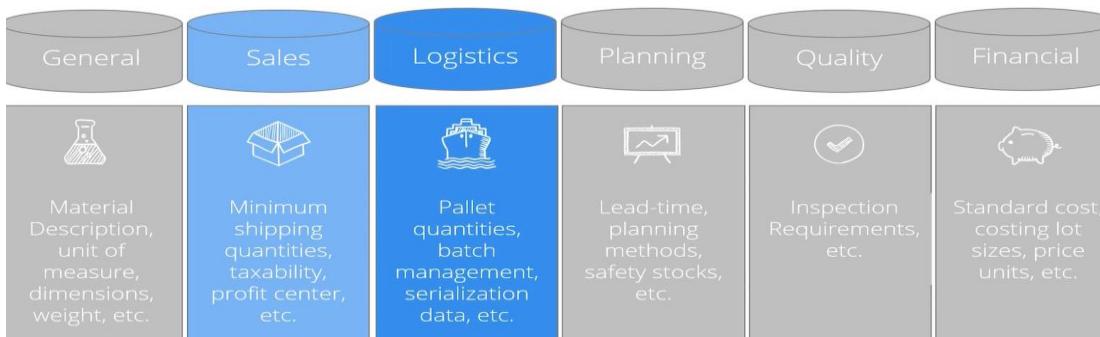
На първата фигура са показани основни модули на SAP, разделени на три основни категории: Логистика, Счетоводство и Човешки ресурси.



Следващата таблица показва ключовите организационни единици в различните модули на SAP. Тези организационни единици са основни структури, използвани за управление и сегментиране на данни в рамките на различните модули.

Finance	Sales and Distribution	Materials Management	Human Capital Management
Chart of Accounts	Sales Organization	Plant	Company Code
Company	Distribution Channel	Storage Location	Personnel Area
Company Code	Division	Purchasing Organization	Personnel Subarea
Business Area	Sales Area	Purchasing Group	

Третата фигура показва различни компоненти и данни, които се управляват в различни модули на SAP ERP системата. Всеки от тези модули включва специфични типове данни и функционалности, които подпомагат управлението на различни аспекти на бизнеса.



Четвъртата фигура описва четири основни роли, които участват в процеса на продажби и дистрибуция в SAP системата. Всяка от тези роли има специфична функция и отговорности в процеса:

- Sold-To Party (SP): Организацията или лицето, на които се продават стоките или услугите. Тази роля представлява клиента, който прави поръчката и с когото се сключва договор за продажба.
- Ship-To Party (SH): Организацията или лицето, на които се изпращат стоките или се предоставят услугите. Тази роля обозначава крайния получател на стоките или услугите. Тя може да се различава от Sold-To Party, ако стоките се доставят на различно място.
- Bill-To Party (BP): Организацията или лицето, на които се изпраща фактурата. Тази роля е отговорна за получаването на фактурите.
- Payer (PY): Организацията или лицето, които са отговорни за извършването на плащането по фактурата. Тази роля обозначава субекта, който ще извърши плащането на издадените фактури. Понякога Payer може да бъде различен от Bill-To Party, ако плащанията се обработват от централизиран финансов отдел или друго трето лице.

Sold-To Party (SP)	Ship-to Party (SH)	Bill-To Party (BP)	Payer (PY)
The entity to which you are selling goods/services.	The entity to where you are shipping the goods or delivering services.	The entity to where you are sending the billing document/invoice.	The entity responsible for remitting payment of a billing document.

Следващата илюстрацията показва еcran от SAP система, който представя информация за цените на материалите. Данните са организирани по продажбени организации, дистрибуционни канали и различни материали.

Material Price							
Material Cntry S		Scale quantity	UoM	Amount	Unit per UoM	Valid From	Valid To
2891	K004	2,300.00-	USD	07/03/2017	07/03/2099		
	PMIN	2,300.00	USD	08/17/2017	12/31/9999		
Sales Org. 0002	CHARLOTTE, NC						
Distr. Channel 01	FOOD						
Material Cntry Rest S		Scale quantity	UoM	Amount	Unit per UoM	Valid From	Valid To
2891	PR00	2,300.00	USD	1	EA	08/13/2017	08/13/2017
Sales Org. 0005	Germany Frankfurt						
Distr. Channel 12	Sold for resale						
Material Cntry Rest S		Scale quantity	UoM	Amount	Unit per UoM	Valid From	Valid To
T-ECO100	PR00	59.99	EUR	1	PC	12/04/2006	12/31/9999
T-ECO101	PR00	59.99	EUR	1	PC	03/27/2006	12/31/9999
T-ECO102	PR00	59.99	EUR	1	PC	12/04/2006	12/31/9999

Следващата илюстрацията показва еcran от SAP система, който представя преглед на стандартна поръчка. Тук са включени различни ключови елементи и данни, свързани с конкретната поръчка.

Display Standard Order 15994: Overview

Standard Order	15994	Net value	799.90	USD																																																															
Sold-To Party	300711	Holden & Associates / 2300 LOOP 410 S.W. / SAN ANTONIO TX 78245																																																																	
Ship-To Party	300711	Holden & Associates / 2300 LOOP 410 S.W. / SAN ANTONIO TX 78245																																																																	
PO Number		PO date																																																																	
<table border="1"> <tr> <th>Sales</th> <th>Item overview</th> <th>Item detail</th> <th>Ordering party</th> <th>Procurement</th> <th>Shipping</th> <th>Reason for rejection</th> </tr> <tr> <td>Req. deliv.date</td> <td>D 05/27/2020</td> <td>Deliver.Plant</td> <td colspan="4"></td> </tr> <tr> <td><input type="checkbox"/> Complete delv.</td> <td></td> <td>Total Weight</td> <td colspan="4">30 KG</td> </tr> <tr> <td>Delivery block</td> <td></td> <td>Volume</td> <td colspan="4">0.000</td> </tr> <tr> <td>Billing block</td> <td></td> <td>Pricing date</td> <td colspan="4">05/15/2020</td> </tr> <tr> <td>Payment card</td> <td></td> <td>Exp.date</td> <td colspan="4"></td> </tr> <tr> <td>Card Verif.Code</td> <td></td> <td></td> <td colspan="4"></td> </tr> <tr> <td>Payment terms</td> <td>ZB01 14 Days 3%, 30/2%, 45 Incoterms</td> <td>FOB</td> <td colspan="4">From the Plant</td> </tr> <tr> <td>Order reason</td> <td colspan="5"></td> <td></td> </tr> </table>					Sales	Item overview	Item detail	Ordering party	Procurement	Shipping	Reason for rejection	Req. deliv.date	D 05/27/2020	Deliver.Plant					<input type="checkbox"/> Complete delv.		Total Weight	30 KG				Delivery block		Volume	0.000				Billing block		Pricing date	05/15/2020				Payment card		Exp.date					Card Verif.Code							Payment terms	ZB01 14 Days 3%, 30/2%, 45 Incoterms	FOB	From the Plant				Order reason						
Sales	Item overview	Item detail	Ordering party	Procurement	Shipping	Reason for rejection																																																													
Req. deliv.date	D 05/27/2020	Deliver.Plant																																																																	
<input type="checkbox"/> Complete delv.		Total Weight	30 KG																																																																
Delivery block		Volume	0.000																																																																
Billing block		Pricing date	05/15/2020																																																																
Payment card		Exp.date																																																																	
Card Verif.Code																																																																			
Payment terms	ZB01 14 Days 3%, 30/2%, 45 Incoterms	FOB	From the Plant																																																																
Order reason																																																																			
All items <table border="1"> <thead> <tr> <th>Item</th> <th>Material</th> <th>Order Quantity</th> <th>Un</th> <th>Description</th> <th>S</th> <th>Customer Material Numb</th> </tr> </thead> <tbody> <tr> <td>101400-400</td> <td></td> <td>10</td> <td>PC</td> <td>Motorcycle Helmet - Stand.</td> <td><input checked="" type="checkbox"/></td> <td></td> </tr> </tbody> </table>					Item	Material	Order Quantity	Un	Description	S	Customer Material Numb	101400-400		10	PC	Motorcycle Helmet - Stand.	<input checked="" type="checkbox"/>																																																		
Item	Material	Order Quantity	Un	Description	S	Customer Material Numb																																																													
101400-400		10	PC	Motorcycle Helmet - Stand.	<input checked="" type="checkbox"/>																																																														

Следващият екранът показва дейностите, които са предвидени за изпращане, под формата на бърз преглед на поръчките за продажби в SAP системата.

Activities Due for Shipping "Sales orders, fast display"											
	Light	Goods Issue Date	DPri	Ship-to	Route	OriginDoc.	Gros	W	Volu	VU	Document
		05/18/2020		2 300711		15992	60	KG			
		05/14/2020		2 300711	000001	15993	60	KG			
		04/21/2020		300711		15987	9	KG			
		04/08/2020		1 1000		15991	300	KG			
				2 300711		15990	40	KG			
		04/02/2020		1 T-S50A	R00025	15986	40	KG			
		04/01/2020		2 300711		15988	60	KG			
		03/25/2020		2 300711		15989	40	KG			
		03/11/2020		2 1033	R00120	15976	20	KG	0.150	M3	
				2 T-S62F	R00110	15975	14	KG			
		03/09/2020		COL112	000001	15953	40	G	0	L	
		02/27/2020		2 T-S62A	R00130	60000123	750	KG			
		02/24/2020		2 T-S62A	R00130	15961	32	KG	0.200	M3	
		02/21/2020		3 1175	R00110	15954	20.0	KG	0.150	M3	
		02/20/2020		2 T-S62A	R00025	15962	205	KG			
		02/18/2020		2 T-S62B	R00135	15949	205	KG			
				2 T-S62A	R00025	15948	328	KG			
		02/17/2020		2 T-S62A	R00130	15947	266.	KG			
				2 T-S62A	R00130	15946	70	KG			
		02/14/2020		2 T-S62B	R00135	15946	410	KG			
				2 T-S62A	R00130	15940	307.	KG			
		02/13/2020		2 T-S62A	R00130	15938	102.	KG			
		02/12/2020		2 T-S62F	R00110	15945	14	KG			
				2 T-S62A	R00130	15939	205	KG			
		12/27/2019		2 41		15932	1,00	KG			
		12/18/2019		2 T-L64A		15928	1.95	KG	11,25	CC	
		12/17/2019		2 2004	R00125	15930	13	KG	90,00	CC	
				2 41		15927	1,00	KG			
		12/13/2019		2 T-L64B	R00125	15931	60	KG	108,0	CC	
		12/09/2019		2 41		15915	51	KG			
				2 41		15916	5,37	KG			
		12/06/2019		2 41		15883	1	KG			

Последният екран показва преглед на фактура (Invoice) в SAP системата. Този преглед предоставя информация за фактурирането на артикулите, включително детайли за клиента, нетната стойност, дати и други съответни данни.

Invoice (F2) 90005177 (F2) Display: Overview of Billing Items										
Accounting		Billing documents								
F2 Invoice (F2)	90005177			Net Value	5,500.00	DEM				
Payer	1390			Technik und Systeme GmbH / Schwalbenweg 43 / D-52078 Aac						
Billing Date	01/03/1997									
Item	Description	Billed Quantity	SU	Net value	Material	Tax amount				
10	Pumpe Stahlguss Etanorm 170-230	1 PC		5,500.00	P-109	825.00				

Приложение 2. Софтуерна сигурност

Софтуерната сигурност в своята основа е защита срещу загуба на данни, прекъсване на услугите, изтичане на информация и/или несъответствие на данните. Следните примери описват случаи на хакерски атаки:

- 1) През 2019 г. хакери атакуват VFEmail, онлайн услуга за електронна поща, като унищожават всички данни и резервни копия. Тази атака води до загуба на информация за всички потребители. VFEmail не успя да възстанови данните на своите клиенти.



- 2) През октомври 2016 г. DDoS атака причинява проблем в интернет услугите на Twitter, Netflix, Reddit и други. Атаката била насочена към DNS сървърите. В резултат, милиони потребители са блокирани да използват приложенията в продължение на няколко часа.



NEWS

DDoS attack takes down Twitter, ramifications for IoT in enterprise

By Freddie Roberts - October 24, 2016

3) <https://www.forbes.com/sites/thomasbrewster/2018/11/30/marriott-admits-hackers-stole-data-on-500-million-guests/?sh=50f10ba46492#786737086492>



Billionaires Innovation Leadership Money Business Small Business

Nov 30, 2018, 08:18am EST

Marriott Hackers Stole Data On 500 Million Guests -- Passports And Credit Card Info Included



Thomas Brewster Forbes Staff
Cybersecurity

Associate editor at Forbes, covering cybercrime, privacy, security and surveillance.