

ИКОНОМИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА  
КАТЕДРА „ИНФОРМАТИКА“



# РЕФЕРАТ

по дисциплината

**„Интернет технологии и комуникации“**

на тема:

**Облачни комуникационни модели в разпределена  
система за управление на поръчки**

Докторант:  
Йордан Йорданов

Научен ръководител:  
доц. д-р Павел Петров

Варна, 2022

## Съдържание

Списък на съкращенията.....	2
Въведение .....	3
1. Синхронна комуникация.....	4
1.1 Механизъм за трансфер на репрезентативно състояние .....	5
1.2 Механизъм за заявки към отдалечени процедури.....	9
1.3 Сравнение на двата стила за реализация.....	11
2. Асинхронна комуникация.....	12
2.1 Базирана на съобщения комуникация .....	12
2.2 Съгласуваност между услугите.....	13
3. Комуникационни модели за достъп до бекенда .....	14
3.1 Директна комуникация на клиент с микроуслуга .....	14
3.2 Използване на шлюз за приложете интерфейси.....	15
4. Препоръки при проектиране.....	17
Заключение.....	18
Използвана литература.....	19

## Списък на съкращенията

<b>REST</b>	Representational State Transfer
<b>GRPC</b>	Google Remote Procedure Call
<b>API</b>	Application Programming Interface
<b>HTTP</b>	Hypertext Transfer Protocol
<b>CNCF</b>	Cloud Native Computing Foundation
<b>WWW</b>	World Wide Web
<b>JSON</b>	JavaScript Object Notation
<b>IDL</b>	Interface Definition Language
<b>SOA</b>	Service-oriented architecture
<b>MIME</b>	Multipurpose Internet Mail Extensions
<b>URI</b>	Uniform Resource Identifier
<b>CRUD</b>	Create, Read, Update, Delete

## Въведение

Комуникационните технологии са от важно значение за много широкомащабни уеб приложения, в частност системи за електронна търговия или управление на поръчки. Те са част от световната мрежа, която сама по себе си представлява разпределена система от взаимосвързани ресурси.

**Актуалността на изследваната тема** се обуславя от тенденцията облачните технологии да се превръщат в инструменти за стратегическа трансформация и дигитализация на бизнеса. Облачните платформи позволяват реализиране на иновативни идеи. Тези предимства поставят компаниите една стъпка пред конкурентите.

**Обект на изследване** в настоящия реферат е разпределена информационна система, базирана на микроуслуги, работеща върху множество процеси и сървъри (хостове). Всяка услуга се изпълнява в отделен процес като контейнер, разположен в клъстер от виртуални машини. Приложенията взаимодействат помежду си с помощта на протоколи за комуникация като TCP , HTTP, AMQP в зависимост от естеството на работа. Инфраструктурата е разгърната в облачната платформа Azure и се управлява от инструмент за оркестрация Kubernetes.

**Целта** на реферата е да представи информационното взаимодействие между подсистемите за управление, които комуникират чрез технологии за изпращане и получаване на данни.

**Основни насоки**, които са поставени:

- да се разгледат актуалните комуникационни модели, които интегрират отделните части на софтуерният продукт;
- да се предложат основни принципи и добри практики при изграждането на облачно базираните приложения;

**Основната теза** на изследването е свързана с внедряването на бизнес процеси „от край до край“, като същевременно се поддържа последователност и съгласуваност в компонентите на системата. Архитектурата трябва да поддържа важни нефункционални изисквания като: висока степен на достъпност<sup>1</sup>, разширяване на мащаба<sup>2</sup> при увеличаващ се потребителски трафик и други.

Клиент и услугите могат да използват различни видове комуникация, насочени към постигането на различни цели. Може да разграничим два основни типа, които се използват между компонентите на системата: синхронна и асинхронна.

## **1. Синхронна комуникация**

Уеб услугите са интерфейси, които са предназначени за комуникация между машини, за разлика от уеб сайтовете, които са насочени към взаимодействието с хората и се достъпват през браузър. Уеб услугите могат да обслужват различни видове клиенти, като мобилни, базирани в потребителския браузър или други сървърни услуги.

В синхрония подход клиентът изпраща HTTP<sup>3</sup> заявка към услуга, която я обработва и връща обратно HTTP отговор. Клиентският код може да продължи своята задача само след получаване на отговор. Тези два компонента имат обща структура.

---

<sup>1</sup> висока степен на достъпност - компонент на технологична система, която елиминира единични точки на повреда, за да осигури непрекъснати операции или време на работа за продължителен период.

<sup>2</sup> мащабируемост - способността на система, мрежа или процес да поддържа увеличаващ се обем на работа.

<sup>3</sup> Протоколът за прехвърляне на хипертекст (HTTP) е мрежов протокол, от приложния слой на OSI модела, за пренос на информация. Използва TCP/IP .

## 1.1 Механизъм за трансфер на репрезентативно състояние

Representational State Transfer представлява софтуерна архитектура за проектиране на уеб услуги, който обхваща основите на световна мрежа. Представен е през 2000г. като част от дисертацията на Рой Т. Филдинг. Продуктите, използващи REST са базирани на хипермедия. Той е независим от протоколите на приложния слой, като концептуалните идеи зад него са взети от HTTP и основавани на WWW.

Основно предимство на REST е, че той използва отворени стандарти и не обвързва внедряването на API или клиентските приложения с конкретна реализация.

REST API са проектирани около ресурси/бизнес обектите. В системата за управление това са потребители и поръчки. Създаването на поръчка може да се постигне чрез изпращане на HTTP POST заявка, която съдържа определена информацията. HTTP отговорът показва дали поръчката е създадена успешно или не. Ресурсът не винаги се основава на един елемент от физически данни. Например, ресурсът за поръчка се внедрява вътрешно от няколко таблици в релационна база данни, но представен като едно цяло. REST е стил за моделиране на обекти и операциите, които приложението изпълнява върху тях. Ресурсите често се групират в колекции. Колекцията е отделен ресурс и притежава собствен идентификатор. Добра практика е да организираме URI в йерархия като например: <https://manager.com/orders> връща колекцията от поръчки. Всеки елемент в колекцията има свой собствен уникален идентификатор, като този за конкретна клиентска поръчка може да бъде <https://manager.com/orders/eu.123123.231>. Представянето на връзките между различните видове ресурси, като например доставки за поръчка:

<https://manager.com/orders/eu.123123.231/deliveries>. Важно да отбележим, че това ниво на сложност може да бъде трудно за поддържане ако връзките между ресурсите се променят в бъдеще.

HTTP протоколът дефинира редица методи, показани в таблица 1.1, които осигуряват различна семантика, когато се прилагат към ресурс:

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

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

Ефектът от конкретна заявка зависи от това дали ресурсът е колекция или отделен елемент. Следващата таблица обобщава общите конвенции, приети от повечето RESTful реализации, използвайки примера за управление на поръчки.

Ресурс	GET	POST	PUT	DELETE
/orders	Извлича всички поръчки.	Създава нова поръчка.	Общо актуализиране на поръчките.	Премахва всички поръчки.
/orders/1	Извлича подробности за поръчка 1.		Актуализира данните за поръчка 1, ако съществува.	Премахва поръчка 1.
/orders/1/deliveries	Извлича доставките за поръчка 1.	Създава нова доставка за поръчка 1.	Общо актуализиране на доставките за поръчка 1.	Премахва всички доставки за поръчка 1.

**Таблица 1.2:** Общи REST конвенции.

В HTTP протокола форматите се определят чрез използване на типове медии, наричани още MIME. За недвоични данни, повечето уеб API поддържат JSON (application/json) или XML (application/xml) като формат за обмен. Те се използват за сбитото представяне на структурирани данни. Например, заявка към посочения по-горе URI за детайли на поръчка, ще върне отговор:

```
{
  "orderId":eu.123123.231,
```



```

"orderValue":99.90,
"productId":1
}

```

Сървърът информира клиента за резултата от заявка чрез използване на предварително зададени “кодове на състоянието”. Класовете кодове на могат да се видят в таблица 1.3.

Статус код	Тип	Описание	Пример
1xx	Информационен	Изпращат се с подготвителна цел	100 Continue
2xx	Успех	Заявката обработена успешно.	200 OK
3xx	Пренасочване	Клиентът трябва да изпрати допълнителни заявка.	303 See Othe
4xx	Грешки в клиента	Резултат от грешна заявка, причинена от клиента	404 Not Found
5xx	Грешка в сървъра	Възниква грешка от страна на сървъра	503 Service Unavailable

**Таблица 1.3:** Таблица с диапазоните на кодовете на HTTP

REST е гъвкав архитектурен стил, който дефинира CRUD-базирани операции. Клиентите взаимодействат със сървъра през HTTP с модел на

заявка/отговор. Той е широко разпространен, но следва да разгледаме по-нова комуникационна технология, набираща скорост в общността на облачните технологии: gRPC.

## *1.2 Механизъм за заявки към отдалечени процедури*

gRPC е модерна, високопроизводителна рамка, която развива дистанционно извикване на процедури (RPC). На ниво приложение, gRPC рационализира съобщенията между клиенти и бек-енд услуги. Произхождащ от Google, това е проект с отворен код и част от Cloud Native Computing Foundation.

Клиентско gRPC приложение разкрива локална функция в уеб услуга, която реализира бизнес операция. Тази локална функция извиква друга функция на отдалечена машина.

В приложенията, базирани на облак, разработчиците често работят на различни езици за програмиране, рамки и технологии. gRPC осигурява „хоризонтален слой“, който абстрахира опасенията от несъвместимост.

gRPC използва HTTP/2 като транспортен протокол, който разполага с много разширени възможности:

- Двоичен протокол за транспортиране на данни, за разлика от HTTP1.1, който е текстов.
- Позволяващо заявки и отговори за асинхронно предаване на голям набор от данни (масиви)

gRPC е базирана на технология с отворен код, наречена Protocol Buffers. Файловете **.proto** осигуряват висока ефективност и платформено-неутрален формат за структуриране на съобщения. Използвайки междуплатформен език за дефиниране на интерфейс (IDL), разработчиците дефинират “договор” за всяка микроуслуга. Договорът, реализиран като текстов .proto файл, описва методи, входове и изходи за всяка услуга.

Същият файл на договора може да се използва за клиенти на gRPC и услуги, изградени на различни платформи за разработка.

Използвайки прото файла, компилаторът може да генерира както клиентски, така и сървърен код за целевата платформа. Кодът включва следните компоненти:

- Строго обособени обекти, споделени от клиента и услугата, които представляват елементи от съобщението;

- Базов клас, който отдалечената gRPC услуга може да наследява;

Може да разгледаме следния пример за **order\_delivery.proto**.

```
syntax = "proto3"; // версия на синтаксиса

option csharp_namespace = "Manager.Protos";

package order_delivery; // идентификатор на пакета

import "google/protobuf/wrappers.proto";

service OrdersDeliveries {
    rpc GetOrder (GetOrderRequest) returns (NullableOrder); // метод за извикване
}

message GetOrderRequest { // формат на съобщението
    google.protobuf.StringValue order_nr = 1;
}
```

По време на изпълнение всяко съобщение се сериализира като стандартно представяне на Protobuf и се обменя между клиента и отдалечената услуга. За разлика от JSON или XML, съобщенията на Protobuf се сериализират като компилирани двоични байтове.

Технологията ни предоставя няколко типа:

- **Unary RPC** - The client sends a request message to the server and receives a response;
- **Server streaming RPC** - The client sends a request message to the server and receives a sequence of responses;
- **Client streaming RPC** - The client sends a sequence of messages and receives a single response from the server;
- **Bidirectional streaming RPC** The client and the server exchange messages in both directions;

### 1.3 Сравнение на двата стила за реализация

Следната таблица сравнява различните архитектурни видове:

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

**Таблица 1.4:** Таблица

Всички услуги, осъществяващи синхронна комуникация имат много знания една за друга. Създава се тясна връзка между различните микроуслуги, което нарушава една от предпоставките за използване на микросервиси. Всеки път, когато бъде добавена нова услуга и тя трябва да бъде актуализирана за нещо, което се случва в системата, ще трябва да се направят промени в кода, за да бъде извикана и тази нова услуга. Така добавянето на нови услуги става все по-трудно. С течение на времето системата може да бъде натоварена на места, които не сме забелязали в началото.

Недостатъците на използването на синхронна комуникация ще бъде „малък мост“, към следващата точка, където ще разгледаме асинхронната комуникация.

## **2. Асинхронна комуникация**

Важен момент при изграждането на микроуслуги е интеграцията помежду им. Комуникацията между вътрешните микроуслуги трябва да е сведена до минимум. Целта на всяка микроуслуга е да бъде автономна и достъпна за потребителя, дори ако другите услуги, които са част от приложението не работят. В случай, когато трябва да се извърши повикване от една микроуслуга към други (като изпълнение на HTTP заявка за данни), за да може да се предостави отговор, отговаря на архитектура, която няма да бъде устойчива, когато някоя от частите се срине. Освен това, създаването на вериги от заявки/отговори, намалява производителността на микроуслугите.

### **2.1 Базирана на съобщения комуникация**

Асинхронните съобщения и управляваната от събития комуникация са от критично значение при разпространението на промените в множество микроуслуги и свързаните с тях домейн модели. Когато настъпят промени, системата се нуждае от начин за съгласуване им в различните модели. Решението е евентуална последователност и комуникация, управлявана от събития, базирана на асинхронни съобщения.

Клиент прави заявка към услуга, като ѝ изпраща съобщение. Тъй като това е асинхронна комуникация, клиентът приема, че отговорът няма да бъде получен веднага или че няма никакъв отговор. Съобщението се състои от заглавие (метаданни като информация за идентификация) и тяло. Съобщенията се изпращат чрез асинхронни протоколи като AMQP. Част от предпочитаната инфраструктура за този тип е брокер на съобщения, който е различен от тези, използвани в SOA. В опростен вариант, той действа като шина.

Налични са различни брокери на съобщения като всеки един има низ за връзка и потребителски достъп. Реалната част на процеса се състои в крайните точки, които публикуват и получават съобщения, тоест в микроуслугите. Важно правило, което системата за управление се опитва да спазва, доколкото е възможно, е да използва само асинхронни съобщения между вътрешните услуги, а синхронна комуникация: само от клиентските приложения към API Gateway.

Когато се използва асинхронна комуникация, управлявана от събития, микроуслуга публикува интеграционно събитие, задействащо се когато нещо се случи в нейния домейн и друга микроуслуга трябва да получи информация за това. Пример е промяна на цената в микроуслуга от продуктов каталог. Микроуслугите се абонираат за събитията, за да могат да ги получават асинхронно. Когато това се случи, получателите може да актуализират собствените си обекти. Тази система за публикуване/абониране се изпълнява чрез използване на реализация на шина за събития.

## *2.2 Съгласуваност между услугите*

Високата степен на наличност и толерантността към частични проблеми не могат да бъдат гарантирани на 100% в разпределените системи. От гледна точка на широкомащабните уеб архитектури, това е основна за съхранението в услугите, тъй като тези компоненти запазват текущите състояния на приложението.

Предизвикателството е да се внедрят бизнес процеси от край до край, като същевременно се поддържа последователност в услугите. Важни предизвикателства за последователността са:

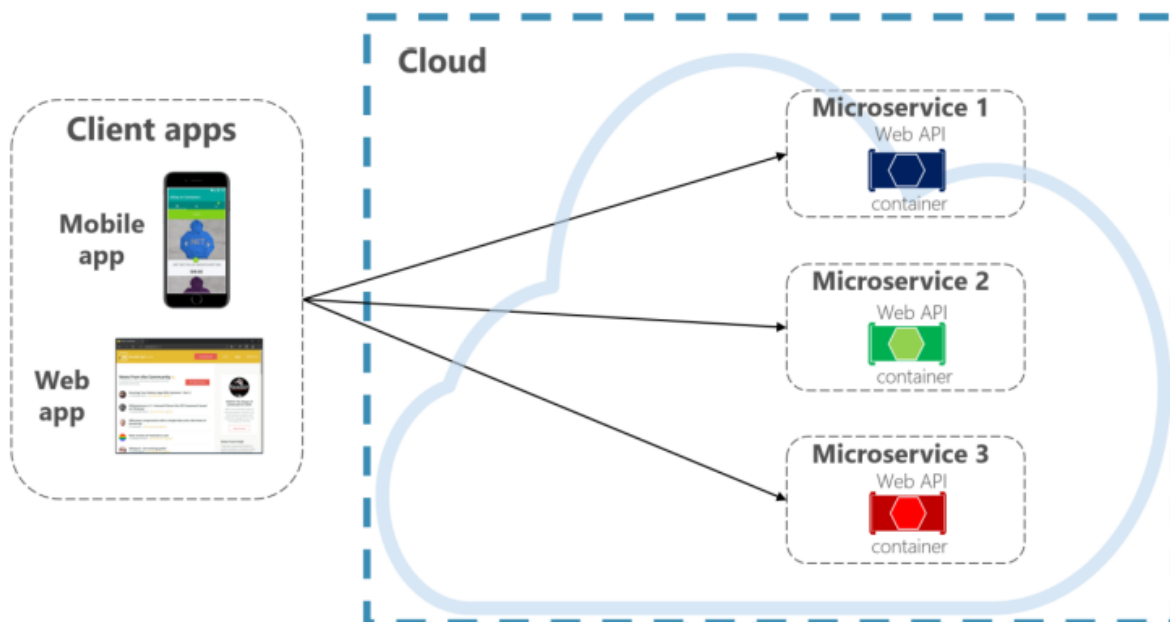
- Нито една услуга не трябва да включва таблици/хранилища за данни от друга и не трябва да извиква директни заявки към тях;
- Предизвикателство при възникване на частична повреда – колкото по-свързани са частите на системата, толкова по-голям проблем.

### 3. Комуникационни модели за достъп до бекенда

В облачната система, мобилните и уеб клиенти изискват комуникационни канали за взаимодействие с микроуслугите. Съществуват два архитектурни подхода, описани в следващите две подточки.

#### 3.1 Директна комуникация на клиент с микроуслуга

За да бъдат нещата опростени, клиент от „предния край“ може да комуникира директно с микроуслугите. Следната фигура показва този вариант:



**Фигура 1.1:** Директна комуникация между клиента и услугата

Този подход се използва, когато различни части от страницата на клиента изискват различни микроуслуги. Всяка микроуслуга има публична крайна точка, която е достъпна от клиентските приложения. Макар и лесна

за изпълнение, директната комуникация с клиента би била приемлива само за прости микросервизни приложения. Този модел свързва тясно клиентите от предния край с основните бек-енд услуги, което отваря врата за редица проблеми, включително:

- Микрослужите трябва да бъдат изложени на „външния свят“;
- Междусекторни проблеми като удостоверяване и оторизация;
- Сложен клиентски код - клиентите трябва да следят множество крайни точки и да се справят с възможни неуспехи;

### 3.2 Използване на шлюз за приложете интерфейси

Като надграждане на първия модел за проектиране, облачната инфраструктура позволява да се внедри услуга: **API Gateway**. Тя предоставя единична точка за група микрослужа. Наподобява модела за дизайн: „фасадата“<sup>4</sup>. Известен е също като „**backend for frontend**“. Изгражда се за конкретните нужди на клиента. Действа като пълномощник между клиентите и микрослужите. Може да осигури удостоверяване, кеширане или други. Azure предоставя няколко готови продукта:

- Azure Application Gateway - насочен към .NET, работещ с архитектура, ориентирана към микро услуги, предоставящ унифицирана входна точка към системата. Услугата поддържа възможности за балансиране на натоварването<sup>5</sup>;
- Azure API Management - шлюз, който позволява контролиран достъп до бек-енд услуги, базиран на конфигурируеми правила. Предоставя уеб портал на разработчиците, които могат да го

---

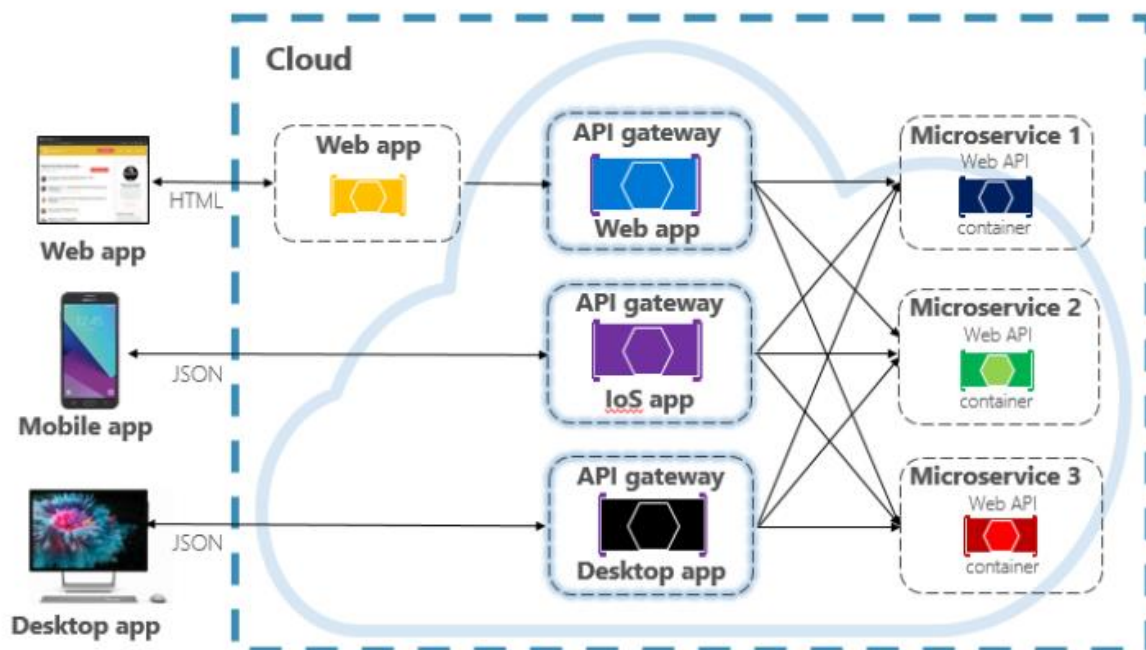
<sup>4</sup> Шаблонът „фасада“ е софтуерен модел, често използван в обектно-ориентирано програмиране, който служи като интерфейс, обхващащ по-сложен структурен код.

<sup>5</sup> Layer 7 Load Balancing, 2022, <<https://www.nginx.com/resources/glossary/layer-7-load-balancing/>>



използват за инспектиране на услугите и анализиране на тяхното натоварване.

Шаблонът е показан на следната фигура:



**Фигура 1.2:** Комуникация от тип “backend for frontend”

API шлюзът може да се превърне в “анти-модел” като монолитно приложение, съдържащо твърде много крайни точки, обединяващо всички микроуслуги. API шлюзовете трябва да бъдат разделени от логически групи въз основа на бизнес ограничения. Протоколи за пренос на данни могат да бъдат HTTP или gRPC.

## 4. Препоръки при проектиране

Примери и указания на водещи експерти от общността, предоставят основни правила за проектиране и разработка, които системата за управление следва:

- В компютърното програмиране създаването, четенето, актуализирането и изтриването са четирите основни операции на съхранение. Най-подходящия стил, в този случай, е REST;
- Ако API е насочено за кратки действия между под-услуги, най-подходящ стил е gRPC;
- Ако не е възможно да се следва REST модела, препоръчителни технологии са ODATA, GraphQL. Те определят набор от практики (стандартизирани са), фокусирани върху бизнес логиката. Дават мощни механизми за заявки, поддържащи готови за използване инструменти;
- Бизнес или клиентски ограничения: ако клиентът очаква REST, не може да ползваме RPC;
- Технологични ограничения<sup>6</sup> като: надеждна и защитена мрежа, нулева латентност, администриране и др;
- Уеб заявките натоварват веб сървъра. Колкото повече заявки, толкова по-голямо натоварване;
- Избягват се зависимости между API и основните източници на данни. Например, ако данните за определена функционалност се съхраняват в релационна база данни, не е нужно веб API да разкрие всяка таблица като колекция от ресурси;
- Най-бавните полета са индикаторът за ефективност;

---

<sup>6</sup> Understanding the 8 Fallacies of Distributed Systems, 2018, <<https://dzone.com/articles/understanding-the-8-fallacies-of-distributed-syste>>

## Заклучение

Облачните информационни системи постигат по-добра организация на търговските процеси, по-висока степен на дигитализация на дейностите.

В първа глава на труда разгледахме опциите за синхронна комуникация с микросервиси. Добре поддържани технологии в ASP.NET Core са REST и gRPC. Въпреки че това работи за по-големи системи, наличието само на синхронни повиквания налага ограничения.

Във втора част на труда, представихме асинхронната комуникация като подходящ метод за комуникация между услуги. Също така и интеграционните събития, които се използват за актуализиране.

Рефератът се фокусира върху важен аспект на ориентираната към услуги архитектурата и се основава система за управление на поръчки към клиенти.

## Използвана литература

1. Robert Vettor (2021) *Architecting Cloud-Native .NET Apps for Azure*. Microsoft. Redmond, Washington.
2. Steve Smith, (2021) *Architecting Modern Web Applications with ASP.NET Core and Azure*. Microsoft. Redmond, Washington.
3. ГАВРАИЛОВ, Е. (2014) *Основи на научните изследвания*. УИ ВСУ „Черноризец Храбър“.
4. ГАНЧЕВ, Г., ДЕЛЧЕВ, М. (2013) *Методика на педагогическите изследвания*. [Онлайн] Достъпно на: <https://obuch.info/metodika-na-pedagogicheskite-izsledvaniya.html> [Достъпено: 20 декември 2021].
5. ДИМИТРОВ, Н. (2013) *Въведение в научните изследвания*. „Интелексперт-94“.
6. ИИКТ – БАН. (2020) *Ефективни методи и алгоритми за Монте Карло симулации, анализ на чувствителността и стохастични оптимизации*. [Онлайн] Достъпно на: [http://ict.acad.bg/?page\\_id=557](http://ict.acad.bg/?page_id=557) [Достъпено: 22 януари 2022].
7. КОРОВКИНА, Н., ЛЕВОЧКИНА, Г. (2022) *Методика подготовки исследовательских работ студентов*. [Онлайн] Достъпно на: <https://intuit.ru/studies/courses/11980/1160/info> [Достъпено: 15 декември 2021].
8. НЕМИНСКА, Р. (2015) *Методи на интердисциплинарно обучение*. Българско списание за образование. Брой 2. с.115-125.
9. ОРЛОЕВ, Н. (2002) *Методология на научните изследвания*. РУ „А. Кънчев“.
10. ПАПАНЧЕВ, Т. (2015) *Обзор на методите за моделиране на надеждността на електронни изделия*. Сп. „Компютърни науки и комуникации“. БСУ – Бургас. Том 4. с. 34-43.