TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

# Vývoj aplikace umožňující propojení vážních jednotek se software INISOFT

## Bakalářská práce

*Studijní program:* B6209 Systémové inženýrství a informatika
*Studijní obor:* Manažerská informatika

*Autor práce:* **Tomáš Drdek**
*Vedoucí práce:* Ing. David Kubát, Ph.D., Ing.Paed.IGIP
Katedra informatiky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

**Zadání bakalářské práce**

# Vývoj aplikace umožňující propojení vážních jednotek se software INISOFT

| | |
|---|---|
| *Jméno a příjmení:* | **Tomáš Drdek** |
| *Osobní číslo:* | E18000500 |
| *Studijní program:* | B6209 Systémové inženýrství a informatika |
| *Studijní obor:* | Manažerská informatika |
| *Zadávající katedra:* | Katedra informatiky |
| *Akademický rok:* | **2020/2021** |

**Zásady pro vypracování:**

1. Analýza typů komunikačních technologií.
2. Volba vhodných technologií.
3. Vytvoření aplikace pro komunikaci mezi vážními jednotkami.
4. Vytvoření aplikace pro testování vážního serveru.
5. Simulace vážního provozu.
6. Vyhodnocení vytvořeného řešení.

| | |
|---|---|
| *Rozsah grafických prací:* | |
| *Rozsah pracovní zprávy:* | 30 normostran |
| *Forma zpracování práce:* | tištěná/elektronická |
| *Jazyk práce:* | Čeština |

**Seznam odborné literatury:**


- AGUILAR, José. 2014. *SignalR Programming in Microsoft ASP.NET*. Microsoft Press. ISBN 9780735683860.
- STRAUSS, Dirk. 2019. *Getting Started with Visual Studio 2019: Learning and Implementing New Features*. Apress. ISBN 9781484254493.
- STURM, Oliver. 2011. *Functional Programming in C#: Classic Programming Techniques for Modern Projects*. John Wiley & Sons. ISBN 9780470744581.
- PROQUEST. 2020 *Databáze článků ProQuest* [online]. Ann Arbor, MI, USA: ProQuest. [cit. 2020‑10-15]. Dostupné z: http://knihovna.tul.cz


*Konzultant:* Ing. Pavel Šír

*Vedoucí práce:* Ing. David Kubát, Ph.D., Ing.Paed.IGIP
Katedra informatiky

*Datum zadání práce:* 1. listopadu 2020
*Předpokládaný termín odevzdání:* 31. srpna 2022

L.S.

Ing. Aleš Kocourek, Ph.D.
děkan

Ing. Petr Weinlich, Ph.D.
vedoucí katedry

V Liberci dne 18. května 2021

# Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

18. května 2021 Tomáš Drdek

## Poděkování

Nejprve bych chtěl poděkovat panu inženýrovi Michalovi Dostálovi a paní docentce Ing. Kláře Antlové, Ph.D. za doporučení společnosti Inisoft s.r.o., ve které mi bylo umožněno vykonávat roční řízenou praxi. Za zvolení mého vedoucího bakalářské práce vděčím doporučení dle pana inženýra Petra Weinlicha, Ph.D. Za vedení, tedy kontrolu a návrhy možných zlepšení bakalářské práci děkuji mému vedoucímu panu inženýrovi Davidovi Kubátovi, Ph.D., ING.PAED.IGIP. V neposlední řadě děkuji jednateli společnosti Inisoft Petrovi Grusmanovi za uvedení do této firmy a seznámení s její činností. Velké díky za trpělivost, rady a řízení mé práce patří mému vedoucímu ze společnosti, inženýrovi Pavlovi Šírovi.

# Anotace

Tato bakalářská práce se zabývá analýzou problematiky v oblasti vývoje software a následujícím vývojem aplikace sloužící k zajištění komunikace vážních jednotek ve firmě Inisoft.

Teoretická část této bakalářské práce zaznamenává postup při prvotním rozboru neboli postup při vývoji aplikace v oboru informačních technologií. Například souhrn vlastností dvou technologií a následné porovnání a výběr vhodnější technologie pro moji aplikaci.

Praktická část představuje aplikaci získaných teoretických informací v praxi, tedy vývoj aplikace.

**Klíčová slova**

komunikační protokol, gRPC, konzolová aplikace, WinForm aplikace, služba, konfigurace, zpětná vazba, multitasking, vlákno, knihovna, rozšíření

# Annotation

This bachelor thesis deals with the analysis of the issue and the subsequent development of an application to enable communication of weighing units in the company INISOFT. The theoretical part of this bachelor thesis records my progress in the initial analysis of the issue, i.e. the development of an application in the section of information technology. For example, summarizing the properties of two technologies and then comparing and selecting a more suitable technology for my application. The practical part is the application of the theoretical obtained information in practice or application development.

**Keywords**

communication protocol, gRPC, console application, WinForm application, service, configuration, feedback, multitasking, thread, library, extension

# Obsah

# Seznam obrázků

# Seznam použitých zkratek

| | |
|---|---|
| WIM | Weight In Motion |
| CRM | Customer Relationship Management |
| ERP | Enterprise Resource Planning |
| PC | Personal Computer |
| COM | Component Object Model |
| WinForm | Windows Form |
| TCP/IP | Transmission Control Protocol / Internet protocol |
| gRPC | Google Remote Procedure Calls |
| HTTP | Hypertext Transfer Protocol |
| API | Application Programming Interface |
| JSON | JavaScript Object Notation |

# Úvod

V oblasti komunikačních technologií jsem vždy jevil zájem o vyhledávání řešení daných problémů, tedy algoritmizaci. Ovšem, co se týče mé představy o nějakém konkrétním zadání z praxe, ale také mých zkušeností a praktických dovedností, nebyl jsem si jist, co bude pozice asistenta programátora obnášet. A právě nyní jsem měl příležitost to zjistit, získat více zkušeností, dovedností a také se v oblasti vývoje komunikačních technologií zdokonalit.

**O společnosti Inisoft s.r.o.**

Společnost Inisoft s.r.o. je česká společnost zabývající se především vývojem a následnou implementací kvalitního software v oblasti odpadového hospodářství a životního prostředí. Další činností této společnosti je poskytování služeb, jako jsou například školení nebo poradenství se zákazníky. Tato společnost má dvacetiletou historii počínaje v roce 2000, kdy byla tato společnost založena. O rok později v roce 2001 byla na trh uvedena verze programu EVI 8 sloužící pro evidenci odpadů. Za další zajímavý milník této společnosti lze považovat například dosažení počtu deseti členů týmu v roce 2005. Důvodem byl vyšší objem práce zapříčiněný nárůstem poptávky ze strany zákazníků.

Obrázek 1: Logo společnosti Inisoft, foto Inisoft

# 1. Analýza typů komunikačních technologií.

Tato část se zabývá analýzou typů komunikačních technologií, tedy rozborem jednotlivých technologií z pohledu jejich vlastností, výhod a nevýhod.

## 1.1. Teoretická analýza

Teoretická analýza dané problematiky je typickým prvním postupem při vývoji aplikace. V první řadě je nezbytné vyslechnout a pochopit požadavky zákazníka, který bude aplikaci používat. Jedná se tedy především o uspokojení potřeb zákazníka. Na druhé straně neboli na straně firmy pak bude výstupem například finanční zisk, nebo také nová reference firmy, která přispěje marketingové strategii firmy do budoucna, a to například potenciálním nárůstem zájemců o služby dané firmy.

## 1.2. Vážní systémy

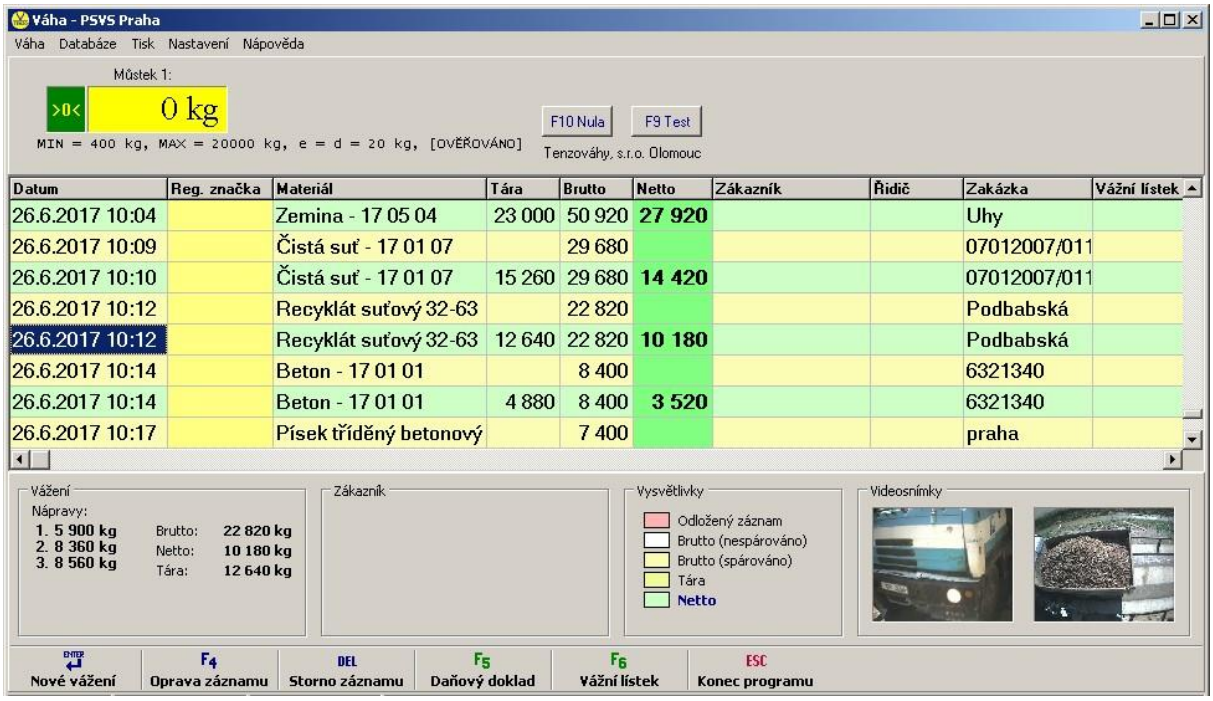
Jednou ze společností, se kterými společnost Inisoft s.r.o. udržuje spolupráci je společnost TENZOVÁHY s.r.o., která se již od roku 1991 také zabývá obdobnou problematikou v oblasti vývoje vážních systémů. Rozšířeným produktem této firmy jsou přenosné váhy PW-10

### 1.2.1. Váhy PW-10

Přenosný váhový systém PW-10 lze použít pro řízené vážení státními orgány a soukromými společnostmi, stejně jako pro namátkové kontroly nakládky vozidel v dopravních společnostech a průmyslových podnicích. Tenké a lehké vážicí zařízení PW-10 kombinuje spolehlivost tenzometrů s vysokou mobilitou v každodenním provozu. Tento systém zajišťuje vynikající manévrovatelnost váženého vozidla. Příprava tohoto zařízení zabere jen několik minut, takže vážení je možné téměř okamžitě. Přenosnou váhu PW-10 lze snadno a rychle používat téměř na jakémkoliv místě. Základní součástí přenosné váhy PW-10 je tenzometr, kterým vozidla postupně procházejí. Údaje o vážení jsou zobrazeny na PC a mohou být dále zpracovávány.

Obrázek 2: Princip vážení, Foto Tenzováhy

Váha vydrží v provozu po celý rok a je nezávislá na externím napájení po dobu až 60 hodin. Další předností těchto vah je zvýšený faktor ochrany proti korozi. Na silniční váze PW-10 je možné vážit ve statickém režimu (přes jednu nápravu) nebo v dynamickém režimu pro rychlejší vážení za jízdy. Pokud jde o software, váha bývá zejména připojena k počítači, což znamená, že smlouva o vážení může být pomocí tiskárny vytištěna a archivována po neomezenou dobu, a to bez potřeby vynaložení dalších nákladů. Tento software za základní cenu produktu implementuje pokročilé funkce, které usnadňují celý proces vážení vozidla včetně rozsáhlé správy a exportu dat.




Obrázek 3: Software firmy Tenzováhy s.r.o., Foto Tenzováhy

Váha je také vybavena režimem dynamického vážení vozidla (WIM), který umožňuje rychle klasifikovat vozidla jako potenciálně přetížená, nebo například naopak prázdná. Tímto způsobem lze nejen provádět kontrolu vážení pomocí vysokorychlostních dynamických vah zabudovaných do vozovky, ale také lze kontrolu vážení provádět na jiných místech (například v případě podezření na přetížení vozidla). To významně zvyšuje efektivitu vážení, a navíc také umožňuje detekovat podezření z nelegálních činností. Software je optimalizován v souladu s požadavky vnitrostátních orgánů České republiky (tj. Policie České republiky nebo Obecná celní služba) a může lišit v závislosti na typu vážního zařízení nebo také požadavcích zákazníka.

Systém vážení poskytuje uživatelům informace v reálném čase o aktuální hmotnosti vozidla, veškerá data o jeho provozu, včetně statistik, ale v poslední řadě také o možných nestandardních situacích, které mohou nastat během vážení. Kromě registrace vozidel, materiálu a dopravy může tento uživatelsky přívětivý software také generovat faktury, zůstatky a exportovat data do příslušného informačního systému společnosti. To pomáhá dále plánovat a optimalizovat operace. Ve srovnání s tradičními mostovými váhami, které jsou vybaveny pouze indikátory hmotnosti, tak může použití softwaru pro přímé ovládání váhy z počítače přinést uživatelům mnoho výhod.

### 1.2.2. Vážení za jízdy (WIM z anglického Weight In Motion)

Jak už vyplývá z názvu, tento způsob vážení umožňuje získání váhy vozidla, a to bez nutnosti jeho zastavení, tedy za jízdy, a to za pomoci váhových senzorů. Nedochází tedy k pozastavení provozu na daném místě, a právě to je důvodem, proč jsou systémy WIM využívány. Při vážení dochází k získání kompletní váhy vozidla, pro získání objemu nákladu vozidla může tedy být potřebné odečíst hmotnost samotného vozidla v případě, že k tomu software není předurčený a neudělá to automaticky.

Dalším softwarovým rozšířením může být také validace naměřených hodnot v případě potřeby automatizace takového vyhodnocení. Pak je také potřeba zvážit tolerované odchylky které mohou vzniknout v důsledku chyb měření. Takovými chybami při měření může být třeba nekvalitní stav pozemní komunikace nebo také nerovnoměrně umístěný náklad který není upevněný a bude se tak v úložném prostoru vozidla volně pohybovat. Tento princip není

uplatňován pouze v oblasti evidování, jako jsou sběrné dvory, ale své uplatnění nachází také v oblasti dopravy na pozemních komunikacích. V kombinaci s použitím rychlostních radarů může také být využíván orgány státní správy, kde může být podstatná právě dříve zmiňovaná validace naměřených dat, tedy jestli hmotnost vozidla spadá do třídy přesnost.

### 1.3. Zákazník a jeho potřeby

První otázkou, kterou je nutné si položit je tedy *„Jaké jsou vlastně potřeby našeho zákazníka?”.* V případě vývoje aplikace s rozsáhlým spektrem funkcí pro uspokojení více potřeb zákazníka najednou může být klíčové, a pro naši budoucí práci ulehčující, navrhnout a vytvořit si vývojový diagram, který slouží ke grafickému znázornění jednotlivých kroků našeho budoucího pracovního postupu.

V druhé řadě je důležité se vžít do role zákazníka neboli podívat se na danou problematiku ze strany zákazníka. K tomu si můžeme napomoci například, zeptáme-li se sami sebe *„Jací jsou budoucí uživatelé mé aplikace? Jsou technicky zkušení?”.* Na tuto otázku existují dvě typické odpovědi. Uživatelé mohou být technicky zdatní například v případě, že se jedná o budoucího uživatele, který již má zkušenosti v oblasti informačních technologií, ovšem je nutno počítat také s druhým případem a to, že uživatel nemusí mít takové zkušenosti s užíváním aplikací. V tomto případě bude tedy naším úkolem najít způsob, kterým aplikace komunikuje a přenést ho do jazyka, který bude srozumitelný i pro laického uživatele.

#### 1.3.1. Řízení vztahů se zákazníky

Řízení vztahů se zákazníky (CRM z anglického Customer relationship management) je strategie společnosti, která bývá používána k analýze interakcí a datového toku se zákazníky po celou dobu již od stanovení poptávky zákazníkem, ale také nadále po uspokojení poptávky zákazníka, jelikož může být požadována komunikace a následně také zpětná vazba. K již hotovému produktu lze také poskytovat služby například v podobě údržby, nebo následnému zdokonalení stávajícího produktu, právě tímto lze dosáhnout nárůstu míry zákaznické spokojenosti a zákazník tedy firmě pravděpodobněji poskytne pozitivní zpětnou vazbu například v podobě recenze a zároveň zůstane společnosti a jejím službám věrný.

Cílem CRM je tedy zlepšit vztahy v oblasti služeb zákazníkům a napomoci k udržení zákazníků a podnítit růst prodeje. Systémy CRM sestavují údaje o zákaznících prostřednictvím různých

kanálů nebo kontaktních míst mezi zákazníkem a společností, což může zahrnovat webové stránky společnosti, telefon, email nebo také sociální sítě. Systémy CRM mohou také poskytnout zaměstnancům orientovaným na zákazníka podrobné informace o osobních údajích zákazníků, historii nákupu, nebo také nákupních preferencích.

### 1.4. Plánování podnikových zdrojů

Plánování podnikových zdrojů (ERP z anglického Enterprise Resource Planning) je proces využívaný k integraci a řízení jednotlivých částí podniku. Systémy ERP jsou pro firmy podstatné, jelikož jim pomáhají realizovat plánování zdrojů, a to integrací všech procesů potřebných pro provoz jejich společností, a to pomocí jediného systému. Systémy ERP můžou také pomoci integrovat například oblasti plánování, marketingu, financí, lidských zdrojů a dalších.

### 1.5. Desktopové aplikace

Desktopové aplikace jsou čím dál více populární a zákazníky poptávané. Důvodů je hned několik, ať už je to automatizace a usnadnění jednotlivých procesů, evidence dat pomocí databáze, nebo například zajištění komunikace více objektů mezi sebou. Uplatnění desktopových aplikací naleznete téměř ve všech oborech, příkladem může být obor účetnictví, nebo také zdravotnictví. Spektrum využití aplikací je tedy velmi široké.

### 1.6. Windows Forms aplikace

Windows Forms je framework uživatelského rozhraní pro sestavování aplikací na ploše systému Windows. Poskytuje jeden z jednoduchých způsobů tvorby desktopových aplikací založených na tvorbě vizuálních návrhů, které jsou dostupné ve Visual Studiu. Funkčnost, jako je přetažení a umístění vizuálních ovládacích prvků, usnadňuje vytváření desktopových aplikací. Pomocí WinForms vyvíjíte graficky bohaté aplikace, které lze snadno upravovat a lze na nich pracovat v režimu offline nebo při připojení k internetu. Aplikace WinForms mají přístup k místnímu hardwaru a systému souborů počítače, kde je aplikace spuštěna a je zde tedy možnost práce s daty v podobě čtení, ale i zápisu.

Používáte-li vývojové prostředí, jako je Visual Studio, můžete vytvořit aplikace pro chytré klientské objekty systému Windows Forms, které mohou vyžadovat vstupy od uživatelů, zobrazují informace a také mohou komunikovat se vzdálenými počítači po síti. Formulář WinForms je vizuální plocha, na které zobrazujete informace uživateli. Aplikace Formuláře

systému Windows obvykle vytvářítе přidáním ovládacích prvků do formulářů a vytvořením odpovědí na uživatelské akce, jako jsou kliknutí myší nebo stisknutí klávesy. Ovládací prvek je samostatný prvek uživatelského rozhraní, který zobrazuje data nebo přijímá vstup dat. Když uživatel něco provede s vaším formulářem nebo jedním z jeho ovládacích prvků, generuje se akce události. Aplikace na tyto události reaguje kódem a události zpracovává, jakmile k nim dojde. Formuláře systému Windows obsahují různé ovládací prvky, které můžete přidat do formulářů: ovládací prvky, které zobrazují textová pole, tlačítka, seznamy atd.

## 1.7. Funkcionální programování

Funkcionální neboli funkční programování je obecně považováno za průkopníka programování, kterého lze použít v mnoha jazycích-dokonce i v jazycích, které původně nebyly v tomto paradigmatu určeny. Jak už říká název, principem funkčního programování je rozdělení kódu do jednotlivých bloků-funkcí. Funkční programátoři mohou k vytváření nových funkcí využívat funkce které byly již dříve vytvořené. Funkce tedy představují jednotlivé stavební bloky, to ovšem neznamená, že programátoři využívající metodu funkčního programování nemají k dispozici žádné další jazykové prvky, ovšem funkce jsou hlavní strukturou pro vytváření architektury a tvorba funkcí je tedy upřednostňována.

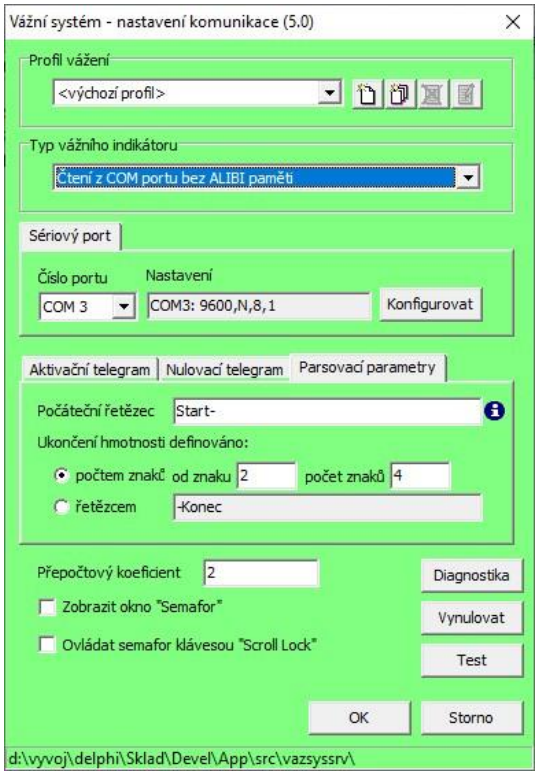
Transparentnost referencí je důležitou myšlenkou v oblasti funkčního programování. Návratová hodnota transparentní funkce závisí pouze na hodnotě předaného vstupního parametru narozdíl od základní myšlenky programování imperativního, u kterého stav programu často ovlivňuje návratové hodnoty funkcí. *Pro některé programátory je funkční programování přirozený způsob, jak říct počítači, co by měl dělat, a to popisem vlastností daného problému ve stručném jazyce. Možná jste slyšeli rčení, že funkční programování je spíše o tom, říct počítačům, jaký problém by měly řešit, a ne tolik o určení přesných kroků řešení.* (STURM, 2011)

## 1.8. Stávající stav aplikace *sklad odpadů*

Vážení v dosavadně společností a jejími klienty používané aplikaci *sklad odpadů* je realizováno pomocí volání COM objektu a jeho funkcí. Jedná se o COM objekt registrovatelný pod systémy Windows. Tato aplikace spravuje připojení vážních indikátorů pro odečet hmotností a dalších

parametrů z váhy. Vážní jednotky je možné připojovat přes sériový port, USB port (simulací sériového), další aplikace s výměnou dat přes soubor, TCP/IP.

V této aplikaci je možné definovat více profilů, pro různé způsoby komunikace s více váhami.

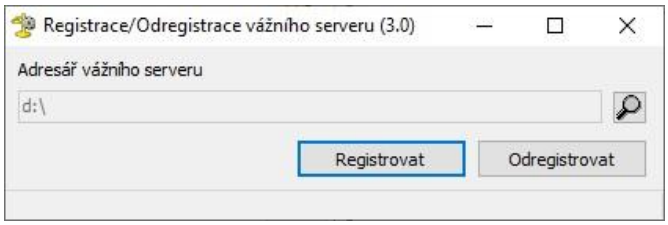


Obrázek 4: Sklad odpadů

Dalším objektem je desktopová aplikace pro náhled váhy (dále TestApp), sloužící pro testování funkčnosti vážení bez nutnosti instalace skladu odpadů.

Obrázek 5: Testovací aplikace

A v poslední řadě desktopová aplikace pro registraci a odregistraci COM objektu v systémech windows.



Obrázek 6: Registrační aplikace

### 1.8.1. Technologie COM

COM (z anglického Component Object Model) je binární, objektově orientovaný systém zprostředkovávající komunikaci dvou objektů. Je to jedna z prvních technologií svého typu, která byla vydána firmou Microsoft. Tato technologie ovšem nebývá nadále aplikována, důvodem je nárůst počtu konkurenčních technologií, které mohou mít oproti této technologií více výhod, jelikož jsou konkurenční společnosti vlastnící tyto technologie schopné je udržet aktuálními a ochotné se i nadále podílet na jejich stálému rozšiřování. Výhodami může být například rychlejší komunikace, nebo také multiplatformnost neboli kompatibilita s více operačními systémy. Proto je nezbytné zvážit, kterou z nově dostupných technologií použít. (Microsoft, 2018)

## 1.9. Analýza typů aplikačních služeb

Prvním bodem mého zadání, které mi bylo ve firmě přiděleno, a tudíž i prvním krokem byla analýza typů aplikačních technologií z teoretické stránky tedy sběr informací, shrnutí příslušných vlastností daných technologií a následné porovnání a výběr optimální technologie. Analýza možností, které máme, v tomto případě tedy analýza typů aplikačních služeb je základním stavebním kamenem pro výběr právě jednoho z daných typů, který bude pro naši aplikaci, pokud možno co nejpřínosnější. Typů aplikačních služeb existuje několik, a právě proto je třeba klást důraz na řádné shrnutí jednotlivých atributů daných typů aplikačních služeb.

### 1.9.1. Vlastnosti aplikačních služeb

**Způsob komunikace**

Způsob komunikace je jeden z hlavních faktorů, který je nezbytné zvážit ještě před implementací dané služby, tedy při výběru typu služby. Rozlišujeme především dva typy komunikace, těmi jsou komunikace jednosměrná a komunikace obousměrná neboli komunikace se zpětnou vazbou. Zatímco při jednosměrné komunikaci dochází ke komunikačnímu vysílání pouze z jedné strany neboli z jednoho objektu, u obousměrné komunikace tomu tak není. Principem obousměrné komunikace je použití zpětné vazby, obě strany se tedy na komunikaci aktivně podílí. Komunikaci prvků informační technologie lze přirovnat ke komunikaci lidské. Typickou formou je tedy vedení dialogů. Komunikaci v oboru informačních technologií si lze představit třeba jako konfigurační aplikaci, která vysílá

informace o konfiguraci straně serveru, který dále tyto informace o nastavení pouze zpracuje a aplikuje.

Oproti oboustranné komunikaci, jejíž příkladem může být komunikace serveru a klientského serveru, kde server vyšle žádost klientskému serveru o získání váhy a klientský server pak zjistí váhu a odpověď vyšle směrem zpět, tedy serveru.

**Multiplatformnost**

„*Bude má aplikace využívána na jednom, nebo na více typech operačních systémů?* “ právě tato otázka nám může pomoci při zvažování výběru technologie. Pokud víme, že naše aplikace bude využívána pouze jedním typem platformy, například operačním systémem Windows, pak nám postačí výběr technologie, která je kompatibilní pouze s operačním systémem Windows, tedy není multiplatformní. Takovou technologií je například služba Windows. V opačném případě, tedy v případě potřebné dostupnosti aplikace z více platforem přichází v úvahu výběr právě takové technologie, která bude kompatibilní s více typy operačních systémů, tedy bude multiplatformní. Na úkor takové volby ovšem mohou přijít ostatní atributy vybrané technologie, které v porovnání s ostatními technologiemi nemusí být tak vhodné a účinné.

**Programovací jazyk**

Faktorů, podle kterých si lze vybrat ze stovek již existujících typů programovacích jazyků právě jeden, ve kterém se chceme v budoucnu rozvíjet je několik. Jeden z hlavních faktorů výběru je platforma, na které bude daný softwarový produkt vystavěn a zároveň, na které bude tento produkt dostupný svým uživatelům. Platformou lze rozumět typ operačního systému, softwarové produkty se tedy dělí na desktopové aplikace a mobilní aplikace a záleží právě na vývojáři, který typ si zvolí. Lze také zvážit výběr vysokoúrovňového jazyka, který se oproti těm nízkoúrovňovým programovacím jazykům liší tím, že má vyšší míru abstrakce, psaní jejich kódu je tedy uživatelsky přívětivější, jelikož má o něco blíže k lidské řeč než ke strojovému kódu jako takovému.

Přínosem pro nás může být také právě to, že si zjistíme, které programovací jazyky jsou aktuálně firmami využívány a poptávány, k tomu nám může pomoci ať už internetový prohlížeč, nebo třeba také exkurze daných společností, které se v tomto odvětví pohybují. Za

předpokladu, že již známe populární programovací jazyky, které stále nalézají nové uplatnění, nezbývá nám, než si nějaký vybrat a začít v něm programovat. Právě již dříve vytvořené referenční projekty v daném jazyce nám mohou v budoucnu pomoci nalézt uplatnění hned v několika směrech.

### 1.9.2. Webová služba

Tato technologie je multiplatformní a své uplatnění nachází především, pokud je vyžadována komunikace a interakce více komponent. Jelikož je komunikace zprostředkována pomocí internetového komunikačního protokolu HTTP, potřebou je hostování webového serveru, na kterém služba poběží a bude tak uživateli dostupná pomocí internetového prohlížeče. Nutností je tedy zabezpečení přenosu dat při probíhající komunikace a vhodné je také zvážit, zda chceme, aby služba byla přístupná každému, či jen naším zákazníkům. (Core Technologies Consulting, 2020)

### 1.9.3. Windows služba

Službu Windows lze aplikovat pouze v rámci operačního systému Windows, není tedy multiplatformní. Tuto službu je nutné spustit manuálně nebo událostně, takovou situací může být například spuštění služby při spuštění zařízení, takovou obslužnou rutinu události pak musí vývojář inicializovat. jednou z předních výhod této služby je právě její chod, který je po celou dobu již od spuštění nepřerušovaný a zároveň běží skrytě v pozadí operačního systému. K službě nemůže být přistupováno ze strany uživatele běžným způsobem, jelikož uživatelská obsluha této služby není běžná, tato služba se využívá především k jednorázovému vykonání určitého sledu funkcí. V případě nutnosti uživatelské interakce se službou je možné si službu přizpůsobit, a to přidělením práv umožňujících interakci s danou službou budoucím uživatelům. (Core Technologies Consulting, 2020)

### 1.9.4. gRPC služba

gRPC (z anglického Google Remote Procedure Calls) je novodobé výkonné rozhraní vyvinuté službou Google, které zprostředkovává vzájemnou komunikaci dvou nebo více objektů. Jednou z předních výhod je způsob dokumentace. Pomocí konfigurace služeb a souborů definice rozhraní API můžete generovat referenční dokumentaci, kterou bude Vaše aplikace používat. Tato konfigurace je také snadné implementovat a přizpůsobit podle vlastních potřeb. (Google, 2020)

## 1.10. Analýza typů komunikačních protokolů

Druhým bodem mého zadání, který bylo potřeba splnit ještě před samostatným vývojem dané aplikace byla analýza typů komunikačních protokolů. Byly mi doporučeny a představeny dva typičtí představitelé komunikačních protokolů, které aktuálně bývají aplikovány a využívány v oblasti vývoje aplikací, ovšem mým úkolem byl výběr optimálního protokolu. Těmito představiteli jsou protokoly REST a gRPC a nezbývá tedy než dané typy porovnat.

### 1.10.1. protokol REST

Tento protokol je sice starší, ale stále kompatibilní se staršími API. Využití nachází především v oblasti vytváření infrastruktury mikroslužeb. Pokud jde o přetížení, tento protokol používá jednoduchý způsob zápisu dat JSON, komunikační zprávy jsou tedy v textovém formátu, ty ovšem mají větší objem dat oproti formátu binárnímu. Běžným používaným přenosovým protokolem je HTTP 1.1, který je příliš velký, a proto se také déle čeká na odezvu. (Envato Pty, 2018)

### 1.10.2. protokol gRPC

Protokol gRPC je novodobý a výkonný protokol dostupný více programovacími jazyky. Využíván je zejména při tvorbě infrastruktury určené pro externí uživatele. K zápisu dat dochází pomocí metody protobuf, tedy protokolových bufferů, které třídí strukturovaná data a to tak, že odesílaná data komprimují do binárního formátu, zprávy tedy zabírají méně místa a rychlost přenosu je tak navýšena. Používaným přenosovým protokolem je HTTP/2, který se pomocí kratší čekací doby na odezvu také podílí na zefektivnění formy komunikace. (Envato Pty, 2018)

# 2. Volba vhodných technologií.

Dalším bodem po provedení analýzy jednotlivých technologií byl výběr optimálních technologií, které jsem použil právě v mé aplikaci, a právě to je shrnuto v této části. Můžete zde také najít například důvody mých výběrů.

## 2.1. Volba programovacího jazyka

Jelikož se mé zadání zabývá problematikou vývoje desktopové aplikace, bylo k uskutečnění daného vývoje aplikace potřeba zvolit vhodný programovací jazyk. Jak jsme se již dříve s vedoucím mého projektu domluvili, programovací jazyk, jehož znalosti budu aplikovat je C#. Důvodem výběru tohoto programovacího jazyka je jeho vlastnost jednoduchého a zároveň širokého spektra využití v právě dané oblasti.

### 2.1.1. Programovací jazyk C#

C# je moderní, vysokoúrovňový a objektově orientovaný programovací jazyk jehož vývoj vznikl na základě jeho předchůdce, jimž je jazyk C, který je, dalo by se říct, základní verzí jazyku C# a podporuje především procedurální programování, zatímco jazyk C# navíc umožňuje programování objektově orientované. Mezi další přednosti jazyka C# patří dostupnost více typy platform mezi sebou nebo také například automatické uvolňování paměti na základě metody garbage collection, která uvolňuje paměť tak, že z ní vymaže data procesů, které již nejsou aktuální, ale stále běží v pozadí. Výhodou volby tohoto programovacího jazyka může pro programátora do budoucna být znalost základů algoritmizace nebo také znalost již podobných programovacích jazyků jako je například již dříve zmíněný jazyk C nebo také jazyk C++. (Microsoft, 2020)

## 2.2. Volba vývojového prostředí

Ačkoliv existuje několik typů, kde je možné uplatňovat své programátorské zkušenosti a vyvíjet tak software, již díky předchozím zkušenostem s vývojovým prostředím Visual Studio tak pro mě právě toto prostředí byla jasná volba. Prvními z vlastností, které mě na Visual Studiu zaujaly byly vzhled a ovládací panely s rozsáhlým množstvím ovládacích prvků ulehčujících samostatnou orientaci a práci programátora v tomto prostředí.

Další z předností Visual studia, která mě postupem času překvapila byla množina chybových hlášek, kterou toto prostředí poskytuje. To může programátorovi ulehčit práci a zároveň ušetřit čas při hledání příčiny nastání chyby při kompilaci kódu. Programátor je tedy informován a nasměrován na část kódu, která brání jeho kompilaci. Tyto chybové hlášky navíc obsahují užitečné informace o tom, jaký typ chyby je příčinou, tedy proč se kód nedaří přeložit tak, aby mu překladač porozuměl a zároveň jsou nabízeny automatické možnosti řešení tohoto problému.

V neposlední řadě programátory oceňovaný aspekt tvoří také rozsáhlé množství takzvaných .NET knihoven, které ulehčují práci svojí schopností zpřístupnění mnoha různých technik programování, a to za pomocí využívání již někým vytvořených a publikovaných funkcí.

### 2.3. Volba služby

Pro můj projekt, tedy vývoj aplikace jsem po veškerém porovnávání výhod a nevýhod typů aplikačních služeb a jejich zvažování zvolil službu gRPC pro její moderní a rychlý způsob zprostředkování komunikace dvou nebo i více objektů mezi sebou a také její jednoduchou možnost přizpůsobení si konfigurace podle vlastních potřeb pomocí protokolových souborů, ve kterých je možné si definovat požadované vstupy a dále také výstupy jednotlivých komunikačních žádostí ale také jejich odpovědí, jak lze vidět v části ukázkového kódu níže.

```
syntax = "proto3"; option
csharp_namespace = "GrpcZdravic";
package pozdrav; service Zdravic {
  rpc Pozdrav (PozdravZadost) returns (PozdravOdpoved);
}
message PozdravZadost {
string jmeno = 1;
}
message PozdravOdpoved {
string zprava = 1;
}
```

## 2.4. Volba komunikačního protokolu

Zvoleným komunikačním protokolem je protokol kompatibilní se zvolenou službou gRPC, jedná se tedy o komunikační protokol gRPC jehož přednostmi jsou již dříve zmíněné formy přenosu dat v binárním formátu a také přenosový protokol který je používán, tedy protokol HTTP/2.

# 3. Vytvoření aplikace pro komunikaci mezi vážními jednotkami.

Po provedení prvotní analýzy a výběru technologií, které budu aplikovat již mám představu, jaká bude struktura mé aplikace a jak začít s vývojem mé aplikace. V této části jsou zaznamenány a popsány kroky mého postupu při aplikaci technologií, které jsem zvolil v předchozím bodě.

## 3.1. Počáteční verze aplikace

Pro testování komunikace dvou objektů, kterými v budoucnu, tedy v praxi budou počítač, ke kterému je připojena váha a počítač, na kterém poběží server nezbývá než danou komunikaci simulovat, tedy vytvořit aplikaci představující jednotlivé vstupy a výstupy obou objektů. Začal jsem jednoduchou verzí, kde jsou oba objekty zastupovány jako konzolové aplikace, tedy aplikace s příkazovou řádkou. Tento typ aplikace ovšem nevyužívá grafické rozhraní.

### 3.1.1. Serverové služby

Server má implementovanou službu *ZískejInfo* jejímž výstupem je časové razítko a také verze aplikace, která je prozatím ve formátu generovaného řetězce.
*Zde můžete vidět ukázkový kód.*

```
namespace GrpcServer.Services
{
    public class InfoSluzba : Info.InfoBase
    {
        public override Task<InfoReply> ZiskejInfo(InfoRequest request,
ServerCallContext context)
        {
            InfoRequest vystup = new InfoRequest();
Random RNG = new Random();
string cas = DateTime.Now.ToString("t");
string verze = "v2." + RNG.Next(1, 9);
vystup.Verze = verze;
vystup.Cas = cas;
             return Task.FromResult(new InfoReply
            {
                Zprava= "Verze serveru je: " + vystup.Verze + "\n" + "Čas: " +
vystup.Cas             });
        } } }
```

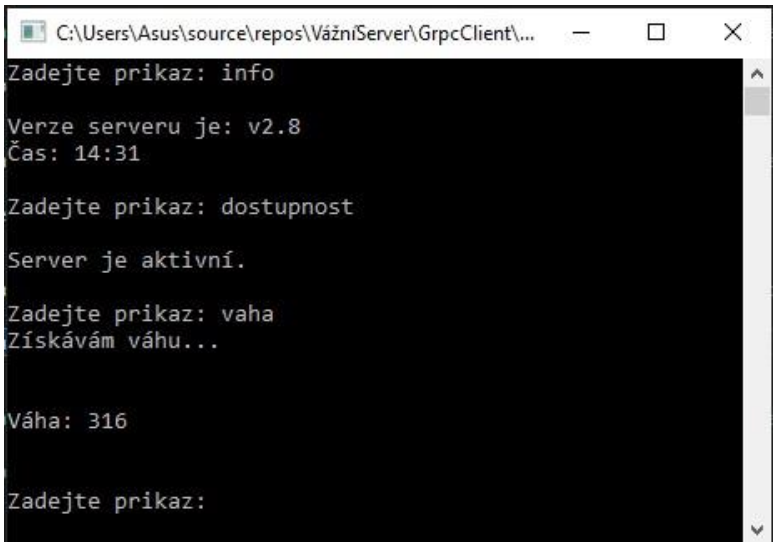
Dále služba *ZjistiDostupnost,* která ověřuje dostupnost serveru například ještě před prvotním připojením klientského serveru, ale také v jeho průběhu a v poslední řadě lze také využít služby *ZiskejVahu* jejímž cílem v budoucnu bude předat naměřený parametr z právě příslušné váhy fyzicky připojené k počítači pomocí COM portu, ovšem pro mé testování prozatím postačí

generování čísla představující naměřenou váhu tato funkce by měla být vykonána s časovou prodlevou, která má simulovat provoz v praxi, tedy čekání na vrácení váhy z vážního indikátoru což nemusí být okamžité, a právě to je funkcí této služby.

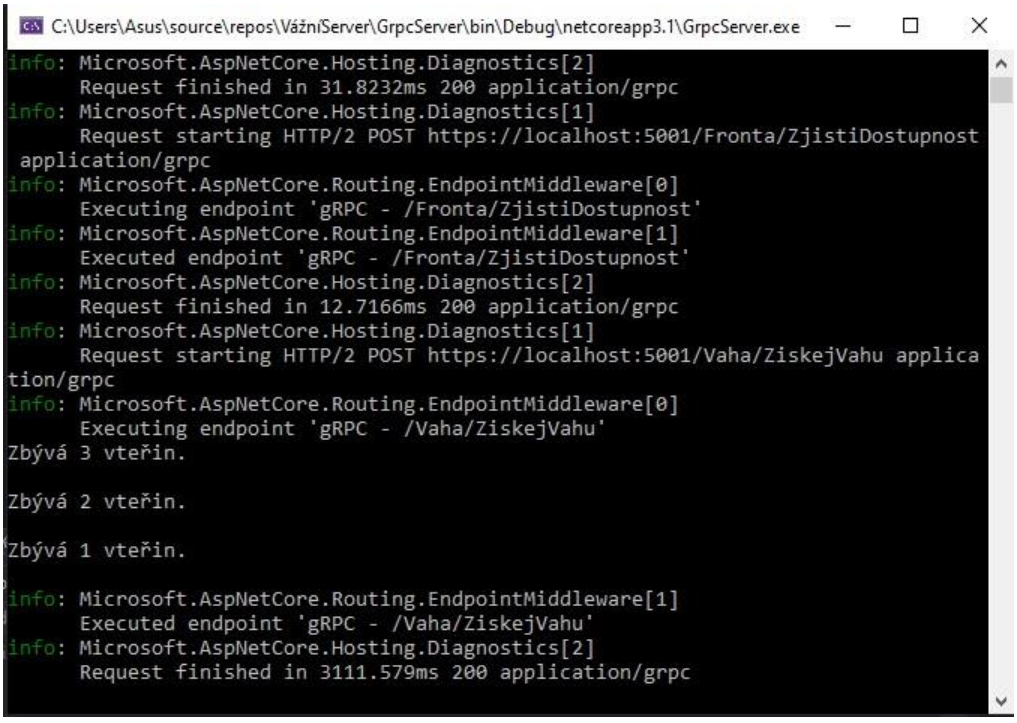
*Zde můžete vidět ukázkový kód.*

```
namespace GrpcServer.Services
{
    public class VahaSluzba : Vaha.VahaBase
    {
        public override async Task<VahaReply> ZiskejVahu(VahaRequest request, Server-
CallContext context)
        {
VahaRequest vystup = new VahaRequest();
Random RNG = new Random();
vystup.Vaha = RNG.Next(1, 999);
int zpozdeni = 3*1000;
while(zpozdeni!=0)
            {
try {
                    cts.Cancel();
                   cancel.ThrowIfCancellationRequested();
                    Console.WriteLine("Zbývá {0} vteřin.\n", zpozdeni /
1000);
await Task.Delay(1000);
zpozdeni = zpozdeni - 1000;
                }
                catch(OperationCanceledException)
{ Console.WriteLine("zastaveno"); }
            }
            return new VahaReply
            {
                Zprava = "Váha: " + vystup.Vaha + "\n"
            }; }
```

Níže lze vidět záznam komunikace serveru s klientským serverem s použitím protokolu gRPC. Pokud běží server i klient, server čeká na příkaz k vykonání služby, která je v něm implementována, tedy za předpokladu správného předání parametrů (v tomto případě jen název příkazu). Server vykoná službu a výstupní parametry jsou zobrazeny na straně klientského serveru.



Obrázek 7: Klientský server–konzole

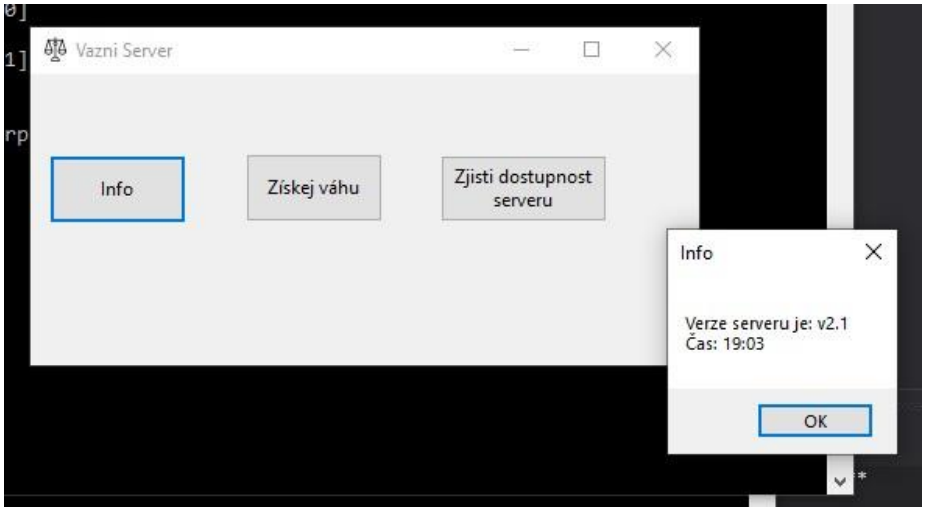


Obrázek 8: Server–konzole

## 3.2. WinForm verze aplikace

V další fázi vývoje jsem aplikaci přiblížil simulaci provozu, a to přizpůsobením aplikace, konkrétně klientského serveru, který jsem vytvořil jako aplikaci s grafickým rozhraním knihovny Windows Forms s ovládacími prvky jako jsou například tlačítka, prvky s textovými poli a také je zde možnost použití vyskakovacích oken zobrazujících informující zprávy, takzvaný MessageBox. Aplikace v této formě je tedy uživatelsky příznivější v porovnání s pouhou konzolovou aplikací, jelikož je možné grafické oddělení a přizpůsobení jednotlivých ovládacích a informačních prvků kterých je zde využíváno. Spuštění jednotlivých služeb

serveru je zde zastoupeno právě tlačítky, které obstarají volení dané služby a zobrazení výstupů těchto služeb zároveň viz obrázek níže.

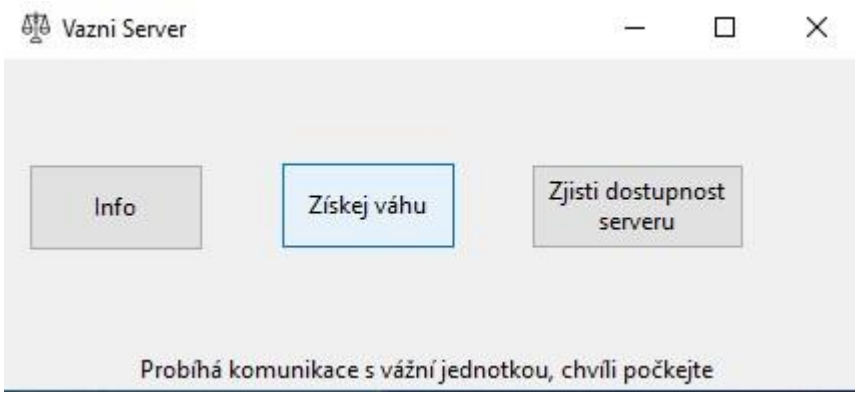



Obrázek 9: Klientský server-WinForm aplikace

### 3.2.1. Uživatel a zpětná vazba

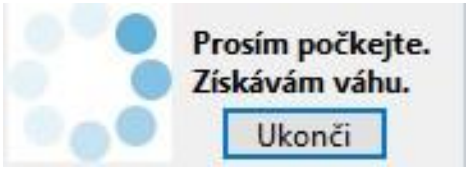
S postupem času a rostoucím objemem používání komunikačních technologií, které uživatelé využívají k uspokojení potřeb jejich každodenního života je nezbytné zařídit, aby se k uživateli dostala zpětná vazba v podobě dat. Místo toho, aby uživatelé museli informace vyhledávat teď chtějí, aby se k nim informace dostaly, jakmile budou vygenerovány. (AGUILAR, 2014)

Po dobu čekání na odpověď serveru, tedy vykonání dané služby by měl být uživatel informován, a to v podobě zpětné vazby obsahující informaci o dané situaci. Takovým typickým příkladem může být informační hláška „Prosím počkejte “. To lze řešit například jednoduchým způsobem jako je použití TextBoxu, jedná se tedy o formu zobrazení textu, který uživatele sice informuje, ovšem grafický vzhled nemusí být ideální.




Obrázek 10: Klientský server-zpětná vazba

Lepším způsobem takovéto zpětné vazby může být vytvoření samostatného objektu tzv WaitForm, který si lze graficky přizpůsobit podle svých představ, a to například vytvořením animace jejíž efekt bude aktivní po dobu zákazníkova čekání na odpověď serveru. Jako vzor můžete níže vidět mnou vytvořený WaitForm jehož animací je načítání.



Obrázek 11: WaitForm

# 4. Vytvoření aplikace pro testování vážního serveru.

Jelikož se jedná o oboustrannou komunikaci, obě strany se tedy budou potýkat s řešením stavu, odezvy, ale také zpětné vazby. Právě podle toho může být server nastaven, ať už inicializován, nebo nastaven pomocí konfigurační aplikace, kterou je potřeba vytvořit a přizpůsobit podle svých potřeb. V této části je tedy popsán postup při tvorbě, obsluze a konfiguraci relací.

## 4.1. Vlastnosti serveru

Na straně serveru musí být řešen stav (v klidu / zaneprázdněn / nastavování apd.). V případě, že je server zaneprázdněn, může tvořit frontu nebo další dotazy odmítat. Jedná se tedy o metodu semaforu sloužící k řízení provozu, například v podobě schvalování připojení, tedy zahajování komunikačních relací, které je nutno programátorem ošetřit, jelikož je to jeden z faktorů ovlivňujících samostatné chování aplikace.

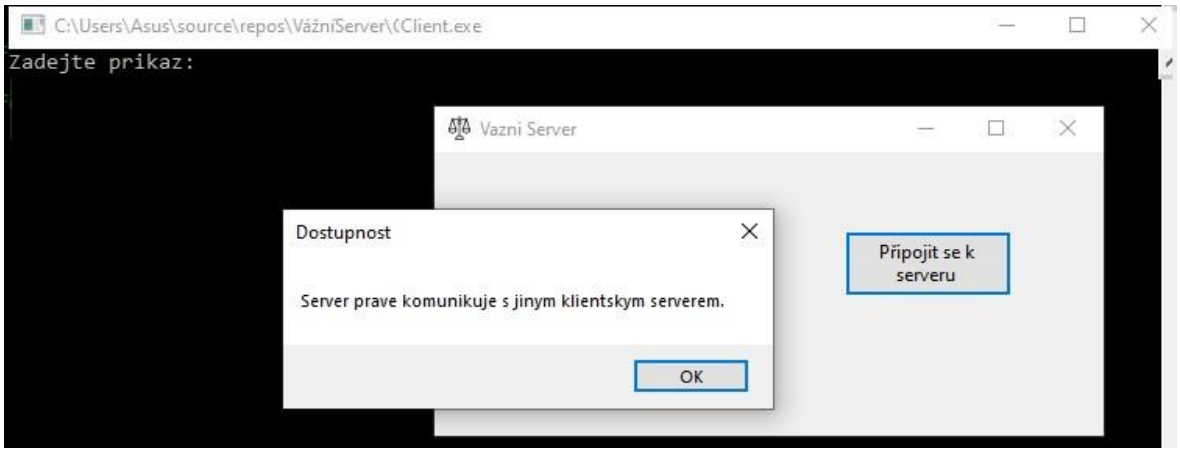
*Zde můžete vidět ukázkový kód.*

```
class Stav
{

public const string AKTIVNI = "Aktivní";
public const string VAHA = "Získávám váhu";
public const string INFO = "Získávám info";
public const string NASTAVUJI = "Nastavuji";
        }
            class ServerJe
        {

public const int VOLNY = 0;
public const int AKTIVNI = 1;
public const int NASTAVOVAN = 2;
 }


namespace GrpcServer.Services
{
 public class FrontaSluzba : Fronta.FrontaBase
{
        class ServerJe
        {
public const int VOLNY = 0;
public const int AKTIVNI = 1;
public const int NASTAVOVAN = 2;
        }
        public override Task<FrontaReply> ZjistiRelace(FrontaRequest request,
        ServerCallContext context)
        {
            return Task.FromResult(new FrontaReply
            {
                Relace = Nastaveni.relace;
            }) ; }
```

Mým řešením byla implementace služby serveru zabývající se stavem serveru a dvou příslušných funkcí, funkce *NastavDostupnost()*, která změní stav serveru při zahájení a také ukončení relace a funkce *ZjistiDostupnost()*, která daný stav serveru zjistí a tato informace je tedy návratovou hodnotou této funkce. Uživatel se v aplikaci musí připojit k serveru pomocí tlačítka, které volá funkci zjišťující dostupnost serveru, v případě, že server komunikuje s jiným klientským serverem, relace není zahájena a uživatel je upozorněn zprávou.
*Zde můžete vidět ukázkový kód.*

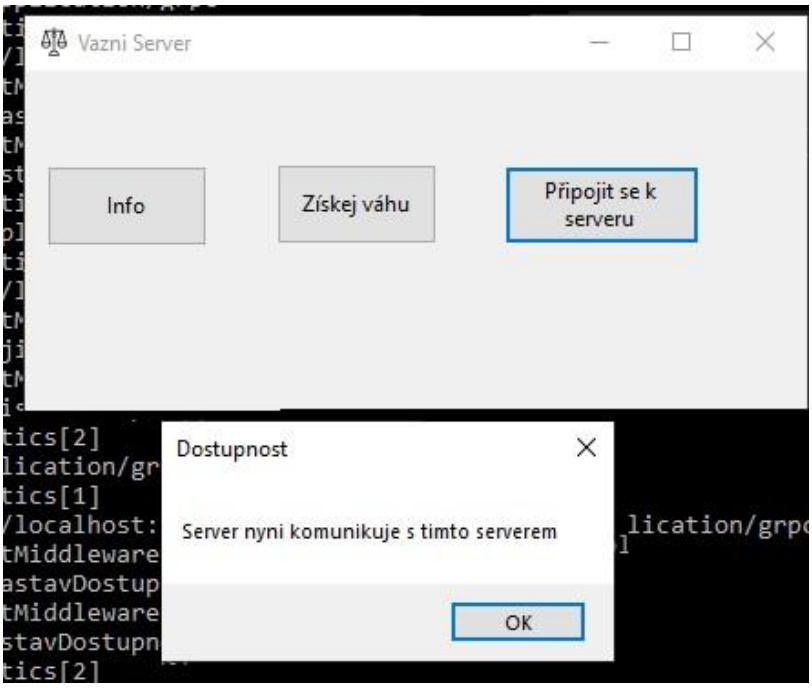
```
public override Task<FrontaReply> ZjistiDostupnost(FrontaRequest request,
        ServerCallContext context)
        {
            int obsazeno = Nastaveni.Dostupnost();
            if (obsazeno == ServerJe.VOLNY)
            {
                return Task.FromResult(new FrontaReply
                {
                    Obsazeno = ServerJe.VOLNY,
                    Zprava = "Server je dostupny."
                });
            }
            else if (obsazeno == ServerJe.AKTIVNI)
            {
                    Nastaveni.NastavDostupnost(ServerJe.AKTIVNI);
return Task.FromResult(new FrontaReply
                    {
                        Obsazeno = ServerJe.AKTIVNI,
                        Zprava = "Server je aktivní."
                    }); }
 else {
                return Task.FromResult(new FrontaReply
                {
                 Obsazeno = ServerJe.NASTAVOVAN,
                 Zprava = "Funkce serveru nejsou dostupné, server se právě nastavuje."
                });
            } }
```

Právě to jsem otestoval při souběžném spuštění všech 3 vytvořených objektů, tedy serveru, klientského serveru v podobě konzolové aplikace a klientského serveru v podobě aplikace WinForm. Je-li zahájena relace ze strany konzolového klienta pak druhému klientovi není umožnění zahájení relace a funkce serveru tedy nejsou zpřístupněny, to je možné až po ukončení předchozí relace serveru, přičemž se změní stav serveru na *volný*.



Obrázek 12: Stav serveru–obsazený

Pokud je server dostupný, dojde k připojení a uživatel má dostupná tlačítka pro volání služeb serveru.



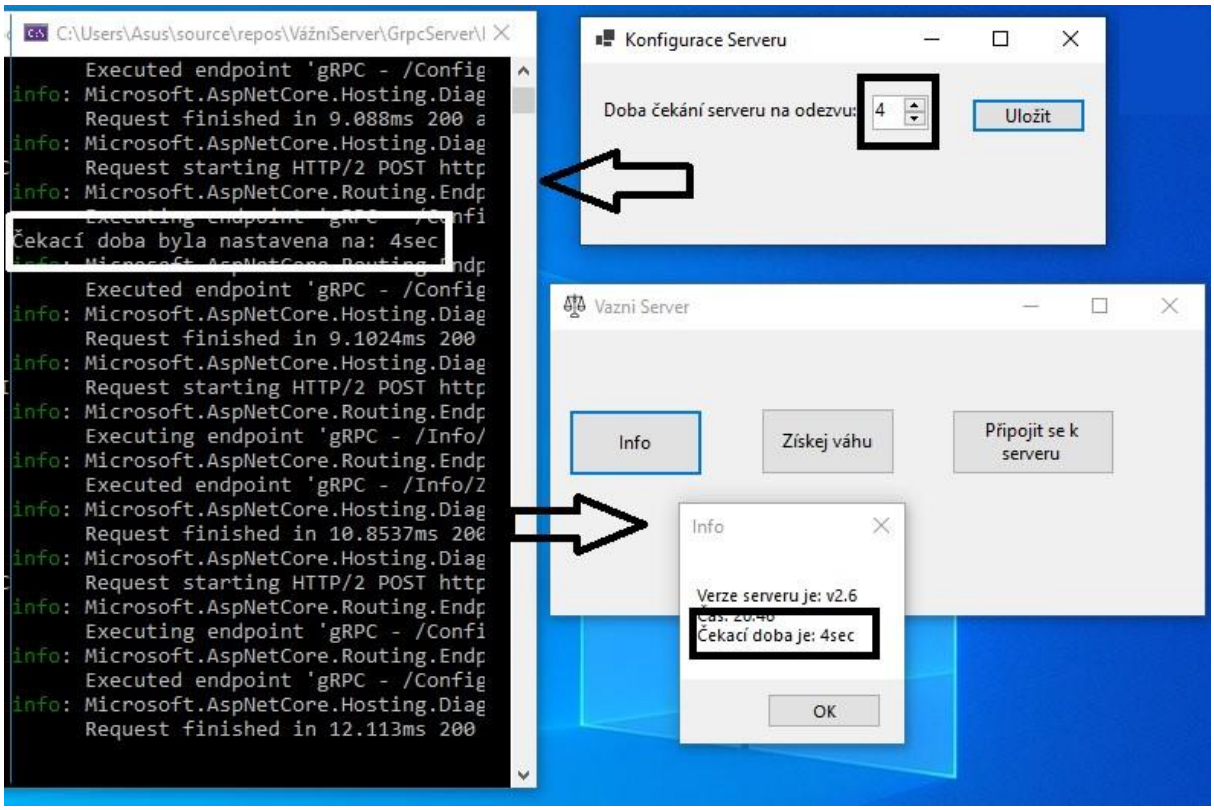
Obrázek 13: Stav serveru–volný

## 4.2. Konfigurace serveru

Dalším faktorem, jehož ošetření je nutno zvážit je plynulý chod aplikace při kterém nedojde k neošetřené výjimce v podobě spadnutí nebo zamrznutí aplikace. Po dobu čekání na odpověď by měl mít uživatel možnost toto čekání na odezvu ukončit v případě, že již nechce čekat a chce nadále pracovat v aplikaci.

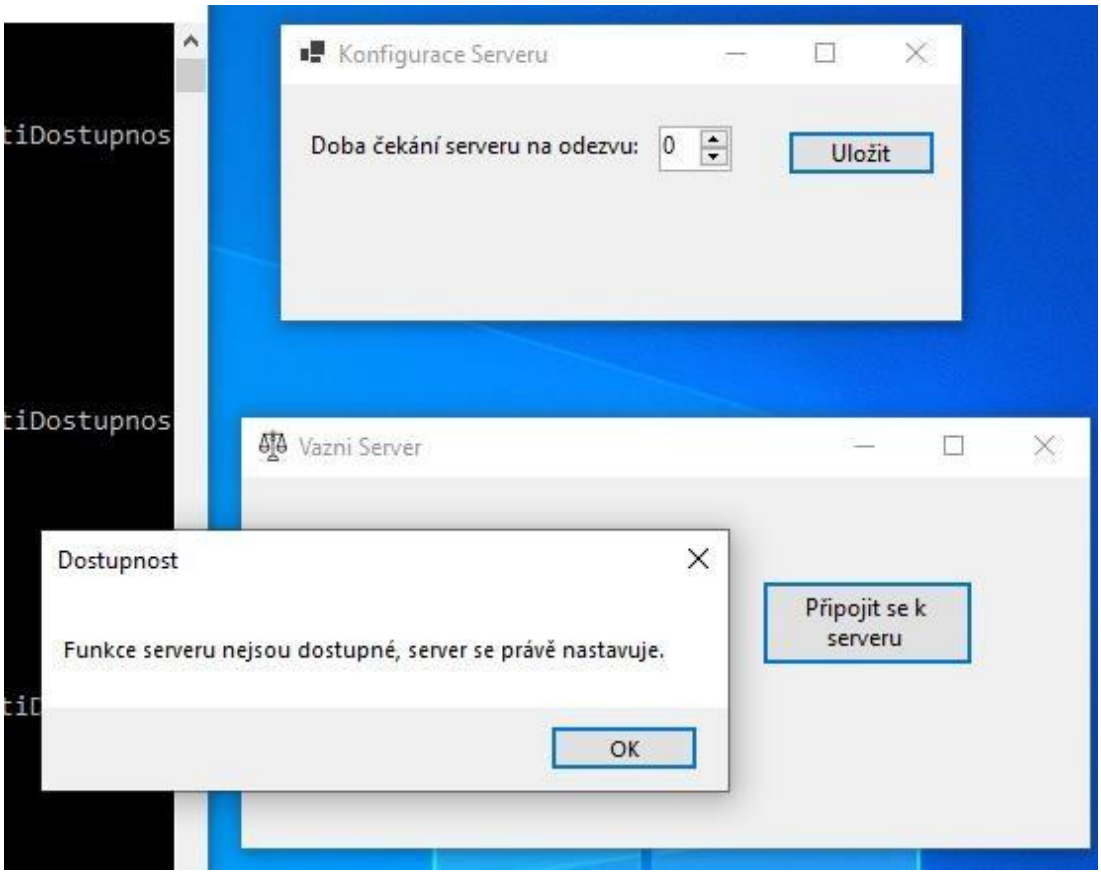
Takový proces ukončení čekání na odpověď se dá také automatizovat nastavením čekací doby, tedy časového intervalu, po který chceme čekat a nedokončí-li se daná operace, čekání bude

samovolně zastaveno, a právě to jsem si v mé aplikaci také simuloval a vyzkoušel, a to nejprve pomocí konstanty představující čekací dobu na odezvu, později pak pomocí konfigurační aplikace, kde má uživatel možnost si tuto čekací dobu na odezvu změnit. Ze strany konfiguračního souboru se vyšle komunikační protokol obsahující námi nastavenou dobu čekání na odezvu serveru. Server si hodnotu uloží. Klientský server na žádost získá tuto hodnotu a dále s ní porovnává, zda byla služba vykonána v daném časovém intervalu. Toto nastavení je uloženo do textového souboru na straně klienta a je načteno při spuštění serveru.



Obrázek 14: Konfigurační soubor

Při spuštění config aplikace serveru je okamžitě odeslána žádost a stav serveru je změněn na *nastavování*, tudíž po dobu, co je tato config aplikace otevřena není stav serveru *volný* a klientský server tedy nemá dostupné funkce, vyskočí chybová hláška. Inicializoval jsem událost *FormClosed* která v okamžiku, kdy je tato config aplikace zavřena znovu vyšle žádost a stav serveru je změněn na *volný* tudíž až pak je na straně klientského serveru opět možné volat služby serveru běžným způsobem.

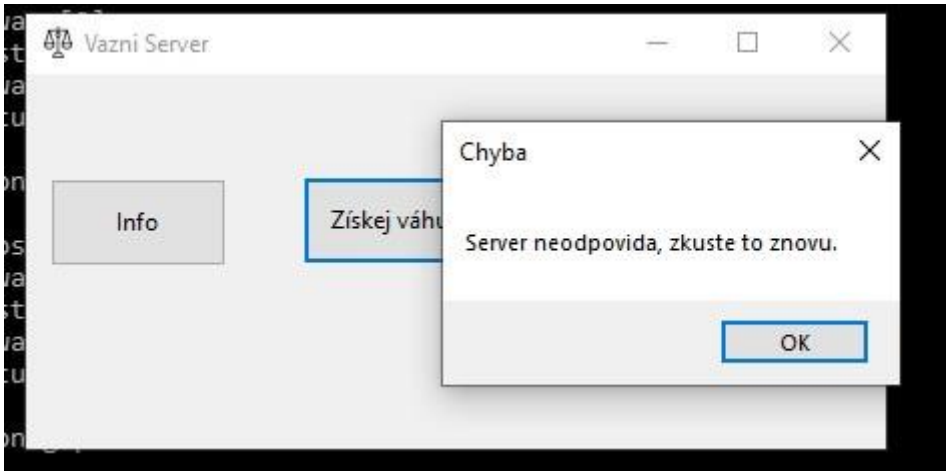


Obrázek 15: Stav nastavování

Funkce *ZískejVáhu* je implementována s časovým zpožděním tří vteřin, pokud je tedy čekací doba na odezvu nižší, od serveru nebude získána odpověď v určitém časovém intervalu, uživatel je o této skutečnosti informován a čekání je ukončeno, ovšem je možné službu volat znovu, případně volat jiné služby serveru v aplikaci. Způsob implementace této funkce lze vidět na následující části zdrojového kódu.

```
        async Task ZiskejVahu()
        {
source = new CancellationTokenSource();
token = source.Token;
try
            {
                if (Nastaveni.preruseni == true)
                    {
                        source.Cancel();
                    }
                    await Task.Run(() =>
                     {
                         vaha = VahaClient.ZiskejVahu(VahaZadost);
                     }, token);
                if (Nastaveni.preruseni != true && Nastaveni.status==Stav.VAHA)
                {
                    cf.Hide();
                    MessageBox.Show(vaha.Zprava, "Váha");
                    log.Info("Operace " + Stav.VAHA + " byla dokončena.");
```

```
                }
}
            catch (OperationCanceledException )
            {

            }
        }

        private async void Vaha_Click(object sender, EventArgs e)
        {
            ZmenStatus(Stav.VAHA);
log.Info(Stav.VAHA);
await Cekej();
await ZiskejVahu();
ZmenStatus(Stav.AKTIVNI);
        }
```




Obrázek 16: Odpověď serveru-chyba

V opačném případě neboli úspěšném získání odpovědi serveru v určitém časovém intervalu (v mém případě ladění programu to bylo prodloužení čekací doby) je zobrazena odpověď serveru běžným způsobem.

# 5. Simulace vážního provozu.

V této části naleznete popis a ukázkové části zdrojového kódu, který byl použit k vytvoření aplikace.

## 5.1. Struktura serveru

Kromě samostatné inicializace jednotlivých protokolových souborů je také nezbytné, aby byl server obeznámen s faktem, že s nimi bude pracovat, a právě to je potřebné serveru sdělit v jeho konfiguračním souboru pomocí následujícího kódu.

```
<ItemGroup>
  <Protobuf Include="Protos\fronta.proto" GrpcServices="Server" />
  <Protobuf Include="Protos\config.proto" GrpcServices="Server" />
  <Protobuf Include="Protos\vaha.proto" GrpcServices="Server" />
  <Protobuf Include="Protos\info.proto" GrpcServices="Server" />
</ItemGroup>
```

### 5.1.1. Soubor startup.cs

Jelikož se softwaroví vývojáři potýkají s širokým spektrem problémů, existuje také mnoho řešení. Pokud tedy chcete začít s vytvářením řešení pro danou problematiku, je možné, že se již právě s danou, nebo obdobnou problematikou zabýval někdo před vámi. Právě proto ve vývojových prostředích existuje velké množství .NET knihoven společně s NuGet balíčky, které nejenže poskytují možnost aplikace veškerých technologií, ale také redukují objem zdrojového kódu, který musíte napsat. Tyto knihovny je nutné dle potřeby a volby nainstalovat a pro jejich správný chod, stejně jako v předchozím bodě, obeznámit server s jejich používáním, a to zejména na začátku souboru.

```
using System;
using System.Collections.Generic;
using System.Linq; using
System.Threading.Tasks; using
GrpcServer.Services; using
Microsoft.AspNetCore.Builder;
using
Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

Obdobně, jako u protokolových souborů musí být server propojen s jeho službami, a to pomocí koncových bodů následovně.

```
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapGrpcService<InfoSluzba>();
endpoints.MapGrpcService<FrontaSluzba>();
endpoints.MapGrpcService<VahaSluzba>();
endpoints.MapGrpcService<ConfigSluzba>();

                endpoints.MapGet("/", async context =>
                {
                    await context.Response.WriteAsync("Communication with gRPC
endpoints must be made through a gRPC client. To learn how to create a client,
visit: https://go.microsoft.com/fwlink/?linkid=2086909");
                });
            });
```

### 5.1.2. Soubor nastaveni.cs

V následující části kódu ze souboru nastaveni.cs lze vidět inicializaci proměnných s metod, se kterými program pracuje zejména v oblastech tvoření relací, změny stavu serveru, a také při konfiguraci čekací doby, jejíž vstupní parametr pochází z konfigurační aplikace a její žádosti o změnu tohoto parametru na straně serveru.

```
    public class Nastaveni
    {

public static int dostupnost;
public static int relace;
public static int cekaci_doba = 4;

        public static void NastavDobu(int zadost)
        {
            cekaci_doba = zadost;
        }
        public static int CekaciDoba()
        {
                return cekaci_doba;
        }


        public static void NastavDostupnost(int zadost)
        {
            dostupnost = zadost;
        }
        public static int Dostupnost()
        {
            return dostupnost;
        }
```

### 5.1.3. Svazek protokolových souborů

**Konfigurační protokol**

V struktuře souboru tohoto protokolu jsou inicializované 3 funkce, přičemž obě vstupní i návratové hodnoty jsou číselného typu, dáváme tedy serveru vědět, o jaký typ žádosti se jedná a následné zpracování jednotlivých žádostí je řešeno na straně serveru. Funkce konfigurační aplikace *ZjistiNastaveni* vyšle žádost serveru, návratovou hodnotou je parametr naposledy uložené čekací doby, která dále může být změněna použitím funkce *Nastav*.

```
syntax = "proto3";

option csharp_namespace = "GrpcServer";

service Config {

rpc ZjistiNastaveni (ConfigRequest) returns
(ConfigReply); rpc Nastav (ConfigRequest) returns
(ConfigReply); rpc ZjistiRelace (ConfigRequest) returns
(ConfigReply);

}  message ConfigRequest {  int32 cekaci_doba = 1;
}

message ConfigReply {  int32 cekaci_doba = 1; }
```

**Semaforový protokol**

Jelikož je na straně serveru nutné řešit stav, je možné, že nastane stav, kdy bude server obsazen, a právě to se ověřuje ještě před zahájením nové relace pomocí funkce *ZjistiDostupnost*. Pokud je server prázdný, podaří se tedy zahájit relaci pomocí funkce *NastavDostupnost* a počet relací je na straně serveru zvýšen o 1 a lze také nastavit obsazení serveru při dosažení určitého počtu relací. V případě komunikace s objektem, který není běžný, tedy není váha, jako je například konfigurační aplikace může být nezbytné předání parametru obsahující tuto informaci jako je to například v mém případě pomocí parametru *nejsem_vaha* dávám tedy serveru vědět, že se nejedná o běžnou komunikaci, nýbrž o konfiguraci a v tuto chvíli je změněn stav serveru na *nastavován* a obdobně jako u obsazenosti serveru není možné tvoření dalších relací.

```
syntax = "proto3";

option csharp_namespace = "GrpcServer";

service Fronta {
```

```
    rpc ZjistiDostupnost (FrontaRequest) returns (FrontaReply);
   rpc NastavDostupnost (FrontaRequest) returns (FrontaReply);
   rpc ZjistiRelace (FrontaRequest) returns (FrontaReply);
   }
message FrontaRequest {
   int32 obsazeno=1;
   int32 nejsem_vaha=2;
   }  message FrontaReply {   string zprava = 1;   int32 obsazeno=2;   int32 Relace=3;
   }
```

**Info protokol**

RPC služba *ZiskejInfo* zprostředkovává předání verze aplikace a aktuálního času klientskému serveru.

```
   syntax = "proto3";
   option csharp_namespace = "GrpcServer";

   service Info {
   rpc ZiskejInfo (InfoRequest) returns (InfoReply);
   }
   message InfoRequest{ string verze = 1;  string cas = 2;
   }
   message InfoReply {
   string zprava=1; }
```

### 5.1.4. Služby serveru

**Konfigurační služba**

Předání a následné uložení změny parametru probíhá pomocí volání služeb se vstupními parametry právě obsluhované žádosti.

```
namespace GrpcServer.Services
{
    public class ConfigSluzba : Config.ConfigBase
    {
        public override Task<ConfigReply> ZjistiNastaveni(ConfigRequest request,
        ServerCallContext context)
        {
            int cekaci_doba=Nastaveni.CekaciDoba();
            {
                return Task.FromResult(new ConfigReply
                {
                    CekaciDoba=cekaci_doba
                });
                }
}
        public override Task<ConfigReply> Nastav(ConfigRequest request,
        ServerCallContext context)
        {
            int cekaci_doba=request.CekaciDoba;
            Nastaveni.NastavDobu(cekaci_doba);
```

```
            Console.WriteLine("Čekací doba byla nastavena na: "
+              Nastaveni.CekaciDoba() + "sec");
```

**Semaforová služba**

V části kód níže jsou řešeny všechny stavy serveru, které mohou nastat. Je zde také řešena problematika relací, při zahájení je zvýšen počet relací o 1 a naopak při ukončení relace je tento celkový počet snížen. Pro přehlednost a ulehčení psaní kódu je pro programátora vhodné si nejprve pro každý stav inicializovat konstantu. Nebude tedy třeba stále opisovat název proměnné příslušné danému stavu, ale postačí jen znak. (viz. 0 namísto volný)

Následné řešení jednotlivých stavů pak závisí na programátorovi a jeho uvážení.

```
public override Task<FrontaReply> NastavDostupnost(FrontaRequest request, Ser
verCallContext context)
        {
            int zadost = request.Obsazeno;
             if (zadost == ServerJe.AKTIVNI) { //žádost o zahájení relace
                Nastaveni.relace++;
             }
            if (zadost == ServerJe.VOLNY && request.NejsemVaha!=1)
            //žádost o ukončení relace
            {
                Nastaveni.relace--;
if (Nastaveni.relace != 0)
                {
                    zadost = ServerJe.AKTIVNI;
                }
            }
                Nastaveni.NastavDostupnost(zadost);
```

## 5.2. Struktura klientské aplikace

Jelikož klientský server používá stejné protokolové soubory jako server, je potřeba mít tyto identické soubory i zde a zároveň tomu musíme přizpůsobit strukturu hlavního souboru této aplikace následovně.

```
  <ItemGroup>
    <Protobuf Include="..\GrpcServer\Protos\config.proto" GrpcServices="Client">
       <Link>Protos\config.proto</Link>
    </Protobuf>
    <Protobuf Include="..\GrpcServer\Protos\fronta.proto" GrpcServices="Client">
      <Link>Protos\fronta.proto</Link>
    </Protobuf>
    <Protobuf Include="..\GrpcServer\Protos\info.proto" GrpcServices="Client">
      <Link>Protos\info.proto</Link>
    </Protobuf>
    <Protobuf Include="..\GrpcServer\Protos\vaha.proto">
      <GrpcServices>Both</GrpcServices>
      <Link>Protos\vaha.proto</Link>
    </Protobuf>
  </ItemGroup>
```

### 5.2.1. Soubor uživatelského formuláře aplikace

**Technika logování**

Logování slouží k záznamu dat, které vznikají při chodu aplikace. Tyto data slouží zejména programátorovi, ale také i uživateli ke zpětné revizi chodu programu, a to pomocí takzvaného loggeru, který je potřeba definovat a také mu přiřadit způsob zápisu dat, který byl v mém případě zápis do textového souboru.

Ukládání loggeru lze najít v následujících částech kódu pomocí příkazu *log*. Dále zde můžete také najít strukturu jednotlivých funkcí používaných pro volání serverových služeb.

```
        ILog log = LogManager.GetLogger(typeof(VazniServer));

        public VazniServer()
        {

log.Info("Aplikace byla načtena");
        }
      int Nastav()
        {

var adresa = GrpcChannel.ForAddress("https://localhost:5001");
var ConfigZadost = new ConfigRequest { };
var ConfigClient = new Config.ConfigClient(adresa);
int cekaci_doba = ConfigClient.ZjistiNastaveni(ConfigZadost).CekaciDoba;
return cekaci_doba;
        }

          private bool KontrolaOdezvy(Task t)
        {
            bool odezva = t.Wait(Nastav() * 1000);
if (odezva==false)
            {

MessageBox.Show("Server neodpovida, zkuste to znovu.", "Chyba");
return false;
            }
else
{
                return true;
            } }
```

Logger má běžně tři hlavní úrovně záznamu a tyto úrovně jsou spjaty s chodem aplikace. Těmito úrovněmi jsou *info, warning* a *error*, přičemž *info* může zaznamenávat například které operace byly vykonány a jaký byl výsledek, *warning* neboli varování může značit nesprávné používání aplikace, tedy neošetřené výjimky a úrovní *error* pak lze zaznamenat například samovolné ukončení aplikace, ke kterému by nemělo dojít. Tyto úrovně také mají rozdílnou

váhu příslušící významnosti dané situace, která nastala. Ještě vyšší váhu, než error pak má úroveň *critical*. Volně dostupné loggery jako je například logger log4net mohou mít předem-definované možnosti záznamu, jako je odlišení písma pomocí barev na základě závažnosti, chybové záznamy pak budou například červené a informační zelené.

```
        private async void Info_Click(object sender, EventArgs e)
        {
            ZmenStatus(Stav.INFO);
log.Info(Stav.INFO);
await Cekej();
if (Nastaveni.preruseni != true)
            {
                cf.Hide();
                MessageBox.Show(info.Zprava
                    + "\nČekací doba je: " + Nastav() + "sec"
                    + "\nServer má právě: " + relace + " aktivních relací."
                    , "Info");
                log.Info("Operace " + Stav.INFO + " byla dokončena.");
}
            ZmenStatus(Stav.AKTIVNI);

}
        private void ZmenStatus(string status)
{
            Nastaveni.status = status;
            StatusLabel.Text = status;
        }

        async Task Cekej()
        {
                cf.StartPosition = FormStartPosition.CenterScreen;
cf.Show();
                await Task.Delay(1500);
      } } }
```

### 5.2.2. Soubor načítacího formuláře aplikace

Zde můžete vidět zdrojový kód načítacího WinForm formuláře, který je zobrazován po dobu, co uživatel čeká na provedení operace tedy odezvu serveru. Pokud uživatel dále nechce čekat a uzavře tento načítací formulář, operace je přerušena.

```
namespace VazniServer
{
    public partial class CekaciForm : Form
    {
        ILog log = LogManager.GetLogger(typeof(VazniServer));

        private void CekaciForm_Activated(object sender, EventArgs e)
{
            Nastaveni.preruseni = false;
            CekejLabel.Text = Nastaveni.status;
        }
```

```
        private void KonecButton_Click(object sender, EventArgs e)
        {
            Nastaveni.preruseni = true;
this.Hide();
            MessageBox.Show("Operace zrušena\n");
log.Info("Operace zrušena");

        } } }
```

### 5.2.3. Struktura konfigurační aplikace

Pokud je konfigurační aplikace spuštěna, dává o tom vědět serveru a zároveň předává parametr *NejsemVaha*, kterým serveru dává vědět, aby zamezil vytváření dalších relací.

V uživatelském formuláři konfigurační aplikace bylo možné změnit čekací dobu, která se prostřednictvím komunikačního protokolu změnila také na straně serveru.

```
namespace NastaveniServeru
{
    public partial class Form1 : Form
    {
        private async void UlozButton_Click(object sender, EventArgs e)
        {
            int doba = Convert.ToInt32(DobaBox.Value);
await ConfigClient.NastavAsync(ConfigZadost);
        }
        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            var adresa = GrpcChannel.ForAddress("https://localhost:5001");
var FrontaClient = new Fronta.FrontaClient(adresa);
            var FrontaZadost = new FrontaRequest { Obsazeno = 0, NejsemVaha = 1 };
            FrontaClient.NastavDostupnostAsync(FrontaZadost);
        } } }
```

# 6. Vyhodnocení vytvořeného řešení

Výstupem mé praxe, během které jsem vyvíjel software byla WinForm aplikace, která posloužila společnosti jako start-up projekt, který byl dále kompletován a jehož aktuální verze zprostředkovává propojení vážních jednotek na straně jedné, a software, tedy aplikace, společnosti Inisoft na straně druhé. Tato aplikace je tvořena především dvěma objekty, a to serverem obsluhujícím veškeré služby, které má implementované a klientským serverem v podobě aplikace, která tyto služby serveru pomocí implementovaného grafického rozhrání může volat. Komunikace probíhá pomocí moderní technologie gRPC. Možným zlepšením této aplikace může v budoucnu být například propojení zaznamenaných dat s příslušnou databází, čímž by se evidovala příslušná data navážených odpadů.

# Závěr

Činnost ve společnosti mi přinesla mnoho nových zkušeností a znalostí zejména v oblasti vývoje komunikačních technologií, ale také mi umožnila představit si, co obnáší pozice asistenta programátora, respektive vývojáře aplikací. Důležité bylo práci si vhodně časově a krokově rozvrhnout, a právě tím jsem se naučil organizaci práce, k tomu mi také napomáhal web Confluence zprostředkovávající komunikaci s mým vedoucím. Mohl jsem si zde zaznamenávat postup práce, ale také kontrolovat zpětnou vazbu a postupně zdokonalovat mou aplikaci, a právě tyto záznamy byly mými hlavními zdroji použitými při tvorbě této bakalářské práce.

# Seznam použité literatury


CAMEA. [online]. Copyright © 1995 [cit. 21.12.2020]. Dostupné z: https://www.camea.cz/cz/doprava/vazeni-za-jizdy-wim/

CORE TECHNOLOGIES CONSULTING.2020. *What's the difference between a Windows service and a Web service?* | The Core Technologies Blog. Experts in Windows Services on Windows 10/8 and Server 2019/2016/2012 | Core Technologies Consulting, LLC [online]. Copyright © 2004 [cit. 13.12.2020]. Dostupné z: https://www.coretechnologies.com/blog/windows-services/windows-service-vs-web-service/

ENVATO PTY.2018. *REST vs. gRPC: Battle of the APIs* | Cloudflare [online]. Dostupné z: https://code.tutsplus.com/tutorials/rest-vs-grpc-battle-of-the-apis--cms-30711

GOOGLE.2020. *Cloud Endpoints for gRPC | Cloud Endpoints with gRPC | Google Cloud. Cloud Computing Services* | Google Cloud [online]. Dostupné z: https://cloud.google.com/endpoints/docs/grpc/about-grpc

INISOFT. *O společnosti. INISOFT.CZ (SEPNO, OLPNO, ROČNÍ HLÁŠENÍ O ODPADECH, PORADENSTVÍ, ŠKOLENÍ, ISPOP)* | INISOFT s.r.o. [online]. Dostupné z: https://www.inisoft.cz/spolecnost

MAILCHIMP. *What is a CRM? A Marketer's Guide to Customer Relationship Managemen*t | Mailchimp. All-In-One Integrated Marketing Platform for Small Business | Mailchimp [online]. Copyright ©2001 [cit. 04.12.2020]. Dostupné z: https://mailchimp.com/crm/what-is-crm/

MICROSOFT.2018. *Component Object Model (COM) - Win32 apps | Microsoft Docs.* [online]. Copyright © Microsoft 2020 [cit. 05.12.2020]. Dostupné z: https://docs.microsoft.com/enus/windows/win32/com/component-object-model--com--portal

MICROSOFT.2020. *Get Started - an introduction to the C# language and .NET" | Microsoft Docs. [online].* Copyright © Microsoft 2021 [cit. 06.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/

ORACLE. *What is ERP (Enterprise resource planning)?*. [online]. Copyright © 2020 Oracle [cit. 04.12.2020]. Dostupné z: https://www.netsuite.com/portal/resource/articles/erp/what-iserp.shtml

TENZOVÁHY. *TENZOVÁHY, s.r.o. [online]. Copyright © 2020*, TENZOVÁHY, s.r.o. [cit. 21.12.2020]. Dostupné z: https://www.tenzovahy.cz/