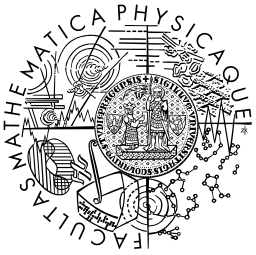


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Milan Truchan

ServIS – webový systém pro firmy zabývající se opravami bagrů

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D.

Studijní program: Informatika

Studijní obor: Programování a vývoj software

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: ServIS – webový systém pro firmy zabývající se opravami bagrů

Autor: Milan Truchan

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Cieľom tejto práce bolo vytvoriť softvérové dielo pre malé firmy zaoberajúce sa zemnými a výkopovými prácami, opravou a predajom bagrov, ktoré nemajú prístup k vhodnému softvérovému riešeniu pre svoju činnosť.

Potreba a správanie funkcionalít bola prekonzultovaná s majiteľom jednej z týchto firiem.

Vzniknutý softvér predstavuje riešenie problému, je schopný zobraziť ponuku (stroje, prídavné zariadenia) a umožňuje užívateľom dopyt (v podobe emailov) na tieto ponuky. V aplikácii tiež existuje aukcia, kde sa dražia opravené bagre. Užívatelia si tiež môžu v aplikácií vytvoriť účet. Vymenované funkcionality môžu využívať prihlásení aj neprihlásení užívatelia. Bežní prihlásení užívatelia nemusia vyplňovať informácie o sebe vo formulároch pri dopytovaní sa na ponuku. Prihlásení admini majú možnosť spravovať stránku. Tj. pridávať nové, editovať a mazať existujúce ponuky, odpovedať na správy atď.

Kľúčová slova: Informačný systém C# .NET Blazor Server MySQL

Title: ServIS – a web system for companies dealing with excavator repairs

Author: Milan Truchan

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek, Ph.D., Department of Distributed and Dependable Systems

Abstract: Abstract.

Keywords: Information system C# .NET Blazor Server MySQL

Obsah

Úvod

Predstavme si malú firmu. Napríklad takú, ktorá práve vznikla. Takáto firma zatiaľ nemá zákazníkov. Ľudia ju nepoznajú. Preto sa firma potrebuje nejakým spôsobom zviditeľniť, napr. prostredníctvom vlastnej stránky. No tvorba stránky môže byť pracná alebo finančne náročná. A kedže sa bavíme o (začínajúcej) malej firme, tak finančný aspekt hraje veľkú rolu. Preto som sa rozhodol vytvoriť softvér, ktorý by umožnil malým firmám prezentovať svoju ponuku strojov a služieb. A to za žiaden alebo minimálny poplatok. Čítateľ by mohol namietať, že v súčasnosti predsa existujú riešenia, ktoré by majiteľom firiem umožnili vytvoriť si vlastný web zdarma. Čítateľ má síce pravdu, no tieto riešenia majú v sebe háčik. Podrobnejšie sa na nich pozrieme neskôr v podkapitole Alternatívne riešenia (0.2). Aby sme videli a pochopili plusy a mínusy alternatívnych riešení a toho môjho, poďme sa najprv pozrieť čo vyžadujú majitelia takýchto firiem.

0.1 Požiadavky na softvér

Po konzultácii s majiteľom jednej z firiem, boli vyhotovené tieto požiadavky:

- **P1 Dostupnosť**

  Softvér by mal byť jednoducho dostupný každému užívateľovi. Či už ide o bežného zákazníka alebo administrátora.

- **P2 Náklady**

  Kedže ide o malé firmy, pri ktorých sa predpokladá nízky rozpočet, chceme, aby náklady spojené s tvorbou a vedením softvéru boli minimálne alebo v ideálnom prípade žiadne.

- **P3 Minimálna obsluha softvéru**

  Systém by mal fungovať a starať sa o seba „sám“. Teda softvér by mal fungovať tak, aby pri ňom nemusel ustavične sedieť človek a obsluhovať ho. Pracovníci firmy, vrátane majiteľa, majú svoju prácu a najímanie si nového pracovníka, ktorý by softvér obsluhoval, nie je z finančných dôvodov žiadúce.

- **P4 Predstavenie ponuky zákazníkom**

  Systém by mal byť schopný prezentovať ponuku bagrov a prídavných zariadení zákazníkom.

- **P5 Aukcia**

  Jednou z činností spomínaných firiem je oprava bagrov. Systém by mal byť schopný poskytnúť administrátorovi možnosť pridať opravený stroj do aukcie.

- **P6 Dopyt**

  Bežnou praxou v tomto odvetví je, že cena strojov sa dopredu neudáva. Zákazník najprv vyjadrí záujem (pošle dopyt), prekonzultujú sa detaily medzi

potenciálnym kupcom a firmou, a až potom prebehne obchod. Z tohto dôvodu systém nebude fungovať na princípe ako bežné internetové obchody (tým myslím pridávanie do košíka s následnou platbou), ale bude fungovať na princípe posielania správ (dopytov). Takže systém by mal umožniť zákazníkom posielať dopyt na položky (stroje, prídavné zariadenia), o ktoré majú záujem.

- **P7 Prístup k súčiastkam strojov**

  Systém by mal umožniť administrátorom jednoducho zistiť, aké náhradné diely obsahuje konkrétny stroj.

- **P8 Registrácia a prihlásenie užívateľov**

  Systém by mal umožniť bežným užívateľom možnosť registrácie a prihlásenia sa do systému. Po prihlásení získajú bežní užívatelia výhodu v tom, že do formulárov už nebudú musieť zadávať svoje údaje.

0.2 Alternatívne riešenia

Teraz, keď už vieme aké sú požiadavky, sa môžeme pozrieť na alternatívne riešenia a zhodnotiť plusy a mínusy.

Jedným z možných riešení by bolo použitie nejakého CMS systému (z ang. content management system), napr. WordPress. Autor práce síce nemá s platformou WordPress žiadne skúsenosti, ale po krátkom hľadaní na internete zistil, že pre túto platformu existuje aukčný plugin. S ním, by bolo dokonca možné na webe prevádzkovať i požadovanú aukciu. Ale toto riešenie by vyžadovalo znalosť platformy WordPress alebo by si majiteľ firmy musel najmúť niekoho, kto túto znalosť má. Kedže znalosť platformy nie je samozrejmesťou a najímanie si niekoho by bolo v rozpore s P2, túto alternatívu môžeme škrnúť.

Pre úplnosť ešte spomeniem, že jedným z riešení by bolo najmúť si inú firmu, ktorá by web vytvorila. No toto riešenie môže byť finančne náročné, a preto je taktiež v rozpore s P2.

0.3 Zhrnutie cieľov

Cieľom tejto práce je implementovať softvérový informačný systém určený pre firmy, ktoré sa zaoberajú predajom a opravou bagrov. Systém bude spĺnať požiadavky P1 až P7.

1. Analýza zadania

V tejto kapitole sa zamyslíme nad tým, ako splniť požiadavky definované v Úvode (0.1).

Pre splnenie požiadavky P1 dáva veľmi dobrý zmysel vytvoriť naše riešenie ako webovú aplikáciu. Týmto spôsobom sa nemusíme starať o distribúciu programu k užívateľom. Stačí ak má zákazník (resp. admin) pripojenie na internet.

Je síce pravda, že voľba webovej aplikácie zahŕňa i voľbu hostingu. A ten nemusí byť lacný. To by mohlo byť v rozpore s P2. Ale je potrebné dodať, že ak by sme zvolili klasickú desktopovú aplikáciu, tak by sme ju museli nejakým spôsobom dodať zákazníkovi. A to by bolo nepraktické, prípadne by mohlo stáť takisto nejaké peniaze. Navyše práve webová aplikácia má potenciál pomôcť firme tak, že ju zákazník objaví pri surfovaní internetu.

Pre splnenie P4, P5 a P8 je jasné, že budeme potrebovať databázu. A to na to, aby si firmy vedeli samé tvoriť ponuku, ktorú si do databázy uložia. Po príchode zákazníka bude možné ponuku z databázy načítať a zobraziť. Podobne v prípade P8. Keď sa užívateľ zaregistruje, jeho údaje sa uložia v databáze a pri prihlásení sa z nej prečítajú a môžu použiť pre vyplnenie formulárov podľa potreby.

Znova sa vrátim k P5. Kedže ide o aukciu, budeme potrebovať nejaký mechanizmus, ktorý by vedel zabezpečiť odpočet, a takisto vyhodnotenie aukcie na pozadí. Taktiež si musíme rozmyslieť, ako sa má aukcia správať v rôznych situáciách.

Na to, aby sme splnili P6, musí byť náš softvér schopný posielať správy. Z podmienky P3 usudzujeme, že nikto nebude pri softvére sedieť, a teda posielanie dopytov by nemalo mať povahu četu. Posielanie správ bude prebiehať prostredníctvom emailov. To nám vytvára novú požiadavku na softvér. Aby administrátor nemusel preklikávať medzi svojou emailovou schránku a naším systémom, bolo by dobre integrovať jeho schránku priamo do systému.

1.1 Voľba typu aplikácie, jazyka a frameworku

Po prejdení požiadaviek vieme, že chceme vytvoriť webovú aplikáciu s bohatým uživateľským rozhraním, ktorá by bola schopná posielať a prijímať správy, pracovať s databázou, umožnila nám autentikáciu a autorizáciu, a taktiež vykonávať prácu na pozadí. Pre túto úlohu sa hodia vysokoúrovňové jazyky, ako sú napríklad C# alebo Java... Na základe autorových skúseností si volíme jazyk C# a platformu .NET, ktorá je s ním spojená.

Platforma .NET nám pre vývoj webových aplikácií poskytuje framework ASP.NET alebo Blazor. Obe frameworky sú si podobné. Rozdiel nájdeme v tom, že Blazor umožňuje vytváranie komponent. Komponent si môžeme predstaviť ako logickú časť stránky (napr. tabuľka, tlačidlo...). Po zadefinovaní komponentu ho vieme „recyklovať“. Tým myslím to, že ho môžeme použiť na viacerých miestach na webe. Na každom mieste sa bude správať a vyzerať rovnako (príp. vieme jeho správanie meniť pomocou parametrov). Táto myšlienka komponentov sa autorovi páči, dobre sa s ňou pracuje a neskôr si ukážeme ako nám pomôže vyriešiť problém s odpočtom.

Blazor poskytuje viacero hosting modelov. V čase rozhodovania existovali dva:

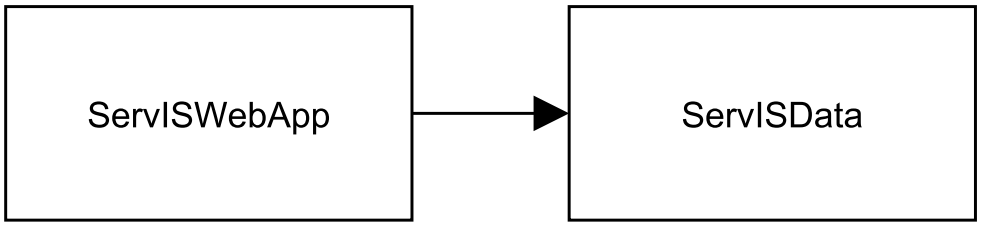
Blazor WebAssembly a Blazor Server. Výber WebAssembly by zahŕňal niekoľko problémov. Pri prvotnej návšteve stránky sa musia klientovi stiahnuť zdrojové kódy aplikácie. To môže chvíľu trvať a mohlo by to odradiť nových potenciálnych zákazníkov. V prípade Blazor Server tento problém nemáme, pretože kód beží na serveri a užívateľovi sa servíruje už len prerenderovaný HTML, CSS, JavaScript kód stránky. Z rovnakého dôvodu sú weby vytvorené Blazor Serverom SEO-friendly (čo znamená, že sú dohľadateľné vyhľadávačmi, akým je napríklad Google). V prípade WebAssembly môžu mať vyhľadávače problém. Na to, aby sa dostali k obsahu stránky si musia obsah vygenerovať zo stiahnutých zdrojových kódov. Ale to môžu mať zakázané z bezpečnostných dôvodov alebo toho nemusia byť schopné. V súčasnosti webové prehliadače (Google Chrome, Mozilla Firefox, ...) podporujú WebAssembly. Ale autor pozorovaním zistil, že vo firmách sa zvyknú využívať staré počítače s potenciálne starým softvérom. Takže hrozí, že by sme mali problém WebAssembly rozbehnúť. Kvôli spomenutým dôvodom si volíme Blazor Server.

1.2 Návrh systému

V Úvode sme rozhodli, že náš systém je webová aplikácia, ktorá pracuje s databázou. Webová aplikácia funguje ako rozhranie pre interakciu s užívateľmi. Databázu potrebujeme kvôli perzistencii dát. Ako vidíme, obe časti poskytujú vlastnú funkcionalitu. Ak tieto časti oddelíme, pomôžeme rozšíriteľnosti systému.

Keď už vidíme, že systém je zložený z dvoch častí, poďme si rozmyslieť ako budú spolu interagovať. Webová aplikácia potrebuje pre svoje fungovanie dáta. Tejto závislosti sa nezbavíme. Lenže dátova časť nepotrebuje webovú aplikáciu pre svoje fungovanie. Závislosť z tejto strany neexistuje.

Majme preto dva projekty: ServISData a ServISWebApp. ServISData slúži ako dátová časť, ktorá je nezávislá a ServISWebApp poslúži ako webová aplikácia, ktorá je závislá na dátach z projektu ServISData. Architektúru demonštruje obrázok 1.1.



Obr. 1.1: Architektúra systému (predstavuje závislosť ServISWebApp od ServISData).

1.3 Voľba databázy

Vieme, že náš systém potrebuje pre splnenie podmienok P4, P5 a P8 databázu. V tejto kapitole si vyberieme typ databázy, databazový server a rozoberieme si aké entity potrebujeme.

1.3.1 Návrh relačného modelu databázy

Z P4 vieme, že potrebujeme entity pre bagre a prídavné zariadenia. Niekoho by mohlo napadnúť spojiť obe entity do jednej, ale to my nespravíme. Ide o rozdielne entity, ktoré môžu uchovávať rozdielne informácie (a ako neskôr v texte uvidíme, skutočne budú uchovávať rozdielne dáta).

Kedže ide o ponuku, ktorú chceme prezentovať zákazníkom, chceme okrem textových údajov prezentovať položku, či už stroj alebo prídavné zariadenie, pomocou fotky. A nie jednej (predpokladám, že položku budú chcieť firmy predviesť zákazníkom z viacerých uhlov). Znovu by niekomu mohlo napadnúť, že by bol dobrý nápad zlúčiť entitu fotky stroja s entitou fotky prídavného zariadenia. Ale tieto veci nie sú totožné. Ak by sme entity zlúčili (a mali teda iba 1 entitu pre fotku všeobecne), tak by existovala možnosť priradiť fotku prídaného zariadenia stroju (a naopak). Ale to je nesprávne. Preto znova vytvoríme dve entity. Jedna bude fotka stroja, druhá bude fotka prídavného zariadenia. Pre fotku stroja platí, že patrí práve jednému stroju. Stroj môže mať viacero fotiek. Analogicky platí pre prídavné zariadenia a ich fotky.

Pri strojoch sa ešte zastavíme. Existujú rôzne značky strojov (napr. Locust, Eurocomach,...), a takisto rôzne ketegórie strojov (napr. šmykom riadené nakladače, pásové bagre,...). Ďalej v texte, keď budem používať spojenie typ stroja, tak tým myslím kombináciu značky stroja a kategórie stroja. Každý typ stroja sa môže líšiť druhom a formou údajov. Napríklad typ A má hmotnosť ako vlastnosť, ktorú chceme spolu so zbytkom údajov zobraziť užívateľovi. Ďalej typ B má namiesto hmotnosti údaj o výške stroja. Existujú údaje (ako sú napr. meno a opis stroja), ktoré existujú pre každý stroj. Ale takisto existujú údaje, ktoré sa líšia v závislosti od typu stroja. Takýmito údajmi sú vlastnosti stroja. Ako budeme tieto premenlivé údaje ukladať? Jedno z riešení, ktoré by nás mohlo napadnúť je vytvoriť rodičovskú entitu, ktorá by obsahovala údaje spoločné pre všetky typy strojov. A v entitách, ktoré by dedili od rodičovskej triedy by sme dodefinovali premenlivé vlastnosti. Toto riešenie by pravdepodobne fungovalo, lenže má zásadnú nevýhodu. Zakaždým keď si firma zmyslí, že potrebuje nový typ stroja, by entita musela byť manuálne doprogramovaná. Ale kedže my chceme systém navrhnúť všeobecne tak, aby si každá firma vedela zadefinovať vlastnú ponuku strojov, volíme inú alternatívu. Vytvoríme si entitu pre typ stroja. Každý stroj bude nejakého typu. Každý typ obsahuje údaj o značke a kategórii. Značka a kategória sú tiež ďalšími entitami. Typ stroja určuje, akého typu budú vlastnosti konkrétneho stroja. Takže budeme potrebovať entitu typ vlastnosti stoja. Tá obsahuje údaje: názov vlastnosti (napr. hmotnosť, výška,...) a typ hodnoty vlastnosti (napr. číslo, text,...). Teda admin bude môcť priradiť typu stroja akého typu bude mať konkrétny stroj vlastnosti.

Stroj vie svoj typ, a ten vie aké (akého typu) má konkrétny stroj vlastnosti. To, čo ešte nevieme, sú konkrétne hodnoty vlastností stroja. Dovolím si vysvetliť na príklade. Momentálne máme informáciu o tom, že konkrétny stroj S, typu T, má vlasnosť hmotnosť, ale stále nevieme konkrétnu hodnotu, teda stále nevieme koľko váži. Preto vytvoríme entitu reprezentujúcu vlastnosť stroja. Táto entita vie, akého je typu. A takisto v sebe uchováva konkrétnu hodnotu vlastnosti (v kontexte príkladu, uchováva v sebe váhu stroja). Každý stroj má v sebe toľko vlastností, koľko ich je zadefinovaných v jeho type.

Teraz sa vráťme k prídavným zariadeniam. Každé prídavné zariadenie má,

podobne ako stroj, takisto svoju značku a patrí do nejakej kategórie. Navyše má oproti strojom aj údaj o tom, pre akú kategóriu strojov je zariadenie vytvorené. Avšak na rozdiel od predošlého prípadu so strojmi, sa tento prípad líši v tom, že každé prídavné zariadenie, bez ohľadu na kombináciu typu, kategórie a kategórie stroja, má druh a formu údajov rovnakú. Takže stačí ak vytvoríme entitu pre značku a kategóriu prídavného zariadenia (entitu pre kategóriu stroja už máme) a entita reprezentujúca prídavné zariadenie si bude v sebe držať informáciu o tom, akej je značky, kategórie a pre akú kategóriu strojov je vytvorená.

Aby sme splnili P5, budeme potrebovať entitu reprezentujúcu aukčnú ponuku. Aukčná ponuka bude držať informáciu o tom, aký stroj je v dražbe. A takisto údaje o ponuke (napr. vyvolávacia cena, koniec aukcie,...).

No okrem udržania údajov o aukčnej ponuke a draženom stroji, si musíme zapamätať i údaje o ponukách užívateľov, ktorý sa do aukcie zapojili. Preto si vytvoríme entitu reprezentujúcu ponuku užívateľa. Bude v sebe niesť údaje o tom, ktorý úžívateľ ponúkol sumu, v akej výške a pre ktorú aukčnú ponuku.

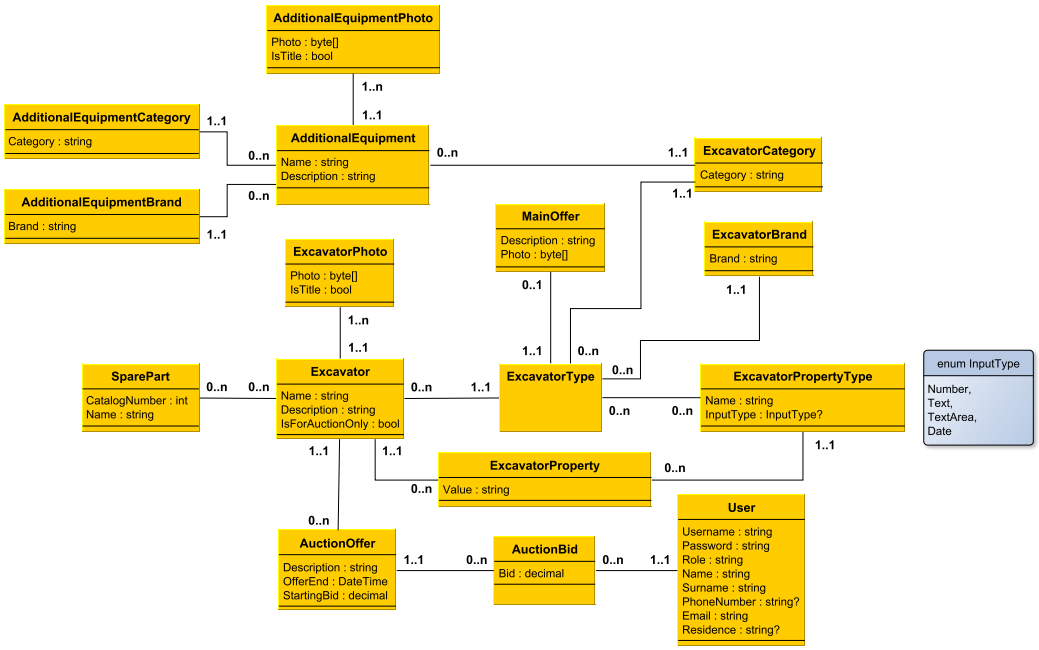
Entitu užívateľa som načal už v prechádzajúcom odstavci. Túto entitu skutočne budeme potrebovať a to aj kvôli splneniu P8. Údaje užívateľov musíme pri registrácii uchovať, aby sme nimi vedeli predvypĺniť formuláre, a takisto aby sme vedeli vytvoriť prihlasovanie.

Čítateľ by mohol navrhnúť, že pre splnenie P6 budeme potrebovať entitu reprezentujúcu správy. Toto riešenie by mohlo fungovať, ale ako si neskôr ukážeme, existuje aj iné riešenie. Také, ktoré nám ušetrí úložisko v databáze a navyše uľahčí implementáciu. Preto entitu pre správy nevytvárame.

Pre splnenie P7 si vytvoríme entitu, ktorá bude reprezentovať náhradné diely stroja. Každý náhradný diel bude niesť informáciu o tom, v ktorých strojoch sa nachádza. A každý stroj bude vedieť aké diely obsahuje. Každý náhradný diel obsahuje katalógové číslo. Toto číslo je unikátne medzi strojmi, a preto by mohlo byť použité ako primárny kľúč entity. Ale autor sa z opatrnosti a kvôli konzistencii (každá entita má id) rozhodol využiť ako primárny kľúč id i pri náhradných dieloch.

Hlavným predmetom predaja sú stroje. A preto chceme aby prvé, čo zákazník po príchode na stránku uvidí bola ponuka strojov. Takže ešte budeme potrebovať entitu na reprezentáciu hlavnej ponuky. Táto entita vie aký typ strojov ponúka, a zároveň obsahuje reprezentatívnu fotku a opis strojov daného typu.

Pre detailnejšiu predstavu môžeme návrh vidieť na obrázku 1.2.



Obr. 1.2: Relačný model databázy.

1.3.2 Voľba typu databázy a databázového servera

TBA

1.3.3 ORM

Prečo by sme chceli ORM.

1.4 Aukcia- odpočet a vyhodnocovanie

V tejto podkapitole predstavím BackgroundServices?, vysvetlím ako odpočítavať (len 1 timer, v osobitnej komponente kvôli rerenderom), ako funguje vyhodnocovanie- rozne scenare (co sa stane ak mame vitaza, co sa stane ak nemamee vitaza).

1.5 Posielanie a príjimanie správ

V tejto podkapitole rozoberieme spôsoby ako posielať/prijímať správy. Gmail api? AE.net? Mailkit?

Závěr

A. Přílohy

A.1 První příloha