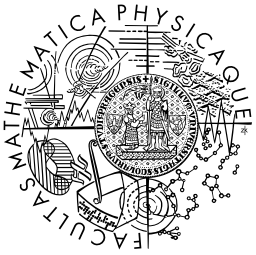


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

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

Rád by som sa poďakoval vedúcemu práce Mgr. Pavlovi Ježkovi, Ph.D. za jeho vedenie, rady, čas a trpezlivosť, ktorú mi venoval. Taktiež by som sa chcel poďakovať svojej rodine, ktorá ma motivovala a podporovala počas vypracovania tejto práce, ale aj počas celej doby štúdia.

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

1. Úvod

V súčasnosti existujú malé firmy, ktoré fungujú ako dodávatelia rôznych drahých produktov. Tieto firmy môžu ponúkať na predaj okrem nových produktov aj staré produkty, ktoré prešli nejakou opravou. Takisto stojí za zmienku, že keďže ide o dodávateľov drahých produktov, tak zákazníci sa najprv s firmou musia dohodnúť na detailoch obchodu, a až potom je možné dodanie produktu.

Spoločným problémom takýchto firiem býva, že ich ľudia nepoznajú. Preto by sa spomínaným firmám hodilo riešenie, ktoré by im umožnilo zviditeľniť ich ponuku produktov. Jednoduché riešenie v podobe statických stránok v tomto prípade nestačí, pretože by neumožnilo dynamicky meniť ponuku danej firmy. Použitie nejakého CMS systému (z ang. content management system), napr. WordPress, takisto nie je optimálnym riešením, pretože vyžaduje znalosť platformy, ktorá nie je samozrejmesťou.

My sme dostali ponuku na tvorbu riešenia od jednej z takýchto firiem. Konkrétne ide o firmu, ktorá sa zaoberá predajom a opravou bagrov (ďalej už len klientská firma). Po konzultácii s majiteľom (ďalej už len klient) sme zistili, že klientksá firma ponúka služby v podobe výkopových prác, predaja a opravy bagrov, a takisto predaja prídavných zariadení pre bagre. Taktiež sme zistili, že doteraz fungovala komunikácia medzi klientskou firmou a zákazníkmi prostredníctvom telefonátov, emailov alebo sa strany fyzicky stretli a dohodli obchod. Klient od nás vyžaduje riešenie, ktoré by spĺňalo následujúce požiadavky.

1.1 Požiadavky na systém

- **P1 Roly užívateľa**

  Jednou z požiadaviek je, že softvér má rozlišovať zamestnancov firmy spravujúcich systém (ďalej už len administrátori, resp. administrátor) a bežných zákazníkov. Administrátorom sa bude zobrazovať plná forma systému so všetkými funkcionalitami (napr. spravovanie ponuky), zatiaľ čo zákazníkom sa bude zobrazovať len obmedzený obsah podľa funkcionalít, ktoré má k dispozícii (čiže napr. bude si môcť zobraziť detail bagra, ale nezobrazí sa mu tlačidlo na vymazanie bagra).

- **P2 Predstavenie ponuky zákazníkom**

  Ako už bolo spomenuté, klientská firma predáva bagre a prídavné zariadenia pre bagre. Klient preto chce, aby sme vytvorili softvér, ktorý by bol schopný prezentovať ponuku firmy (bagre a prídavné zariadenia), pričom hlavná ponuka je tvorená bagrami. Klient vyžaduje, aby po príchode užívateľa na domovskú (úvodnú) stránku sa zobrazila hlavná ponuka.

  - **P2.1 Hlavná ponuka**

    Hlavná ponuka predstavuje bagre určitého typu (t. j. určitej kombinácie značky a kategórie). Hlavná ponuka obsahuje opis typu bagrov a fotku reprezentujúcu daný typ bagrov. Po rozkliknutí nejakej hlavnej ponuky sa zobrazia bagre typu asociovaného s vybranou ponukou.

- **P2.2 Bager**

  Každý bager má obsahovať informácie: názov, značku, kategóriu, opis, fotky a vlastnosti. Tieto informácie majú byť viditeľné pre každého užívateľa, t. j. ako pre bežného zákazníka, tak aj pre administrátora. Navyše má ešte stroj obsahovať informáciu o náhradných dieloch – táto informácia má byť viditeľná iba pre administrátorov. Navyše môžu existovať bagre, ktoré nepatria do ponuky a môžu byť využité výhradne iba v aukcii (viac o akcii v P3).

- **P2.3 Prídavné zariadenie**

  Každé prídavné zariadenie má obsahovať informácie: názov, značka, kategória, pre akú kategóriu strojov je zariadenie určené, opis a fotky. Tieto informácie majú byť viditeľné rovnako pre každého užívateľa.

- **P2.4 Správa bagrov, prídavných zariadení a hlavných ponúk**

  Aby mohol admin spravovať bagre, prídavné zariadenia ale takisto aj hlavné ponuky podľa potreby, tak je tiež nutné vytvoriť miesto, ktoré mu ich umožní pridávať, odstraňovať a editovať.

- **P3 Posielanie dopytu**

  Takisto klient od softvéru vyžaduje, aby umožnil zákazníkom objednať si daný produkt alebo službu prostredníctvom emailových správ. Bežnou praxou v tomto odvetví je, že cena strojov sa dopredu neudáva. Zákazník najprv vyjadrí záujem (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 sa myslí pridávanie do košíka s následnou platbou), ale bude fungovať na princípe posielania správ (dopytov).

  - **P3.1 Dopyt**

    Dopyt by mal v sebe obsahovať informácie o žiadanom predmete, údaje o užívateľovi, a tiež správu užívateľa. Uživateľskými údajmi sa myslí meno, priezvisko, email – tie sú povinné údaje. A takisto telefónne číslo, mesto – tie sú nepovinné údaje.

- **P4 Aukcia**

  Keď sme v predošlých podmienkach spomínali ponuku strojov a prídavných zariadení, tak išlo o nové produkty. No ako už bolo skôr spomenuté, klientská firma sa špecializuje aj na opravu bagrov. Klient vyžaduje, aby mohol admin v systéme vytvoriť aukčnú ponuku, do ktorej by okrem špecifikovania jej konca a počiatočnej sumy (vyvolávacej ceny), vedel zaradiť opravený bager, a aby systém umožnil zákazníkom ponúkať sumy (prvá ponúknutá suma môže byť rovná počiatočnej sume, nasledujúce ponúkané sumy musia mať medzi sebou rozdiel aspoň 100 eur), pričom po skončení dražby zákazník s najvyššou ponúknutou sumou vyhráva dražený bager.

  - **P4.1 Správanie aukcie**

    Keď aukcia skončí, systém má upozorniť jej účastníkov (poslať email) na to, či vyhrali alebo prehrali dražbu. Taktiež má softvér upozorniť

administrátora systému na to, že aukcia skončila a kto je jej víťazom. V prípade, že aukcia skončila bez víťaza (nikto sa jej nezúčastnil), tak má softvér dražbu automaticky reštartovať, posunúť termín konca dražby o týždeň a upozorniť o tom administrátora (poslať mu email).

- **P4.2 Odpočet a ďalšie údaje**

  Okrem toho sa od nášho softvéru vyžaduje, aby bol pri každej aukčnej ponuke zobrazený odpočet do konca danej dražby, počet účastníkov, a taktiež aktuálna (najvyššia ponúknutá) suma.

- **P5 Správy**

  Keďže posielanie dopytov a správanie aukcie zahŕňa posielanie emailov administrátorom, tak je tiež žiadúce, aby sa emaily dali prečítať nielen z Gmailu (emailová služba používaná klientom), ale aj z nášho systému a rovnako aby systém administrátorom umožnil na ne odpovedať.

  - **P5.1 Podobnosť s Gmailom**

    Nakoľko je klient zvyknutý na prácu s Gmailom, tak sa má schránka podobať na Gmail. Teda aspoň funkcionalitou, t. j. pri príchode do schránky sa zobrazia najnovšie správy pre každú konverzáciu (vlákno) zoradené zhora smerom dole od najnovšej po najstaršiu.

    Po rozkliknutí nejakej zo správ sa zobrazí celá konverzácia (každá správa vo vybranom vlákne) zoradená zhora dole od najstaršej po najnovšiu.

    Ďalej má schránka umožňovať označovanie správ, pričom označené správy budeme môcť hromadne vymazať alebo označiť za prečítané, resp. neprečítané. Ak sú označené správy neprečítané, zobrazí sa tlačidlo umožňujúce označenie vybraných správ ako prečítané, ak sú všetky označené správy prečítané, tak sa zobrazí tlačidlo umožňujúce označiť vybrané správy ako neprečítané, a ak označené správy obsahujú aj prečítané, aj neprečítané správy, tak sa zobrazí tlačidlo umožňujúce označiť vybrané správy ako prečítané.

    Podobne po rozkliknutí nejakej zo správ sa nám zobrazí celá konverzácia a administrátor bude môcť celú konverzáciu vymazať alebo označiť za neprečítanú, a taktiež bude môcť odoslať novú správu do konverzácie (odpovedať na správy).

    Čo sa týka mazania správ, tak po kliknutí na tlačidlo vymazania správy (resp. správ) stačí ak sa zobrazí potvrdzovacie okno, nie je nutné vytvárať osobitné miesto pre vymazané správy (kôš).

  - **P5.2 Prepojenie správy s predmetom**

    Okrem toho budeme ešte od softvéru vyžadovať, aby v správach, ktoré boli odoslané z nášho systému, ako napr. dopyt alebo správy z aukcie, tak aby v sebe obsahovali okno, ktoré prepojí správu a vec, ktorej sa daná správa týka. Teda napríklad ak zákazník odošle dopyt na stroj X, tak po otvorení správy nájde administrátor okrem predmetu a tela správy, takisto nejaký odkaz (prepojenie) odkazujúci na stroj X, ktorým sa dá jednoducho dostať k údajom o stroji X.

  - **P5.3 Automaticky generované správy**

    Okrem toho je tiež žiadúce, aby systém umožňil adminom upravovať formát automaticky odosielaných (generovaných) správ týkajúcich sa aukcie.

- **P6 Registrácia a prihlásenie užívateľov**

  Ďalšou požiadavkou je, aby softvér umožnil zákazníkom registrovať sa do systému a následne sa doň prihlásiť. Do systému sa môžu prihlasovať rovnakým spôsobom ako bežní zákazníci aj administrátori. Systém má rozlíšiť či ide o účet bežného zákazníka alebo o administrátorský, prípadne podľa toho upraviť obsah stránky ako bolo spomenuté v P1. Každý užívateľ má mať po prihlásení výhodu v tom, že do formulárov nemusí zadávať svoje osobné údaje.

  - **P6.1 Funkcionality pre neprihlásených užívatelov**

    No požiadavkou je takisto to, aby aj neprihlásení užívatelia neboli o funkcionality, akými sú posielanie dopytov alebo účastnenie sa aukčných dražieb, nijako ukrátení a mohli ich využívať.

  - **P6.2 Registrácia a prihlasovanie užívateľa**

    Pri registrácií si bude môcť užívateľ vybrať svoje užívateľské meno, heslo, takisto bude môcť zadať svoje meno, priezvisko a email. Všetky spomenuté údaje sú povinné. Nepovinnými údajmi, ktoré môže užívateľ ďalej zadať, sú telefónne číslo a mesto.

    Pri prihlasovaní do systému má užívateľ zadať svoje prihlasovacie meno a heslo.

  - **P6.3 Profil užívateľa**

    Takisto je nutné vytvoriť miesto, kde si užívateľ môže svoje údaje upravovať (profil).

- **P7 Prístup k súčiastkam strojov**

  Keď si zákazník zakúpi bager, tak po nejakom čase má firma vykonať kontrolu tohto bagra. Ale predtým než zamestnanci pôjdu vykonať kontrolu si musia zistiť, aké súčiastky obsahuje daný bager. A preto klient od softvéru vyžaduje, aby umožňoval administrátorom zobraziť aké náhradné diely obsahuje konkrétny bager.

  - **P7.1 Náhradný diel**

    Jeden bager môže obsahovať viacero náhradných dielov (počet nás nezaujíma) a jeden náhradný diel sa môže nachádzať vo viacerých bagroch. Náhradný diel obsahuje informácie: katalógové číslo, názov.

  - **P7.2 Správa náhradných dielov**

    Taktiež je potrebné vytvoriť miesto, ktoré adminovi umožní náhradné diely pridávať, editovať, mazať a upravovať ich vzťahy s bagrami.

- **P8 Objednávanie výkopových prác**

Klient taktiež vyžaduje miesto v systéme, kde budú opísané služby (presnejšie výkopové práce), ktoré firma poskytuje, a aby užívateľ mohol odtiaľ o dané služby požiadať (odoslať email, v ktorom opíše svoje požiadavky).

- **P9 O nás a Kontakt**

  Navyše klient žiada miesto v systéme, kde bude opísaná firma a jej história, a takisto miesto, kde bude zobrazený kontakt (email, telefónne číslo) na klientskú firmu (príp. jej pridružené firmy). V oboch prípadoch pôjde len o statický text, príp. fotky.

- **P10 Dostupnosť**

  Nakoľko klientovi ide o to, aby dostal ponuku viac do povedomia (i potenciálne nových) zákazníkov, je žiadúce, aby bol softvér jednoducho dostupný každému užívateľovi – tým sa myslí, že užívateľ si nemusí sťahovať, inštalovať žiaden softvér, a takisto v prípade administrátorov sa chceme vyhnúť problémom s kompatibilitou (s pozorovania autor vie, že firmy častokrát používajú staré počítače s potenciálne starým softvérom, čo by mohlo spôsobovať problémy s prevádzkou nášho systému).

- **P11 Náklady**

  Kedže ide o malú firmu, tak chceme, aby náklady spojené s tvorbou a vedením softvéru boli minimálne alebo v ideálnom prípade žiadne. Konkrétne sa myslia náklady spojené s potenciálnym využitím softvéru, balíčkov tretích strán alebo databázových serverov atď.

1.2 Zhrnutie

Po hľadaní alternatívnych riešení sa nám nepodarilo nájsť žiaden už existujúci systém, ktorý by spĺňal predchádzajúce požiadavky. Podobný systém by si mohol vytvoriť majiteľ firmy sám napr. pomocou WordPressu, ale to by vyžadovalo pokročilejšie znalosti platformy.

Preto je cieľom tejto práce implementovať systém spĺňajúci požiadavky P1 až P11, určený pre firmy, ktoré sa zaoberajú predajom a opravou bagrov, ktorý by majiteľom firiem umožnil sústrediť sa len na ich doménu (t. j. napr. pridávanie bagrov do ponuky) a neriešiť detaily implementácie funkcionalít a vzhľadu systému (ako by tomu bolo v prípade WordPressu).

2. Návrh systému

Kvôli P10 (1.1) dáva dobrý zmysel vytvoriť náš systém ako webovú aplikáciu – vyriešime tým problém s distribúciou softvéru k (aj potenciálne novým) zákazníkom, a takisto vyriešime problém s potenciálne zastaralými počítačmi vo firme. Statická stránka by nám nestačila nakoľko chceme administrátorom umožniť dynamickú zmenu obsahu.

2.1 Splnenie P2 a P3

TBA: Pridal by som PP slide s hlavnou strankou, s kartickami, a detailom predmetov (vratane dopytoveho formulara) a opisal by som "Po prichode na stranku sa nam vylistuju karty hlavnej ponuky po prejdeni kurzorom na kartu hlavnej ponuky sa obrazok prepne na text (opis typu stroja)... Na obrazku X vidime aj formular na posielanie dopytu, ten sa otvori ked klikneme na tlacidlo Dopyt v Detaile, ktory vidime na obrazku Y...

2.2 Splnenie P4

TBA: Podobne ako v predchadzajucej sekcii... Znovu by sa v podstate opísali slidy. Možno by sa tiež hodilo spomenúť to, že keď sa skončí aukcia (už či s víťazom alebo bez), tak môže admin prísť a reštartovať, takze bolo by dobre vytvoriť separatny slide (zatiaľ neni) a ukázať na ňom rozdiel (rozdiel bude jedno tlačidlo, ak ešte beží tak bude písať uložiť a keď už skončila tka uložiť a reštartovať alebo tak nejak...)

2.3 Splnenie P5

TBA: podobne ako v predchadzajucej sekcii... Opisal slidy a spomenul predosle podmienky, ze prepojeni msa dostaneme na detaily (slidy spomenute uz boli v predoslych sekciach).

2.4 Splnenie P6

TBA: Podobne ako v predchadzajucej sekcii... Prilhasenie a Registraciu slidy spol us mojimi udajmi v profile co je ukazat a opisat.

2.5 Splnenie P7

TBA: podobne ako v predchadzajucej sekcii... TODO dorobit slide kde by bololepsie vidno tuto funckionalitu lebo zatial nemam

2.6 Splnenie P8

TBA: podobne ako v predchadzajucej sekcii...

2.7 Splnenie P9

TBA: podobne ako v predchadzajucej sekcii...

2.8 Splnenie P1 a správy predmetov

TBA: podobne ako v predchadzajucej sekcii by som opisal slidy ktore zobrazuju spravu (create, update, delete) predmetov... ALE s tym ze by som mozno spomenul ze vyuzijeme Syncfusion a ze to neni v rozpore s P11 (naklady) lebo existuje community verzia ktora je bezplatna. A taksito asi by som (momentalne nemam) dorobil verzie slidov, ktore uz boli predstavene v niektorych predoslych sekciach (karticky), v adminskom pohlade teda ze tam su tie tlacidla na vymazanie atd.

2.9 Výber jazyka a frameworku

TBA: Ako vidíme tak potrebujeme bohaty frontend, na to sa hodi napr JavaScriptové frameworky ako su React, Vue atd., a takisto nam nestaci Backend as a Service (ktory poskytuje napr Amazon...), lebo napr kvoli vyhodnocovaniu aukcie, takze preto vlastny backend. Na tvorbu webovej aplikacie s bohatym UI sa hodi high level jazyk, napr Java alebo C#. Autor ovlada C#, preto volime C# a platformu .NET, ktora je s nim spojena. Backend by sme mohli implementovat pomocou frmeworku ASP.NET MVC. No este existuje ina varianta a tou je Blazor. Ten nam umozni vyuzivat komponenty (to sa nam hodi napr. aj kvoli implementácii odpočtu) a takisto nám umožní písať frontend aj backend v rovnakom jazyku – C#.

Blazor poskytuje viaceo hosting modelov a v dobe vyberu technologii existovali dva – Blazor WebAssebmly a Blazor Server. Blazor WebAssembly funguje skor na sposob SPA, kde logika aplikacie bezi na klientskom pocitaci vo webovom prehliadaci, a su s nim spojene nejake problemy [vymenujem, napr to ze search engingy mozu mat problem s nim, P10 aby to uzivatelom rozbehol pc, aby nebol zastaraly prehliadac, prvotne nacitanie trva dlho (lebo stahuje zdrojaky) co by mohlo potencialnych zakaznikov odradit, a tiez nejake WebApi na server by sme museli vytvarat). Naproti Blazor Server sa podoba skor tradicnemu web app pristupu a hodi sa nam viac lebo je CEO-friendly, kod bezi na servery a uzivatel dostane len HTML, CSS, JS, preto aj starsie pc by nemali mat problem s rozbehnutim, a takisto nema preto problem s dlhym prvotnym nacitanim, a navyse nemusime vytvarata WebApi.

Preto si volime C# (.NET) a Blazor Server.

3. Analýza zadania

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

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.

3.1 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:

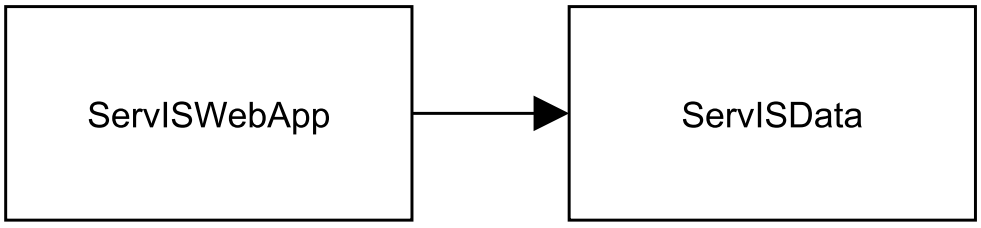
Blazor WebAssembly a Blazor Server. Výber WebAssembly by zahŕňal niekoľko problémov. Pri prvotnej návšteve stránky sa musia klientovi stiahnuť zdrojové kódy aplikácie. To môže chvíľu trvať a mohlo by to odradiť nových potenciálnych zákazníkov. V prípade Blazor Server tento problém nemáme, pretože kód 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.

3.2 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 3.1.



Obr. 3.1: Architektúra systému (predstavuje závislosť ServISWebApp od ServISData).

3.3 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.

3.3.1 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.

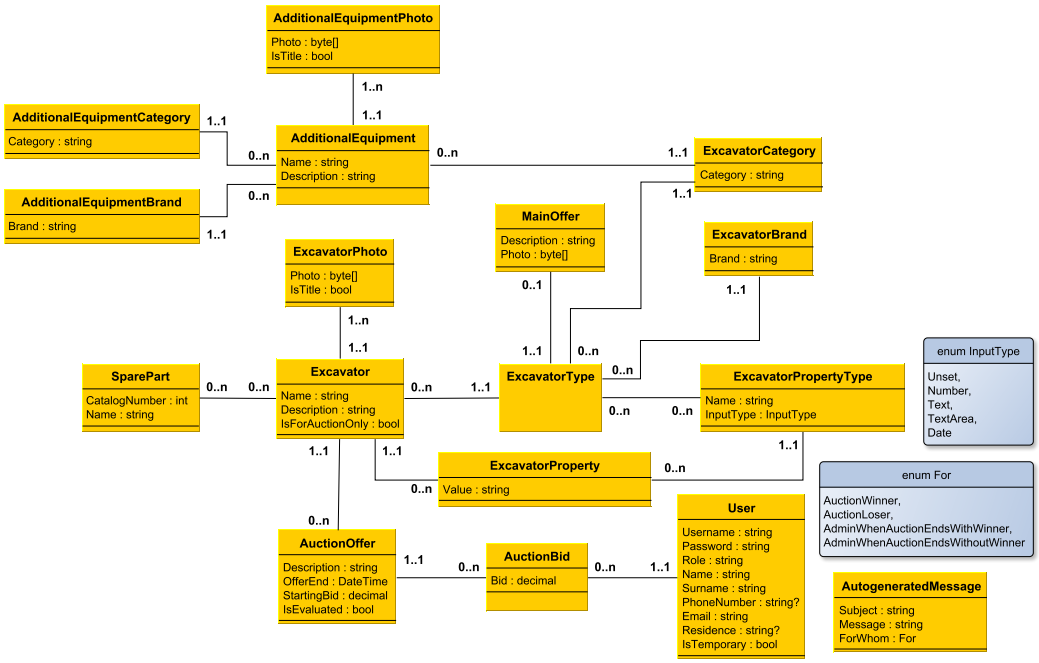
Entitu užívateľa som načal už v prechádzajúcom odstavci. Túto entitu skutočne budeme potrebovať a to aj kvôli splneniu P8. Údaje užívateľov musíme pri registrácii uchovať, aby sme nimi vedeli predvypĺniť formuláre, a takisto aby sme vedeli vytvoriť prihlasovanie.

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 (okrem iného, ale detailnejšie rozobranie príde neskôr) ušetrí úložisko v databáze. 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 3.2.



Obr. 3.2: Relačný model databázy.

Pozorný čítateľ si mohol všimnúť, že na obrázku existuje entita, ktorú sme ešte nespomenuli. Ide o entitu reprezentujúcu automaticky generované správy. Na dôvod jej existencie sa pozrieme neskôr. Zatiaľ len prezradím, že táto entita sa využije pri fungovaní aukcie.

3.3.2 Voľba typu databázy a databázového servera

Kvôli tomu, že naše dáta majú presne definovanú štruktúru, si volíme relačnú databázu. Tá nám poskytne rýchly prístup k dátam, a takisto zaručí ich neporušenosť. Ďalším plusom je, že autor má s relačnými databázami trochu viac skúseností, než s NoSql databázami.

Čo sa týka databázového servera, tak si najprv poďme rozmyslieť, čo od neho budeme vyžadovať. Kvôli P2 chceme, aby bol server (v najlepšom prípade) bezplatný. Taktiež musí byť kompatibilný s platformou .NET. Tieto požiadavky spĺňajú servery: Microsoft SQL Server, Oracle Database a MySQL. V prípade prvých dvoch serverov je síce pravda, že poskytujú aj bezplatnú verziu, ale tie sú obmedzené na kapacitu. Ale aj napriek obmedzeniu je kapacita dosť veľká. V prípade Community verzie MySql takéto obmedzenie kapacity nemáme žiadne. V budúcnosti, ak by sa firme darilo a získala by viacej zákazníkov, prípadne rozšírila ponuku, by sa nám mohla neobmedzená kapacita servera hodiť. Preto si volíme MySql.

3.3.3 ORM

Keď už máme vybraný databázový server, poďme si rozmyslieť ako s ním budeme pracovať. Potrebujeme vytvoriť databázu s tabuľkami, vkladať, editovať, čítať a mazať dáta. Všetky tieto úkony by sme mohli vykonať tak, že budeme písať kód s SQL syntaxou. Ale keďže autor nemá veľa skúseností s jazykom SQL,

prichádza do úvahy využiť nejaký z ORM frameworkov (z ang. object relational mapping). Asi najznámejšími na platforme .NET sú: Dapper a Entitfy Framework Core. Využívanie frameworku Dapper zahŕňa písanie SQL kódu. Pretože sa snažíme obmedziť písanie SQL kódu, nedáva zmysel si vybrať Dapper. Našou voľbou je preto Entity Framework Core. Konkrétne s prístupom založeným na kóde (anglicky code first approach). To znamená, že entity z relačného modelu najprv napíšeme do C# kódu ako triedy a z nich Entity Framework Core vytvorí tabuľky databázy.

3.4 Aukcia- odpočet a vyhodnocovanie

Ako už bolo skôr v texte spomenuté, budeme potrebovať nejaký mechanizmus, ktorý zvládne vykonávať odpočet do konca každej aukčnej ponuky, a takisto aby ju po skončení vedel vyhodnotiť.

Odpočet by sme mohli implementovať tak, že by sme si vytvorili inštanciu triedy `Timer`[1] pre každú aukčnú ponuku. Každá z inštancií by každú sekundu odpálila event, pri ktorom by došlo k prerenderovaniu komponentu s časom do skončenia ponuky. Toto riešenie by síce fungovalo, ale je to mrhanie zdrojmi. Stačí nám jeden `Timer`. Na jeho event `Elapsed` si každý komponent s odpočtom zaregistruje metódu na prerenderovanie. Týmto spôsobom sa nám podarí ušetriť zdroje, ale vyskytá sa otázka. Kde bude inštancia triedy `Timer` uložená? Prirodzenou odpoveďou by bolo, že ju uložíme do rodičovského komponentu. Limitácia tohto riešenie spočíva v tom, že ak budeme potrebovať vykonať nejakú operáciu periodicky každú sekundu, ale v komponente, ktorá sa nenachádza v rodičovi s triedou `Timer`, tak nemáme prístup k triede `Timer`. Lepším riešením bude presunúť `Timer` do nového vlákna, ktoré bude bežať na pozadí. Takýmto spôsobom si môžeme kedykoľvek a odkiaľkoľvek registrovať periodické operácie.

Tu by som sa ešte rád zastavil a vrátil k poznámke z úvodu, kedy som spomenul, že komponenty nám pomôžu vyriešiť problém s odpočtom. Vieme, že kvôli odpočtu dôjde každú sekundu k prerenderovaniu stránky. No určite nechceme, aby sa nám každú sekundu prerenderovávala celá stránka (resp. hľadali zmeny na celej stránke). Ak izolujeme odpočet do samostatného komponentu, prerenderovanie sa vykoná iba v ňom. Zbytok stránky to neovplyvní.

Ďalej si potrebujeme rozmyslieť ako bude prebiehať vyhodnocovanie aukcie. Náš systém bude fungovať tak, že užívateľ (ak má záujem o dražený stroj) odošle svoju ponúkanú sumu. Tá sa uloží v databáze. Potrebujeme mechanizmus, ktorý by sledoval aukčné ponuky. Ak by uvidel, že nejaká z ponúk už skončila, tak ju vyhodnotí. Periodické sledovanie aukčných ponúk je dlhodobo bežiaca operácia, a preto sa hodí ju takisto vykonávať vo vedľajšom vlákne na pozadí.

Poďme sa ešte pozrieť na to, čo presne zahŕňa vyhodnocovanie jednej aukčnej ponuky. Prejdeme všetky ponúknuté sumy, nájdeme najvyššiu sumu a užívateľ, ktorý ju ponúkol bude vyhlásený za víťaza. Víťaz aukcie bude informovaný o skutočnoti, že vyhral stroj v aukcii. Porazení budú takisto oboznámení o svojej prehre. Okrem tohto scenára existuje ešte jeden. Čo ak sa nikto nezapojil do aukcie? Má sa aukčná ponuka zmazať? Alebo iba označiť za ukončnenú a schovať sa

[1] `https://learn.microsoft.com/en-us/dotnet/api/system.timers.timer?view=net-8.0`

pred bežnými zákazníkmi? Alebo sa má koniec aukcie presunúť na neskôr? Ako vidíme možností je viacero. Prvé dve možnosti vyžadujú zásah admina, ktorý by musel nanovo vložiť ponuku do systému (prvý prípad) alebo upraviť ponuku a znova ju zviditeľniť zákazníkom (druhý prípad). Všimnime si, že posledná z možností nevyžaduje akciu admina. Administrátor môže ponuku upraviť, ale nemusí. Systém sa postará sám o seba a funguje aj bez zásahu admina. A to je presne to, čo od nás vyžuje P3. Preto volíme tretí prístup. V prípade, že aukcia skončí bez víťaza, tak systém o tom upozorní administrátora a koniec ponuky posunie na neskôr (napr. o týždeň).

V oboch prípadoch (odpočet i vyhodnocovanie) sme sa rozhodli využiť vlákna na pozadí. Vo frameworku Blazor existuje trieda `BackgroundService`[2]. Tá nám umožní rozbehnúť kód na pozadí. Takže logiku behu vlákien na pozadí si nemusíme implementovať sami.

3.5 Posielanie a prijímanie správ

Z pozorovania autor vie, že zákazníci týchto firiem, a rovnako i samotné firmy, sú zvyknuté na komunikáciu pomocou emailov.

V predchádzajúcom texte sme už viackrát spomínali, že budeme posielať správy. Jednak kvôli spĺneniu P6, ale takisto pre upozorňovanie účastníkov a administrátorov na priebeh aukcií. Tiež sme si povedali, že správy nebudú reprezentované entitami v databáze. Ako teda vyriešime túto úlohu? Ak bude posielanie správ (dopytov, notifikácií z aukcie) implementované prostredníctvom posielania emailov do emailovej schránky firmy, tak ušetríme miesto v databáze. To nám môže potenciálne znížiť náklady, čo je v prospech P2. Nevýhodou by však bolo, že administrátor by musel chodiť kontrolovať emaily mimo našu aplikáciu. Tejto nevýhody sa dokážeme zbaviť. A to tak, že vytvoríme zjednodušenú emailovú schránku priamo v našej aplikácii. Týmto spôsobom docielime toho, že užívateľ vie odoslať správu z nášho systému, náš systém vie odoslať správu o stave aukcií, admin si ich v našej aplikácii vie prečítať a reagovať na ne. Zároveň sa nestaráme o ukladanie správ v databáze, takže šetríme úložisko. A čo viac, zákazníci vedia kontaktovať firmu i mimo nášho systému (keď zákazník pošle firme email z miesta mimo nášho systému, admin si ho vie aj napriek tomu v našej aplikácii prečítať).

Dospeli sme k tomu, že nepotrebujeme správy len odosielať ale potrebujeme ich aj čítať. Ešte predtým, než si zvolíme spôsob akým budeme správy prijímať a posielať, si poďme vybrať akú emailovú službu budeme využívať. Od služby vyžadujeme, aby podporovala protokoly IMAP (pre prijímanie správ) a SMTP (pre posielanie správ). Jednou zo služieb ktorá spĺňa podmienku je Gmail od spoločnosti Google. Ide o veľkú spoločnosť, ktorú pozná snáď každý a autor dôveruje ich zabezpečeniu. Ďalším plusom je to, že k tejto službe existuje dokumentácia. Navyše služba poskytuje API (z ang. application programming interface), ktoré by sme mohli využiť v prípade potreby. Čítateľ by mohol poznamenať, že API by sme mohli vyvužiť ku kompletnej implementácii našej schránky. Dôvodom prečo nevyužijeme API služby pre kompletnú implementáciu je ten, že

[2] `https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-7.0&tabs=visual-studio#backgroundservice-base-class`

to čo ideme vytvoriť sa skôr podobá emailovému klientovi. Na jeho implemenáciu je vhodnejšie použiť protokoly IMAP a SMTP (rovnako to spomína i dokumentácia API[3]). Ako vidíme, Gmail sa zdá byť dobrým kandidátom, a preto si ho zvolíme.

Ďalej potrebujeme zistiť ako môžeme využívať protokoly IMAP a SMTP z nášho programu. Zdá sa, že najznámejšími balíčkami pre prácu s týmito protokolmi sú MailKit a AE.Net.Mail. Obe umožňujú prácu s IMAP i SMTP, ale AE.Net.Mail nebol už dlhšiu dobu aktualizovaný. A preto, ak sa z tohto systému stane dlhodobejší projekt, sa zdá byť lepšou možnosťou MailkKit.

3.5.1 Automaticky generované správy

Spomínali sme, že systém bude sám automaticky odosielať správy, či už víťazom aukcie, porazeným alebo adminom, o stave aukcie. Ak by náš systém využívali viaceré firmy, je prirodzené, že by si chceli tvar automaticky generovaných správ upraviť. Náš systém firmám túto funkcionalitu umožní. Preto sme si v časti Návrh relačného modelu databazy (3.3.1) dopredu pripravili entitu reprezentujúcu automaticky generované správy.

[3] `https://developers.google.com/gmail/api/guides`

4. Vývojová dokumentácia

V tejto kapitole sa pozrieme na implementáciu nášho systému. Zameráme sa na organizáciu a implementáciu dôležitých častí aplikácie. Cieľom tohto textu nie je opísať správanie každého riadku kódu. Na to slúži kód samotný (prípadne komentáre, ktoré autor pridal na miesta, kde to uznal za vhodné). Odkaz k zdrojovým kódom aplikácie sa nachádza v prílohe (A.1).

Celá aplikácia je rozdelená do dvoch projektov- ServISData a ServISWebApp.

4.1 ServISData

Zmyslom projektu ServISData je správa databázových entít a komunikácia s databázou. Teraz si prejdeme jednotlivé priečinky tohto projektu a opíšeme si ich obsah.

4.1.1 Koreňový priečinok projektu

Nachádza sa tu `ServISDbContext` a spolu s ním i `ServISDbContextFactory`. Tieto triedy sú zodpovedné za konfiguráciu databázy, a tiež samotné pripojenie aplikácie k databáze. Viac o týchto triedach a spôsobu ich použitia v projekte nájdeme tu.

V priečinku sa tiež nachádzajú triedy `AutogeneratedMessageForExtensions` a `InputTypeExtensions`. V oboch prípadoch ide o extension metódy umožňujúce získať metadáta z atribútov, ktoré sme použili
v enumoch `AutogeneratedMessage.For` a `InputType`.

Takisto sa v priečinku nachádza už spomenutý `InputType`. Ide o enum, ktorého hodnoty predstavujú možné typy hodnôt vlastností stroja.

Ďalej sa tu nachádza trieda `ServISApi`. Tá slúži pre komunikáciu s databázou. Respektíve umožnuje iným projektom (v našom prípade ServISWebapp) modely ukladať, čítať, editovať a mazať z databázy.

4.1.2 Attributes

Priečinok obsahuje atribúty[1]. Konkrétne ide
o `AutogeneratedMessageDataAttribute` a `InputTypeLabelAttribute`.

Prvý slúži na nastavenie predvoleného predmetu a textu automaticky generovaných správ, ale takisto aj na nastavenie ich podporovaných tagov.

Druhý slúži pre uloženie užívateľsky prívetivejšieho názvu typu hodnoty vlastnosti stroja. Tieto názvy sa zobrazujú napríklad pri vytváraní typu vlastnosti stroja.

4.1.3 DataOperations

Priečinok obsahuje triedy, ktoré ich užívateľom umožnia vykonávať rôzne operácie nad dátami (filtrovanie, stránkovanie,...).

[1] `https://learn.microsoft.com/en-us/dotnet/csharp/advanced-topics/reflection-and-attributes/`

4.1.4 Interfaces

Priečinok obsahuje rozhrania (ang. interfaces).

`IServISApi` nás zbaví potreby upravovať kód využívajúci API projektu ServISData v prípade, ak sa rozhodneme vymeniť `ServISApi` za nejakú inú implementáciu.

`IPhoto` nám umožní všeobecne pracovať s fotkami aj napriek tomu, že pôjde o fotky rôznych entít.

`IItem` nám dovolí písať všeobecný kód pre prácu s modelmi entít (využíva sa napr. pri mazaní entít z databázy; viď metódu `DeleteItem` v triede `ServISApi`).

4.1.5 Migrations

Tento priečinok bol vygenerovaný frameworkom Entity Framework Core a obsahuje vygenerovaný kód. Ide o migrácie[2]. Nad migráciou môžeme rozmýšlať ako nad commitom v gitu. Po aplikovaní migrácie dôjde k zmene v databáze (napr. sa pridá nový stĺpec do nejakej z tabuliek).

4.1.6 Models

Priečinok obsahuje modely- triedy reprezentujúce entity uložené v databáze.

Špeciálne sa zastavíme pri triede `AutogeneratedMessage`
a enume `AutogeneratedMessage.For`. Platí, že každá hodnota enumu predstavuje druh automaticky generovanej správy.
Napríklad `AutogeneratedMessage.For.AutionWinner` predstavuje automaticky generovanú správu pre víťaza aukcie. A tiež platí, že pre každú hodnotu tohto enumu existuje inštancia triedy `AutogeneratedMessage` uložená v databáze.
Predvolené hodnoty správ sa nachádzajú v atribútoch hodnôt enumu a administrátormi definované hodnoty sa ukladajú do databázy.

4.2 ServISWebApp

Projekt ServISWebApp predstavuje rozhranie pre užívateľov a zároveň obsahuje logiku nášho systému. Znova, ako v predošlej podkapitole, si prejdeme jednotlivé priečinky tohto projektu a opíšeme si ich obsah.

4.2.1 Koreňový priečinok projektu

Priečinok obsahuje súbory _Imports.razor (nachádzajú sa v ňom `using` direktívy aplikované pre každý komponent v projekte), App.razor (koreňový komponent), appsettings.json (ide o konfiguráciu aplikácie, v ktorej okrem iného vieme nastaviť emailovú adresu, ktorú systém využíva pre odosielanie a prijímanie emailov) a Program.cs (obsahuje kód zodpovedný za spustenie celej aplikácie). Pre viac informácií o štruktúre Blazor projektu môže čítateľ kliknúť tu.

V časti Models 4.1.6 bolo spomenuté, že pre každú hodnotu
enumu `AutogeneratedMessage.For` existuje inštancia triedy

[2] `https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli`

`AutogeneratedMessage` uložená v databáze. To zabezpečuje metóda `StoreNewAutogeneratedMessagesAsync`, ktorá sa nachádza v Program.cs.

4.2.2 Auth

Priečinok obsahuje triedy zodpovedné za prihlasovanie užívateľa a hashovanie hesla.

4.2.3 BackgroundServices

V tomto priečinku sa nachádzajú triedy predstavujúce služby bežiace na pozadí. Konkrétne ide o službu zodpovednú za vyhodnocovanie aukcie a službu, ktorá každú sekundu vykoná akciu, ktorú si užívateľ triedy v službe zaregistroval (v našom prípade ide o prerenderovanie komponentov pri odpočte).

4.2.4 Components

V tomto priečinku sa nachádzajú komponenty využité v aplikácii.

Komponenty ktorých meno končí na slovo "Lister", slúžia na zobrazenie (vylistovanie) nejakého obsahu. Väčšinou sú tým obsahom komponenty končiace na slovo "Card".

Ďalej sa tu nachádzajú komponenty začínajúce na slovo "Input". Tieto komponenty spolu s komponentom `ChecklistTable` slúžia na získavanie vstupu užívateľa.

Komponenty `Message`, `Messages`, `ThreadRow`, `ThreadView` a `ThreadsView` tvoria miesto pre manažovanie správ. Umožňujú adminovi čítať správy, a takisto na nich odpovedať. Navyše `MessageSettings` umožňuje adminovi meniť tvar automaticky generovaných správ.

Komponent `CountdownDisplayer` sa využíva na zobrazenie odpočtu do konca aukcie.

`CustomDataAdaptor` sa využíva v komponente `ItemsManagement` na čítanie dát.

`LoginPanel` umožňuje neprihlásenému užívateľovi prístup k prihlasovaciemu a registračnému formuláru. Alebo v prípade prihláseného užívateľa poskytuje možnosti prejsť na profil a odhlásiť sa.

`PhotoSlider` sa využíva na zobrazovanie obrázkov (môžeme ho vidieť napríklad na stránke detailu ľubovoľného stroja).

Komponent `TabControl` v sebe obsahuje komponenty `TabPage`. Umožňujú užívateľom (najmä adminom, pretože bežní zákaznici majú iba jeden tab) preklikávať sa medzi rôznymi časťami profilu.

Ďalej sa tu nachádzajú priečinky Buttons, Forms a Managements.

V priečinku Buttons sa nachádzajú komponenty reprezentujúce rôzne tlačidlá.

Vo Forms sa nachádzajú rôzne formuláre ktorými dokážeme vytvárať a do databázy ukladať inštancie modelov (napr. stroje, prídavné zariadenia,...). Ale takisto sa tam nachádza i komponent `DemandForm`, ktorý zákazníkom umožňuje odosielať dopyt.

A v priečinku Managements sa nachádza komponent `ItemsManagement`. Tento komponent tvorí rozhranie pre administrátorov na správu entít (napr. strojov,

náhradných dielov,...). Umožňuje adminom entity vytvárať, editovať a mazať. Takisto im pri prezeraní existujúcich entít umožňuje ich vyhľadávanie a triedenie.

4.2.5 CssProviders

Triedy v tomto priečinku sú zodpovedné za zmenu css štýlov vo formulároch pri modifikovaní a validácií políčok formuláru. O triede `FieldCssClassProvider` sa čítateľ môže dozvedieť viac tu.

4.2.6 Pages

V tomto priečinku sa nachádzajú stránky a priečinok Admin, ktorý obsahuje stránky exkluzívne pre administrátorov.

4.2.7 Resources

Tento priečinok obsahuje ".resx" súbory so slovenskými prekladmi slov a viet. Využívajú ich komponenty z balíčku od spoločnosti Syncfusion.

4.2.8 Shared

V tomto priečinku sa nachádzajú triedy zdieľané (potenciálne) celým projektom.

Triedy `Email` a `Thread` slúžia ako dátonosné alternatívy k triedam balíčka MailKit. Môžeme nad nimi rozmýšľať ako nad fasádami, ktoré zjednodušujú prístup k dátam.

`EmailManager` pokrýva funkcionalitu práce s emailami, napr. ich posielanie a prijímanie.

`MainLayout` a `NavMenu` sú štandardné Blazor komponenty. Prvý definuje rozloženie stránky a druhý navigáciu.

`SyncfusionLocalizer` a `SyncfusionDataOperations` sú triedy, ktoré využívajú kód z balíčka od spoločnosti Syncfusion. Prvá sa stará o nastavenie lokalizácie užívateľského rozhrania Syncfusion komponentov. Druhá vykonáva operácie nad dátami (napr. vyhľadávanie, triedenie,...). Čítateľ môže byť v tento moment mierne zmätený, pretože raz sme tu už podobnú triedu mali. Konkrétne v časti DataOperations (4.1.3). Rozdiel je v tom, že trieda spomenutá v predošlej časti využíva na konfiguráciu operácií našu vlastnú triedu `MyDataOperations.Configuration`, kdežto `SyncfusionDataOperations` využíva triedu `DataManagerRequest`, ktorá je takisto z balíčka spoločnosti Syncfusion.

`FileTools` združuje metódy pre prácu so súbormi. Konkrétne v našom prípade ho využívame pre prácu s fotkami.

Ostali nám už len trieda `AuctionSummary` a trieda `AutogeneratedMessageExtensions`, ktorá sa nachádza v priečinku Extensions. Obe triedy spolu úzko súvisia.
`AutogeneratedMessageExtensions` obsahuje extension metódy pre triedu `AutogeneratedMessage`. `AutogeneratedMessage` obsahuje text (predmet a telo správy). V texte sa nachádzajú tagy a slúži ako šablóna. Metódy triedy

`AutogeneratedMessageExtensions` fungujú tak, že vezmú túto šablónu a namiesto tagov dosadia skutočné dáta vzaté z inštancií
triedy (príp. tried) `AuctionSummary`.

5. Užívateľská dokumentácia

Užívateľská dokumentácia existuje v podobe videonávodu pre administrátorov na obsluhu systému. Odkaz na video sa nachádza v prílohe (A.2).

Záver

V závere zhodnotíme ako sa nám podarilo naplniť požiadavky definované v Úvode, konkrétne v podkapitole Požiadavky na softvér (**??**).

- **P1 Dostupnosť**

  Náš systém je webovou aplikáciou, ktorá je dostupná pre užívateľov odkiaľkoľvek (samozrejme za predpokladu, že majú prístup k internetu).

- **P2 Náklady**

  Všetky balíčky využívané aplikáciou sú buď úplne zdarma alebo využívame ich bezplatné verzie. Čo sa týka databázového servera, tak ten takisto využívame v jeho bezplatnej verzii.

- **P3 Minimálna obsluha softvéru**

  Táto požiadavka sa odzrkadľuje v správaní aukcie. V prípade, že aukčná ponuka skončí bez toho, aby sa jej niekto účastnil, posunie sa jej termín ukončenia aj bez zásahu administrátora.

- **P4 Predstavenie ponuky zákazníkom**

  Systém je schopný načítať z databázy dáta strojov aj prídavných zariadení a zobraziť ich užívateľom.

- **P5 Aukcia**

  V aplikácií existuje aukcia strojov. Administrátorom je umožnené vytvárať aukčne ponuky, v ktorých sa draží (adminom vybraný) stroj. Užívateľom je v prípade záujmu umožnené ponúkať sumy do dražby. Po skončení odpočtu prebehne vyhodnocovanie aukčných ponúk, kde sa rozhodne kto je ich víťazom. Víťaz je oboznámený prostredníctvom emailu, že vyhral. Porazení sú informovaní o skutočnosti, že sa im aukciu nepodarilo vyhrať. Admin je upozornený, že aukčná ponuka skončila a kto je jej víťazom.

- **P6 Dopyt**

  Systém umožňuje užívateľom podávať dopyt prostredníctvom formulára, ktorý odošle email administrátorovi s potrebnými informáciami o užívateľovi a jeho záujme o danú položku.

- **P7 Prístup k súčiastkam strojov**

  V aplikácií si vie administrátor po kliknutí na profil v správe strojov rozkliknúť detail nejakého konkrétneho stroja. Okrem iných údajov o spomínanom stroji sa mu zobrazí aj zoznam náhradných dielov, ktoré stroj obsahuje.

- **P8 Registrácia a prihlásenie užívateľov**

  Systém umožňuje bežným užívateľom vytvoriť si účet v systéme, a takisto sa doň prihlásiť (možnosť odhlásenia je samozrejmosťou). A po prihlásení už nemusia do formulárov (pri posielaní dopytu alebo pri zapájani sa do aukcie) zadávať svoje údaje.

5.1 GDPR

Keďže je náš systém webovou aplikáciou zhromažďujúcou užívateľské údaje, tak je potrebné riešiť zásady ochrany osobných údajov. Autor nie je právnikom, a teda nemôže oficiálne prehlásiť, že systém je v súlade s GDPR[1]. Ale systém bol navrhnutý tak, aby zásady splnil. Užívateľa upozorňuje na podmienky a zásady používania systému. Takisto užívateľovi umožnuje vlastné dáta zmazať zo systému, zobraziť a exportovať ich na vyžiadanie. Dáta síce nie sú šifrované softvérovo, ale táto podmienka sa dá splniť pri výbere hostingu. Stačí si vybrať hosting, ktorý ponúka disky podporujúce šifrovanie dát.

5.2 Možné vylepšenia

Softvér síce spĺňa požiadavky a jeho funkčnosť je dostatočná, ale stále existuje priestor pre rôzne vylepšenia a pridanie nových funkcionalít. V tejto podkapitole si o niektorých povieme.

- **Výkonnosť listovania ponuky**

  Ponuku firmy (napr. stroje) prečítame z databázy a vylistujeme jednotlívé položky. Nevyužívame žiadnu virtualizáciu, ani stránkovanie. Keďže systém je určený pre malé firmy a nepredpokladá sa veľká ponuka, tak táto skutočnosť nepredstavuje problem. No v budúcnosti, ak by sa firme používajúcej náš systém darilo a rozhodla by sa rozšíriť ponuku, mohla by sa virtualizácia alebo stránkovanie hodiť.

- **Vylepšenie manažovania správ**

  Systém síce umožňuje administrátorovi prijímať emaily, a takisto na nich odpovedať, mazať ich, a tiež označovať správy ako prečítané/neprečítané. No je v tom trocha pomalý. Akcie síce netrvajú hodiny, a ani minúty, ale do budúcna je to určite niečo, na čom by bolo dobré popracovať.

  Takisto by bolo dobré správam pridať stránkovanie, vyhľadávanie a triedenie podľa dátumu. Hlavným dôvodom existencie správ je umožniť administrátorovi reagovať na dopyt užívateľov, a takisto admina upozorňovať na stav aukčných ponúk. Z podstaty obsahu správ sa čas na ich vybavenie odhaduje na jednotky dní. A teda predpokladáme, že počet správ v schránke nebude príliš vysoký. V prípade, že by ich admin nemazal stále platí, že relevantné správy sa budú nachádzať na vrchu schránky. Ako vidíme, spomínané vlastnosti preto nie sú pre naše účely nevyhnutné. Ale takisto ide o funkcionality, ktoré by mohli oceniť firmy obzvlášť v budúcnosti, kedy by sa zvýšením počtu zákazníkov zvýšila aj frakvencia prichádzajúcich správ.

- **Upozornenie na prehliadku stroja**

  Vo svete to funguje tak, že keď si užívateľ zakúpi stroj, tak po určitom čase by firma mala prísť na prehliadku a stroj skontrolovať. Preto by sme do nášho systému mohli pridať funkcionalitu, ktorá by fungovala následovne. Ak užívateľ odošle dopyt na nejaký stroj a obchod prebehne úspešne,

[1] `https://en.wikipedia.org/wiki/General_Data_Protection_Regulation`

admin by túto skutočnosť zaznačil v systéme, čím by užívateľovi priradil dopytovaný stroj. Potom by sa spustil časovač, ktorý by po uplynutí definovaného času upozornil administrátora na blížiacu sa prehliadku.

A. Prílohy

A.1 Implementácia

Implementácia aplikácie (Visual Studio solution) sa nachádza v priečinku ServIS na adrese: `https://github.com/milantru/ServIS`.

A.2 Videonávod

Odkaz k videonávodu pre administrátorov: .