



**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: Cílem 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 zobrazíť 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 aplikácii vytvoriť účet. Vymenované funkcionality môžu využívať prihlásení aj neprihlásení užívateľia. Bežní prihlásení užívateľia 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.

Title: ServIS – a web system for companies dealing with excavator repairs
Kľúčová slova: Informačný systém C# .NET Blazor Server MySQL

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	2
0.1 Požiadavky na softvér	2
0.2 Alternatívne riešenia	3
0.3 Zhrnutie cieľov	3
1 Analýza zadania	4
1.1 Voľba typu aplikácie, jazyka a frameworku	4
1.2 Návrh systému	5
1.3 Voľba databázy	5
1.3.1 Návrh relačného modelu databázy	6
1.3.2 Voľba typu databázy a databázového servera	8
1.3.3 ORM	8
1.4 Aukcia- odpočet a vyhodnocovanie	9
1.5 Posielanie a prijímanie správ	10
1.5.1 Automaticky generované správy	11
1.6 Ďalší kód tretích strán	11
1.7 GDPR	11
2 Vývojová dokumentácia	12
2.1 ServISData	12
2.1.1 Koreňový priečinok projektu	12
2.1.2 Attributes	12
2.1.3 DataOperations	12
2.1.4 Interfaces	13
2.1.5 Migrations	13
2.1.6 Models	13
2.2 ServISWebApp	13
Záver	14
A Přílohy	15
A.1 První příloha	15

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 keďže 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. Čitateľ by mohol namietat, že v súčasnosti predsa existujú riešenia, ktoré by majiteľom firiem umožnili vytvoriť si vlastný web zdarma. Čitateľ 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, podme 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**

Keďže 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 zákazníkovi poslať dopyt na položky (stroje, prídavné zariadenia), o ktoré majú záujem.

- **P7 Prístup k súčiastkam strojov**

Systém by mal umožniť administrátorovi 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. Keďže znalosť platformy nie je samozrejmosťou 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 spĺňať 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 stať 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 byť použité pre vyplnenie formulárov podľa potreby.

Znova sa vrátim k P5. Keďže 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 posilať správy. Z podmienky P3 usudzujeme, že nikto nebude pri softvéri 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 preklikať medzi svojou emailovou schránkou a naším systémom, bolo by dobre integrovať jeho schránku priamo do systému.

1.1 Voľba typu aplikácie, jazyka a frameworku

Po prečítaní požiadaviek vieme, že chceme vytvoriť webovú aplikáciu s bohatým užívateľským rozhraním, ktorá by bola schopná posilať 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 vyzeráť 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:

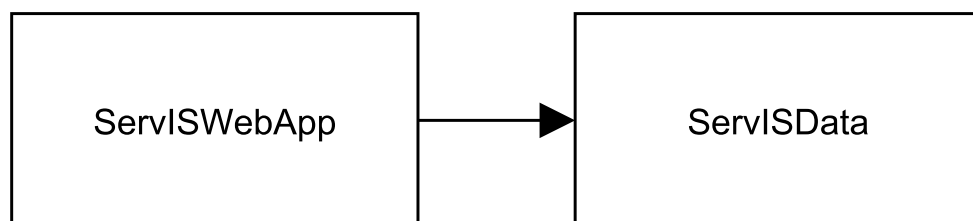
Blazor WebAssembly a Blazor Server. Výber WebAssembly by zahŕňal niekoľko problémov. Pri prvotnej návšteve stránky sa musia klientovi stiahnuť zdrojové kódy aplikácie. To môže chvíľu trvať a mohlo by to odradiť nových potenciálnych zákazníkov. V prípade Blazor Server tento problém nemáme, pretože kód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

Keď už vidíme, že systém je zložený z dvoch častí, podme si rozmyslieť ako budú spolu interagovať. Webová aplikácia potrebuje pre svoje fungovanie dáta. Tejto závislosti sa nezbavíme. Lenže dátová 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. Nieкого 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).

Keďže ide o ponuku, ktorú chceme prezentovať zákazníkovi, chceme okrem textových údajov prezentovať položku, či už stroj alebo prídavné zariadenie, pomocou fotky. A nie jednej (predpokladám, že položku budú chcieť firmy predviesť zákazníkovi 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 prídavného 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 kategórie 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 keďže my chceme systém navrhnuť všeobecne tak, aby si každá firma vedela zdefinovať 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 stroja. 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 zadaných 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. vyvolávacía 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 užívateľ 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 predvypĺňať 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 1.2.

prichádza do úvahy využiť nejaký z ORM frameworkov (z ang. object relational mapping). Asi najznámejšími na platforme .NET sú: Dapper a Entity 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.

1.4 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 riešenia 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 prerenderovala 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. Naš 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 dlhodobá bežiacia operácia, a preto sa hodí ju takisto vykonávať vo vedľajšom vlákne na pozadí.

Podme 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 skutočnosti, 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 ukončenú 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 zákazníkovi (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.

1.5 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 posilať správy. Jednak kvôli splneniu 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 posilať, 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 využiť 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 implementáciu je vhodnejšie použiť protokoly IMAP a SMTP (rovnako to spomína i dokumentácia API³). 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 dlhodobější projekt, sa zdá byť lepšou možnosťou MailKit.

1.5.1 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 funkcionality umožní. Preto sme si v časti Návrh relačného modelu databázy (1.3.1) dopredu pripravili entitu reprezentujúcu automaticky generované správy.

1.6 Ďalší kód tretích strán

TBA? (Fontawesome, Syncfusion) + pridať že management treba na začiatok analyzy (ako do db sa dajú veci? Treba interface pre adminov!)

1.7 GDPR

TBA?

³<https://developers.google.com/gmail/api/guides>

2. 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).

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

2.1 ServISData

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

2.1.1 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 umožňuje iným projektom (v našom prípade ServISWebapp) modely ukladať, čítať, editovať a mazať z databázy.

2.1.2 Attributes

Priečinok obsahuje atribúty¹. 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.

2.1.3 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/>

2.1.4 Interfaces

Priečinkok 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`).

2.1.5 Migrations

Tento priečinkok bol vygenerovaný frameworkom Entity Framework Core a obsahuje vygenerovaný kód. Ide o migrácie². Nad migráciou môžeme rozmýšľať ako nad commitom v gitu. Po aplikovaní migrácie dôjde k zmene v databáze (napr. sa pridá nový stĺpec do nejakej z tabuliek).

2.1.6 Models

Priečinkok obsahuje modely- triedy reprezentujúce entity uložené v databáze.

2.2 ServISWebApp

TBA

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

Závěr

A. Přílohy

A.1 První příloha