**Abstrakt**

Cieľom tejto práce bolo vytvoriť softvérové dielo pre malé firmy zaoberajúcimi 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 jeden 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.

**Úvod**

Krátky text o probléme zviditeľniť malé firmy za nízku cenu.

*Požiadavky na softvér*

Snažil som sa o riešenie ktoré by spĺňalo podmienky. Napr. P1 Dostupnosť (pre adminov, zákazníkov, takže cez internet), P2 neni práca s webom (teda že nikto pri tom nemusí extra sedieť a riadiť to tam, odklikávať niečo atď.), P3 nízka cena, P4 Aukcia strojov....

*Alternatívne* riešenia

Nenašiel som žiadne softvérové riešenie podobné tomuto. Ak si firmy chcú vytvoriť vlastnú stránku, musia o to žiadať firmy (1.), aby im vytvorili softvér na mieru. Prípadne použiť WordPress (2.) (alebo nejakú jeho alternatívu; áno existuje dokonca aj Auction Plugin).   
1. môže byť príliš drahá, 2. vyžaduje znalosť WordPressu (alebo znova len niekoho najmuť za peniaze).

*Zhrnutie*

Moje riešenie možno poskytne menšiu prispôsobiteľnosť, ale je to hotové riešenie (čiže nemusíme ako pri wordpresse vytvárať web „od základu“, nám stačí len dodať dáta), kde sa admin stará iba o vlastné doménové veci- tým myslím svoju ponuku strojov.

**Analýza**

V tejto kapitole sa pozrieme na výber jednotlivých technológií, a takisto si vysvetlíme rozhodnotia pri vymýšľaní architektúry.

*Voľba jazyka a vývojového prostredia*

Na základe naších požiadavok z predchadzájucej kapitoly vieme, že naša aplikácia bude potrebovať na fungovanie internet (takže pôjde o webovú aplikáciu), bude pracovať s databázou, a taktiež by mala obsahovať oku lahodiace grafické rozhranie. Preto volíme vysokoúrovňový programocí jazyk. Na základe autorových znalostí sa nám naskytujú jazyky C# alebo JavaScript. Vzhľadom k tomu, že autor má s jazykom C# viacej skúseností, vyberieme si ho a spolu s ním i platformu .NET, ktorá je s jazykom C# spojená.

Ako vývojové prostredie si volíme Visual Studio 2022, ktoré je vyvíjané spoločnosťou Microsoft rovnako ako aj C#. Toto prostredie poskytuje veľa funkcií, kvôli ktorým je písanie kódu (obvzlášt v C#) pohodlnejšie, napr. kódové snippety, preklikávanie cez referencie volaní metód, predgenerované templaty zdrojákov atď.

Pre verzovanie použijeme git spolu s GitHubom, ktorý sa dobre využíva i z už spomínaného Visual Studia.

*Typ aplikácie a výber frontendového frameworku*

Ako už bolo spomenuté, potrebujeme internet (aby sa užívatelia vedeli dostať jednoducho k obsahu), preto volíme webovú aplikáciu. Platforma .NET nám ponúka framework ASP.NET alebo Blazor. Obe frameworky toho majú veľa spoločného, ale jedným z rozdielov je, že Blazor umožňuje vytváranie komponent, ktoré môžeme „recyklovať“ (používať na viacerých miestach na webe). Autor na začiatku práce neovládal ani jeden zo spomínaných frameworkov. V tom čase ho však zaujal Blazor a plánoval sa ho naučiť. Hodil by sa tento framework pre túto prácu?

V danom čase existovali 2 Blazor hostovacie modely- Blazor Server a Blazor WebAssembly. Pre systém potrebujeme miesto (centrálny server), kde by sa ukladali dáta, aby k ním užívatelia pristupovať odkiaľkoľvek. Preto vylučujeme Blazor WebAssembly, ktorého cieľom je preniesť aplikáciu ku klientovi. Na druhej strane Blazor Server nám dokonalo vyhovuje. Aplikácia beží na serveri. Aktualizácie užívateľského rozhrania, event handling a JavaScriptové volania prebiehajú medzi serverom a klientom po SignalR spojení. Takisto kedže server prerenderúva HTML a to posiela klientovi, je aplikácia „search engine friendly“ (narozdiel od WebAssembly). Takisto pri WebAssembly, kedže je to nová technológia, nemáme záruku, že bude fungovať na všetkých browseroch, kdežto pri Server by sme takýto problém mať nemali. Aby toho nebolo málo, tak prihodím ešte jeden dôvod a tým je rýchlosť prvotného načítania. Nevýhodou WebAssembly je dlhý čas ptvotného načítania stránky, pretože sa musia ku klientovi posťahovať zdrojáky. Nemyslím si, že dlhé načítavanie by bolo pre zákazníkov príjemným zážitkom.  
Viac tu: <https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0>

*UI knižnice tretích strán*

Aj napriek tomu, že v Blazeri je zabudovaná nejaká verzia bootsrapu, a teda môžme si relatívne ľahko vytvoriť aj vlastné pekné užívateľské rozhranie, tak pre urýchlenie a zjednodušenie implementácie celého systému som za rozhodol využiť kódy tretích strán. Menovite ide o:

1. NuGet balíčky s Blazor kompontmi od Syncfusionu
2. Ikonky od FontAwesome

V prípade 1. poskytujú Community verziu, ak sú splnené určité podmienky (počet developer, nepresiahnutie 1M$). V našom prípade sú podmienky splnené a predpokladám, že ešte aj nejakú dobu splnené ostanú.  
Viac na: <https://www.syncfusion.com/sales/communitylicense>.

V prípade 2. poskytujú určitú podmnožinu ikoniek zdarma.  
Viac na: <https://fontawesome.com/plans>

Takže v oboch prípadoch spĺňame podmienku na cenu (spomenutá v úvode).

*Posielanie emailov*

Rozobrať možnosti ako posielať emaily. Manuálne implementovať protokoly? API (napr. Gmail API)? Knižnice? Ak knižnicu, tak akú? AE.Net.Mail libaray alebo Mailkit?

*Databáza*

Databázu si vyberieme relačnú, konkrétne MySql.

Reláčnú preto, lebo záznamy, ktoré budeme ukladať majú presne definovanú štruktúru. Reláčná databáza nám k ním poskytne rýchly prístup, a takisto zaručí neporušenosť dát. Navyše autor má s relačnými trochu viac skúseností, než s NoSql databázami.

Ako databázový server si vyberieme MySql preto, lebo vo verzii Community je zdarma. To vyhovuje jednému z požiadaviek spomenutých v úvode. Existujú aj platené verzie, ktoré poskytujú viacero funkcionalít, ale to je pre naše potreby irelevantné nakoľko my potrebujeme databázu iba na perzistenciu dát.

*UML databázy*

Obrázky, opis.

*Zmena databázového providerav kóde*

Z Oracle na Pomelo, pretože som nemohol robiť migrácie... ak som mal staršiu verziu packagov, mohol som robiť migrácie, ale nejaké z queries nefungovali dobre (problém s .Include()). Ak som updatol na najnovšiu verziu, queries fungovali dobre, ale nešli migrácie. Viackrát som sa na internete stretol s tým, že Pomelo je lepší provider(rýchlejšie reagujú + na nuget.org viac stiahnutí) a vraj ak ho zmením všetko bude fungovať. A áno, po zmene všetko funguje a mám najnovšie (toho času) verzie. (Ďalšie info tu: <https://stackoverflow.com/questions/70224907/unable-to-resolve-service-for-type-microsoft-entityframeworkcore-diagnostics-idi>)

*Výber frameworku pre prácu s databázou (ORM)*

Rozoberiem možnosti, napr. manuálne všetko (SQL scripty), Dapper, Entity Framework Core. A prečo som vybral EF Core.

*Návrh systému*

Opísať to, že existujú 2 projekty- data a web app. Vysvetliť, že to je preto, lebo takto máme oddelené dáta (modely) od bussines logiky. V prípade potreby sa môžu dáta oddeliť od web app a spojiť s iným softvérom (napr. mobilná aplikácia).   
(**Pozn.:** Myslím, že to oddelenie momentálne nie je dokonalé. Bolo by dobre pozrieť a napraviť to, prípadne v závere spomenúť ako vylepšenie.)

*UML programu*

Obrázky, opis.

*Adresárová štruktúra.*

Opísanie adresárovej sťruktúry, v akom priečinku je čo.   
(**Pozn.:** Tiež sú sem-tam veci ktoré lepšie sa dali spraviť, napr. som mohol spraviť Services, a v tom BackgroundServices. Ale dopredu som nevedel, že takto to bude, lebo nevedel som o Background Servicoch. Alebo Attributy/Extensiony v data projekte. Takže buď opraviť alebo asi dať do záveru ako možnosť vylepšenia.)

*Listovanie položiek (strojov, prídavných zariadení) pomocou Virtualize*

Virtualize je vstavaná komponenta v Blazeri určená pre virtualizáciu, pomáha to výkonnosti programu ak chceme vylistovať viacero položiek. No pri stylovaní som zistil, že pri využití určitých css štýlov táto komponenta nefunguje správne. Chcel som využiť display: flex ale nemzem, viac na: <https://learn.microsoft.com/en-us/aspnet/core/blazor/components/virtualization?view=aspnetcore-6.0#advanced-styles-and-scroll-detection-1>. Nechať Virtualize? Ak ostane, tak musím vymýšľať ako napísať štýly aby to vyzeralo dobre a zároveň to nerobila problém komponente Virtualize (viacero css pravidiel robí problém, nie len flex). Alebo sa Virtualize jednoducho zbavím, veď kedže to robím pre malé firmy, tak je asi rozumné predpokl´dať že tých strojov nebudú milony. Riešenie je (asi) vyhodiť. (**Pozn.:** Myslím, že toto som ešte nespravil, nevymazal som Virtualize.)   
(**Pozn.:** Ono by sa síce ešte dalo, relatívne jednoducho, vytvoriť paging kompontu. Problém je v tom, že by to vzalo čas a paging tam nechcem, nehodí sa mi tam ak bude len zopár strojov. Ak by tých strojov bolo naozaj veľa, tak dobre, hodí sa mi tam. Čo ma privádza opäť k tomu, že ak ich je málo tak netreba ani paging ani Virtualizáciu. Pri veľkom počte sa môže spraviť vlastný paging.   
A to by som mohol napísať tiež do záveru do možných rozšírení.)

*Osobitná validácia vo Forme*

Popísať trik s validáciou formu (v AuctionBidForm). Nevyužijú sa len validačné atribúty modelu, ale „dodefinujeme“ si pri forme validáciu.

*InputSelector*

Keď som použil klasický input select, resp. InputSelect vstavaný v blazoru, tak som zistil, že okrem intu a enumu nemôžem využiť iný typ. No ja som chcel v selecte využiť vlastné typy (napr v ExcavatorForme, pri výbere ExcavatorTypu), preto som nejako „ohackoval“ InputSelect a vytvoril si vlastný InputSelector, ktorý vie pracovať aj s mojimi typmi.

*Background Services*

Krátke predstavenie Background Servicov, čo/na čo sú, pomocou čoho sa implementujú v Blazeri.

*TimerService*

Pôvodne mala táto služba v sebe 2 timery. 1. odpálil event každú sekundu (aby som vedel odpočítavať/renderovať odpočet pri aukcii), 2. odpálil event za nejakú dlhšiu dobu (napr. 1 minútu), aby sa skontrolovali aukčné ponuky. Ak nejaká skončila tak sa rovno aj vyhodnotila.

Takto sa mi to ale nepáčilo, miešal som viaceré funkcionality dokopy... Preto som sa rozhodol pre lepšiu prehľadnosť, rozšíriteľnosť a udržateľnosť tieto funkcionality rozdeliť. TimerService sa stala abstraktnou triedou, od ktorej dedí EverySecondTimerService a AuctionEvaluatorService.

Pôvododne mal TimerService v sebe statický field updateEvent, kde si užívateľ tejto služby vedel zaregistrovať nejaký handler. To sa ukázal ako problém, pretože po rozdelení využíval tento field ako EverySecondTimerService, tak aj AuctionEvaluatorTimerService (viedlo to k tomu, že každú sekundu sa odpálil event na timeru EverySecondTimerServicu a zavolal sa eventHandler, ktorý kedže bol statický tak zavolal nie len countdown updaty ale aj vyhodnotenie aukcie). Na prvý pohľad triviálna chyba, no s nie úplne triviálnym riešením.

TimerService dedí od BackgroundService, vieme ho zaregistrovať v Program.cs pomocou AddHostedService (bude ako singleton, niekedy bol ako transient vraj). Ak ho registrujeme takto, spustí sa metóda ExecuteAsync (dedená z BackgroundService) automaticky- čo chceme. No nemôžeme (z nejakého dôvodu) túto službu v komponentách injectnúť, a teda registrovať na inštancii tejto služby (EverySeceondTimerService) event handlery.

Čiže v rodičovi nemôžeme mať statický field, no ani inštančný (event field, na ktorý by sme registrovali handlery). Jediné čo môžme je presunúť ho do potomkov, ale to by znamenalo že potomkom musím odkryť timer a to nechcem (nechcem aby potomkovia museli riešiť "low-level" veci ako je registrácia vlasntných eventov do Elapsed eventu na timeru... Chcem, aby len povedali za aký interval sa má stať čo). Čo teda spravíme?

Vytvoríme v EverySeceondTimerService statické metódy -stačila by properata/field (ale statická!), no s metódou to bude vo Visual Studiu prehliadnejšie kvôli možnosti cestovať po referencíach metód-- pre (od)registrovanie užívateľských event handlerov. Tieto event handlery sa registrujú do statického event fieldu updateEvent, ktorý musí byť statický, pretože k inštančnému by sme sa nedostali (inject nejde).

A v AuctionEvaluatorService vytvoríme inštančný field (event), kde si registrujeme svoje metódy (teda nie užívateľ triedy definuje handlery ako tomu je v EverySeceondTimerService).

V rodičovi, TimerService, vytvoríme abstraktnú metódu GetEventHandlers, ktorú si potomkovia implementujú (EverySeceondTimerService vráti svoj statický field, AuctionEvaluatorService vráti svoj inštančný). Táto metóda sa volá v lambda funkcii, ktorá sa registruje na Elapsed event timeru. Čiže po prejdení intervalu sa odpáli Elapssed, ten aktivuje registrovanú lambda funkciu (registrovaná v rodičovi), tá v sebe zavolá abstraktnú metódu GetEventHandlers a na jej výsledku sa zavolá Invoke.

*Komponenty s odpočtom*

O tom, že vyriešené pomocou 1 timeru (pomocou TimerService), a tiež že musia byť ako osobitná komponenta, aby sa nerenderovalo každú sekundu viac ako je treba. Možno spomenúť problém, že keď to nebolo v osobitnej komponente, tak form sa stále prerenderovával a nedalo sa doň písať. (Čiže ak len statická komponenta tak to možno nevadí, myslím že Blazor len diffy posiela po SignalR, ale jakmile tam je niečo komplexejšie, tak to vadí a musíme dať odpočet do osobitnej komponty.)

*Automapper*

Mohol by som tiež opísať problém s EF Core a ako som ho vyriešil s pomocou/bez pomoci Automapperu. Porovnať aké by to bolo s Automapperom, prečo ho nechcem využiť, ako som to vyriešil bez neho.  
(**Pozn.:** Momentálne to ale vyriešené ešte nemám, takže viac info tu asi zatiaľ neposkytnem.)

*Tagy v šablónach automaticky generovaných správ*

Úvaha o tom, ako to mohlo byť implementované. Default texty dať do databázy? Napísať ich do configu? Nechať ich v triedach?

*Automatické vpisovanie tagov do šablón na klik*

Keď admin píše vlastné automaticky generované správy (resp. ich šablóny), tak chcem aby stačilo kliknúť na tag a doplní sa mu do input textu/text arey daný tag. Problém je v tom, že (zdá sa, že) s Blazorom sa nedá získať pozíciu kurzoru. Takže musíme použivať JavaScript, ale tam sa to trochu komplikuje s vyberaním elementu.

(**Pozn.:** Nie je zatiaľ implementované. Ak sa nestihne, môžem pridať do záveru ako možné rozšírenie.)

*Opustené featury (nie radšej toto do záveru?)*

Tu by som písal o nejakých featurách, ktoré som vypustil a prečo. Sú to história nákupov a zoznam vyhraných aukcií. Vypustil som to preto, lebo neexistoval spôsob ako zistiť, či užívateľ kúpil stroj/prídavné zariadenie alebo či vyhral aukciu. Teraz sa to už dá, pretože v aplikácii existujú správy (emailová schránka). No z časových dôvodov a veľkosti diela sa už k spomínaným featuram nevrátilo.

**Vývojová dokumentácia**

V tejto kapitole rozoberiem kde a aké(, a ako?) featury systému sú implementované.  
V tejto kapitole sa pozrieme na implementáciu systému. Ale rád by som podotkol, že tento text neposkytne vysvetlenie každého riadku kódu. K tomu slúži zdrojový kód samotný (prípadne komentár obsiahnuté v ňom na miestach, kde to autor uznal za vhodné).

*Dáta*

O dáta (ich mapovanie, uloženie do databázy) sa stará projekt ServISData. V tejto podkapitole si povieme niečo o tomto projekte a jeho fungovaní.

*Databázové Entity*

Vysvetlím, že triedy v priečinku Models sú entity v databázi, neobsahujú logiku, sú to čisto dátonosné triedy. Sú oatributované. ...

(**Pozn.:** Nechcem ešte presne rozvádzať čo tu budem písať, lebo zrejme sa to trochu zmení po tom ako vyrieším problém s EF Core/Automapperom.)

*Atribúty entít*

Momentálne mám 2 atribúty:

* AutogeneratedMessageDataAttribute
* InputTypeLabelAttribute

Opísať na čo sú, ako využívať...

*Extensions entít*

Kedže nechcem mať logiku v modelových triedach, tak som využil Extension metódy. Tak tu by som opísal to. Prečo/na čo sú tie extensions.

*ServISDbContext*

V tejto triede sa vytvárajú a konfigurujú tabuľky.   
(**Pozn.:** Možno by bolo dobre spomenúť, že sa využíva odporučaný postup s CreateDbContext metódou, ako je opísané tu: <https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration/#using-a-dbcontext-factory-eg-for-blazor>. Alebo toto skôr do analýzy? Ale tak čo by som tu potom písal?)

*ServISApi*

Pomocou tohto API sa viem dostať k entitám z WebApp projektu, používam ho na na CRUD operácie nad modelovými triedami.

*DataOperations*

Get metódy v API som navrhoval, tak aby boli prispôsobené na dátovéhé operácie také, aké si zadá užívateľ (napr. paging, filter...). Tieto operácie sa vedia definovať pomocou tried, ktoré sa nachádzajú v DataOperations.

*Webová aplikácia*

V tejto kapitole si povieme ako funguje biznis logika, ako prezentujeme dáta užívateľom atď.

*Listovanie položiek (strojov, prídavných zariadení...)*

Krátky text o tom, že existujú komponenty „\*Lister“, ktoré robia to, že si natiahnu z databázy entity a vylistujú ich. Tieto komponenty vylistujú komponenty „\*Card\*“ (stroje majú svoj druh kartičiek, prídavné zariadenia majú svoj iný druh kartičiek...). Na kartičky sa dá kliknúť a užívateľ je prenesený na stránku s detailom položky. Ak je prihlásený admin, tak jemu sa zobrazujú aj viaceré možnosti. Nad vylistovanými kartičkami je odkaz, ktorý zobrazí adminovi formulár pre vytvorenie novej položky. Na kartičkách je zobrazené tlačístko X, ktoré polozku zmaže. Prípadne tlačítko na editáciu položky.

*Detaily položiek*

Každá položka ako stroj, prídavné zariadenia alebo aukčná ponuka majú svoj detail, kde si užívateľ vie pozrieť dodatočné info ale takisto vie danú vec dopytovať. A to prostredníctom formuláru. V prípade prvých dvoch položiek sa využíva DemandForm. Ten pošle pomocou triedy EmailManager email adminovi s vyplneným textom od užívateľa ale takisto vyplní i hlavičky mailu s dodatočnými informáciami, konkrétne s url položky. V prípade aukčnej ponuky sa využíva AuctionBidForm. Ten uloží AuctionBid do databázy.

*Posielanie emailov*

Tu by som popísal triedu EmailManager.  
 (**Pozn.:** Neviem, či je táto sekcia potrebná, myslím, že kód je dostatočne samovysvetľujúci sa. Ale kedže ide o dosť dôležitú vec, tak možno nejako stručne aspoň by sa hodilo.)

*Autentikácia a autorizácia*

Tu opíšem ktoré komponenty sa podieľajú na autentikácií a autorizácií. Jednak komponenty, ktoré v sebe držia logiku prihlásovania, kontrolu prihlásenia (a následného renderu stránky podľa toho či je užívateľ prihlásený alebo nie; tiež by som mohol spomenúť to, že SessionStorage využívam), ale takisto aj komponenty, ktoré majú iba vizuálnu funkciu a privádzajú užívateľia k prihlasovaciemu formuláru (LoginPanel).  
(**Pozn.:** Mal by som tu alebo v analýze spomenúť problém s revalidáciou prihlásenia? Pretože asi by sa to hodilo viac do analýzy ale tak dáva väčší zmysel o tom hovoriť tu, pretože v analýze čítateľ ešte nevie ako funguje auth, a nebudem mu to vysvetľovať 2x, nie?)

*Profil*

V tejto podkapitole by som opísal ako vyzerá profil bežného užívateľa a ako to vyzerá u admina. Keď sa užívateľ (už či ten bežný alebo admin) prihlási a klikne na profil, tak sa mu zobrazí komponta Profile. Tá obsahuje panel (komponent TabControl) s určitým počtom tabov (TabPage). U bežného užívateľa vidíme iba tab Moje údaje, kde má formulár (komponenta UserForm) so svojími dátami ako sú meno, priezvisko atď.. Tieto dáta vie upravovať. V prípade admina vidíme okrem tabu Moje údaje oveľa viac. Poďme sa na to pozrieť v následujúcej kapitole.

*Profil admina*

Tu by som opísal jednotlivé taby, že skoro každý obsahuje komponentu ItemsManagement (ktorá využíva Syncfusion grid, môžem dať link na grid). Že je napísaná genericky tak, aby umožnila fungovanie v každom tabe pre rôzne typy. Každý ItemsManagement vyzerá ako stŕankovaná tabuľka s možnosťou pridania, editácie a vymazania záznamu. Pridanie a editácia je uskutočnená tak, že sa v modálnom okne zobrazí formulár pre danu entitu.

Okrem spomínaného tabu Moje údaje a tabov s Managementami existuje aj tab Správy. Po kliknutí naň sa nám zobrazí Komponenta Messages.

(**Pozn.:** Pre Messages by som asi vyhradil novú podkapitolu. Tu sa toho deje viac. Napr. url v hlavičke mailu...)

*Vytváranie položiek*

Vytváranie položiek bolo načrtnuté už v minulej podkapitole. Každa položka (či už stroj alebo prídavné zariadenie...) má svoj formulár „{názov položky, napr. Excavator}Form“. Väčšina entít sa dá vytvoriť/upraviť v spomínaných managementochna profile admina. Ale MainOffer, Excavator, AdditionalEquipment, AuctionOffer majú vyhradenú špeciálnu stránku pre pohodlie admina. Keď si dá prihlásený admin vylistovať napr. stroje, tak pred ich kartičkami je odkaz na pridanie nového stroja (podobne pri hlavných ponukách, prídavných zariadeniach a aukčných ponukách). V kóde sa tieto stránky volajú „Create{názov položky, napr MainOffer“.

*Aukcia*

Jeden z modelov (AuctionOffer) predstavuje aukčnú ponuku, ktorú vie admin vytvoriť. Užívatelia vedia pristúpiť na detail tejto ponuky (komponenta AuctionOfferDetail), a to prostredníctom kliknutia na jej kartičku (komponenta AuctionOfferCard) vylistovanú pomocou komponenty AuctionOffersLister. V detaile existuje komponenta CountdownDisplayer, ktorá odpočítava čas do konca danej aukcie (aukčnej ponuky). Ako už bolo skôr spomenuté, užívateľ vie pomocou formuláru AuctionBidForm pridať svoju ponuku. Tým sa vytvorí AuctionBid v databázi. Keď sa aukcia skončí, už sa nedajú pridávať (o to sa postará AuctionOfferDetail). Po nejakom čase sa začne vyhodnotenie aukcie (kontrola, či existuje skončená aukcia, a následné vyhodnotenie sa deje každých X minút). Kontrola skončených aukcií a ich vyhodnotenie sa deje v triede AuctionEvaluatorService (konkrétne v metóde RegisterEventHandler). Pri vyhodnotení sa vezme z databázy AuctionBid s najvyššou ponúknutou sumou (property Bid). Výhercovi sa oznámi, že vyhral a účastníkom, ktorí prehrali sa ich prehra oznámi tiež. Adminovi sa oznámi, že aukcia má víťaza a kto to je. Navyše sa aukcia označí ako vyhodnotená (zabráni opätovnému vyhodnocovaniu). V prípade, že Aukcia nemá účastníkov (teda skončila bez víťaza), sa aukcia reštartuje s posunutým koncom aukcie (splnená jedna z podmienok z úvodu) a upozorní sa admin, že aukcia skončila bez víťaza. Vráťme sa k prípadu, že víťaz existuje... každá strana bola upozornená a aukčná ponuka je označená ako vyhodnotená. Keď má admin upozornenie otvorené v správach, vie cez vloženú stránku prejsť na detail spomínanej aukcie. Má 2 možnosti. Ak sa s užívateľom dohodol a všetko prebehlo v poriadku, vie aukčnú ponuku vymazať (vymaže sa AuctionOffer spolu so svojimi AuctionBidmi). Druhá možnosť nastáva v prípade, ak výherca aukcie nespolupracuje, nastal nejaký problem a o vyhraný stroj nemá záujem. V tom prípade admin vie rovnako cez vloženú stránku v správach prejsť na detail spomínanej aukcie, kliknúť na upraviť, zmení parametre aukcie (nový koniec, cena...), namiesto tlačítka „Uložiť“ sa adminovi zobrazuje tlačítko „Uložiť a spustiť odznova“, po kliknutí sa aukcia reštartuje (vymažú sa všetky doterajšie AuctionBidy danej aukčnej ponuky) a aukcia funguje akoby ju práve nanovo vytvorili.

*Automaticky generované správy*

V predošlej kapitole som spomínal odosielanie správ výhercom, porazeným a adminom. Šlo o automaticky generované správy. V tejto kapitole by som pohovoril o tom ako sú implementované. Defaulty, tagy, adminom definované správy...

**Užívateľská dokumentácia (asi skor do priloh video tutorial)**

Toto by som asi vynechal a dodal len do prílohy video (tutoriál pre adminov, že ako využívať aplikáciu).

**Uživateľský feedback**

Nejaké úlohy by som si pripravil (test), dal by som nejakým užívateľom spraviť a zapísal by som ako sa im darilo- teda ako veľmi užívateľsky prívetivá aplikácia to je.

**Záver**

V tejto kapitole sa pozrieme na možné rozšírenia, a takisto čo mohlo byť spravené ináč/lepšie...

*Môžné rozšírenia*

Tu by som písal o možných rozšíreniach... Napr. o tých ktoré boli v skôr v texte spomuté ako opustené (história nákupov, zoznam vyhraných aukcií).

*Čo som mohol spraviť ináč...*  
Mohol by som opísať to, že API je obrovský zdroják, radšej som to asi mohol rozdeliť na časti (služby), aby sa v tom ľahšie orientovalo. Síce som sa snažil v tom mať systém, že časti, ktoré spolu súvisia sú pri sebe, ale bolo by priehľadnejšie ak by boli menšie zdrojáky.

**Citácie**

...